



City Research Online

City St George's, University of London

Citation: Youssef, M. W. A. F. (1993). Transaction behaviour in large database environments: A methodological approach. (Unpublished Doctoral thesis, City, University of London)

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: <https://openaccess.city.ac.uk/id/eprint/29631/>

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to [City Research Online policy](#).

575

**Transaction Behaviour in Large Database Environments
A Methodological Approach**

Mohamed Wagdy Abdel Fattah Youssef

*Submitted for part of examination for
Doctor of Philosophy*

**Department of Business Computing,
School of Informatics,
CITY University,
London.**

August 1993

1

*Dedicated to my parents
if not for their continuous encouragement and support
this work would have never been completed.*

Table of Contents

	Page
CHAPTER 1 INTRODUCTION	
1.1 Some Performance Concepts.....	1
1.1.1 Some Indices of Performance	4
1.1.2 Analytical Models.....	5
1.1.2.1 Statistical Models	6
1.1.2.2 Graphical Models.....	6
1.1.2.3 Algorithmic Models	6
1.1.3 Simulative Models	7
1.1.4 Benchmarks.....	7
1.1.4.1 The Role of Benchmarks	8
1.1.4.2 Selecting a Benchmark Mix	8
1.1.4.3 Weaknesses of Benchmarks.....	9
1.1.4.4 The main Characteristics of a Good Benchmarks.....	9
1.2 Problem Definition.....	10
1.3 Research Objectives and Approach.....	11
1.3.1 Empirical Approach to Problem Solving	12
1.3.2 Specification of the Research Domain	13
1.3.3 Development Basis of the CITY Benchmark	13
1.3.4 Test Capabilities of the CITY Benchmark.....	14
1.3.5 Test and Verification of the CITY Benchmark Results	14
 CHAPTER 2 AN ANALYSIS OF DATABASE PERFORMANCE MEASURES	
2.1 Introduction	18
2.2 Historical Background of Database Benchmarks.....	19
2.3 Database Benchmarks.....	20
2.3.1 Benchmark Methodology for IMS DBMS	22

	Page
2.3.1.1 System Configuration and Running Environment	23
2.3.1.2 Database Definition.....	23
2.3.1.3 Measures of Space Utilisation.....	23
2.3.1.4 Measures for the IMS operators	24
2.3.2 A Benchmark Methodology for CODASYL DBMS	25
2.3.3 Cellular Systems Benchmarks.....	25
2.3.4 An Interactive Benchmark for INGRES DBMS.....	26
2.3.5 A Benchmark By M. Stonebraker.....	26
2.3.6 Benchmark Methodology By Benigni and Yao.....	27
2.3.6.1 System Configuration.....	27
2.3.6.2 Test Data.....	28
2.3.6.3 Benchmark Workload.....	28
2.3.7 Benchmark Methodology By P. Strawser	30
2.3.8 Performance Evaluation of Temporal DBMS.....	30
2.3.9 Benchmark Methodology Using Simple Queries.....	31
2.3.10 The SCAN Benchmark	33
2.3.11 The Onekay (IBM 1987).....	33
2.3.12 The RAMP-C	34
2.3.13 Performance Evaluation of Main Memory DBMS.....	34
2.3.14 BYTE Benchmark	35
2.3.15 The Set Query Benchmark.....	35
2.3.16 The HyperModel Benchmark	36
2.3.17 The Engineering Database Benchmark.....	37
2.4 Activities of Naval Postgraduate School.....	38
2.4.1 The MBDS Hardware Configuration	39
2.4.2 The Attribute Based Model.....	40
2.4.3 The Benchmark Strategy	40
2.4.4 System Configurations Considerations.....	41
2.4.5 Database Size Considerations.....	41
2.5 The Most Widely Used Benchmarks.....	41
2.5.1 The Wisconsin Benchmarks.....	42
2.5.1.1 Description of the Test Database.....	42
2.5.1.2 Description of the Benchmark Queries.....	42
2.5.1.3 Performance Metric.....	43

	Page
2.5.2 Comments on The Wisconsin Benchmark	43
2.5.3 The Debit-Credit (TP1).....	46
2.5.3.1 Description of the Test Database.....	47
2.5.3.2 Description of the Benchmark.....	47
2.5.4 The Transaction Processing Performance Council (TPC)	
Activities.....	48
2.5.5 The TPC-A.....	49
2.5.6 TPC-B.....	50
2.5.7 The TPC-C benchmark.....	51
2.5.7.1 The TPC-C Logical Database Design	51
2.5.7.2 The TPC-C Transactions Profile.....	52
2.5.7.3 Limitations of The TPC-C Transactions	55
2.6 Limitations of The Existing Benchmarks.....	55
2.7 Conclusion.....	57

CHAPTER 3 PROBLEM DEFINITION AND DATA GATHERING AND ANALYSIS TECHNIQUES

3.1 Introduction	60
3.2 Problem Definition.....	62
3.2.1 Inconsistency of TPC Benchmarks Implementation	62
3.2.2 Technical Limitations of The TPC Transactions.....	65
3.2.3 Inconsistency of the TPC Transaction Comparisons	67
3.3 Empirical Approach To Solve The Defined Problem	68
3.3.1 Specification of the Domain of Studies.....	70
3.3.1.1 Definition of High-volume Transactions	
Environments (OLTP)	70
3.3.2 Criteria for Organisations Selection.....	71
3.3.2.1 The Local Authorities Computer Centre	71
3.3.2.2 The Airlines Computer Centre	72
3.3.2.3 The Bank Computer Centre	73

3.4 Empirical Studies to Test Database Performance	
Factors	75
3.4.1 Table Size	77
3.4.2 Row Size Effect	78
3.4.3 Database Indexed Attributes	79
3.4.4 Attributes Types and Distribution	80
3.4.5 Transaction Database Operations	80
3.4.6 Transactions time Utilisation and I/O Operations	81
3.4.7 JOIN Operation	81
3.4.8 Transaction Complexity	82
3.4.9 Transaction Output	83
3.4.10 Background workload	85
3.4.11 Discussion of Performance Factors	86
3.5 Studying Organisations and Data Gathering Techniques	87
3.5.1 Approaches of Studying Organisations	88
3.5.1.1 Pure Basic Research	88
3.5.1.2 Basic Objective Research	88
3.5.1.3 Evaluation Research	88
3.5.1.4 Applied Research	89
3.5.1.5 Action Research	89
3.5.2 Data Gathering Techniques	90
3.5.2.1 Formal Meetings	91
3.5.2.2 Interviews	91
3.5.2.3 Studying Systems' Documents	92
3.5.2.4 Studying Systems' Outputs	92
3.5.2.5 Systems Monitoring	93
3.6 Statistical Techniques	93
3.6.1 Some Definitions	93
3.6.1.1 Arithmetic Mean	93
3.6.1.2 Sample Variance	94
3.6.1.3 Standard Deviation	94
3.6.1.4 Number of Measurements for Reliable Sample	95
3.6.1.5 The Normal Distribution	96
3.6.2 Design of Experiments	96

	Page
3.6.2.1 Replication of Experiments.....	97
3.6.2.2 Randomisation of Experiments' Events.....	97
3.6.2.3 Analysis of Covariance	97
3.6.2.3.1 One-Way Analysis of Variance.....	97
3.6.2.3.2 Two-Way Analysis of Variance.....	99
3.6.3 Comparing The Workloads Of Different Transactions	101
3.6.3.1 Calculating Average Rate of Change (Difference Quotient).....	102
3.6.3.2 Calculating Relative Change.....	103
3.6.4 Ordinary Least Squares Method (OLS).....	104
3.7 Summary.....	105

CHAPTER 4 DATABASE CHARACTERISTICS AND TRANSACTION BEHAVIOUR IN LARGE DATABASE ENVIRONMENTS

4.1 Introduction	107
4.2 A Study In a Large Local Authority Computer Centre.....	110
4.2.1 Introduction.....	110
4.2.2 The Business of the Local Authority.....	110
4.2.3 Organisational Structure of the Local Authority	112
4.2.4 The Local Authority Processing Environment.....	113
4.2.5 The Local Authority Main Applications	114
4.2.6 The Main Characteristics of The Application Databases.....	115
4.2.7 Transactions Behaviour of the Running Systems	116
4.2.7.1 On-line Transactions Database Operations.....	116
4.2.7.2 Ratios of Key Utilisation	117
4.2.7.3 Average Number of Retrieved Records per Transaction.....	117
4.2.8 Background Workload	118
4.3 A Study In a Large British Airlines' Computer Centre	119
4.3.1 Introduction.....	119
4.3.2 Data Gathering.....	119
4.3.2.1 Monitoring the Running Systems.....	119

	Page
4.3.2.2 Studying Systems Outputs and Systems Documents	120
4.3.2.3 Interviews	121
4.3.3 Processing Environment	122
4.3.4 The Main Characteristics of the Application Databases.....	123
4.3.4.1 Application Databases on the S2 Processor	124
4.3.4.2 Application Databases on the S3 Processor	124
4.3.4.3 The Acceptance Test System (ATS).....	125
4.3.5 Transactions Behaviour of the Running Systems	125
4.3.5.1 On-line Transaction Database Operations.....	126
4.3.5.2 Time Utilisation and I/O Operations.....	126
4.3.5.3 Number of Databases Accessed by One Transaction (JOIN Operation)	127
4.3.5.4 Transactions Complexity (Nested Selects)	127
4.3.6 Number of Retrieved Records per Transaction.....	129
4.3.7 The Background Workload	129
4.4 A Study In a Large UK Bank Computer Centre	129
4.4.1 Introduction.....	129
4.4.2 Data Gathering Techniques	130
4.4.2.1 Interviews	130
4.4.2.2 Monitoring the Running Systems.....	131
4.4.2.3 Studying Systems Documentation.....	132
4.4.3 Processing Environment	132
4.4.3.1 Hardware and Software.....	132
4.4.3.2 The Bank Running Applications.....	136
4.4.3.3 On-line Transaction Flow.....	137
4.4.3.4 The Bank Database Management Systems.....	137
4.4.4 Characteristics of Application Databases	138
4.4.5 Transaction Behaviour of the Running Systems	139
4.4.5.1 On-line Transactions Database Operations.....	139
4.4.5.2 ATM Transactions Database Operations	140
4.4.5.3 CPU Time Utilisation and I/O Operations	142
4.4.5.4 Number of Databases Accessed by One Transaction (JOIN Operation)	143
4.4.5.5 Transactions Complexity (Nested Selects)	144

	Page
4.4.5.6 Average Number of Retrieved Records per Transaction.....	144
4.4.6 The Background Workload	145
4.5 Discussion.....	145
4.5.1 On-line Transaction Behaviour Represented as Number of Database Operations.....	145
4.5.2 Comparison Between TPC-A, TPC-B and On-line Transaction At the Three Organisations.....	147
4.5.3 Comparison Between the Bank ATM Transaction and The TPC-A Benchmark	148
4.5.4 Comparison Between TPC-C and On-line Transaction At the Three Organisations.....	149
4.5.5 Common Characteristics Between Databases.....	153
4.5.5.1 Database Size.....	153
4.5.5.2 Row Size	153
4.5.5.3 Indexed Attribute and Attribute Types	154
4.5.6 On-line Transactions Behaviour.....	154
4.5.6.1 JOIN Operation	154
4.5.6.2 Transaction Complexity.....	156
4.5.6.3 Number of Retrieved Records per Transaction.....	156
4.5.6.4 On-line Transaction Ratios of CPU to I/O Utilisation	157
4.5.6.5 Background Workload.....	157
4.6 Conclusion.....	158

CHAPTER 5 The CITY BENCHMARK

5.1 Introduction	160
5.2 The CITY Benchmark Domain Specification.....	162
5.3 The CITY Benchmark Objectives	163
5.4 The CITY Benchmark Metrics.....	163
5.4.1 Response Time and Transactions per Second	164

	Page
5.4.2 How Time is Measured.....	166
5.5 The CITY Benchmark Database.....	167
5.5.1 Number of Tables.....	168
5.5.2 Row Size.....	169
5.5.3 Attributes' Types and Accessibility of Rows (Indices).....	169
5.5.4 Distribution of Attributes in the Tables.....	173
5.5.5 The CITY Benchmark Database Schema and Creation Rules.....	173
5.5.6 Table Size (Number of rows per table).....	178
5.6 The CITY Benchmark Transaction Script.....	178
5.6.1 The CITY Benchmark Basic Database Operations.....	181
5.6.2 JOIN Operation.....	182
5.6.3 JOIN Transaction Complexity.....	182
5.6.4 Number of Retrieved Rows per Transaction.....	182
5.6.5 Background Workload.....	183
5.6.6 The CITY Transactions Workload.....	183
5.6.6.1 Qualified Retrieval.....	184
5.6.6.2 Sequential Retrieval.....	185
5.6.6.3 JOIN Two Tables.....	185
5.6.6.4 Update a Row in DBUPD.....	185
5.6.6.5 Insert a Row in DBINS.....	185
5.6.6.6 Sequence of Operations Execution in The Benchmark Script.....	186
5.6.7 Control of Transactions Execution.....	186
5.6.8 Random Number Generators.....	187
5.7 Requirements of Transaction Processing Council.....	188
5.8 Summary.....	189

CHAPTER 6 PRELIMINARY TEST AND VERIFICATION of The CITY BENCHMARK

6.1 Introduction	191
-------------------------------	------------

	Page
6.2 Test of the Main Characteristics of The CITY	
Benchmark.....	191
6.2.1 Reproducibility of the Benchmark Measures	192
6.2.1.1 Clock Adjustment and Eliminating First Transaction Effect	193
6.2.1.2 Reproducibility of the Benchmark Measures Within a Single Loop	194
6.2.1.3 Reproducibility of the Benchmark Runs.....	198
6.2.1.4 Duration of Loops Time	198
6.2.1.5 Test of Transaction Atomicity.....	199
6.2.1.6 Test of Transaction Consistency.....	199
6.2.1.7 Test of Transaction Isolation	200
6.2.1.8 Test of Transaction Durability.....	200
6.2.2 Test for Scalability of the CITY Transaction.....	200
6.2.2.1 Transaction Scalability VS Usage Cost.....	201
6.2.2.2 Scalability of The CITY Transaction in Comparison To Different Mixes in Two Standalone PC Environments.....	203
6.2.2.3 Scalability Ratio of The CITY Transaction in Comparison To Different Mixes in Two Standalone PC Environments.....	205
6.2.3 Simplicity of Construction and Clarity Code	207
6.2.3.1 Flexibility and Simplicity of Construction.....	207
6.2.3.2 Clarity of Code.....	207
6.2.3.3 Interpretation of the Results	208
6.3 Comparison between the CITY Transaction and the TPC	
Transactions in a Standalone PC Environment.....	209
6.3.1 Comparison Between the CITY and the TPC Transactions' Scalability Levels	209
6.3.2 Comparison Between the CITY and the TPC Transactions' Scalability Ratios.....	211
6.4 Summary.....	212

CHAPTER 7 **LARGE SCALE TEST AND VERIFICATION of The CITY BENCHMARK**

7.1 Introduction	215
7.2 Reproducibility of the Benchmark Measures.....	216
7.2.1 Clock Adjustment and Eliminating First Transaction Effect.....	216
7.2.2 Reproducibility of the Benchmark Measures Within a Single Loop	217
7.2.3 Reproducibility of the Benchmark Runs	220
7.2.4 Duration of Loops Time.....	222
7.2.5 Duration of Full Test in Multi-User Environment.....	223
7.2.6 Test of Transaction Atomicity	224
7.2.7 Test of Transaction Consistency	224
7.2.8 Test of Transaction Isolation.....	224
7.2.9 Test of Transaction Durability	225
7.3 Portability of the CITY Benchmark.....	225
7.3.1 Hardware and Software Independence	226
7.3.2 Database Management System Independent	227
7.4 Large Scale Test of the Scalability of the CITY Transaction	227
7.4.1 Large Scale Test for Scalability.....	228
7.4.2 Transaction Scalability VS Usage Cost in a Range of Computer Environments.....	229
7.4.3 Scalability of The CITY Transaction in Comparison To Different Mixes in SUN SPARC Environment	230
7.4.4 Scalability Ratio of The CITY Transaction in Comparison To Different Mixes in Three Computer Environments	233
7.5 Comparability of The Benchmark Measures	234
7.6 Summary.....	236

CHAPTER 8 DISCUSSION

8.1 Introduction	239
8.2 Comparison between the Behaviour of the CITY Transaction and the Behaviour of the TPC Benchmarks A and B Transactions	240
8.2.1 Comparison Between the Scalability Levels of the CITY Transaction and the TPC (A and B) Transactions	241
8.2.2 Comparison Between the CITY and the TPC Transactions' Scalability Ratios.....	243
8.2.3 Comparability of Transactions' Measures Between DBMS.....	245
8.2.4 Compliance of The TPC Benchmark with Future Trends of DBMS	247
8.3 Comparison between the CITY Benchmark and the TPC-C Benchmark	249
8.3.1 Comparison between the Cost of the CITY Transaction and the Cost of the TPC-C Benchmark Transaction.....	249
8.3.2 Database Size of the TPC-C Benchmark	252
8.4 Utilisation of The CITY Benchmark.....	252
8.5 Summary.....	253

CHAPTER 9 CONCLUSION AND FUTURE WORK

9.1 Conclusion.....	256
9.2 Contribution of The Work.....	258
9.3 Limitations of The CITY Benchmark.....	259
9.4 Future Work.....	261

10. REFERENCES

	Page
APPENDIX A: Create The CITY Benchmark Tables	273
APPENDIX B: The CITY Benchmark Tables Load	284
APPENDIX C: The CITY Benchmark Transactions	293
APPENDIX D: Random Numbers Generators	315
APPENDIX E: Analysis of Variance	317
APPENDIX F: The City Benchmark Operation Book	322
APPENDIX G: Printing The City Benchmark Results	325
APPENDIX H: Letters From Organisations	328

Table of Figures

The thesis has the following figures:

	Page
 Chapter 3	
Fig. 3.1, Effects of three database operations	69
Fig. 3.2, The Local Authority average daily load	72
Fig. 3.3, The Airlines average daily load	73
Fig. 3.4, The Bank average weekly load	74
Fig. 3.5, The Bank average daily load	75
Fig. 3.6, Effect of table size on transaction response time	77
Fig. 3.7, Effect of row size on transaction response time	78
Fig. 3.8, Effect of unique index on transaction response time	79
Fig. 3.9, Effect of JOIN operation on transaction response time	81
Fig. 3.10, Effect of transaction output on transaction response time	83
Fig. 3.11, Effect of result display on transaction response time	84
Fig. 3.12, Effect of background workload on transaction response time	86
 Chapter 4	
Fig. 4.1, The business of the Local Authority	111
Fig. 4.2, Organisational structure of the Local Authority Computer Centre	112
Fig. 4.3, Processing Environment of the Local Authority	114
Fig. 4.4, Ratios of the Local Authority On-line Transaction Operations	117
Fig. 4.5, Ratios of Key utilisation	118
Fig. 4.6, Organisational structure of the Airlines computer centre	121
Fig. 4.7, Processing Environment of the Airlines computer centre	122
Fig. 4.8, Ratios of the Airlines On-line Transaction Operations	126
Fig. 4.9, The Airlines On-line transaction I/O time ratios	127
Fig. 4.10, The Airlines On-line transaction JOIN ratios	128
Fig. 4.11, The Airlines On-line transaction nested selects ratios	128
Fig. 4.12, The Bank branches to data centres communication	133
Fig. 4.13, The Bank data centre	134
Fig. 4.14, Customer Account in relation to running applications	135
Fig. 4.15, The Bank On-line transaction flow	136
Fig. 4.16, Ratios of the Bank On-line Transaction Operations	140
Fig. 4.17, Ratios of the Bank ATMs Transaction Operations	141
Fig. 4.18, The Bank On-line transaction ratios of CPU and I/O times	142
Fig. 4.19, The Bank On-line transaction JOIN ratios	143
Fig. 4.20, The Bank On-line transaction nested selects ratios	144
Fig. 4.21, Average of On-line Transaction Operations in all environments	146
Fig. 4.22, Comparison between the TPC-A, TPC-B and On-line transaction	148
Fig. 4.23, The full TPC-C database operations	150
Fig. 4.24, The On-line TPC-C database operations	151
Fig. 4.25, Comparison between complete TPC-C and On-line transaction	152
Fig. 4.26, Comparison between On-line TPC-C and On-line transaction	152
Fig. 4.27, Average number of JOINed tables	155
Fig. 4.28, Average number of nested selects	156
Fig. 4.29, Average of On-line transaction I/O time ratios	157

Chapter 5

Fig. 5.1, The relations between the CITY benchmark tables	175
---	-----

Chapter 6

Fig. 6.1, First transaction response time in PC environment	194
Fig. 6.2, Spread of CITY measures in PC environment	196
Fig. 6.3, Frequency values of CITY measures in PC environment	197
Fig. 6.4, Scalability of The CITY measures in PC environment	201
Fig. 6.5.1, The CITY transaction scalability level against other transactions	204
Fig. 6.5.2, The CITY transaction scalability level against Join transactions	205
Fig. 6.6, The CITY transaction scalability ratio against other transactions	206
Fig. 6.7, Comparison between the CITY and TPC-A, B scalability levels	211
Fig. 6.8, Comparison between the CITY and TPC-A, B scalability ratios	212

Chapter 7

Fig. 7.1, First transaction response time in SUN SPARC environment	217
Fig. 7.2, Spread of CITY measures in SUN SPARC environment	218
Fig. 7.3, Frequency values of The CITY measures in a SUN SPARC environment	219
Fig. 7.4, Sample of The CITY measures from different runs	221
Fig. 7.5, Comparison between The CITY measures based on test loop duration	223
Fig. 7.6, Scalability of The CITY measures in several computer environments	229
Fig. 7.7, The CITY transaction scalability level against other transactions	232
Fig. 7.8, The CITY transaction scalability level against JOIN transaction	232
Fig. 7.9, The CITY transaction scalability ratio against other transactions' mixes	234
Fig. 7.10, Comparison between two DBMS using The CITY benchmark	235

Chapter 8

Fig. 8.1, Comparison between CITY and TPC-A, TPC-B in several computer environments	242
Fig. 8.2, CITY and TPC-A, TPC-B in multiprocessors environments	243
Fig. 8.3, Scalability ratio of CITY and TPC-A, TPC-B in several computer environments	244
Fig. 8.4, Scalability ratio of CITY and TPC-A, TPC-B in multiprocessors environments	245
Fig. 8.5, Comparing DBMS using both The CITY and The TPC-A and TPC-B	246
Fig. 8.6, Effect of row size on transaction response time (future trends)	248
Fig. 8.7, The TPC-C complete transaction workload in SUN SPARC environment	250
Fig. 8.8, The TPC-C on-line transaction workload in SUN SPARC environment	251

Thesis Tables

	Page
Chapter 1	
Table 1.1, The different test environments	15
Chapter 3	
Table 3.1, Three database operations in PC environment	66
Table 3.2, Three database operations in SUN SPARC environment	67
Table 3.3, The TPC-A against different table sizes	67
Table 3.4, Comparison between an IBM and Tandem	68
Table 3.5, Steps of calculating ANOVA	100
Table 3.6, Example of two transactions' workloads	103
Chapter 4	
Table 4.1, Ratios of the Local Authority On-line Transaction Operations	116
Table 4.2, Ratios of the Airlines On-line Transaction Operations	126
Table 4.3, Ratios of the Bank On-line Transaction Operations	140
Table 4.4, Ratios of the Bank ATMs Transaction Operations	141
Table 4.5, Average of On-line Transaction Operations in all environments	146
Table 4.6, Comparison between the TPC-A, TPC-B and typical On-line transaction	147
Table 4.7, Comparison between the TPC-A and ATM transaction	149
Table 4.8, Comparison between the TPC-C and On-line transaction	150
Table 4.9, Average number of JOINed tables	155
Table 4.10, Average number of nested selects	156
Chapter 5	
Table 5.1, The CITY benchmark TPs tables' sizes	165
Table 5.2, DB100	176
Table 5.3, DB200	176
Table 5.4, DB300	177
Table 5.5, DBUPD	177
Chapter 6	
Table 6.1, Spread of CITY measures in PC environment	196
Table 6.2, Frequency values of CITY measures in PC environment	198
Table 6.3, Effect of loop time on CITY measures in PC environment	199
Table 6.4.1, The CITY transaction scalability level against other transactions	203
Table 6.4.2, The CITY transaction scalability level against other transactions	204
Table 6.5, The CITY transaction scalability ratio against other transactions	206
Table 6.6, Comparison between The CITY and TPC-A, TPC-B scalability level	210
Table 6.7, Comparison between The CITY and TPC-A, TPC-B scalability level	210
Table 6.8, Comparison between The CITY and TPC-A, TPC-B scalability ratio	211

Chapter 7

Table 7.1, Spread of CITY measures in a SUN SPARC environment	218
Table 7.2, Frequency values of CITY measures in a SUN SPARC environment	219
Table 7.3, Sample of The CITY measures from different runs	220
Table 7.4, Calculation of ANOVA of The CITY measures from different runs	221
Table 7.5, Calculation of ANOVA of the CITY measures from different runs	221
Table 7.6, Comparison between CITY measures based on test loop duration	222
Table 7.7, The CITY transaction scalability level against other transactions' mixes	231
Table 7.8, The CITY transaction scalability level against other transactions' mixes	231
Table 7.9, The CITY transaction scalability ratio against other transactions' mixes	233
Table 7.10, Comparison between two DBMS using The CITY benchmark	235

Chapter 8

Table 8.1, Sizes of rows in row test of row size effect	248
Table 8.2, Comparison between the CITY measures and full TPC-C measures	250
Table 8.3, Comparison between The CITY measures and on-line TPC-C measures	251

Acknowledgements

I would like to thank my parents for their support and continuous encouragement and my brother and sister for all the help they have offered.

I would like to thank my supervisor, Norman Revell, for his useful guidance, tolerance and invaluable advice during this research study.

Also I would like to thank my second supervisor, Simon Grant, for his optimism, comments and constructive criticism.

I would like to express my deepest gratitude to Prof. Huissin Abdel Aziz and Dr. Galal Ismail for their continuous encouragement.

I would like to thank Mr. David Notley, Department of Statistics, CITY University, for his useful advice in statistical matters.

For allowing me to either to study their environments or to run my tests on their computers I would also like to thank the following people: Vince Padi from LOLA; Peter Blundell and Roger Penton from British Airways; Andy Pearse from NatWest; Clare Whitiker and Dave Tomlinson from AT&T (NCR 3600 and Teradata branch); Galal Hasan from Brunel University; Walaa Mohamed and George Karakitsos from UNL.

I am grateful to Dr. Allan Cook, Mr. Alwyn Jones, Prof. Owen Hanson, and Prof. Alistair Sutcliffe for their encouragement, guidance and support.

Additionally, I would like to thank all my friends and colleagues in the CITY University specially Akmal Chaudhri, Tony Valsanidis, Dr. Julie McCann, Dr. Robin Swain, and Anwar Hegab for their help throughout this research and Dr. Neil Maiden and Uma Patel for using their printer to print this work.

Finally, I would like to thank the secretaries in the general office of Department of Business Computing at CITY University and Beverley Copeland the computer laboratory manager for their assistance during this research.

Declaration

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgements.

Statement of Contribution

This disclaimer is to state that the research reported in this thesis is primarily the work of the author and was undertaken as part of his doctoral research. Referenced papers of which the student is not the sole author represent the role of the supervisor in the research, to direct the work and enhance the written style of these papers.

Abstract

This thesis presents the CITY benchmark, a database benchmark that fairly represents On-Line Transaction Processing (OLTP) environments. It analyses the most widely used benchmarks in general putting more emphasis on the Wisconsin benchmark and the Transaction Processing Council (TPC) benchmarks (TPC-A, TPC-B and TPC-C) in particular. It also presents an empirical approach to examine the workload of those benchmarks and discovered several technical limitations in their scripts. The thesis also presents an investigation of on-line transactions in large database environments. The tested environments were three of the largest organisations in the UK, those organisations were different in objectives and activities. The investigation identified on-line transaction behaviour and defined the salient characteristics of databases in high-volume transaction environments. The findings from those studies established the basis of a transaction and set of tables that are representative of them. The CITY benchmark design is directly driven from the findings from the empirical studies. The benchmark design took into consideration all the critiques directed towards the TPC benchmarks A, B and C. It is the first benchmark that is designed as a result of studying the behaviour of on-line transactions and databases in large database environments. The CITY benchmark is mainly designed to test and compare database systems performance in high-volume transaction environments (OLTP).

The work revealed the salient characteristics of large database environments and identified a typical behaviour of on-line transaction in OLTP environments. This research has clearly shown that the TPC benchmarks are not representative to the domain of high-volume transactions environments (OLTP) and it explained why they could be misleading if used to test database management systems in this domain. Additionally, this thesis presents a database performance evaluation methodology that is based on in-depth studies in large database environments.

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

For several years, there has been an awareness that the existing database performance benchmarks have some technical limitations, and the most important one is the simplification of the benchmark transaction mix. These limitations have left database users with only an intuitive idea of the behaviour of their database management systems. To achieve realistic metrics, a database system should be tested with a load pattern as close as possible to real life. It is clearly desirable that any accurate performance measurement should be representative of the overall picture of the performance of the database environment under test. Some ideas of the type of information contributing to this achievement are: classification of the dominant processing tasks; their transactions; their relative frequencies; the size of the database; and patterns of behaviour to be expected. To this end, several in-depth studies were conducted in several large UK organisations. Those studies aimed to define the salient characteristics of large databases and to identify typical transaction behaviour in high-volume transaction environments. This thesis presents the CITY benchmark, whose design is based on the results of those studies.

1.1 Some Performance Concepts

Several database performance evaluation methods have evolved over years, but benchmarks have been accepted by the database industry as the most accurate performance evaluation method. Generally, three main types of benchmarks have evolved: user-designed benchmarks, database manufacturers' benchmarks and industry-standard benchmarks.

Because different users value different performance aspects, some users might set their own benchmark test to satisfy their own requirements. A good example is presented by Gleser [GLES81], where they decided to convert their running software to a fully supported database management system (DBMS).

Database manufacturers run different types of benchmarks. They run large and comprehensive tests to examine every aspect of their database management systems before its large scale marketing. Usually those benchmarks are very expensive to run and testing takes a long time to complete; moreover, results are usually confidential and seldom get published.

Industry standard benchmarks represent the common basis of comparison between database management systems that can be used by both manufacturers and users. The database industry has seen several attempts at standard database benchmarks. The most famous benchmarks are the Wisconsin benchmark [DeWi85], and the Debit/Credit (TP/1) benchmark [ANON85]. The Wisconsin benchmark has taken a smaller role over the years and TP/1 has become the most frequently quoted in the market.

The development of a standard benchmark posed several problems. The main one was the difficulty of comparing different database systems due to reasons such as: the absence of database performance theory; the difficulty in measuring qualitative aspects of databases; and the fact that quantitative measurements are not standard for different database systems. The research in the database area has offered several trials but the database industry is still seeking a more comprehensive method of measuring database performance.

The other problem that faces developers is that the criteria for measuring database systems performance depend largely on the role that system is to play in the organisation. For example, if the database mainly serves Decision Support Systems (DSS) transactions, the performance criteria might be focused on direct transaction cost with resource utilisation. On the other hand, on-line systems focus on transaction response time which is usually an essential factor for such systems. Another problem is that different users value different performance factors. For example, a user with mainly batch environments will care little for background workload. On the other hand, in on-line environments, background workload will have a high priority. Consequently, defining meaningful performance concepts for various environments requires finding a common basis between those environments.

Since the introduction of the relational database model by Codd [CODD70], several database researchers such as Hawthorn [HAWT82], Benigni [BENI84], Boral [BORA84], DeWitt [DEWI85], Youssef [YOUS86c], and others whose work is presented in chapter two tried to develop a standard benchmark for the database industry. Gradually, use of the Debit/Credit (TP/1) benchmark [ANON85], and later

the TPC benchmarks A and B [TPC 89, TPC 90], has become the standard practice in the database industry. They were all constructed to quantify and compare the throughput and price/performance ratio of various transactions processing systems. The Debit-Credit (TP/1) modelled an Automatic Teller Machine (ATM) environment and simulates random withdrawals being made against bank accounts at a large bank environment. It includes simulating many terminal users, their "think time" and network traffic time. TP/1 was mainly designed to measure the performance of transaction processing in on-line transaction processing (OLTP) environments.

Because TP/1 did not contain specific guidelines for implementation [TPC 92], it was not standard in its application and different users implemented different versions of the benchmark. In 1988 eight software and hardware companies set up the Transaction Processing Council (TPC) to build a standard benchmark. The council modified the TP/1 benchmark into an industry standard benchmark TPC-A [TPC89, TPC 90, GRAY91], and later TPC-B [TPC 91]. Both benchmarks have specific guidelines for the measurement of performance and price. The TPC benchmarks A and B have become the database industry standard, but both inherited the same limitations as TP/1. They are based on the same database operations of TP/1 with some added implementation standard guidelines. As in the case of TP/1, the full implementation of the TPC benchmarks is still too expensive and users are always constrained by the resources available in their environments.

The transaction processing council published a draft specification of a new benchmark called the TPC-C benchmark in December 1991 [TPC 91]. The TPC-C is a large benchmark that consists of five programs which access nine tables contain around 40,000,000 rows. The benchmark includes two long batch transactions. When the TPC-C benchmark script was tested, it was too expensive to run in terms of response time. The results of those tests are presented in chapter seven. An example of expensive benchmarks was published by Strawser [STRA84], and that benchmark has never been used due to its high cost. Chapters six and seven show that the high cost of a transaction does not automatically imply the efficiency of that transaction load. One of the most important factors in any benchmark success is this subtle combination of its transaction mix against cost. Additionally the TPC-C benchmark encapsulated some of the TPC-A and TPC-B benchmarks problems such as including aspects that do not directly relate to DBMS such as think time and testing network effect. Chapter two and chapter three discuss in detail the specifications and limitations of the TPC-C that have hindered users from using it.

Hoffman and Caniano [HOFF87, CANI88], had criticised TP/1 for being too expensive to implement in its full form, and every database user implemented a different version of it. That variation in the TP/1 implementation methods meant that the results of different benchmark tests were not comparable to each other. The TPC benchmark did not have specifications such as: complete specification; verifiable results; full disclosure of system configuration; benchmark methodology; and no fair means of comparing one system against another. The TPC benchmarks (A and B) have been criticised for being difficult to implement in their full form and include aspects like network protocols and think-time that do not specifically relate to DBMS performance. Additionally, similar to the TP/1, the methodology of the TPC benchmarks (A and B) do not also supply a fair means of comparing between systems.

The present research applies an empirical approach to examine the transactions of the TPC benchmarks. The results of the studies demonstrated the limitations of the TPC benchmarks. Perhaps the most important observation is that they are not representative of real database systems transaction operations, even of the ATM transaction they are supposed to simulate. The TPC benchmarks' limitations were discussed by Revell and Youssef in previous work [REVE90, REVE92a]. Revell and Youssef [REVE92b] mainly criticised the TPC benchmarks because they do not provide a realistic characterisation that can be set as the target for the benchmarking process. As benchmark results are representative of those types of transactions actually included in the benchmark set, the results from a benchmark can not be mapped to a domain that the benchmark does not represent and it is impossible to generalise those results to all kinds of systems transactions. The empirical studies revealed that the basic database operations of the TPC benchmarks are widely different from the basic database operations of the high-volume transaction environments and showed that there is great doubt that they could realistically represent that environment. In this thesis, the term "the TPC benchmarks" implies the three TPC benchmarks, TPC-A, TPC-B and TPC-C unless otherwise stated.

1.1.1 Some Indices of Performance

A number of performance indices are generally accepted in the computer and database industry. Those indices include things such as: capacity; response time; throughput rate; overhead percentage; components overlap measure; software time measure; system utilisation measures; and transaction cost.

Over the years most database benchmarks relied on just two performance indices, transaction response time and transaction throughput. Transaction response time is measured as the time elapsed between submission of a transaction to DBMS and receiving the result. Transaction throughput is measured as the average number of transactions completed per unit of time.

1.1.2 Analytical Models

Analytical models are mathematical representations of database systems and are mainly used for evaluation and analysis of the performance of database components [BORO79]. Modelling is widely used and many database performance models were developed varying in degree of accuracy, cost to build and to run. The majority of those models are concerned with evaluating some particular database parameters that run on a particular system.

Analytical models are useful tools for capacity planning and usually techniques such as queuing models are one of the least costly methods of evaluating system performance. They are also useful tools when used in database design stage, but due to considerations such as simplification level, they are less useful in database selection process. The advantage of algorithmic models is they offer a better understanding of the systems under test, and a good representation of variable interrelations in those systems.

Because it is impossible to model everything in real life, the main disadvantage of modelling is it implies some level of simplification and approximation [FERR83]. The quality of models is measured by the level of simplification involved in building the model. In addition to the previous point, the other disadvantages are:

- their development requires a great deal of time and effort;
- it is difficult to analyse random influences of systems' variables in analytical models.

Analytical models are generally divided to three main groups according to the mathematical tools used. These model groups are statistical, graphical and algorithmic.

1.1.2.1 Statistical Models

This group includes models mainly based on queuing theory and its interest focuses on distribution of waiting times for different tasks, queue length, and similar aspects [BORO79].

The importance of this type of models lies in the possibility of investigating the exact relationship among the various variables in the system, as well as forecasting the effect of the software and hardware modifications. The main disadvantage of this method is the reliance on statistical assumptions which may largely call into question the validity of the results achieved.

1.1.2.2 Graphical Models

In graphical models, the system dynamics are represented by a graph, where each node is a transaction dominated by input conditions (entering arrows) and generating output conditions (represented by the outgoing arrows), which then constitute several input conditions to further nodes. In such a model it is customary to distinguish between two kinds of conditions: conjunctive conditions (AND operator), where each input condition must be satisfied for a transaction to occur; and disjunctive conditions (OR), where one condition is enough to allow a transaction to come through. Graphical models serve two main purposes:

- estimating execution times for systems represented by the model.
- as a mean to a new and better structuring of systems test.

The theoretical basis for this kind of analysis is found in graph and automata theory [BORO79]. The main advantage of those models lies in affording a better, deeper, and more comprehensive view of the systems tested, thus leading to their improvement. They are limited to software and not hardware evaluation and demand considerable effort for each program modelled.

1.1.2.3 Algorithmic Models

In this model, we find analytical investigations of computer performance when working under a given control algorithm and defined workload; for instance, the

behaviour of a program under conditions of memory saturation (memory required is greater than memory available), where memory is allotted by a paging algorithm.

1.1.3 Simulative Models

Simulation can be applied to testing an entire system or subsystems and components at almost any level or detail, for purposes of selection, planning, or optimisation. It is only applicable when the system or processes are at least partially known.

However, even though it is expensive and depends on deep analysis of the problem and careful definition of the physical and logical processes taking place in the system, simulation is one of the most flexible methods in evaluating performance. It may be said that its most valuable condition lies in the discipline it imposes on its user, besides providing good models.

Among all performance methodologies, it failed the largest number of times [BORO79]. This is because it has been the most extensively used method without proper preparation or without proper understanding of its nature and the conditions required to implement it. It is therefore often tried in trivial cases or without properly structured models.

1.1.4 Benchmarks

Benchmarking is the first technique that takes into account not just the DBMS involved but all the systems' parameters included in the tested environments. Those parameters include the software involved, the hardware platform and the communication network. Benchmarks have been widely used to evaluate and select database management systems, they are less useful in diagnosing database problems and improving database performance. The following sections discuss the nature of benchmarks, selection of their transaction mix and their main disadvantages.

A review of the widely used benchmarks will be discussed in chapter two. For each benchmark the transaction script, database used and technical limitations will be discussed.

1.1.4.1 The Role of Benchmarks

This technique is most useful for selection evaluation and to a lesser extent applicable for forecasting behaviour of planned systems or supervision of an existing system. The only way the benchmark method can be applied to software or hardware design is by comparing a performance index before and after for a set of benchmark loads. The main advantage of the benchmark method is its realistic evaluation of the tested systems as it tests real databases, running under actual operating system and real peripherals and programs. This way it overcomes modelling problems represented as the need for approximations, estimations, simplifications, and all of the other assumptions required to build a model. The proper use of benchmarks is based on applying the following requirements:

- a. Running a set of database transactions that simulates a user's operational scenario and environment.
- b. Requiring the exact system component of the original benchmark test.
- c. Requiring the exact manner in which the vendor timed the original benchmark test.
- d. Analysis of the benchmark results to extract the essentials of the systems performance.

1.1.4.2 Selecting a Benchmark Mix

The number of transactions that may be performed on a database is in principle infinite. There are several methods for selecting the mix of transactions that will form a benchmark. The problem is that the performance of two databases varies with the nature of the benchmark script, and accurately comparing database performance will not be possible due to the absence of the basis upon which to compare. Accordingly, different comparisons based on the results from different benchmarks' on number of databases are impossible due to the variety in transaction mixes.

Generally, the criteria for the performance of database systems depend on the role that system is to play in the organisation and results from one benchmark can not be mapped to domains that are not represented in their mix. Usually a benchmark mix will comprise a limited number of transactions that will be run number of times against one or more databases to serve the required test. In this way that transaction mix can produce a performance figure that can be used to calibrate and compare that environment against other environments.

1.1.4.3 Weaknesses of Benchmarks

The main disadvantages of the benchmark method are the following [BORO79, FERR83, STON85]:

1. Benchmark results are representative of those types of transactions actually included in the benchmark set. It would be impossible to generalise those results to all kinds of environments, it is therefore vital to select a set that reflects the expected work load of real environments.
2. Benchmark results could be misleading if the test applied partial or modified work load.
3. Building a representative benchmark involves considerable time and effort.
4. The need for a number of benchmark test runs can lead to significant expense.
5. There might be some practical difficulties in replicating the same benchmark runs due to differences in configuration or the unavailability of all the required parameters.

1.1.4.4 The main Characteristics of a Good Benchmarks

The main characteristics of a good benchmark have been discussed by several researchers in the performance field such as Ferrari and Gray. Ferrari [FERR78, FERR83] describes a good benchmark as being written in a high level language, properly debugged, its files being reproducible on all systems, and its documentation being well written so that users can reproduce the same results. Gray [GRAY91] characterises a good benchmark by being relevant, portable, scalable and simple.

The present author emphasis on one more characteristic that is representative to the tested domain. Representativeness could be defined as the accuracy of the benchmark workload in simulating the tested environment transactions. This characteristic is added because users can not generalise the results of a specific domain transaction to all other domains.

1.2 Problem Definition

The TPC benchmarks [TPC89, GRAY91], were constructed to quantify and compare the throughput and price/performance ratio of various transaction processing systems. The TPC-A and TPC-B benchmarks measure transactions per second from the time a teller issues a request to the time the customer receives acknowledgement that the transaction has completed. The TPC-C simulates an inventory control system and measures transactions per minute. They include simulating many terminal users, their "think time", and network traffic time.

However, those benchmarks are difficult to implement in their full form, and include aspects, like network protocols and think-time, that do not specifically relate to DBMS performance. The TPC benchmarks (A and B) have become a fairly standard practice. To market any DBMS, their vendors have to advertise it as the number of TPC-A that DBMS can produce. But due to technical constraints, such as insufficient space, absence of terminal emulator or the non-existence of X25 network, it is difficult to run the TPC-A benchmarks in their standard form. Each vendor has had considerable differences in implementing the TPC-A as it saw fit, and so a comparison based on TPC-A benchmark results can be misleading. Chapter three discusses in detail the different ways of implementing the TPC benchmarks.

Additionally, because the scripts of TPC benchmarks (A and B) are relatively simple, some vendors have tailored their DBMS in a way that shows their product as the best. Those limitations have been discussed by Revell and Youssef [REVE90] where they traced those limitations to the lack of background study before the benchmark design stage. The main criticism of the TPC benchmarks is that they do not provide a realistic characterisation that can be set as the target for the benchmarking process, moreover, they do not even provide a model of the ATM systems whose performance they are supposed to simulate [REVE92b].

Generally, the TPC benchmarks suffer from several technical limitations that will be discussed in details in chapter three, those limitations are the following.

- Inconsistency in the implementation of TPC benchmarks. Some other has chosen to omit or modify portions from the TPC benchmarks without regard for the preservation of the original TPC definition. An example was given by Fox [FOX 89].

- Technical limitations of the transactions of the TPC benchmarks. The database industry and most researchers in the database performance area have criticised the TPC transactions for being too simple to apply realistic workload on the tested systems. This assumption has been confirmed by the results of this research.
- Inconsistency of the comparisons of the TPC benchmarks. This is due to the nature of the mix of the TPC benchmarks. The TPC transaction database operations are too simple to represent all the domain of OLTP and one system may be excellent at performing one transaction type and behave in a different manner when performing another transaction type.
- The future of new DBMS carries concepts such as OODB, DSS, and graphical databases where row length and sizes of databases are large. The TPC benchmarks as implemented will not be sufficient to represent those environments.

Due to the previous limitations, the problem can be defined as the lack of a representative benchmark that can adequately test the performance on DBMS in OLTP environments.

1.3 Research Objectives and Approach

In the absence of performance metrics, the majority of database users will have an intuitive idea of the performance of different DBMS, and a performance metric will support such intuition with performance results. This research results will allow DBMS users to obtain accurate and representative measure of what they get against what they pay for. It will also help them choose the most cost-effective system for the required needs. In addition, this methodology will supply the DBMS companies with a measure of a real life standard workload based on studying large systems in real life.

The main research objective is to create a comprehensive benchmark methodology that takes into account the behaviour of typical DBMS applications in high-volume transaction environments. That methodology should represent the widest variety of systems variables that exist in real life environments. That methodology should also take into consideration all the limitations discussed in previous sections. The research objectives could be summarised in the following:

- define the main characteristics of databases in large database environments;
- identify the typical behaviour of on-line transactions in large database environments;
- build a comprehensive methodology that is driven directly from the previous definitions and realistically represents real life environments.

1.3.1 Empirical Approach to Problem Solving

Generally, to achieve realistic metrics, a database system should be tested with a load as close as possible to that which it will be running in real life. The empirical approach was recommended by Ferrari [FERR83]. He recommended that a test workload consisting of the load that is actually processed during a measurement session is the most representative type of workload. Additionally, Turbyfill [TURB88], in her PhD thesis discussed future directions in database performance, recommended that any future benchmark should characterise real life workload to develop metrics. The information required to this achievement are: identify the dominant processing tasks; what queries will be run; what is the relative frequency of each query; what is the size of the database; what patterns of behaviour are expected.

To this end, the present author conducted a series of field studies to identify the previous factors. This thesis presents research work that adopted field studies and case analysis techniques to identify the main characteristics of on-line databases by examining real database performance factors such as:

- table size;
- row size;
- common types of attributes;
- distribution of attributes;
- on-line transaction database operations;
- on-line transaction complexity;
- on-line transaction JOIN operations;
- on-line transaction resource utilisation;
- on-line environment background workload;

Those factors have been discussed by many researchers in the performance field such as Ferrari [FERR78, FERR83], Dongara [DONG87], and Stonebraker

[STON85]. The analysis of those performance factors in several different environments should defined a common basis between those environments. The results from that analysis can supplement the data from the live environments to evaluate their performance.

1.3.2 Specification of the Research Domain

This research domain is the set of high-volume transaction environments. Equally some people call it "On-line Transaction Processing (OLTP)" environments. The importance of high-volume transaction environments (OLTP) is due to its share of the database industry; this environment is the dominant environment at the database market and sets the rules for database standards.

High-volume transaction environments (OLTP) are those environments that support a large number of on-line terminals that originate heavy transaction load on the running environments. The number of terminals in those environments is in the thousands and the number of transactions is in thousands of transactions per hour. Those environments support large number of databases that are characterised by their large sizes. Examples of such environments include banks, airlines, government organisations, and large business organisations.

1.3.3 Development Basis of the CITY Benchmark

The CITY benchmark is designed based on the findings from the empirical studies that were conducted in several large database environments. The empirical studies collected in-depth information about the systems under study by monitoring the systems for a representative period of time, studying systems documents, studying systems output and interviewing staff. The studies examined over 4800 different applications and investigated over 40,000,000 on-line transactions accessing around 5000 on-line databases. The studies revealed a common patterns of behaviour among the studied organisations despite the difference between those organisations in activities. The CITY benchmark represents those common patterns of behaviour. Chapter four discusses those studies in detail and chapter five discusses the design rules of the CITY benchmark.

1.3.4 Test Capabilities of the CITY Benchmark

As mentioned before, benchmarks can be used to fulfil several objectives depending on the benchmark design rules. Database benchmarks can be used for three main purposes, those purposes are the following:

- Selection and comparison. This approach deals with the problems of comparing one DBMS to another; selecting processing environment for a specific DBMS; improving the running system performance by testing and modifying the running system to enhance its performance. This type of benchmark runs in uncontrolled environments to simulate real life behaviour. This approach produces one figure that represents the overall performance of the tested system.
- Diagnostic studies of running systems. This approach is implemented by applying comprehensive benchmarks that run in controlled environments to detect a specific bottleneck. The main objective of those benchmarks is to produce a figure for each of the tested system activities. Usually those benchmarks include long and very expensive testing process.
- Benchmarks that are used in design stage. Those benchmarks are function oriented benchmarks, they produce micro level results and mainly used to test specific model hypothesis. Those benchmarks run just in the DBMS design stage and usually their results are used for further enhancement of specific data DBMS when developed from one version to the next.

The CITY benchmark main target is to be used for selection and comparisons. The benchmark will be mainly used to evaluate different DBMS performance on the same architecture or on different architectures.

The CITY benchmark test domain is high-volume transaction environments, that are fairly similar to those environments covered by the empirical studies. The benchmark specifically represent the behaviour of on-line transactions in those environments.

1.3.5 Test and Verification of the CITY Benchmark Results

Verification of the CITY benchmark results addressed problems such as: what performance factors to test; how to test those factors; and the approach of implementing

the benchmark test. The first choice was running the tests in a controlled environment, hence the benchmark variables should be the only test factors involved in the benchmark test. That allowed the replication of the benchmark results.

The second question was "What are the factors to be measured?". People such as Gray [GRAY91] and Ferrari [FERR78, FERR83] discussed several benchmark factors, those factors are the following:

- the benchmark measures are reproducible;
- the benchmark is hardware independent;
- the benchmark is software independent;
- the benchmark is DBMS independent;
- the benchmark is application independent;
- the benchmark measures are comparable between machines and DBMS;
- the benchmark is clear and simple to construct;
- the benchmark usage cost is low.

The CITY benchmark test was conducted in several computer environments ranging from standalone PC to large mainframes and multiprocessor environments. The benchmark test took place in the several environments using different operating systems and different DBMS. The different test environments are presented in Table 1.1.

Test Environment	Operating System	DBMS
Standalone 386 PC	MS DOS	ORACLE version 5
Standalone 486 PC	MS DOS	ORACLE versions 5 & 6
SUN 386i workstations	UNIX	ORACLE version 6
SUN SPARC workstations	UNIX	ORACLE version 6
SUN SPARC in a network environment	UNIX	ORACLE version 6
VAX 4000	VMS	INGRES version 6;
IBM 3090 mainframe	MVS	DB2 for large mainframes
Teradata System 3 multiprocessing	VM & UNIX	Teradata DBMS
Teradata System 4 multiprocessing	VM & UNIX	Teradata DBMS
NCR 3600 multiprocessing	VM & UNIX	Teradata DBMS

Table 1.1, The different test environments

The benchmark verification process was conducted on two phases. The first phase was the preliminary test in two completely controlled standalone PC environments. The second phase, was large scale verification in all the other environments. The first phase did not stop by the start of the second phase but was continued while the second phase was done. The first phase took on all around 8000 computer hours, where two PC were working 24 hours every day including weekends for more than six month. The second verification phase took around 2000 computer hours, where each run was replicated for at least fifteen times and each run took on average six hours to be completed.

The interpretation and verification of the benchmark results required applying some statistical and mathematical techniques. The results were verified using the variance, one way analysis of variance and two way analysis of variance. The different results were compared used difference quotient and factor of relative change.

In all environments, except for the mainframe environment, the tests were conducted in a single user mode to eliminate the effects of non-controllable workload and to isolate the benchmark results.

There is one case in this research where the CITY benchmark was used to detect bottlenecks, but that required the existence of several other software tools that allows this function to be performed. But in all other environments implementing diagnostic tools in association with the benchmark was not performed.

Chapters six and seven discuss in details the verification process of the CITY benchmark. Chapter six discusses the preliminary test of the benchmark results in two completely controlled standalone PC environments and chapter seven discusses the large scale test of the benchmark results in several industrial organisations.

CHAPTER 2

AN ANALYSIS OF DATABASE PERFORMANCE MEASURES

CHAPTER 2

AN ANALYSIS OF DATABASE PERFORMANCE MEASURES

2.1 Introduction

The wide spread of database systems has resulted in an increase in the number of new systems being used. Database systems have been implemented in many different forms, for mainframes, minicomputers and microcomputers. The performance of DBMS represents the main factors in the decision to use a certain database. In comparing database management systems (DBMS) an important factor is their performance and database selection is usually based on information that compares the performance of one database management system to another. Performance allows the user to measure potential system capability and thus helps in choosing the best database for the required needs.

Benchmarks have proved to be the most reliable method to measure different database management systems performance. They evolved over time from very simple trials to complex processes. The most quoted benchmark in the market was the Debit/Credit TP1 benchmark. The benchmark simulates a bank withdrawal transaction and measures the performance of database systems as the number of transactions per second, TPS. The benchmark was too flexible to provide comparable results and database vendors tailored the benchmark to produce any number of TPs they required. The need for industry standard benchmarks led to the development of the transaction processing council (TPC). The council modified the TP1 benchmark into an industry standard benchmark TPC-A, which has specific guideline for the measurement of performance and price.

Although TPC-A provides very specific guide-lines, the benchmark inherited all the problems of the TP1. The benchmark is difficult to implement in its full form and to complete it might take weeks or months. The benchmark results are tuneable and database vendors can tune the system configuration to produce any number of TPS they like.

This chapter discusses computer and database benchmarks. Computer benchmarks are briefly discussed as they occupy lower priority in this research but they show the early trials in the area of performance evaluation. Database benchmarks are discussed in greater detail. The early database benchmarking trials are presented separately as well as the activities of some research centres working in the database performance area. The most widely used benchmarks, the Wisconsin, the Debit/Credit (TP1) and the TPC benchmarks (TPC-A, TPC-B, and TPC-C) will be examined in greater depth.

In addition, this chapter discusses the difficulties involved with the utilisation of the presented benchmarks and explains why they were not as successful as expected in testing different systems performance. The chapter conclusion discusses the required work to overcome these difficulties and to produce a benchmark that is general enough to represent the domain of 'high volume transactions' environment.

2.2 Historical Background of Database Benchmarks

Benchmarks were born when a customer asked the IBM to characterise their system before agreeing to buy it. That resulted in the development of Gibson mix, eventually known as MIPS [GIBS70]. Several other computer benchmarks were developed since. But, computer systems benchmarks have never been good enough to measure database performance. They gave no indication of how systems might behave under different working load and did not give enough guidance to database performance pattern in transaction processing environments. This is because database performance is susceptible to different factors than just sheer hardware speed. These other factors are things like indexes and retrieval algorithms that can deeply affect transaction processing applications. Consequently, the database industry decided to conduct its own research to develop adequate database performance measures and several studies have been conducted to evaluate performance and test enhancements. Some general concepts were discussed in [BUTL87, DATE86, FERR78, FERR83, ULLM82]. Additionally, Deen provided a comprehensive understanding to the whole subject in [DEEN90].

Due to the inadequacy of computer benchmark to test DBMS, database benchmarks followed suit. The earliest trials was undertaken by the bank of America in 1973. They developed a transaction processing benchmark that uses one transaction type consists of 25 COBOL statements, and indicates the performance obtained when 95% of all transactions are processed in a second. They called that benchmark The

Debit/Credit. Eventually, that benchmark became the basis of the most widely used benchmark, the famous TP/1. There were two variations of The Debit/Credit benchmark notably ET/1 and TP/1.

The ET/1 is a cut down version of Debit/Credit which allows testing of performance independently of the communications infrastructure (terminals, TP monitors, etc.) and is used widely by database suppliers. The communications infrastructure identification was replaced with a transaction generator package run on an external driver system.

The TP/1 [ANON85], was similar to ET/1 but the main difference was that the transaction generator was run on the hardware tested, rather than an external system. The TP/1 became the database industry standard practice and was the most widely used benchmark for a long time. In recognition of the importance of transaction processing, a council named Transaction Processing Council (TPC) was formed to produce a standard transactions processing benchmark. The council adopted the TP/1 and published it as the database standard practice under the name of TPC-A [TPC 89], and a modified version called the TPC-B [TPC 90]. Despite the publication of the TPC-C [TPC 91], the TPC-A is still the market standard and database industry still relay on the TPC-A figures as the base line for DBMS performance.

2.3 Database Benchmarks

Performance implies two fundamental problems, the first is the logical database structure (i.e. the data model), and the second is the performance parameters of the application.

In the first area, a great deal of research work was applied to database design, access methods, and query optimisation. Wiederhold [WIED83] examined loading, insertion, deletion, reorganisation of file structures, access time and storage requirements. Agrawal [AGRA87], studied database concurrency control performance and implications. The access path optimisers are studied in [ASTR80, MACK86]. Bell, from university of Ulster, conducted several researches in distributed databases and parallel processors [BELL84, BELL92a, BELL92b, McCA92b, McCA92c], he was also involved in several researches to examine buffer management algorithms, query optimisation and resource utilisation [HULL88, McCA92a]. Revell [REVE88], studied the database logical models that were used later on by case tools' users for access path analysis in data modelling.

The performance parameters of the applications have suffered less interest from different research groups, and until now the exact parameters of applications have yet to be clearly defined. The majority of research in this area, however, is usually based on one type of database transaction, and may not be representative of the majority of database applications. That left the majority of DBMS users with an intuitive idea of the power of each database and facing great difficulty in choosing the database that will be best for a particular application environment. Accordingly, the selection of a database system among several varied alternatives requires a rigorous performance measure.

Database benchmarks have become the dominant performance evaluation tool in the database market. The main advantage of the benchmark method is that it checks a real database with an actual operating system and real computers, and there is no need to have recourse to approximations, estimates or simplifications. The benchmarks generally try to establish an objective methodology for the measure of performance that is independent of applications or machine configurations and demonstrate a particular hypothesis: this hypothesis may be the superiority of one database over the other. Several early benchmarking trials took place to develop performance measures of the different database management systems. These performance measures tested hierarchical data models (i.e. IMS systems) [YOUS86a, YOUS86b, YOUS86c], network data models (i.e. CODASYL) [NIED79], and relational data model [HAWT79].

Previous work [REVE90, REVE92a, REVE92b] discussed several database benchmarks placing emphasis on the most widely used benchmarks, the Debit/Credit benchmark (TP1) [ANON85], the Wisconsin benchmark [BITT83, DeWI85] and the TPC activities (TPC-A, TPC-B, TPC-C). The research discussed three aspects for each one of the reviewed benchmarks: database used; transaction set of the benchmark test; and the benchmark performance metrics. In that work the existing benchmarks were analysed and criticised for lack of proper background study. These benchmarks can be classified to three categories.

- Trials in database benchmarks.
- Activities of some research groups.
- Widely used benchmarks.

The first category "Trials in database benchmarks", includes the following benchmarks:

1. A Benchmark Methodology for hierarchical (IBM IMS) Database System [YOUS86a, YOUS86b and YOUS86c].

2. A Benchmark Methodology for CODASYL Database System [NIED79].
3. Cellular Systems Benchmarks [HAWT82].
4. An Interactive Benchmark of the INGRES Database System [MART83].
5. A Benchmark By M. Stonebraker [STON83b].
6. Benchmark Methodology Presented By Benigni and Yao [BENI84, BENI85, YAO 84].
7. The Benchmark Methodology Presented By Strawser, P. [STRA84].
8. Performance Evaluation of a Temporal Database System [AHN 86].
9. A Benchmark Methodology for Simple Database Transactions [RUBE87].
10. The SCAN Benchmark [GRAY87a].
11. The Onkey (IBM 1987).
12. RAMP-C (Requirement Approach for Measuring Performance-COBOL).
13. Performance Evaluation of Main Memory Database Systems [BITT87, TURB88].
14. BYTE Benchmark [GREH88, LANG88, TAZE88 and NADE90].
15. The HyperModel Benchmark [ANDE89].
16. The Set Query Benchmark [GRAY91].
17. The Engineering Database Benchmark [CATT90].

The second category “Activities of some research groups“, includes the following benchmark:

- Research activities of Naval Postgraduate School [BOGD83, STON83a, DEMU84, DEMU85a, DEMU85b, VINC85, FENT86, and KELB87].

The third category “Widely used benchmarks“, includes the following benchmarks:

1. The Wisconsin Benchmarks [BITT83, BORA84, DeWI85].
2. The Debit-Credit Benchmark (TP1) [ANON85].
3. The Transaction processing Council (TPC) Benchmarks, the TPC-A [TPC 89], the TPC-B [TPC 90] and the TPC-C [TPC 91].

2.3.1 Benchmark Methodology for IMS DBMS

Youssef [YOUS86a, YOUS86b, YOUS86c], designed a database and created a software monitor to evaluate the performance of the IBM computers IMS database management system. The methodology used three performance metrics: query response time in a single user mode; query response time in multiprogramming mode; and storage space utilisation.

2.3.1.1 System Configuration and Running Environment

The hardware configuration was an IBM 4341 II (DOS/VSE). The disk system was IBM 3330 series model II disk units with 200 MB formatted capacity. The databases based on two types of DBMS, Hierarchical Direct Access Method (HDAM) and Hierarchical Indexed Direct Access Method (HIDAM). The database sizes varied from 1000 to 20000 tree occurrences (records). The measurements were taken on databases of the sizes 100, 1000, 10000, 20000 tree occurrences.

A job script for each benchmark run generated the benchmark workload. The job script is a file of query numbers. Query streams were generated and stored on disk files for later use by the monitor when it runs. These queries represented the workload on the database. The same query streams subjected both HDAM and HIDAM databases. The job measured the elapsed time for each query independently. The resulted times are either on average of query elapsed times computed as the arithmetic means of the measures taken after repeating the same query N times.

The research addressed subjects such as: database space utilisation; database loading time; database unloads time; impact of first time call; impact of multiprogramming level; effect of physical buffer size; effect of the HDAM DB creation parameters; effect of query complexity; effect of database size; and effect of not found condition.

2.3.1.2 Database Definition

The methodology used an inventory database. The record (tree) composed of six segments distributed over four hierarchical levels. The segments and the definitions were:

1- CSTNMAD	56 BYTES.	Level 0.
2- CUSTLOC	56 BYTES.	Level 1.
3- CSTORDR	49 BYTES.	Level 2.
4- ORDRITEM	38 BYTES.	Level 3.
5- CSTSTTS	18 BYTES.	Level 1.
6- CSTHIST	122 BYTES.	Level 1.

2.3.1.3 Measures of Space Utilisation

1. Storage space required for a tree.
2. Storage space required for all the database after the first time load.

3. Storage utilisation after several insertions or several deletions.
4. Space overheads wasted due to the indexes and the pointers required for the database and any control information, or the space overhead in the form of fragmentation due to the randomising module.

2.3.1.4 Measures for the IMS operators

1. Time required to fetch an arbitrary tree or segment from the database.
2. Time required to get the next tree in the database.
3. Time required to get the segment in the tree.
4. Time required to insert a tree.
5. Time required to update the database by inserting a segment.
6. Time required to delete a tree.
7. Time required to delete a segment from a tree.
8. Time required to update the database by changing an element of a segment.
9. Time required for exhaustive reading of the entire database.
10. Time required for the database loading.
11. Time required for the database unloading.

The study started with 10 query types [YOUS86a] then expanded to 12 query types [YOUS86b]. The final test used 20 query types [YOUS86c]. The benchmark queries were the following:

- | | |
|-----|--|
| Q1 | GN a segment moving from a specific root using segment name only as a partial key. |
| Q2 | GN a segment moving from a specific root using qualified key. |
| Q3 | Insert a segment occurrence. |
| Q4 | GN retrieve the next segment (unqualified retrieval). |
| Q5 | GN a segment moving from a specific root using qualified keys for all the path to the required segment. |
| Q6 | Go to the database start. |
| Q7 | Retrieve a root segment using GU then retrieve any segment in the tree using GNP (qualified with segment key). |
| Q8 | Retrieve a root segment using GU then retrieve any segment in the tree using GNP (qualified with segment name only). |
| Q9 | Retrieve a root segment using GU with a qualified key. |
| Q10 | Retrieve a root segment using GU then retrieve any segment in the tree using GN (qualified with segment name only). |
| Q11 | Retrieve a segment using GHN for subsequent update using REPL. |
| Q12 | Retrieve a root segment using GU then retrieve any segment in the tree using GN (all the path to the segment is qualified by key). |
| Q13 | Retrieve a root segment using GU then retrieve any segment in the tree using GN (the segment is qualified by key). |
| Q14 | Retrieve a segment using GHU for subsequent update using REPL. |

- Q15 Retrieve a root segment using GU then retrieve all the dependent segments in the tree using successive GNP calls.
- Q16 Retrieve a root segment using GU then retrieve all the dependent segments in the tree using successive GN calls.
- Q17 Retrieve a specific segment using GN searching from database start.
- Q18 Go to the database start using GU then retrieve a specific segment using GN qualified by segment name only.
- Q19 Retrieve all the segments of a specific tree through GU to the database start then GN to the required segment. (Multiple GU then GN).
- Q20 Retrieve all the segment occurrences having the same properties in the database.

2.3.2 A Benchmark Methodology for CODASYL DBMS

Niedereichholz [NIED79], from Institut für Wirtschaftsinformatik Universität Frankfurt, presented a test analysis and a performance study of the CODASYL database. This study showed some results of a test of a personnel database. It used a UNIVAC 1108 with the CODASYL-TYPE DATABASE SYSTEM DMS 1100 (LEVEL5).

2.3.3 Cellular Systems Benchmarks

Hawthorn and DeWitt [HAWT82], conducted an experiment to predict the performance of several proposed database management machines. They created benchmark and used a real database. This benchmark used several INGRES queries. The systems analysed include associative disks, RAP, CASSM, DBC, DIRECT and CAFS.

Each of the machines was a cellular system that stored data in cells with a processor per cell (in some cases dynamically assigned). Operations on the cells took place in parallel. The purpose of their evaluation was to determine the effect of the major differences on machine performance. They examined the following differences between the machines: query processing algorithms implemented in the machines; the performance of caching database machine; and the effect of the transfer of the entire records to the host.

The three classes of relational queries overhead-intensive, data-intensive, and the multi-relational queries [HAWT79] were used to compare the performance of each database machine. Q1 is an Overhead-Intensive query. Q2 is a data-intensive multi-relational query (i.e., a join between two relations); and Q3 is a data-intensive query on

a single relation which includes an aggregate operation. The total work and response times were calculated for each query on each machine.

The database for the three queries was the university of California at Berkeley, department of electrical engineering and computer science course and room scheduling database. This database contains 24704 pages of data in 102 relations. The data were information about the university courses.

The relation QTRCOURSE contained 1110 records. Each record had 24 attributes and was 127 bytes long. The attribute "day" was a character field, seven bytes long; "hour" is also a character field, and is 14 bytes long.

The relation "COURSE" contained 11436 records in 2858 pages, and used indexed sequential access method (ISAM) storage structure, keyed on instructor name and course number. "COURSE" required 130 tracks (seven cylinders) of disk space. The relation "ROOMS" contained 282 records in 29 pages, and was hashed on room number. "ROOMS", could be stored on two tracks of one cylinder.

The relation GMASTER contained 194 records, 2 records per page, and resided on a single cylinder. There were 17 unique values for the (acct, fund) pairs along with their associated sums.

2.3.4 An Interactive Benchmark for INGRES DBMS

Martinez [MART83], described an interactive benchmark for the INGRES database system. The benchmark includes single user runs and multi-user runs composed by synthesising different workloads. Basic estimates were obtained for the response type of a range of interactive queries and several other measures of the database management system performance.

The benchmark did not include update operations and did not apply exhaustive workload characterisation that is essential for real system test.

2.3.5 A Benchmark By M. Stonebraker

Stonebraker [STON83b], built a benchmark that examined performance enhancements to a database management system. The benchmark tested the dynamic compiler and the special purpose file system. It consisted of two sections: the first to

test commands that would sequentially scan a relation stored as a heap; the second tried to illustrate performance when a keyed access path could be used.

All measurements were in seconds and obtained from a VAX-11/780 computer running a UNIX operating system. In all cases they indicated elapsed clock time for the command and total CPU time spent on command processing, either by INGRES directly or by UNIX on behalf of INGRES. Moreover, the benchmark obtained the measurements as the only task executing.

In selecting the benchmark and timing technique, they attempted to choose a collection of commands that would illustrate the behaviour of dynamic compilation and a special purpose file system. The benchmark does not contain a large collection of simple commands typically found in on-line environment. In such applications, there can be extra overhead due to concurrency control conflicts or competition for buffer space. No one can draw any conclusions from this benchmark for such environment. In addition, the benchmark could not be chosen to represent ad hoc query and update environment.

2.3.6 Benchmark Methodology By Benigni and Yao

Benigni [BENI84], presented a methodology for database performance evaluation. This methodology was later on enhanced by the same group [BENI85]. Yao [YAO 84, YAO 87] used this methodology to evaluate the performance of different database architecture of several database machines.

2.3.6.1 System Configuration

The database system for the conventional architecture was ORACLE. They installed ORACLE on a VAX 11/750 running VM 3.0. The VAX system contained two M-bytes of main memory. The mass storage available on the VAX 11/750 consists of Digital Equipment RL02 system disk, and a 9766 Control Data Corporation disk drive with a not formatted capacity of 300 M-bytes.

The IDM-500 database machine represented the database machine architecture in the benchmark study. The benchmark test installed and used an IDM-500 database machine with one M-bytes memory. The IDM used release 24 software. Mass storage for the IDM was 9766 Control Data Corporation disk drive. The IDM operated by a

VAX 11/750 front end computer running Berkeley UNIX 4.1. The data transfer between the VAX and the IDM-500 was through a 9600 baud RS232 communication line.

2.3.6.2 Test Data

For benchmark testing they used a file of personnel data from a large application system. From these data, they extracted a personnel database schema. The schema modelled an entity-relation diagram. The benchmark studied three different levels of indexing. Level 0 contained no indexes on any of the database attributes. Level 1 provided primary indexes on the database. Unique clustered indexes were built on all primary keys. They also tested the database system ability to provide combined indexes. In level 1 they included three combined indexes. Level 2 included indexes from level 1 and added additional secondary indexed attributes that were for retrieval in benchmark query set. Because of storage constraints, they could apply further indexing tests.

Several databases of different sizes were constructed by eliminating a percentage of records from the database. The original personnel file data formed a single, un-normalized relation of size approximately 56 M-bytes and contained 189960 records.

By randomly eliminating records, they were able to form several database sizes for their benchmark experiments. A program to divide the data in normalised relations in the conceptual schema was run to build the test database.

2.3.6.3 Benchmark Workload

They designed a number of queries to test the retrieval and update capabilities of the database systems. The queries were written in the SQL query language for the conventional database system, and in QUEL for the database machine system. They divided queries into several categories. For data retrieval they developed ten query sets and several special query sets to test specific database system features. Each query set contained from four to seven queries that vary in complexity based on the number and type of conditions in the query predicate. They used the complexity classification developed by Cardenas [CARD75] in their benchmark methodology.

Each query set was designed so that the complexities of the query predicates increased with increased numbers in the set. The number of records retrieved changed from one query to the next depending on whether an OR condition (increase in records) or an AND condition (decrease in records) was added.

The query sets were designed as follows: Query set 1-5 tested single relation retrieval. The query sets ranged from retrieval on small size relation (Query set 1), medium size relation (Query sets 2 and 3), and large size relations (Query sets 4 and 5). Query sets 6-10 tested multiple relation retrieval. Query set 6-8 were two relation queries. Query set 9 was a three relation query and query set 10 was a four relation query. The basic query sets were numbered for reference as qx-y, where x is the query set number and y is the query number in the set.

To test the sorting facility they modified Query set 1 through 4 by adding an "ORDER BY" clause on an appropriate attribute. To test aggregates they modified query sets 4 and 5 by adding the "COUNT" aggregate function in the output list.

They also tested the performance of several update commands. They designed representative insertions, deletions, and modifications.

Benchmark workloads were generated by defining job script for each benchmark run. The job script was a file of query numbers. For example, a job script for a benchmark run could be (q1-1,qx5-3,q9-4,qi-3,q4-5). The job script file was read into a program called "Runner". This program was located in the host computer for the database machine and the micro computer for the conventional database system. "Runner" executed the queries in job script order on the database system. Statistics were recorded on the queries' performance in the system. Times were recorded to the sixtieth of the second to three decimal places for the conventional and database machine benchmarks. The statistics gathered were the following:

First	: Time until first record is received at the host.
Last	: Time until last record is received at the host.
Records	: Number of records returned.

They recorded parse time of queries in the host system for the database machine architecture.

For each benchmark test, they defined a job script and executed the "Runner" on the different database systems. Statistics for each query in the script were collected. For

multi-user tests and background load tests on the database, they ran multiple copies of "Runner" simultaneously on separate job scripts and gathered the statistics on each.

2.3.7 Benchmark Methodology By P. Strawser

Strawser proposed a methodology to create a batch benchmark job structure in here PhD dissertation [STRA84]. The objective of the benchmark was to establish optimum performance characteristics for the machine. There were two batch job models, one for single-tuple queries and a second for multiple-tuples queries. A job for single-tuple queries executed a collection of single tuple queries against a single relation. A job for multiple-tuple queries executed multiple queries against multiple relations at a single-tuple size. The batch consists of 25 query types. The output is directed to spool file. The batch output accumulated statistics to a statistic file, and written queries to a query file for constructing query pools.

Because this benchmark was complex and required long time to run, it has never been used since it has been published.

2.3.8 Performance Evaluation of Temporal DBMS

A prototype of a temporal management system was built by extending INGRES [AHN 86]. It supports the temporal query language TQuel, a super-set of QUEL, handling four types of databases : Static, Rollback, Historical and Temporal. A benchmark set of queries was run to study the performance of the prototype on the four types of databases.

They tested the databases of the four types. For each one of the four types, they created two databases, one with 100% loading factor and the other with 50% loading factor [WOOD81]. Each database contains two relations, type-h and type-i, where Type is one of Static, Rollback, Historical and Temporal.

Each tuple has 108 bytes of data in four attributes, Id, Amount, Seq and String. Id, a four integer, is the key in both relations. Amount and String are randomly generated as integers and strings respectively, and Seq is initialised to zero. In addition, rollback and historical relations carry two time attributes, while temporal relations contain four time attributes. Attributes' transaction Start and Valid From were randomly

initialised to values between, Jan. 1 and Feb. 15 1980, while attributes' transaction Stop and Valid To were set to "forever" indicating that they were the current versions.

Each relation was initialised to have 1024 tuples using a copy statement. The page size in the prototype is 1024 bytes. With 100% loading, there are 9 tuples per page in static relations, and 8 tuples per page in Rollback, Historical or Temporal relations. Therefore, the database consisted of 114 pages for each Static relation, and 128 pages for each of the others. The actual size depended on: the database type; the access method; the loading factor; and the average updates count.

The average number of disk access was used as the benchmark metric. The benchmark focused solely on the number of disk access per query at a granularity of a page.

The number of disk access can not be taken as a performance measure as varies greatly on many factors irrelevant to the tested DBMS. For example the buffer management algorithms or the number of disk buffers could be crucial factors in determining the disk speed.

2.3.9 Benchmark Methodology Using Simple Queries

Rubenstein [RUBE87], proposed a benchmark measurement to measure response time specifically designed for the simple, object-oriented queries that engineering applications perform. They reported results from running this benchmark against some database systems they use themselves. They discussed number of factors such as: caching the entire database in main memory; avoiding query optimisation overhead; using physical links for pre-Join; and using an alternative to the generally accepted database "Server" architecture on distributed networks. They were interested in database response time as the time elapsed from the issue of a database query until the results were returned. These queries were issued by a program, and the programs must issue many simple queries to update a window or graphical display.

The benchmark database consists of three record types:

1. A PERSON record type with three fields: Person ID, Number; Name; and Birth date. The ID is a 4 byte integer and is the key for the record (e.g. each ID is unique). The Name may contain up to 40 bytes, and the Birth-date was a 4 byte integer. There were 20,000 person record in the database, with

randomly distributed names and birth-dates. The program regenerated ID fields as ascending integers (From 1 to 20,000).

2. A DOCUMENT record type consisted of six fields: a document ID; Title; Page count; Document type; Publication date; Publisher; and Description. The document ID is a 4 byte integer; it is the key for this record type. The Title, publication, and Description fields are strings containing up to 80 bytes each. The Page count, document type, and Publication date are integers. There are 5000 documents in the database with randomly generated attributes. The attributes were the following: Titles; Page counts; Types; Publication dates; Publishers; and Descriptions. The program generated ID fields as ascending integers (from 1-5000).
3. AUTHOR record type, with two fields: a person ID, which references the Key of the person table; and document ID, which references the Key of the document table. This table therefore, connected each person to zero or more documents, and each document to zero or more persons. There are 15,000 author records in the database. Each document record was associated with three randomly selected person records. The author records were not clustered or co-located with either document or the person records that they reference. The database comprised approximately 3 megabytes of data (ignoring all overhead introduced by the data manager). There was another larger database called "Large"; it was identical to the smaller one except that all of the record counts were scaled up by a factor of 10. This database required approximately 30 megabytes of storage plus overhead.

The following benchmark script was proposed for response time measurements for simple operations:

1. Name Lookup: This is the simplest database operation, to look for a record with a particular key value.
2. Range Lookup: Finding the records with a particular range of values in a particular set of fields.
3. Group Lookup: To find all of the records in one table that pertain to a particular logical entity in another table.
4. Reference Lookup: Finding a record that is referenced by a field of a particular record.
5. Record Insert: Insert a new record.
6. Sequential Scan:

7. Database Open: The time to initialise the database.

As presented in the paper, the authors stated that, the performance measurements for databases on this benchmark alone do not make a system acceptable and they recommended the use of any other benchmark for further complicated operations.

2.3.10 The SCAN Benchmark

Gray [GRAY87a], described a benchmark that consisted of two batch operations that perform long search against a small database. The benchmark was as follows:

- * SCAN - A mini-batch operation to sequentially copy 1000 records.
- * SORT - A batch operation to sort one million records.

Elapsed time for database SCAN and SORT were the performance metrics for this benchmark. A system could be analysed as follows:

SCAN: Begin Transaction
Perform 1000 Times
Read Sequential
Insert Sequential
Commit Transaction

The atomic weights for BEGIN, READ SEQUENTIAL, INSERT SEQUENTIAL, and COMMIT were measured for each release. The atomic weight usually consisted of CPU instructions, message bytes, and disk IOs for a "typical" call to that operation. These weights could be converted to service times by knowing the speeds and utilisation of the devices (processors, disk, lines) used for the application. The molecular weight and service time SCAN can then be computed as the sum of the atomic weights.

2.3.11 The Onkey (IBM 1987)

It is a transaction processing benchmark. It simulates a point of sale credit card environment. It measures performance using three different transaction types. It differs in two technical respects from Dept-Credit technique:

- it does not require an X25 presentation services;
- it does contain requirements for auditing and recovery.

2.3.12 The RAMP-C

The word RAMP-C stands for "Requirement Approach for Measuring Performance-COBOL". It is a transaction processing benchmark. Its specifications have not yet been published by IBM. There is another RAMP-C (ICL, UNISYS) that uses a mix of four different transaction types with among 70 and 625 COBOL statements per transaction, and indicates performance when the system is loaded to 70% of capacity.

2.3.13 Performance Evaluation of Main Memory DBMS

In [BITT87] they presented the results of a benchmark of the relational Main Memory Database System (MMDBS). They determined relevant performance metrics and described techniques for benchmarking MMDBS's.

The benchmark database was the synthetic database used in the Wisconsin benchmark [BITT83].

They used the space-time product metric. The space time product for a query was defined to be the average working set size during the execution of the query multiplied by the time required for the query [BUZE76, DENN80].

To measure the performance of the MMDBS functions, they wrote a benchmarking program that took the names of the query windows from an input file and obtained measures of interest before and after each query was executed. The program executed as many queries as were in the input file by invoking the MMDBS query processor.

They used a simplified method to obtain the space requirements that resulted in an approximation to the space time value. Without sophisticated tracing hardware and software tools, the space-time product for a query cannot be computed precisely [ALAN80, ALAN84, HEID84].

They studied several factors such as real and virtual memory requirements, paging, CPU speed, data structures and algorithms in isolation, but it is known that there are complex interactions between all these factors.

Finally, the experiments were conducted in single user, standalone mode. Single user means that there was only one user of the DBMS active when they obtained their measures. Standalone means that there were no other users on the machine when the performed their experiment.

2.3.14 BYTE Benchmark

The Byte benchmark was introduced for the first time in [GREH88, LANG88, TAZE88]. A new version of the benchmark was published in [NADE90]. The benchmark included a set of low level tests and a set of application level tests. The benchmark based on the Small-C compiler. The Small-C attraction is its ability to compile itself. The benchmark is divided to several procedures each procedure tests a different area of system performance. This research is mainly interested in the DBMS procedure.

The benchmark is limited in use to PC level, and difficult to be applied in higher computer levels. Cohen [COHE90], argues that the benchmark is not suitable for computers with embedded microprocessors in automotive applications. Also others [WILL90], complain that the Byte shell script should yield portability. However, there are pitfalls in the form of unknown but systematic errors in user time to search the PATH for the script executables.

2.3.15 The Set Query Benchmark

The Set Query Benchmark is designed to measure the performance of strategic data access (SDA). The words SDA, DSS, management reporting and direct marketing systems are synonyms.

Database functions that support SDA applications generally use 'set queries', which are queries that take into account data from numerous table rows at once in each question. The Set Query benchmark has been created to aid decision makers who require performance data relevant to strategic data applications.

The Benchmark database consists of one table named BENCH. The BENCH table has 13 indexed columns; these columns are KSEQ that is a clustered primary key. The other columns are K500K, K250K, K100K, K40K, K10K, K100, K25, K10, K5, K4, and K2. Each column has an integer value ranging from 1 to its cardinality and column name indicates column cardinality.

The benchmark is not recognised by the database market up to now and many database users still to be convinced to use it.

2.3.16 The HyperModel Benchmark

Anderson and others at Tektronix [ANDE89], developed the HyperModel benchmark. The HyperModel benchmark is based on hyper text model consisting of a graph containing nodes and arcs, or entities and relationships. There are four entities and four relationships in HyperModel, as opposed to one entity and one relationship in the sun benchmark. HyperModel has twenty-five operations; the Sun benchmark ended up with three. HyperModel includes part or subpart relationships commonly found in ODBMS applications, as well as "Blobs", (binary large objects, which are stored in databases and managed as un-interpreted objects and called blobs). The operations measured in the HyperModel Benchmark are divided into:

1. Name lookup operations, which return an object based on an attribute ID or an object ID.
2. Range lookup operations, which return all objects whose attribute fall under a specified range of values.
3. Retrieval operations, which retrieve objects by traversing the relationships between them.
4. Reverse retrieval operations, which traverse relationships in the reverse order.
5. Sequential Scan operations, which retrieve all objects in the database.
6. Closure traversal operations, which start with a randomly selected object and perform an operation on that object, requiring the operation to be performed recursively on all objects reachable from that objects to n level.
7. Editing operations, which update objects already in the database and commit the updates.

The HyperModel Benchmark measures a larger set of operations, with more interrelationship among objects than the Sun benchmark. It measures both cold start and warm start results, as does the Sun benchmark.

The benchmark does not specifically measure local versus remote database access the way the Sun benchmark does. There are several important aspects of ODBMSs that the HyperModel does not measure, the authors of this benchmark recognise this and point out in their work that raw performance is only part of an ODBMS evaluation.

2.3.17 The Engineering Database Benchmark

Cattle, Skeen and others [CATT90], developed the Engineering Database Benchmark (also called 001 for Object Operations Version 1). They created the benchmark in an attempt to highlight some performance problems that RDBMS have when processing engineering information. They wanted the benchmark to be used in determining whether ODBMSs can provide the necessary performance. The benchmark set out to prove or disprove two hypotheses: an ODBMS can achieve better raw performance than RDBMS for engineering operations by a factor of 10 to 100; and certain characteristics common to ODBMS architecture make difference to systems performance, in particular, ODBMS ability to cache large working sets in memory and efficiently access remote data.

The authors emphasise that engineering applications use a programming language interspersed with operations on persistent data, and they have designed the benchmark accordingly. They also used characteristics of real engineering applications as their base for the ratios of reads to write and traversals to look-up. The benchmark measures performance for several operations:

1. Create a database consisting of 20,000 parts and 60,000 interconnections.
2. Select 1000 part IDs at random, retrieve the parts with those IDs, and perform an operation on each.
3. Select a part at random, traverse all connections from it to other parts in a depth-first manner to seven levels, and perform an operation on each.
4. Execute the same traversal in reverse order, starting with a randomly selected part and retrieving all the objects connected to it, and so forth.
5. Enter 100 new parts and three connections from each to other randomly selected parts, then commit changes to disk.

6. Perform all operations with the application and database located on the same machine, and again with them located on two different machines.
7. Record all results for both a "cold start" (the database exists only on disk and nothing is cached in memory) and a "warm start" (after ten iterations of each operation, so that many objects are cached).

The main comment on the benchmark is it includes session time as part of the transaction response time. Session time represents over 99.99% of any transaction response time as it implies human computer interaction that, in comparison to computer speed, is very slow. Session time is included as part of transaction response time in very specific to special cases such as real conversational mode. In real life, most on-line systems are designed as pseudo-conversational systems to save computer resources.

2.4 Activities of Naval Postgraduate School

Several research groups in Naval Postgraduate School, Monterey, California, supervised by Steven A. Demurjian, and David K. Hsiao conducted several studies on the performance of the multiple-backend database systems (MBDS). The following sections will be briefly review those activities and discuss the performance techniques implemented in each one of that research.

Bogdanowicz [BOGD83], presented an experiment in benchmarking a database machine. The purpose of the work is to present an approach to benchmarking database machine using a synthetic database. The work describes a database generation tool that allows the user to build a synthetic database through an interactive interface. The description of the testing is quit general. The system configuration of the database machine was not fully described. The testing in this research is limited to the single user case.

Stone [STON83a], focused on measurements of the response time. A development of a system to measure components of the response time are discussed. The system involves generation of synthetic database. The system also measures the benchmarked machine in using that database. The data is generated through a Relational Generator (RG). The RG is a parameterised program for generating relations for a database. First the user is instructed to enter the relation name and size (i.e., number of tuples). Then, the program requests data about each attribute. All the relations are characterised by the same general template. Four specific templates are derived from the general one. These templates correspond to the four tuple lengths used for testing (i.e.,

100 bytes, 200 bytes, 1000 bytes, 2000 bytes). Each template is used to generate the relations of sizes (500-10000 tuples). The benchmark consisted of calls involving only one relation (i.e., selection and projection) and experiments involving more than one database (i.e. joins).

The response time measurement was taken from the backend machine clock. This clock has a resolution of 1/60 of the second and an accuracy within 1/50th of a second. The query scripts were used to implement the selected designed benchmarks.

Curtis [RYDE83], described the functions of the DBAs and how they are supported by the benchmarked relational database machine. The host machine for the benchmark is Univac 110/42. The hardware interface between the host and database machine is through a Univac 1100/42 I/O channel. This interface channel has a 200-thousand bytes/Sec capacity and the transmission unit is either a byte or a word. The database machine that interfaces with the host is a Burton Lee IDM 500. Several query streams were formed and submitted against four databases.

Demurjian [DEMU84, DEMU85a, DEMU85b], aimed to devise benchmarking strategies for and applying methodologies to the measurement of a prototype database system in multiple backend configurations, and to verify the performance claims as projected or predicted by the designer and implementor of the multi-backend database system known as MBDS. By collecting macroscopic data such as the response time of the request, the external performance measurements of MBDS have been conducted.

Fenton [FENT86], designed a computer aided tool for the generation of test transactions and test database for the benchmarking of parallel, multiple-backend systems used in that research.

2.4.1 The MBDS Hardware Configuration

The hardware configuration of MBDS consists of VAX-11/780 (VMS OS) running as the controller and two PDP-11/44s (RSX_11M OS) and their disk systems running as backends. The disk system on each backend is a DEC RM02 disk drive, which has a 67 MB formatted capacity, a peak transfer rate of 806 KB/s and an average access time of 42.5 ms (30 ms average seeks time + 12.5 ms average latency time). Inter-computer communication is supported by three parallel communication links (PCL-11Bs), which is a time division multiplexed bus.

2.4.2 The Attribute Based Model

In the attribute based model, data is modelled with: the construct database; file; record; attribute value pair; directory key word; record body; key word predicates; and query. A Database consists of a collection of files. Each File contains a group of records that are characterised by a unique set of directory key word. A Record is composed of two parts. The first part is a collection of attribute- value pairs or key-words. An attribute-value pair is a member of the Cartesian product of the attribute name and the value domain of the attribute. As an example: <POPULATION,25000> is an attribute-value pair having 25000 as the value for the population attribute. A record contains at most one attribute-value pair for each attribute defined in the database. Certain attribute-value pairs of a record (or a file) are called the directory-keywords of the record (file). That is because either the attribute-value pairs or their attribute-value ranges are kept in a directory for addressing the record (file). Those attribute-value pairs that are not kept in the directory for addressing the record (file) are called non-directory key word. The rest of the record is textual information that is referred to as the record body.

2.4.3 The Benchmark Strategy

The benchmark strategy focuses on collecting macroscopic measurements on the system performance. Macroscopic measurements correspond to the external performance measurements of the system, which collects the response time of requests that are processed by the system. The test database was constructed using a record size of 200 bytes. A total 24 clusters are defined for the test database. The database size was set to a maximum of 1000 records per backend. At the beginning, they specified three different system configurations for the MBDS performance measurements, the databases were eventually extended [DEMU85b], to five system configurations.

They created a set of retrieve requests to benchmark MBDS. The retrievals are a mix of single and double predicates. There are two directory attributes and 31 non-directory attributes in each record. The directory attributes INTE1 and INTE2 are integer valued, and are used for the cluster definition and formation. INTE1 is defined using 5 attribute- value ranges, while INTE2 is defined using 24 attribute-value ranges. The non-directory attributes are used as fillers to the 200-byte record.

Target: INTE1, INTE2, MULTI, STR00 ---> STR29. These requests, will cause several numbers of clusters to be examined and part of the database to be retrieved.

Vincent [VINC85], presented a methodology for creating test database sets and test-transaction mixes for the MBDS. He named this methodology CAD. It is driven by three elements of information solicited from the user. These elements are: number of backends in the system to be tested; amount of disk storage per backend; and size of the data transfer from the secondary storage (disk) to the primary storage (memory). The database design factors are: system configurations; database size considerations; and test transaction mixes considerations.

2.4.4 System Configurations Considerations

For a given test database, two sets of configurations must be generated, a set for measurement of the response time reduction, and a set for the measurement of response-time variance. The number of configurations within each set is determined by the number of backends of the system to be tested. Depending on the configuration being used, the database must be evenly distributed to 1, 2, 3 or M backends.

2.4.5 Database Size Considerations

Three different database sizes are preferred, all of which are multiple of the base (original) size N . One size presents a small database ($N/4$), another size presents an intermediate size database ($N/2$), and the final size is the database size multiple (DBM) N . The CAD implements the scheme described [STRA84] by having the four record sizes as fixed multiples of one another. The final criteria for how large the database size may be is the available disk storage of the type of backend to be used in the system.

2.5 The Most Widely Used Benchmarks

Some benchmarks have established themselves in the database market. These benchmarks gained acceptance and many database systems have been measured using them. The benchmarks are the following:

1. The Wisconsin Benchmarks [BITT83, BORA84, DeWI85].
2. The Debit-Credit Benchmark (TP1)[ANON85].
3. The Transaction processing Council (TPC) Benchmarks [TPC 89, TPC 90, TPC 91].

In the following sections examine in depth and discuss these benchmarks, in addition, they present the comments concerning the inadequacies of these benchmarks.

2.5.1 The Wisconsin Benchmarks

Bitton [BITT83], presented the well-known "Wisconsin Benchmark", which was subjected to further changes by Boral and DeWitt [BORA84]. DeWitt [DeWI85] concluded the final form of this benchmark and used it as a standard methodology for evaluating the performance of database management systems. The benchmark was used to evaluate the performance of NON-VON's computers [HILL86] and database machines [BORA84] in single and multi-user environments.

2.5.1.1 Description of the Test Database

The database used for the experiments is based on synthetic database described in [BITT83]. Two basic relations were used to generate the database, "oneKtup", and "tenKtup" as they contain respectively, one and ten thousand tuples. Each tuple is 182 bytes long and consists of number of integer and string attributes. The first attribute, "unique1" assumes unique values throughout the relation (and hence constitute a key). For the "thoustup" relation, "unique1" assumes the values 0,1,...,999. For the "tenthoustup" relation the values of "unique1" are 0,1,...,9999. The second attribute, "unique2", has the same range of values as "unique1" but a random number generator was used to scramble the values of "unique1" and "unique2" when the relations were generated. The remaining integer attributes are names after the range of values each attribute assumes. That is, the "two", "ten", "twenty", "hundred", ..., "tenthous" attributes assume respectively values uniformly distributed over the range (0,1), (0,1,...,9), (0,1,...,19), (0,1,...,99), ..., (0,1, ...,9999). Finally, each tuple contains three 52 bytes string attributes.

Each relation was sorted on its "unique2" attribute (thus leaving the relation unsorted on its "unique1" attribute). For each of the tenKtup relations, a clustered index was constructed on the "unique2" attribute and a non-clustered index was constructed on the unique1 attribute. No indices were constructed on the oneKtup relations.

2.5.1.2 Description of the Benchmark Queries

The Resource Utilisation Approach to Query Mix Selection. The CPU and disk resources consumptions were classified into two classifications: "Low" or "High".

Type I : Low CPU utilisation, Low disk utilisation.

Type II : Low CPU utilisation, High disk utilisation.

Type III : High CPU utilisation, Low disk utilisation.

Type IV : High CPU utilisation, High disk utilisation.

The Benchmark Queries are:

Type I : Select 1 tuple from 10,000 using a clustered index.

Type II : Select 100 tuples from 10,000 using non-clustered index.

Type III : Join 10,000 tuples with 1000 tuples using a clustered index on
Join attribute of 10,000 tuples relation.

Type IV : Aggregate Function on 10,000 tuples relation (100 partitions).

2.5.1.3 Performance Metric

They used the system throughput measured in queries-per-second as their principal performance metric. Where illustrative, response time has also been used as performance indicator.

2.5.2 Comments on The Wisconsin Benchmark

Two of the most active members in the area of database benchmarks, P. Hawthorn and D. DeWitt, conducted two benchmark experiments. P. Hawthorn was the base for the Wisconsin benchmark and D. DeWitt is one of the three members of the group that originally designed and implemented the Wisconsin benchmark. In both cases they either had to change the original benchmark script or used a different benchmark. The two examples are the following:

1. Variation on the Wisconsin Benchmarks by P. Hawthorn.

As P. Hawthorn, from the Britton Lee Inc., was not happy with the results obtained by DeWitt and Boral in [BORA84] she presented an augmentation to the Wisconsin benchmark [HAWT85]. She concluded that this benchmark does not include "amount of data returned" as a factor in the performance of a DBMS. She changed the benchmark as follows.

- Created a new relation substituting character strings for each of the integer strings.
- Changed Query3 to look as follows:

qry3cp() /* Join Using Clustered index on join */

```
/* attribute "unique2D" */  
/* The query produces 1000 tuples */  
/* Further qualification to produce one Tuple */  
Char a[20];  
range of t is tenKtup  
range of w is oneKtup  
retrieve (a = w.unique1A)  
where t.unique2D = w.unique1A  
and w.thousandD = "10000"
```

- As a final test, she changed the database itself by creating a relation made up of 10,000 tuple, each having 2 uncompressed 20-character attributes. Then they ran the following query:

```
range of p is packed  
retrieve (cnt = count(p.thousandD  
where p.thousandD = "10000"))
```

2. Gamma Database Machine Benchmarks by D. DeWitt.

DeWitt [DeWI88] presented the results of a single-user performance evaluation of the Gamma and Teradata database machines. The interesting point is he did not use the Wisconsin benchmark. He changed the data size and created a new set of queries.

DeWitt constructed 100,000, and 1,000,000 tuple database based on the Wisconsin synthetic databases' 1,000 and 10,000 tuple relations.

The benchmarks were divided to three parts, six Selection Queries, three Join Queries, and Update Queries (append, delete and modify).

Several researchers in the database performance field criticised the Wisconsin benchmark for one problem or another.

Hawthorn [HAWT85] criticised the data format and transaction qualification of Transaction three as she thinks they are not realistic enough to measure any database performance.

Stonebraker [STON85] criticised the Wisconsin benchmark for not having floating point operations, copy operation, and schema modification. He criticised the

benchmark transaction mix for being appropriate only for Decision Support System (DSS) environments that represent very small percentage of the running transactions.

Serlin [SERL86], criticised the Wisconsin benchmark because it does not map its results into overall user environment.

Bitton [BITT87], one of the Wisconsin benchmark group, criticised the benchmark as it has a default result size of 1000 rows that is too large for testing interactive transactions, and made it an inappropriate default. She changed that to a result of 10 rows and 5 attributes.

Finally, Turbyfill [TURB88] criticised the default result size of the Wisconsin benchmark for being too large. It had too many attributes and the cost of formatting and outputting the default result overshadowed other effects.

Generally, the Wisconsin benchmark results provide a number of points of comparison, what they do not provide is any mapping of these points into the overall user environment.

These problems can be seen to arise from two main causes:

1. Benchmark Design

The authors of the benchmark simplified their initial experiments ignoring factors that are important. For instance, they ignored the following factors:

- a. They used only two data types, fixed length strings, and two byte integers.
- b. All attributes were uniformly distributed.
- c. The initial tests were all performed in single user mode.
- d. The benchmark script is appropriate only for decision support systems.

2. The Database

The Wisconsin benchmark compares two systems are using 52 byte string and 2 byte integers only. While some comparisons are independent of data type, others clearly are not. For instance, one system had special hardware that was supposed to speed up certain operations such as comparisons of character strings. The results concerning the improvements obtained by that hardware were valid only for the two data types tried, and did not reflect in any way improvements that may be gained for other types such as variable

length strings and packed decimal numbers. Generally, the databases used for the benchmark can be criticised for the following:

- a. Quite a few database columns, such as Odd100, have never been used.
- b. All values of the attributes were uniformly distributed, all the data in real world is not uniformly distributed, and at least a few non-uniformly distributed attributes should have been included.
- c. The strings have been criticised for being too long, and for having only one significant character in the beginning of the string, the next significant character occurring 25 bytes later. According to expert opinion [GRAY87b], fixed length strings of 20 or 30 characters are more realistic. Furthermore, most strings can be differentiated by the first few characters in the string.
- d. Data types more representative of those used in the benchmark should have been used instead, such as 4 byte integers and 20 byte strings.
- e. The row length should be an easier number for calculation, such as a multiple of ten.
- f. Finally, the database is too small for the DBMS that is being tested. The database needs to be scalable.

The role of the Wisconsin benchmark has declined over the years and lost its importance after the domination of the Debit-Credit (TP1) benchmark. The benchmark is rarely used now but it is mentioned in this context because it is the first trial to produce a standard measure of DBMS performance.

2.5.3 The Debit-Credit (TP1).

The most frequently quoted measure of database performance in the marketplace has become TP1/sec [ANON85]. Many computer vendors have used this benchmark to evaluate a new product [TAND88, ORAC91]. This benchmark that measures the database transaction capability has evolved from a wider benchmark called Debit-Credit. The names Debit-Credit and TP1 are both used to describe the basic benchmark.

The benchmark modelled a banking environment and simulates the random withdrawals being made against bank accounts at a large bank. The banking model defined by Debit-Credit consists of a bank with many branches each with 10 tellers and 10000 accounts.

2.5.3.1 Description of the Test Database

Briefly, the database consists of four SQL tables:

ACCOUNT : a table of bank accounts. Each record is 100 bytes long and holds the account number and balance.

TELLER : a table describing bank tellers. Each record is 100 bytes long and holds the teller number and cash position.

BRANCH : a table describing the cash positions of each bank branch. Each record is 100 bytes long and holds the branch number, and the cash position of all tellers at that branch.

HISTORY : an entry-sequence table containing records of all transactions. Each record is 50 bytes long and holds the account, teller, branch, delta, and time stamp of the transaction.

In database terms the tellers are considered to be the database users. The tellers use block mode terminals (like IBM 3270 terminals) which are configured to have 10 input and output fields. The input message is 100 bytes and the output message is 200 bytes. Each teller requests an account update every 100 second on average. Thus, 100 terminals, each operating at this rate would average one transaction per second on the system under test. Systems that run more than one transaction per second have the database and network scaled linearly. For example, a 100 TPs system has a database and network 100 times larger.

2.5.3.2 Description of the Benchmark

The transaction coded in SQL is as follows:

```
READ 100 BYTES FROM TERMINAL;
PERFORM TRANSACTION SERVICES GIVING
EXEC SQL BEGIN;
EXEC SQL UPDATE ACCOUNT SET balance = balance + :delta
WHERE account_number = :account;
EXEC SQL UPDATE TELLER
SET balance = balance + :delta /
WHERE teller_number = :teller;
EXEC SQL UPDATE BRANCHT SET balance = balance + :delta
WHERE branch_number = :branch;
EXEC SQL INSERT INTO HISTORY VALUES
(:timestamp,:account,:teller,:branch,:delta);
EXEC SQL COMMIT WORK
PERFORM PRESENTATION SERVICES;
```

WRITE 200 BYTES TO TERMINAL;

A system that can run one such transaction per second, given less than one-second response time to 95% of the transactions, is defined to be a one transaction per second (TPs) system. The database of a 1-TPs system is defined as:

10,000	Accounts.
100	Tellers.
10	Branches.
2,590,000	History Records.

The HISTORY table is sized to accommodate 90 days of history records assuming the average throughput is one third of the peak throughput.

Messages are transmitted through the X.25 protocol. It is part of the Debit-Credit specification that X.25 communications are used between the terminals and the computer. If the database is distributed, then 15% of the transactions arrive at branches other than the account's home branch. These 15% are uniformly distributed among the other branches.

The benchmark requires that the transactions be run with UNDO and REDO transaction protection (abort, auto-restart, and roll forward recovery). In addition, it specifies that the transaction log must be duplexed. Each withdrawal transaction is regarded as an atomic action. That means that, however complex the transaction, if the user has been informed that it has been completed then it will be remembered by the DBMS even if the system crashes immediately after the completion message. The withdrawal transaction will force the system to update information regarding the amount of money the branch teller and account each has. If any of these updates is incomplete at the point at which the system fails, then it must be possible to recover the database to a consistent state.

2.5.4 The Transaction Processing Performance Council (TPC) Activities.

This section presents the description of the TPC benchmarks A, B and C. Chapter three discuss in detail the technical limitations of those benchmarks.

Due to the lack of standards and in recognition of the importance of the database performance issue, the database industry formed the Transaction Processing

Performance Council (TPC). The TPC consists of major RDBMS vendors and users, and hardware vendors. It has taken on the task of defining benchmarks, primarily for RDBMSs with the goal of making the implementations consistent across vendors. TPC has put specifications for RDBMS benchmarks [TPC 89]. These specifications include the following points:

1. System properties that must be in effect during the transaction execution. These are called ACID properties: Atomicity; Consistency; Isolation; and Durability. The DBMS must ensure atomicity during the benchmark so no partial transactions are permitted to modify the database. The DBMS must support consistency during the benchmark by ensuring that each transaction takes the database from one consistent state to another. The database must ensure isolation, also known as serialisability, of all transactions during the benchmark so the results of concurrently executing transactions are the same as the results that would have been achieved by some serial execution of the transactions. Finally, the DBMS must ensure durability by preserving the effects of all committed transactions even in the face of system and media failure.
2. A detailed description of transactions to be run.
3. Table layouts, number of records for each table, and minimum number of rows of each table that must be accessed per test.
4. Rules for distributing and partitioning data among tables, for timing transactions, and for configuring hardware for the benchmark.
5. Requirements and recommendations for reporting benchmark activity and results so the benchmark can be repeated. This information is used by independent agencies who audit vendors' benchmark implementations to be sure there are no violations or misinterpretations of benchmark requirements.

2.5.5 The TPC-A

The TPC-A benchmark was launched in November 1989, the benchmark was based on the famous TP1. It measures performance using update intensive database transactions.

This benchmark as described in previous section uses a single, simple, update-intensive transaction to test database systems. The workload does not reflect the entire

range of the On-Line Transaction Processing (OLTP) requirements typically characterised by multiple transaction types of varying complexities.

Similar to the TP1 benchmark, the TPC-A benchmark measures the number of transactions per second a system can perform when driven from multiple terminals, TPC-A does not specify the number of terminals. The TPC-A can be run in a wide area or local area network configuration, with the performance described by the two metrics "TPC-A local throughput" and "TPC-A wide throughput", measured in transactions per second. The two metrics are different and can not be compared.

2.5.6 TPC-B

The TPC benchmark B (TPC-B) was launched in August 1990 [TPC 90]. The TPC-B is not an OLTP benchmark and is mainly based on a batch transaction that does not require any terminal networking, or think time. This transaction is characterised by:

- Significant disk input/output.
- Moderate systems and application execution time.
- Transaction integrity.

Similar to both TP1 and TPC-A, the TPC-B measures systems performance with how many transactions per second a system can perform per second (tps), subject to residence time constraint; and the associated price-per-tps. The metric for this benchmark is "tpsB". TPC-B results can not be compared to TPC-A. The TPC-B benchmark database is the same database used for the TPC-A.

TPC-B transaction is a simple, update intensive transaction. Similar to TPC-A, the workload of this benchmark does not reflect the entire range of OLTP requirements.

* Transaction Profile.

Begin Transaction

```
Update Account where Account_ID = Aid:
    Read Account_Balance from Account
    Set Account_Balance = Account_Balance + Delta
    Write Account_Balance to Account
Write to History:
    Aid, Tid, Bid, Delta, Time_stamp
Update Teller where Teller_ID = Tid:
    Set Teller_Balance = Teller_Balance + Delta
```

```
Write Teller_Balance to Teller
Update Branch where Branch_ID = Bid:
    Set Branch_Balance = Branch_Balance + Delta
Write Branch_Balance to Branch
COMMIT TRANSACTION
Return Account_Balance to driver
```

Aid (Account_ID), Tid (Teller_ID) and Bid (Branch_ID) are keys to the relevant records/rows.

2.5.7 The TPC-C benchmark

Due to TPC-A and TPC-B limitations, the Transaction processing Performance Council (TPC) published a new benchmark, the TPC-C [TPC 91]. As the TPC-A and TPC-B were so insufficient to measure database performance, the TPC have put every database transaction type in the new benchmark. The Benchmark C (TPC-C) is modelled after an Order-Entry workload. The benchmark is a mixture of read-only, read-write, scan, sort, count and update intensive transactions.

The performance metric reported by TPC-C is a "Business throughput" measuring the number of orders processed per minute. The performance metric for this benchmark is expressed in transaction-per-minute-C (TPM-C). All references to TPM-C results must include both TPM-C rate and the price-per-TPM-C to be compliant with TPC-C standard.

The benchmark does not reflect the entire range of OLTP requirements. In addition, the extent to which a customer can achieve the results reported by the vendor is highly dependent on how closely the TPC-C approximates the customer application. The relative performance of systems derived from this benchmark does not necessarily hold for other workloads or environments. The extrapolation to unlike environments is not recommended. TPC-C results are not comparable to other TPC benchmarks' results.

2.5.7.1 The TPC-C Logical Database Design

TPC benchmark C simulates the activity of a wholesale supplier. The workload is centred around the activity of processing orders.

The company is a wholesale supplier with a number of geographically distributed sales districts is created. Each regional warehouse covers 10 districts. Each district

serves 3000 customers. All warehouse maintain stocks for 50,000 items sold by the company.

Orders are composed of an average of 10 order lines (i.e., line item). One per cent of all order lines are for items not in-stock at the regional warehouse and must be supplied by another warehouse.

The components of the TPC-C database are defined to consist of nine separate and individual tables.

2.5.7.2 The TPC-C Transactions Profile

The Benchmark script consists of five different transactions each one of these transactions consists of several database operations. The first three transactions are on-line transactions and the last two are batch transaction executed with relaxed response time. The transactions are:

1. The New-Order Transaction.
2. The Payment Transaction.
3. The Order-Status transaction.
4. The Delivery Transaction.
5. The Stock-Level Transaction.

1. The New-Order Transaction

The New-Order transaction consists of entering a complete order through a single database transaction. Entering a new order is done in a single database transaction that is shown below.

1. Create an order header, comprised of:
 - 2 row selections with data retrieval,
 - 1 row selections with data retrieval and update,
 - 2 row insertions.
2. Order a variable number items (average $ol_cnt = 10$) comprised of:
 - (1* ol_cnt) row selections with data retrieval,
 - (1* ol_cnt) row selections with data retrieval and update,
 - (1* ol_cnt) row insertions.

2. The Payment Transaction

The Payment transaction updates the customer's balance and reflects the payment on the district and warehouses sales statistics. In addition, this transaction includes non-primary key access to the CUSTOMER table. The Payment transaction enters a customers payment with a single database transaction, and is shown below.

Case 1, Customer is selected based on customer number:

3 row selections with data retrieval and update,
1 row insertion.

Case 2, Customer is selected based on customer last name

2 row selections (on average) with data retrieval,
3 row selections with data retrieval and update,
1 row insertion.

3. The Order-Status transaction

This transaction queries the status of a customer last order. It represents a mid-weight read-only database transaction with a low frequency of execution and response time requirements. In addition, this table includes non-primary key access to the CUSTOMER table. Querying for the status of an order is done in a single database transaction that is presented below.

1. Find the customer and his/her last order comprised of:

Case 1, Customer is selected based on customer number:

2 row selections with data retrieval.

Case 2, Customer is selected based on customer last name:

3 row selections (on average) with data retrieval.

2. Check status (delivery data) of each item on the order (average items-per-order = 10), comprised of:

(1 * items-per-order) row selections with data retrieval.

4. The Delivery Transaction

The Delivery transaction consists of processing a batch of 10 new (not yet delivered) orders. Each order is processed (delivered) in full within the scope of read-write database transaction. The number of orders delivered (batched) within the same database transaction is implementation specific. The transaction composed of one or more (up to 10) database transaction, has a low frequency of execution and must complete within a relaxed response time.

The Delivery transaction is intended to be executed in deferred mode through a queuing mechanism, rather than interactively with terminal response indicating transaction completion. The result of the deferred execution is recorded into a result file.

Deferred execution is characterised by queuing the transaction for deferred execution, returning control to the originating terminal independently from the completion of the transaction, and recording execution information into a result file.

The deferred of the Delivery transaction delivers one outstanding order (average items-per-order = 10) for each one of the 10 districts of the selected warehouse using one or more (up to 10) database transactions. Delivering each order is done in the explained below.

1. Process the order, comprised of:
 - 1 row selection with data retrieval.
 - (1 + items_per_order) row selection with data retrieval and update.
2. Update the customer balance, comprised of:
 - 1 row selection with data update.
3. Remove the order from new order list, comprised of:
 - 1 row deletion.

5. The Stock-Level Transaction

The Stock-Level transaction determines the number of recently sold items that have a stock level below a specified threshold. It represents a heavy read only database transaction with a low frequency of execution, a relaxed response time requirement, and relaxed consistency requirements.

Examining the level of stock for items on the last 20 orders is done in one or more database transactions with the steps shown below.

1. Examine the next available order number, comprised of:
 - 1 row selection with data retrieval.
2. Examine all items on the last 20 orders (average item_per_order = 10) for the district, comprised of:
 - (20 * items_per_order) row selection with data retrieval.

3. Examine, for each distinct item selected, if the level of stock available at the local warehouse is below the threshold, comprised of:
At most $(20 * \text{items_per_order})$ row selections with data retrieval.

2.5.7.3 Limitations of The TPC-C Transactions

The main limitation of the TPC-C is having an exceptionally expensive transaction mix in terms of response time. The benchmark script consists of five programs and 39 different database operations.

Additionally, as the research will present in later chapters, the benchmark design can not be directly mapped to the OLTP environment. When the transaction mix of the TPC-C benchmark was compared to real life OLTP transactions, the benchmark workload patterns was widely different from the research results. The benchmark measurements are difficult to compare as the benchmark includes two long batch transactions with relaxed response time. Relaxed response time will be difficult to be compared from one system to another. The TPC-C technical limitations are due to the over complication of the following points:

- Number of Tables (9 tables);
- Table size (over 40,000,000 rows occupying around 5.2E10 bytes);
- Transaction mix.

Due to those limitations, since the benchmark was first published in December 1991, after extensive enquiry, it appears that at the time of writing this thesis not a single user has used it and the current author could not find a single paper discussing the TPC-C results. The technical limitation of the TPC-C benchmark are discussed in details in chapters three (§3.2), chapter four (§4.5) and chapter seven (§7.3).

2.6 Limitations of The Existing Benchmarks

The presented benchmarks do not rigorously discuss the possible testing variables and design characteristics necessary for a generalised methodology. They provide an approximate figure of the database systems performance. What they will not provide is a realistic characterisation that is set as the target for the benchmarking process. This is due to several pitfalls most of the research did not take into consideration. This research managed to overcome some of those limitations and did not manage to overcome some

others, this will be discussed in more details in chapter nine. An ideal benchmark will overcome the following pitfalls:

1. The benchmarks are data model dependent, (e.g., tree dependent or network dependent or relational dependent). A benchmark should be general enough to tolerate all the popular database models, since many large organisations will be using more than one.
2. The benchmarks assumed uniform demand for requested records. Uniform demand for requested records is a reasonable assumption for a very limited set of applications.
3. The benchmarks were applied as a single user benchmark.
Benchmarks that involve multiple jobs will give a more realistic picture of database performance. Included in any benchmark should be a methodology for constructing job mixes for multiple-user benchmark. The performance indices of the multi-user benchmark will be compared to the performance indices derived from single-user benchmarks.
4. The effect of increasing the number of database users on the database performance was not studied.
5. The data models in the benchmark experiments did not relay on studying real life database systems.
6. Benchmark database size is too small to be realistic.
The behaviour of DBMS is not linear concerning size. Accordingly, to scale the performance by multiplying both size and response time by a certain factor is a wrong assumption. In this context, size is the actual size in bytes occupied by the database on the disk, the number of transactions that the DBMS must handle in a given interval, number of relations, number of rows per relation, width of rows, and distribution of attributes in the rows. Each one of these items should be isolated and tested to create a realistic benchmark.
7. The benchmarks use only simple transaction types. Real life database systems use, relatively, more complex transaction mix.

8. Transaction results and how they are treated is not clear. Transaction results can be directed to one of three things: the main memory, a log file, and computer terminal, in each case response time or system throughput will be widely different.

There are several other performance criteria that none of the discussed benchmarks studied carefully. These factors include: database reorganisation time, the effect of the single and massive updates operations, the effect of compound keys, and the database capabilities of report generation facilities.

2.7 Conclusion

Due to design limitations, the benchmarks presented in this chapter might provide an estimate of the system's performance but will not provide a realistic characterisation of database management, that is the main objective for the benchmarking process. A benchmark methodology for database systems must consider a wide variety of systems variables to fully evaluate performance. To achieve that, the system must be tested with a load as close as possible to that which it will be running in real life [MOHR84]. Some idea of the type of information contributing to this achievement is identifying the following database applications characteristics:

1. The dominant processing tasks.
2. The types of queries will be run.
3. The relative frequency of each query.
4. The size of the database.
5. The patterns of behaviour are expected.
6. The types of references to the database (e.g., random or localised).

The limitations of the previous benchmarks stem from their lack of proper background study and being built on purely intuitive basis. To that end, series of field studies have been conducted in large UK organisations. These organisations are characterised by:

1. Large number of running applications.
2. Large number of on-line users.
3. Large number of on-line transactions.
4. Large number of large databases.
5. Complex database environments.

These studies are based on conducting in depth data collection and analysis to all the transactions issued in those environments over the period of study. The studies aimed to identify the main characteristics of on-line database and on-line transactions in high-volume transactions' environments

CHAPTER 3

PROBLEM DEFINITION AND DATA GATHERING AND ANALYSIS TECHNIQUES

CHAPTER 3

PROBLEM DEFINITION AND DATA GATHERING AND ANALYSIS TECHNIQUES

Chapter two reviewed several database performance benchmarks. For each one of the reviewed benchmarks three aspects were examined: database used, transaction set of the benchmark test and the performance metrics. These benchmarks were criticised for suffering from one limitation or another. Those problems stem from the lack of proper background studies. This chapter discusses the technical limitations of the TPC benchmarks and proposes empirical solutions to overcome those limitations.

3.1 Introduction

The Debit/Credit (TP/1) [ANON85] and the Transaction Processing Council (TPC) benchmarks [TPC 89, GRAY91] have become the database market standard practice. Those benchmarks have a number of practical limitations [REVE90]. Most of those limitations stem from the lack of background study before the benchmark design stage. The main criticism of the TPC benchmarks is that they do not provide a realistic characterisation that can be set as the target for the benchmarking process, and more than this, they do not even provide a model of the ATM systems whose performance they are supposed to simulate [REVE92b].

As benchmark results are representative of those types of transactions actually included in the benchmark set, it is impossible to generalise those results to all kinds of systems transactions. This research work aimed to identify the main characteristics of on-line databases by examining real database performance factors. Those factors have been discussed by Ferrari [FERR78, FERR83], Dongara [DONG87], and Hawn [HAWN87]. By applying the analysis of those performance factors at several different environments, the data from that analysis can supplement the data from the live environments once the basis of this analysis has been defined to evaluate performance in a transaction processing environment.

This chapter presents the technical limitations of the TPC benchmarks. It discusses those limitations explaining why they might not be representative when comparing database management systems. It also specifies the nature of high-volume transactions' environments and discusses processing environments that can represent those domains. This chapter also specifies the basis of selecting the studied organisations and presents the processing environments of those organisations. Additionally this chapter discusses data gathering methodologies that were utilised by this research. Finally, it presents some statistical techniques to analyse the collected information.

Section 3.2 defines the problem of the database industry as the complete reliance on the TPC benchmarks as the standard practice in the database industry. It shows that the TPC benchmarks' results suffer from inconsistency between systems, in consistency when applied and several technical limitations.

Section 3.3 proposes an empirical approach to solve the problem. That approach will examine transaction behaviour and large database characteristics in large database environments. It defines the characteristics to be studied and presents the criteria of selecting the studied organisations.

Section 3.4 presents the results from several experiments to test some performance factors to be investigated in the selected environments. It experimented database performance factors such as :database size, row size effect, database indexed attributes, attributes types and distribution, transaction database operations, on-line transactions time utilisation and i/o operations, join operation, transaction complexity, transaction output, and background workload.

Section 3.5, presents different approaches of studying organisations and methodologies of gathering data. It discusses approaches such as: pure basic research; basic objective research; evaluation research; applied research; and action research. Additionally this section presents data gathering techniques in general and the techniques used in this research in particular.

Section 3.6, present some statistical techniques used to analyse the collected data. The section defines basic statistical terms and explain the process of experiments design. It also presents some comparative techniques that are eventually used by this research.

Section 3.7, present the chapter summary and conclusion .

3.2 Problem Definition

The Debit/Credit (TP/1) [ANON85] and the Transaction Processing Council (TPC) benchmarks [TPC 89, TPC 90, TPC 91, GRAY91] have become the database market standard practice. Those benchmarks have a number of practical limitations [REVE90]. Most of those limitations stem from the lack of background study before the benchmark design stage. The main criticism of the TPC benchmarks is that they do not provide a realistic characterisation that can be set as the target for the benchmarking process, moreover, they do not even provide a model of the ATM systems whose performance they are supposed to simulate [REVE92b]. In the following sections, those limitations will be discussed.

3.2.1 Inconsistency of TPC Benchmarks Implementation

The TPC benchmarks were constructed to quantify and compare the throughput and price/performance ratio of various transactions processing systems. The TPC-A and TPC-B benchmarks measure transactions per second from the time a teller issues a request to the time the customer receive acknowledgement that the transaction has completed. They include simulating many terminal users, there "think time" and network traffic time.

However, both benchmarks are difficult to implement in their full form, and include aspects like network protocols and think-time that do not specifically relate to DBMS performance. TPC-A has become a fairly standard practice, i.e., just about every DBMS benchmarker runs TPC-A. The actual benchmark, though, is not so standard and once one goes beyond the surface, it becomes clear that no two versions of TPC-A can be considered equivalent. Without a formal specification, each vendor has had considerable differences in defining TPC-A as it saw fit, and so a comparison based on TPC-A benchmark results can be misleading. It seems that each vendor has included the portions of TPC-A that will show the particular product in the best light and has chosen to omit or modify other portions without regard for the preservation of the original TPC-A definition. An example was given by R. Fox [FOX 89], DEC published Debit/Credit figures for both its own mid-range VAXs and two of IBMs competing machines, the 9377-90 and 4381-22. These figures showed that the DEC machines had three times better price/performance than the IBM systems. Accordingly, IBM countered by running its own version of TPC-A, the results they got were very

different: the IBM machines processed over three times as many transactions in IBM own research centre and that translates into better price/performance than DEC.

Generally, database vendors can implement the TPC benchmarks in different ways. Any of the three TPC benchmarks can be implemented in the following ways:

1. Both benchmarks simulate users with block-mode terminals sending and receiving messages through communication lines using X.25 line protocol. Different types of communications will cause different degrees of overhead on the system. Obviously, no communications overhead should allow the benchmark transaction to provide better response time.
2. The TPC-A benchmark transaction simulates users entering transactions at a rate of one per 100 seconds. This can be thought of as the users "think time". That means a 100 TPs system will consist of 10,000 terminals. Users found that too difficult to implement. Sometimes users define their own think time or variable think time and in some cases no think time. The think time ratio will greatly affect the response time.
3. The benchmark transaction imposed the requirement that 95% of all transactions must be completed within one second. Vendors do not restrict themselves to 95% ratio and some published results are based on 92%, 90% or even 85% ratios.
4. The TPC-A database consists of four databases:

Branch	:	100 KB, random access.
Teller	:	1 MB, random access.
Account	:	1 GB, random access.
History	:	10 GB, sequential access.

That means a 10 TPs system would consist of:

Branch	:	1 MB.
Teller	:	10 MB.
Account	:	10 GB.
History	:	100 GB.

Many users find this size is too much to handle and usually they scale down the database. The use of a database that is greatly scaled down may indicate that the system has problems handling large amounts of data and may also provide better results. If the files are scaled down enough, they may be

residing in core memory during the execution of the benchmark, providing much better results than can be expected in a real system due to the elimination of much I/O.

5. A transaction can be rejected due to several reasons such as syntax errors or deadlocks. If rejection of transactions is permitted and the benchmark counts rejected transactions, the performance results are improperly high since all transactions have not done the intended work. In fact, with this scenario, the system with the most deadlocks or input errors would provide the best results since these transactions would do no work, yet would be counted as though they had been successfully completed.
6. The benchmark might count all transactions or define a "stable-state", a period when all users are executing concurrently. Transactions completed outside a stable-state will provide better results due to a lighter load on the system.
7. If the input data are not randomised, it could be that much work is being performed in memory, eliminating the need for I/O, which may not be realistic.
8. The concept of price/performance should be studied in conjunction with the application type or application domain. If the database is to be primarily used in batch operations, the performance criteria will focus on direct computer costs. For a system to be used mainly in interaction with human beings, the response time will be an important factor and computer costs will come second. Any way, usually the published prices differ largely from the agreed prices at purchase time.
9. The benchmark required all updates be logged and that the log file be duplexed. Omitting logging would probably increase transaction performance.

The extent to which a customer can achieve the results reported by a vendor is highly dependent on how closely TPC-Benchmarks approximates the customer application. The TPC [TPC 92] agrees that:

"The benchmark results are highly dependent upon workload, specific application requirements, and systems design and implementation. Relative

system performance will vary as a result of these and other factors. There, TPC-Benchmarks should not be used as a substitute for a specific customer application benchmarking when critical capacity planning and/or product evaluations are contemplated"

3.2.2 Technical Limitations of The TPC Transactions

The database industry and most researchers in the database performance area has criticised the TPC transactions for being too simple to apply realistic workload on the tested systems. To investigate this assumption the present author has conducted several empirical studies to examine the scalability of the basic database operations of the TPC-A and the TPC-B transactions. the basic database operations of the TPC-A and the TPC-B transactions are:

- Select one row using unique key;
- Update one row using unique key;
- Insert one row.

Insert operation was tested by loading four tables with the required number of rows, then insert row in a fifth table for fifteen minutes and calculate the average time. The scalability of those database operations was tested under increasing database size. Transaction response time was taken as the main index for measuring the effect of increasing table size. The main assumption was, if the basic database operations can apply significant workload on database management systems, that will be reflected as decreasing response time as the test table size increase.

The experiments took place in two computer environments, a standalone PC environment and a SUN SPARC environment. In the standalone PC environment, table size started by 5000 rows and was increased by 5000 rows until it reached 30,000 rows. In the SUN SPARC environment table size started by 5000 rows and increased by 5000 rows until it reached 100,000 rows. In both environments, rows size was 200 bytes.

The results of the experiment in the standalone PC environment are presented in table 3.1, and the results of the experiment in the SUN SPARC environment are presented in table 3.2. Fig. 3.1, summarises the result from both experiments. The results shows the following.

1. Transactions' cost in both environments are too ineffectual. Even after increasing table size to 100,000 rows in the SUN SPARC environment, the most expensive operation was qualified retrieval that required around .2 of a second to finish. This cost is too small to test real systems in real environments.
2. In the standalone PC environment, qualified retrieval and update operations showed positive scalability, but the difference between transactions response time against the largest table size and the smallest table size is so trivial. For qualified retrieval operation the difference was 0.329 of a second. For update operation, the difference was 0.052. Insert operation was not scalable. Having in mind that PCs resources are limited, those operations in a larger environments will not apply sufficient workload scalability.
3. The same pattern was found in the SUN SPARC environment. Insert operation took negative slope, and the smallest response time was against the largest table size. Qualified retrieval produced values around 0.2 of a second and its response time at 100,000 rows was not much different than that at 10,000 rows (0.01 of a second).
4. Update operation response time did not reflect the difference between the standalone PC resources and the SUN SPARC resources. Comparing the two computers resources, the SUN SPARC environment was considerably larger than the PC environment in all aspects of memory size, processor power and disk speed. Despite that difference update operation response time in the PC environment was 0.420 of a second and 0.031 for the SUN SPARC at table size of 5000 rows. When table size was increased to 30,000 rows (maximum for PC), the difference in response time in the PC environment and the SUN SPARC still did not reflect the difference in the resources of the two environments. The PC response time was 0.474 and the SUN SPARC response time was 0.032.

Size in 1000 Rows	5	10	15	20	25	30
Select	2.764	2.932	2.980	3.024	3.054	3.093
Update	0.420	0.438	0.451	0.460	0.468	0.474
Insert	0.490	0.521	0.480	0.504	0.508	0.510

Table 3.1, Three database operations in PC environment

Size in 1000 Rows	10	20	30	40	50	60	70	80	90	100
Select	.188	.210	.204	.173	.206	.201	.233	.182	.184	.198
Update	.031	.033	.032	.039	.044	.044	.045	.041	.042	.046
Insert		.188		.188		.147		.134		.133

Table 3.2, Three database operations in SUN SPARC environment

The experiments showed that the previous database operations suffer several technical limitations such as insufficient workload, and poor scalability level. Since the TPC benchmarks' transactions consist of those operations, they will inherit the same limitations. The benchmark will suffer from the following limitations:

1. Transaction cost will be too small to reflect realistic workload.
2. Transaction scalability will be too small to properly test systems under increasing loads.
3. Transaction cost between different systems is insignificant to establish a realistic comparison.

Recently, one of the leading database companies in the world has conducted similar experiment. They isolated the TPC-A transaction mix and tested it against two database sizes, the first is relatively small, 0.75 Gb, and the second is much larger, 30.2 Gb. In both cases they fixed all other parameters such as number of active terminals and computer architecture. The result was similar to our findings. While database size has increased 40.3 times, the difference in response time between the two tests was around three seconds. That small difference in response time shows that the TPC-A mix is not sufficient to apply realistic load in large database environments. The result is presented in table 3.3.

Run Number	Database Size	Number of Terminals	Response Time
Run 1	0.75 Gb	1300	0.63 secs
Run 2	30.2 Gb	1300	0.66 secs

Table 3.3, The TPC-A against different table sizes

3.2.3 Inconsistency of the TPC Transaction Comparisons

The TPC transaction database operations are too simple to represent all the domain of OLTP. That is because one system may be excellent at performing one

transaction type and behave in a different manner when performing another transaction type. Table 3.4 presents a good example of changing behaviour according to different transactions' scripts. It shows a comparative study between IBM 3090/400S and Tandem CYCLONE [SIVU90], when a small query and big query were added to the OLTP test (Update and Insert only, similar to TPC-A), system behaviour responded differently.

The previous example showed that systems behaviour varies depending on transaction scripts. That is why benchmarks should be as accurate as possible in simulating different domains.

Test Type	IBM TPs Performance	Tandem TPs Performance
OLTP only	30.5	30.4
OLTP + small query	30.0	30.5
OLTP + big query	30.3	30.5
OLTP + RAMP up	48.2	44.3

Table 3.4, Comparison between an IBM and Tandem

OLTP : On-line Transaction processing.
TPS : Transactions per second.
RAMP : Requirement Approach for Measuring Performance.

3.3 Empirical Approach To Solve The Defined Problem

The TPC benchmarks have become the standard practice in the database industry, however, as we saw in previous sections when it comes to implementation, they are not so standard. Additionally, the TPC benchmarks' transactions suffer several technical limitations that affect their ability to realistically compare real database systems. As benchmark results are representative of those types of transactions actually included in the benchmark set. It is impossible to generalise those results to all kinds of systems transactions.

Since this research objective is to create a comprehensive benchmark with increasing number of users that approximates the behaviour of typical DBMS applications, that benchmark methodology should represent a wide variety of systems variables. That benchmark should also take into consideration all the limitation discussed in previous sections.

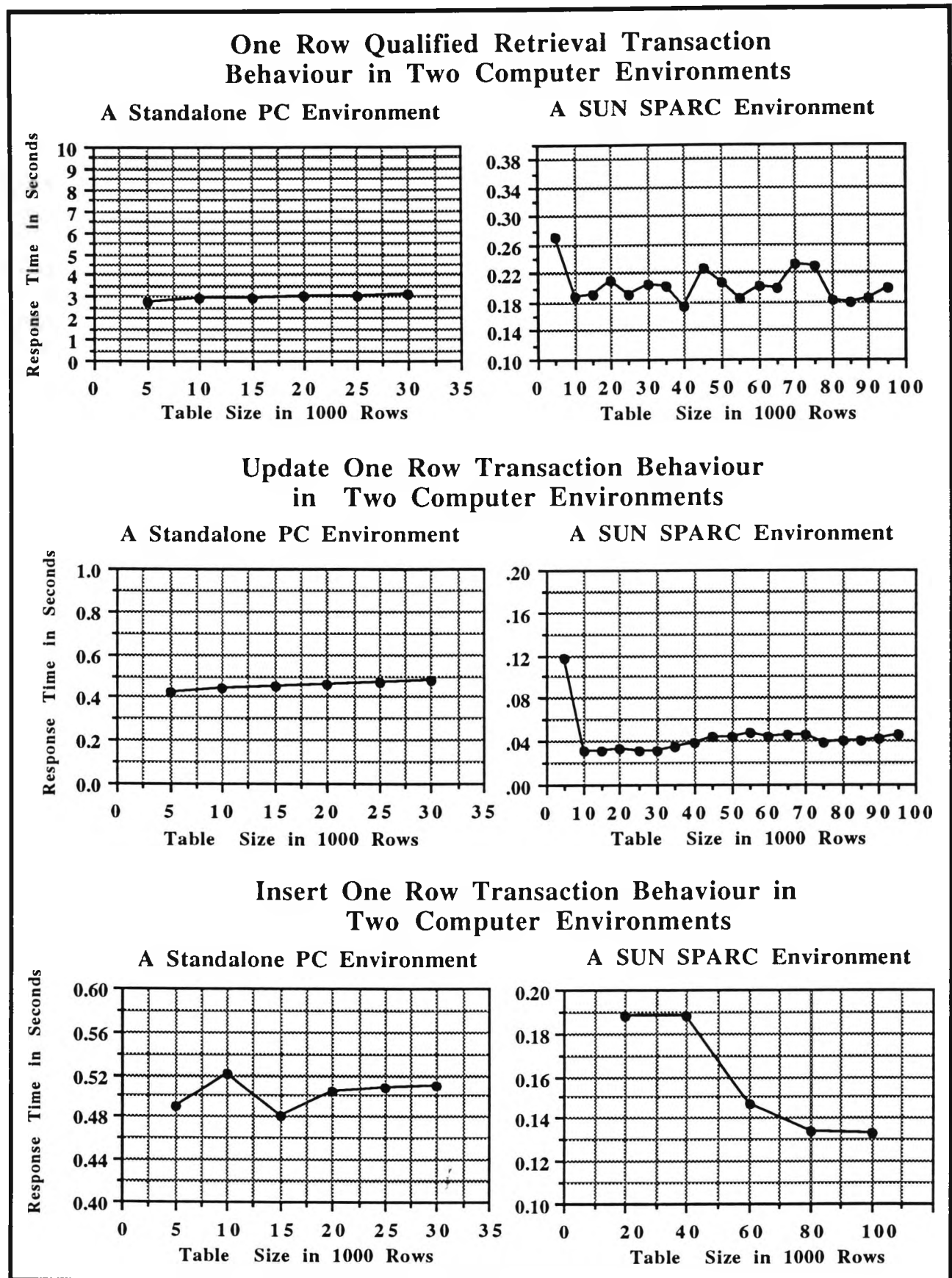


Fig. 3.1, Effects of three database operations

Generally, to achieve realistic metrics, a database system should be tested with a load as close as possible to that which it will be running in real life. Some idea of the type of information contributing to this achievement are: identify the dominant processing tasks, what queries will be run, what is the relative frequency of each query, what is the size of the database, and what patterns of behaviour are expected. To this end, the present author developed a series of studies to examine these factors.

Those studies aimed to identify the main characteristics of on-line databases by examining real database performance factors. Those factors have been discussed by Ferrari [FERR78, FERR83], Dongara [DONG87], and Hawng [HAWN87]. By applying the analysis of those performance factors at several different environments, the data from that analysis can supplement the data from the live environments once the basis of this analysis has been defined to evaluate performance in a transaction processing environment.

The following sections identify the domain of study and discuss the basis of organisations selection for this research. They also discuss several experiments that aimed to examine the effect of several database factors on database transactions' response time.

3.3.1 Specification of the Domain of Studies

This section defines the term "high-volume transaction environments" and discusses the characteristics of those environments. Equally some people call it "On-line Transaction Processing (OLTP) environments". It also discusses the criteria for selecting the organisations covered by this research and explains why they were good examples of high-volume transaction environments. Additionally, it presents and justifies the selection of database performance factors covered by this research.

3.3.1.1 Definition of High-volume Transactions Environments (OLTP)

The importance of high-volume transaction environments (OLTP) is due to its share of the database industry; this environment is the dominant environment at the database market and sets the rules for database standards.

High-volume transaction environments (OLTP) are those environments that support large number of on-line terminals that originate heavy transaction load on the

running environments. The number of terminals in those environments is in the thousands and the number of transactions is in thousands of transactions per hour. Those environments support large number of databases that are characterised by their large sizes. Examples of such environments include banks, airlines, government organisations, and large business organisations. Organisations that fall into the domain of high-volume transaction environments have to comply with the following criteria:

- have a large number of running applications;
- have a wide variety of those applications;
- have a large database environment;
- have a heavy transactions load;

3.3.2 Criteria for Organisations Selection

The studies examined the running systems at three large UK organisations. The main objectives of those studies were to identify the salient characteristics of actual database applications in high-volume transaction environments to build a database benchmark that represents those environments. The research selected those organisations because of the large number of applications they have and the wide variety of those applications. Also they have a large database environment and heavy on-line transaction load. The following sections will discuss the features in each organisation that qualified it for this research selection.

3.3.2.1 The Local Authorities Computer Centre

The first study took place at a large local authorities' computer centre. That centre offers information technology services and software consultants to four London boroughs. Each of those has its own income sources and spending activities. This organisation was selected because of its large number of running applications, (over 300 running applications), their large database environment that comprises 150 on-line databases occupying 12.5 Gbytes and the large number of on-line terminals, (over 2500 on-line users), connected to the system.

The local authority on-line system users heavily access the systems. The on-line system utilisation could peak to reach 20,000 transactions per hour on a normal working day. On the busiest day of the period, the system load peaked to reach 23,000 transactions per hour. During the period of study, the running systems were under an

average load of about 17,000 transactions per hour. Fig. 3.2 illustrates the pattern of average system load over thirty working days.

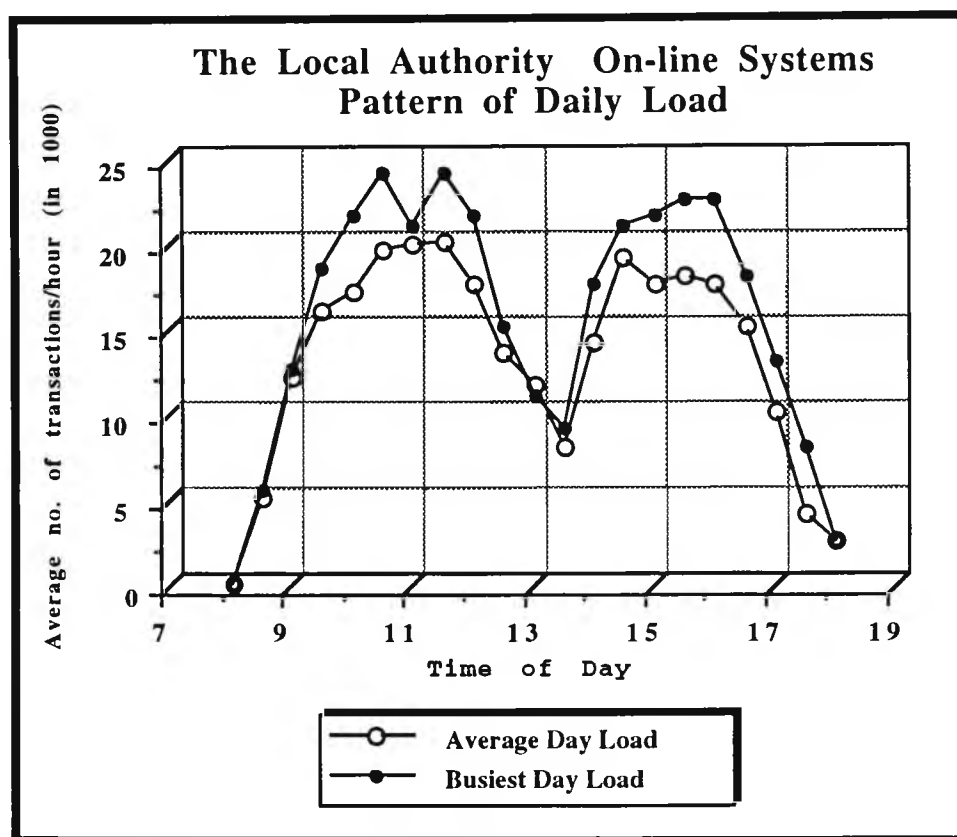


Fig. 3.2, The Local Authority average daily load

3.3.2.2 The Airlines Computer Centre

The second study took place at a large airline computer centre. The centre computing power supplied the running environment by 600 MIPS, where 286 MIPS were dedicated to on-line service. The application databases investigated comprised over 2000 hierarchical and relational databases, occupying 436 G-Bytes. The running system terminal population was around 250,000 terminals, out of those, over 27000 concurrently accessed the on-line service.

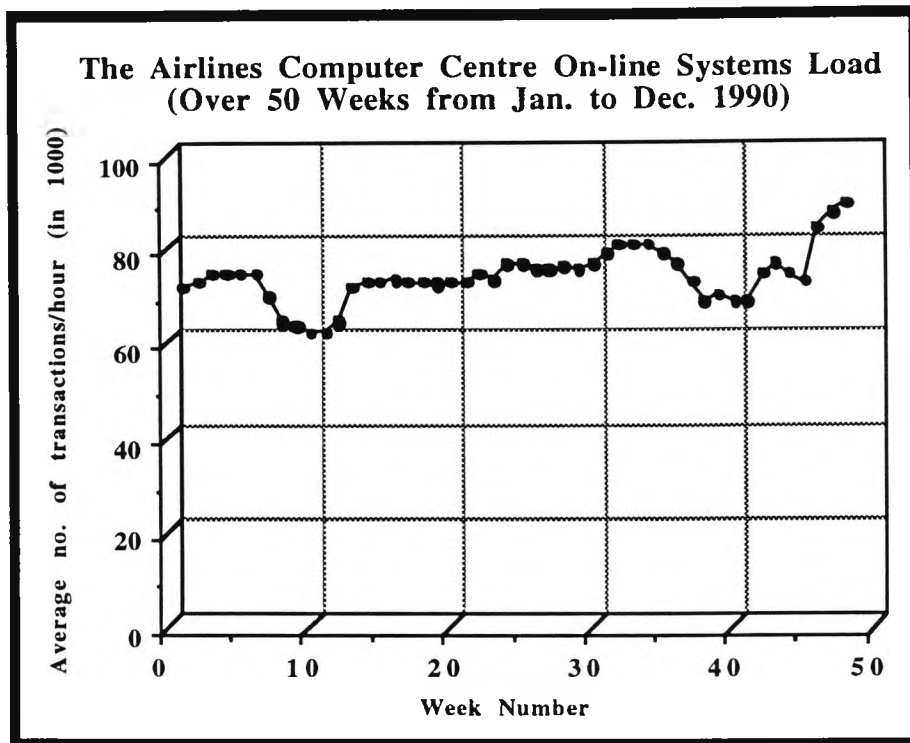


Fig. 3.3, The Airlines average daily load

On-line system load was measured over a period of fifty weeks as the average of number of transactions per hour. The average number of transactions issued to the system were measured by snap shots at fifteen minutes' intervals. These snap shots were accumulated to give the average system load per hour.

Systems load was measured separately for both IMS environment and DB2 environment then the total was calculated. The on-line IMS database environment was older than the relational DB2 database environment and many applications were still running under the IMS system, but the airlines' computer centre was in the process of converting the running IMS hierarchical systems to relational DB2 systems. The average load at the IMS systems environment was around 45,000 transactions per hour. The average system load at the DB2 systems environment was around 25,000 transactions per hour. The average of total load for all the running systems at the airlines' environment was around 70,000 transactions per hour. Fig. 3.3 illustrates the average of the airlines on-line systems load.

3.3.2.3 The Bank Computer Centre

The third study took place at a large bank computer centre. The bank computing power was divided between two production centres and a development centre. Each

one of those centres produced 500 MIPS. When one of the production centres goes down, the development computer centre works as a back-up for that centre, which allows the bank to maintain 1000 MIPS of computing power to on-line service under different circumstances. The data centres serve more than 3500 branches with a terminal population of about 22,000 that supplies on-line service to over 60,000 users. The number of customers' accounts was around 17M with average accounting entries per day of around 6M accounts whereas on peak days it reaches 20M accounts per day and peak number of transactions per second that reaches 650 transactions. In addition, the system supported more than 3500 ATMs having on-line access to the system. The bank supplied its services not only to UK branches but also to several other branches all over the world, which results in 24-hour access to the computer system.

The bank branches (on-line terminals) load peak to reach 75,000 transactions per hour and the ATM transactions' load reaches a peak of 130,000 transactions per hour. Fig. 3.4, illustrate a one week pattern of transactions' load. The average day load of the running systems was around 90,000 transaction per hour, that was rather heavy system load to choose the bank environment for this research.

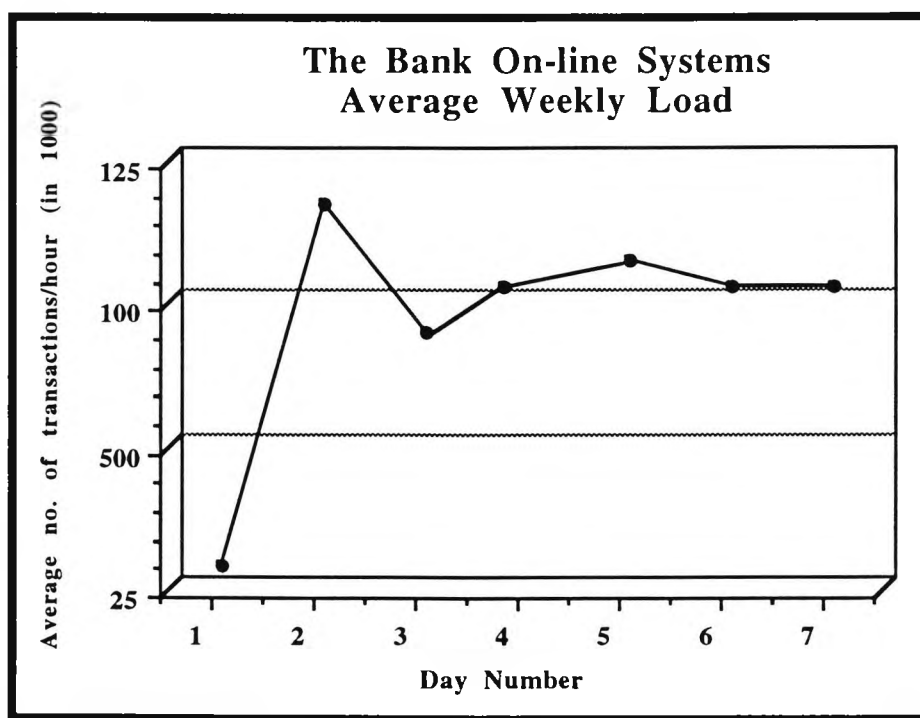


Fig. 3.4, The Bank average weekly load

Fig. 3.5 presents the on-line systems day load at the bank environment. The bank branches and the ATM dispensing machines accessed the running systems over the 24 hours. The service was available to the bank branches from 7:00 to 22:00 on working days, and from 7:00 to 19:00 on weekends. The ATM service was available 24 hours a

day seven days a week. Fig. 4.4 presents the day load of the bank branches and the day load of the ATM machines. A third line shows the total day load at the bank environment.

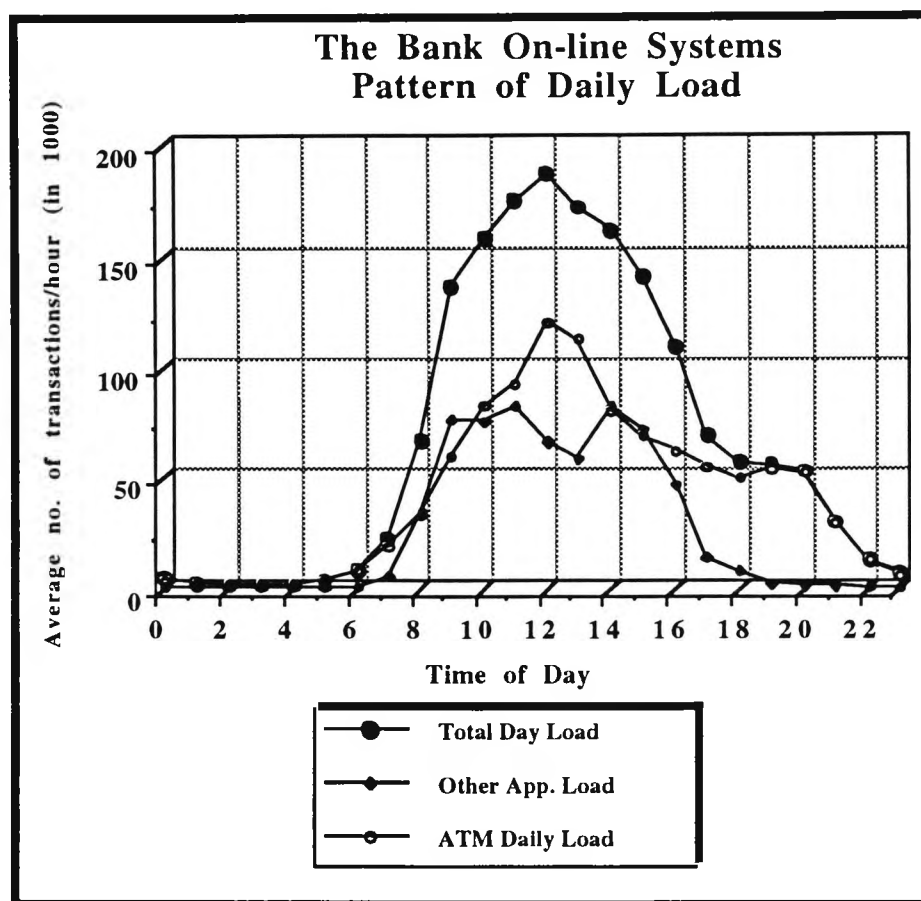


Fig. 3.5, The Bank average daily load

3.4 Empirical Studies to Test Database Performance Factors

To design a benchmark that compares different DBMS, further detailed information about database characteristics should be investigated. The main limitation of other benchmarks is that they do not provide a realistic characterisation that is set as the target for the benchmarking process, more than this, they can not even provide an estimate of the systems performance. Benchmarks that involve realistic database operations will give a more realistic picture of database performance. To achieve that, a system must be tested with a benchmark script as close as possible to real life. In this research several experiments that involve independent performance components of database management systems were conducted. Those experiments identified the main characteristics of on-line databases by examining the main factors affecting database performance. Those factors were discussed by researchers such as Ferrari [FERR78,

FERR83], and Dongara [DONG87]. By applying the analysis of those performance factors at several different environments, that data from that analysis can supplement the data from the live environments. Once all the information is analysed, the tested applications can be simulated. Those factors are the following:

- database size;
- row size;
- attributes types;
- database indexed attributes;
- Distribution of attributes
- identify on-line transactions script represented as number of database operations;
- identify on-line transactions time utilisation and I/O operations;
- identify number of databases accessed by one transaction (JOIN Operation);
- identify transactions complexity (number of nested selects);
- identify key utilisation;
- identify the nature of transaction output;
- identify the background workload.

To select the previous factors, the present author conducted several empirical studies to test the effect of each of them. Factor effect on transaction response time was the main index to measure the factor effect. The tests took place at two different environments. The first was an IBM PC running relational DBMS, and the second was a SUN SPARC Micro System running relational DBMS (we are not allowed to publish the name of the relational DBMS). The findings from studying the two environments showed the severe effect of those factors on transaction response time had justified the selection of those factors for this research. The factors' tests used different transactions' mixes one mix was the CITY benchmark transaction mix, (next chapter discusses the CITY benchmark). Some of the tests used pure sequential retrievals and some others used a mix of different database operations. For each study, the transactions mix used for the test will be presented.

The tests databases' sizes were scaled up gradually from 5000 rows to 20,000 rows in the PC environment and from 5000 rows to 100,000 records in the SUN SPARC environments. The final tables' sizes in the PC environment was smaller than that of the SUN SPARC because of disk space constraint. Row size in both environments was 200 bytes.

In all experiments, the CITY transaction was used as the main testing workload, unless otherwise mentioned. The CITY transaction is discussed in details in chapter five.

The following sections present the results from those empirical studies. The effects of factors regarding database characteristics are presented first then the factors regarding on-line transactions' behaviour are presented later.

3.4.1 Table Size

Usually people assume that the behaviour of DBMS is linear concerning table size. It is important to mention that response time might not increase linearly against the increase in table size because table size on its own is not the only factor that affect response time, some other factors such as: the effect of table index type; cardinality; page size; and operating systems can also have an effect on transaction response time.

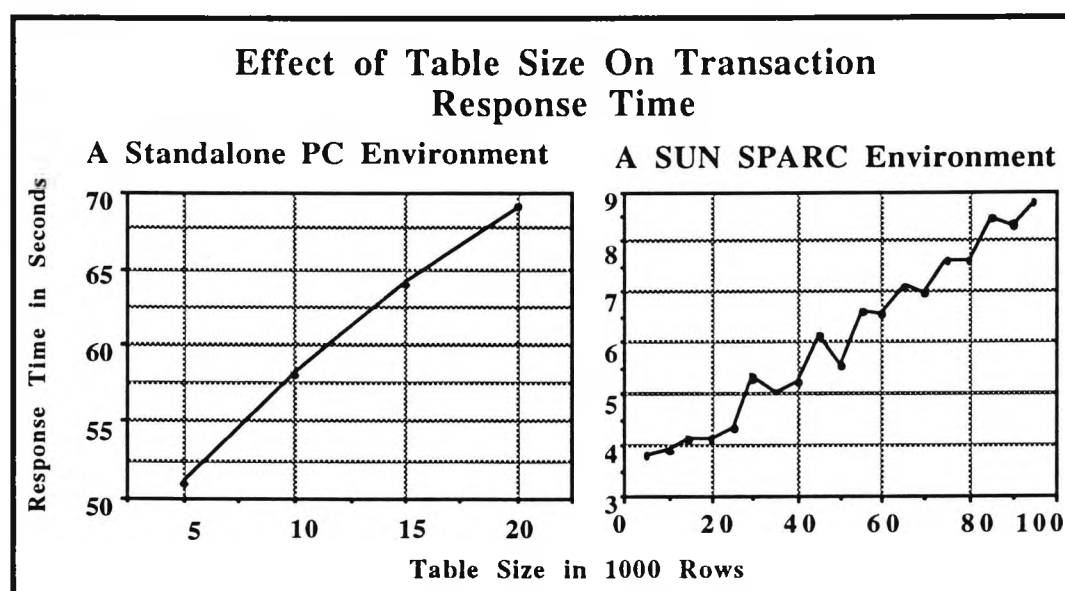


Fig. 3.6, Effect of table size on transaction response time

Also database size could mean several things. It could mean: total table size including overheads; actual table size in bytes; number of tables in database space; and it could mean the number of rows in the table.

The only way to answer those questions are by examining large number of databases and to try to find a common pattern between them. Fig 3.6 shows the effect

of database size on transactions' response time in a standalone PC environment and a SUN SPARC environment. In both environments, transaction response time increased linearly as database size increased. As the following chapters will show, transaction behaviour in relation to table size depends largely of two factors, transaction mix and database environment.

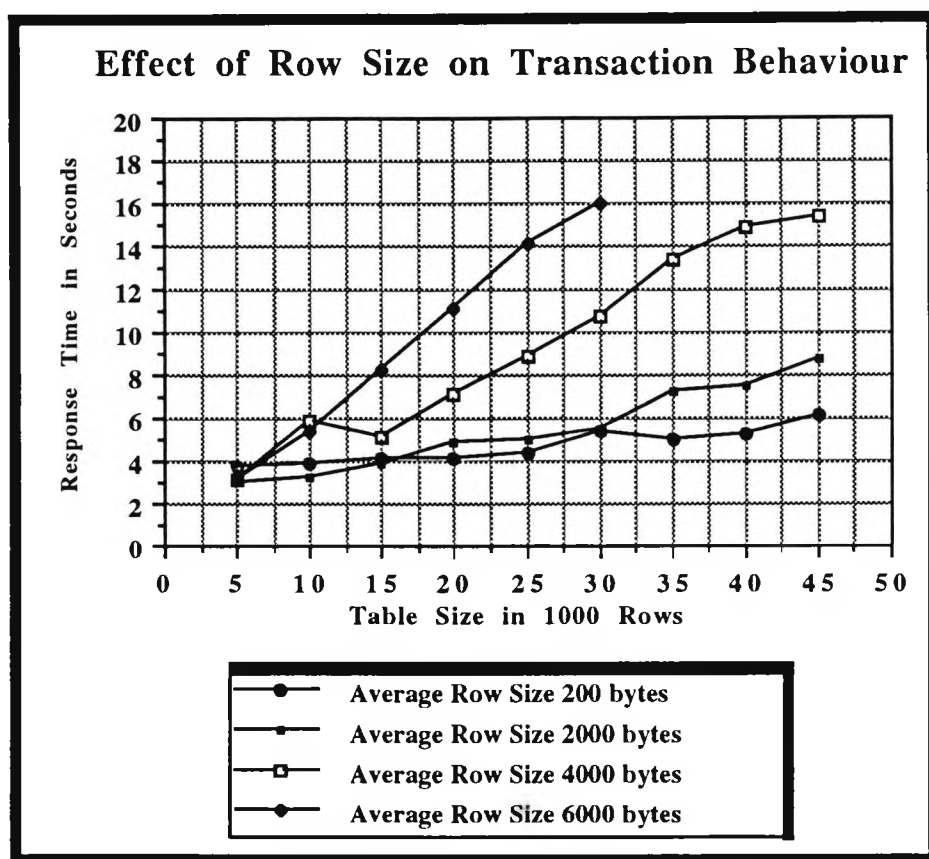


Fig. 3.7, Effect of row size on transaction response time

3.4.2 Row Size Effect

Row size is one of the factors that might have a tangible effect on database transaction response time. To test row size effect the same transaction mix was used with four row sizes, 200 bytes, 2000 bytes, 4000 bytes and 6000 bytes. The four row sizes effects were experimented using scaled up tables sizes ranging from 5000 rows per table to 50,000 rows per table. Fig. 3.7 shows that row size has a dramatic effect on transaction response time. If the 200 bytes line is taken as a base for comparison, when table size was increased to 30,000 rows, the 4000 bytes row size transaction took twice the response time of the 2000 bytes row size transaction and the 6000 bytes row size transaction took four times the response time of the 2000 bytes row size

transaction. Due to space limitation, the 6000 bytes table size could not be increased to 45,000 rows. At 45,000 both the 2000 bytes transaction response time and the 4000 bytes transaction response time showed significant deviation from the 200 bytes transaction. The difference between 200 bytes transaction response time and the 2000 bytes transaction was around three seconds, and between the 200 bytes transaction response time and the 4000 bytes transaction response time was around 10 seconds.

3.4.3 Database Indexed Attributes

An important feature is the accessibility of rows and whether it is through: indices; calculated addresses; or just sequential access. Indices provide a performance tool in database where performance is the primary requirement. Indices have always been employed to enhance systems performance, if a database is subject to frequent and numerous retrieves and joins, then placement of indices over the key attributes will enhance the overall performance of the database system.

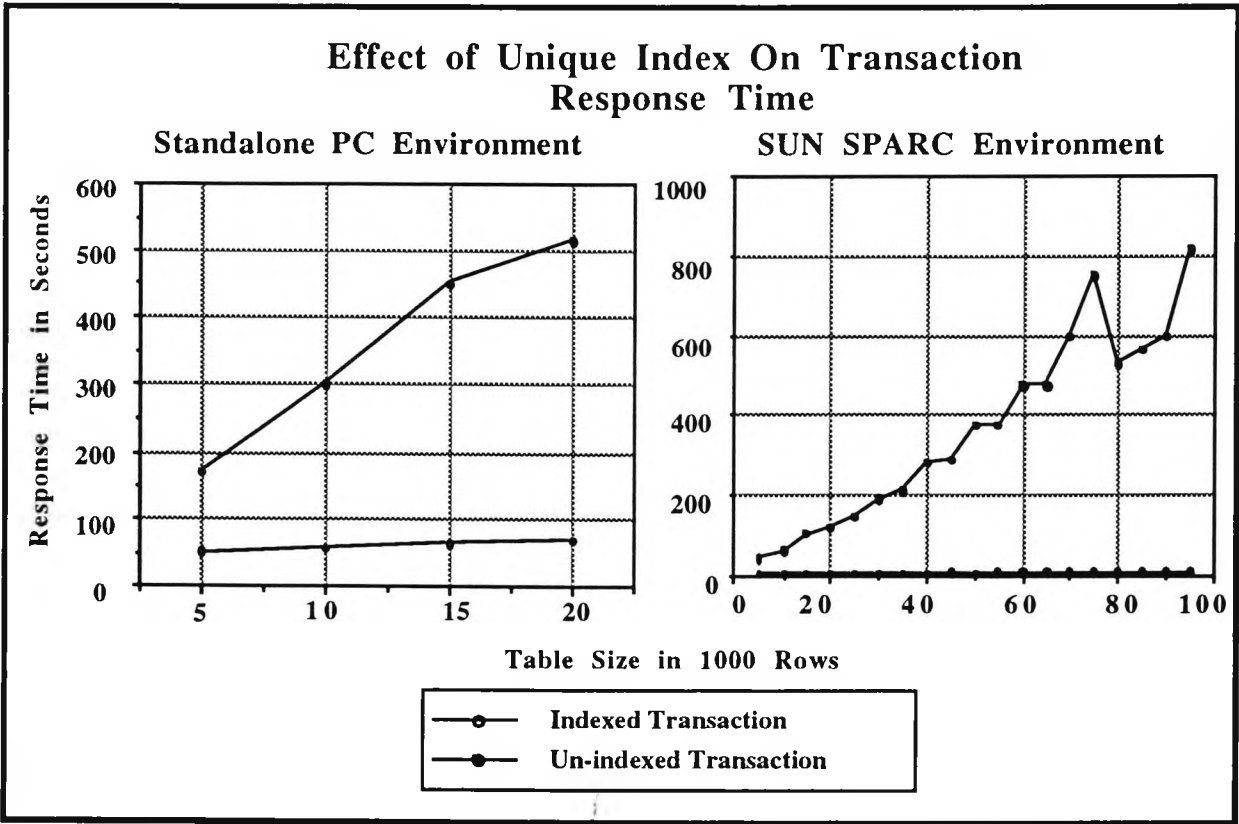


Fig. 3.8, Effect of unique index on transaction response time

The database indices effect on transaction response time was tested at two different environments. The test showed that utilisation of database indices dramatically

affected transaction response time. Fig. 3.8 shows the effect of using database unique index on transaction response time standalone PC environment. The test showed that as database size increases the effect of database index gain more importance. A not indexed transaction response time at the size of 20,000 records was almost ten times more than the indexed one.

Database indices' effect was more dramatic in the SUN SPARC environment. Fig. 3.8 shows the effect of using database unique index on transaction response time at SUN SPARC environment. The graph shows that while database indices' effect is minimal at smaller database size the indices' effect was quit tangible at larger sizes. At database size of 100,000 records a not indexed transaction response time was around 800 seconds, in the mean time an indexed transaction took only around 9 seconds.

3.4.4 Attributes Types and Distribution

Systems behave differently when using different attribute types. Turbyfill C., [TURB88] given an example of two systems, the first was a business system and the other was a scientific system. The business system supported packed decimal numbers and did not support floating point numbers; the scientific system supported floating point numbers but not packed decimal numbers. On queries using only data types that were available on both systems, the scientific system had response time that is 10 times faster than the business system. In addition, by representing the same numbers as floating point and packed decimal the arithmetic operations on the scientific systems were 10 times faster than on the business systems. The research tried to identify which attributes types are more prevalent at the studied environments because despite that difference some users still prefer to use packed decimal attributes to have more control on their arithmetic operations results.

3.4.5 Transaction Database Operations

Transaction database operations of the running systems are the most important point to study because it determines the main criteria when designing a benchmark script. One system may be excellent at performing one transaction type for on-line database and behave in a different manner when performing another transaction type. A example was presented in section (§3.2.3). In that example systems behaviour varied depending on the database operations in the transaction scripts. Consequently, benchmarks should be as accurate as possible in simulating different domains.

3.4.6 Transactions time Utilisation and I/O Operations

Some researchers classify on-line transactions into three classes of relational queries: overhead-intensive; data-intensive; and multi-relational Queries [HAWT79]. Others, [BITT83], adopt the resource utilisation approach to select query mix. They classified CPU and disk resources consumption into two classifications low consumption or high consumption. The Wisconsin benchmark adopted this idea and classified its transactions to: low CPU utilisation, low disk utilisation; low CPU utilisation, high disk utilisation; high CPU utilisation, low disk utilisation; high CPU utilisation, high disk utilisation. The research aimed to identify the actual nature of on-line transactions by examining the number of I/O per on-line transaction and calculating the ratios of CPU time to I/O time.

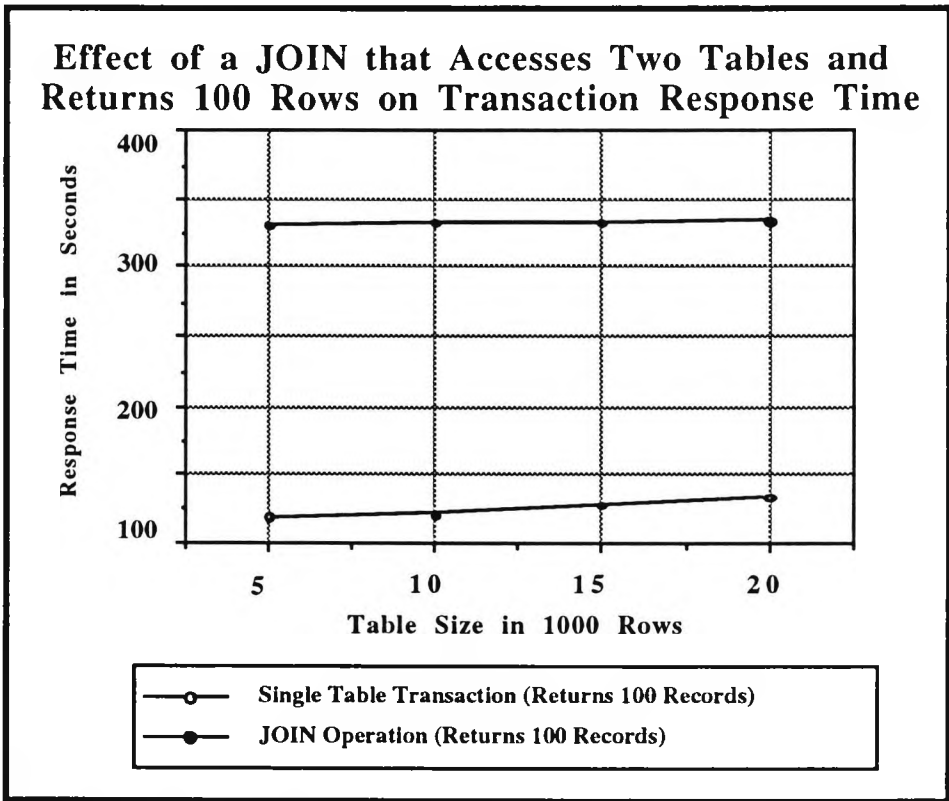


Fig. 3.9, Effect of JOIN operation on transaction response time

3.4.7 JOIN Operation

JOIN operation is a time consuming operation at database environments, hence DBMS adopt different strategies when executing them. The examination of JOIN operation took place because it puts extra pressure on the running systems and database performance varies dramatically when executing this operation. To JOIN operation

effect was tested by measuring the response time of a transaction that joins two 10 records from a tables to 100 records from another. The transaction used in comparison is a sequential retrieval transaction that access one table and return 100 records. Both the JOIN transaction and the sequential retrieval transaction used indexed key.

Fig. 3.9, shows a comparison between the two transactions' response time. The JOIN operation response time was about seven times higher than the single table transaction response time. The JOIN operation was studied to reveal its utilisation scheme at different database environments in relation to other database operations and to define the average number of tables accessed per a JOIN transaction.

3.4.8 Transaction Complexity

There are numerous ways to form the same query, but some forms of a transaction might be executed faster than the other to obtain the same information. Knowledge about these characteristics can achieve more rapid responses from the database. Transaction complexity was studied as the number of nested selects per transaction. An example is JOIN operation, where there are two ways of implementing JOIN transactions at on-line environment:

Transaction 1:

```
select      tickets from customer
           where tickets.fl_no = flights.fl_no
           and   flights.fl_no = in_flight
```

Transaction 2:

```
select customer
   from tickets
  where fl_no in
        (select fl_no from flights
         where fl_no = in_flight)
```

Database management systems performance varies between the two different forms. Therefore, some systems designers review user commands to provide guidance to users to exploit the rapid features. The research tried to identify which one of two forms is most common at the studied environments.

3.4.9 Transaction Output

There is a difficulty of finding the typical nature of transactions' output, the research tried to identify this nature by examining what happens in real life to that output and tried to investigate the following questions:

- how many rows are retrieved on average by one transaction?
- does a transaction retrieve the results of the query into a table?
- does a transaction retrieve the results of the query to a screen?

Each answer has a consequence on response time and in turn on the benchmark design. If a transaction retrieves the results of a query into a table, it will be measuring the time it takes to eliminate duplicates as well and if a transaction retrieves the results of the query to a screen, depending on the number of rows it retrieves per transaction and the time it takes to transfer the data to terminal, may swamp the time it took to execute a query. To demonstrate those effects, the research examined the effect of the number of retrieved records per transaction and the effect of sending the output to computer screen on transaction response time.

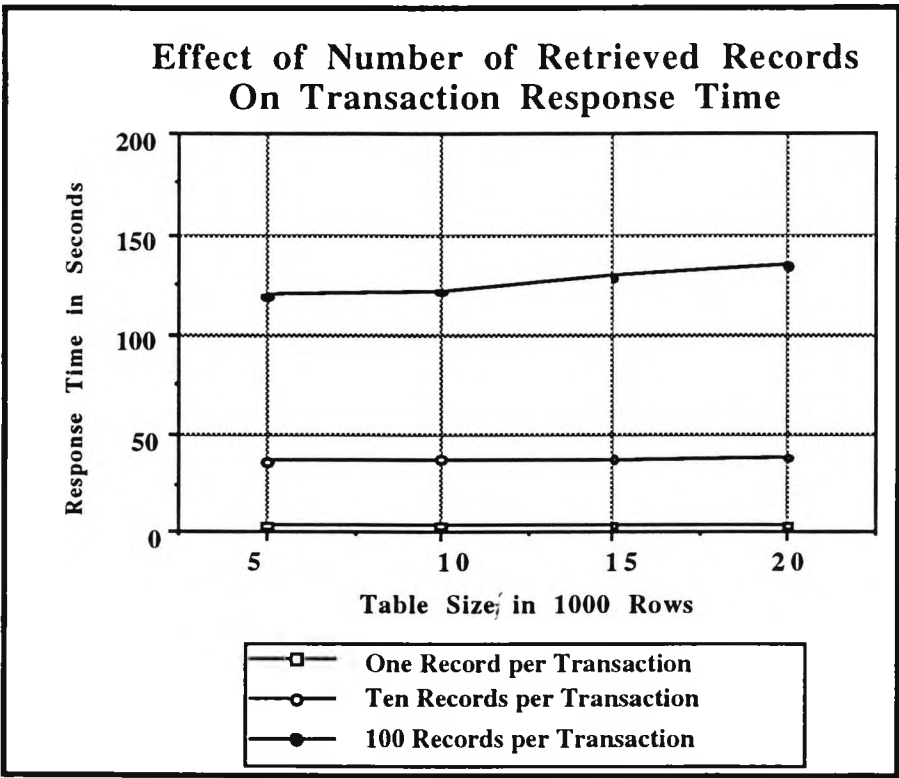


Fig. 3.10, Effect of transaction output on transaction response time

Fig. 3.10 shows the effect of retrieved records per transaction on transaction response time. It shows that when the transaction retrieved one record the response time was within three seconds. When the transaction retrieved ten records, the response time stepped to be around 45 seconds. When the transaction retrieved 100 records, the response time reached 125 seconds.

Fig. 3.11 shows the effect of sending transaction output to computer screen on transaction response time. The test used a program that sends 100 bytes of data to computer screen after each iteration. The program sent 20,000 bytes of data by the end of each transaction completion. Sending transactions' output to the computer screen was a constant value added to database operations' time. This added value in some cases, with smaller database sizes, was larger than response time itself. That might easily swamp the DBMS operation response time. The test presented below shows that directing transactions' output to computer screen consumed more time than DBMS operations and swamped the effect of the database test transaction.

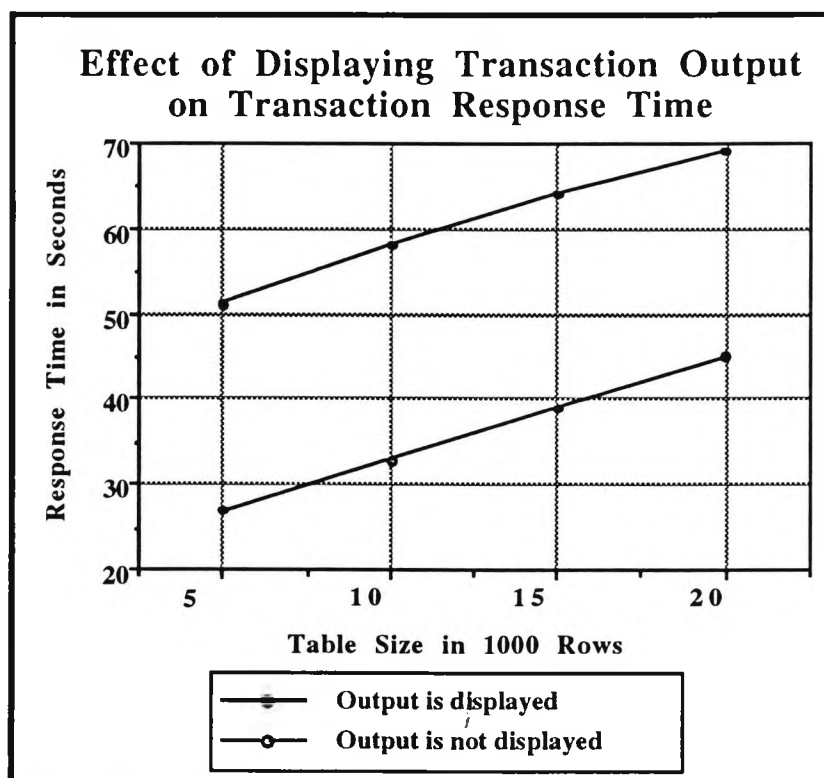


Fig. 3.11, Effect of result display on transaction response time

3.4.10 Background workload

Because it could be one of the most vague performance factors, database benchmark designers usually choose background workload on purely intuitive basis. Some benchmarkers use simple background workload that consists of simple selection transactions, such as: simple join with clustered index; simple aggregate with clustered index; and selection with non-clustered index, and run it repeatedly until completion of the benchmark run. Some other benchmarks concentrated on complex operations and varied the type of jobs in the background workload and chosen several background scripts, such as: complexes join; report generation; I/O bound operations (copy and sort); and CPU bound operations (complex calculations), in their benchmarks. Generally there are two types of background workloads:

1. The Database Workload
2. Non-Database Workload

This research aimed to identify the nature and the types of background work load at on-line environments. The non DBMS workload is considered because it creates bottlenecks in the DBMS as it interfere with DBMS processing. At any computer centre there are two types of background workloads: CPU bound workload, and I/O bound workload. Each workload uses different resources. The CPU bound workload performs mainly computations and requires very little or no I/O. The opposite is the I/O bound workload. This research defines background workload as any computer activities that are running in parallel with on-line service. Accordingly, transaction types such as: database batch transactions; DSS; and QMF transactions that run in the background, fall in this domain. That included some operations such as:

- copy operations;
- sort operations;
- systems back-up operations;
- systems restore operations;
- report generation;
- batch programs.

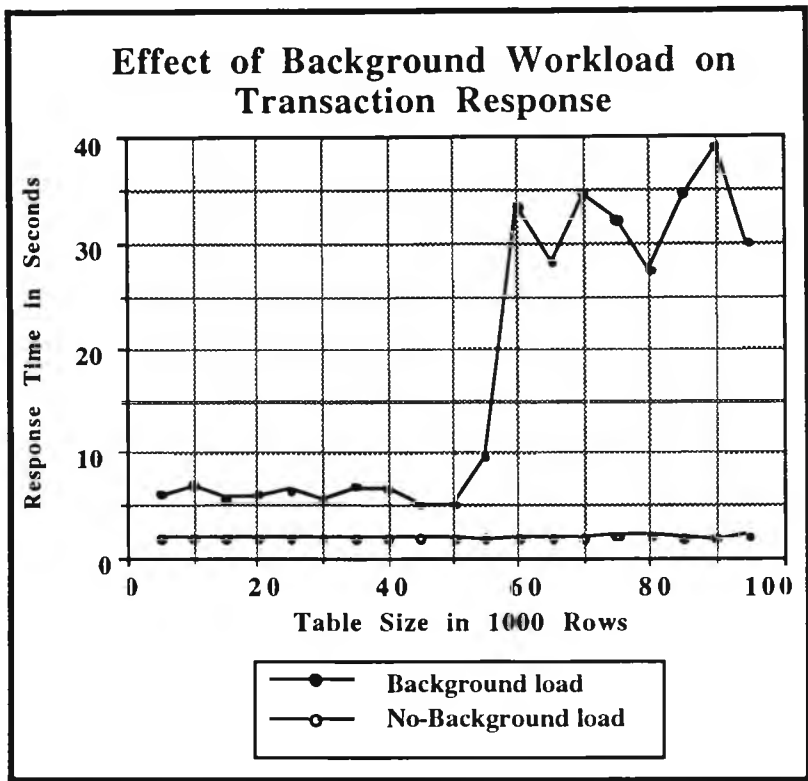


Fig. 3.12, Effect of background workload on transaction response time

Fig. 3.12 shows the result of running background load on transaction response time at SUN SPARC environment. The load consisted of continuous programs compilations. The tested database size was increasing and the background load was kept constant through all the duration of the test. As shown, as the database size increased, the effect of the background load on transactions' response times dramatically increased. The studies will try to identify the nature of background load at high-volume transactions' environments. The identification of background workload will allow a benchmark to simulate the impact of typical background workload on transaction response time.

3.4.11 Discussion of Performance Factors

Some other factors such as CPU utilisation for the various control modules and buffer management was described by Ferrari [FERR83]. Those factors were not covered by this work because they were either irrelevant or neutral to the tested benchmarks' measurements. For instance, because buffer size was fixed for all tests, its effect on different measurements could be assumed constant and discarded.

Having discussed the basis of selecting the domain environments and the factors affecting database performance, the following sections discuss data gathering techniques and the statistical methods used by this research.

3.5 Studying Organisations and Data Gathering Techniques

Searching for a proper technique to study transaction behaviour in OLTP environments it was found that many information systems researchers were concerned with using behavioural science techniques or a mix of behavioural science and other techniques to examine information systems properties. Examples of such are plentiful. They were conducted to study information systems behaviour or to gain better understanding of a certain problem.

Mckaskill [McKA78] examined the relationship between the functional staff, who use computer based information system, and the data processing staff, who are responsible for designing and managing these systems, in order to identify organisational characteristics, information systems characteristics, and management choice and personal factors. Zelkowitz [ZELK84], conducted a field study to identify discrepancies between the state of the art and the state of practice in using software engineering tools and methods. Card [CARD87], measured technology use in a production environment and evaluated the effects of some specific technologies on productivity and reliability. Then developed a statistical model for the effectiveness of those technologies. Curtis, from University of Texas, has relied heavily of field studies to study software design process for large systems; some of his work are presented in [CURT80, CURT85, CURT86, CURT88]. Ord [ORD 88], studied the cultural aspects due to the introduction of information technology in an urban health district. And Santos [SANT89], proposed a mixed methodological approaches to IS design based on empirical studied in an information centre. All the previous researcher studied organisations' behaviour to come to their conclusions. In those studies they conducted a mix of behavioural science techniques to collect the information required for their research.

This research has adopted the approach of studying systems in action (action research) to study the selected organisations. Action research techniques are based on interviews, questionnaires, observations, and document analysis to examine systems under study.

3.5.1 Approaches of Studying Organisations

Within the context of behavioural science techniques, there are wide range of approaches for understanding and learning about information systems. A good discussion of those approaches is presented by Silverman [SILV70] and Clark [CLAR72].

3.5.1.1 Pure Basic Research

This type of research arises out of the perceived requirements for the development of a basic discipline and is typically oriented towards resolving, illuminating or exemplifying a theoretical question. Typically, this type of research is at a high level of generality and the primary channel of diffusion is the learned journal. Hence the feedback of the results is not typically to the wider public or to potential users in organisations, but to the stock of knowledge of the scientific community. Consequently, the findings from such work may take some time before they penetrate the practice of professionals within enterprises or become part of the general stock of knowledge.

3.5.1.2 Basic Objective Research

This category of research is concerned with the taking of general problem that occurs in some field of the application of knowledge, but it does not aim to provide a prescription for a particular practical problem. The level of generality in this type of research is high and primary channels of dissemination are professional journals, the major line of feedback is to the professional community and associated academics.

3.5.1.3 Evaluation Research

Evaluation research tackles an existing and continuing practical problem within an enterprise, but it does not include a change strategy. It is distinguished by its methods and strategy which are at a high level of generality. The aim of evaluation research is to provide an assessment of some aspect of performance of an enterprise. Typically, evaluation research has included the assessment of the effectiveness of change programs both within enterprises, and in recent period, for the assessment of community based change programs designed to tackle several social problems.

Evaluation research is frequently and widely used and the results of such researches are typically of interest to the implicated parties. The channel of diffusion of these highly particularistic report is to sponsor. Occasionally, they may be published in professional journals because they illustrate a development in research technology, and additionally, there may be some discussion of the reports at the local level amongst interested evaluation researchers in their district branches of professional associations.

Evaluation research is the dominant type of research in the relationship of exchange between scientists and professional practitioners, and is also one which has recently been subjected to considerable criticism. Before considering these criticisms, it is relevant to restate the observation that evaluation research was the first major form of exchange between the scientific community and sponsors in enterprises.

3.5.1.4 Applied Research

Applied research is directed towards the solution of a practical problem arising within a specific system by the application of appropriate knowledge. It is undertaken by practitioners who are qualified in one or more of the problem area. Unlike evaluation research, there is no special reason why the definition of the problem would be automatically accepted as specifying the terms of reference. However, in practice it is quite likely that systems users will only identify the problems for which they believe there are solutions.

Case studies are published in professional journals. Many people in the academic world see the value of the applied work specially when they examine systems content and utilisation. Typically, the applied researchers will continue resolving immediate problems on the basis of existing knowledge in some cases, it might add to the total stock of knowledge. Recently, this methodology has been used for in research on software development projects. researchers like Buchanan, Benbasat and Swansen [BUCH82, BENB87, SWAN88], applied this approach to study software management and information systems.

3.5.1.5 Action Research

Action research has three task masters: the system under test, the practitioner, and the scientific community. The fact that these dissimilar groups are related for the duration of a project imposes many strains, and it must be observed that one of the

major problems facing the action researcher is the devising of appropriate administrative mechanisms for simultaneously linking and separating these groups.

Action research is one strategy for influencing the stock of knowledge of the tested enterprise. It uses data feedback in a cyclical process in order to assist in applied problem study and analysis. It also expands the domain of the existing knowledge in the area of study by offering visible solutions to the studied problems. It is based on questionnaires, system observation, interviews, and examine system documents. In that sense, it is a strategy for distributing knowledge, but the ideal project is one in which both the scientists and the tested system benefit through having a better understanding of a particular problem. Typically, the tested system are concerned to ensure that "get value for money spent", and hence not always especially concerned with the theoretical activities and interests of practitioners. Action research is also concerned with adding to the stock of knowledge of scientists and is said to be a very gaining "privileged access" to data and situations which are normally not easily accessible to the basic researcher. There it can make important contribution to the theory.

3.5.2 Data Gathering Techniques

The required information for this research included some low abstracted information such as on-line transaction basic database operations and ratios of I/O to CPU utilisation. Consequently, one single type of organisations study was not adequate on its own to serve this research purposes. That is due to their high level of generality. However, data gathering in this research was based on applying a mix of case analysis techniques. The main advantage of that approach is having a broad spectrum of techniques available to the researchers, ranging from highly generalised techniques such as, participation, observation and interviews to very low abstracted techniques such as low level systems monitoring. These techniques will be presented as separate entities, but in practice they might be combined. Most of these techniques are discussed at length in [McCA69, LOFL71, BOUC76, HAMM83 and VAN 82]. Those techniques can be summarised in the following:

1. Face to Face Co-Operative Interaction.
2. Interviews.
3. Informal Conversations.
4. Formal Meetings.
5. Convivial Meetings.

6. Ritual Meeting and Celebrations.
7. Demonstrations.
8. Shadowing.
9. Systems Documents study.
10. Systems Monitoring.

Data gathering for this research did not require the application of all the previous techniques. Only some of those techniques were sufficient to satisfy the research objectives. Data gathering techniques in this research were the following:

1. Formal meetings.
2. Conducting several interviews with key persons in the computer centre representing different organisational levels.
3. Studying the running systems' documents.
4. Studying computer system outputs.
5. Monitoring the systems over a representative period.

3.5.2.1 Formal Meetings

In every organisation at least one formal meeting took place to plan the required research work in that organisation. But generally, at least two formal meetings were required, the first was to explain the research objective, the research plan, and how it might contribute to the organisation by the end of it. The second usually took place after the organisation agreement to start the research study. In that meeting they usually plan and explain the rules the research has to comply to. Those meeting were usually a logical introduction the interviews.

3.5.2.2 Interviews

In this research interviews and attending meetings could nor be separated from each other. Interviews were very carefully planned because they were the means of making the first contact with people in organisations. Before the first interview, the organisational structure of each organisation was examined to target key people in the selected organisations. The first interview was always conducted with the one on the top of the organisational structure. That granted permission to follow up with people lower in the organisational hierarchy.

The interviews were kept as formal as possible from the beginning. From my personal experience, tape recorders are found to be rather intrusive instrument, hence interviews were not tape recorded. But notes were taken through the conversation. Formal and introductory interviews were usually pre-arranged to take place in the interviewees office or place of work.

Whenever possible, some informal interviews were conducted with technical staff responsible for the design and implementation of the studied organisations database management system. That helped in solving several technical problems and in interpreting some of the computer output results.

3.5.2.3 Studying Systems' Documents

Some information were difficult to get through interviews or systems monitoring. Those information included some important performance factors such as transaction complexity and number of databases accessed by a single transaction. The only way to get such information was by examining the running systems documents from the design stage. System documents contained system specification and design rules for both applications and databases. Those documentation included in details some important information such as how many databases will be accessed by each transactions and how many selects in a nested select. Studying systems' documents was the source of information such as: number of databases accessed by a single transaction (JOIN operation); database record size; number of indexed attributes; attributes' values; distribution of attributes in the database; and number of nested selects in a single transaction (transaction complexity). Such information were impossible to get using any other technique. The interpretation of those documents required the help the database administrators in the studied organisations.

3.5.2.4 Studying Systems' Outputs

The other part of information that were difficult to get were obtained by examining the running systems' output. System outputs included information such as patterns of on-line system load, available disk space, and ratios of systems utilisation. Most of systems output or collected in disk spaces then treated by one statistical method or another to minimise the size of printouts. The only problem with systems' output was the large size of the printed material. Despite the first statistical analysis phase, the

amount of printed material were extremely large and needed another phase of treatment until a reasonable output size was obtained.

3.5.2.5 Systems Monitoring

The technique that contributed most to this research was monitoring the studied systems over a representative period. Most of the important findings in this research are based on systems monitoring. For each one of the studied systems all the running applications were monitored for a representative period of time. All transactions issued to different systems have been collected and examined. The operational systems in the first organisation were monitored over twenty working days. For the second organisation the operational systems were monitored for fifteen working days 24-hours a day. Due to the extremely large size of the third organisation and the high costs of observing such a system for a long time, the running systems in this organisation were monitored for eight days 24-hours a day.

Monitoring systems is extremely expensive process but was adopted because it is the most credible way to collect representative data about the observed systems.

3.6 Statistical Techniques

Statistical techniques can be used to fit a model of performance or test a hypothesis about the factor that affect performance. The type of analysis depends on the factors involved. The following sections present some important definitions and discuss this research data analysis techniques. Those techniques are discussed in details by Ostle [OSTL88], Hoel [HOEL82] and Wonnacott [WONN85].

3.6.1 Some Definitions

3.6.1.1 Arithmetic Mean

If there exist n sample values that have been obtained from population n . These values can be denoted by $X_1, X_2, \dots, X_n$. This implies that X_1 is the first value and X_n , is the last value. Those sample values can be written as X_i , where $i = 1 \rightarrow n$. The arithmetic mean of those values:

$$\mu = (X_1 + X_2 + \dots + X_n) / n$$

This formula can be written in a neater form as:

$$\mu = \sum_{i=1}^n x_i / n$$

3.6.1.2 Sample Variance

Sample variance is a quantity that measures the extent to which a set of measurements varies about their mean. In other words it is a measure of values spread around the sample mean. The smaller the variance the less the spread from the mean. An example can be given by two samples:

sample 1 values: 4,5,6

sample 2 values: 1,5,9

The arithmetic mean of both samples is 5. Since the difference between consecutive pairs of measurements in the second set is four times that is in the first set, one could say that the second sample varies four times as much around its mean as does the first sample.

Sample variance is calculated as follows:

$$s^2 = \sum_{i=1}^n (x_i - \bar{x})^2 / n - 1$$

3.6.1.3 Standard Deviation

Since the variance involves the squares of deviations, it is a number in squared units. In many cases it is desirable that quantities describing a distribution possess the same units as the original set of measurements. The mean satisfies this requirement, but the variance does not; however, by taking the positive square root of the variance the desired effect can be achieved. The resulting quantity is called the standard deviation. Standard deviation is calculated as follows:

$$s = \sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 / n - 1}$$

3.6.1.4 Number of Measurements for Reliable Sample

The reliability of the benchmark results is related to the number of measurements taken from that benchmark. Usually, several measurements are required to have a reliable measurement and a single measurement can never be considered as reliable index. The question is the size of adequate sample will always be controversial. One way to find adequate sample size is using the Central Limit Theorem.

Theorem 1

"if random samples, size n , to be taken from a distribution with mean μ and standard deviation s , then the sample means would form a distribution having the same μ but with a smaller standard deviation given by $s / \sqrt{n}$ "

The quantity $s / \sqrt{n}$ is often called the standard error of X to distinguish it from the standard deviation, s , of the original distribution. As n increases, the standard error decreases.

Theorem 2

"if random samples, size n , are taken from a normal distribution, the sampling distribution of μ will also be normal, with the same mean and standard error given by $s / \sqrt{n}$ "

When reasonably large samples are taken ($n > 30$), the sampling distribution of μ will be approximately normal whatever the distribution of the parent population. This remarkable result known as the central limit theorem. Consequently, if the measurements from a benchmark transaction are based on random requests, the benchmark results could be treated as a random sample the central limit theorem could be applied on the results and the measurements from a benchmark job represent a sample from an unknown distribution of elapsed times could be treated as normally distributed samples if the sample size is large enough.

3.6.1.5 The Normal Distribution

The normal distribution is the most important distribution of all distributions since it has a wide range of practical applications. The normal distribution is a mathematical model which adequately describes most natural phenomena. The height of the normal probability curve is given by:

$$f(x) = (1 / \sqrt{2\pi}\sigma) e^{-((x-\mu)^2)/2\sigma^2} \quad -\infty < x < +\infty$$

where μ , σ are parameters such that $-\infty < x < +\infty$ and $\sigma > 0$.

The total area under the normal curve is equal to one for all values of μ and σ .

The importance of the normal distribution can be summarised in the following:

1. Many physical measurements are closely approximated by the normal curve.
2. Most physical phenomena are normally distributed, even when some physical phenomena are not normally distributed, they can be easily transformed to normality.
3. Any random variable formed by taking a linear combination of independent normally distributed random variables will itself be normally distributed. This property can be applied to any random measurements.

3.6.2 Design of Experiments

The design of experiments is one of the most important areas in an empirical research. It is important to any experimenter to guarantee that his/her experiments results are free from uncontrolled errors or residual variations. Also it is important for any experiment to separate any systematic errors in the experiment results. In this research are several techniques that can minimise systematic errors and increase the precision of experiments results were applied. Those techniques were:

- replication of experiments;
- randomisation of experiments' events;
- analysis of experiments' results covariance.

3.6.2.1 Replication of Experiments

The number of observations belong to the same experiment is called number of replications. The more the number of observations the more accurate the result from the experiment. In comparative experiments it is essential to obtain more than one observation to estimate the experiment results. However, it is difficult to decide the best number of replications for a specific experiments. Generally, the larger the number of replications, the smaller will be the standard error of the difference between two treatments means.

3.6.2.2 Randomisation of Experiments' Events

Randomisation of experiments' events is a simple but powerful way to ensure that the experiment results are as free as possible from systematic errors. In computer measurements, there are a substantial amount of uncontrolled variation. That variation prevents the exact replication of the same experiment. Those variations are due to factors that change with the experiment conditions and in many cases they cause deviation in the experiment results apart from the effects tested by the experiment. The best way to eliminate the effect of those factors is by randomising the experiment events.

3.6.2.3 Analysis of Covariance

The techniques described in the following sections are useful whenever the effects being investigated are liable to be masked by experimental variation which outside the control of the experimenter. For completely controlled experiments, these techniques might have little value since all variables are controlled within minimum variance. However, for computer measurements complete control is impossible and there will always exist considerable variation in the taken measurements. The following statistical methods will separate the effect of the uncontrolled and residual variance. Additionally, they will ensure that there is no systematic error.

3.6.2.3.1 One-Way Analysis of Variance

One way analysis of variance is a method that tests the hypothesis that there is no difference between a number of columns. The total variation of the observations is partitioned into two components:

- the variability between the group means;
- the variation within each group.

These two components are compared by means of an F-test. The one-way analysis of variance divides the experiment observations into mutually exclusive categories.

If the null hypothesis is true the mean square between columns divided by the mean square within columns will follow F-distribution with $c-1$ and $c(n-1)$ degrees of freedom.

If the null hypothesis is not true, the mean square between columns will be increased by the columns and F-ratio will be significantly large. In this case the null hypothesis should be rejected as it indicates a significant difference between the different columns.

One way analysis of variance is calculated as follows:

$$SS_{\text{tot}} = SS_{\text{cols}} + SS_{\text{res}}$$

$$\text{Source of variance between columns } SS_{\text{cols}} = r \sum_{i=1}^c (\bar{X}_i - \bar{\bar{X}})^2 \quad (1)$$

with $c-1$ degrees of freedom.

$$\text{Mean square between columns } (MS_{\text{cols}}) = r \sum_{i=1}^c (\bar{X}_i - \bar{\bar{X}})^2 / (c-1) \quad (2)$$

$$\text{Source of variance within columns } SS_{\text{res}} = \sum_{i=1}^c \sum_{j=1}^r (x_{ij} - \bar{X}_i)^2 \quad (3)$$

with $c(n-1)$ degrees of freedom.

$$\text{Mean square within columns } (MS_{\text{res}}) = \sum_{i=1}^c \sum_{j=1}^r (x_{ij} - \bar{X}_i)^2 / c(r-1) \quad (4)$$

$$F = (MS_{\text{cols}}) / (MS_{\text{res}}) \quad (5)$$

For insignificant difference between observations, the calculated ratio of F test should be less than or equal to the F value equivalent to number of degrees of freedom $(rc-1)$ in the F table.

3.6.2.3.2 Two-Way Analysis of Variance

In a randomised block experiment, the data is classified according to two characteristics in a two way table. The main problem is to see if there is a significant difference between the observed values. The null hypothesis in a two-way analysis of variance is:

$$H_0 : \text{all } V_j = 0, j = 1 \text{ to } c$$

The total corrected sum of squares is given by $\sum_{i=1}^c \sum_{j=1}^r (x_{ij} - \bar{\bar{X}})^2$. That sum could be split up into three components, one that measures the variance between columns, one measures the variance between rows, and the third measures the residual variation. The columns variation is then compared with the residual variation by means of F-test. Since the data is classified according to two characteristics, this method is called two-way analysis of variance. The algebraic identity of the two-way analysis of variance is as follows:

$$\sum_{i=1}^c \sum_{j=1}^r (x_{ij} - \bar{\bar{X}})^2 = r \sum_{i=1}^c (\bar{X}_i - \bar{\bar{X}})^2 + c \sum_{j=1}^r (\bar{X}_j - \bar{\bar{X}})^2 + \sum_{i=1}^c \sum_{j=1}^r (x_{ij} - \bar{X}_i - \bar{X}_j + \bar{\bar{X}})^2 \quad (6)$$

where:

Source of Variance due to difference between column means:

$$SS_{\text{cols}} = r \sum_{i=1}^c (\bar{X}_i - \bar{\bar{X}})^2 \quad (7)$$

Source of Variance due to difference between row means:

$$SS_{\text{rows}} = c \sum_{j=1}^r (\bar{X}_j - \bar{\bar{X}})^2 \quad (8)$$

Residual:

$$SS_{\text{res}} = \sum_{i=1}^c \sum_{j=1}^r (x_{ij} - \bar{X}_i - \bar{X}_j + \bar{\bar{X}})^2 \quad (9)$$

Where:

r: number of rows

c: number of columns
j: rows index
i: columns index

The first component on the right hand side of this equation measures the rows variation. The second component measures the columns variation, and the third component measures the residual variation. In a more simplified way it the previous formula can be written as:

$SS_{tot} = SS_{rows} + SS_{cols} + SS_{res}$ (10)

Mean square between columns $MS_{cols} = r \sum_{i=1}^c (\bar{X}_i - \bar{\bar{X}})^2 / (c - 1)$ (11)

Mean square between rows $MS_{rows} = c \sum_{j=1}^r (\bar{X}_j - \bar{\bar{X}})^2 / (r - 1)$ (12)

Mean square of residual $MS_{res} = \sum_{i=1}^c \sum_{j=1}^r (x_{ij} - \bar{X}_i - \bar{X}_j + \bar{\bar{X}})^2 / (c - 1)(r - 1)$ (13)

Where:

$F_{cols} = MS_{cols} / MS_{res}$
 $F_{rows} = MS_{rows} / MS_{res}$

The same rule applies for the two way analysis of variance, for insignificant difference between observations, the calculated ratio of F test should be less than or equal to the F value equivalent to number of degrees of freedom (rc-1) in the F table.

The two-way analysis of variance can be summarised in table 3.5.

Source of Variance	Variation Sum of Squares (SS)	Degrees of Freedom	Mean of Squares (MS)	Calculated F
Between Rows	$SS_{rows} = c \sum (x_i - \chi)^2$	r-1	$SS_{rows} / r-1$	MS_{rows}/SS_{res}
Between Columns	$SS_{cols} = r \sum (x_j - \chi)^2$	c-1	$SS_{cols} / c-1$	MS_{cols}/SS_{res}
Residual	$SS_{res} = \sum (x_{ij} - \chi_i - x_j - \chi)^2$	(r-1)*(c-1)	$SS_{res}/(r-1)(c-1)$	
Total	$\sum (x_{ij} - \chi)^2$	cr-1		

Table 3.5, Steps of calculating ANOVA

If H_0 is true, the results then has an **F** distribution. In this case the calculated **F** ratio should be smaller than the **F** table values. If the calculated **F** ratio lies beyond the **F** table value, then it indicates a strong evidence against H_0 and the null hypothesis should be rejected.

3.6.3 Comparing The Workloads Of Different Transactions

Comparing the behaviour of the workloads of different transactions posed a problem for this research because the review of literature could not find a well-accepted comparison technique that could be relied on for such comparisons. This thesis proposes the use of the rate of change of DBMS behaviour against a test parameter as a common basis of comparison between different transaction workloads. The hypothesis is based on the relationship between the rate of change of a specific workload and the degree of load it applies on the tested parameters, which implies less test time and cost.

Usually there are three factors controlling the quality of any test, the first is the fulfilment of the test to its objectives, the second is the time required to achieve that objective and the third is the ability of the test parameters to examine the tested variables. Generally, in comparing different testing procedures, the more they affect the tested parameters and the faster they achieve their objectives the better they are because that implies less test time and consequently less test cost.

Because the main function of any benchmark is to strain the tested DBMS, this research proposes the use of the rates of change of a specific benchmark workload as an indicator to the level of that strain which indicates the goodness of a benchmark workload. Statistically, the rate of change is measured by the estimate of the parameters of the function under consideration, which is practically estimating the rate of the response of the dependent variable for any changes in the corresponding independent variable. It is well established that a higher value of rate of change is an indicator to an acceleration in the response of the dependent variable. Based on that, the rate of change can be used to measure the quality of the workload of a specific transaction in comparison to any other transaction. The transaction workload with a higher value of rate of change of its slope indicates that the test workload is applying significant testing load on the variables under test which in turn indicates faster test speed, and as a consequence less time and cost to achieve the test objectives.

Finally, there is always a subtle compromise between the rate of change and the cost involved in expanding the test parameters. This compromise is the main difference between an acceptable transaction workload and an unacceptable one.

Generally, this point of using the rate of change as a quality factor between different transactions is presented in [REVE94c] to raise a question for future research in the techniques that are statistically solid and could be used to compare the workloads of different transactions.

For readability purposes, this thesis refers to difference quotient by the term "*scalability level*", and the relative rate of change by the term "*scalability ratio*".

3.6.3.1 Calculating Average Rate of Change (Difference Quotient)

The slope of a straight line can be determined by applying the two point formula. The two point formula is as follows:

$$m = \frac{\Delta y}{\Delta x}$$
$$\Delta y = y_2 - y_1$$
$$\Delta x = x_2 - x_1$$

With linear functions the slope is constant over the domain of the function. The slope provides an exact measure of the rate of change in the value of y with respect to a change in the value of x . The slope indicates the rate at which total cost increases with respect to change in the level of output.

With non-linear functions, the rate of change in the value of y with respect to a change in x is not constant. However, one way of partially describing non-linear functions is by the average rate of change over some interval. In moving from point A to point B, the change in value of x is Δx . The associated change in the value of y is:

$$\Delta y = f(x + \Delta x) - f(x).$$

And the ratio of these changes is:

$$\frac{\Delta y}{\Delta x} = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

The rate of change of a smooth, continues function can be represented geometrically by the slope of the line drawn tangent to the curve at the point of interest.

3.6.3.2 Calculating Relative Change

One of the problems this research has encountered is the need to compare the rate of change of two functions that widely differ in cost. An example can be given by two functions $f^a(x)$ and $f^b(x)$. The values are presented in table 3.6.

	$f(x_{i+1})$	$f(x_i)$	Δy_i
$f^a(x)$	300.6	300	0.5
$f^b(x)$	1.5	1.0	0.5

Table 3.6, Example of two transactions' workloads

In this example the slope is the same in both cases and does not give an indication about which function is changing faster than the other. A better index is the relative rate of change. The relative rate of change is calculated as follows:

$$\text{Relative change } (R_i) = \Delta y_i / y_i$$

When the relative rate of change is used in the previous example it will give the following results:

$$\begin{aligned} R_1 &= (300.6 - 300)/300 \\ &= 0.6/300 \\ &= 2E-3 \\ R_2 &= (1.6 - 1)/1 \\ &= 0.6/1 \\ &= 6E-1 \end{aligned}$$

Which shows that $f^b(x)$ has a better relative change rate than $f^a(x)$, despite having the same slope value.

3.6.4 Ordinary Least Squares Method (OLS)

The objective of this method is to fit a line to the data in order to predict the mean value of the dependent variable for a given value of the controlled variable. The linear equation is of the form:

$$y = a + bx + \varepsilon$$

Where:

y is the dependent variable;

x is the controlled variable;

and ε is a random error term.

This method fits a line to n pairs of measurements $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$.

Ordinary square estimate of the slope b is calculated from the following formula:

$$b = \frac{\sum xy}{\sum x^2}$$

Where:

$$x \equiv X - \bar{X}$$

$$y \equiv Y - \bar{Y}$$

The correlation coefficient, R^2 , is an indicator of the goodness of fit of the line.

The correlation coefficient is calculated from the following equation:

$$R^2 = \frac{S_{xy}^2}{S_{xx}S_{yy}}$$

Where:

$$S_{xx} = \sum (x - \bar{x})^2$$

$$S_{yy} = \sum (y - \bar{y})^2$$

$$S_{xy} = \sum (x - \bar{x})(y - \bar{y})$$

R^2 varies between +1 and -1. The bigger the value of R^2 the better the fit of the line.

3.7 Summary

This chapter defined one of the major problems in the database industry as the reliance of the database industry on the TPC benchmarks, TPC-A and TPC-B. Almost every database vendor in the market is using the TPC metric to market their database management systems and the database industry has centred itself around the TPC transactions. As presented in this chapter, the TPC benchmarks suffer serious technical limitations. Those limitations are the way the benchmarks are implemented and the simplicity of the benchmark script. that affected their ability to realistically compare database systems.

As benchmark results are representative of those types of transactions actually included in the benchmark set. It is impossible to generalise those results to all kinds of systems transactions.

The goal is to create a comprehensive benchmark with increasing number of users that approximates the behaviour of typical DBMS applications and takes into consideration all the limitation discussed in the previous sections. A benchmark methodology for database systems must consider a wide variety of systems variables in order to fully evaluate performance. To achieve realistic metrics, the database system must be tested with a load as close as possible to that which it will be running in real life. Some idea of the type of information contributing to this achievement are: identify the dominant processing tasks, what queries will be run, what is the relative frequency of each query, what is the size of the database, and what patterns of behaviour are expected. To this end, the present author developed a series of studies based on.

1. Non synthetic data.
2. Database model independent.
3. Realistic data structure.
4. Typical transaction types.

CHAPTER 4

DATABASE CHARACTERISTICS AND TRANSACTION BEHAVIOUR IN LARGE DATABASE ENVIRONMENTS

CHAPTER 4

DATABASE CHARACTERISTICS AND TRANSACTION BEHAVIOUR IN LARGE DATABASE ENVIRONMENTS

The TPC benchmarks have become the standard practice in the database industry. Due to lack of background studies, the TPC benchmarks have sever technical limitations. Those limitations were discussed in chapter three. This chapter sets the empirical basis for the CITY benchmark design. It presents the results of in depth studies at three large UK database environments. These studies aimed to identify the salient characteristics of large databases at large database environments and to identify a typical pattern of behaviour of on-line transactions at high-volume transactions' environments. The data from that identification can supplement the data from the live environments once the basis of this analysis has been defined to evaluate performance in a transaction processing environment.

4.1 Introduction

Building an appropriate benchmark for large database environments is a complex process and depends on accurate information about the required domain environment. The success or failure of a benchmark depends largely on the clear identification of the characteristics of the domain to be tested. Some ideas of the type of information contributing to this achievement are: identifying the dominant processing tasks; identifying the queries to be run; identifying the relative frequency of each query; identifying the size of databases; and identifying the patterns of behaviour to be expected.

Most existing benchmarks suffer from one problem or another because they provide some estimate of the systems performance but do not provide a realistic characterisation of the database management system under test. This research conducted three field studies at three of the largest organisations in the UK. The organisations were selected because of being good examples of the domain of high-volume transaction environments. They were characterised by their large number of

applications, the wide variety of those applications, and their large database environments.

The first study investigated the running environment at a large local authority's computer centre. It offers information technology services and software consultants to four London boroughs. The second took place at a large airline computer centre that is running one of the largest database environments in UK. The third study investigated the running environment at a large UK bank computer centre. The studies aimed to achieve the following objectives.

1. Identify the main characteristics of on-line databases in high-volume transactions environment.
2. Identify a typical transaction behaviour based on a large number of real database applications.
3. Compare the findings to the TPC benchmarks.

Previous writers in this field [FERR78, FERR83, DONG87] have suggested some factors that determine the characteristics of on-line database and identify the behaviour of on-line transaction. The factors that determine the characteristics of on-line database are the following:

- database size;
- record size;
- attributes types;
- number of indexed attributes;
- types of attributes;
- distribution of attributes.

And following factors are to identify the behaviour of on-line transaction:

- the average number of basic database operations resulted from one transaction;
- the average ratio of CPU to I/O time utilisation;
- the average number of I/O operations resulted from one transaction;
- the average number of sort operations resulted from one transaction;
- the average number of databases accessed by one transaction (JOIN Operation);
- the average number of nested selects in one transaction (transactions complexity).

Monitoring the running systems over a representative time was the main data collection technique; the studies collected all on-line transaction over the period of study. Transactions' behaviour analysis included breaking down the collected transactions to their basic database operations, "qualified retrieval, insert a record, update a record, delete a record, and sequential retrieval", and the average number of each operation was calculated to determine a typical script of on-line transactions. In addition, the studies included the examination of systems output and documents and conducted several in depth interviews with key people representing different levels of management at the studied organisations. Systems documentation covered areas such as: database size; records size; types of attributes; number of indexed attributes per database; and attributes' distribution.

Section 4.2 presents the result of a study at the local authority computer centre. As the activities of local authorities might be unknown to many people, this section briefly reviews those activities and discusses the main applications that identify the nature of this environment. This section describes the following subjects:

- business of the local authority;
- organisational structure;
- data gathering technique;
- processing environment;
- running applications;
- main characteristics of the local authority application databases;
- transactions behaviour of the local authority running systems;
- background workload at the local authority environment.

Section 4.3 presents the result of the second study of this research. It took place at a large UK airlines' computer centre. This section discusses:

- data gathering technique;
- the airlines processing environment;
- main characteristics of the application databases;
- transactions behaviour of the airlines on-line systems;
- the airlines on-line system background workload.

Section 4.4 presents the result of a study at a large UK bank computer centre. This section discusses:

- data gathering technique;

- the bank processing environment;
- the main characteristics of the application databases of the bank;
- transactions behaviour of the bank on-line systems;
- the background workload at the bank on-line environment.

Section 4.5, is chapter four discussion. It compares the findings from the three studies and demonstrates the common characteristics among the three environments. The three organisations were similar concerning the database characteristics and the on-line transaction behaviour despite the difference in activities among the three organisations. This section also compares the findings from the three studies to the TPC benchmarks, it shows how those benchmarks are widely different from the behaviour of the presented environments and explain why those benchmarks can be misleading if used to test similar environments.

Section 4.6, present the chapter conclusion and briefly discusses the criteria for building the CITY benchmark.

4.2 A Study In a Large Local Authority Computer Centre

4.2.1 Introduction

This section presents the results of a study at a large local authority computer. This organisation was selected because of its large number of running applications and the large sizes of its database environment. The span of this study covered thirty working days, with each being investigated from 8:00 to 18:00 (the duration of on-line service). The local authority collected all on-line transactions issued to the running systems during the period of study.

4.2.2 The Business of the Local Authority

The local authority computer centre is an information technology services organisation. It was formed to offer information technology services and software consultants to four London boroughs. The services to those boroughs include: consultants; development; implementation and management of business and technical systems; telecommunications and training. The local authority recovers the service cost from its members with no profit.

The main factor that affected the local authority environment and applications was having to respond to changes in local government services as imposed by legislation. In this respect, the computer centre was a dynamic environment where new systems and amendments to the running systems were requested every time legislation changed, Fig. 4.1 presents a diagram of the business of local authority.

Each one of the boroughs has its own income sources. These income sources could be: community charge; non domestic rates; housing rents; social amenities; and central government grants. The income was spent on the borough services such as: precepting authority (police, fire, ambulance, etc.); Transport; housing; social services; education; roads; refuse collection & cleaning; planning and technical services; electors; and social amenities (e.g. Libraries, community centres, swimming pools, public conveniences, etc.). Each one of the previous income sources as well as spending activities was represented at the local authority centre to give each borough a daily image of its financial situation. Also it helps the boroughs to keep track of the services offered by each borough to its local community.

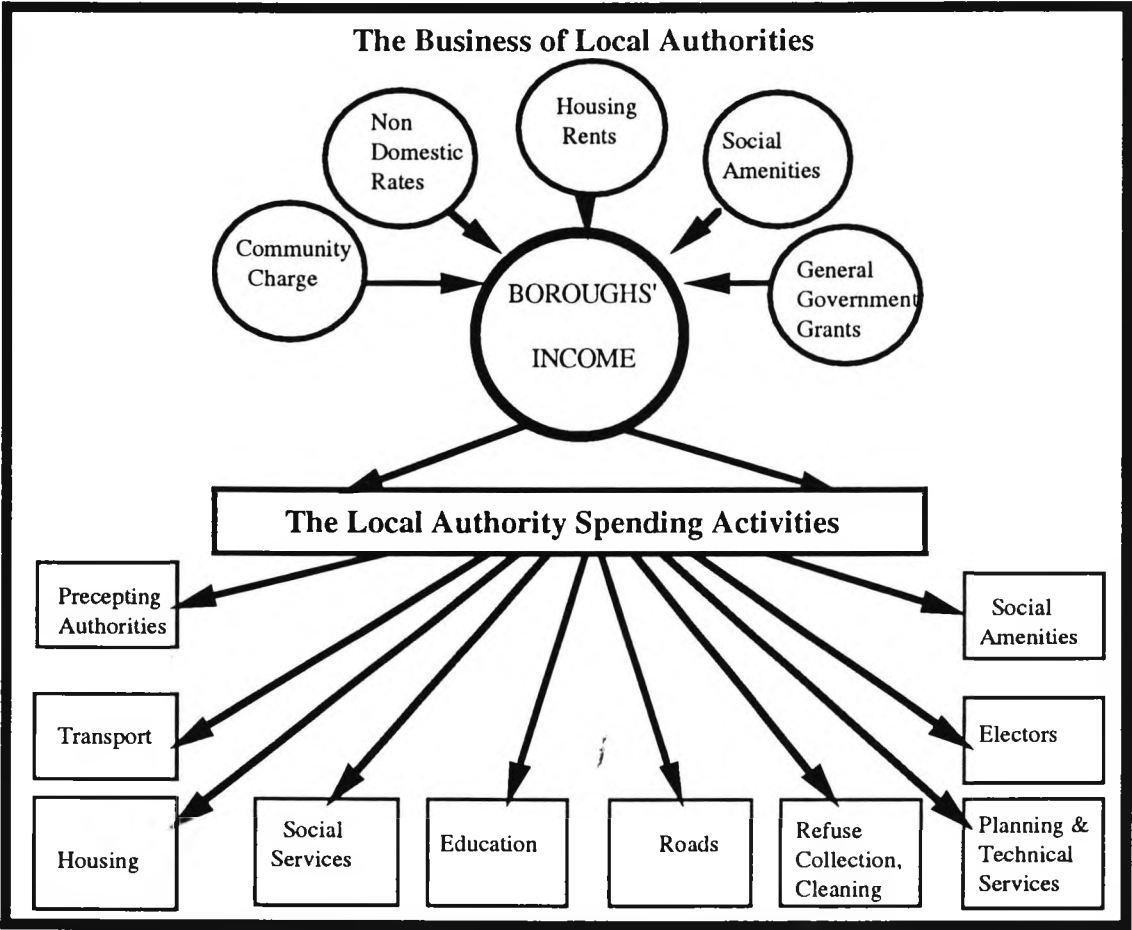


Fig. 4.1, The business of the Local Authority

4.2.3 Organisational Structure of the Local Authority

Levels of management and job description were examined at the local authority organisational structure to select the right people for the interviews. The research interviewed the staff who deals with databases or makes decisions regarding the running databases' environment.

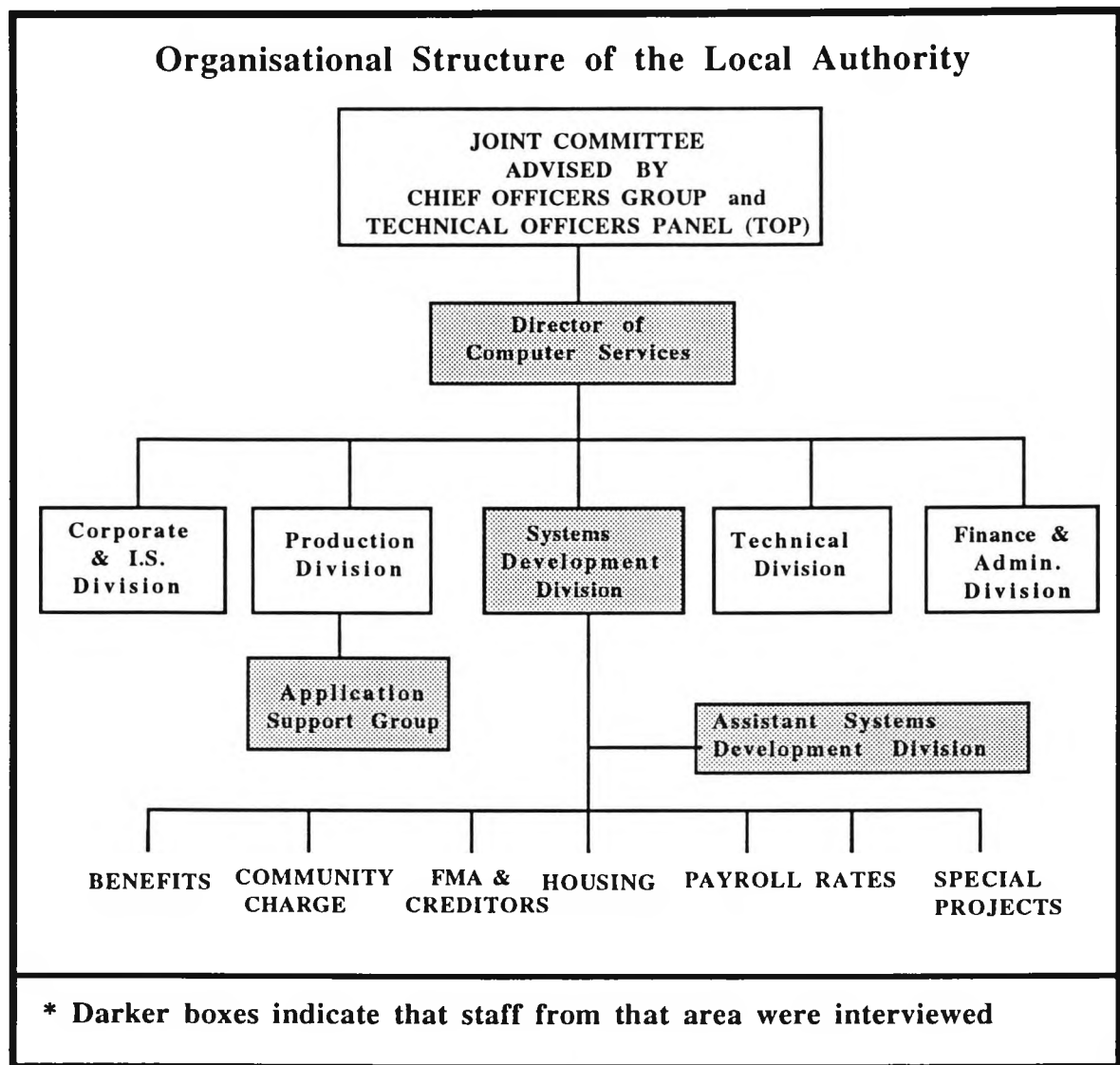


Fig. 4.2, Organisational structure of the Local Authority Computer Centre

Fig. 4.2 shows the local authorities organisational structure diagram where the darker boxes indicate the people who were interviewed. The interviews were conducted with four key people in the local authority computer centre. In hierarchical order, the interviews were conducted with the following people:

- the director of computer services;
- the systems development manager;
- the assistant systems development manager;
- the manager of the application support group (the one responsible for the running databases).

One interview was conducted with the director of computer services. Two interviews were conducted with the systems development manager. Four interviews were conducted with the assistant systems development manager and four interviews were conducted with the manager of the application support group (the one responsible for the running databases).

4.2.4 The Local Authority Processing Environment

The investigation of the local authority processing environment required the examination of the centre: hardware; operating system; database data communication (DBDC); and database management systems (DBMS).

The computer centre had three different categories of hardware, PCs, minicomputers, and mainframes. The centre mainframe operates under the MVS/XA operating system. The computer centre used the micro systems to take some load off the mainframe environment. The centre relied on it to perform several data manipulation tasks such as sorting operations, report generation and for decision support system (DSS) transactions. Only the mainframe environment was examined by this research.

The database environment was mainly IBM hierarchical database management system (DBMS) Information Management System (IMS) Data Language/I (DL/I). The IMS DL/I database management systems support several types of access methods such as: Hierarchical Sequential Access Method (HSAM); Hierarchical Indexed Sequential Access Method (HISAM); Hierarchical Direct Access Method (HDAM); and Hierarchical Indexed Direct Access Method (HIDAM). The on-line database environments at the local authority computer system were HIDAM databases where all users' queries used indexed attributes. The centre built a user interface written in customer information and control facilities (CICS) and PL/I programming language to access on-line databases. Fig. 4.3, is an illustration of the processing environment at the local authority computer centre.

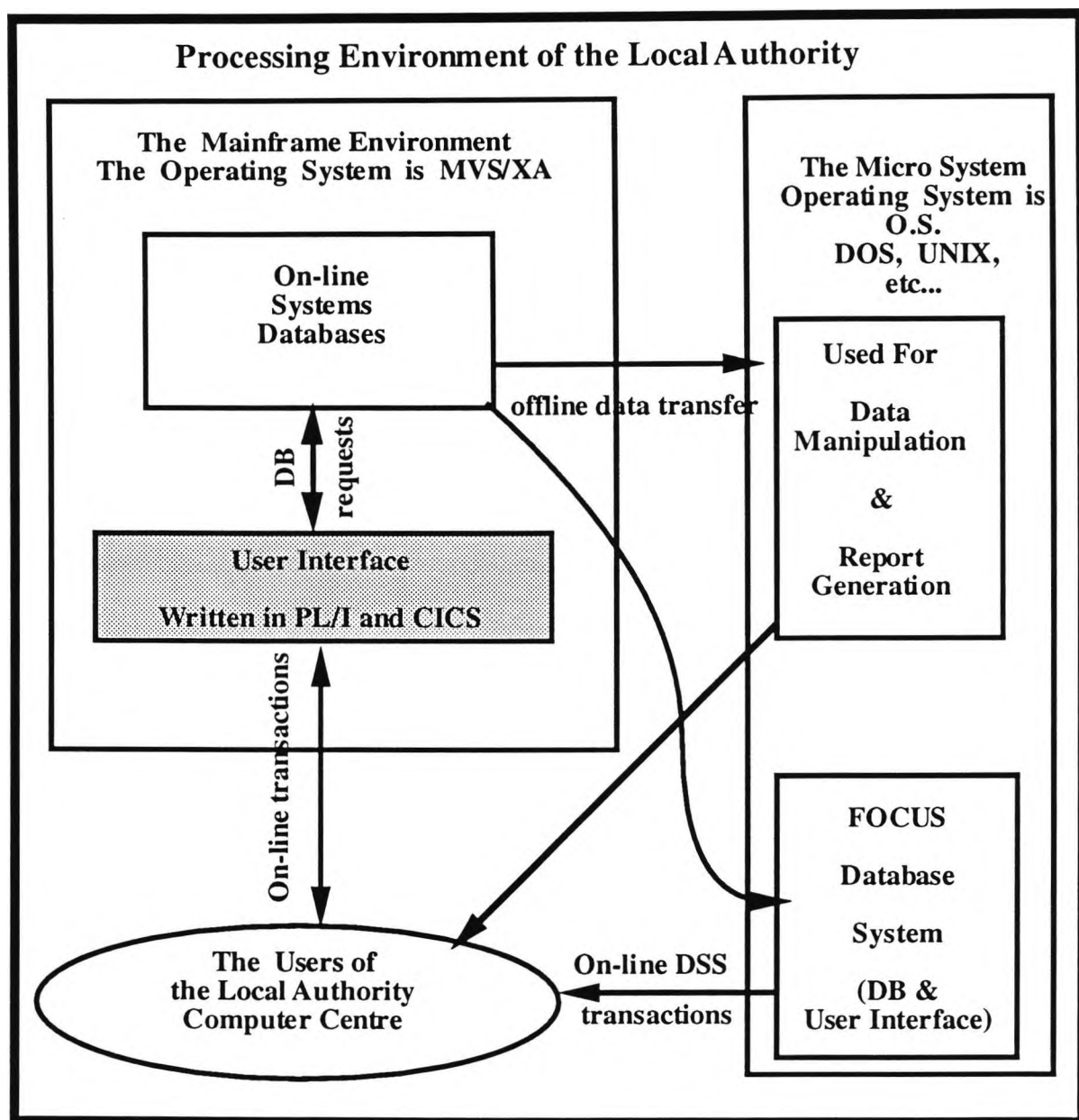


Fig. 4.3, Processing Environment of the Local Authority

4.2.5 The Local Authority Main Applications

The computer centre handled many different applications dealing with different activities. These activities varied from financial management and payroll systems to inventory control systems and housing systems. The computer centre supported over 300 different applications. Most of these applications were shared between the boroughs, except few of them were dedicated to a single borough. In addition the computer centre kept track of each borough income source and spending activities. Examples of the running applications are presented below.

1. Financial Management Application (FMA).
2. Creditors.
3. Payroll.
4. Rates.
5. Housing.
6. Housing Advances.
7. Housing Benefits Public and Private.
8. Property Information and Processing System (PIPS).
9. Stores.
10. Generalised Persons Application (GPA).
11. Cash Income System.
12. Direct Debit.
13. Bankers Automatic Clearing Service (BACS).
14. Community Charge.
15. Electoral Registration.

4.2.6 The Main Characteristics of The Application Databases

The research examined the local authority running systems' documentation to acquire the application databases' information. The examination included the database definitions (DBD) and program specification blocks (PSB) with the help of the manager of the application support group (the one responsible for the running databases) to obtain information such as: records size; number of attributes per record; number of indexed attributes per database; and attributes' types. Systems' output was the source of some information such as database sizes.

The research identified the running database characteristics by examining the following factors:

- database size;
- record size;
- number of indexed attributes;
- attributes types;
- distribution of attributes in databases.

A typical database size was difficult to identify, but the overall number of databases was large and over all on-line databases' size was around 12 Gbytes, whereas the most common record size was around 185 bytes.

All on-line databases were HIDAM databases that used indices, consequently, all on-line queries used indexed database attributes and non of the on-line queries used non-indexed attributes. The number of database indexed attributes were three indices on average. Attribute types were character, integer, and fixed point.

4.2.7 Transactions Behaviour of the Running Systems

The examination of several on-line transactions' performance factors identified the of on-line transactions' behaviour at the local authority processing environment. The performance factors are the following:

- on-line transactions database operations;
- Key and No-key Transaction Types;
- Average Number of Retrieved Records per Transaction.

4.2.7.1 On-line Transactions Database Operations

Breaking down on-line transactions into their basic database operations identified typical transactions' behaviour at the local authority processing environment. As shown in Table 4.1 and Fig. 4.4, sequential retrieval was the dominant database operations in this environment followed by qualified retrieval. The difference between retrieval operations ratios and other operations ratios was large, retrieve operations (both sequential and qualified) represented 91%, records update operation represented 4% and insert operation represented 4%. Delete operation was so small and represented a ratio of about 1%.

Operation Name	Qualified Retrieval	Insert a Record	Update a Record	Delete a Record	Sequential Retrieval
Average of DB Operations	36%	4%	4%	1%	56%

Table 4.1, Ratios of the Local Authority On-line Transaction Operations

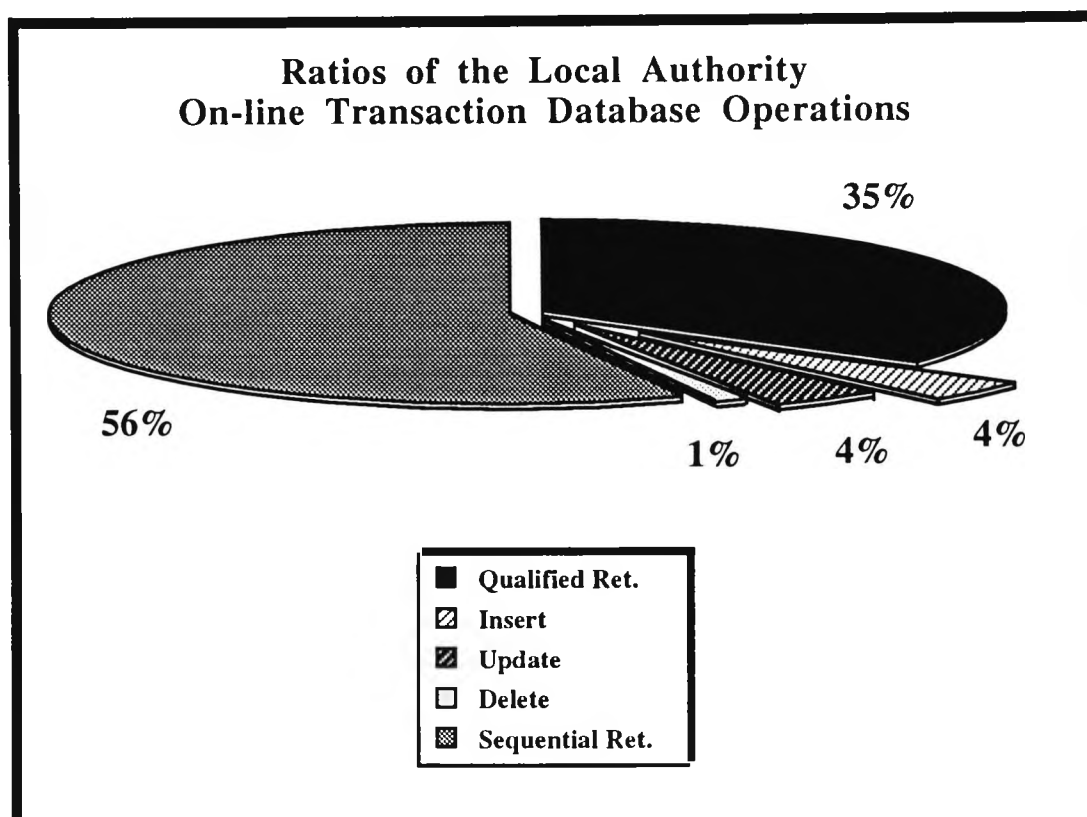


Fig. 4.4, Ratios of the Local Authority On-line Transaction Operations

4.2.7.2 Ratios of Key Utilisation

Transactions were classified to key retrieval transactions and no-key retrieval transactions to identify the nature of the on-line transactions at the local authority environment. Update, Insert and Delete operations were not included because their ratios were too small to be taken into consideration.

The ratios of the key retrieval transactions and no-key retrieval transactions are presented in Fig. 4.4. The ratios of the two types of transactions were as follows:

- No-key transactions represent 79% of the retrievals types;
- Key transactions represent 21% of the retrievals types.

4.2.7.3 Average Number of Retrieved Records per Transaction

The division of physical buffer size by the physical record size gave the average number of retrieved records per transactions. The average number of retrieved records by a single on-line transaction was around seven records.

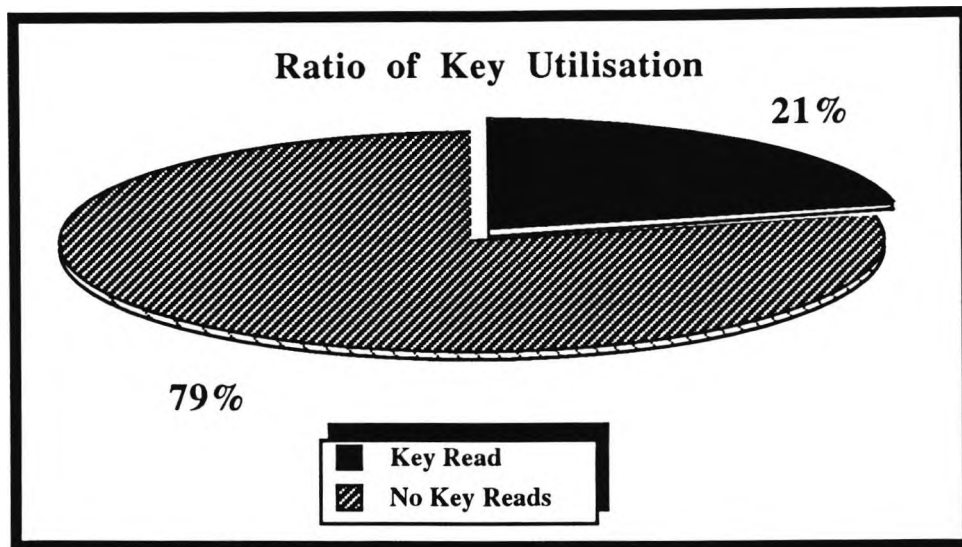


Fig. 4.5, Ratios of Key utilisation

4.2.8 Background Workload

All data manipulation operations such as sort operations and copy operations were performed during night shift when the on-line service was shut down and not available to the centre users.

Micro computer environment performed all report generation operations. All files to be printed were transferred to the micro system from the mainframe by off-line means then all the required reports were generated from that system.

Decision support system (DSS) transactions usually consist of long search in databases (I/O bound transaction) and require several sort paths before completion. The local authority separated the DSS transactions on micro computer environment.

The previous decisions freed the mainframe for on-line service and kept the background workload as low as zero.

4.3 A Study In a Large British Airlines' Computer Centre

4.3.1 Introduction

The second study took place at a large UK airlines' computer centre. The computer centre has a large database environment and large number of database applications that support different services. The airline computer centre runs around 2000 different applications accessing over 2500 hierarchical and relational databases occupying over 450 Gbytes.

Three powerful computer processors maintained 24 hours uninterrupted on-line service to the computer centre users. The processors supplied around 600 MIPS of computing power in all, where 286 MIPS were dedicated to on-line service. The on-line systems terminal population was approximately 250,000 terminals whereas around 27,000 on-line users accessed the system concurrently.

4.3.2 Data Gathering

Gathering the required data from the airlines' environment required the application of several techniques. Those techniques were the following:

1. monitoring the system over a representative period;
2. studying systems output and documents;
3. interviewing four staff at the airlines' computer centre and two staff at the airlines data centre.

4.3.2.1 Monitoring the Running Systems

The airlines' computer data centre was a separate body from the airlines' computer centre and has different responsibilities. While the airlines' computer centre task was supplying computer services to the users, the main task of the data centre was monitoring the running environment and recording the running systems' behaviour with system load, used disk space, and on-line transactions' response time. Then try to tune the running environment depending on that information. The data centre supplied this research with the running systems load over a period of fifty weeks.

The data centre did not record some information such as: transactions' types; transactions' operations; database buffer management; the average number of I/Os resulted from on-line transactions; and the average number of sorts resulted from each transaction. To get that information, the airlines' computer centre database administrators (DBA) monitored the running databases for fifteen days that were randomly selected from a period of four months. Systems' monitoring process collected all the transactions issued to the running systems during the period of study. A software available in the computer centre performed the following tasks:

- classified the collected transactions to their different types (on-line, DSS, QMF, and batch);
- sorted the transactions according to system name and transaction name;
- calculated the averages of number of database operations resulted from each transaction;
- calculated the averages of number of database buffer updates;
- calculated the averages of number of I/Os resulted from each transactions;
- calculated the averages of number of sorts resulted from each transaction;
- calculated the averages of transaction response time;
- calculated the averages of transaction CPU time.

4.3.2.2 Studying Systems Outputs and Systems Documents

System outputs included information such as system load and available disk space. System documents contained system specification and design rules for both applications and databases. Studying systems output and documents was the source of information that the research could not obtain by monitoring the running systems, this information was such as: number of databases accessed by a single transaction (JOIN operation); database record size; number of indexed attributes; attributes' values; distribution of attributes in the database; and number of nested selects in a single transaction (transaction complexity).

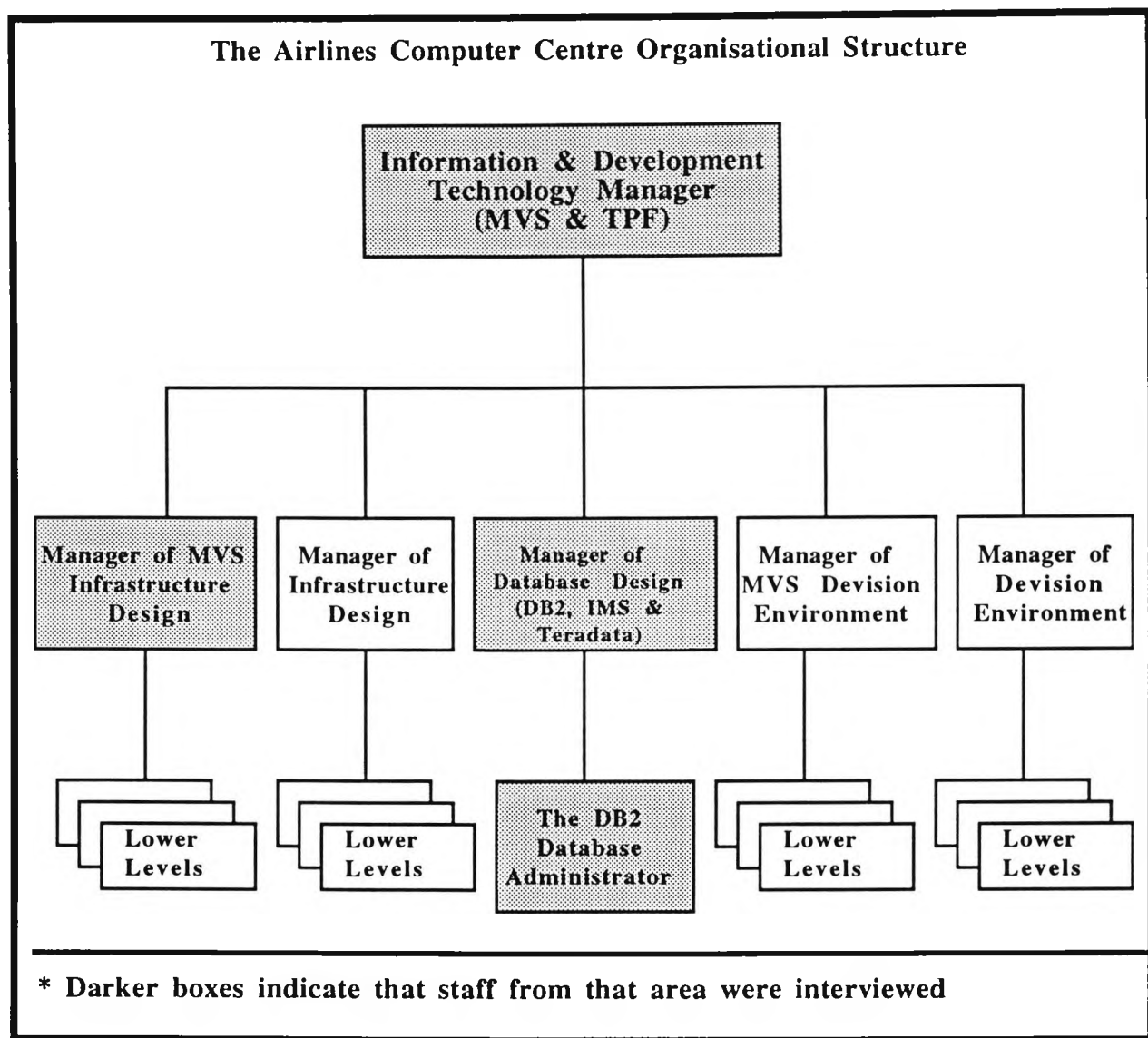


Fig. 4.6, Organisational structure of the Airlines computer centre

4.3.2.3 Interviews

Interviews were conducted with six key people, four from the computer centre and two from the data centre. The interviews were conducted with: the information and development technology manager (MVS & TPF); the manager of the MVS infrastructure design; the manager of database design (DB2, IMS and Teradata); and the DB2 database administrator from the computer centre; and the manager of the data centre and one of his assistants from the data centre. The interviews helped in gaining better understanding of the computer centre environment, but did not play a main role in collecting a great deal of technical information.

The organisational structure of the airlines' computer centre is presented in Fig. 4.5. Darker boxes indicate that staff from that area was interviewed. The organisational structure of the data centre was not required as only the head of the centre and one of his assistants were interviewed.

4.3.3 Processing Environment

The airlines' computer centre had several levels of hardware; mainframes, mini-computers, micro-computers, and PCs. This research covered only the mainframes' environment, and did not cover mini-computers; micro-computers; and PCs.

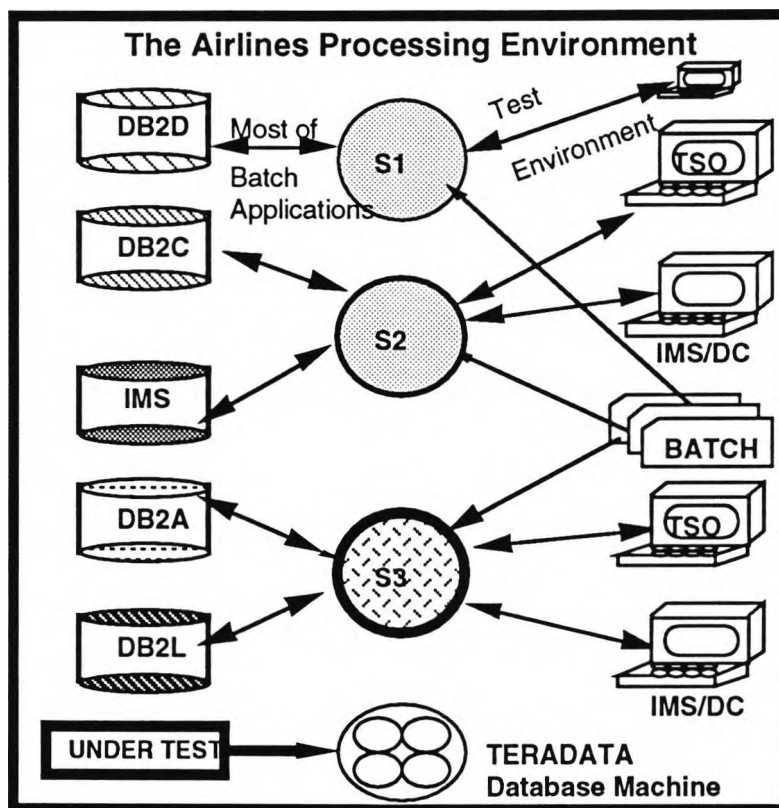


Fig. 4.7, Processing Environment of the Airlines computer centre

The on-line service computing power consisted of three mainframes (physically, three different processors). The first mainframe ran at 66 MIPS and each one of the other two processors ran 110 MIPS. All three processors ran under the MVS/XA operating system. In addition there were several other processors used for systems monitoring, analysis, and tuning that raises the total amount of MIPS to 600. The airlines' computer centre processing environment is illustrated in Fig. 4.7.

The first processor (S1), was dedicated to systems development and testing of applications. This system was mainly used for applications development and served relational database DB2 applications. This processor also served to perform most of batch applications runs.

The second processor (S2) performed the IMS applications at the airlines' environment. It also served newly developed on-line relational database (DB2 system) applications. This processor was also used for turning round large volumes of transactions quickly. In addition, it ran a very limited number of batch applications.

The third processor (S3) supported the on-line services of relational database systems (DB2). This processor main function was to support the on-line service but when the S1 processor was not available, a limited number of batch applications could run on this processor.

The airlines on-line service supplied its services to several countries from all over the world. Users from UK, USA, Canada, and Australia were able to use the on-line service and access the running database. Due to the difference in time zones between these countries, the system was available and accessed by on-line users over the 24 hours. The centre supported more than 250,000 terminals in all, but due to the difference in time zones between the countries using this system only around 27,000 on-line users concurrently used the system.

The airlines' computer centre covered a wide variety of applications whereas the airlines' company activities were represented at the computer centre; they supplied more than two thousands different applications covering all the company activities to their users. The applications covered different activities such as: flights (fuel consumption, catering, etc.); Engineering (workshops, spare parts, inventories); management (personnel, payroll, etc.). In addition, they had a large ticket reservation system.

4.3.4 The Main Characteristics of the Application Databases

The identification of the main characteristics of the on-line databases required the examination of several database performance factors. Those factors were the following:

- database size;
- number of databases;

- typical record size (high occurrences of records with the same length);
- number of on-line databases indexed attributes.
- types of database attributes.
- distribution of database attributes.

The running systems' databases were resident on three processors. The first was the S2 processor, which mainly supports the IMS hierarchical databases and newly created relational databases. The second was the S3 processor, which mainly supports the main service of the relational databases. The third database environment was the acceptance test system (ATS environment, resident on S1) which was an intermediate system between the test applications and running applications. The following sections will discuss database characteristics at the three environments.

4.3.4.1 Application Databases on the S2 Processor

Two different types of database management systems were running in this environment, an IMS database management system and DB2 database management system. The majority of the databases on the S2 processor were IMS databases. The IMS database consisted of around 780 on-line databases taking 31 Gbytes of data over 65 disk packs. Those disk packs were mainly single density. The centre used data packer to compress the larger IMS databases. A typical record size at this environment was around 185 bytes.

The DB2 databases occupied about three Gbytes of data. Under S2 there were around 200 live relational DB2 tables. The total numbers of rows were around 6,000,000 rows; where the largest table contains about 600,000 rows. The maximum row length was 4036 bytes and the minimum row length was 5 bytes; but typical row length was around 175 bytes.

4.3.4.2 Application Databases on the S3 Processor

All the databases on the S3 system were relational DB2 database management system. It was originally designed for large data quantities with the ability for end users, using QMF, to interrogate their data.

The databases under this system were occupying around 50 Gbytes of data over 46 disk packs. This includes single, double, and triple density disks. Under S3 there were more than 500 live tables.

The total rows in these databases were around 300,000,000 rows, where the largest table contains around 98,000,000 rows, and the smallest table contains just one row. The maximum row length was 4025 bytes with minimum row length of 2 bytes and most common row length of around 200 bytes.

4.3.4.3 The Acceptance Test System (ATS)

The Acceptance Test System (ATS) was an intermediate system between test systems and real system. As systems were completed they were moved to the ATS for a test period where they were tested in conditions similar to those of real world, if they show satisfactory results they were moved to live systems or else they were returned to development environment for further tuning. The ATS environment consisted of about 300 IMS databases and 40 DB2 databases.

4.3.5 Transactions Behaviour of the Running Systems

Transaction analysis process investigated all on-line transactions issued to the running systems over a period of fifteen days. These fifteen days were selected randomly by the airlines' computer centre from a period of four months. The total number of collected transactions through this study were around 25 million transactions. The research examined several performance factors to identify a typical behaviour of transactions at the airlines' environment. Those factors were the following:

- number of on-line transaction basic database operations;
- number of DSS transaction basic database operations;
- number of records retrieved per on-line transaction;
- number of databases accessed per on-line transaction (JOIN operation);
- number of nested selects per on-line transaction (transaction complexity);
- ratio of response time to CPU time;
- number of I/Os resulted from each transaction;
- number of database buffer updates;
- number of sorts per transaction.

4.3.5.1 On-line Transaction Database Operations

Sequential retrieval represented the dominant database operation with 51%, qualified retrieval represented 39%; the three updates operations (insert a record, update a record, and delete a record) represented 11% in total. The ratios of on-line systems transaction operations are presented in Table 4.2 and Fig. 4.8.

Operation Name	Qualified Retrieval	Insert a Record	Update a Record	Delete a Record	Sequential Retrieval
Average of DB Operations	39%	5%	4%	1%	51%

Table 4.2, Ratios of the Airlines On-line Transaction Operations

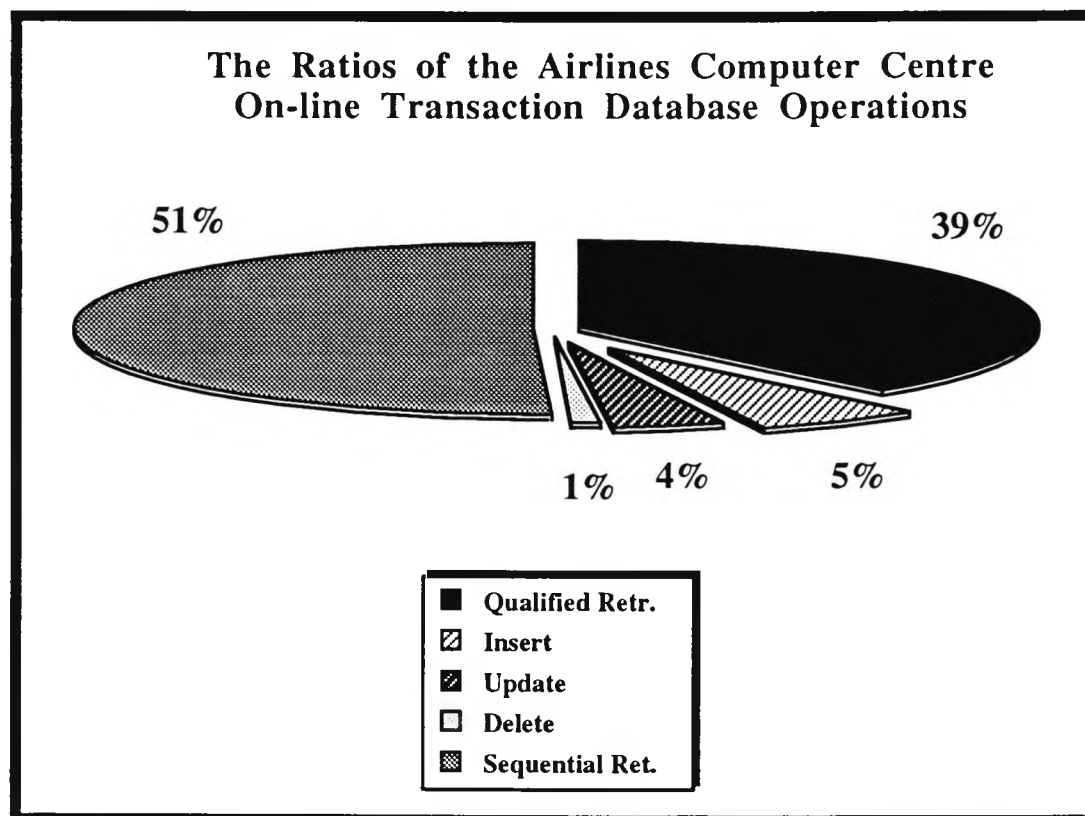


Fig. 4.8, Ratios of the Airlines On-line Transaction Operations

4.3.5.2 Time Utilisation and I/O Operations

The ratios of the CPU utilisation time to I/O time are illustrated in Fig. 4.9. The average response time of on-line transactions was around three seconds, whereas the

CPU used 25% of that time. On-line transactions produced around 900 I/O operations per transaction and generated about 10 sort paths per transaction.

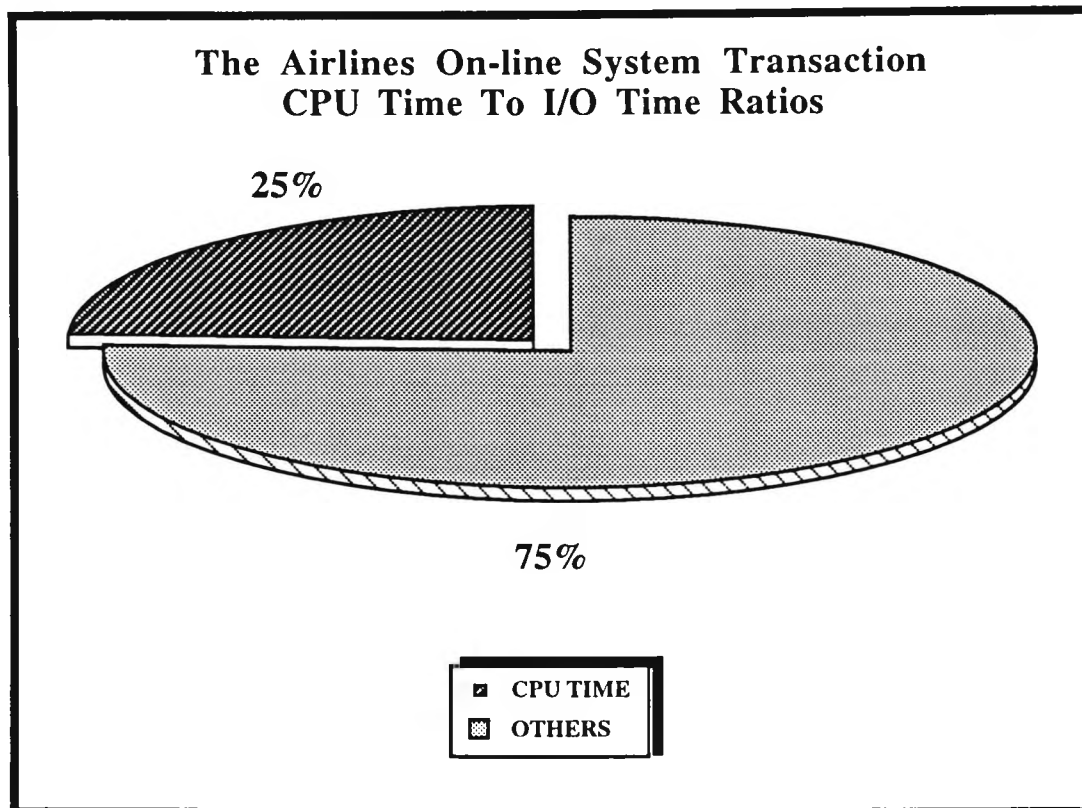


Fig. 4.9, The Airlines On-line transaction I/O time ratios

4.3.5.3 Number of Databases Accessed by One Transaction (JOIN Operation)

The airlines' computer centre avoided JOIN operations since most transactions accessed one database (93%). When JOIN operation was used, it was mainly used to access three databases (5%). Fig. 4.10, illustrate the ratios of the number of databases accessed by one transaction.

4.3.5.4 Transactions Complexity (Nested Selects)

The majority of transactions consisted of one select, around 98% of the running transactions, even when they accessed more than one database. In the mean time a small percentage, around 2%, consisted of two selects. Transactions that consisted of more than two selects represented less than .5%. Fig. 4.11 illustrates the ratios of the number of nested selects per an on-line transaction.

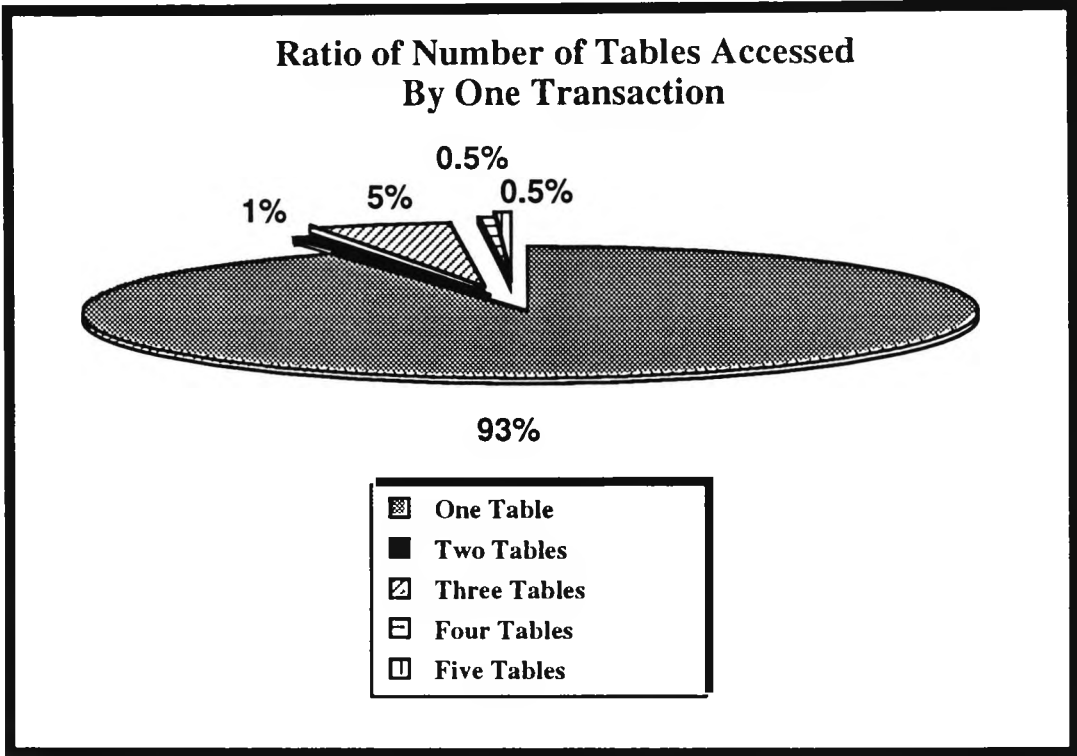


Fig. 4.10, The Airlines On-line transaction JOIN ratios

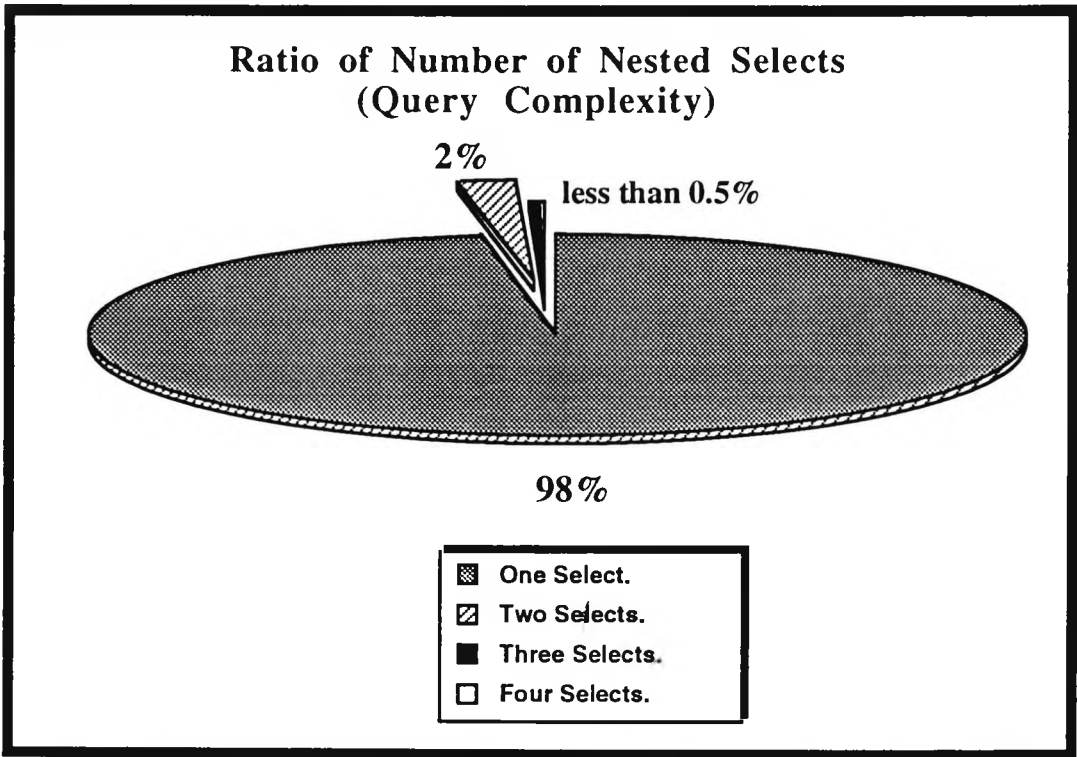


Fig. 4.11, The Airlines On-line transaction nested selects ratios

4.3.6 Number of Retrieved Records per Transaction

Generally, there was a trade-off between transactions' database operations and number of retrieved transactions, if transactions tend to perform more sequential retrieval transactions than qualified retrieval, the number of retrieved records per transactions will be relatively high, and the opposite if transactions tend to perform qualified retrieval transactions more than sequential retrieval transactions. The average number of retrieved records per transaction was around seven records.

4.3.7 The Background Workload

The background workload of on-line environment at the airlines' computer centre was very small. The centre performed most of batch jobs, copy operations, and sort operations, on processor S1 where the on-line service was not available. The other processors S2 and S3 supplied the on-line service to the system users and the background workload on those processors were very small. The background workload of the S2 processor and S3 processor consisted mainly of back-up operations and restore operations that were usually run during night shifts when on-line service was at its minimum.

4.4 A Study In a Large UK Bank Computer Centre

4.4.1 Introduction

The previous sections, discussed two studies at two large database environments, the first took place at a large local authority computer centre and the second examined a large UK airlines' computer centre. This section, discuss a study at a large UK bank environment. The results from this study were important to this research because the bank database environment was the largest in UK and because the TPC-A, which was the standard benchmark of the database market, simulated random transactions at large bank environment.

The bank runs a large number of applications (more than 2500 applications) and has a wide variety of those applications. The bank computer environment was one of the largest on-line environments not only in UK but also in Europe and now it is the

leading international bank trading in over 30 nations. It supplied more than 150 financial services products using more than 65 different account types generating over 9,000,000 daily entries producing 200 reports and 150 interface data streams. The bank has a large database environment that consisted of more than 2500 application databases occupying over three Terabytes of data, out of that, the bank on-line users accessed 750 Gbytes of data.

Data gathering used telephone calls, face to face interviews, systems monitoring, and examined the running systems documents. The bank computer centre management agreed to monitor the system for fifteen days in the same way as the airlines' computer centre was monitored. Because the bank computer centre was exceptionally large, it was too expensive to monitor the system for fifteen days and the bank monitored the running environment for eight days.

During the eight days of system monitoring the bank collected all on-line transactions issued to the on-line systems. Then, the bank computer centre sorted and classified the collected transactions to on-line transactions from terminals in the bank branches and on-line transactions from the automatic teller machines (ATM); for short they will be called on-line transactions and ATM transactions. The studied transactions were then broken down to their basic database operations and the ratios of those operations were calculated to give typical transaction behaviour.

4.4.2 Data Gathering Techniques

Several data gathering techniques were employed to collect the required information about the running environment. Those techniques were: interviews; monitoring the running systems; and studying systems documentation. The following sections will discuss those techniques.

4.4.2.1 Interviews

Before starting the interview, the author made several informal telephone calls to the bank information systems manager that explained the nature of the study and discussed the main objectives of this research. Then the author sent a list of questions and requirements to the bank information centre manager to study before agreeing to have one interview. The bank information centre manager promised that if one

interview was not enough to cover all the required points he will arrange another interview.

One long interview that took around six hours was enough. The interview discussed the following points:

- the bank activities;
- the running applications;
- the application databases;
- the running systems load.

In addition, the bank information centre manager prepared a list of the answers to the questions and to some other technical points such as JOIN operations and transactions' complexity that were sent to him before the interview was started.

4.4.2.2 Monitoring the Running Systems

By the end of the interview the bank agreed to monitor the running environment for fifteen days, but due to the exceptionally large size of the bank computer environment, monitoring the bank environment was extremely expensive and the bank monitored the running environment for eight days, that were more than enough for this research.

Monitoring the running systems was the only way to study the behaviour of the running transactions in the bank environment. The bank computer centre collected all the on-line transactions issued to the running systems and processed those transactions to produce the following information:

- classified the collected transactions to their different types (on-line and ATM);
- calculated the average number of database operations per transaction;
- calculated the average number of database buffer updates;
- calculated the average number of I/Os resulted from each transactions;
- calculated the average number of sorts resulted from each transaction;
- calculated the averages of the transaction response time;
- calculated the averages of the transaction CPU time.

4.4.2.3 Studying Systems Documentation

The examination of systems' documents clarified some information that was impossible to collect by monitoring the running systems or by the interview. The information was the following:

- disk space utilisation;
- database size;
- number of indexed attributes;
- types of attributes;
- distribution of attributes;
- transactions complexity (number of nested selects);
- number of databases accessed by one transaction (JOIN operation).

4.4.3 Processing Environment

The examination the bank processing environment covered the running systems' hardware, software and database management systems (DBMS). Three equally important factors influenced the bank processing environment, those factors were: transaction response time; system integrity and recovery; and system security, the bank environment with both hardware and software were designed to serve those three factors.

4.4.3.1 Hardware and Software

The bank divided its computing power between two production centres and a development centre. Each one of those centres produced 500 MIPS. When one of the production centres goes down, the development computer centre works as a back-up for the down centre, that allowed the bank to maintain 1000 MIPS of computing power to on-line service under different circumstances.

The data centres served more 3500 branches with terminal population of more than 22,000 terminals supply on-line service to around 60,000 users. In addition, more than 3500 automatic teller machines (ATM) with on-line access to the system over the 24 hours heavily accessed the system. The bank offered its services not only to UK branches but also to several other branches all over the world, those results in 24

hours' access to the computer system. Fig. 4.12, shows the bank branches to data centres connections.

The bank branches accessed the running systems through non-programmable terminals. The user terminals were connected to a local server that was responsible for session control and screen formatting. Local servers also acted as a branch interface equipment (BIE) and stored transactions on recoverable media before sending them to the mainframes at the data centres.

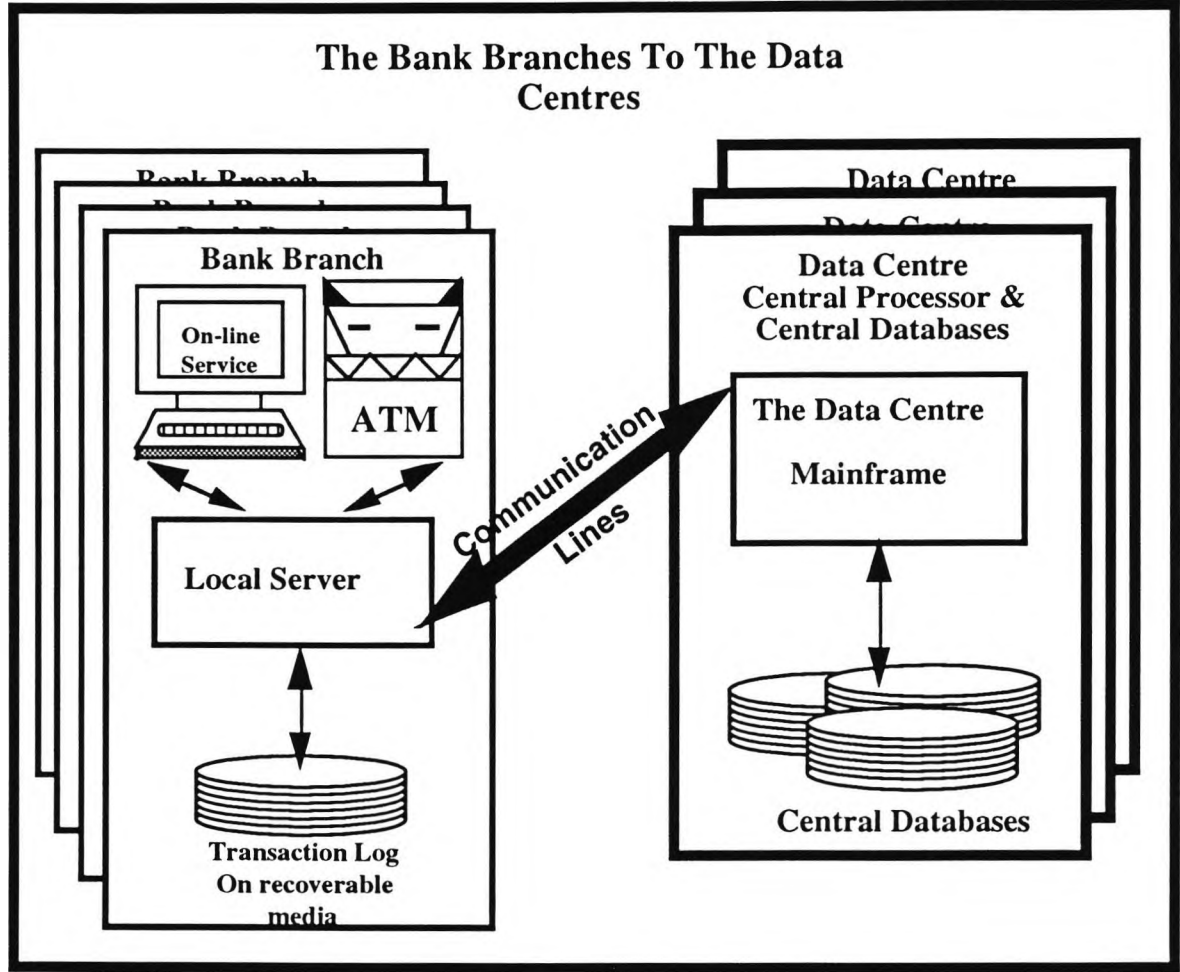


Fig. 4.12, The Bank branches to data centres communication

Given the heavy transaction load, the bank divided on-line transactions between the two main UK data centres. Within each data centre, transactions were further subdivided between the data centre processors. Fig. 4.13, shows the bank branches to data centres connections.

On-line transactions' response time was the major constraint at the bank environment. On-line transaction response time must be within a certain time limit or an internal timer in the communication controller will time out the transaction and treat it as transaction failure. To enhance on-line transactions' performance, each data centre had two types of processors, on-line processors, and off-line processors. On-line processors were dedicated to receiving on-line transactions from the bank branches through communication lines and serve those transactions. Off-line processors were dedicated to perform all other types of operations in the data centre, they performed tasks such as: back-up the running databases; report generation; sort operations; copy operations; and block updates to some application databases.

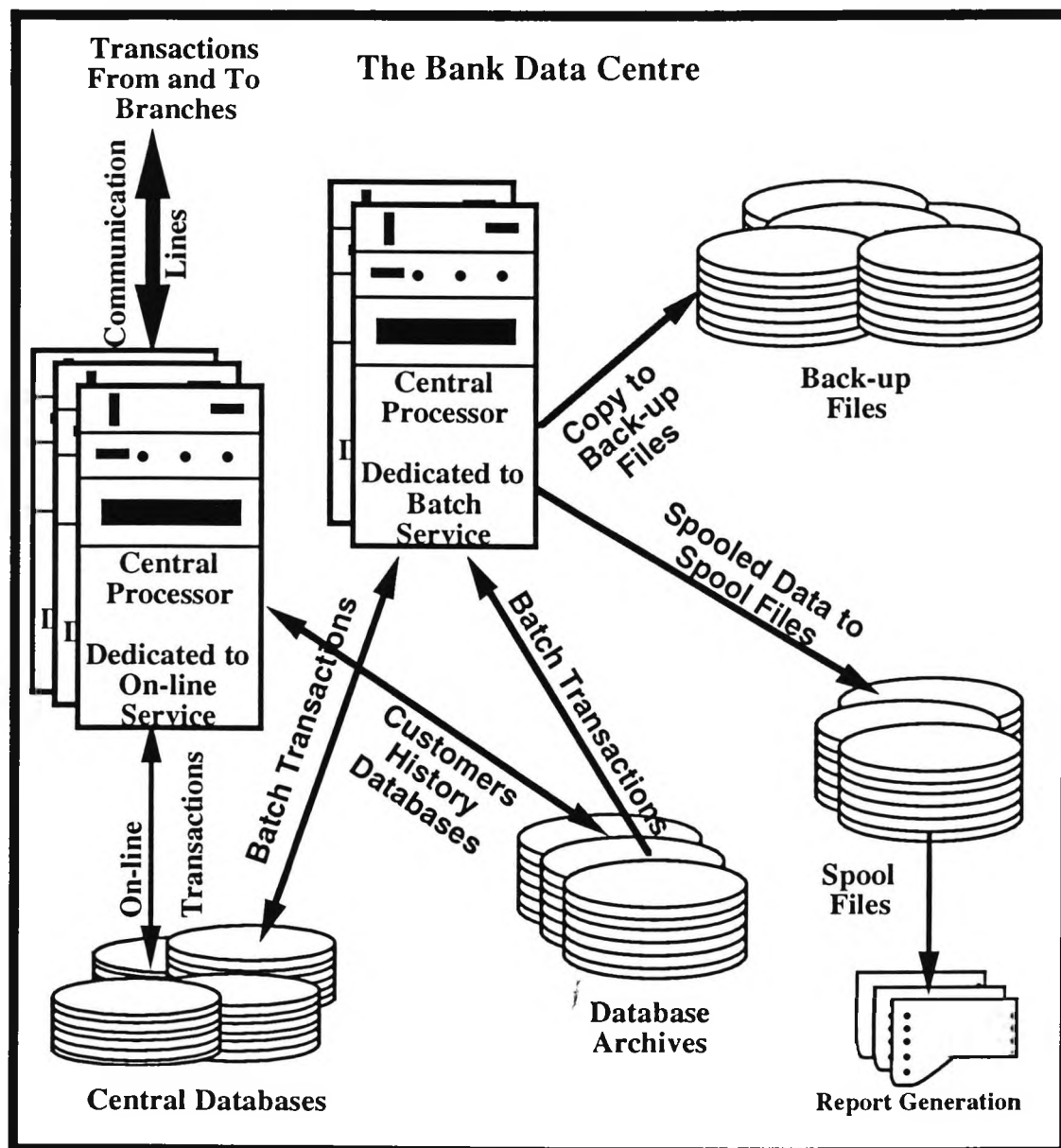


Fig. 4.13, The Bank data centre

The bank kept its customers' history in archives that were cumulative storage and their records were never deleted. Because the bank archives were mass storage devices and not on-line database, they were not covered by this research.

Data centres kept several log files of transaction operations that contain all before image and after image transactions. Log files were not databases and were not covered by this study.

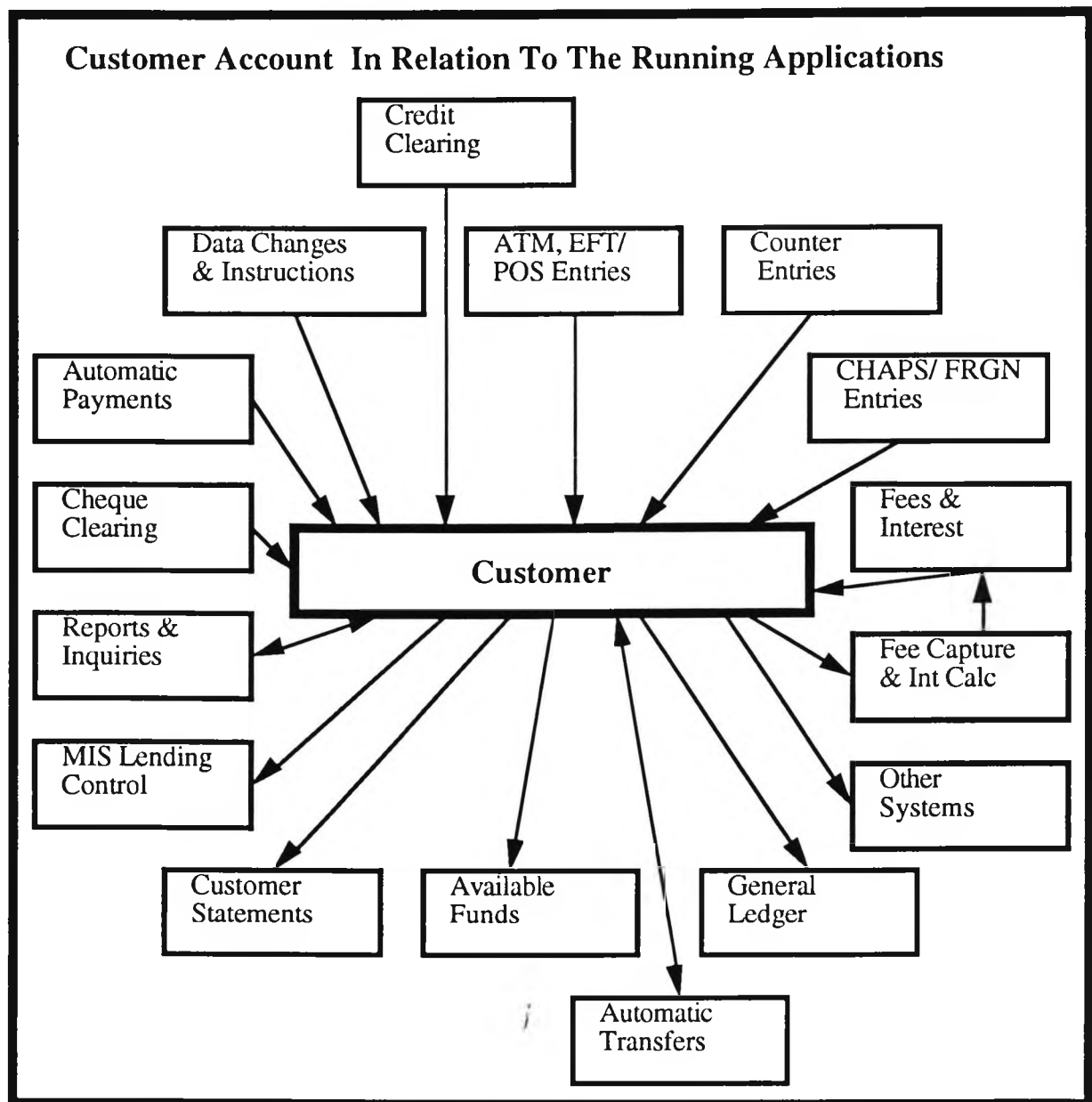


Fig. 4.14, Customer Account in relation to running applications

4.4.3.2 The Bank Running Applications

Customer accounting system was the very hub of the bank business division where most of the bank business was products they developed around that customer accounting system. The number of customers' accounts was over 17M accounts whereas the average accounting entries per day was about 6M accounts and on a peak day it reached 20M accounts per day and in some cases it reached 650 transactions per second. Fig. 4.14 presents the connections between the different applications and customers' accounts.

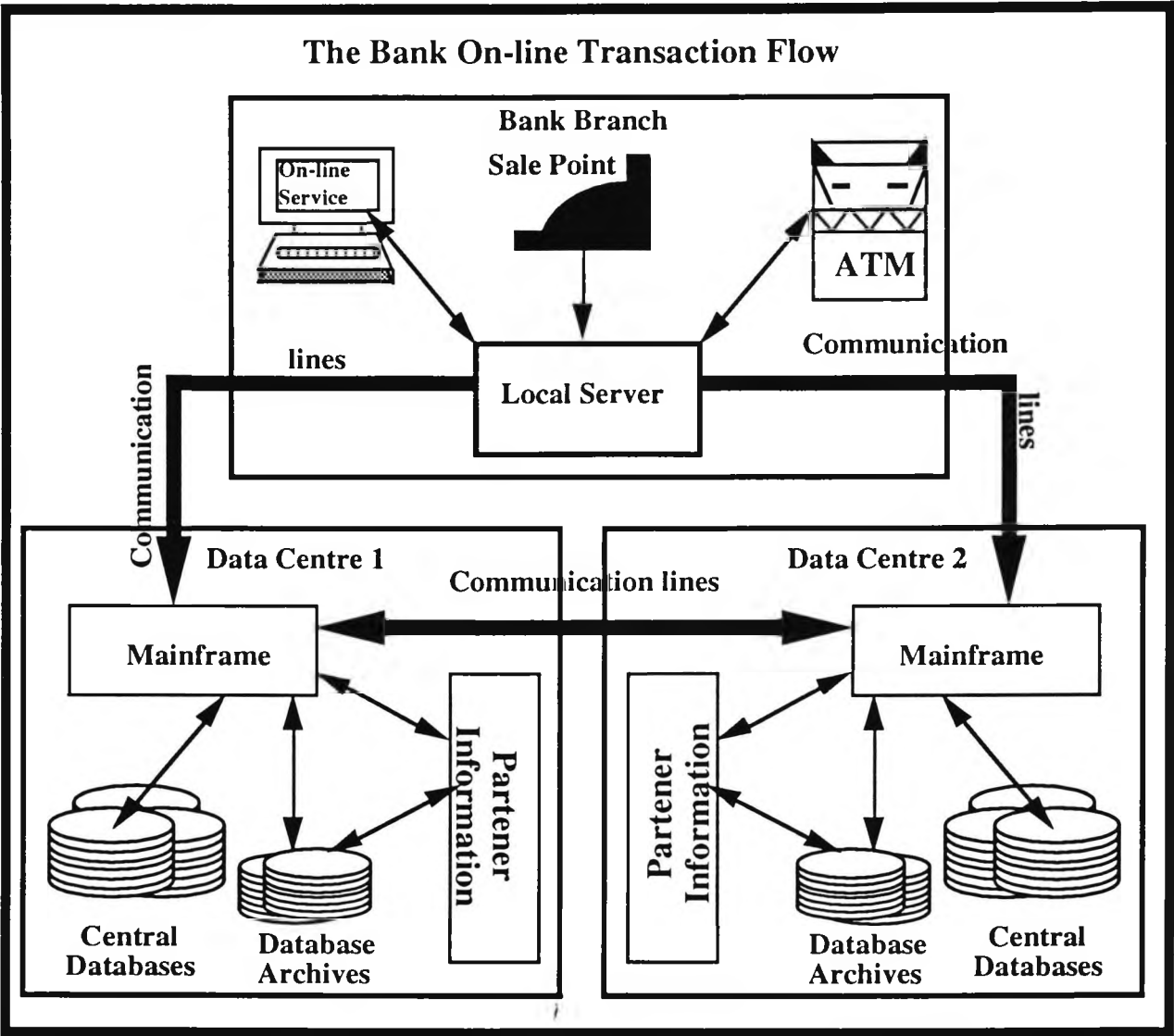


Fig. 4.15, The Bank On-line transaction flow

4.4.3.3 On-line Transaction Flow

On-line applications had a service level agreement covering systems availability and transaction response time. If just human factor was the only element involved, they would have a chance to tolerate a flexible response time, but the problem was the time limit of the internal timers in the network controllers. Any failure to respond within its time-out period equates to a system failure and the transaction was flushed. To solve this problem two autonomous partner servers in geographically remote sites were used and on-line systems transactions were divided between the two servers. In normal operations, the workload was balanced by half the controllers having primary affinity with one partner, and the other half to the other partner. A controller directs its request to its primary partner, and if it does not receive a response within three seconds, retires the request to the other partner. Each server notifies its partner of the business effects of authorisation it has performed to maintain data concurrency. This process of transactions' re-distribution was completely transparent to both the end users and application programs.

Each data centre has its own full copy of databases to eliminate distributed read. The problem was some database applications were characterised by relatively high read-write ratio. To overcome data consistency problem when updating those databases, all on-line updates and copy were blocked for those applications. Update, were data collected and applied in batches to the nominated master only before being replicated through copy utilities. Fig. 4.15, illustrate on-line transaction flow at the bank environment.

4.4.3.4 The Bank Database Management Systems

The bank mainframe supported three DBMS; hierarchical IMS DBMS, relational DB2, and the bank self made DBMS. Transaction analysis process investigated the three types of database management systems.

The first DBMS was an IMS hierarchical/database management systems. Most of the bank older applications were still running under IMS environment when the study took place, applications supported by IMS represented around 30% of the running database environment. This ratio is expected to go down because the bank was converting all its IMS databases to DB2 relational database management systems.

The second DBMS was a DB2 relational database management systems. The bank was developing all its new applications in DB2 DBMS and the percentage of the applications running under this system was increasing. The bank new information system, Information Systems Service (ISS) for UK banking division, was the largest DB2 application systems in UK and one of the largest in Europe.

The third DBMS was the bank self made database management system that the bank staff has developed used for so long to support the bank applications. Recently, the bank management decided that they were not a research centre and they can not invest in this DBMS to keep track with new products in the market. They were converting all the bank self made DBMS databases to DB2 relational database management system, as a result. This DBMS represented a small percentage of the running database environment.

4.4.4 Characteristics of Application Databases

The identification the main characteristics of the bank application databases required the examination of the following factors:

- database size;
- record size;
- attributes types;
- number of indexed attributes;
- types of attributes;
- distribution of attributes.

The examination of database sizes could not identify a typical database size at the bank on-line environments, but the bank had a very large database environment. Those databases occupied around three Terabytes of data and characterised by their large number of databases and large number of records per database. The application databases consisted of over 2500 hierarchical, relational and the bank system databases.

Record sizes of the bank on-line database environment was relatively small whereas the majority of systems used a record size around two hundred bytes. This was another way to keep response time down.

All the different types of DBMS at the bank database environment used indices. The IMS systems used HIDAM DBMS that uses index with each created database. The

bank relational database management systems databases used indexed attributes for all on-line transactions. The bank used an average of five indexed attributes per on-line databases. The bank database attributes were mainly integers, fixed point and character attributes.

4.4.5 Transaction Behaviour of the Running Systems

The bank collected over 11,000,000 on-line transactions over the period of study. Then it sorted and classified those transactions to on-line transactions and ATM transactions. Transaction's analysis broke down transactions to their simplest database operations and obtained the overall transactions' behaviour by calculating the following operations:

- the average number of basic database operations resulted from one transaction;
- the average ratio of CPU to I/O time utilisation;
- the average number of I/O operations resulted from one transaction;
- the average number of sort operations resulted from one transaction;
- the average number of databases accessed by one transaction (JOIN Operation);
- the average number of nested selects in one transaction (transactions complexity).

4.4.5.1 On-line Transactions Database Operations

The dominant database operation at the bank environment was the qualified retrieval (50%). Sequential retrieval came second with 33%. The other observation was the ratio of database record update operation was relatively higher than the same ratios for the previous two studies. This can be referred to the policy at the bank environment that put performance ahead of everything else. At this environment, most of on-line transactions retrieved a qualified record from one of the databases using customer key, which was a unique key, and returns one record. Transactions that performed sequential operations were less frequently used. Table 4.3 and Fig. 4.16 illustrate the bank on-line transactions' operations.

Operation Name	Qualified Retrieval	Insert a Record	Update a Record	Delete a Record	Sequential Retrieval
Average of DB Operations	50%	5%	12%	0%	33%

Table 4.3, Ratios of the Bank On-line Transaction Operations

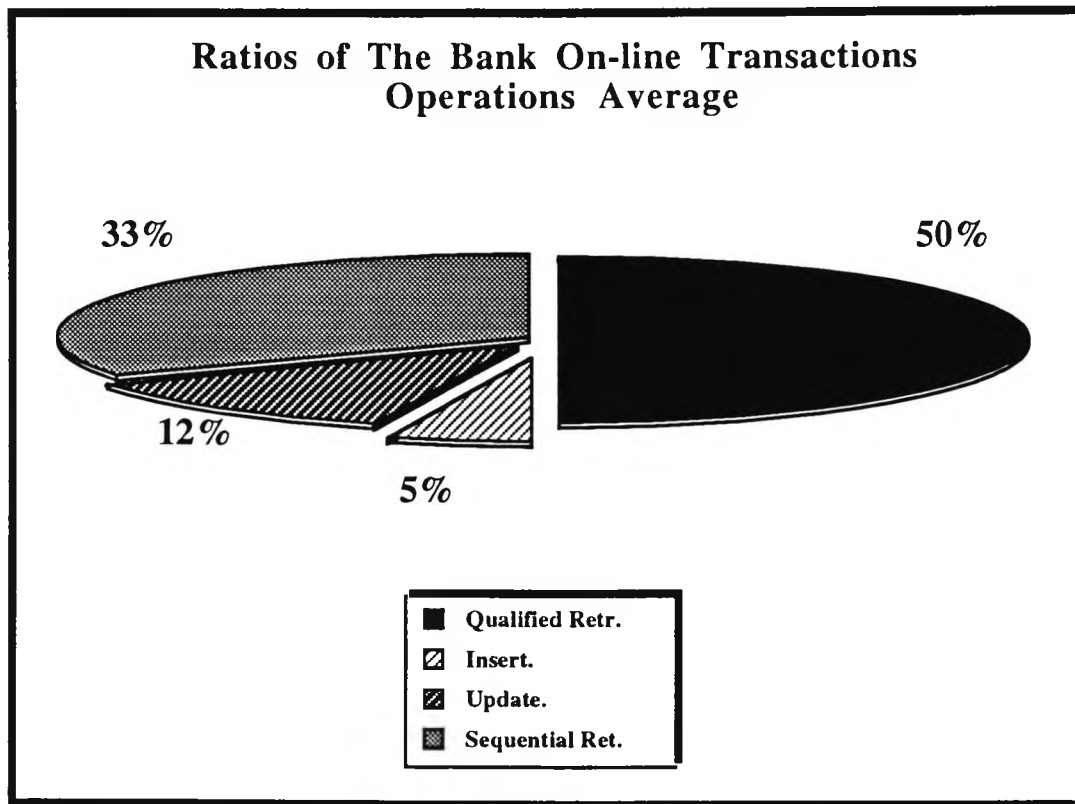


Fig. 4.16, Ratios of the Bank On-line Transaction Operations

4.4.5.2 ATM Transactions Database Operations

The bank offered ATM services through two ATM systems, ATM system one and ATM system two. The bank physically separated the two systems on two different processors. Table 4.4 and Fig. 4.17 illustrate the ratios of the ATM systems one and system two transactions' operations.

At the ATM system one environment, sequential retrieval represented 42%, qualified retrieval represented 27%, insert a record represented 6%, update a record represented 24%, and delete operation represented 0%.

Operation Name	Qualified Retrieval	Insert a Record	Update a Record	Delete a Record	Sequential Retrieval
ATM-1	27%	6%	24%	0%	42%
ATM-2	24%	6%	20%	0%	50%
ATM Average	26%	6%	22%	0%	46%

Table 4.4, Ratios of the Bank ATMs Transaction Operations

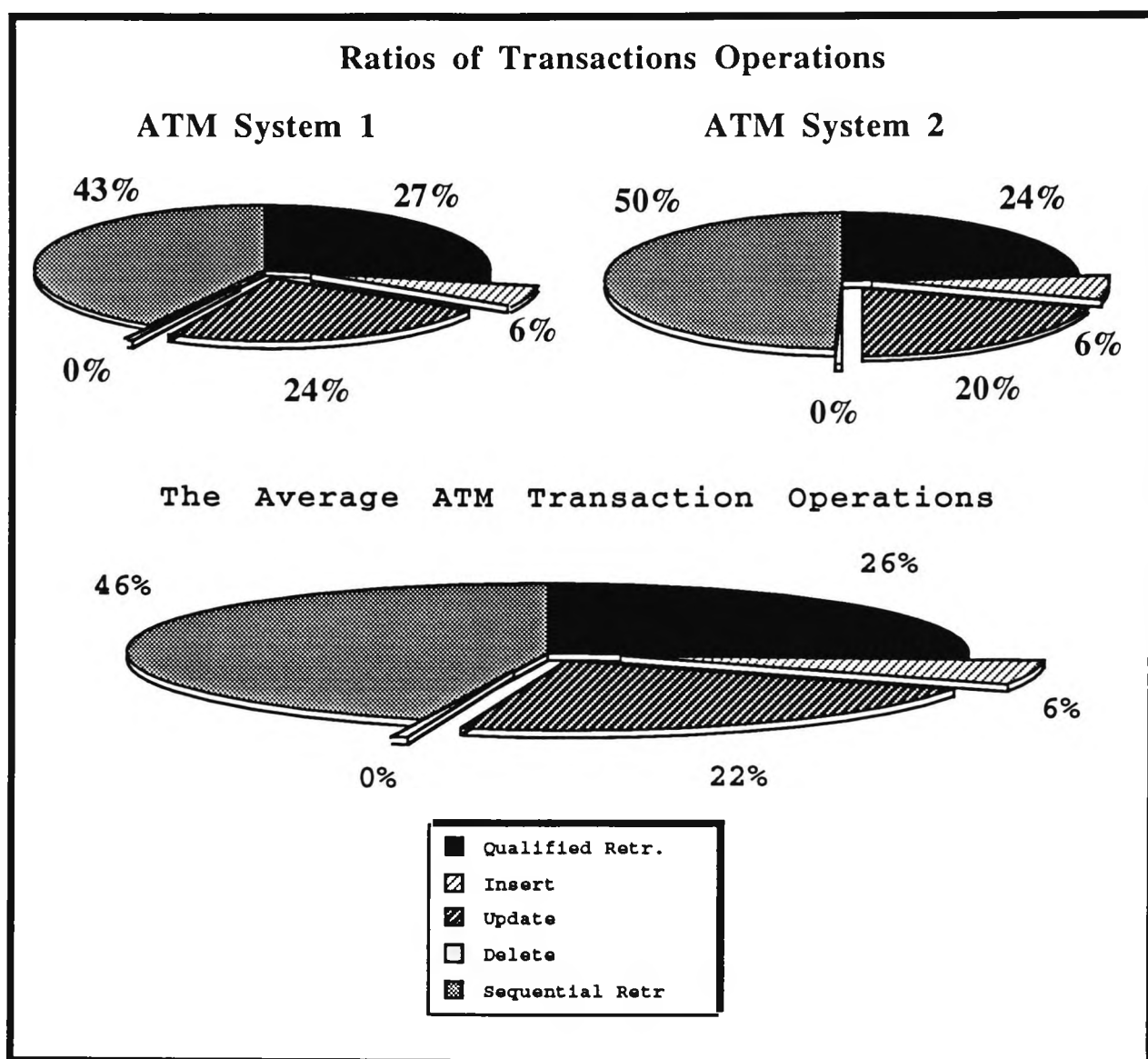


Fig. 4.17, Ratios of the Bank ATMs Transaction Operations

The ATM system two followed a similar trend. Sequential retrieval represented 50%, qualified retrieval represented 24%, insert a record represented 6%, update a record represented 20%, and delete a record represented 0%. The average of the two systems was calculated and presented in Table 4.4 and in Fig. 4.17. Sequential retrieval

represented 46%, qualified retrieval represented 26%, insert a record represented 6%, update a record represented 22%, and delete a record represented 0%.

Having in mind that the TPC-A benchmarks supposed to simulate a bank transaction, similar to the ATM transaction. We find that the ATM transactions' operations at the bank environment were widely different from the TPC-A benchmarks script that consists of three updates operations and one insert. This result will be discussed in greater detail in the chapter discussion.

4.4.5.3 CPU Time Utilisation and I/O Operations

The research investigated on-line transaction time utilisation as transaction response time and the way it spent that time between the CPU and I/O operations. The subtraction of transaction CPU utilisation time from transaction response time gave transaction I/O time.

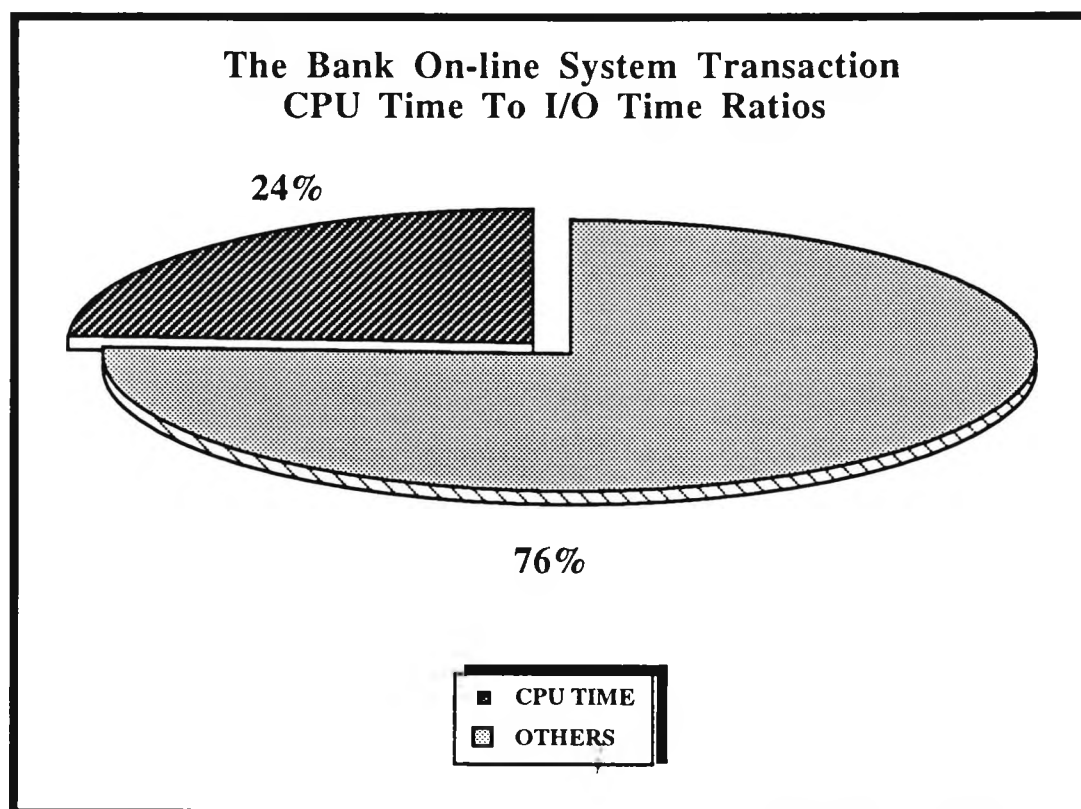


Fig. 4.18, The Bank On-line transaction ratios of CPU and I/O times

In addition, transaction behaviour included the examination of the number of I/Os generated by each on-line transaction to associate it with transaction I/O time. Fig. 4.18, illustrate the ratios of on-line transactions' CPU time to on-line transactions' I/O time. The average response time of the on-line systems was around the two seconds. On-line transactions spent around 24% of that time using the CPU and spent the rest of the transactions' response time, around 76%, on I/O operations. The bank on-line transactions produced around 39 I/O operations and generated about seven sort paths per transaction.

4.4.5.4 Number of Databases Accessed by One Transaction (JOIN Operation)

Most on-line transactions accessed only one database (94%) and when transactions performed JOIN operation they mainly accessed two databases (4%). JOINS that accessed three databases and four databases represented a small ratio, around 1% each, which shows that the bank computer centre avoided the JOIN operation because of its negative effect on system response time. The JOIN operation ratios are presented in Fig. 4.19.

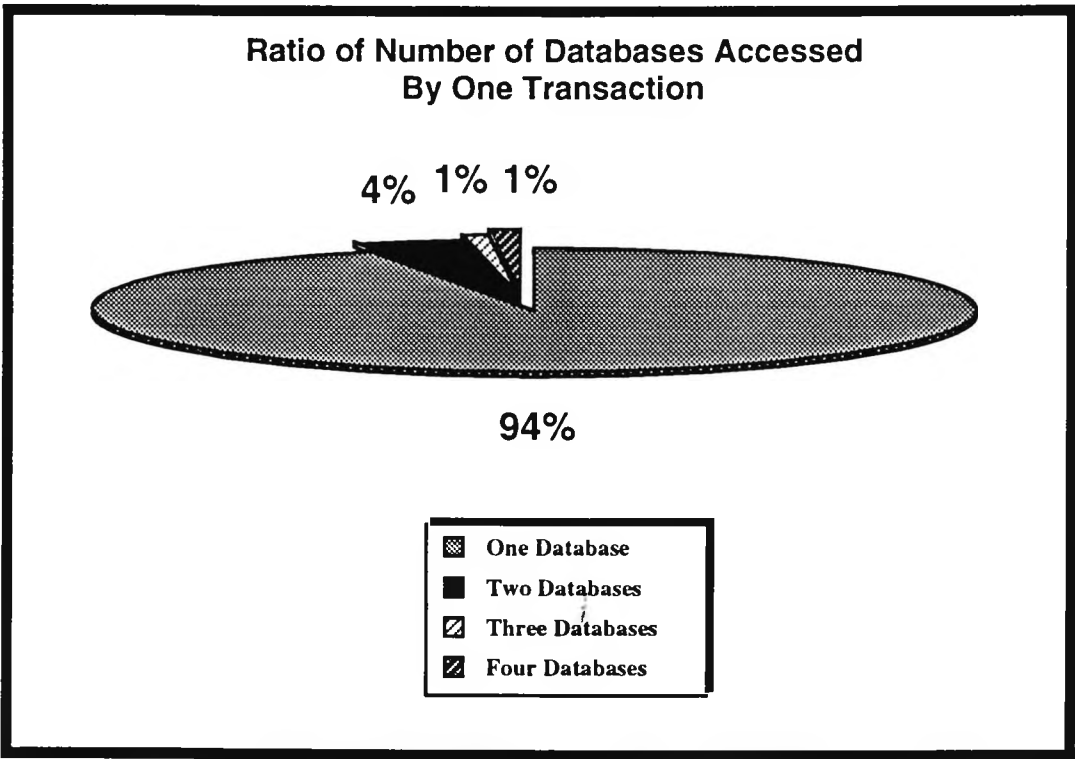


Fig. 4.19, The Bank On-line transaction JOIN ratios

4.4.5.5 Transactions Complexity (Nested Selects)

The number of nested selects per on-line transaction represents an index to transaction complexity. Fig. 4.20 illustrates the ratios of the number of nested selects per an on-line transaction. The majority of transactions consisted of one select, around 98%, and very small percentage, around 2%, consisted of two selects. The bank avoided using more than two selects to their negative affects on systems performance.

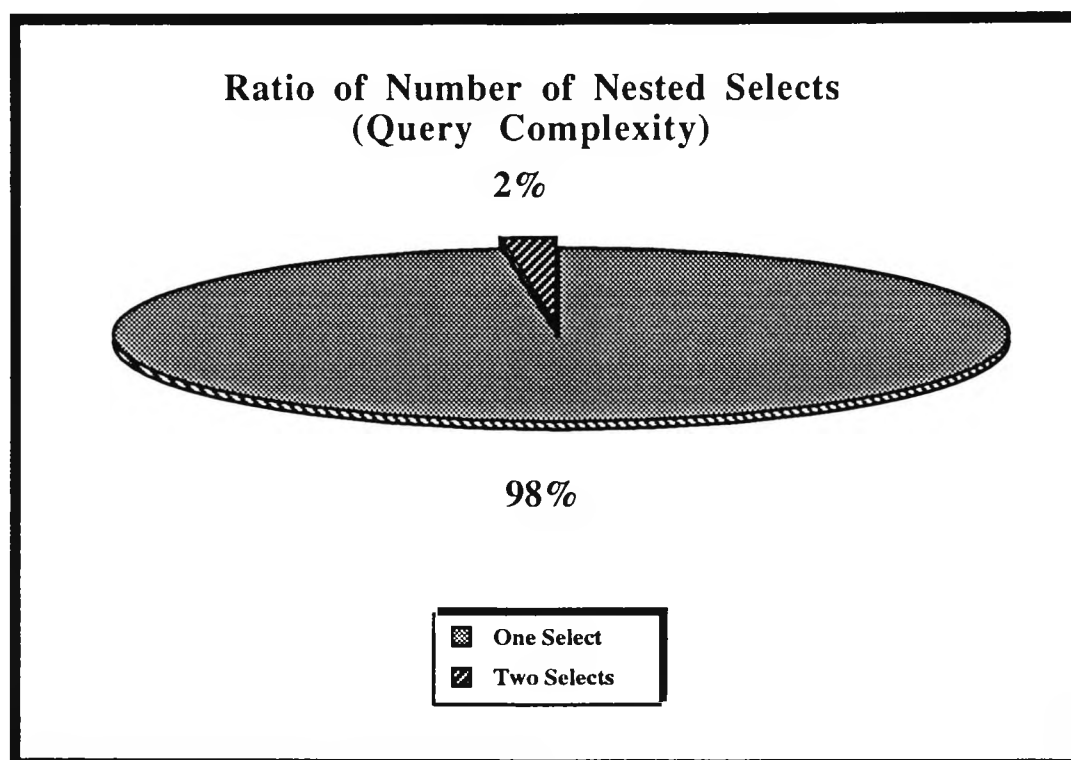


Fig. 4.20, The Bank On-line transaction nested selects ratios

4.4.5.6 Average Number of Retrieved Records per Transaction

The bank on-line transactions retrieved around four records per transaction on average. This small number of retrieved records per transaction was due to the high ratio of qualified retrievals that represented around 50% of the transactions' operations; qualified retrieval operation retrieved one transaction.

4.4.6 The Background Workload

The bank separated all the background workload on separate processors. At each data centre they dedicated some processors to on-line applications, and dedicated some other processors to off-line services. On-line processors performed one function that was supplying on-line services to on-line users and accessing on-line databases. The bank policy was to completely free the on-line processors to enhance on-line transactions' response time and maintained background workload of zero to the on-line service.

4.5 Discussion

The previous sections discussed the findings from three studies at three large organisations. This section will concentrate on the similarities between on-line transaction behaviour at the three organisations represented as the average number of database operations generated from each transaction. On-line transaction behaviour in this section is the result of investigating over 4800 different applications and examining around 40,000,000 on-line transactions that accessed around 5000 interactive databases.

This section also compares the TPC benchmark scripts, which are the database market standard, to on-line transactions' behaviour that was matching among the three studies, it explains why those benchmarks are not representative to high-volume transactions' environments. It also demonstrates that the TPC benchmarks differ from the environments they supposed to simulate and as a result they can be misleading when used to test environments similar to those discussed in this research.

4.5.1 On-line Transaction Behaviour Represented as Number of Database Operations

A two-way analysis of variance (ANOVA) was used to test the difference among the three organisations' patterns of behaviour. The observed value of F test for the between organisations (rows) was 0.003. Clearly this value is highly insignificant indicating virtually no difference among the three organisations, (the local authority, the airlines, and the bank), transaction behaviour. The effect of these results is that overall average over the three organisations is representative for each of the organisations and

this is true across the spectrum of the database operations. However, the analysis did reveal highly significant difference between the various database operations; qualified retrieval, insert, update delete, and sequential retrieval. In this case the between columns F test value was 47619 and the critical value at 99% was 7.01 with 4 and 8 degrees of freedom. Table 4.5 summarises transactions' patterns at the three organisations and Fig. 4.21 illustrates the calculated mean among the three organisations.

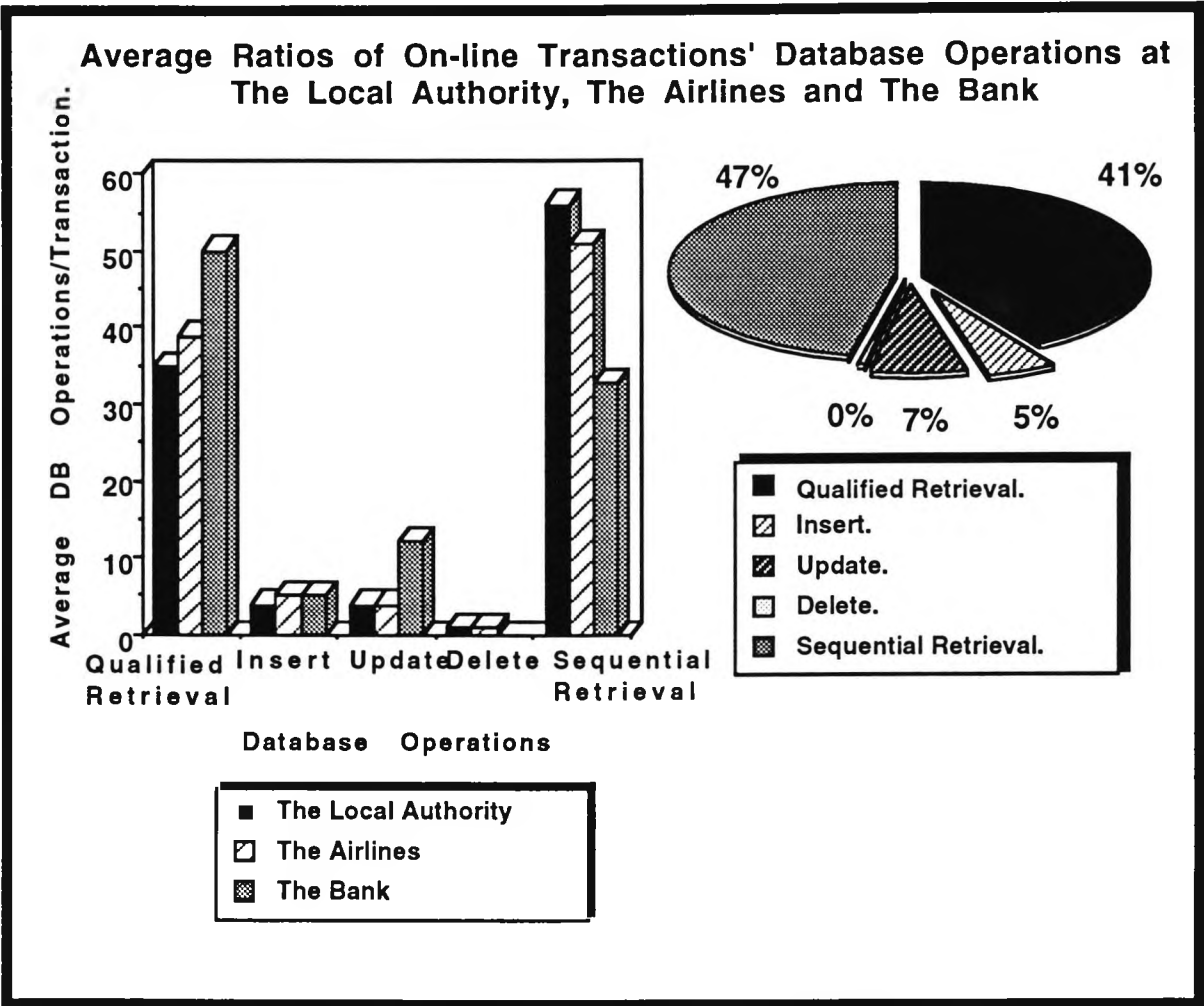


Fig. 4.21, Average of On-line Transaction Operations in all environments

Operation Name	Qualified Retrieval	Insert a Record	Update a Record	Delete a Record	Sequential Retrieval
The Local Authority	35%	4%	4%	1%	56%
The Airlines	39%	5%	4%	1%	51%
The Bank	50%	5%	12%	0%	33%
Average	41%	5%	7%	0%	47%

Table 4.5, Average of On-line Transaction Operations in all environments

4.5.2 Comparison Between TPC-A, TPC-B and On-line Transaction At the Three Organisations

This section compares the average number of database operations per on-line transaction at the three studied environments to the TPC-A and TPC-B benchmarks. Those benchmarks were selected because they are the database market standard and DBMS to be accepted it must be built around one of those benchmarks. Table 4.6, and Fig. 4.22, present a comparison between the ratios of database operations of the TPC-A benchmark, the TPC-B benchmark, and the average of the three organisations' on-line transaction.

The comparison shows that the TPC-A benchmark database operations are different from typical on-line transaction database operations. While sequential retrievals represent 88% of on-line transactions' operations at the studied environments, the TPC-A does not include qualified retrievals. The difference between the TPC-A database operations and typical on-line transaction database operations are large enough to ignore applying any statistical comparison between the two.

The TPC-B transaction database operations are not much different from the TPC-A transaction operations. The TPC-B transaction is a TPC-A transaction in addition one qualified retrieval. A comparison between the TPC-B benchmark transaction database operations and typical on-line transaction database operations shows that the TPC-B transaction operations are still widely different from a typical on-line transaction operation.

Operation Name	Qualified Retrieval	Insert a Record	Update a Record	Delete a Record	Sequential Retrieval
The TPC-A Transaction	0%	25%	75%	0%	0%
The TPC-B Transaction	20%	20%	60%	0%	0%
On-line Transaction	41%	5%	7%	0%	47%

Table 4.6, Comparison between the TPC-A, TPC-B and typical On-line transaction

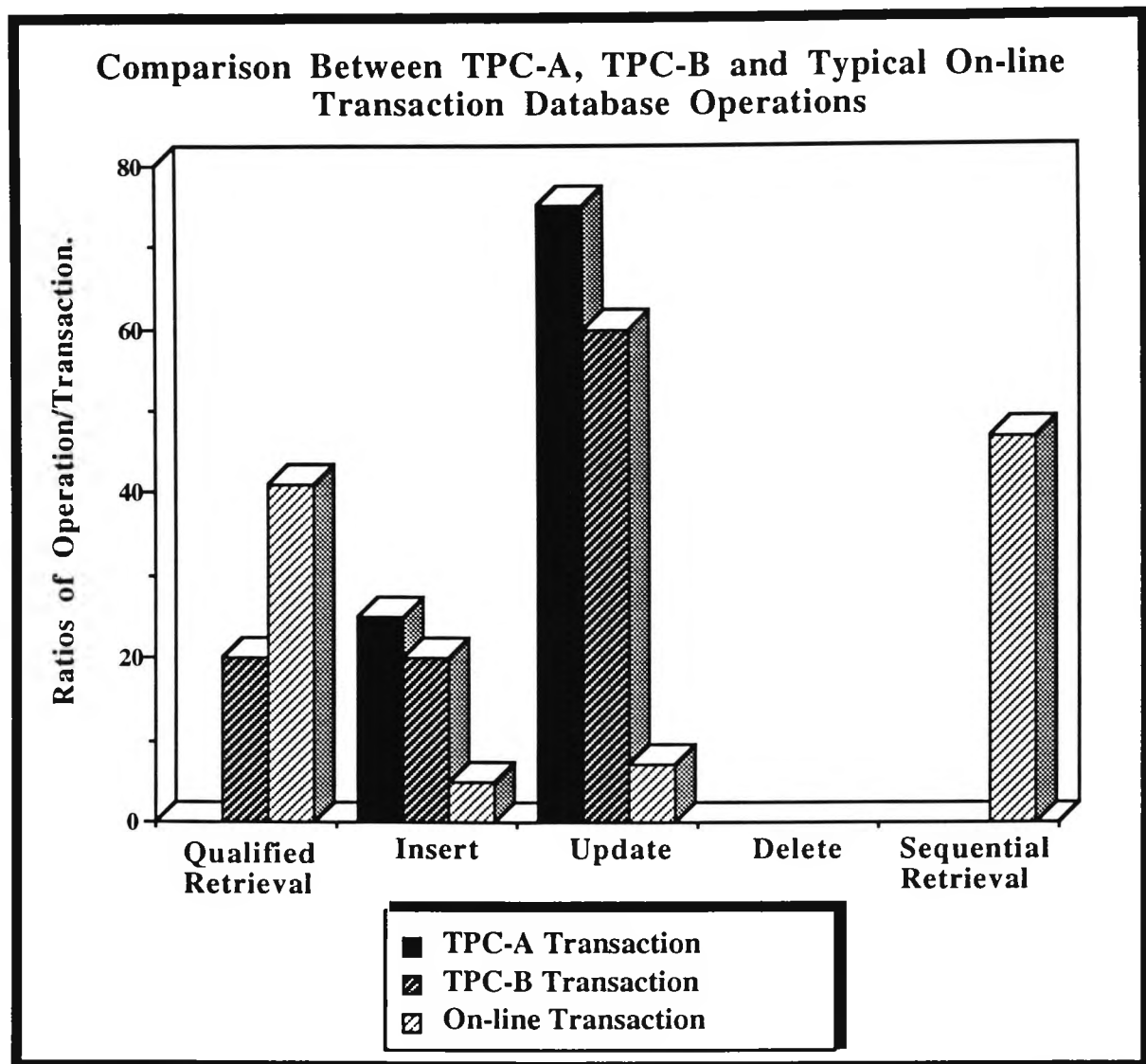


Fig. 4.22, Comparison between the TPC-A, TPC-B and On-line transaction

4.5.3 Comparison Between the Bank ATM Transaction and The TPC-A Benchmark

The TPC-A, the standard benchmark at the database market, simulates random withdrawals at a large bank environment. A transaction that is similar to an automatic teller machine (ATM) transaction. This section compares the TPC-A benchmark transaction database operations to the bank ATM transaction operations.

The findings from this study raise several points to discuss concerning TPC-A benchmark. These points can be summarised in the following:

1. TPC-A relies on three update operations and one insert.

2. From the analysis of ATM systems transactions the average of transaction operations have been presented in Table 4.7.

Operation Name	Qualified Retrieval	Insert a Record	Update a Record	Delete a Record	Sequential Retrieval
ATM System 1	27%	6%	24%	0%	42%
ATM System 2	24%	6%	20%	0%	50%
ATM Average	26%	6%	22%	0%	46%
TPC-A Transaction	0%	25%	75%	0%	0%

Table 4.7, Comparison between the TPC-A and ATM transaction

Table 4.7 shows that TPC-A benchmark database operations differs from the two ATM systems transaction operations presented in this work. The TPC-A benchmark hypothesis predicts a typical database transaction operation to be three update operations and one insert operation. In the mean time, the studied ATM transaction took a different pattern and consisted of more equally balanced transaction with more emphasis on sequential operations and qualified operations. The comparison does not require any rather statistical analysis techniques to show the difference between the two database operations. The result shows that the TPC-A benchmark database operations was different from the presented ATM systems.

3. While TPC-A consists of two database operations (UPDATE and INSERT), the ATM systems transaction operations took a different pattern among four database operations. TPC-A according to the previous analysis represents just 28% (22% Updates and 6% Insert) of the performed operations and neglects 72% of the performed transaction operations (46% Sequential retrieval and 26% Qualified retrieval). That means if during the design stage one relies on TPC-A to predict the expected performance of this system it will misrepresent the real system performance.

4.5.4 Comparison Between TPC-C and On-line Transaction At the Three Organisations

Comparing the TPC-C benchmark to a typical on-line transaction was rather difficult because the formal specification of implementing the benchmark is too expensive as the full benchmark consists of five programs two of them are batch

programs. Consequently, the TPC-C benchmark in this comparison was broken down to its basic database operations of the TPC-C transaction in two forms, the first was the full benchmark script and the second was the three on-line programs. Similar approach was presented in [LEUT93] where they tested a database model using the TPC-C basic database operations. The database operations of the full script of the TPC-C benchmark are presented in Fig. 4.23. The database operations of the on-line programs of the TPC-C benchmark are presented in Fig. 4.24. Table 4.8, presents a comparison between the ratios of database operations of the full script of the TPC-C benchmark, the on-line programs of the TPC-C benchmark and database operations of a typical on-line transaction.

Transaction Name	Qualified Retrieval	Insert a Row	Update a Row	Delete a Row	Sequential Retrieval	Functions
TPC-C Full Transaction	20%	4%	61%	10%	3%	2%
TPC-C On-line Transaction	33%	17%	39%	0%	11%	0%
On-line Transaction	41%	5%	7%	0%	47%	0%

Table 4.8, Comparison between the TPC-C and On-line transaction

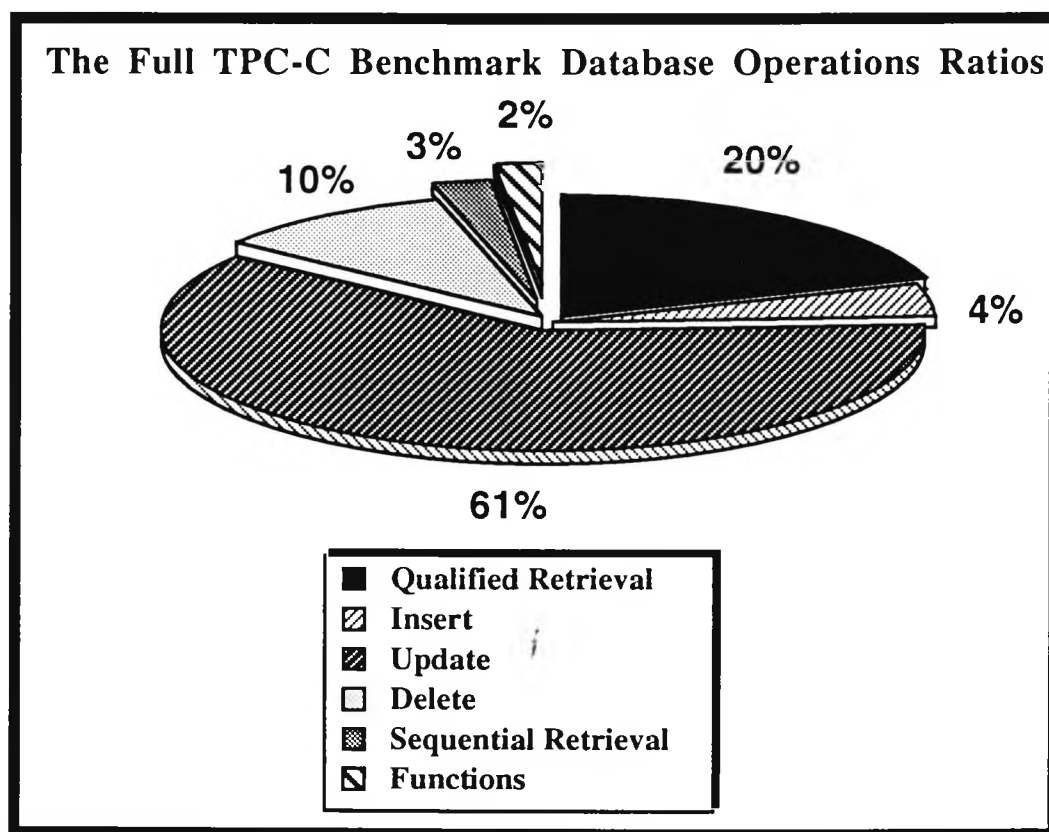


Fig. 4.23, The full TPC-C database operations

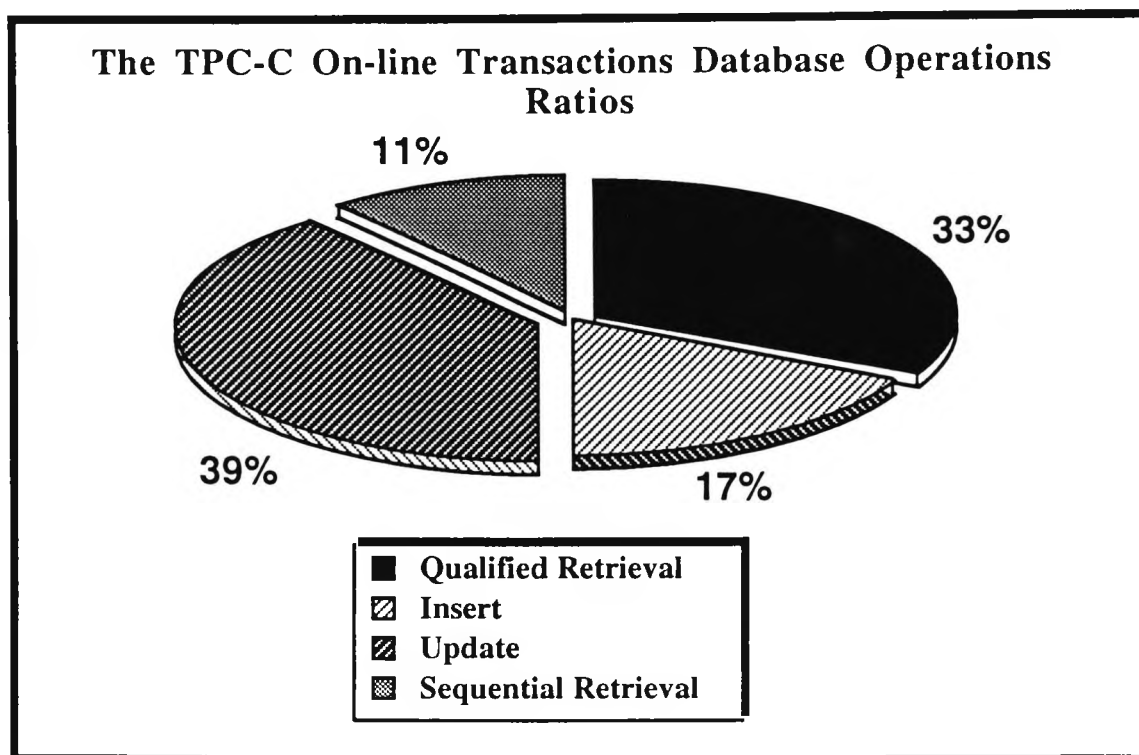


Fig. 4.24, The On-line TPC-C database operations

Fig. 4.25 illustrates a comparison between the database operations of the full script of the TPC-C benchmark and the database operations of a typical on-line transaction. The comparison shows that the full script of the TPC-C benchmark is dominated by database update operations where they represent more 65% (row update is more than 61% and row insert is more than 4%) of the TPC-C benchmark database operations.

Similarly, the on-line programs of the TPC-C benchmark are still dominated by database update operations where they represent more than 56% (row update is more than 39% and row insert is more than 17%). Fig. 4.26 illustrates the comparison between the database operations of the on-line programs of the TPC-C benchmark and the database operations of typical on-line transaction.

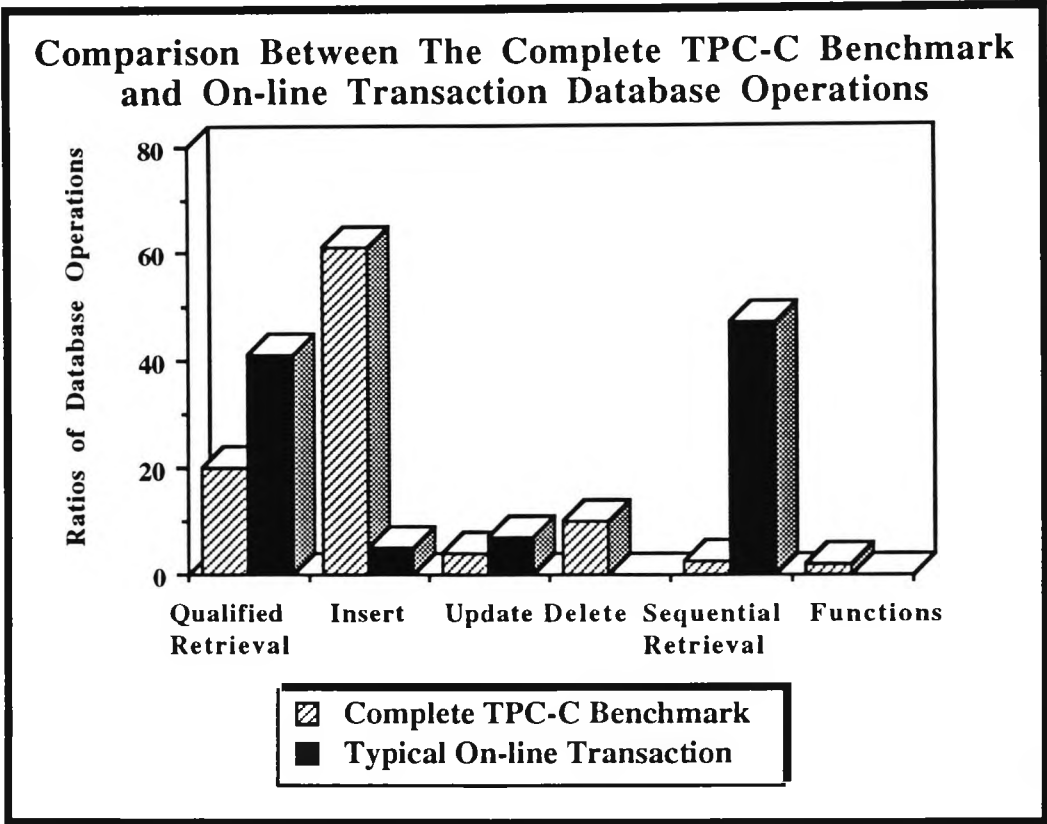


Fig. 4.25, Comparison between complete TPC-C and On-line transaction

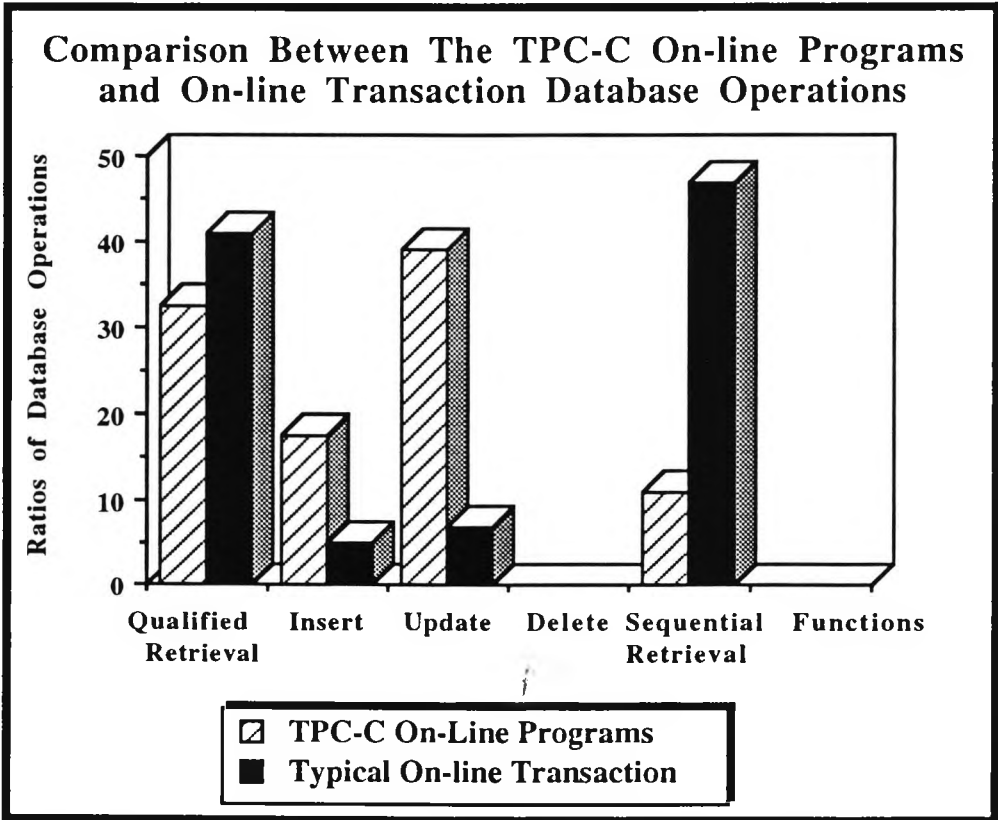


Fig. 4.26, Comparison between On-line TPC-C and On-line transaction

4.5.5 Common Characteristics Between Databases

There are certain characteristics that reflect the physical dimensions of the database. These include: the number of tables in a database; record size; number of attributes per record; and table size. Other characteristics such as data types and the distribution of the attribute values reflect the content of a database. In addition there are characteristics relating to access paths such as index structure. The studies examined those characteristics at the three environments to find the most common characteristics between them.

The three organisations shared several similarities in database characteristics. The similarities were the in following points:

- database size;
- record size;
- attributes types;
- number of indexed attributes;
- types of attributes;
- distribution of attributes.

4.5.5.1 Database Size

On-line database sizes at the three organisations were extremely large with space utilisation and number of records per database. Analysis of database size at the three organisations could not identify a typical database size that is common among them. Accordingly, benchmark design should take into consideration a methodology to test database environments under scalable loads.

4.5.5.2 Row Size

The three organisations had in common a record size that did not exceed 200 bytes. The local authorities used a record size of 185 bytes on average, the most common record size at the airlines' environment was around 170 bytes, and the average record size at the bank databases' environment was around 200 bytes. A benchmark database that uses a record length of around 200 bytes would be representative to the databases this work has examined.

4.5.5.3 Indexed Attribute and Attribute Types

The three organisations built indices to provide performance enhancement for retrievals and joins operations. Even if these operations were not the most prevalent, indices were still employed to enhance the overall system performance they were still used with insertions and update operations.

The most prevalent types of attributes at the three environments were integers, characters and fixed point attributes. Some other users may be willing to different attributes' types but the studied organisations required strict control over their arithmetic operations specially things like round off error.

A representative benchmark would use indexed database attributes and if it restricts itself the just integers, characters and fixed point data types it will be good enough to represent the studied organisations.

4.5.6 On-line Transactions Behaviour

Transaction behaviour followed a similar trend at the three organisations. The studies identified the behaviour of on-line transactions by investigating the following performance factors:

- JOIN Operation
- Transaction Complexity
- Number of Retrieved Records per Transaction
- On-line Transaction Ratios of CPU to I/O Utilisation
- Background Workload

4.5.6.1 JOIN Operation

This research investigated JOIN operation by studying the number of tables accessed per on-line transaction. The research could not study JOIN operation at the local authority computer centre because the databases at that environment were IMS DBMS databases that can not efficiently perform JOIN operations. At the airlines' computer centre and the bank computer centre they used relational DB2 DBMS that allows joining two or more tables in one transaction.

At both the airlines' environment and the bank environments the JOIN operation was avoided as much as they can. That was due to the negative effect of that operation on database performance. The findings from the two organisations were close enough to ignore statistical tests to check that difference. The 94% of on-line transactions accessed one table, a small percentage, 3%, accessed two tables and a similar percentage accessed three tables. Table 4.9 and Fig. 4.27 present the average number of tables accessed by one transaction at the airlines and the bank environments.

Number of Tables	One Table	Two Tables	Three Tables	Four Tables	Five Tables
The Airlines	93%	1%	5%	0.5%	0.5%
The Bank	94%	4%	1%	1%	0%
Average	94%	3%	3%	1%	0%

Table 4.9, Average number of JOINed tables

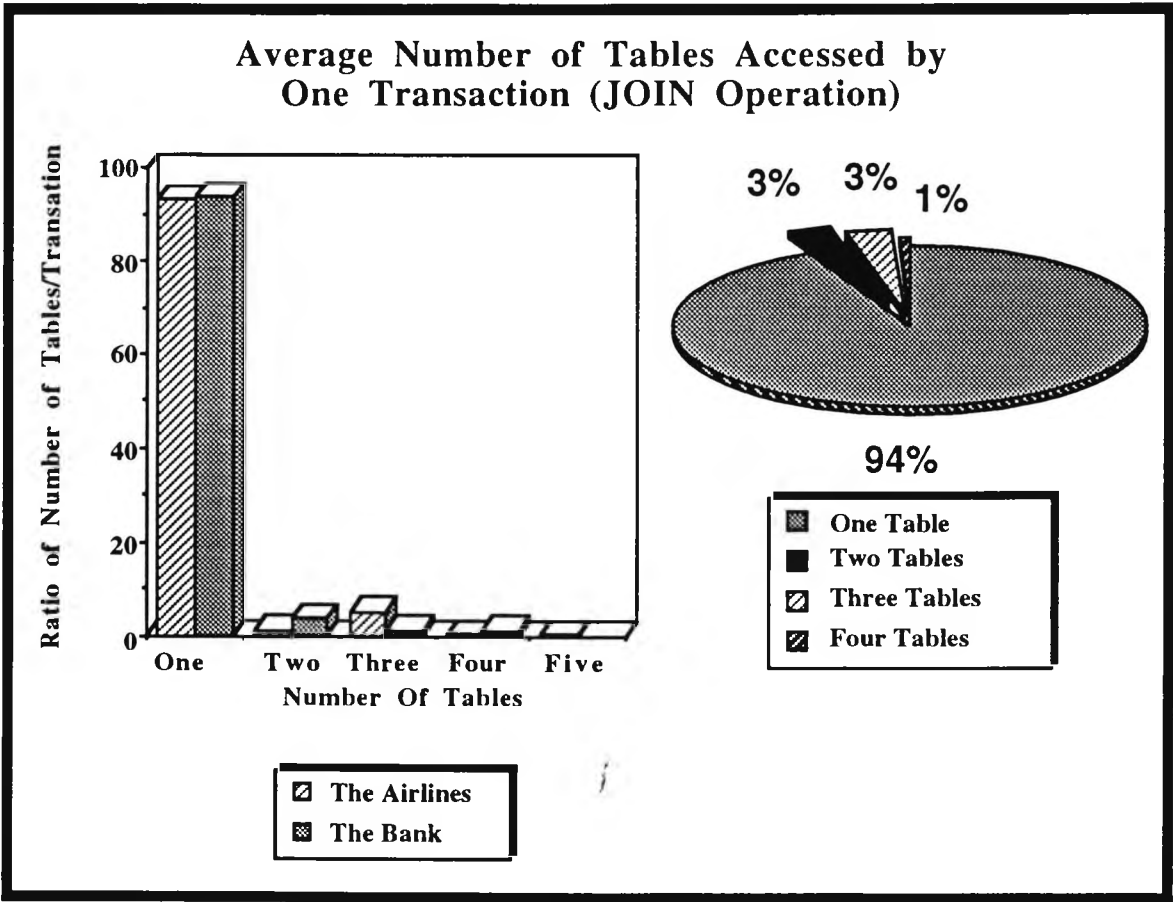


Fig. 4.27, Average number of JOINed tables

4.5.6.2 Transaction Complexity

JOIN operations might consist of one select or more than one select, the research investigated the most common form of JOIN operation. The research considered that transaction complexity increases by the increase of number of selects per transaction. The great majority of on-line transactions, around 98%, consisted of one Select and around 2% of on-line transactions consisted of two selects. Transactions that consisted of three and four selects represented a very small ratio that can be discarded. Transactions did not use more than four selects. Table 4.10 and Fig. 4.28 illustrate the ratios of transactions' complexity.

Number of Selects	One Select	Two Selects	Three Selects	Four Selects
The Airlines	98%	around 2%	less than 0.5%	less than 0.5%
The Bank	98%	2%	Too small	Too small
Average	98%	2%	0%	0%

Table 4.10, Average number of nested selects

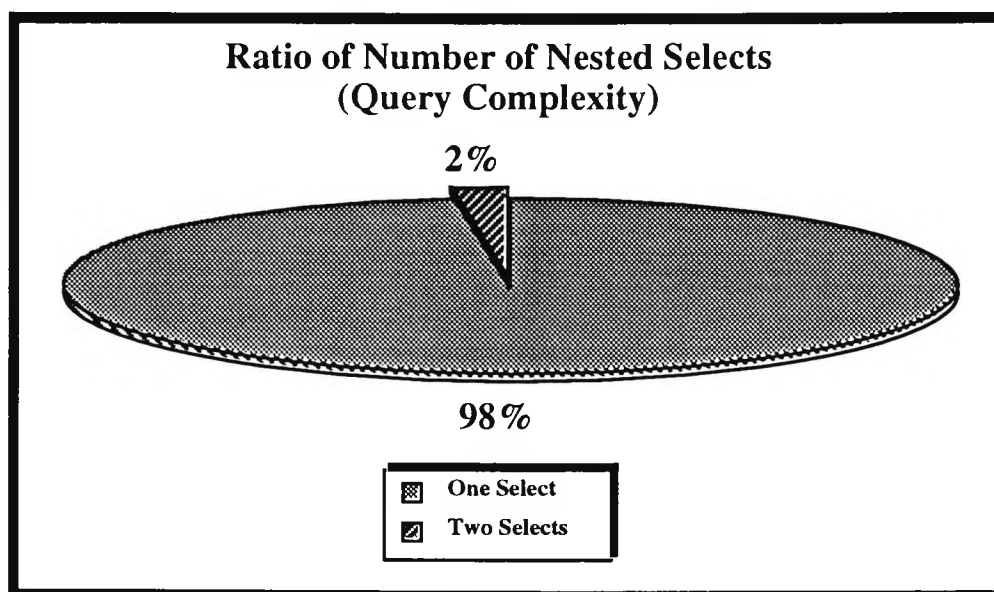


Fig. 4.28, Average number of nested selects

4.5.6.3 Number of Retrieved Records per Transaction

An average number of retrieved rows by on-line transaction was around seven rows. The Local Authorities on-line system transaction returns on average seven segments. The Airlines on-line system transaction returns on average 6.6 rows per

transaction. For the Bank on-line system transaction returns on average four rows per transaction.

4.5.6.4 On-line Transaction Ratios of CPU to I/O Utilisation

The research examined CPU to I/O utilisation because several benchmarks used them as the main hypothesis for their database operations (the most famous one was the Wisconsin Benchmark). A typical on-line transaction consumes around 25% of transaction response time using CPU and used about 75% of response time to perform I/O operations. Fig. 4.29 illustrates on-line transactions' ratios of CPU to I/O utilisation.

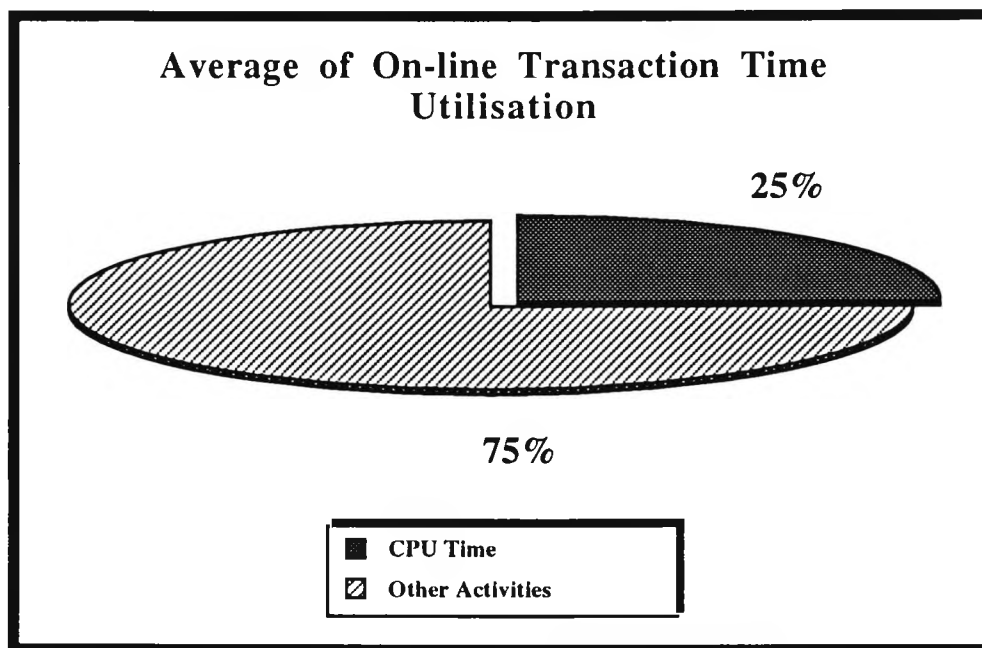


Fig. 4.29, Average of On-line transaction I/O time ratios

4.5.6.5 Background Workload

The three organisations maintained a very small background workload that is either zero or very close to zero. The local authority computer centre either separated all the off-line activities on the micro system or performed off-line jobs during night shifts when on-line service was not active. The airlines' computer centre had a processor to perform most of the off-line workload; in some rare cases the on-line processors performed part of that off-line load. The bank computer centre dedicated some

processors to on-line service and dedicated some other to off-line jobs and did not perform and off-line jobs on on-line processors. As we see, the three organisations' policies were to run on-line service without background workload to enhance on-line service performance.

4.6 Conclusion

This chapter presented the results from three studies at large database environments. The three organisations were different in many aspects including the kind services they supply and the nature of their applications. The three organisations were selected because of their large database environments, the wide variety of their applications and the exceptionally heavy transaction load. Each one of the three organisations represented a good example of high-volume transactions' environments. The studies aimed to identify the main characteristics of large databases at large database environments and identify a typical transaction behaviour at high-volume transaction environments.

Having discussed the common findings among the three organisations the main conclusion is the TPC benchmarks' transaction operations are widely different from the studied systems transactions' operations. As presented in section (§4.2.4.1) transaction database operations plays an important role when measuring database performance and DBMS behaviour varies according to the used database operations. Hence, the TPC benchmarks' database operations can be misleading if used to test environments similar to those presented in this chapter.

The second conclusion is performance was the most prevalent factor among the three organisations. The three organisations' hardware configurations and software tuning were employed to enhance on-line service performance.

The next chapter presents the CITY benchmark. The CITY benchmark mainly simulates a typical on-line transaction at large database environments. The benchmark is designed and built based on the findings presented in this chapter.

j

CHAPTER 5

The CITY BENCHMARK

CHAPTER 5

The CITY BENCHMARK

As benchmark results are representative of those types of transactions that are included in the benchmark set, it is impossible to generalise those results to all kinds of systems transactions. Research work has been conducted to identify the main characteristics of on-line databases by examining real database performance factors. By applying the analysis of those performance factors at several different environments, the data from that analysis can supplement the data from the live environments once the basis of this analysis has been defined to evaluate performance in a transaction processing environment.

In chapter four a discussion of the findings from three in-depth studies at large UK database environments were presented. The studies identified the main database characteristics at large database environments and identified on-line transactions' behaviour at high volume transactions' environments. This chapter presents the CITY benchmark. This benchmark is a methodology for evaluating database management systems' performance. The benchmark is based on examining over 4800 applications issuing around 40,000,000 transactions accessing over 5000 databases. The CITY benchmark is designed to simulate on-line transactions' behaviour at high-volume transactions' environments. As suggested by Gray [GRAY91], the CITY benchmark is characterised by being relevant, portable, scalable and simple.

5.1 Introduction

The Debit/Credit (TP/1) [ANON85] and the Transaction Processing Council (TPC) benchmarks [TPC 89, GRAY91] have become the database market standard practice. Those benchmarks have a number of practical limitations [REVE90]. Most of those limitations stem from the lack of background study before the benchmark design stage. The main criticism of the TPC benchmarks is that they do not provide a realistic characterisation that can be set as the target for the benchmarking process, and more than this, they do not even provide a model of the ATM systems whose performance they are supposed to simulate [REVE92b].

As benchmark results are representative of those types of transactions actually included in the benchmark set. It is impossible to generalise those results to all kinds of systems transactions. Research work has been conducted at City University to identify the main characteristics of on-line databases by examining real database performance factors. Those factors have been discussed by Ferrari [FERR78, FERR83], Dongara [DONG87], and Hawng [HAWN87]. By applying the analysis of those performance factors at several different environments, the data from that analysis can supplement the data from the live environments once the basis of this analysis has been defined to evaluate performance in a transaction processing environment.

This chapter presents the CITY benchmark [REVE93]. The benchmark design is based on empirical studies in large database environments. The studies examined over 4800 different applications and investigated over 40,000,000 on-line transactions accessing around 5000 on-line databases. The study discovered a common pattern between the studied systems, despite the difference in activities and nature of the organisations concerned. The CITY benchmark transactions are derived from the findings from those studies.

The CITY benchmark is a comprehensive methodology that rigorously tests DBMS behaviour. The CITY benchmark design is based on in-depth empirical studies at large database environments and took into consideration the critiques directed towards the TPC benchmarks. Researchers such as Gray [GRAY91], and Ferrari [FERR78], have established some guidelines for evaluating systems performance. The most important guideline is, the benchmark should be independent of the application or the architecture involved. Ferrari and Gray [FERR78, ALLE89, GRAY91], recommends that a benchmark should be characterised by the following.

1. Relevant: represents the tested domain.
2. Portable: the benchmark will be easy to implement on many different systems and architectures.
3. Scalable: the benchmark should apply to small and large computer systems.
4. Simple: the benchmark should be understandable and well documented.
This will allow the exact replication/of the original benchmark.
5. Its results should be acceptable by all systems.

The CITY benchmark workload simulates the exact behaviour of real database workloads that were investigated and presented in the previous chapter. The main advantage of the CITY benchmark is being representative of high volume transaction

domain. Representative is defined as the accuracy of the benchmark in reflecting the characteristics of real transactions at real environments. The benchmark script is a direct mapping of on-line transaction operations at large database environments and models a typical workload of high volume transaction environments. Due to the benchmark portability, it is applicable to a wide range of database management systems and architecture. Because it is based on studying large number of real databases and applications, the benchmark results are representative of a wide variety of database management systems and database applications. The benchmark is machine independent, database management system independent, application independent and characterised by being: representative; reproducible; system independent; simple to construct; and low in usage cost.

The chapter begins with characterisation of performance factors and metrics that are relevant to relational database systems, then it describes the methodology used in the CITY benchmark. Section two presents the CITY benchmark domain specification. Section three demonstrates the CITY benchmark objectives. Section four discusses the CITY benchmark metrics. Section five describes the CITY benchmark database. Section six describes the CITY benchmark transaction script. Section seven presents the requirements of transaction processing council. Section eight presents the chapter conclusion.

5.2 The CITY Benchmark Domain Specification

In comparative benchmarking, the first problem to be addressed is domain specification. The question is always which domain this benchmark can represent. The CITY benchmark is designed to represent the domain of high volume transactions' environments. Environments similar to those presented in the previous chapter. The benchmark will test database management systems that support the data types and functions specified in the ANSI SQL. Hence, besides the operations specified in relational calculus, the benchmark includes update and insert operations. The CITY benchmark is designed to address issues of quantity, not of quality, the system throughput and response time of a transaction are a question of quantity. Therefore functions provided by a database management system such as the support of a certain data type is not tested and because not all systems provide the complete functionality specified in the standard ANSI SQL, the benchmark has no penalty for this incompleteness. Furthermore, the CITY benchmark does not provide a test to evaluate the performance of a system on different data types.

5.3 The CITY Benchmark Objectives

The CITY benchmark is the first benchmark designed as a result of studying several large database systems at large UK organisations, regardless of the organisations' nature and activities. The CITY benchmark mainly tests and compares database systems performance at high volume transaction environments. Additionally, the benchmark can serve more than one purpose and achieves more than one objective. It can achieve the following objectives:

1. evaluate the performance of a database management system;
2. compare different database management systems;
3. evaluate the impact of system modifications such as a new version of a database management system;
4. evaluate the effect of system parameters, such as buffer size;
5. examine the fit of a performance model to the system modelled.

5.4 The CITY Benchmark Metrics

Ferrari [FERR78], identifies a performance metric as:

"A descriptor that is used to represent a system performance or some of its aspects"

Different benchmarks addressed different metrics. For example some benchmarks used virtual memory requirements for using a system. Some other benchmarks used disk space utilisation and measured space overhead. A third type of benchmarks uses the time and resources required to set up the database. Ferrari [FERR83], included some other database performance indices such as:

- global CPU utilisation by DB/DC system;
- CPU utilisation for the various control modules;
- CPU utilisation for the query language;
- CPU utilisation for individual transactions of on-line users and for those input by application programs;
- I/O requests to each database divided by transaction type;
- number of accesses to the database during a given time period;
- average time the requests spend in the I/O queues;

- CPU overhead due to buffer handler;
- memory space demanded by each transaction.

5.4.1 Response Time and Transactions per Second

Generally, in benchmarks, transaction response time and transaction throughput are the most frequently used performance metrics for database systems. Transaction response time is the time elapsed between submission of a transaction to DBMS and receiving the result. Transaction throughput is the average number of transactions completed per unit of time.

There might be different variations in defining transaction response time in various benchmarks. It must be specified whether transaction response time includes transaction compilation time as pre-compiled queries may be submitted. Queries may be pre-compiled with variables that are given values as arguments when the transaction is submitted. The advantage of pre-compilation is that transaction parsing and optimisation is performed once but the transaction can be executed many times with a response time that excludes parsing and optimisation. However, pre-compilation is of limited utility with range queries, as they can not be effectively optimised until the constant(s) in the range predicate are specified. In addition, any selection predicate on a highly skewed distribution may be improperly optimised in pre-compilation if the selection predicate is pre-compiled with a variable to be specified at run time.

A distinction must be made between the time it takes for the first result row to be received to the time it takes for all the result rows to be received. In [YAO 87], these response times are called time-to-first and time-to-last, respectively. This distinction is necessary for two reasons. First, some systems sent out the first result row before the entire result is computed. Hence, the time-to-first can not necessarily be used to determine the time required for a system to perform a relational operation. Second, time-to-last includes the time required to format the result and the time required to transmit the result to its final destination. Hence, this time varies greatly with the size of the result and can not necessarily be used to determine the efficiency with which a system performs relational operations.

The utilisation of components of a system may also be measured. The utilisation of resources is the percentage of time it is kept busy. Disk, CPU, and communication line utilisation figures can be used to identify bottlenecks in systems. Performance models of database systems frequently concentrate on the amount of CPU time and the number of disk accesses required to execute a query. With CPU time, a distinction is

5. The CITY Benchmark.

frequently made between CPU time used by the operating system in support of a user, and CPU time that is charged directly to the user. This distinction is important because the former is often more sensitive to system load than the latter [BITT86]. Disk accesses are measured in disk reads, and disk writes. In addition, paging by virtual memory manager may have an effect [BITT87].

The CITY benchmark metric is the system throughput. The throughput will be measured as the number of transactions performed per second under specified load. Number of transactions per second (TPS) are using tables of the sizes that are presented in Table 5.1.

Table Name	Table Size
DB100	1,000,000 rows
DB200	1,000,000 rows
DB300	1,000,000 rows
DBUPD	1,000,000 rows
DBINS	should be large enough to accommodate 1,000,000 rows by the end of the run

Table 5.1, The CITY benchmark TP's tables' sizes

A system than runs its transactions in less than or equal to one second, is one CITY transaction per second (TPS) system.

A system than runs its transactions in less than or equal to one minute, is one CITY transaction per minute (TPM) system.

The previous sizes were selected based on the empirical studies presented in chapter four. Through the interviews that were conducted with the top management and database administrators in those organisations they were not interested in any database size less than one million rows. Additionally, when their running systems were investigated, the running systems were accessing on-line tables that contain millions of rows and therefore any database size less one million rows will not be realistic to represent those environments.

Response time against a scalable load could also be used as another performance metric. The database, when required, can calculate systems' response time against

specified load that can be scaled up by the benchmark users. The advantage of this metric is giving the user a clear view of the system expected behaviour under any required load. The benchmark uses response time because the response time figure is usually of more interest to on-line systems users and neglecting response time is neglecting an important user requirement.

5.4.2 How Time is Measured

To guarantee consistency of the benchmark measures, each time measure is calculated several times within a loop. Within each loop the benchmark transaction operations are repeated as many times as the loop time allows, those loops are called time loops. Each time loop starts by getting the duration of the benchmark loops. The program uses two different time variables for both response time and transactions per second measurements. Two time variables were used to gain more control on the measured time and keep it as accurate as possible.

The computer clock time is then obtained to get the loop start time. The loop duration is then added to the start time to decide the loop end. The benchmark transaction operations are executed within this time loop until the allowed test time is reached. Then transaction response time is calculated as $(\sum \text{Time of Loop}_i) / \text{number of loops}$. Transactions per seconds are calculated by dividing number of completed transactions by the total test time. Discussion of loop duration and time controls are presented in chapter 6 section (§6.2). The loop and the process of response time and transactions per seconds calculation is as follows:

```
start= time(&g_time1); /* Get system elapsed time in seconds. */
end= start + SECONDS; /* Calculate end time in seconds */
do
{ /* start of do loop body. */

/* ***** */
/* Get start time to calculate response time per loop */
/* ***** */

time(&resp_time_start);

THE TEST TRANSACTIONS      ;

/* ***** */
/* Get end time to calculate response time. */
/* ***** */

time(&resp_time_stop);
resp_time_1 = difftime(resp_time_stop, resp_time_start);
```

```
resp_time_t = resp_time_t + resp_time_1;

} /* end of do loop body. */
while (time(&g_time1) < end);

Transaction_resp_time = resp_time_t / (double) trans_count;
Transaction_per_sec   = (float) trans_count / (float) SECONDS;
```

5.5 The CITY Benchmark Database

There are certain characteristics that reflect the physical characteristics of the database. These include the number of tables in a database, record size, number of attributes per record, and table size. Also there are characteristics that reflect the content of a database, these are data types of the attribute values and the distribution of the attribute values. In addition there are characteristics relating to access paths such as index structure.

The CITY benchmark tables are a reflection to database characteristics that were found from studying over 5000 databases. The CITY database represent typical real life characteristics and simulate the structure of the database. The number of relations in the database is fixed. Record size and indexed attributes are based on the empirical studies. Number of attributes per record and data types are based on further analysis of attributes number and types effect. The benchmark database size is scalable. The structure of the database that includes factors such as the following.

1. Database tables represented as:
 - the number of tables;
 - the relation between tables (database schema)
 - database attributes and their dependencies;
 - the degree of normalisation of the database tables.
2. Tables attributes represented as:
 - the tables and their attribute types;
 - correlation between attributes;
 - distribution of individual attribute types;
 - joint distributions of attributes.
3. Size of the database tables represented in number of rows per table and table size in bytes.
4. Accessibility of the database in terms of indices.

Tables are generated by a database generator program that is part of the benchmark structure. The database generator allows some flexibility in regard to database size. The benchmark allows the user to generate variable database sizes depending on the computer and database management system available. In the mean time, the database model, attribute types and values, and record sizes can not be changed by the user.

Data types in the CITY benchmark are integers, characters, decimal , and variable character. But, as presented later, there is no statistically significant difference between using different types of attributes, though all the enquiries are issued to integer keys.

One of the puzzling factors was the data distribution in the tables. In our contention that the important factors in performance are the volume of relevant data and the volume of participating data. The volume of relevant data is the amount of data which must be accessed to process a transaction. The volume of participating data is the volume of data which participates in the answer to a query. The volume of relevant data determine the I/O workload. The volume of relevant , together with the number of number of records per block, determine a component of processor workload. The additional component of processor workload is determined by the volume of participating data and the number of records per block. Clearly the performance indices will vary as the ratio of these two factors vary. Rather than provide facilities to generate values according to different distributions, we provide the facility to generate values according to discrete distributions in intervals of 10 records. It was sufficient for benchmarking purpose to use only uniform distributions. The volume of relevant data is not a factor to control in this work but the volume of participating data is decided depending on the findings from the empirical studies.

5.5.1 Number of Tables

The CITY benchmark database consists of five tables. Those tables are divided into three retrieval tables; one table for table updates and one table for row insertions. Update operations were separated on a different table to protect retrieval tables from the effect of attributes changes. The CITY benchmark consists of the following five tables:

- Retrieval tables:
 - DB100
 - DB200
 - DB300

- Update table:
 - DBUPD
- Insert table:
 - DBINS

5.5.2 Row Size

The CITY benchmark tables have a row size around the 200 bytes. As presented in the previous chapters, row width at the three organisations varied from 100 bytes to 300 bytes with most common row size of 200 bytes. The CITY benchmark tables row width are 100 bytes, 200 bytes and 300 bytes with 200 bytes as the most common row length which is typical to what has been found in the studied environments. The benchmark tables' sizes are as follows.

1. DB100, is the first enquiry table. The row length of this table is 100 bytes.
2. DB200, is the second enquiry table. The row length of this table is 200 bytes.
3. DB300, is the third enquiry table. The row length of this table is 300 bytes.
4. DBINS, is a table created for Insert operation. The row length of this table is 200 bytes.
5. DBUPD, is a table created for Update operation. The row length of this table is 200 bytes.

5.5.3 Attributes' Types and Accessibility of Rows (Indices)

Different database management systems support wide variety of data types, those data types could be any of the following:

1. Character.
2. Variable character.
3. Internal numeric (Decimal).
4. 8.bit fixed point.
5. 16.bit fixed point.
6. 32.bit fixed point.
7. 32.bit floating point.
8. 64.bit floating point.
9. Null terminated strings.

10. Packed decimal.
11. Long data types.
12. Internal date.
13. Julian date.

The present author conducted an empirical study to test the effect of unique key attribute type on transaction response time. The effect of three attributes' types, (integers, decimal, characters, and variable characters) were examined. The integer field width was four bytes, the decimal field width was eight bytes, character field width was eight bytes and variable character field width was eight bytes. The study took place at two environments, a PC environment running relational DBMS and SUN SPARC environment running relational DBMS. At the PC environment, the difference between integer and decimal key attributes was small enough to be truncated during its calculation and the same went for character and variable character key attributes. Integer and decimal key attributes gave a slightly better response time than character and variable character key attributes. The difference between integer key response time and character key response time was around .02 of the second. The same behaviour was found at the SUN SPARC environment, the response time was in favour of integer key attribute with a very small value of around 0.003. Those findings were in accord with other empirical studies conducted by [HAWN87 and McCA92a].

The empirical studies found that the majority of the examined transactions used a decimal key attribute which was common between them all. In addition, the most common attributes' types for other data fields were integers, characters decimal and float attributes. Based on those findings, the CITY benchmark rows consisted of the following attributes' types with no for the absence of any one of them.

1. Integer.
2. Char
3. Short
4. Long
5. Real
6. Double
7. VARCHAR

The CITY benchmark rows look as follows:

1. DB100 is an enquiry table. It has a row length of 100 bytes and one decimal unique key (db1f11). The table fields are as follows:

```
db1fc1 char(8),          /* No-key field. */
db1fi1 number(8) not null, /* Unique Key using unique index. */
db1fc2 char(30),         /* No-key field. */
db1fl1 number(4),        /* No-key field. */
db1fr1 number(4,2),      /* No-key field. */
db1fd1 number(6,3),      /* No-key field. */
db1fv1 char(40));       /* No-key field. */
```

2. DB200 is the second enquiry table. It has a row length of 200 bytes. The table has two key fields a unique key (db2fi1) and a secondary key (db2fi2). The table fields are as follows:

```
db2fc1 char(8),          /* No-key field. */
db2fi1 number(8) not null, /* Unique Key using unique index. */
db2fc2 char(30),         /* No-key field. */
db2fl1 number(4),        /* No-key field. */
db2fr1 number(4,2),      /* No-key field. */
db2fd1 number(6,3),      /* No-key field. */
db2fv1 char(40),         /* No-key field. */
db2fc3 char(8),          /* No-key field. */
db2fi2 number(8) not null, /* Secondary Key 10 times. */
db2fc4 char(30),         /* No-key field. */
db2fl2 number(4),        /* No-key field. */
db2fr2 number(4,2),      /* No-key field. */
db2fd2 number(6,3),      /* No-key field. */
db2fv2 char(40));       /* No-key field. */
```

3. DB300 is the third enquiry table. It has a row length of 300 bytes. The table has three key fields a unique key (db3fi1) and two secondary keys (db3fi2 and db3fi3). The DB300 table fields are as follows:

```
db3fc1 char(8),          /* No-key field. */
db3fi1 number(8) not null, /* Unique Key using unique index. */
db3fc2 char(30),         /* No-key field. */
db3fl1 number(4),        /* No-key field. */
db3fr1 number(4,2),      /* No-key field. */
db3fd1 number(6,3),      /* No-key field. */
db3fv1 char(40),         /* No-key field. */
db3fc3 char(8),          /* No-key field. */
db3fi2 number(8) not null, /* Secondary Key 10 times. */
db3fc4 char(30),         /* No-key field. */
db3fl2 number(4),        /* No-key field. */
db3fr2 number(4,2),      /* No-key field. */
db3fd2 number(6,3),      /* No-key field. */
db3fv2 char(40),         /* No-key field. */
db3fc5 char(8),          /* No-key field. */
db3fi3 number(8) not null, /* Secondary Key 100 times. */
db3fc6 char(30),         /* No-key field. */
db3fl3 number(4),        /* No-key field. */
db3fr3 number(4,2),      /* No-key field. */
```

```
db3fd3  number(6,3),      /* No-key field. */
db3fv3  char(40));        /* No-key field. */
```

4. DBINS, is a table created for Insert operation. The table row length is 200 bytes. The table has one secondary key field that allows repeated insertions in the table using the same key value. The secondary key field is (dbifi1) which is a decimal key. The table fields are as follows:

```
dbifc1  char(8),          /* No-key field. */
dbifi1  number(8) not null, /* Secondary Key */
dbifc2  char(30),         /* No-key field. */
dbifl1  number(4),        /* No-key field. */
dbifr1  number(4,2),      /* No-key field. */
dbifd1  number(6,3),      /* No-key field. */
dbifv1  char(40),         /* No-key field. */
dbifc3  char(8),          /* No-key field. */
dbifi2  number(8) not null, /* No-key field. */
dbifc4  char(30),         /* No-key field. */
dbifl2  number(4),        /* No-key field. */
dbifr2  number(4,2),      /* No-key field. */
dbifd2  number(6,3),      /* No-key field. */
dbifv2  char(40));        /* No-key field. */
```

5. DBUPD, is an Update dedicated table. The table row length is 200 bytes. The tables has one decimal unique key (dbufi1) and one decimal secondary key (dbufi2). The row layout is as follows:

```
dbufc1  char(8),          /* No-key field. */
dbufi1  number(8) not null, /* Unique Key using unique index. */
dbufc2  char(30),         /* No-key field. */
dbufl1  number(4),        /* No-key field. */
dbufr1  number(4,2),      /* No-key field. */
dbufd1  number(6,3),      /* No-key field. */
dbufv1  char(40),         /* No-key field. */
dbufc3  char(8),          /* No-key field. */
dbufi2  number(8) not null, /* Secondary Key. */
dbufc4  char(30),         /* No-key field. */
dbufl2  number(4),        /* No-key field. */
dbufr2  number(4,2),      /* No-key field. */
dbufd2  number(6,3),      /* No-key field. */
dbufv2  char(40));        /* No-key field. */
```

As presented, all the CITY benchmark tables are indexed, and rows are always accessed through either unique index or secondary index. The benchmark does not allow a non indexed operation through its test. This is similar to the findings presented in the previous chapter where as demonstrated, all on-line databases at the three

organisations used indexed attributes due to their dramatic effect on transaction performance.

5.5.4 Distribution of Attributes in the Tables

The attributes are randomly distributed in the CITY benchmark database tables by applying a random attributes selection process. The attributes are stored in a file. The file is accessed by the data loading table to select random values for the tables attributes. The random pattern of attributes selection is decided by a random number generator program. For testing reasons, a modified version of the C language compiler random number generator was used to generate random numbers within required limits.

The key values are uniformly distributed from a sequential number generator before rows insertions in the CITY database. The uniform distribution of the key values is similar to real life situations where databases are frequently reorganised either over every night shift or every weekend at most. Databases reorganisation process results in having uniformly distributed database key attributes after each reorganisation.

5.5.5 The CITY Benchmark Database Schema and Creation Rules

The creation of the CITY benchmark database tables are based on non normalised data, but all the attributes in each table depend on and can be retrieved through a unique key which is the main key in each row. Additionally, DB200 row has a secondary index and DB300 row has two secondary rows. The CITY benchmark database tables are connected to each through the following links:

- a unique key value to unique key value that establishes a one to one relationship;
- a unique key value to secondary key value that establishes a one to many relationship;
- a secondary key value to secondary key value that establishes a many to many relationship.

There is an established relation between the four tables, DB100, DB200, DB300, and DBUPD. All of them have an identical unique key when the database is completely created. The rows insertions table, DBINS, uses a secondary index because it contains

duplicates of the same key value. The DBINS is not connected to the other four tables. The tables creation process takes into account the following steps:

1. Because the table DB100 has just one indexed attribute, unique index, this table has no duplicates and all the rows in this table are used in just qualified retrieval operation.
2. For every unique key value inserted in the table DB100 three other unique key values are inserted in three tables: DB200, DB300, and DBUPD. That allows qualified retrieval operation that retrieves one row per key to be performed using any of the retrievals tables and allows qualified update that updates one row to be applied on DBUPD.
3. For every unique key value inserted in the table DB100 ten secondary key values are inserted in three tables: DB200, DB300, and DBUPD. That allows sequential retrieval operations from the tables DB200 and DB300 using one key value that retrieves ten rows. It also allows sequential update operation that updates ten rows using one key value to be applied on the row update table DBUPD.
4. For every unique key value inserted in the table DB100 hundred secondary key values are inserted in the table DB300 secondary index. This step established a relation between the tables DB200 and DB300 for each ten rows having the same secondary key value in DB200 their exit hundred rows having the same key value in DB300. This relation between the tables DB200 and DB300 permits the application of a JOIN operation between the two tables that retrieves one hundred rows.
5. The rows insertion table DBINS starts by zero value when the database are first created. Then for each transaction path a row is inserted in the DBINS table. This process results in increasing load on the tested system as the benchmarking process goes on.

Fig. 5.1 illustrates the relations between previous tables. The five tables will look as presented in tables 5.2, 5.3, 5.4 and 5.5 after inserting one hundred and one rows in DB100 with data values. DBINS table will be empty.

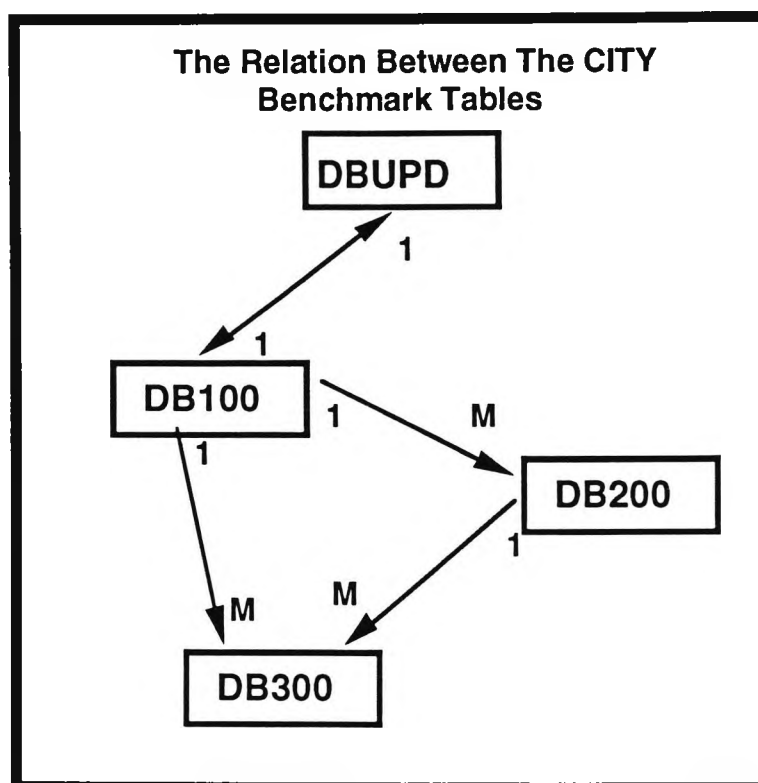


Fig. 5.1, The relations between the CITY benchmark tables

Unique Key	Row Values
00000001	* ----- random values ----- *
00000002	* ----- random values ----- *
00000003	* ----- random values ----- *
00000004	* ----- random values ----- *
00000005	* ----- random values ----- *
00000006	* ----- random values ----- *
00000007	* ----- random values ----- *
00000008	* ----- random values ----- *
00000009	* ----- random values ----- *
00000010	* ----- random values ----- *
.....	* *
.....	* *
00000099	* ----- random values ----- *
00000100	* ----- random values ----- *
00000101	* ----- random values ----- *

Table 5.2, DB100

Unique Key	Secondary Key	Row Values
00000001	00000001	* ----- random values ----- *
00000002	00000001	* ----- random values ----- *
00000003	00000001	* ----- random values ----- *
00000004	00000001	* ----- random values ----- *
00000005	00000001	* ----- random values ----- *
00000006	00000001	* ----- random values ----- *
00000007	00000001	* ----- random values ----- *
00000008	00000001	* ----- random values ----- *
00000009	00000001	* ----- random values ----- *
00000010	00000001	* ----- random values ----- *
.....		* *
.....		* *
00000099	00000010	* ----- random values ----- *
00000100	00000010	* ----- random values ----- *
00000101	00000011	* ----- random values ----- *

Table 5.3, DB200

Unique Key	Secondary Key-1	Secondary Key-2	Row Values
00000001	00000001	00000001	* -- random values -- *
00000002	00000001	00000001	* -- random values -- *
00000003	00000001	00000001	* -- random values -- *
00000004	00000001	00000001	* -- random values -- *
00000005	00000001	00000001	* -- random values -- *
00000006	00000001	00000001	* -- random values -- *
00000007	00000001	00000001	* -- random values -- *
00000008	00000001	00000001	* -- random values -- *
00000009	00000001	00000001	* -- random values -- *
00000010	00000001	00000001	* -- random values -- *
.....			* *
.....			* *
00000099	00000010	00000001	* -- random values -- *
00000100	00000010	00000001	* -- random values -- *
00000101	00000011	00000002	* -- random values -- *

Table 5.4, DB300

Unique Key	Row Values
00000001	* ----- random values ----- *
00000002	* ----- random values ----- *
00000003	* ----- random values ----- *
00000004	* ----- random values ----- *
00000005	* ----- random values ----- *
00000006	* ----- random values ----- *
00000007	* ----- random values ----- *
00000008	* ----- random values ----- *
00000009	* ----- random values ----- *
00000010	* ----- random values ----- *
.....	* *
.....	* *
00000099	* ----- random values ----- *
00000100	* ----- random values ----- *
00000101	* ----- random values ----- *

Table 5.5, DBUPD

5.5.6 Table Size (Number of rows per table)

The CITY benchmark tables' sizes are flexible so the tables can fit on any level of computers.. The benchmark users are allowed to define minimum database size, maximum database size and increment database size depending on the architecture available for test. This feature is included in the CITY benchmark to permit the applicability of the benchmark on all ranges of computers. Hence, for small computer, for example PCs, a small database size could be defined and for larger computers larger databases could be defined. When different DBMS are compared, they must be compared using identical database sizes. Examples of those comparisons are presented in the next chapter.

When the tables loading stage is completed, the four tables DB100, DB200, DB300, and DBUPD contain equal numbers of rows. The rows insertions table, DBINS, starts by zero rows when the benchmark starts. For each benchmark loop a row is inserted to the table. The space allocated for this table should be large enough to accommodate all the rows inserted during the benchmark run.

5.6 The CITY Benchmark Transaction Script

This aspect transaction script comprises the classification of the numbers and types of retrievals and updates expected. Classification of queries is a difficult problem in the design of a benchmark because there are no fixed limits to the number of types of transactions that can be performed on a database. What complicates the problem is the fact that benchmarks in many cases do not just test systems performance but also systems functionality which depends largely on application type. Within the broad category of database systems, there is substantial diversity among the performance of systems on different problem domains. Systems behaviour varies enormously from one application to another. As presented in the previous chapter, one system may be excellent at performing simple update operations for on-line database, other may be dedicated to support queries. For example, an on-line transaction on a database will generally cause several other sub-transactions to be processed. These sub-transactions depend mainly on the database application. Unless the benchmark represent the specific domain it suppose to represent, the results from this benchmark can not be mapped to that domain. That is why some benchmarks are designed to cover the broadest range of applications within the tested domain, but by defining a practically manageable set of basic sub-transactions the task of modelling the database usage can be kept applicable.

Based on transaction analysis, the main characteristics of on-line transactions can be identified and an exact replication of the transaction is visible.

Over a period of time, researcher in the database area have proposed different classification of database transactions.

Gray [GRAY87a], classifies transactions as simple or batch. A simple transaction is submitted to the database in a single message and produces a result in a single message. Simple transactions do not consume a lot of time or resources and can be performed interactively. Batch transactions, on the other hand, are big jobs that are not done interactively. In fact, batch transactions may be set to run when the system load is low, because they either monopolise system resources, or take too much time to complete under normal system loads. Generally, there are several ways in characterising benchmark transaction scripts.

Turbyfill [TURB88], classified benchmark transactions scripts to the following types:

1. A utilisation transaction script.
2. A functional transaction script.
3. Branch testing.
4. A real transaction script.
5. A synthetic transaction script.

A utilisation transaction script tests the type and amount of resources consumed by executing the benchmark transaction script. That type of transactions was first discussed by Hawthorn and Stonebraker [HAWT79] where they classified queries as overhead intensive or data intensive. An overhead-intensive transaction references little data. The response time is dominated by overhead such as communication with the user, parsing the query, and overhead for operating system calls. A data-intensive transaction is one where the response time is dominated by manipulation of data in a relation due to data access time or computation. Eventually, the same approach was adopted by several other researchers such as Boral [BORA84], DeWitt [DeWI85] and Bitton [BITT87] where they classified a benchmark transaction script as CPU intensive or I/O intensive.

A functional benchmark transaction script is characterised by its high level of performing database functions. Database functions are such as a row retrieval or row insertion. Functional testing involves testing a DBMS through the level and quality of

functions it provides. A benchmark transaction script that test database functionality consist of long series of database operation and apply penalty for poor execution functions. Functional benchmark transaction scripts consisting of relational queries are described in [YAO 87].

Branch testing involves testing particular execution paths in a program. This may require access to the source code, which functional testing does not require. In several benchmarks program branches such as procedures responsible for relation scanning, sorting, and duplicate elimination are isolated through functionally specified queries. An example of this types is the work done by Yao, Havner, and Myers [YAO 87] where they used transaction complexity classification developed by Cardenas [CARD75]. Cardenas classified queries according to the complexity of qualification predicates to four predicate types: atomic condition; item condition; row condition; and transaction condition. An atomic condition has a form of unique qualifier value. An item condition is a disjunction of atomic conditions. A row condition is a conjunction of item conditions. A transaction condition is a disjunction of row conditions.

A real benchmark transaction script consists of actual application queries using real life database. An example of those types of scripts is presented in [GLES81], where a decision to selected a new DBMS was based on running real life application using every day use DBMS. This approach suffers from several problems: it has high running cost; it is inconsistent; and it leaves the tested systems at the mercy of security breaches.

A synthetic benchmark transaction script is one that is constructed from components that have never been part of an actual application. This approach enjoys several merits: it has consistency of the results; it maintains the tested system security; and the most important one is having full control on the tested system. As discussed in previous chapters the main problem with synthetic benchmark transaction is that in most cases they are not realistic enough and not representative to real life applications. The other problem is the degree of transaction complexity. Because most existing benchmarks are not based on background field studies, their transactions are either too complex to be realistic [HAWT79, YAO 87], or too simple to rigorously test database management systems [GRAY87a]. ;

The previous transaction characterisations are not mutually exclusive and can be mixed. Therefore, there might exist a functional benchmark that is based on real application transaction and a functional benchmark that is based on synthetic benchmark transaction. The best option is to use a synthetic benchmark transaction script that

reflects real life applications and based on in-depth analysis of real life database transactions, similar to the CITY benchmark.

The CITY benchmark applies functional testing for RDBMS using transactions written in ANSI SQL standard transaction language [AMER86, AMER87] to enhance the benchmark portability. The benchmark transactions include several transaction types such as select, project, insert, Update, and Join. The benchmark takes into consideration intermediate and final results represented as number of bytes, rows, types of attributes in result, and final transaction output destination. The benchmark also reflects a typical relative frequency of queries in its mix. The main advantage of the CITY benchmark is being constructed based on the findings from three studies at three large organisations, where the studies identified the main characteristics of on-line transactions' behaviour at high volume transactions' environments. The CITY benchmark transaction is built based on the similarities between on-line transactions behaviour at the three organisations presented in chapter four. The CITY benchmark transaction is a typical replication of the following operations as found in the empirical studies.

- Basic database operations.
- JOIN operation.
- Transaction complexity.
- Number of retrieved rows per transaction.
- Background workload.

5.6.1 The CITY Benchmark Basic Database Operations

The CITY benchmark transaction is an on-line transaction. As found from the studies, typical on-line database transaction consisted of 41% qualified retrieval, 5% insert, 7% update and 47% sequential retrieval. The CITY benchmark transaction script can be directly mapped to the studied systems and consists of twenty database operations distributed as follows:

- eight qualified retrievals (around 40%);
- nine sequential retrievals (around 45%);
- one insert operation (around 5%);
- two update operation (around 10%).

5.6.2 JOIN Operation

At both the airlines' environment and the bank environments the JOIN operation was avoided as much as they can. That was due to the negative effect of that operation on database performance. The findings from the two organisations were close enough to ignore statistical tests to check that difference. The 94% of on-line transactions accessed one database, a small percentage, around 2.5%, accessed two databases and similar percentage accessed three databases. The CITY benchmark reflect those ratios by having one JOIN operation out of 17 database retrieval operations in the benchmark script. This one JOIN represents around 6% (5.9%) of all retrieval operations in the benchmark script.

5.6.3 JOIN Transaction Complexity

As presented in the previous chapter there are several methods of implementing JOIN operations. One method could use a single select to access two tables, the other method might relay on nested sequence of select to perform the same function. As presented in the previous chapter, The first method of implementing JOIN operation is more common than the second method. JOIN operation that consisted of one select represented around 98% and JOIN operations that consisted of more than one select represented 1.8%. Therefore, JOIN transaction in the CITY benchmark consists of one select that accesses two tables.

5.6.4 Number of Retrieved Rows per Transaction

An average number of retrieved rows by on-line transaction was around seven. The bank systems returned just four rows per transaction, that kept the average of number of returned transactions relatively low. The CITY benchmark script returns ten rows on average per each transaction execution path. The number of rows retrieved by one execution path are as follows.

- Eight qualified retrievals return eight rows.
- Eight sequential retrievals return eighty rows.
- One JOIN operation returns hundred rows.
- One qualified update returns one row.
- Two sequential updates return twenty rows.

5.6.5 Background Workload

First, we distinguish between the system workload and the test workload. The test workload is the benchmark. The system workload consists of all other jobs running on a system while the benchmark is running. The system workload may consist of other, unrelated database jobs. A portion of a workload may be random, while another is controlled. For instance, the benchmark may be pre-defined, but run on a loaded system. In this case, the benchmark is controlled but the background workload is random.

The three organisations maintained a zero level background workload. The local authority computer centre, the airlines' computer centre and the bank computer centre dedicated some of their processors to perform all of the background workload they had at their environments. In the mean time, they dedicated some other to support the on-line service. That freed on-line processors to on-line service and reduced the level of background workload to zero value. Consequently, the CITY benchmark does not apply any kind of background load on the tested systems.

5.6.6 The CITY Transactions Workload

Benchmarks transactions workloads should take into consideration several factors that identify and characterise the quality of workloads. Those factor are the type of workload, the way it is applied and how it is controlled.

There are two main types of transactions workload, Random workload and Controlled workload. Random workloads depends on generating pure or pseudo random keys that access the tested system. Controlled workloads are calculated workloads before application and could be used to test things like databases access paths or a specific database function under predetermined load.

Workloads could be applied using techniques such as pure transactions scripts, programs, and computer processors emulating on-line terminals. When pure transactions scripts are used, workloads are encapsulated in a workload script that contain all the test requirements. Scripts are the simplest way of applying workloads but suffer from lack of control over all the parameters of the tested domain. Programs are similar to pure scripts but the script is embedded in a program code. The main advantage of this technique is beside being easy to apply it offers proper control over all

the parameters of the tested domain. Terminal emulators is the third method of applying workloads. This method is the best of three methods but characterised by the difficulty in use and complexity of application as it could be available to large computer company with unlimited resources but ordinary users usually do not have the means of applying it.

The most difficult part of workload application is the control of the workload. Workload load control includes aspects such as the accuracy of the workload, proper execution of workload script, completion of workload and making sure that the workload is doing exactly what it is designed to do and its measurements are as accurate as they should be. Workload control might utilise several tools such as system monitors and hardware probes. Workload control is mainly used at the early stages of designing and building the benchmark and must be adjusted to both the architecture of the system and the purpose of the benchmark.

The CITY benchmark is based on a random workload, similar to what exists in real life. The benchmark keys are generated by a random number generator that generates a randomly distributed keys covering all the tested domain. The CITY benchmark database operations are written in ANSI SQL [AMER86, AMER87], and embedded in a program. This approach allows simplicity of application and rigorous control over the benchmark execution process. The program is written in C high level programming language, a good guide to C language programming can be found in [LAWR86, BARA89, KANT90, MAST91]. The utilisation of ANSI SQL and C programming is for portability reasons. ANSI SQL is the standard practice for using DBMS and most DBMS have a SQL interface. Also, C programming language, is a high level programming language with a compiler available for almost all computer systems.

5.6.6.1 Qualified Retrieval

Qualified retrievals consist of eight queries each of which retrieves one row from one of the three retrievals tables. All retrievals use decimal indexed key. The queries are as follows:

1. Retrieve 1 Row from DB100.
2. Retrieve 1 Row from DB100.
3. Retrieve 1 Row from DB200.
4. Retrieve 1 Row from DB200.
5. Retrieve 1 Row from DB200.
6. Retrieve 1 Row from DB200.

5. The CITY Benchmark.

7. Retrieve 1 Row from DB300.
8. Retrieve 1 Row from DB300.

5.6.6.2 Sequential Retrieval

Eight sequential retrieval queries are used in the benchmark script. The nine retrievals use decimal key attributes in the query qualification parameter. The queries are as follows:

1. Retrieve 10 Rows from DB200.
2. Retrieve 10 Rows from DB200.
3. Retrieve 10 Rows from DB200.
4. Retrieve 10 Rows from DB200.
5. Retrieve 10 Rows from DB200.
6. Retrieve 10 Rows from DB300.
7. Retrieve 10 Rows from DB300.
8. Retrieve 10 Rows from DB300.

5.6.6.3 JOIN Two Tables

One JOIN operation is used in the benchmark script. The JOIN query accesses the tables DB200 and DB300 using decimal key attributes in the query qualification parameter. The JOIN is as follows:

9. Join DB200 (10 rows) and DB300 (100) Returns 100 rows.

5.6.6.4 Update a Row in DBUPD

The CITY benchmark script contains two update operations, the first updates one row and the second updates ten rows. The first simulates a single entity update and the other simulates a full screen update activity. The two update operations are the following.

1. Update a single Row in DBUPD.
2. Update 10 Rows in DBUPD.

}

5.6.6.5 Insert a Row in DBINS

The last operation in the CITY benchmark script is an insert row operation. The script contains one insert that looks as follows.

1. Insert a Row In DBINS.

5.6.6.6 Sequence of Operations Execution in The Benchmark Script

The database operations in the CITY benchmark script are organised and executed according to a rule that any table is not accessed twice by two consecutive queries. That is to force database buffer flushing and minimise memory retrievals. Database operations in the CITY benchmark script are executed in the sequence presented below.

1. Retrieve 1 Row from DB100.
2. Retrieve 10 Rows from DB200.
3. Retrieve 1 Row from DB100.
4. Retrieve 10 Rows from DB200.
5. Update a single Row in DBUPD
6. Retrieve 1 Row from DB200.
7. Retrieve 1 Row from DB300.
8. Retrieve 10 Rows from DB200.
9. Retrieve 10 Rows from DB300.
10. Retrieve 1 Row from DB200.
11. Retrieve 10 Rows from DB300.
12. Retrieve 10 Rows from DB200.
13. Retrieve 1 Row from DB300.
14. Retrieve 1 Row from DB200.
15. Retrieve 10 Rows from DB300.
16. Retrieve 10 Rows from DB200.
17. Update 10 Rows in DBUPD.
18. Retrieve 1 Row from DB200.
19. Join DB200 (10 rows) and DB300 (100) Returns 100 rows.
20. Insert a Row In DBINS.

5.6.7 Control of Transactions Execution

All transactions in the benchmark test must run to completion. To guarantee transaction execution error traps were set before each transaction execution so that in abnormal condition occurrences, the program prints an error message indicating the error type then terminate not allowing the benchmark test to continue. Also, the test did not allow "NOT FOUND" condition to occur as all transactions must run to

completion, when "NOT FOUND" condition occurs the program generates an error message and the program is abnormally terminated. The error traps and error handling routine are presented below.

1. Trap any error that might occur:

```
Prepare Any Condition Error Message;  
EXEC SQL WHENEVER SQLERROR GOTO Error_Trap_Routine;
```

2. Trap not found condition when it occurs:

```
Prepare Not Found Condition Error Message;  
EXEC SQL WHENEVER NOT FOUND GOTO Error_Trap_Routine;
```

3. The errors trap routine:

```
Error_Trap_Routine:  
    sound error alarm  
    print error message and all related information  
    Terminate processing
```

5.6.8 Random Number Generators

The CITY benchmark test used two random number generators. The first is a modified version of the C compiler. The second is a random number generator developed by the national physics laboratory (NPL) [WICH82].

The C language random number generator is based on the magic formula:

$$\text{rand_value} = (\text{randx_value} * 25173 + 13849) \% 65536$$

The C language random number generator subroutine was modified to generate random numbers within a given limit. This random number generator was used for two purposes. The first is to generate a random value for attributes selection. The second was to generate three seed values for the NPL random number generator.

The random number generator subroutine receives three global values, such as x, y and z, to be given initial random integer values which should be less than 30,000 and generate a random number from 0 to infinity. The random number generator was modified to return a random number within specified limit. This random number generator was used to generate random key values for the database operations.

5.7 Requirements of Transaction Processing Council

Due to the growing importance of transaction processing at the database environment several computer and database vendors formed a council to establish standards for On-line Transaction Processing (OLTP) benchmarks. The name of the council is Transaction Processing Council (TPC). The council has constituted standard criteria for any benchmark testing to be accepted by the council. For any benchmark to be accepted by the council it has to follow the following requirements:

1. Proper specification of table layouts, number of rows for each table, and minimum number of rows of each table that must be accessed per test.
2. A detailed description of transactions to be run.
3. Clear specification of rules for distributing and partitioning data among tables, for timing transactions, and for hardware configuration for the benchmark.
4. Requirements and recommendations for reporting benchmark activity and results should be available, so the benchmark can be repeated and the information can be used by independent agencies.
5. The Transaction Processing Council (TPC) has set system properties that must be in effect during the transaction execution. These properties are called ACID properties (atomicity, consistency, isolation and durability). The DBMS must ensure atomicity during the benchmark so no partial transactions are permitted to modify the database. The DBMS must be consistent during the benchmark by ensuring that each transaction takes the database from one consistent state to another. The database must ensure isolation (serializability) of all transactions during the benchmark test, so the results of concurrently executing transactions are the same as the results that would have been achieved by the serial execution of the same transactions. The DBMS must ensure durability by preserving the effects of all committed transactions even in the face of system and media failure.

In compliance with the TPC requirements, the CITY benchmark users should observe the previous points when running the benchmark test. The creation procedure of the CITY benchmark tables is presented in appendix A. The load procedure of the

CITY benchmark tables is presented in appendix B. A description of the CITY benchmark transactions is presented in appendix C. Random number generators are described in appendix D. The operation manual of the CITY benchmark is presented in appendix F.

5.8 Summary

This chapter presented the CITY benchmark. The benchmark is the first database performance evaluation methodology based on in-depth empirical studies. The benchmark transaction script mainly represents a typical on-line transaction at high-volume transactions' environments accessing large databases, those environments similar to what was presented in the previous chapter.

The next chapter discusses the benchmark test verification process. It demonstrates the main characteristics of the CITY benchmark and presents the results of the benchmark verification procedure. It also compares the results from the CITY benchmark to the TPC benchmarks showing the difference in accuracy between the two benchmarks.

CHAPTER 6

PRELIMINARY TEST AND VERIFICATION of The CITY BENCHMARK

CHAPTER 6

PRELIMINARY TEST AND VERIFICATION of The CITY BENCHMARK

6.1 Introduction

Chapter five presented The CITY benchmark. The benchmark design is based on empirical studies in large database environments. The research revealed a common pattern between the studied systems, despite the difference in activities and nature of the organisations concerned. The CITY benchmark transactions are derived from the findings from those studies.

This chapter presents the preliminary test and verification of the CITY benchmark. This test took place in two standalone PC environments. Those environments were selected to ensure proper control over the benchmark test variables and to allow thorough examination of the benchmark results. The chapter starts by explaining the examination process of the benchmark variables then discusses the verification process of the benchmark results. Two main benchmark characteristics were examined through the verification process, those characteristics were reproducibility of the benchmark measures and scalability of the CITY benchmark transaction.

In compliance with the Transaction Processing Council (TPC) requirements, the researcher has conducted tests of Atomicity, Consistency, Isolation, and Durability (ACID) as independent experiments that are not related to the CITY benchmark transaction.

6.2 Test of the Main Characteristics of The CITY Benchmark

Verification of benchmark results involves addressing several important problems, the first is the benchmark test approach. As there are two approaches in dealing with benchmark test (controlled test where all factors that may affect a

performance metric are controlled, and non-controlled test where all factors that may affect a performance metric are randomised) the first problem is defining the approach to be conducted. The choice depends mainly on the benchmark objective and whether it is a special purpose benchmark or a general purpose benchmark. General purpose benchmarks should be tested in a controlled environment, hence the benchmark variables should be the only test factors involved in the benchmark test. That allows the replication of the benchmark results.

In this work the benchmark test was conducted as a controlled test, where a single factor was tested at a time and then systematically varied. That factor was isolated and kept independent of all other factors that were controlled and observed at fixed number of levels. To be able to do that, the preliminary test took place in two standalone PC environments, 386 PC and 486 PC. The verification process aimed to test the following factors:

- the benchmark measures are reproducible;
- the benchmark is scalable;
- the benchmark usage cost is low.

This stage also checked the benchmark variables by dumping the values in those variables during the benchmark run; and all other parameters involved in the benchmark test like the random numbers that were used as row key. Finally this process tested the effect of changing some benchmark variables such as attribute type, and attribute distribution.

The test of the benchmark results depended on replication of the benchmark runs, randomisation of the test keys and analysis of the covariance of the results of the runs.

6.2.1 Reproducibility of the Benchmark Measures

Reproducibility of the benchmark measures is the most important factor when any benchmark is verified because it represents the benchmark results consistency. Reproducibility could be defined as:

"The ability to produce consistent results when repeating the same benchmark test"

Reproducibility of the benchmark measures can be tested by replicating the benchmark run then analysing and comparing the results obtained from all runs. It is recommended that each test must be conducted at least twice to gain an indication of the reliability of the metric, and in accordance with this in most published work benchmarkers conducted the test only twice. In this work each test was repeated at least hundreds of times as it took in all over 8000 computer hours. Reproducibility of the benchmark measures were tested by examining the following factors.

- Clock adjustment and first transaction effect.
- Reproducibility of the benchmark measures within a single loop.
- Reproducibility of the benchmark runs.
- Duration of loops time.
- Duration of full test of DBMS.
- Transaction atomicity.
- Transaction consistency.
- Transaction isolation.
- Transaction durability.

6.2.1.1 Clock Adjustment and Eliminating First Transaction Effect

During the test process two observations that might affect the test measurements were noticed. Those observations are the following.

1. The first reading from the clock and consequently time calculation was abnormally high then the time stabilises after that reading and enjoys a high level of consistency. That time measurement did not involve database operations and the deviation of the calculated time could not be justified. Consequently a dummy loop was implemented to eradicate that deviation in time. That loop gets the first time calculated and then runs a dummy transaction for a predetermined time. That method succeeded in overcoming the first time deviation problem and all the subsequent time measurements were tested and found within acceptable range of values.

The dummy loop looks as follows:

```
start time = system clock time;  /* Get first system time to s  
clock. */  
end time = start + DSECONDS;    /* Add number of dummy seconds.  
do  
{ /* start of do loop body. */
```

```
Dummy non-database operation to adjust system time  
} /* end of do loop body. */  
while (system clock time < end time);
```

2. Database management systems take from one to several loops until the collected response time steadies. That is because database management systems set several database requirements such as database buffers before being ready for full utilisation by database users. That database environment preparation puts an over-head on the first database transaction, and in some cases it takes several database transactions until transactions' response time stabilises. Fig. 6.1. shows that response time required several transactions until transaction response time stabilises and concentrated around a fixed value. This deviation in first transaction response time was included in the presented measurements and was not discarded because it was considered a fixed cost over different database management systems transactions that should be presented.

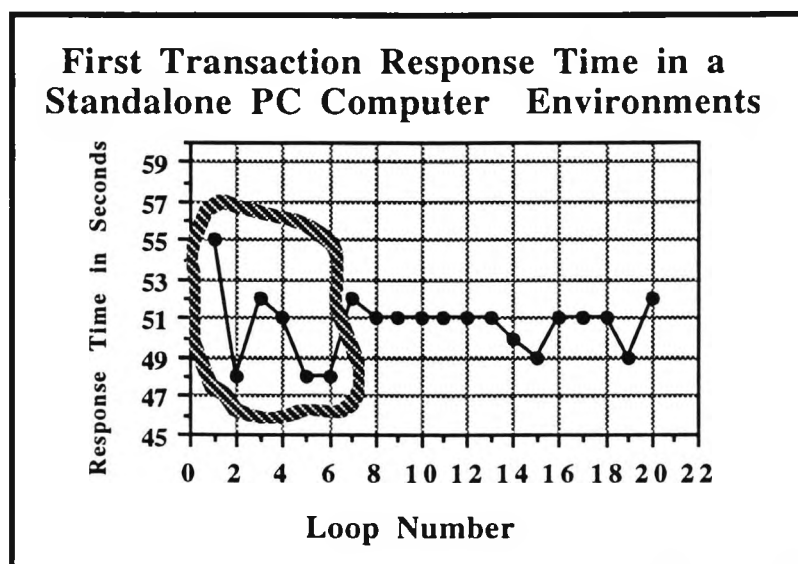


Fig. 6.1, First transaction response time in PC environment

6.2.1.2 Reproducibility of the Benchmark Measures Within a Single Loop

As discussed in chapter 5 (§5.4.2), transaction response time and number of transactions per seconds are calculated through time loops. The benchmark transaction is repeated several times within a loop that is controlled by time factors, then the average of all response times and transactions per second are calculated as one figure from the loop either by dividing total response time by number of transactions to give

transaction response time or by dividing total number of transaction by total test time to give the number of completed transaction per second.

The second test was to check that time measurements calculated from each loop did not statistically vary and all the obtained values were spread within acceptable range and did not deviate from the calculated mean. The variance of the calculated measure was used as measure of spread of the collected values. The main advantage of using the sample variance is that it gives a statistical inference about the underlying population. Sample variance was discussed in detail in chapter three.

To that end, loop time was calculated and recorded for each independent loop. In each experiment each loop time was calculated for at least 500 times, then the variance of the collected sample was calculated.

Table 6.1 presents a small sample of loop times that were collected from the PC environment under scalable database load. The sample presented in Table 6.1 was taken from the results of 100 runs. Out of those 100 runs, only ten runs' values are presented for space and presentation considerations. The presented variance in this context was calculated for the test runs. The runs' values in Table 6.1 show low spread pattern between the collected runs' times under all database loads. The variances of the values presented in Table 6.1 are as follows:

Variance of 2500 Rows:	2.0
Variance of 5000 Rows:	0.4
Variance of 75000 Rows:	0.5
Variance of 10000 Rows:	0.1
Variance of 12500 Rows:	0.5
Variance of 15000 Rows:	0.2
Variance of 17500 Rows:	0.5
Variance of 20000 Rows:	0.7

The highest variance belongs to 2500 rþws' table size. This is due to the time taken by the DBMS to set the database parameters.

Loop number	2500 Rows	5000 Rows	7500 Rows	10,000 Rows	12,500 Rows	15,000 Rows	17,500 Rows	20,000 Rows
Loop 1	55	54	57	61	62	66	69	72
Loop 2	48	54	57	61	63	68	68	74
Loop 3	52	54	58	61	64	67	70	72
Loop 4	51	55	58	61	64	67	70	74
Loop 5	48	54	56	61	65	68	70	74
Loop 6	48	55	58	62	62	67	71	73
Loop 7	52	55	59	61	64	67	71	73
Loop 8	51	54	58	61	64	67	69	73
Loop 9	51	55	58	61	64	68	71	74
Loop 10	51	54	59	61	64	67	70	73

Table 6.1, Spread of CITY measures in PC environment

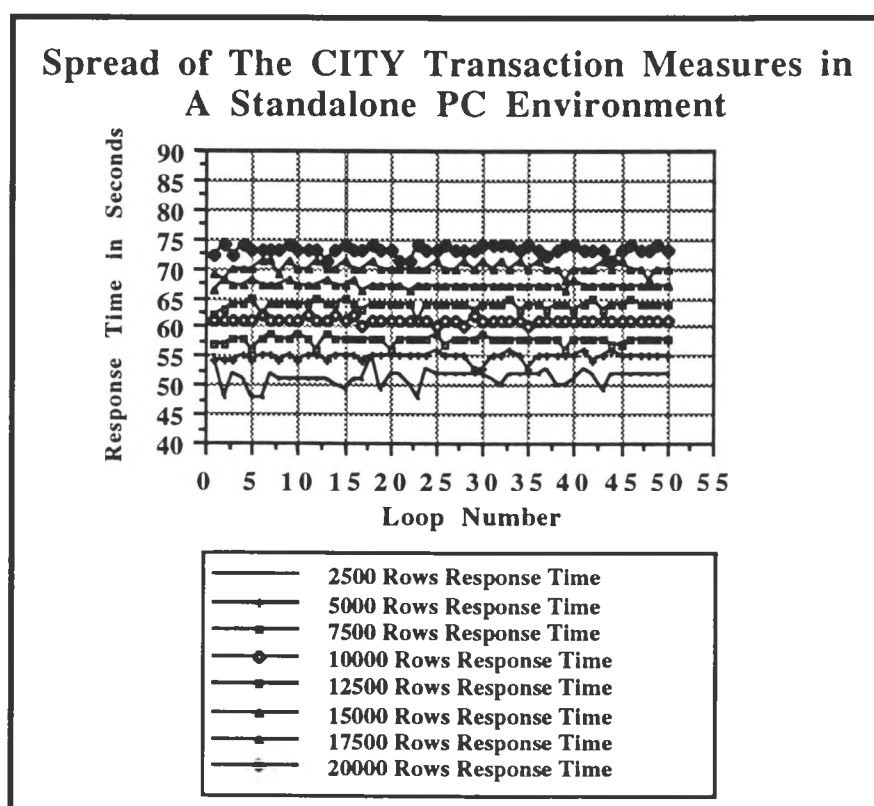


Fig. 6.2, Spread of CITY measures in PC environment

Fig. 6.2 shows a graphical representation of fifty runs in the PC environment. The graph shows high stability of measurement from one loop to the next. The loop variance was less than 0.5 in all cases except for database size of 2500 rows. The test

result indicated significantly low spread of the collected values and guaranteed that the calculated mean will be representative value to all the results collected from all loops.

The sample presented in the previous example were classified in a frequency table, Table 6.2, that classifies the 50 response times sample collected from the standalone PC environment test runs. In that table four table sizes are presented 5000 rows, 10,000 rows, 15,000 rows and 20,000 rows. For all measurements, response times of the collected transactions were spread between just three or four different response times that were highly centred around the sample mean. Fig. 6.3 is a graphical representation of response time frequencies that are presented in Table 6.2.

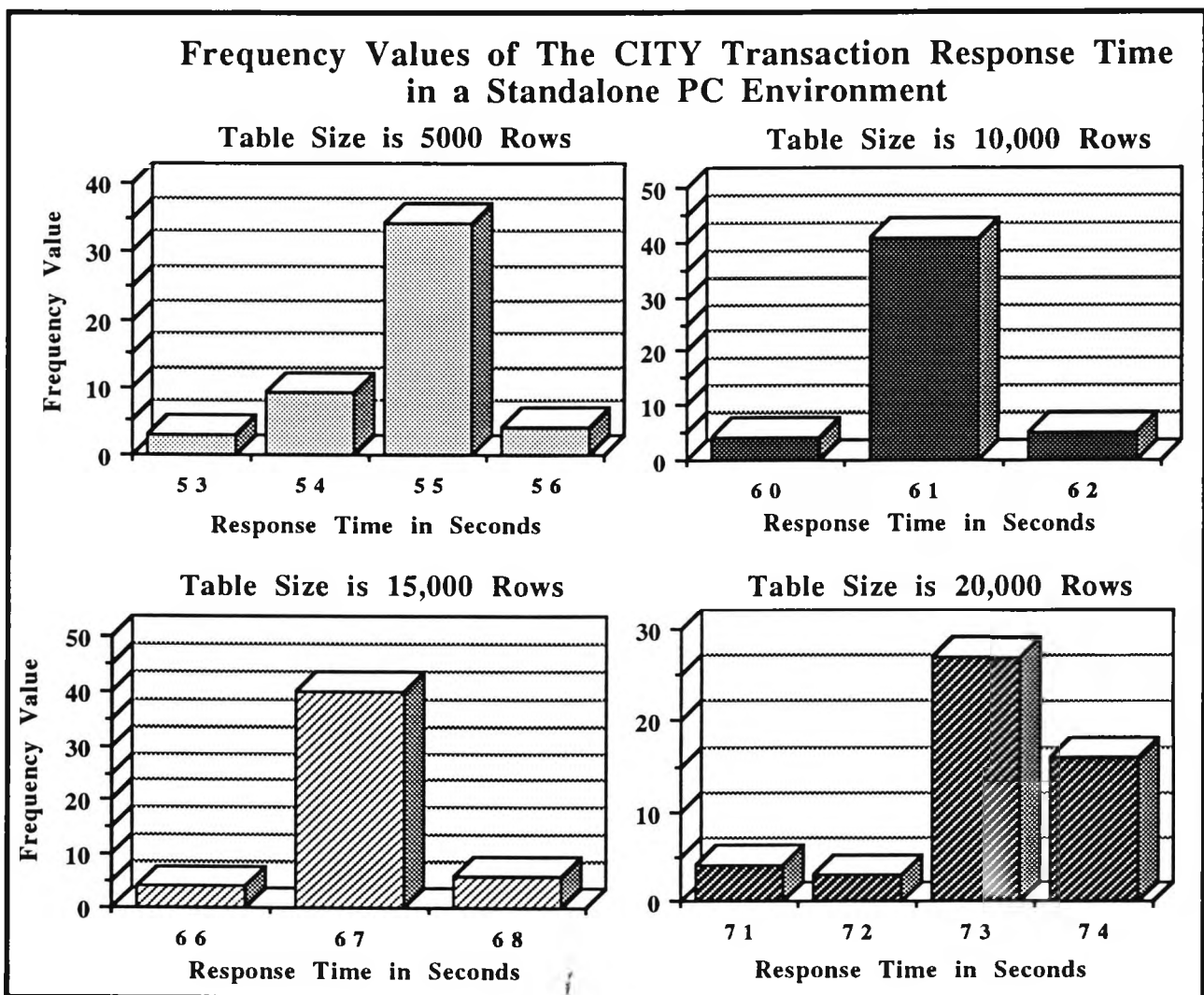


Fig. 6.3, Frequency values of CITY measures in PC environment

5000 Rows Response	Freq.	10,000 Rows Response	Freq.	15,000 Rows Response	Freq.	20,000 Rows Response	Freq.
53	3	60	4	66	4	71	4
54	9	61	41	67	40	72	3
55	34	62	4	68	6	73	27
56	4					74	16
5000 Rows Average Response Time.		10,000 Rows Average Response Time.		15,000 Rows Average Response Time.		20,000 Rows Average Response Time	
54.78		61.02		67.04		73.10	

Table 6.2, Frequency values of CITY measures in PC environment

6.2.1.3 Reproducibility of the Benchmark Runs

In the PC environments the benchmark results from one run to the next were identical. For example if run1 response time was 65 seconds, run2 was also 65 seconds. That was found for all runs using different tables' sizes. Because the results were identical, analysis of variance was not applied.

6.2.1.4 Duration of Loops Time

Loop time duration is an important factor in obtaining consistent measures from the benchmark test. The longer the loop time the more measures collected regarding the same database size which means larger sample and more accurate result. Several loop time duration were tested against a range of database sizes to study the effect of loop time duration on the calculated transaction response time accuracy. The obtained results decided the smallest loop duration to obtain enough measurements to produce consistent results.

In the PC environment two loop times were tested. The first loop time duration was 900 seconds and the second loop time duration was 3600 seconds. Each run was repeated fifteen times to collect a reasonable number of measurements. The results of testing loop duration in the PC environment are presented in Table 6.3. The test did not find significant difference in response time measurements when running each loop for 900 seconds and 3600 seconds. Accordingly other time intervals, including 9000 seconds, were not tested and the subsequent measurements in the PC environment were based on 900 seconds loop. The present author does not recommend running the test

for less than 900 seconds per loop because they might not produce statistically credible number of measurements.

Loop Time	5000 Rows	10000 Rows	15000 Rows	20000 Rows
900 Seconds	50	56	64	69
3600 Seconds	51	58	64	69

Table 6.3, Effect of loop time on CITY measures in PC environment

6.2.1.5 Test of Transaction Atomicity

The CITY benchmark test process guaranteed that transactions were atomic in the systems under test. The tested systems performed all individual operations on the data. Test of transaction atomicity was done as follows:

1. the benchmark was run for randomly selected records and verify that those records have been changed;
2. the benchmark was run for randomly selected records substituted an ABORT COMMIT of the transaction. The appropriate records were checked to ensure that they had not been changed.

6.2.1.6 Test of Transaction Consistency

The experiments guaranteed the execution of transactions to take the database from one consistent state to another. This property was tested as follows:

- a. One random record was updated and its integer value is increased by one.
- b. Ten random records are updated and their integer values are increased by one.
- c. The insert database had one logical record added for each committed transaction, none for any aborted transactions.

6.2.1.7 Test of Transaction Isolation

This property is commonly called serializability. Sufficient conditions must be enabled at either the system or application level to ensure serializability of transactions under any mix of arbitrary transactions. The system or application must have full serializability enabled, i.e., repeated reads of the same records within any committed transaction must have returned identical data when run concurrently with any mix of arbitrary transactions. This property was not tested during the benchmark verification process because all runs took place in a single user mode.

6.2.1.8 Test of Transaction Durability

The last property to test was the ability of the tested systems to preserve the effects of committed transactions and to ensure database consistency after recovery from any one of the database failures. This property was tested by forcing the tested systems to fail during the benchmark test by either forcing a quit from the DBMS or systems shut down. The committed transactions were then checked for transaction completeness.

6.2.2 Test for Scalability of the CITY Transaction

Scalability in this chapter concentrated on increasing database size and stepping processing power one step from 386 processor to 486 processor. Scalability was discussed in chapter three (§3.6.3).

It should be mentioned that factors such as: page size; operating systems; index type; and several other factors can affect transaction scalability. If those effects are ignored then an arguable case can be made that transaction response time will be linear in respect to the increase in workload. This research assumes that the scalability will be linear against the increase in the workload.

In designing for scalability, the experiments tested the CITY transaction under the following scalability conditions:

1. scaling up tables' sizes from 1000 rows to 25,000 rows;

2. scaling up the tested system processor power from a standalone 386 PC to a standalone 486 PC.

The CITY transaction took scaling behaviour in the two tested environments regardless of database size and processor power. Fig. 6.4 demonstrates the CITY transaction behaviour in the tested environments.

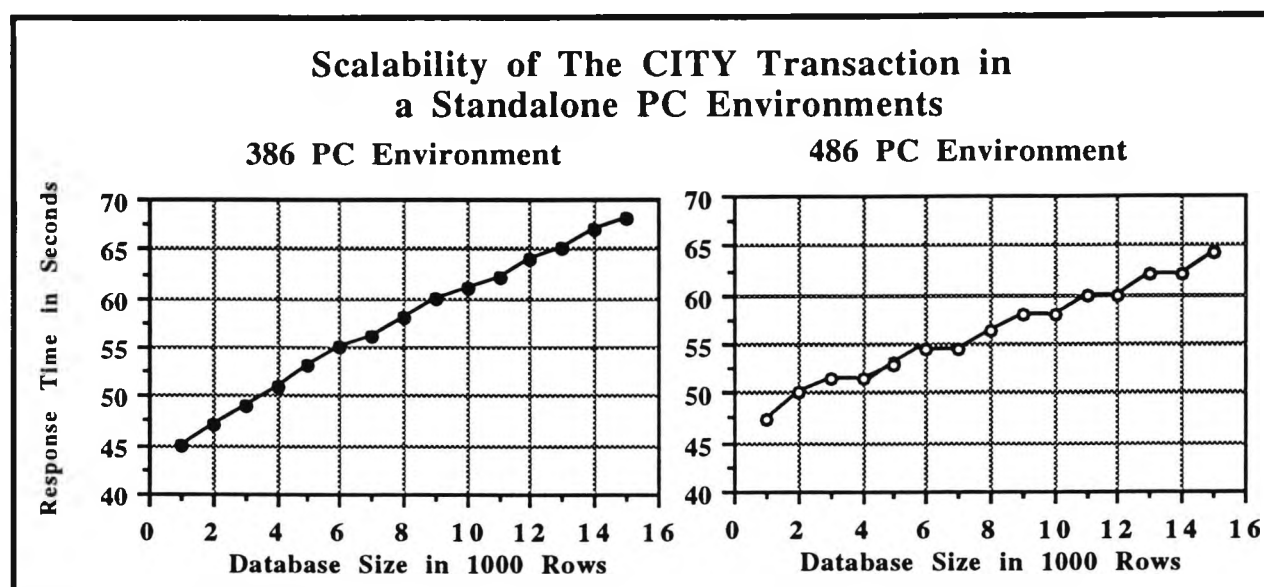


Fig. 6.4, Scalability of The CITY measures in PC environment

6.2.2.1 Transaction Scalability VS Usage Cost

Transaction cost can be calculated as the time a transaction takes from its start to its finish. The longer the time it takes, the more expensive the transaction. An optimum benchmark transaction would apply sufficient workload on the tested database within minimum cost. Examples of expensive transactions are presented in a work by Strawser [STRA84], where she created a comprehensive benchmark that consisted of twenty-five database operations, but because it was too expensive (required long time to run as it consisted of fifty five queries varied from simple to very complex joins), it has never been put into practical use.

To test scalability versus usage cost, the present author is proposing a scalability ratio. It is an unbiased measure of transaction cost that can be used to compare different transactions. It offers a test of transaction scalability versus its cost no matter how

expensive it is. The advantage of this method is that it takes transaction cost into consideration and eliminates the effect of expensive transaction.

Previous empirical studies of the behaviour of basic database operations showed that regardless of transaction cost, not all database operations are scalable. Consequently a mix of those database operations might not be scalable. To test that assumption, several transactions' mixes varying in basic database operations and transaction workload were tested in a standalone PC environment.

The empirical studies tested eight mixes that vary in transaction operations' workload and transaction cost. Scalability of the eight mixes were tested against increasing database size in both environments. Each mix consisted of twenty database operations to equal the number of database operations in the CITY benchmark script. The mixes were designed based on the resource utilisation approach proposed by Hawthorn [HAWT79], where she defined the three classes of relational queries overhead-intensive, data-intensive, and the multi-relational queries. That approach was also adopted by the Wisconsin benchmark group [DeWI85]. That approach was discussed in greater detail in chapter two. The mixes are presented below.

1. Mix 1: One row qualified retrieval + one insert + one row qualified update.
2. Mix 2: 100% one qualified update + one row qualified retrieval + one row insertion.
3. Mix 3: 100% one row insertion + one row qualified retrieval + one row qualified update.
4. Mix 4: 75% one row qualified retrieval + 25% Ten rows sequential retrieval + one insert + one row qualified update.
5. Mix 5: 55% one row qualified retrieval + 50% Ten rows sequential retrieval + one insert + one row qualified update.
6. Mix 6: 25% one row qualified retrieval + 75% Ten rows sequential retrieval + one insert + one row qualified update.
7. Mix 7: Ten rows sequential retrievals + one insert + one row qualified update.
8. Mix 8: JOIN mix that accesses two tables and returns 100 rows.

6.2.2.2 Scalability of The CITY Transaction in Comparison To Different Mixes in Two Standalone PC Environments

Transactions test in two different environments, a 386 standalone PC environment and a 486 standalone PC environment showed that the CITY transaction has produced higher scalability levels than all the other tested mixes. The CITY transaction scalability in comparison to other mixes' scalability in a PC environment are presented in Table 6.4.1 and Table 6.4.2. Fig. 6.5.1, summarise the results of testing the CITY transaction scalability pattern in comparison to other mixes.

The experiment also showed that having just CPU based mix or just I/O based mix do not guarantee transaction scalability, but the overall mix scalability is a function in all the database operations contained in that mix and scalability can not be referred to one single operation. A clear example is the JOIN mix (mix 8) that despite being the most expensive mix it does not produce the highest scalability level. Fig. 6.5.2, illustrate the CITY transaction scalability in comparison to JOIN mix scalability.

Database Size	5000 Rows Response Time	10,000 Rows Response Time	15,000 Rows Response Time	20,000 Rows Response Time
CITY 386 Env.	53.000	61.000	68.000	75.000
CITY 486 Env.	51.000	58.000	64.000	69.000
Mix 1	10.070	10.256	10.360	10.405
Mix 2	9.756	10.198	10.435	10.496
Mix 3	11.688	11.803	11.723	11.726
Mix 4	18.947	18.848	19.048	19.251
Mix 5	27.273	27.481	27.692	27.907
Mix 6	36.000	36.735	36.735	37.113
Mix 7	45.567	45.950	46.000	46.650
386 Mix 8	394	401	402	403
486 Mix 8	331	333	333	334

Table 6.4.1, The CITY transaction scalability level against other transactions

Scalability Level	Δx_1 10,000-5000 Rows	Δx_2 15,000-10,000 Rows	Δx_3 20,000-15,000 Rows
CITY 386 Env.	8.000	7.000	7.000
CITY 486 Env.	7.000	6.000	5.000
Mix 1	0.186	0.103	0.045
Mix 2	0.442	0.236	0.061
Mix 3	0.115	-0.077	0.000
Mix 4	-0.099	0.199	0.204
Mix 5	0.208	0.211	0.215
Mix 6	0.735	0.000	0.379
Mix 7	0.570	0.584	0.000
386 Mix 8	7.000	1.000	1.000
486 Mix 8	2.000	0.000	1.000

Table 6.4.2, The CITY transaction scalability level against other transactions

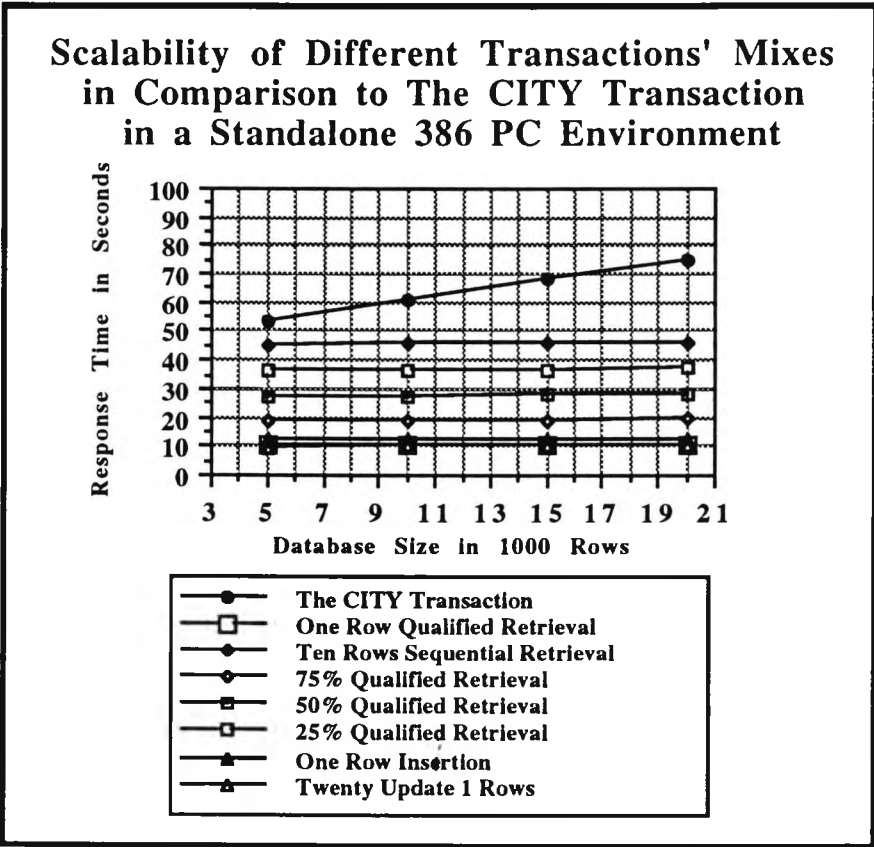


Fig. 6.5.1, The CITY transaction scalability level against other transactions

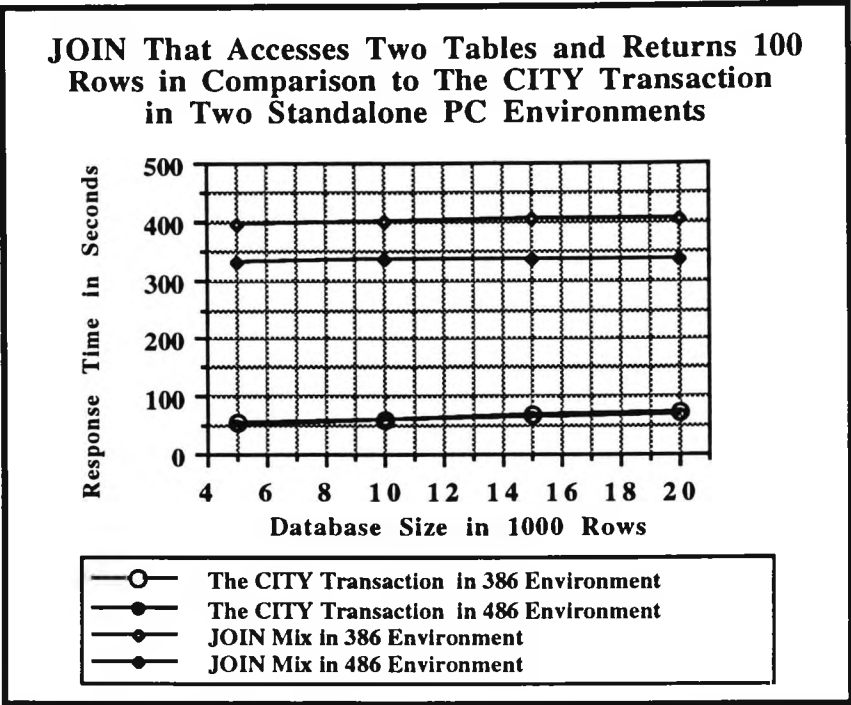


Fig. 6.5.2, The CITY transaction scalability level against Join transactions

6.2.2.3 Scalability Ratio of The CITY Transaction in Comparison To Different Mixes in Two Standalone PC Environments

The previous section showed that the CITY transaction has a better scalability level than all the other mixes. In this section scalability ratio of the CITY transaction is compared to scalability ratio of the eight mixes presented in the previous section. This comparison justifies the CITY transaction scalability versus its cost in comparison to different mixes representing scalable transaction cost.

The comparison of the CITY transaction scalability ratio against the other eight mixes took place in a 386 standalone PC and a 486 standalone PC, the results of the comparison are presented in Table 6.5 and illustrated in Fig. 6.6.

The CITY transaction has produced a better scalability ratio than all the other eight mixes in both environments. That is because some transactions consist of database operations that originally produce low scalability ratios such as qualified retrieval of one row using unique key which in many cases produces a negative scalability ratio due to its index dependence. On the other hand some other transactions could produce high levels of scalability, yet they are too expensive to produce equally high scalability ratio. A clear example is the JOIN mix (mix 8), which is most expensive mix in all

environments but it produces the lowest scalability ratio due to the relatively high cost of transaction execution.

Scalability Ratio	$\Delta y_1/y_1$ 5 -> 10	$\Delta y_2/y_2$ 10 -> 15	$\Delta y_3/y_3$ 15 -> 20	Sum of Ratios
CITY 386 .Env.	0.151	0.115	0.103	0.369
CITY 486 Env.	0.137	0.103	0.078	0.319
Mix 1	0.019	0.010	0.004	0.033
Mix 2	0.045	0.023	0.006	0.074
Mix 3	0.010	-0.007	0.000	0.03
Mix 4	-0.005	0.011	0.011	0.016
Mix 5	0.008	0.008	0.008	0.023
Mix 6	0.020	0.000	0.010	0.031
Mix 7	0.013	0.013	0.000	0.025
386 Mix 8	0.018	0.002	0.002	0.023
486 Mix 8	0.006	0.000	0.003	0.009

Table 6.5, The CITY transaction scalability ratio against other transactions

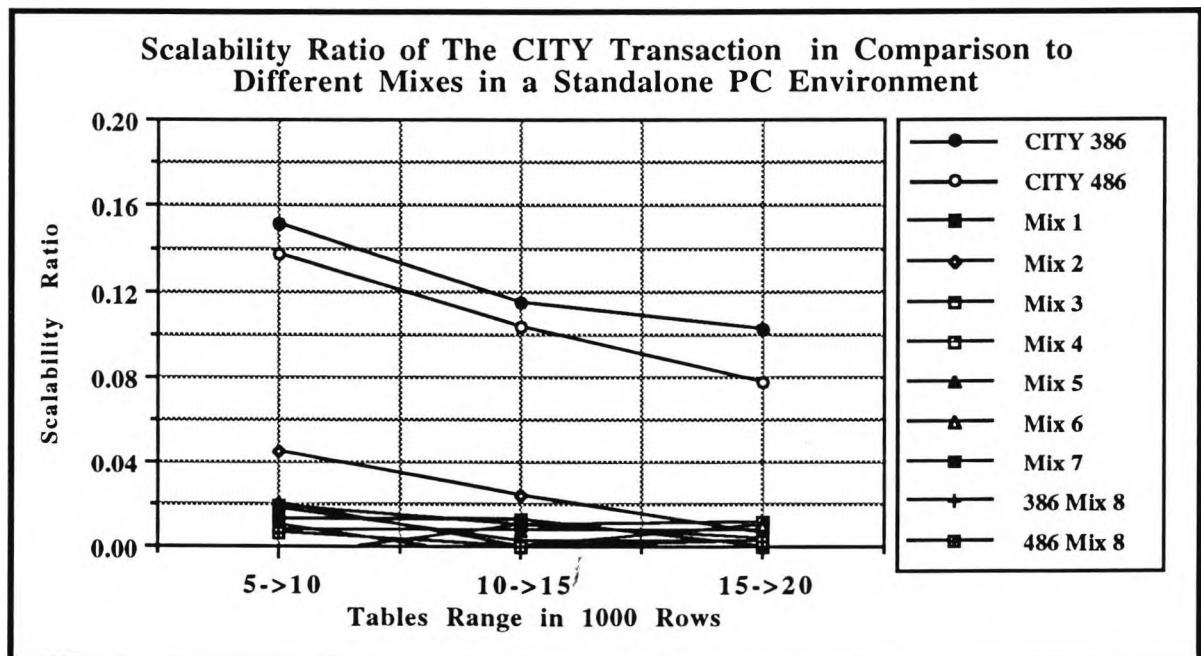


Fig. 6.6, The CITY transaction scalability ratio against other transactions

6.2.3 Simplicity of Construction and Clarity Code

Simplicity of construction includes aspects such as: the time required to understand the benchmark script; the resources required to complete a comprehensive test; and the degree of complexity to gather and interpret the test results. All those factors are added to produce the benchmark usage costs from the stages of constructing and implementing the benchmark to the stage of collecting and interpreting the test results.

6.2.3.1 Flexibility and Simplicity of Construction

The CITY benchmark allows some degree of flexibility regarding two benchmark parameters, test tables' sizes and non-key attributes. When using response time under scalable load as the benchmark metric, users are asked to set the test tables' sizes. The benchmark asks the users about start table size, end table size and step table size. That allows the benchmark users to tailor the test database size according to the available space in the tested system.

Because not all attributes' types are available on all computer and DBMS, the CITY benchmark users are allowed to change non-key attributes to whatever available in their environment.

The CITY benchmark script is simple to construct and easy to run. The script is compact, and consists of twenty database transactions that run in sequential order within time controlled loops. The code itself is compact, all the commonly used routines are kept as independent modules that are called by the main program when required. The program requires a C compiler and 450k of main memory to run, which are available on most computers. The disk requirements are system dependent and could be tailored according to systems' resources as explained in the previous section.

6.2.3.2 Clarity of Code

The Benchmark script is well documented and self explanatory. At the different stages of the benchmark test and verification, the benchmark script was given to some of the covered organisations and they were able to run it themselves with minimum help from the present author.

Each variable used in the program code is commented on specifying what it is and why it is used. An example is the following:

```
/* ***** */
/* Database size variables */
/* ***** */
unsigned long  Rows_Start, /* Database size start value */
               Rows_End,   /* Database size end value */
               Rows_Step;  /* Database size step value */
```

Also each line of code in the program is commented on to explain why and how it is used. An example is the following:

```
/* ===== */
/* Transaction 1. Retrieve 1 Row from DB100 using unique key. */
/* ===== */

/* Get a random key */
x = rand2(r_limit); /* get the first random seed */
y = rand2(r_limit); /* get the second random seed */
z = rand2(r_limit); /* get the third random seed */
key_value = randp(x,y,z,ROWS); /* get random unique key */

strcpy(error_message,err_msg); /* set error message */
EXEC SQL WHENEVER SQLERROR GOTO SQL_error_routine; /* error trap */
EXEC SQL SELECT
        DB1FC1, DB1FI1, DB1FC2, DB1FL1, DB1FR1, DB1FD1, DB1FV1
INTO    :dbfc1, :dbfi,  :dbfc2, :dbfl,  :dbfr,  :dbfd,  :dbfv
FROM    DB100
WHERE   DB1FI1 = :key_value;
```

6.2.3.3 Interpretation of the Results

The benchmark results are presented in an easy to understand and interpret form. After the end of each run, the test results are presented in either number of transactions per second or the average response time against different database sizes. The test results take the following layout.

- Database type;
- Database size;
- Transactions per second;
- Transaction response time.

6.3 Comparison between the CITY Transaction and the TPC Transactions in a Standalone PC Environment

The Debit/Credit (TP/1) [ANON85] and the Transaction Processing Council (TPC) benchmarks [TPC 89, GRAY91] have become the database market standard practice. Those benchmarks have a number of practical limitations [REVE90]. Most of those limitations stem from the lack of background study before the benchmark design stage. The main criticism of the TPC benchmarks is that they do not provide a realistic characterisation that can be set as the target for the benchmarking process, and moreover, they do not even provide a model of the ATM systems whose performance they are supposed to simulate [REVE92b].

Chapter two discussed the TPC benchmarks. The main limitation of those benchmarks was their transactions' scripts. The TPC-A consisted of three updates using unique key and one insert. The TPC-B consists of a TPC-A transaction plus one select operation that uses a unique key. Due to the over simplification of those transactions' scripts the present author conducted several empirical studies to test the cost of those operations and to examine their testability under increasing database load. The studies found that the basic database operations of the TPC benchmarks do not apply significant workloads on the tested systems, and this represents a severe limitation to the TPC benchmark scripts and leads to inadequacy when comparing database management. Consequently, the TPC benchmarks are expected to suffer limitations in four areas. Those areas are the following.

- Transactions' scalability levels.
- Transactions' scalability ratios.
- transactions' measures comparability between DBMS.
- TPC and the future of DBMS.

In the following section the TPC benchmarks (TPC-A and TPC-B) are compared to the CITY transaction in each one of the previous areas. The comparison showed that the CITY transaction produced better results in all areas.

6.3.1 Comparison Between the CITY and the TPC Transactions' Scalability Levels

Several empirical studies were conducted to compare the behaviour of the CITY transaction script against the TPC-A and the TPC-B transactions scripts. The basic

database operations of the TPC-A and the TPC-B were isolated and tested against increasing database size. Tables' sizes were scaled up gradually from 1000 rows to 15,000 rows. Row size in all experiments was 200 bytes. This section presents the results from those studies.

The assumption was that, because the TPC benchmarks consisted of operations that are not scalable, the result from mixing those operations will not be scalable. The other assumption was that, because the CITY transaction contains scalable database operations, it will produce better scalability level than the TPC transactions. The findings that are presented in Tables 6.6 and 6.7 and Fig. 6.7 can be summarised in the following terms:

1. Both the TPC-A and the TPC-B transactions did not apply enough load on the tested systems, the TPC benchmarks response time at 1000 rows is not much different than their response time at 15,000 rows.
2. There was a significant deviation between the CITY transaction response time and the TPC transactions response time specially as the sizes of the tables increases.

Size in 1000 Rows	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TPC-A	1.23	1.33	1.35	1.32	1.34	1.37	1.39	1.41	1.39	1.38	1.39	1.42	1.40	1.40	1.43
TPC-B	1.38	1.48	1.52	1.52	1.55	1.57	1.56	1.58	1.60	1.63	1.61	1.65	1.65	1.66	1.67
CITY (386)	45	47	49	51	53	55	56	58	60	61	62	64	65	67	68
CITY (486)	47	50	51	51	52	55	55	56	58	58	60	60	62	62	64

Table 6.6, Comparison between The CITY and TPC-A, TPC-B scalability level

Range in 1000 Rows	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15
TPC-A	.1	.02	-.03	.02	.03	.02	.02	-.02	-.01	.01	.03	-.02	0	.03
TPC-B	0.1	.04	0	.03	.02	-.01	.02	.02	.03	-.02	.04	0	.01	.01
CITY (386)	2	2	2	2	2	1	2	2	1	1	2	1	2	1
CITY (486)	3	1	0	2	3	0	1	2	0	2	0	2	0	2

Table 6.7, Comparison between The CITY and TPC-A, TPC-B scalability level

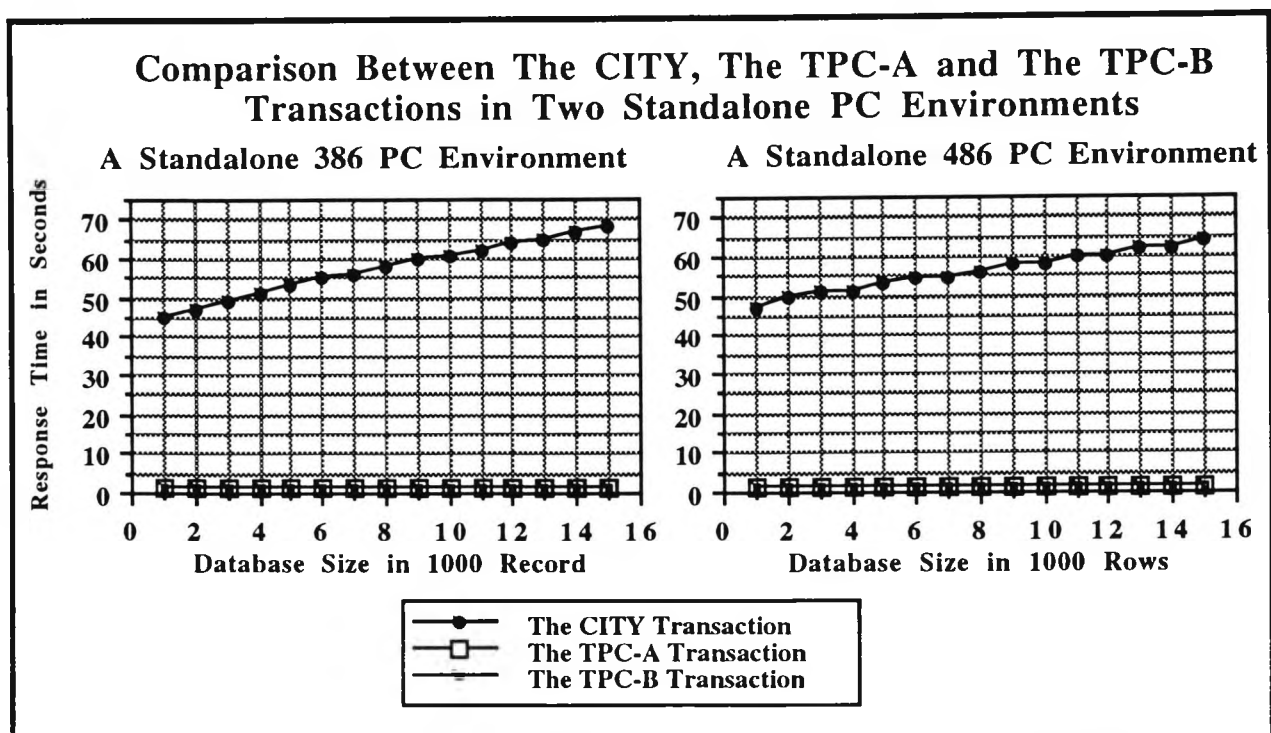


Fig. 6.7, Comparison between the CITY and TPC-A, B scalability levels

6.3.2 Comparison Between the CITY and the TPC Transactions' Scalability Ratios

This research looked at transaction scalability ratio as an important factor in comparing transactions' behaviour. The scalability ratio as discussed before eliminates any bias due to transaction cost. A comparison between the CITY transaction scalability ratio and the TPC transactions' scalability ratios were conducted in the two tested environments, the 386 and the 486. Fig. 6.8 and Table 6.8 present the results of the comparisons.

Range in 1000 Rows	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15
TPC-A	.08	.01	-.02	.01	.03	.01	.01	-.01	-.01	.01	.02	-.01	.01	.02
TPC-B	.07	.03	0	.02	.01	0	.01	.01	.02	-.01	.02	0	.01	0
CITY (386)	.04	.04	.04	.04	.04	.02	.04	.03	.02	.02	.03	.02	.03	.02
CITY (486)	.06	.03	0	.03	.03	0	.03	.03	0	.03	0	.03	0	.04

Table 6.8, Comparison between The CITY and TPC-A, TPC-B scalability ratio

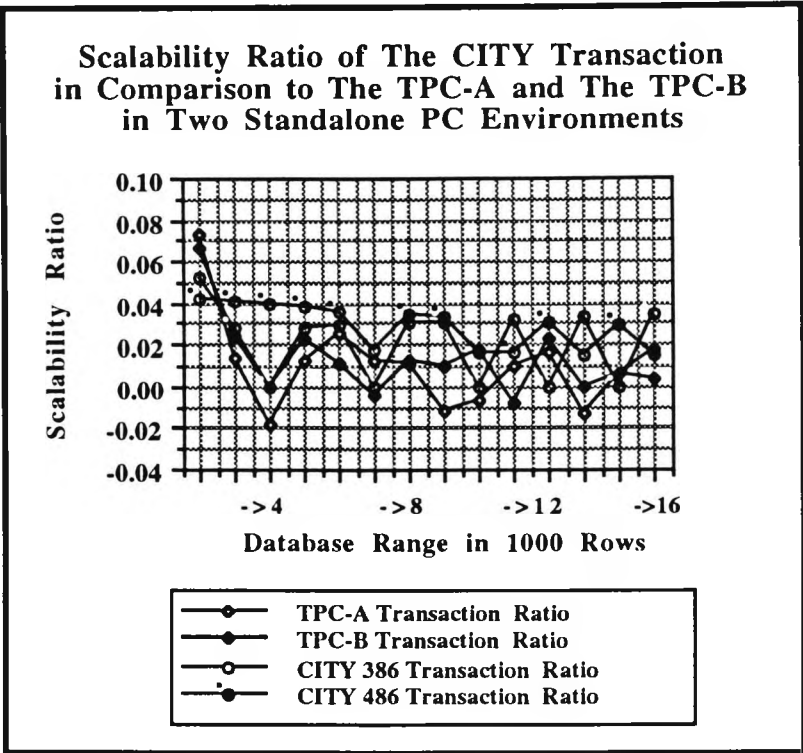


Fig. 6.8, Comparison between the CITY and TPC-A, B scalability ratios

6.4 Summary

This chapter discussed the preliminary test and verification of the CITY benchmark. The test and verification process aimed at examining the consistency and credibility of the benchmark results in a controlled environment. To maintain an overall control over the tested system parameters, the preliminary test took place in two standalone PC environments where all the parameters are controlled and programs can run in a real single user mode. This test phase took over 8000 computing hours of benchmark testing. Two PCs were running the benchmark 24 hours every day including weekends for over six month. The results from each run was printed and thoroughly examined to guarantee the accuracy of the results.

The CITY benchmark measures were found to be reproducible. The benchmark test was replicated for over 1000 times, the variations in the benchmark measures were highly insignificant.

One of the problems that were found was the first time measure. The computer clock gave an exceptionally high first reading before giving consistent readings for all

the subsequent results. To solve this problem, a dummy loop was created specially to eliminate those effects.

The verification also examined the CITY transaction atomicity, consistency and durability. As all verification runs were conducted in a single user mode, isolation was not tested.

Test of the CITY transaction scalability took the direction of testing transaction scalability under increasing load and stepping processing power one step. The benchmark was able to strain systems resources in the tested environments. The benchmark scalability was tested against eight other mixes selected according to resource utilisation requirement. The eight mixes ranged from simple low cost transaction to expensive transaction. Transaction cost was calculated as a function of response time, the more time it uses the more expensive it is. The CITY transaction scalability ratio was better than the scalability ratios of all the other mixes.

Finally, this chapter presented a comparison between the CITY benchmark transaction behaviour and the behaviour of the TPC benchmarks' transaction in two standalone PC environments. The comparison covered scalability levels and scalability ratios. The TPC benchmarks suffered limitations in both areas. Comparing scalability levels, the CITY benchmark transaction showed great diversion from the TPC benchmarks under all loads and the CITY benchmark produced better scalability ratio in all the tested environments.

This preliminary verification phase showed that the CITY benchmark has produced reliable and consistent results. The Transaction mix has proved superiority over a range of transaction mixes ranging from simple inexpensive transactions to expensive transactions. Additionally, the CITY transaction has shown superiority over the TPC benchmarks' transactions in all areas of comparisons. The next chapter presents the large scale test of the CITY benchmark. The large scale test examined the same performance parameters in addition to several other parameters that can not be tested in a PC environment. The large scale test took place in seven different computer environments ranging from SUN work stations to large mainframes. Those environments run wide variety of DBMS and run under different operating systems.

CHAPTER 7

LARGE SCALE TEST AND VERIFICATION of The CITY BENCHMARK

CHAPTER 7

LARGE SCALE TEST AND VERIFICATION of The CITY BENCHMARK

7.1 Introduction

Chapter six presented the preliminary test of the CITY benchmark. That chapter presented the test of the basic requirements of benchmark verification. This chapter presents the results of the large scale test of the CITY benchmark in several industrial organisations. The test was conducted in different environments ranging from standalone workstations to large mainframes and multiprocessors environments. The test covered some areas that were covered by the previous chapter and some other areas that can not be tested in a PC environment.

Large scale verification followed the same line of test that was presented in chapter six with additional test of portability, consistency between runs, and very large databases. All the presented results in this work has been calculated in a single user controlled environment except for the large mainframe environment which was very difficult to control due to the large number of users connected to the system. In each experiment, a single factor was tested at a time. That factor was measured independently of other factors.

The CITY benchmark large scale test was repeated in each of the computer environments presented above. The large scale verification process of the CITY benchmark tested the following factors:

- the benchmark measures are reproducible;
- the benchmark is hardware independent;
- the benchmark is software independent;
- the benchmark is DBMS independent;
- the benchmark is application independent;
- the benchmark measures are comparable between machines and DBMS;

- the benchmark is clear and simple to construct;
- the benchmark usage cost is low.

7.2 Reproducibility of the Benchmark Measures

Similar to what was done in the standalone environment, reproducibility of the CITY benchmark results was tested by replicating the benchmark run and comparing the results in every tested environment. For each environment, the CITY benchmark runs were replicated for at least fifteen times except for the multiprocessors environment where they were replicated for just seven times. For each environment, the basic characteristics of benchmark reproducibility were tested by examining the same factors that were discussed in chapter six (§6.2.1), as a reminder, those factors are the following.

- Clock adjustment and first transaction effect.
- Reproducibility of the benchmark measures within a single loop.
- Reproducibility of the benchmark runs.
- Duration of loops time.
- Duration of full test of DBMS.
- Transaction atomicity.
- Transaction consistency.
- Transaction isolation.
- Transaction durability.

7.2.1 Clock Adjustment and Eliminating First Transaction Effect

The clock adjustment problem that was found in the PC environment was found again in all other environments that were examined by this research. In all cases the first clock reading was exceptionally high and the first database operation took longer than all the subsequent operations. An example for that first time behaviour is illustrated in Fig. 7.1. The figure demonstrates the first time collected in a workstation environment and shows that also in this environment transaction response time took several iterations to settle down

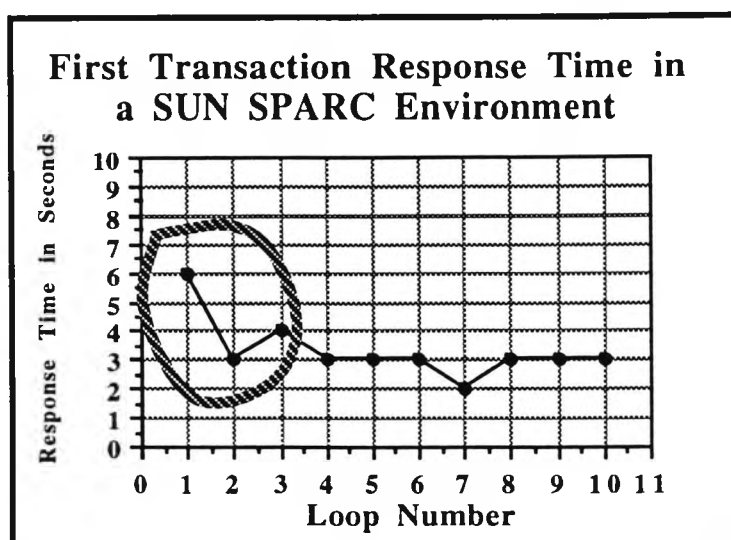


Fig. 7.1, First transaction response time in SUN SPARC environment

7.2.2 Reproducibility of the Benchmark Measures Within a Single Loop

Similar to the PC environment, the verification process examined the time measurements calculated from each loop. The variance between single loop response times was used as an index of spread of the collected measurements. The variance is discussed in details in chapter three (§3.6.1.2).

An example of the collected results is presented in Table 7.1. The table presents a subset of the collected loop times collected from a workstation environment under scalable database load. Each loop in that test was repeated over 2000 times before the variance was calculated. The same process was repeated for each set of loops representing different database load ranging from 5000 rows to 100,000 rows. The calculated variances of the loops presented in Table 7.1 are as follows: variance of 5000 rows = 0.5; variance of 10000 rows = 0.6; variance of 15000 rows = 0.3; and variance of 20000 rows = 0.8.

Transactions in the SUN SPARC environment followed a pattern of behaviour similar to that in the PC environment presented in chapter six. All calculated response times were concentrated around the calculated mean. The variance as presented above shows that the spread between the calculated response times is negligible and permits using loops mean response times as a representative value of response times of all loops. Fig. 7.2 shows the behaviour of a sample of fifty time measurement representing different loads.

Table 7.2, classifies the 50 response times sample collected from the SUN SPARC environment. Four tables' sizes are presented in that table, 5000 rows, 10,000 rows, 15,000 rows, and 20,000 rows. The results in this environment were similar to the results obtained from the PC environment. The collected response times showed very little spread between three or four response times and were highly centred around the sample arithmetic mean. Fig. 7.3, illustrates the distribution of the SUN SPARC fifty response times sample readings.

Loop number	2500 Rows	5000 Rows	7500 Rows	10000 Rows
Loop 1	6	5	3	3
Loop 2	3	3	3	4
Loop 3	4	3	4	7
Loop 4	3	3	3	5
Loop 5	3	3	3	6
Loop 6	3	3	3	4
Loop 7	2	3	3	5
Loop 8	3	3	3	4
Loop 9	3	2	3	4
Loop 10	3	3	3	3

Table 7.1, Spread of CITY measures in a SUN SPARC environment

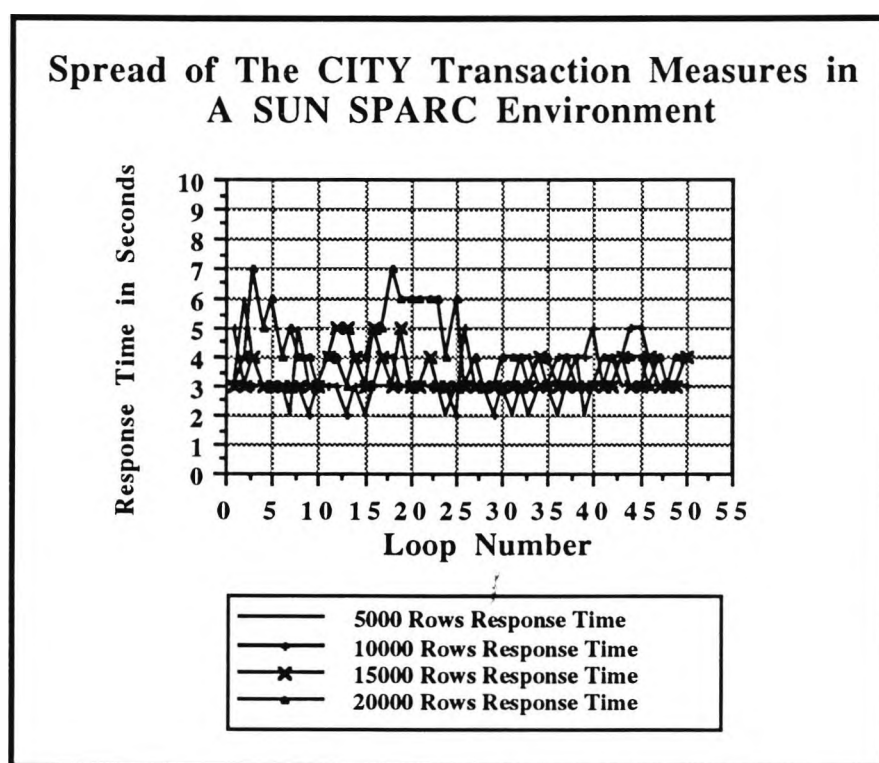


Fig. 7.2, Spread of CITY measures in SUN SPARC environment

Response Time in Seconds	Frequency of 5000 Rows	Frequency of 10,000 Rows	Frequency of 15,000 Rows	Frequency of 20,000 Rows
2	9	4		
3	36	29	37	13
4	3	12	9	24
5	1	5	4	4
6	1			7
7				2
	5000 Rows	10,000 Rows	15,000 Rows	20,000 Rows
	Average Response	Average Response	Average Response	Average Response
	2.98	3.36	3.34	4.22

Table 7.2, Frequency values of CITY measures in a SUN SPARC environment

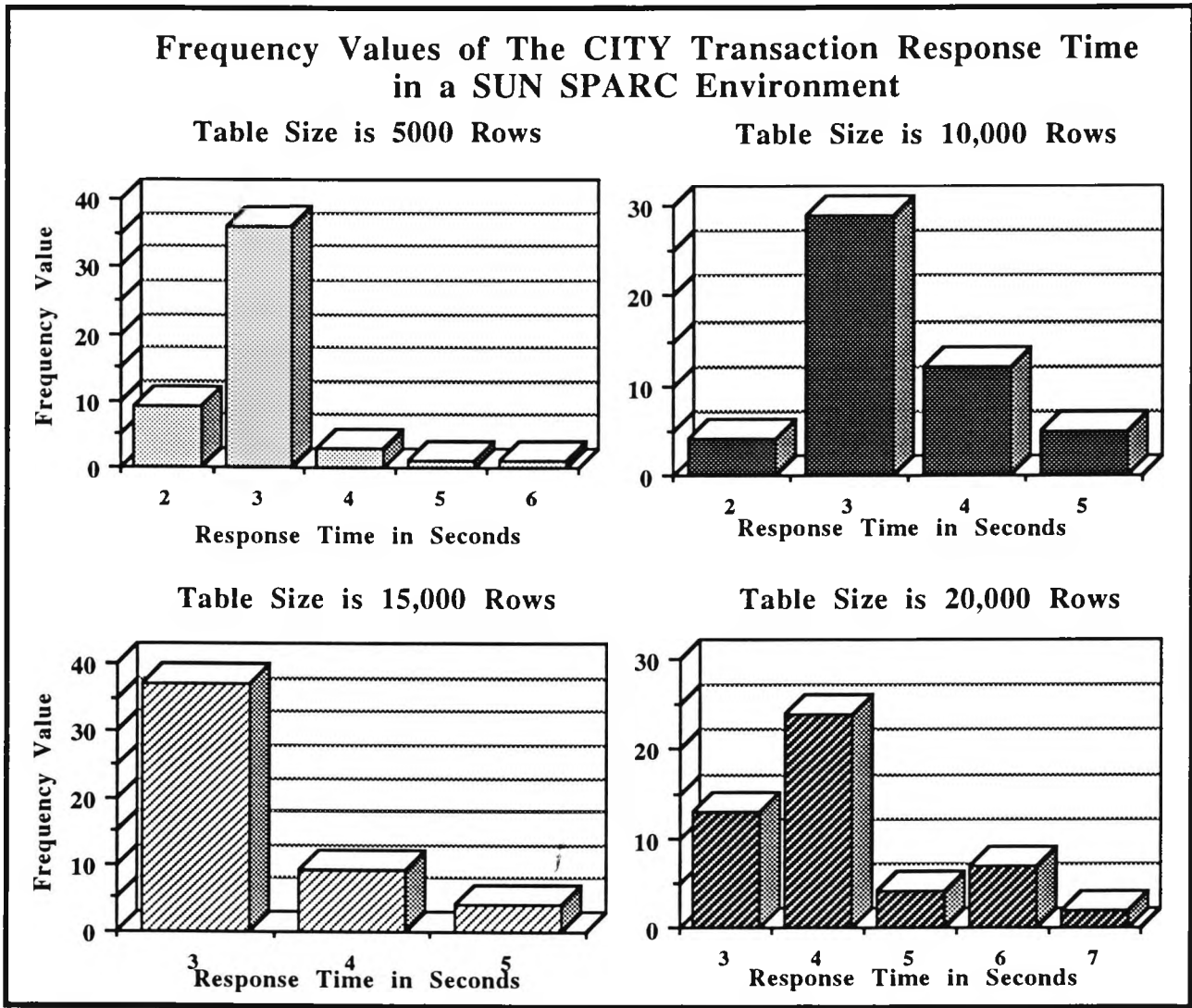


Fig. 7.3, Frequency values of The CITY measures in a SUN SPARC environment

7.2.3 Reproducibility of the Benchmark Runs

To compare the results obtained from the different runs, one way analysis of variance (ANOVA) was conducted. The one way ANOVA is a special case of more general problems in which one may have taken samples from a number of different classes, or categories and wish to test the hypothesis that the population means of the classes are equal. If the ANOVA result shows that there is no significant difference between all values, the mean of those values can be used to represent all the collected values.

An example of the CITY benchmark results is presented in tables 7.3, 7.4 and 7.5. The values in Table 7.3 are just a subset of the CITY benchmark results from the SUN SPARC environment or show how the ANOVA was calculated.

Database Size	Run 1	Run 2	Run 3	Run 4	Run 5
10,000 Rows	3.346	3.409	4.390	4.369	3.930
20,000 Rows	3.488	4.036	4.615	4.639	3.913
30,000 Rows	4.257	4.036	4.865	4.918	4.286
40,000 Rows	4.615	6.250	5.294	5.325	4.712
50,000 Rows	5.114	6.294	5.806	5.696	4.972
Average($\bar{x}_i$)	4.164	4.859	4.994	4.989	4.363
$\bar{\bar{x}}$	4.674	4.674	4.674	4.674	4.674
$(\bar{x}_i - \bar{\bar{x}})$	-0.510	0.185	0.320	0.316	-0.311
$(\bar{x}_i - \bar{\bar{x}})^2$	0.260	0.034	0.103	0.100	0.097

Table 7.3, Sample of The CITY measures from different runs

Table 7.3 presents the source of variation between columns due to difference between columns' means. From Table 7.3, variation between columns = 2.966.

Table 7.4 presents the residual due to difference between observations and columns mean. From Table 7.4, Residual between columns = 12.608.

The calculated ratio of F test for the between runs (rows) was 1.176. The critical value at 99% was 4.43 with 4 and 20 degrees of freedom. The calculated ratio of F test is highly insignificant indicating virtually no difference among the five runs. That result

indicates that the CITY benchmark runs were reproducible and there was no significant difference between the different test runs. It also indicated that users can use the overall average over the five runs as a representative value for each of the runs. Fig. 7.4 presents a graphical representation of the five runs.

Database Size	$X_1 - \bar{x}_i$	$X_2 - \bar{x}_i$	$X_3 - \bar{x}_i$	$X_4 - \bar{x}_i$	$X_5 - \bar{x}_i$
10,000 Rows	0.669	2.103	0.365	0.385	0.187
20,000 Rows	0.457	0.677	0.144	0.123	0.202
30,000 Rows	0.009	0.306	0.017	0.005	0.006
40,000 Rows	0.203	1.935	0.090	0.113	0.122
50,000 Rows	0.902	2.059	0.659	0.499	0.371

Table 7.4, Calculation of ANOVA of The CITY measures from different runs

Table 7.5 presents the ANOVA table, for observations given in Table 7.3.

Source	SS	DF	MS	F ratio= $MS_{\text{res}} / MS_{\text{res}}$	p-Value
Between Columns	2.966	4	0.741	$0.741 / 0.630 = 1.176$	$p < .01 = 4.43$
Residual	12.608	20	0.630		
Total	15.574	24			

Table 7.5, Calculation of ANOVA of the CITY measures from different runs

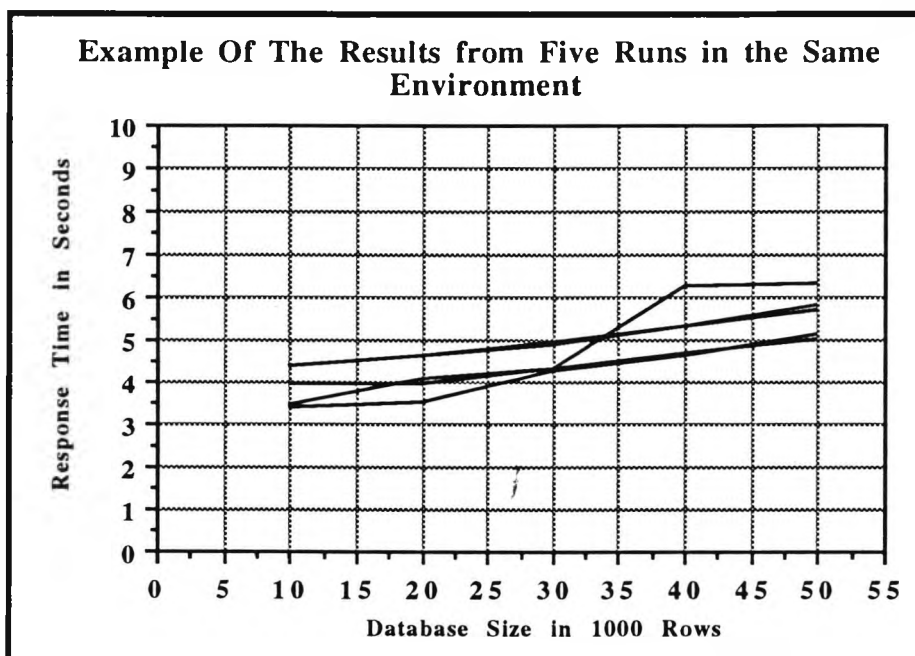


Fig. 7.4, Sample of The CITY measures from different runs

Bearing in mind that the original test of the previous example was repeated 35 five times and the database sizes for those runs varied from 5000 rows to 100,000 rows with 5000 rows' interval, also bearing in mind that the verification process was conducted in eight different environments, and reproducibility of the results was not just tested for the CITY runs but for all the test runs that were required for the verification stage, it was impossible to manually repeat the previous process for all the runs that were required for the verification test. Accordingly, the present author has developed a program written in C programming language to calculate the ANOVA of any given sample. The program obtained its input values from the benchmark result file and automatically calculated the ratio of F test for those obtained results. The program code is presented in appendix E and the ANOVA is discussed in detail in chapter three.

7.2.4 Duration of Loops Time

Testing loop time duration in the PC environments found that using 900 seconds loop was quit sufficient to obtain a good sample of measurements that enjoys highly insignificant variances. The same test was repeated in a SUN SPARC environment. That test indicated whether the originally selected loop time were good enough for larger environments or not.

In the SUN SPARC environment, loop duration was tested using 900 seconds loop, 1800 seconds loop, 3600 seconds loop and 9000 seconds loop. Each run was repeated thirist times to collect a reasonable number of measurements. The results of loop duration test are presented in Table 7.6.

Loop Time	10000 Rows	20000 Rows	30000 Rows	40000 Rows	50000 Rows
900 Seconds	6.122	5.732	6.207	5.806	7.031
3600 Seconds	6.143	5.210	5.180	4.993	5.634
9000 Seconds	4.836	8.721	6.461	5.784	6.589

Table 7.6, Comparison between CITY measures based on test loop duration
i

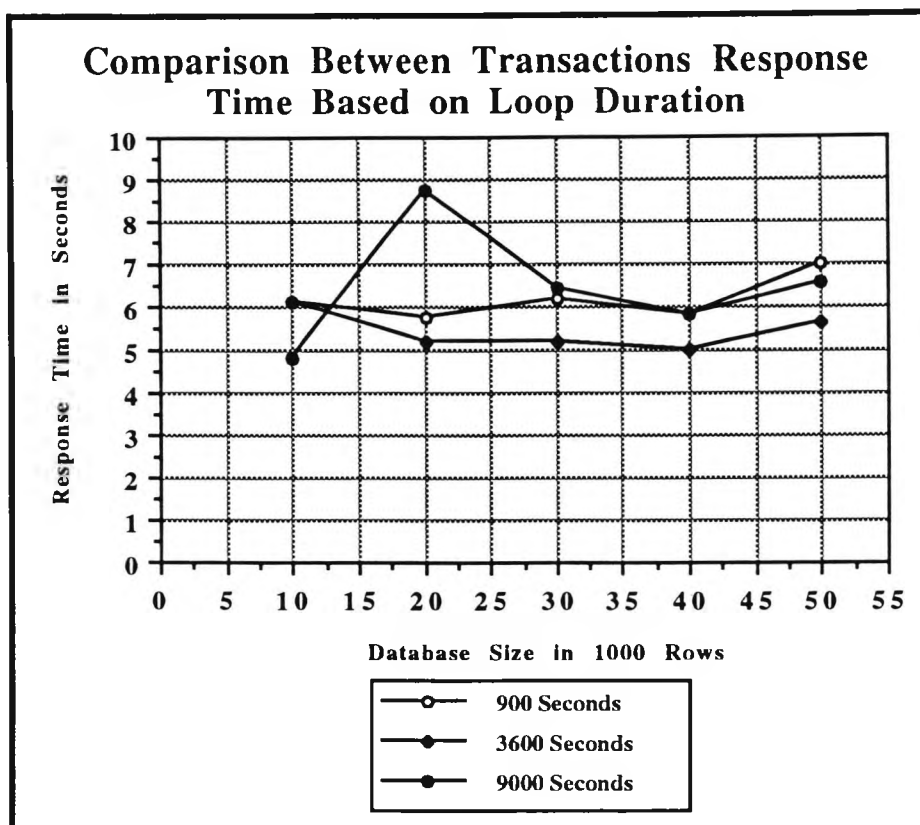


Fig. 7.5, Comparison between The CITY measures based on test loop duration

A one way analysis of variance ANOVA was calculated for the three different loop duration. The calculated ratio of F test for the between rows was 1.496. The critical value at 99% was 5.99 with 4 and 10 degrees of freedom. The calculated ratio of F test shows that the difference between rows is statistically insignificant indicating that any one of the three loop duration can be used to test database. Fig. 7.5 illustrates transaction patterns using the three loops' duration.

7.2.5 Duration of Full Test in Multi-User Environment

As presented before, all test runs for the CITY benchmark was conducted in a single user mode. That is to maintain overall control on the test measurement and allowing the ability to test one system against another. However, the CITY benchmark is designed to be used in multi-user mode in OLTP environment. In this case for all test runs steady state condition must be achieved before the measurement interval begins. That can be decided by examine data generated by earlier tests and empirically determine when the first log would happen, and consequently, the ramp-up period can be decided. The ramp-up period must guarantee that the database has performed at least

one log file switch and the system had reached steady state condition. The database to ramp down period must also be determined.

7.2.6 Test of Transaction Atomicity

The preliminary test in the PC environments guaranteed the atomicity of the CITY benchmark in that system. The same process was repeated in all the tested environments by:

1. running the benchmark for randomly selected records and verify that those records have been changed;
2. running the benchmark for randomly selected records substituting COMMIT transaction by ABORT. The records were checked to ensure that they had not been changed.

7.2.7 Test of Transaction Consistency

The CITY benchmark consistency was tested in large environments by applying the following steps:

- a. One random record was updated and its integer value is increased by one.
- b. Ten random records are updated and their integer values are increased by one.
- c. The insert database had one logical record added for each committed transaction, none for any aborted transactions.

The step showed that the CITY benchmark transaction took the database from one consistent state to another.

7.2.8 Test of Transaction Isolation

Isolation of transactions means that the operations of concurrent transactions must yield results which are indistinguishable from the results which would be obtained by forcing each transaction to be serially executed to completion in some order. This property is also called serializability.

This property was not tested during the benchmark verification process because all runs took place in a single user mode.

7.2.9 Test of Transaction Durability

The CITY benchmark durability was tested by forcing the tested systems to fail during the benchmark test by forcing a quit from the DBMS. The committed transactions were then checked for transaction completeness.

This test ensured the ability of the tested systems to preserve the effects of committed transactions. It also ensured the consistency of the database after recovery from database failures.

7.3 Portability of the CITY Benchmark

The problem of adapting benchmarks to different computer architecture is called benchmark portability. Ferrari [FERR78] defines portability as:

"The ability to transport workload W_i running on system S_i to S_n provided that W_i will not change"

In other words portability implies that, for any given benchmark script it should be hardware independent, software independent and database management system independent to satisfy portability requirements. Writing a benchmark in high-level programming language implies hardware independence provided that the compiler is available. And a benchmark based on logical rather than physical resources could be used as a common metric in performance comparison studies. Ferrari [FERR78], lists the following characteristics of the ideal benchmark for use in selection studies.

1. A benchmark should be coded in a standard high level language to minimise code conversion.
2. Debugged to the extent that results are predictable on all machines being compared.
3. Data files should be readily acceptable by all systems, also consistent with the architecture of each system so as to avoid unfair comparisons or lack of validity of results.

4. Benchmark conversion should be monitored to determine the ease of conversion of the benchmark to run on a particular system.
5. Automatic checks on benchmark results.

For portability reasons, the CITY benchmark is coded in C high-level language. The C high-level language compiler is widely available on all ranges of computers. Through the CITY benchmark test the program was performed in a wide range of environments, the C compiler was available in all those environments and the test did not require changing the benchmark programming code.

The CITY benchmark database management system data manipulation language is written in standard SQL interface. The SQL is the standard practice in the computer market and available for all ranges of database management systems.

The files resulted from the benchmark test are in a standard ASCII code or EBCDIC code that were the available codes for the computers used for the benchmark tests. The ASCII code and the EBCDIC code are readily accepted by all ranges of computer systems.

Some benchmark conversion was required due to the lack of some types of attributes in some of database management systems. The conversion was easy enough to be conducted by external users. The benchmark results were checked and examined after each run to check the correctness of the benchmark results. The benchmark portability was examined by testing three main portability features:

- the benchmark is hardware independent;
- the benchmark is software independent;
- the benchmark is database management system independent.

7.3.1 Hardware and Software Independence

The benchmark was constructed without bias toward any particular architecture. To test the benchmark portability between different architecture, it was tested on different levels of hardware running under different operating systems. The test computers ranged from standalone PC running under MS/DOS operating system to large mainframes running under MVS/XA operating system. In between, other tests took place in other environments running different operating systems such as UNIX and VMS. The test and verification process took place in the following environments.

- Standalone PC based on 386 Intel processor running under MS/DOS operating systems and using commercial RDBMS.
- Standalone PC based on 486 Intel processor running under MS/DOS operating systems and using commercial RDBMS.
- SUN SPARC running under UNIX operating systems and using commercial RDBMS in single user mode.
- SUN SPARC running under UNIX operating systems and using commercial RDBMS in single user mode with network connection (database management system run on a different computer to the tables).
- VAX 4000 running under VMS operating systems and using commercial RDBMS in single user mode.
- Teradata system 3 multiprocessing database machine.
- Teradata system 4 multiprocessing database machine.
- NCR 3600 multiprocessing computer running under UNIX operating systems and using commercial RDBMS in single user mode.
- IBM 3090 mainframe computer running under MVS/XA operating systems and using commercial RDBMS.

7.3.2 Database Management System Independent

The second objective was to prove that the CITY benchmark is database management system independent. As presented before, the CITY benchmark database operations (transaction script) were written in standard ANSI SQL language. The SQL is the standard interface of most of the running database management systems in the market. The CITY benchmark was tested using the leading database management systems in the market, ORACLE, INGRES, DB2, and Teradata database machine. Running the benchmark under different database management systems did not require changing the benchmark script.

7.4 Large Scale Test of the Scalability of the CITY Transaction

Scalability implies portability, but includes some factors that are scalable but not portable. For example, floating point operations that are scalable but not portable due to the difference in their precision and round-off error. Benchmark scalability could be defined as:

"the ability of the benchmark test to strain DBMS resources regardless of the architecture and the amount of resources available for the benchmark test"

7.4.1 Large Scale Test for Scalability

Database management systems are available on a variety of architectures and operating systems ranging from standalone PC to large mainframes and parallel processors. Depending on the hardware and the available resources, a benchmark workload that might be adequate to strain the resources of one system might be inadequate to test scalability of another. Workloads of general purpose benchmarks should be adequate to test different DBMS on all ranges of computers and should be able to scale up different systems regardless of the capabilities and architecture of the system under test.

Scalability in this phase took three directions: increasing database size, increasing processor power, and increasing number of available processors (parallel database machines). Comparing different database management systems was achieved by scaling up database size. Scaling up processor power tested CPU intensive operations. Scaling up number of processors tested I/O intensive operations (JOIN).

Scalability of the CITY transaction was tested in several computer environments ranging from SUN SPARC workstations to large mainframe computers. The CITY benchmark was also tested in multi-processors' environments supporting up to four parallel processors. In designing for scalability, the experiments tested the CITY transaction under the following scalability loads:

1. scaling up tables' sizes from 1000 rows to one million rows;
2. scaling up the tested system processor power from a standalone 386 PC to a large mainframe.
3. scaling up the number of serving processors from a single processor system to four parallel processors.

In all cases the CITY transaction took scaling behaviour. Fig. 7.6 demonstrates the CITY transaction behaviour in all the tested environments. Regardless of database size, processor power and number of parallel processors the CITY transaction applied testable load.

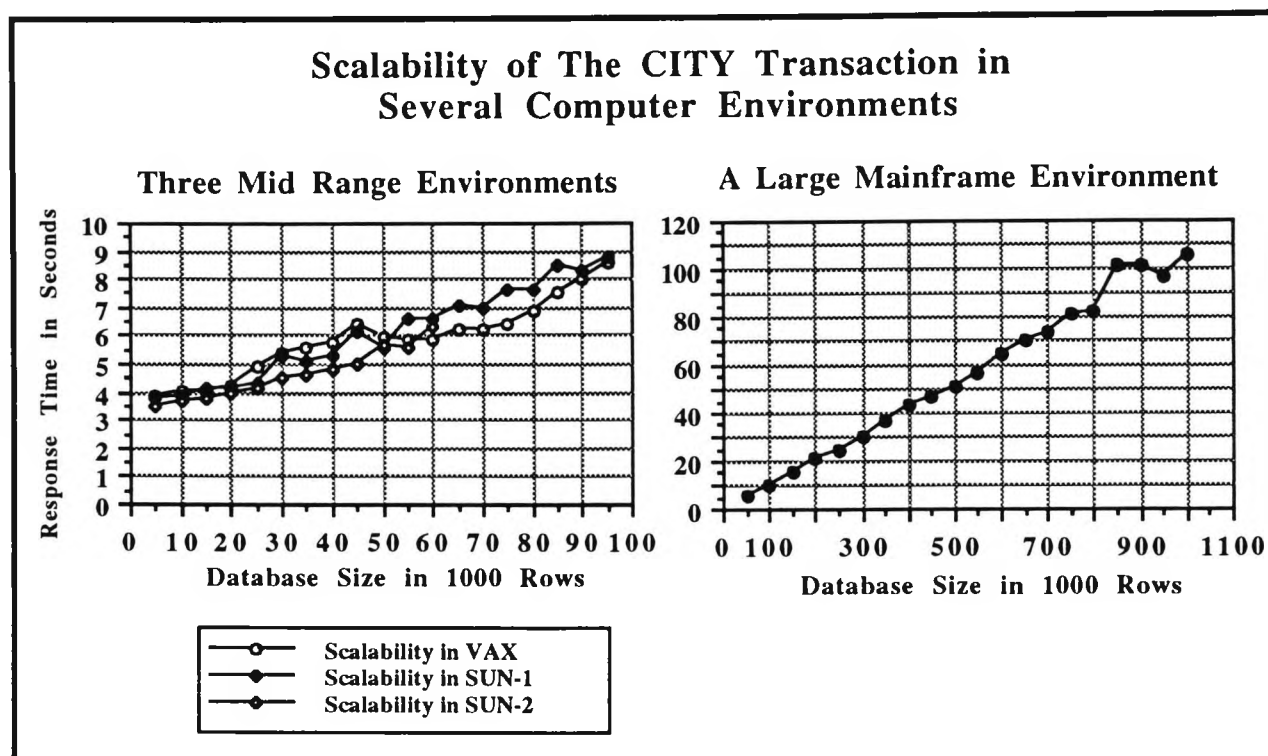


Fig. 7.6, Scalability of The CITY measures in several computer environments

7.4.2 Transaction Scalability VS Usage Cost in a Range of Computer Environments

Chapter six (§6.2.2.1), showed that transaction load of the CITY benchmark produced better scalability ratio than eight different mixes of transactions in two standalone PC environments. This phase tested the same eight mixes that were presented in chapter six against the CITY transaction workload in a SUN SPARC environment. Those transactions varied in basic database operations and transaction workload.

The empirical studies tested eight mixes that varied in workload and transaction cost. Scalability of the eight mixes were tested against increasing database size. Each mix consisted of twenty database operations to equal the number of database operations in the CITY benchmark script. To remind the reader, the mixes are presented below.

1. Mix 1: One row qualified retrieval + one insert + one row qualified update.
2. Mix 2: 100% one qualified update + one row qualified retrieval + one row insertion.

3. Mix 3: 100% one row insertion + one row qualified retrieval + one row qualified update.
4. Mix 4: 75% one row qualified retrieval + 25% Ten rows sequential retrieval + one insert + one row qualified update.
5. Mix 5: 55% one row qualified retrieval + 50% Ten rows sequential retrieval + one insert + one row qualified update.
6. Mix 6: 25% one row qualified retrieval + 75% Ten rows sequential retrieval + one insert + one row qualified update.
7. Mix 7: Ten rows sequential retrievals + one insert + one row qualified update.
8. Mix 8: JOIN mix that accesses two tables and returns 100 rows.

7.4.3 Scalability of The CITY Transaction in Comparison To Different Mixes in SUN SPARC Environment

The comparison in the SUN SPARC environment produced similar results to the PC environments. It showed that the CITY transaction has produced scalability levels higher than that of other tested mixes. The results in SUN SPARC environment are presented in Table 7.7. Fig. 7.7, summarise the results of testing the CITY transaction scalability pattern in comparison to the patterns of other mixes.

The CITY transaction scalability has produced better result because as shown in chapter three, the basic components of some the tested mixes such as the insert mix and the update mix are not scalable, therefore the mix itself could never be scalable no matter the database size or the CPU speed. Even after adding some other database operations, such as qualified retrieval operation, the mix scalability still not high enough for testability. For some other mixes such as the sequential retrieval mix and the 25% sequential retrieval mix, despite the expensive transaction mix, due to the relatively high number of I/O operations, the mixes do not translate that cost to higher value when moving from one database size to a higher database size.

Similar to the PC environments, the test showed that expensive mix does not guarantee transaction scalability, but the overall mix scalability is a function in all the database operations contained in that mix. For example, JOIN mix (mix 8) that is expensive mix does not produce the best scalability level. Fig. 7.8, illustrate the CITY transaction scalability in comparison to JOIN mix scalability.

Database Size	20,000 Rows Response Time	40,000 Rows Response Time	60,000 Rows Response Time	80,000 Rows Response Time	100,000 Rows Response Time
CITY	5.634	5.357	6.721	8.654	9.678
Mix 1	0.818	0.812	0.819	0.732	0.733
Mix 2	0.430	0.440	0.440	0.470	0.480
Mix 3	0.188	0.188	0.147	0.134	0.133
Mix 4	1.205	1.059	1.037	1.056	1.103
Mix 5	1.602	1.625	1.233	1.192	1.238
Mix 6	2.083	1.604	1.787	2.560	1.981
Mix 7	1.259	1.365	1.745	1.809	1.996
Mix 8	11.688	9.677	10.843	11.392	13.043

Table 7.7, The CITY transaction scalability level against other transactions' mixes

Scalability Level	Δx_1 40,000-20,000	Δx_2 60,000-40,000	Δx_3 80,000-60,000	Δx_4 100,000-80,000
CITY	-0.277	1.364	1.933	1.024
Mix 1	-0.006	0.007	-0.087	0.001
Mix 2	0.010	0.000	0.030	0.010
Mix 3	0.000	-0.041	-0.013	-0.001
Mix 4	-0.147	-0.022	0.019	0.046
Mix 5	0.023	-0.392	-0.040	0.045
Mix 6	-0.479	0.183	0.773	-0.579
Mix 7	0.106	0.380	0.064	0.187
Mix 8	-2.011	1.166	0.549	1.651

Table 7.8, The CITY transaction scalability level against other transactions' mixes

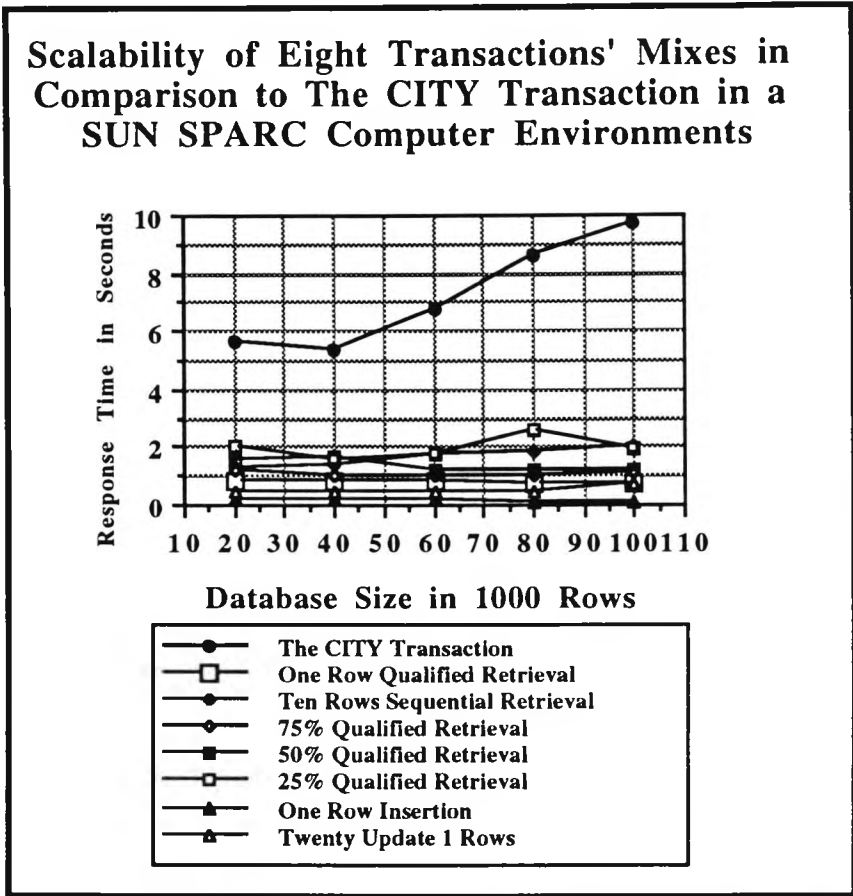


Fig. 7.7, The CITY transaction scalability level against other transactions

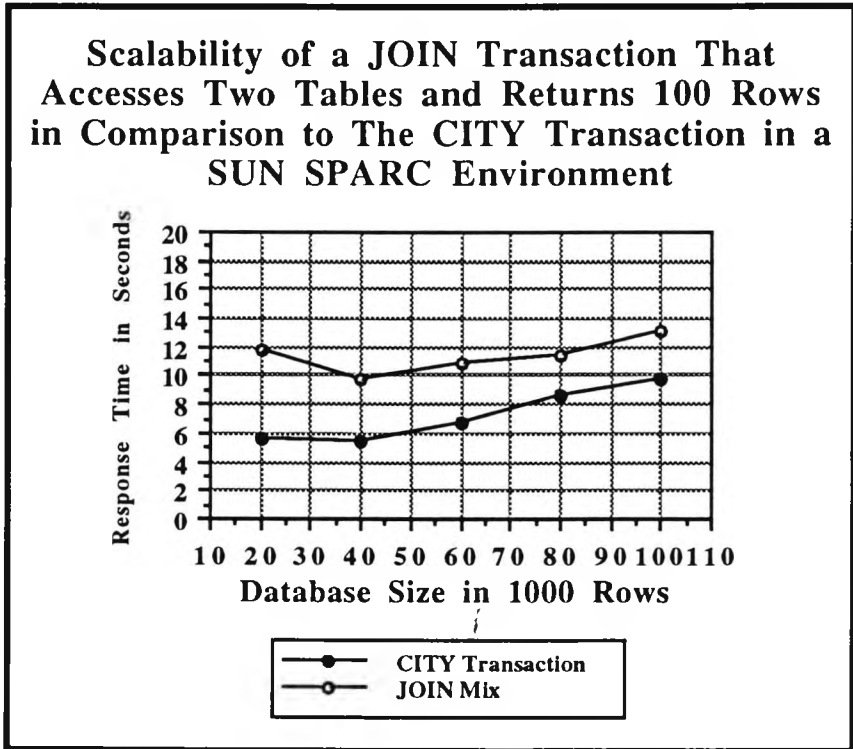


Fig. 7.8, The CITY transaction scalability level against JOIN transaction

7.4.4 Scalability Ratio of The CITY Transaction in Comparison To Different Mixes in Three Computer Environments

This section compares the scalability ratio of the CITY transaction to the scalability ratio of the eight mixes presented in the previous section. The results of this comparison is similar to the result found in the standalone PCs environments. The result of the comparison in the SUN SPARC environment is presented in Table 7.9.

Fig 7.9 illustrates transactions' scalability ratio in the SUN SPARC environment. It shows that the CITY transaction scalability ratio was better than all the scalability ratios of all other eight mixes. That is because some transactions consist of database operations that originally produce low scalability ratios such as qualified retrieval of one row using unique key which in many cases produces a negative scalability ratio due to its index dependence. On the other hand some other transactions could produce high levels of scalability, yet they are too expensive to produce equally high scalability ratio. The JOIN mix (mix 8), can be used as an example because it is the most expensive mix produces the lowest scalability ratio. This can be referred to the relatively high cost of transaction execution that swamps the benefit from the level of the workload.

Scalability Ratio	$\Delta y_1/y_1$ 20->40	$\Delta y_2/y_2$ 40->60	$\Delta y_3/y_3$ 60->80	$\Delta y_4/y_4$ 80->100	Sum of Ratios
CITY	-0.049	0.255	0.288	0.118	0.611
Mix 1	-0.007	0.009	-0.106	0.001	-0.104
Mix 2	0.023	0.000	0.068	0.021	0.113
Mix 3	0.000	-0.218	-0.088	-0.007	-0.314
Mix 4	-0.122	-0.020	0.019	0.044	-0.080
Mix 5	0.014	-0.241	-0.033	0.038	-0.222
Mix 6	-0.230	0.114	0.433	-0.226	0.091
Mix 7	0.084	0.278	0.037	0.103	0.503
Mix 8	-0.172	0.120	0.051	0.145	0.144

Table 7.9, The CITY transaction scalability ratio against other transactions' mixes

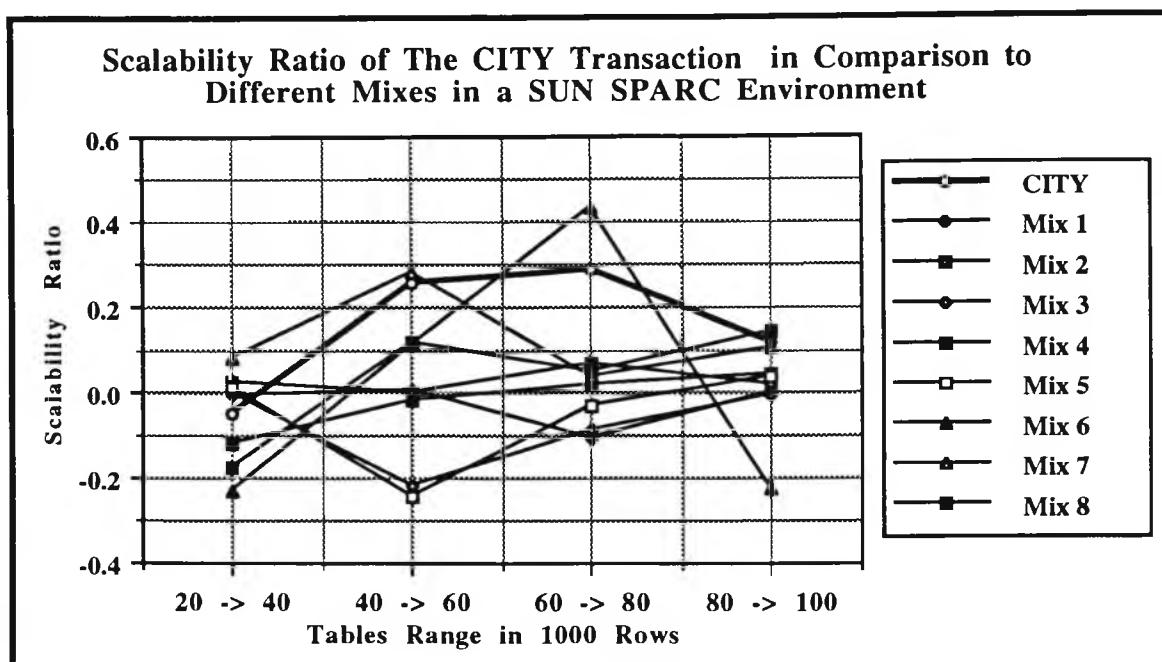


Fig. 7.9, The CITY transaction scalability ratio against other transactions' mixes

7.5 Comparability of The Benchmark Measures

Comparability of test results is defined by "the ability to produce results that can be meaningful and comparable from one system to another".

Comparisons between systems could be made based on one of three main indices:

- productivity;
- responsiveness;
- resource utilisation.

Resource utilisation depends on some non-DBMS factors such as platform specification. Hence, response time and throughput have been used as the database industry standard practice to compare different systems. In the CITY benchmark produces both response time and system throughput that is calculated as the reciprocal of response time for each tested DBMS. Those measures could then be used as basis for different DBMS comparisons. In comparing DBMS systems, the CITY benchmark could give the tested system throughput as number of CITY TPS or calculates systems' performance at specified database size. In this work all examples are based on comparing response time, but CITY TPS could be equally used. This research did not consider factoring system cost to system cost comparison because system cost is not a

reliable figure as the cost of different components may depend on a variety of factors outside the domain of the benchmark test and their cost may vary depending on things such as price reduction and technology advancement.

This methodology could be used for either to compare one DBMS behaviour running on different architectures or more than one DBMS running on the same or different architecture. The main advantage of this methodology is its ability to give DBMS user a clear understanding about the expected behaviour of the tested database management system under different conditions and loads.

Table 7.10 and Fig. 7.10 present comparative measurements of two different DBMS where response time has been used as the comparison performance index. It was measured at scaled up tables' sizes to compare DBMS behaviour under scalable load.

Size 1000 Rows	10	20	30	40	50	60	70	80	90
System 1	4.045	4.245	5.341	5.714	5.980	5.882	6.207	6.897	8.036
System 2	3.889	4.138	5.326	5.239	5.576	6.573	6.983	7.615	8.332

Table 7.10, Comparison between two DBMS using The CITY benchmark

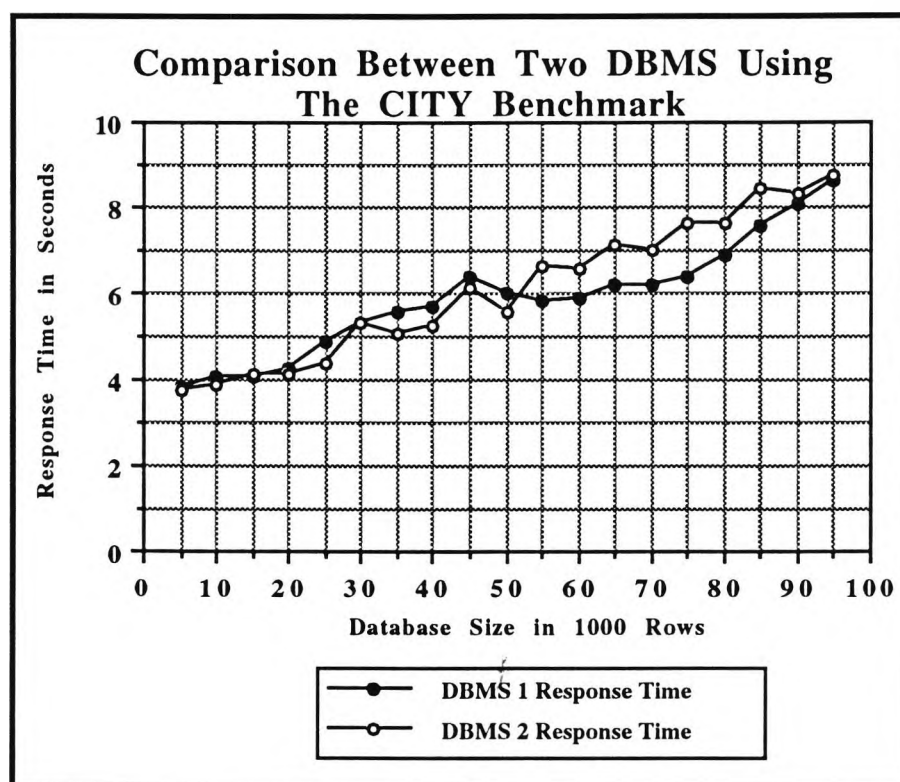


Fig. 7.10, Comparison between two DBMS using The CITY benchmark

7.6 Summary

This chapter discussed large scale test and verification of the CITY benchmark, which is phase two in the benchmark test. As presented in chapter six, the CITY benchmark results were reproducible, consistent and portable. The large scale test aimed at investigating whether those characteristics still exist in larger environments.

To that end, several experiments were conducted in a wide range of computer environments. The experiments took place in several computer environments ranging from standalone SUN workstation environment to large mainframes and multiprocessors environments. The CITY benchmark measures reproducibility was tested by repeating the benchmark test for at least fifteen times, to gather a representative sample size, in every tested environment, then applying a one way analysis of variance (ANOVA). In all environments, the variations in the benchmark measures were highly insignificant. This verification phase took over 2000 computer hours.

The tests of the CITY benchmark portability examined three main portability features, hardware independence, software independence and database management system independence. Portability was tested by running the benchmark in eight different environments varying in hardware configuration and their operating systems. Additionally, the benchmark script was run under different database management systems including the three leading database management systems in the market, DB2, Oracle and Ingres. Because the benchmark script was written in C high level language and standard SQL interface commands, no changes were required to run the program from one environment to the other and the benchmark was portable between all the tested environments.

Test of the CITY transaction scalability took three directions. The first direction was testing transaction scalability under increasing load. The second direction was testing transaction scalability under increasing processing power. The third direction was the ability to test database management systems running in multi-processors' computers. The benchmark was able to strain systems resources in the tested machine. The benchmark scalability was tested against eight other mixes selected according to resource utilisation requirement. The eight mixes ranged from simple low cost transaction to expensive transaction. Transaction cost was calculated as a function of response time, the more time it uses the more expensive it is. The CITY transaction scalability ratio was better than all the other mixes' scalability ratio.

This phase has also examined the CITY transaction atomicity, consistency and durability. As all verification runs were conducted in a single user mode, isolation was not tested.

This chapter demonstrated how the CITY benchmark could be used to evaluate the performance of a specific database management system or compare different database management systems.

By the end of this chapter it can be concluded that the CITY benchmark has proved a reliable tool in evaluating database performance. The Transaction mix has proved superiority over a range of transaction mixes ranging from simple inexpensive transactions to expensive transactions.

CHAPTER 8

DISCUSSION

CHAPTER 8

DISCUSSION

8.1 Introduction

This chapter presents a comparison between the CITY benchmark transaction results and the TPC benchmarks (TPC-A, TPC-B and TPC-C) transactions. The comparison demonstrated the technical limitations of the results of the TPC benchmarks and explains the advantages of the CITY benchmark transaction over those benchmarks. The comparison was conducted in different computer environments ranging from SUN workstations to large mainframes and multiprocessors environments and using the most widely used DBMS. Those environments were the following :

- SUN 386i workstations environment;
- SUN SPARC workstations environment;
- SUN SPARC workstations environment using network;
- VAX environment;
- IBM mainframe environment;
- NCR 3600 multiprocessing environment;
- Teradata System 3 multiprocessing environment;
- System 4 multiprocessing environment.

Those environments run different operation systems. Those operating systems are the following:

- MS DOS;
- UNIX;
- VMS;
- IBM MVS;
- IBM VM;

The CITY benchmark was also tested using wide the most widely used DBMS. Those DBMS are the following:

- ORACLE version 5;
- ORACLE version 6;
- INGRES version 6;
- DB2 for large mainframes;
- Teradata.

This chapter also presents the pragmatic application of the CITY benchmark. It presents several examples of the utilisation of the CITY benchmark in real life and how it is accepted in the database industry.

8.2 Comparison between the Behaviour of the CITY Transaction and the Behaviour of the TPC Benchmarks A and B Transactions

Chapter six presented a comparison between the TPC benchmarks A and B. The comparison showed that the CITY transaction has produced better results than the TPC benchmark in two standalone PC environments. This chapter extends this comparison to seven other environments ranging from SUN workstation to large mainframe environments. The comparison between the CITY benchmark transaction and the TPC benchmarks A and B were based on the following criteria.

- Scalability levels of the benchmark transactions.
- Scalability ratios of the benchmark transactions.
- Comparability measures of the benchmark transactions between DBMS.
- Compliance with future trends.

In the following section the TPC benchmarks (TPC-A, TPC-B and TPC-C) are compared to the CITY transaction in each one of the previous areas. The comparison showed that the CITY transaction produced better results in each of the four areas.

8.2.1 Comparison Between the Scalability Levels of the CITY Transaction and the TPC (A and B) Transactions

The basic database operations of the transactions of the TPC-A and the TPC-B were used in this test. Those database operations were used against increasing database size. The studies covered several levels of computer architectures and the three leading DBMS in the database market. Tables' sizes were scaled up gradually from 1000 rows to 1,000,000 rows. Record size in all experiments was 200 bytes. The main findings from the comparison can be summarised in the following points:

1. The TPC transactions' workload was not indicative of computing power in any of the studied computer architectures, that is despite the wide difference in the available resources for each environment.
2. Both the TPC-A and the TPC-B transactions transaction did not apply enough load on the tested systems even at 1000,000 rows and the TPC benchmarks response time at 5000 rows is not much different than their response time at 1000,000 rows.
3. There was a significant deviation between the CITY transaction response time and the TPC transactions response time specially at larger databases' sizes.

The experiments revealed the same patterns of behaviour between the CITY transaction and the TPC transactions when those transactions were tested in a range of computer environments, those environments are listed below.

- Standalone PC based on 386 Intel processor.
- Standalone PC based on 486 Intel processor.
- Two SUN SPARC environments.
- VAX 4000.
- Teradata system 3 multiprocessing database computer.
- Teradata system 4 multiprocessing database computer.
- NCR 3600 multiprocessing database computer.
- IBM 3090 mainframe computer. }

Those findings agreed with the research hypothesis that because the TPC benchmarks consisted of operations that are not scalable, the result from mixing those operations will not be scalable, on the other hand, because the CITY transaction

contains scalable database operations, it will produce better scalability level than the TPC transactions. An illustration of the findings are presented in Fig. 8.1 and Fig. 8.2.

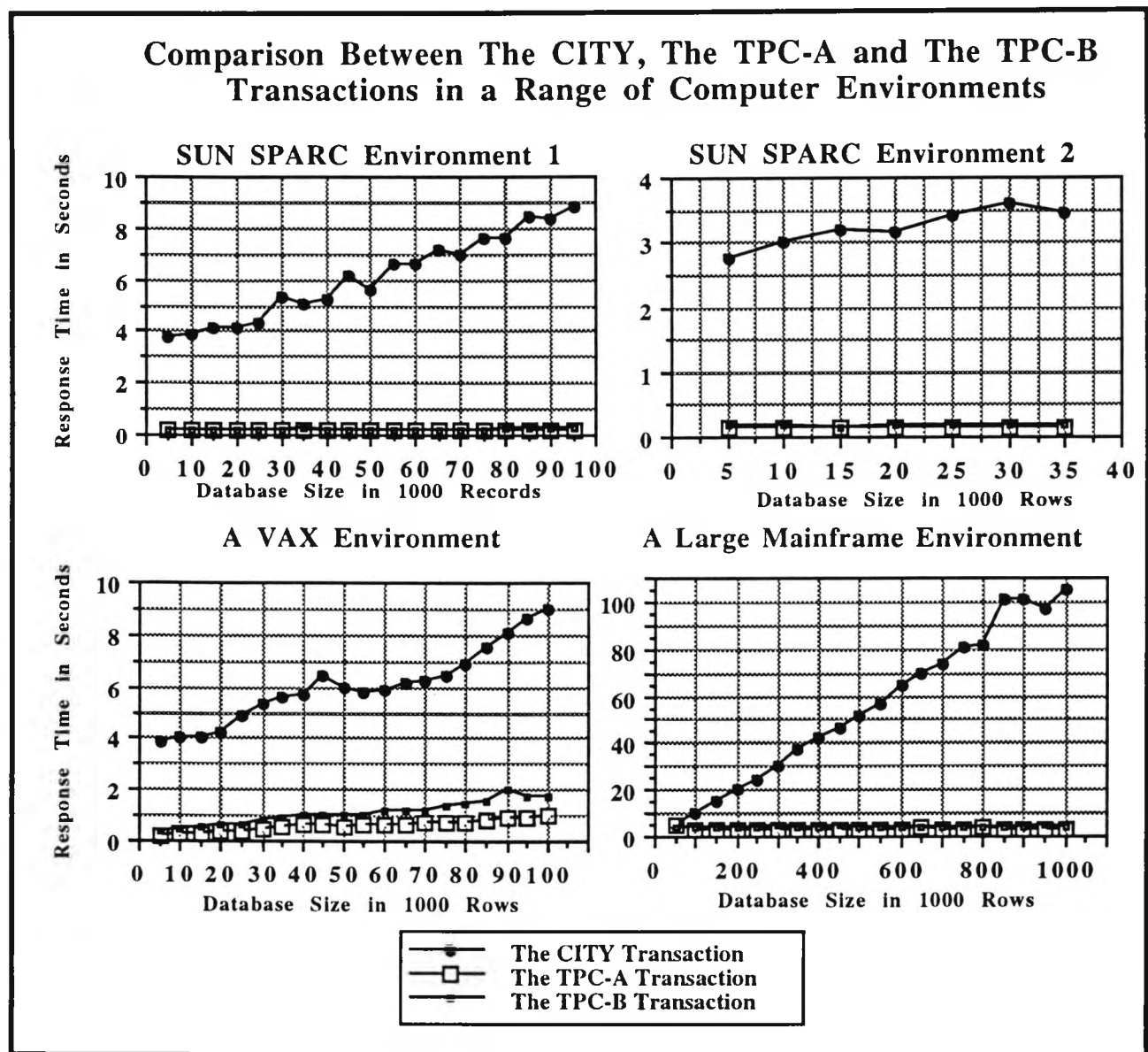


Fig. 8.1, Comparison between The CITY and TPC-A, TPC-B in several computer environments

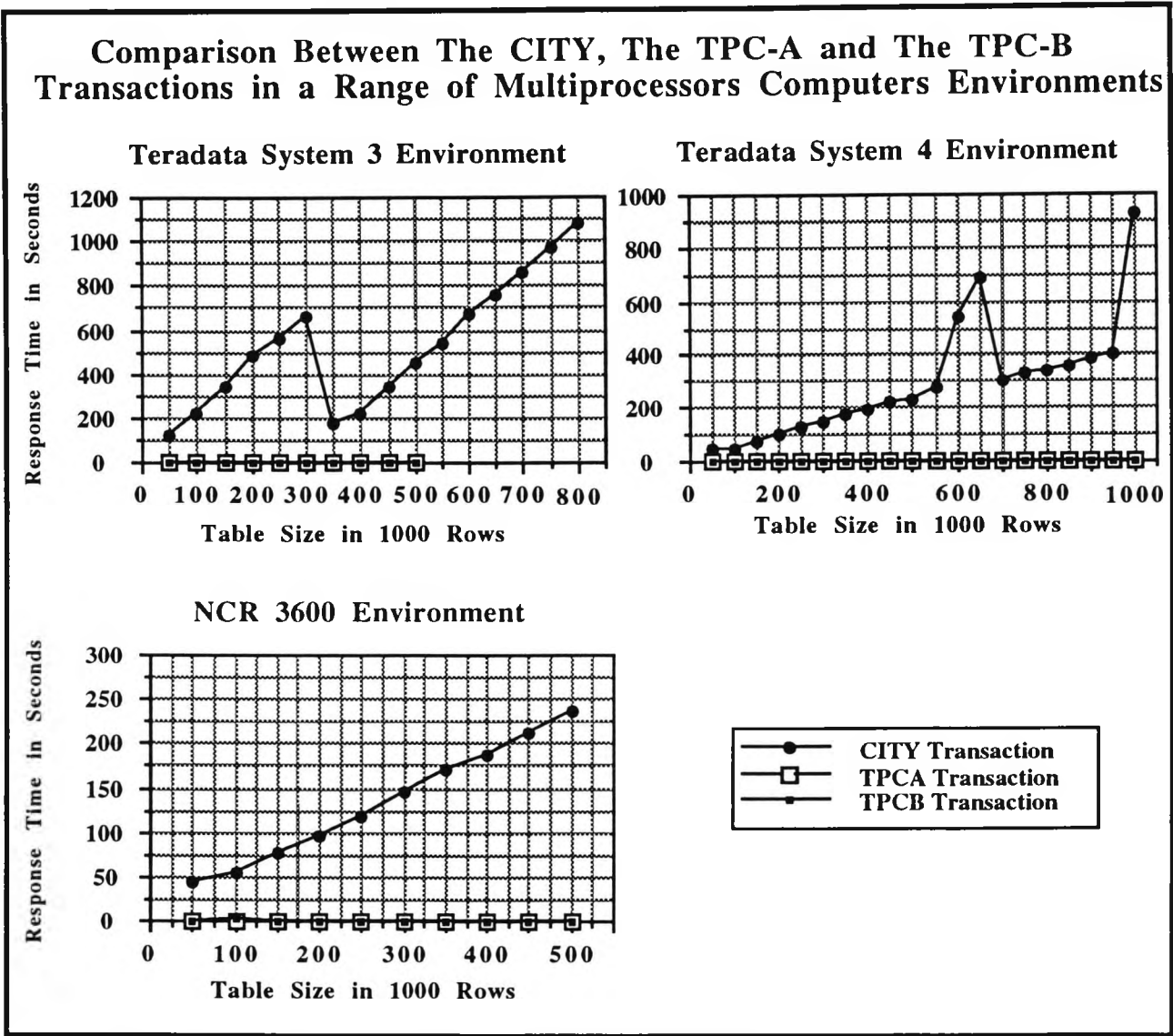


Fig. 8.2, CITY and TPC-A, TPC-B in multiprocessors environments

8.2.2 Comparison Between the CITY and the TPC Transactions' Scalability Ratios

A comparison between the CITY transaction scalability ratio and the TPC transactions' scalability ratios were conducted in all the previous environments. Fig. 8.3 and Fig. 8.4, present the results of the comparisons. The result can be summarised in the following points:

1. In general, the CITY transaction has produced a better scalability ratio.
2. When the experiments used small tables' sizes, less than 100,000 rows, the CITY transaction scalability ratio was 20% on average better than the TPC transactions scalability ratios. When tables' sizes increased, over 100,000,

- the CITY transaction started to deviate significantly from the TPC transactions and produced 40% on average better scalability ratio.
3. The previous pattern was found with increasing processing power. As processing power increased, the CITY transaction produced better scalability ratio than that of the TPC transactions.
 4. At the mainframe environment, which represents the real index for any test, as tables' sizes and processing power were increased the maximum of all experiments, the CITY transaction showed significant diversion from the TPC transactions.

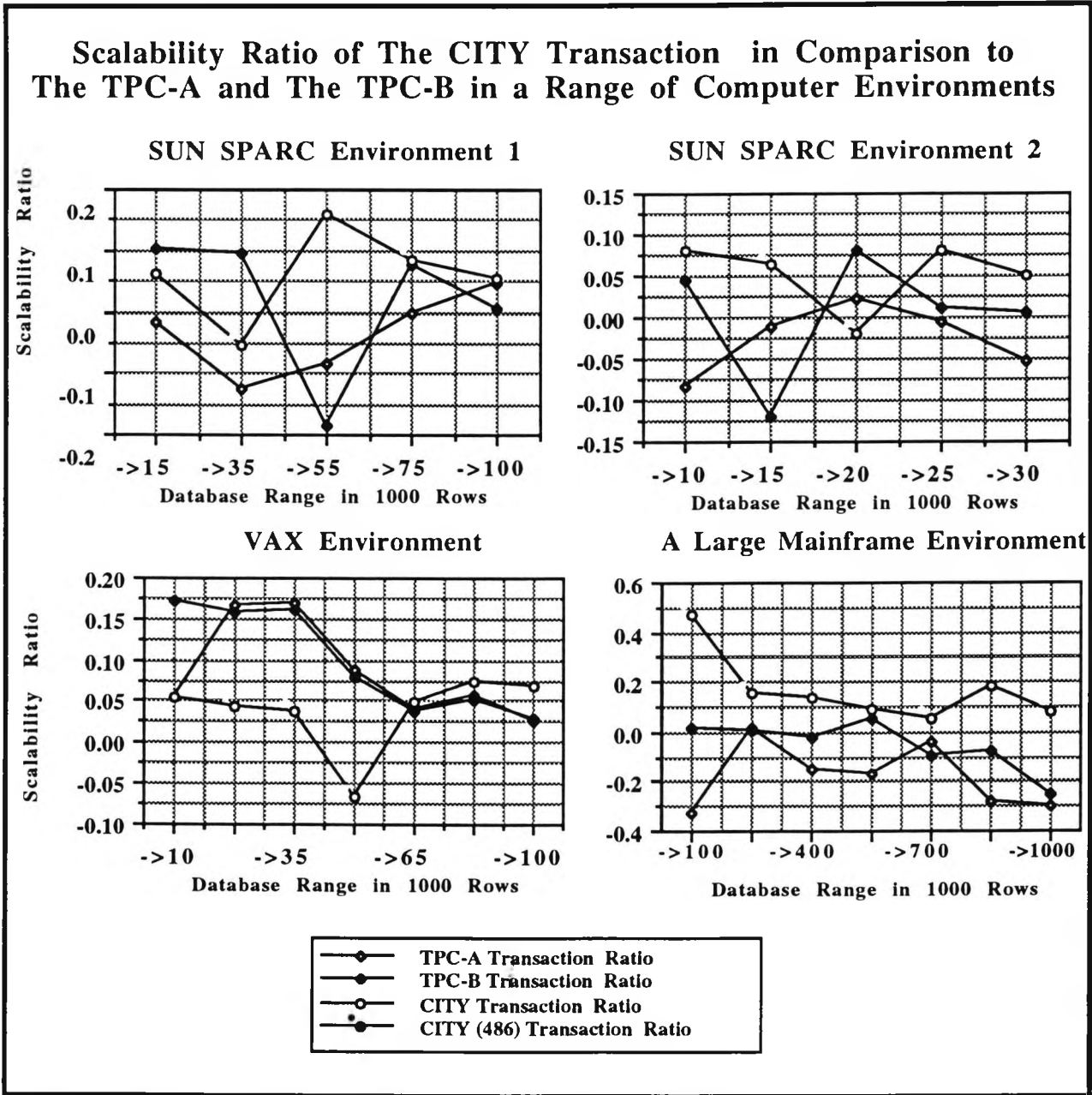


Fig. 8.3, Scalability ratio of The CITY and TPC-A, TPC-B in several computer environments

Scalability Ratio of The CITY Transaction in Comparison to The TPC-A and The TPC-B in a Range of Multiprocessors Computers Environments

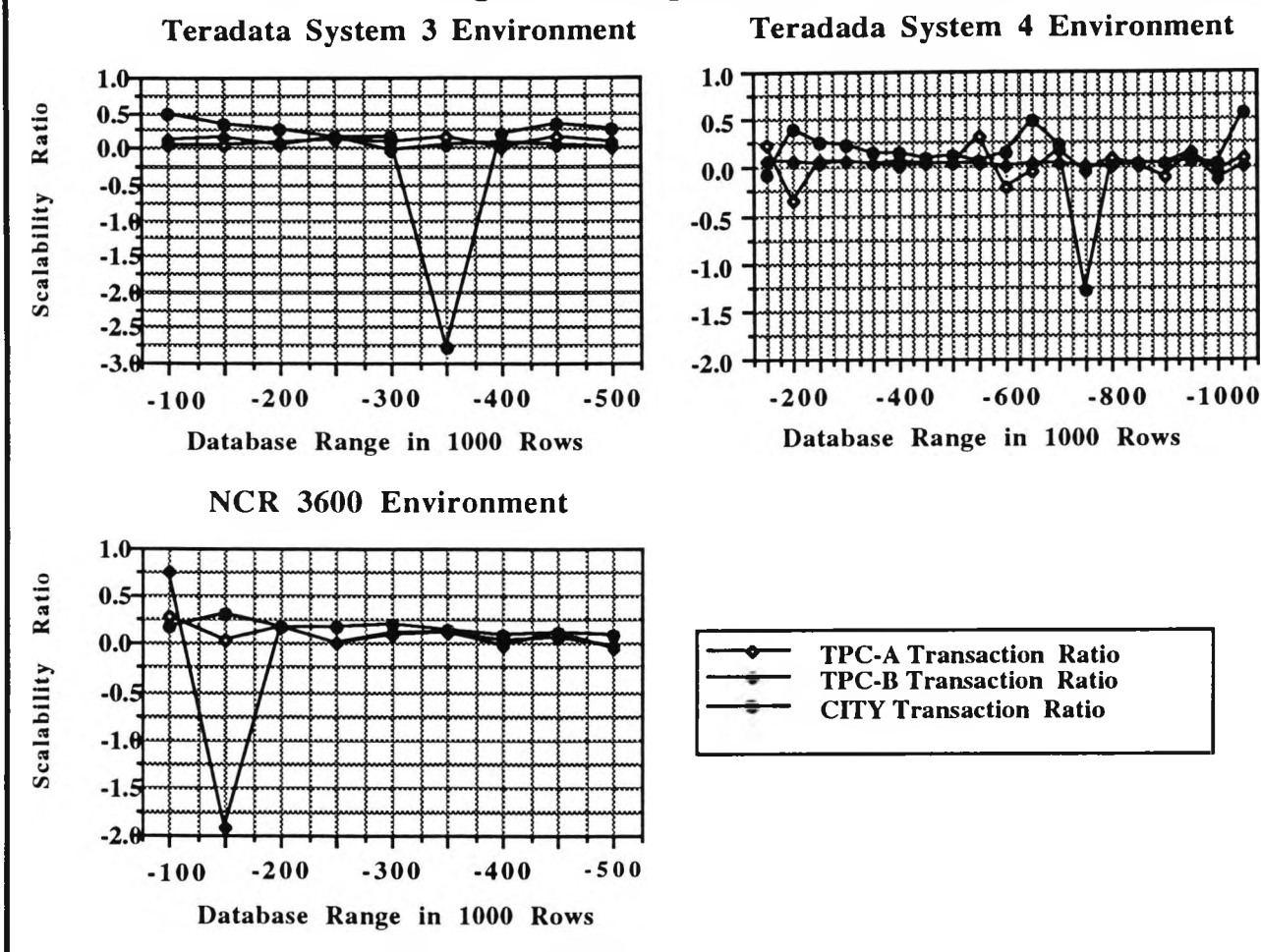


Fig. 8.4, Scalability ratio of The CITY and TPC-A, TPC-B in multiprocessors environments

8.2.3 Comparability of Transactions' Measures Between DBMS

One of the main limitations of the TPC benchmarks is having a simple transaction mix that consists of simple database operations. That mix is three update operations and one row insertion. The TPC-B benchmark transaction consists of three update operations, one row insert and a qualified retrieval. Those transactions will severely penalise those systems with poor update or insert operations performance, ignoring that they might have an overall good level of performance.

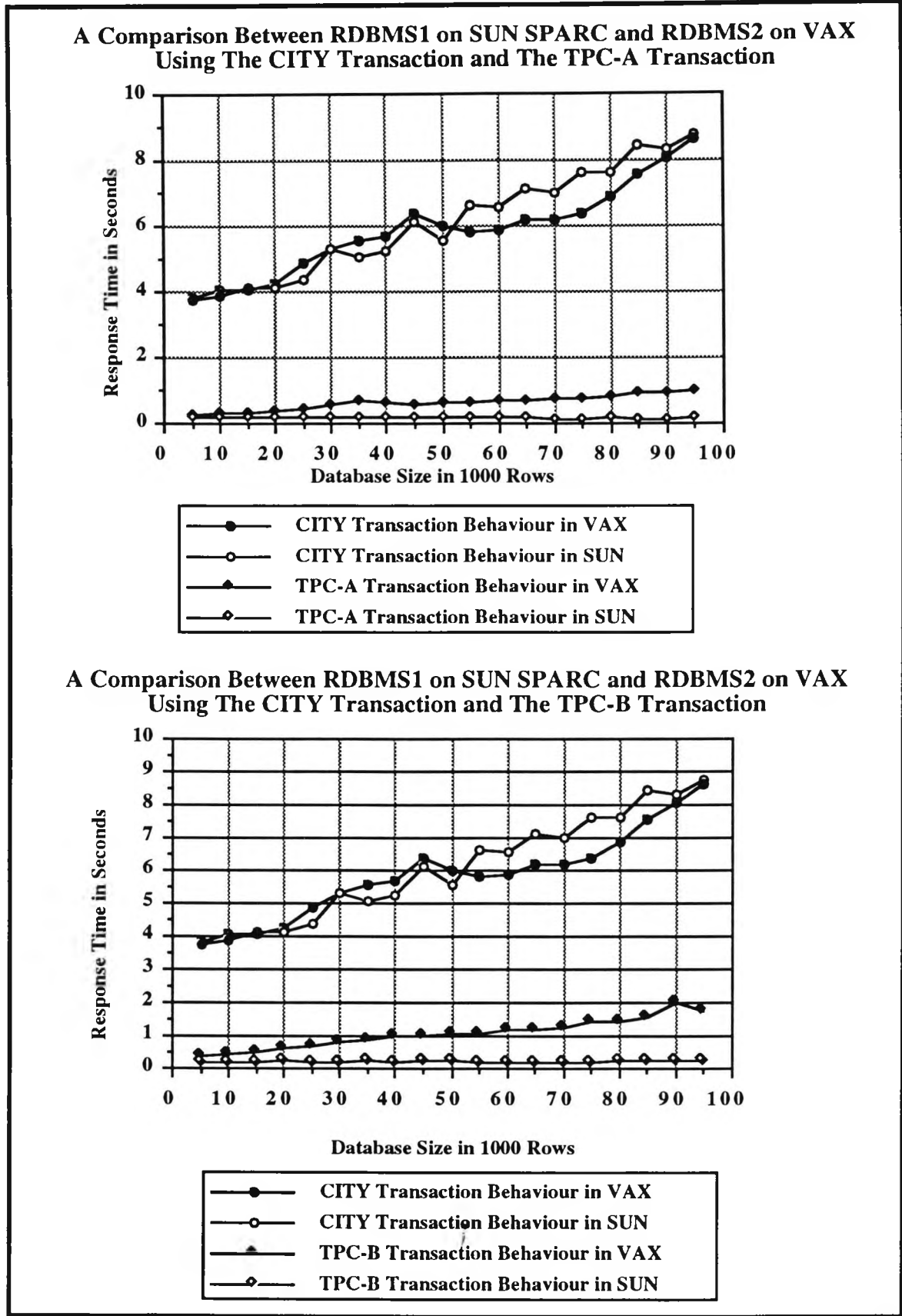


Fig. 8.5, Comparing DBMS using both The CITY and The TPC-A and TPC-B

To test that assumption, the CITY transaction behaviour was compared to both the TPC-A and the TPC-B transactions' behaviour in two different environments running two different database management systems. The result of the comparison is presented in Fig. 8.5. The database management system running on the SUN SPARC produced better performance than the database management system running on the VAX using both the TPC-A and the TPC-B transaction scripts. The DBMS running on VAX has produced better performance pattern using the CITY transaction specially at larger tables' sizes.

This result could be because the database management system running on the SUN SPARC performs update operation better than the database management system running on the VAX computer. The CITY transaction, which is a balanced transaction mix of database operations, favoured the database management system running on the VAX computer. This could be due to the balanced nature of that database management systems and the way it performs its operation.

8.2.4 Compliance of The TPC Benchmark with Future Trends of DBMS

Database market industry is moving towards multi-media, object orientation and decision support systems. Rotzell and Loomis [ROTZ91, LOOM92], have discussed some of those directions and their application. Those environments, if not characterised by their large database sizes they definitely characterised by their large row size. An example can be given by the new Microsoft product, Microsoft Access. That product gives the database user a pictorial interface where he/she could see the tables' layout, system design, key connection and foreign keys' connections. That system is able to store pictures that were either drawn or scanned as a field in the row. Users could eventually access that field through table index. That stored picture takes at least 100k.

In Fig. 8.6 an illustration of the effect of increasing row size on transaction response time is presented. In that study the TPC-A and the TPC-B and the CITY transactions' response times were compared with the CITY benchmark transaction response time after increasing average row size from 200 bytes to 2000 bytes, 4000 bytes and 6000 bytes consecutively. Row size effect was studied in three independent experiments, each one of which used one of the tested rows' sizes. After the completion of each test, the tables were deleted and database management was terminated. Tables' sizes for the experiments are presented in Table 8.1.

Table Name	Test 1 Row Size	Test 2 Row Size	Test 3 Row Size
DB100	1000 Bytes	2000 Bytes	4000 Bytes
DB200	2000 Bytes	4000 Bytes	6000 Bytes
DB300	3000 Bytes	6000 Bytes	8000 Bytes
DBUPD	2000 Bytes	4000 Bytes	6000 Bytes

Table 8.1, Sizes of rows in row test of row size effect

The experiments showed that the TPC benchmarks, due to its low scalability, will not be able to represent environments that are using any of the three row sizes examined by those experiments. The CITY transaction was still sufficiently representative when using 2000 bytes' tables, but when row size was increased to 4000 bytes and 6000 bytes, response time started to take wider diversion from the CITY transaction response time. Due to the CITY transaction high scalability level, the CITY transaction script could be used in the future to primarily test object oriented database (OODB) environments by replacing the 200 bytes row size with larger row size to test those environments.

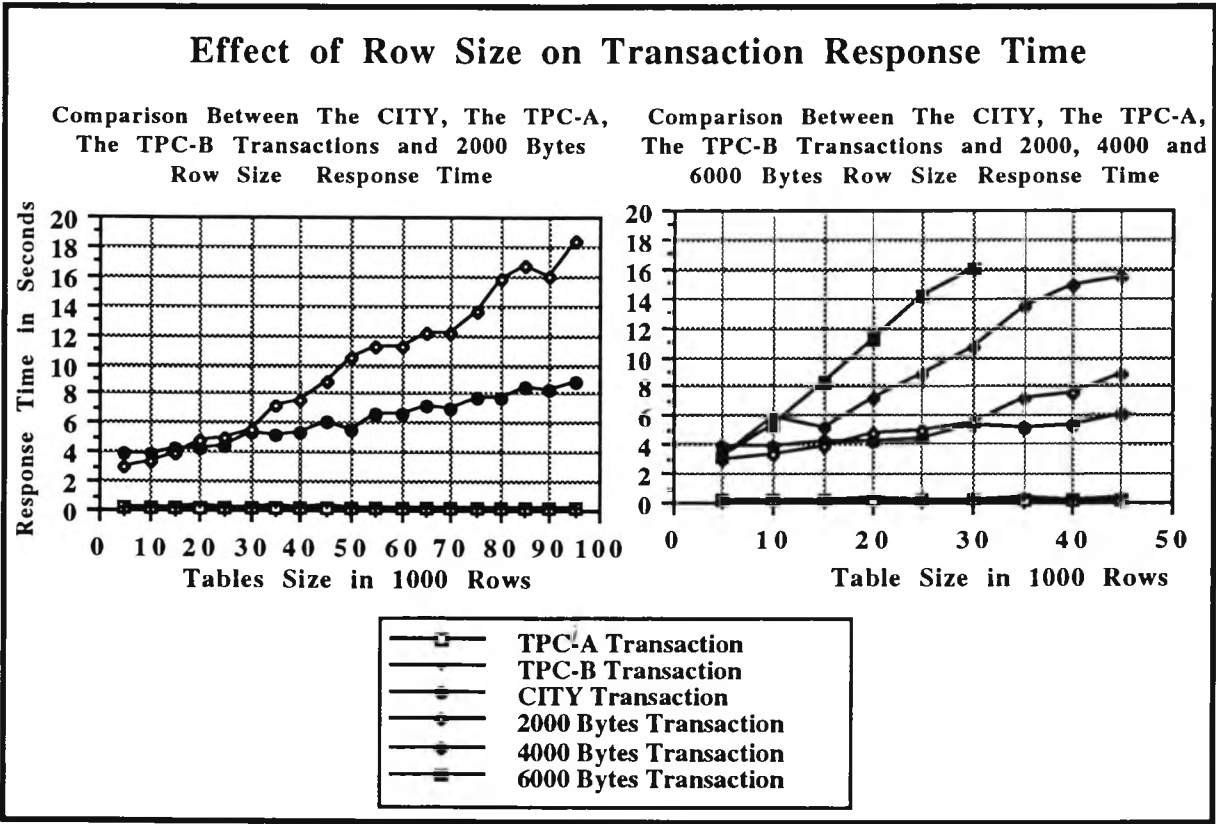


Fig. 8.6, Effect of row size on transaction response time (future trends)
Transaction Behaviour in Large Database Environments, A Methodological Approach. 248

8.3 Comparison between the CITY Benchmark and the TPC-C Benchmark

In recognition of the technical limitations of the TPC-A and the TPC-B benchmarks, the transaction processing council published a third benchmark called the TPC-C. The draft of that benchmark was published in December 1991. The following sections present a comparison between the TPC-C benchmark and the CITY benchmark. The comparison covered the two areas.

- Transaction cost in terms of response time.
- Database size.

8.3.1 Comparison between the Cost of the CITY Transaction and the Cost of the TPC-C Benchmark Transaction

Since publication of the TPC-C benchmark in December 1991 [TPC 91], it is yet to be used. This could be referred to the exceptionally high cost of the TPC-C transaction in terms of response time. This thesis tried to demonstrate this issue by calculating the cost of the full script of the TPC-C benchmark and the cost of the on-line programs of the TPC-C benchmark. The studies took place in a SUN SPARC environment, an environment similar to that used to test the CITY transaction cost.

The first study tested the full script of the TPC-C using tables varying in size from 1000 rows to 20,000 rows increasing by 1000 rows at a step. Table 8.2 presents the results of the study and Fig. 8.7 illustrates the behaviour of the full script of the TPC-C benchmark. The results of the study showed that the cost of the full script of the TPC-C benchmark was over 11 times more expensive than the CITY benchmark cost when the test used tables of 20,000 rows.

When a curve fit using ordinary least squares method to the values presented in Table 8.2, the equation was the following:

$$y = 0.75789 + 2.73x \quad (1)$$

with Correlation Coefficient (R^2) = 0.984

The CITY benchmark transaction takes around 9 seconds using tables of 100,000 rows, if the previous equation (1) is used to forecast the response time of the TPC-C benchmark using tables of 100,000 rows it will be around 76 hours.

Size (1000 rows)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
CITY Response					3.7					3.8					4.1					4.8
TPC-C Response	7	8	9	11	13	15	19	23	24	24	31	33	38	36	41	43	51	52	53	55

Table 8.2, Comparison between the CITY measures and full TPC-C measures

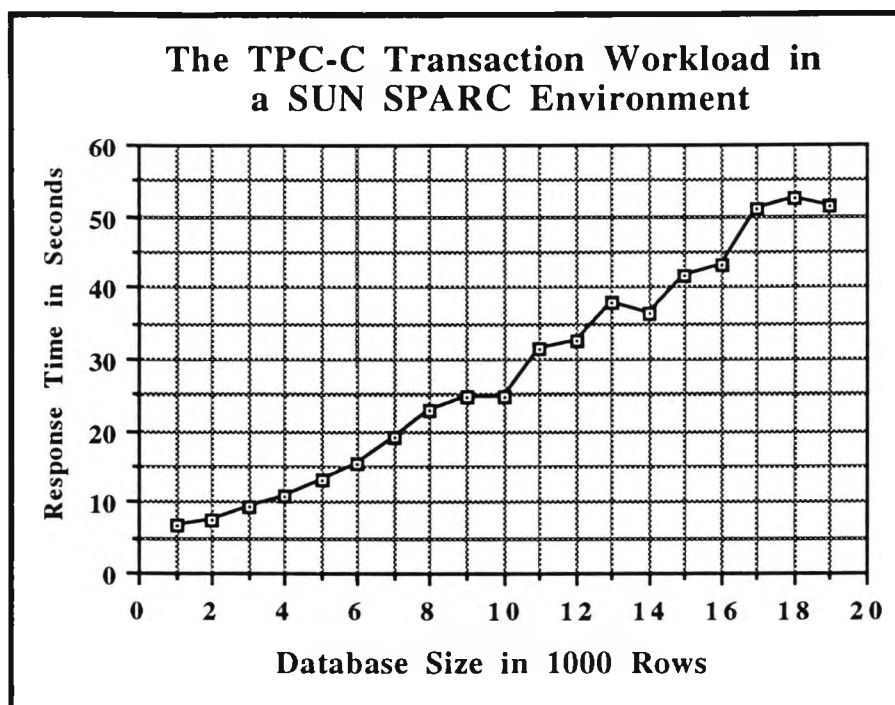


Fig. 8.7, The TPC-C complete transaction workload in SUN SPARC environment

The second study tested the script of the on-line programs of the TPC-C using tables varying in size from 1000 rows to 12,000 rows increasing by 1000 rows at a step. These sizes were constrained by the space available on the tested system. Due to several software and hardware problems at a time the study could not use more than 12,000 rows' tables. Table 8.3 presents the results and Fig. 8.8 illustrates the behaviour of the on-line script of the TPC-C benchmark. The studies showed that the slope of the TPC-C transactions is steep, and while the response time is not high for the smaller sizes of the test tables it increased sharply with larger sizes of the test tables.

The results in Table 8.3 shows that while the CITY response time and the on-line programs of the TPC-C response time were almost equal at 5000 table size, the response time of the TPC-C on-line programs at 10,000 rows was more expensive than that of the CITY benchmark at 20,000 rows. When a curve fit using ordinary least squares method to the values presented in Table 8.3, the equation was the following:

$$y = 1.9939 + 0.39484x \qquad (2)$$

with Correlation Coefficient (R^2) = 0.89

When the CITY benchmark transaction response time using tables of 100,000 rows is compared to the response time of the on-line programs of the TPC-C calculated from the previous equation (2), we find that while the CITY transaction response time was 9 seconds the calculated response time of the TPC-C transaction was around 11 hours.

Size in 1000 rows	1	2	3	4	5	6	7	8	9	10	11	12
CITY Response					3.7					3.8		
TPC-C Response	3.1	3.1	3.1	3.1	3.5	3.9	4.3	5.8	5.2	5.7	6.3	7.5

Table 8.3, Comparison between The CITY measures and on-line TPC-C measures

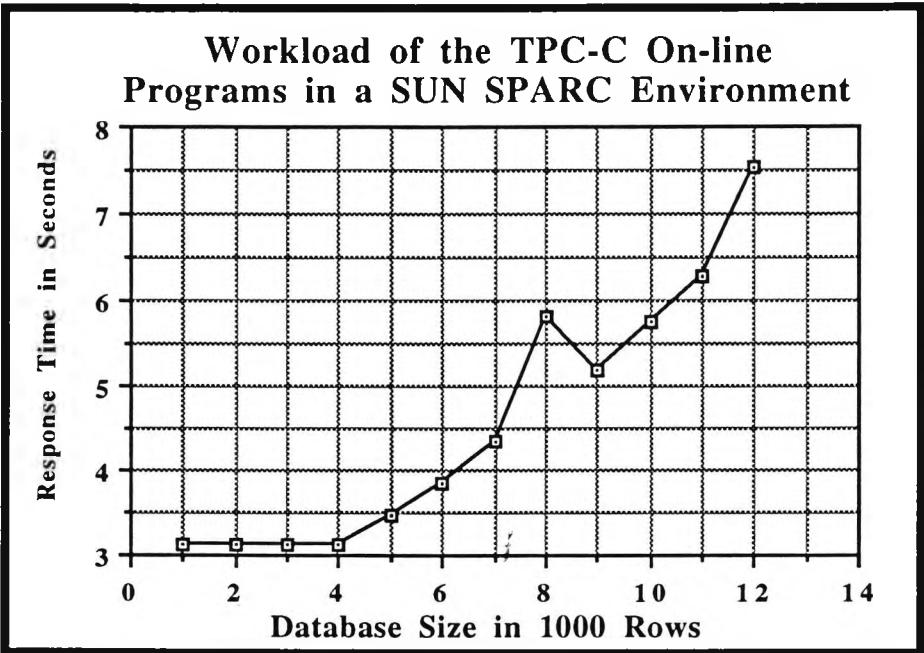


Fig. 8.8, The TPC-C on-line transaction workload in SUN SPARC environment

8.3.2 Database Size of the TPC-C Benchmark

The latest TPC-C benchmark publications [TPC 91], specifies the TPC-C database size to be over 40,000,000 rows where the total sizes of rows is around 1500 bytes. During the large scale verification of the CITY in a large mainframe environment (the test used an IBM 3090), loading 1,000,000 rows took over four hours. Similarly, in the Teradata environment loading 1,000,000 rows took also around six hours. The current author expect that load 40,000,000 rows database will would take more that 160 hours (around seven days) in the mainframe environment and over 250 hours (around eleven days) in the Teradata environment.

8.4 Utilisation of The CITY Benchmark

Since the first publication of the CITY benchmark [REVE92a], it has been well accepted by several users in the database industry. Several database users found in the CITY benchmark a reliable tool that gives representative indications regarding their DBMS performance.

The NCR computer company has consented publishing the performance results of its new multiprocessor computer, NCR 3600, in this thesis and in any other publications discussing database performance. The Teradata branch has permitted the publication of their systems, Teradata System3 and Teradata System4 as performance indicators of those database machines. The letter of consent is presented in appendix H.

In their process of re-engineering their database environment, a local authorities computer centre has been using the CITY benchmark to produce a performance index that can be used as a factor in the database re-engineering process. That was possible using the CITY benchmark because it can run on any range of computers ranging from standalone PCs to large mainframes. That process is presented and discussed by Dobson in [DOBS93].

The CITY benchmark is going to be used as the performance index for the new TOPSY database machine. The TOPSI is a database machine under construction in the CITY university. A large group of researchers in association with the INGRES database company are working in that project. The project is approaching its final stages and the CITY benchmark will be used as the TOPSY database machine performance index.

During the large scale verification of the CITY benchmark in a large airlines company, the benchmark was used to detect a bottleneck in the system. The bottle neck existed due to the selection of a wrong parameter that caused system delay during insert operation. The parameter was corrected and the system was tested one more time using the CITY benchmark to check the disappearance of that system bottleneck. The was possible due to the existence of a monitor that showed the execution steps of the benchmark transaction.

Finally, Abbey National, have decided to use the CITY benchmark to undertake a commercial and technical evaluation for selecting a new product to replace the existing software and DBMS. They are interested in examining three domains: OLTP; Batch; and DSS. They are going to use the CITY benchmark as the index for the OLTP environment performance. The letter of acceptance is presented in appendix H.

The previous users selected the CITY benchmark based on the quality of its quantitative aspects, this research could be extended by conducting several field studies to examine the effect qualitative aspects on benchmark selection.

8.5 Summary

This chapter presented a comparison between the CITY benchmark transaction behaviour and the transaction behaviour of TPC benchmarks (A, B and C). The comparison covered five areas: scalability levels; scalability ratios, consistency of the benchmark measures, suitability of the benchmark for future prospects and transaction cost. The TPC benchmarks suffered limitations in all areas. Comparing scalability levels, the CITY benchmark transaction showed great diversion from the TPC benchmarks under all loads and the CITY benchmark produced better scalability ratio in all the tested environments.

In comparing systems, the TPC benchmarks (A and B) favoured database management systems that perform better update operations. When two database management systems, one that favours update operations and another that performs database operations in a balanced way, were tested using both the CITY benchmark and the TPC benchmark the TPC benchmarks favoured the first over the second and the CITY benchmark favoured the second over the first.

A test of the cost of the TPC-C transaction represented as transaction response time found that the full script of the TPC-C would take around 75 hours to complete one loop using table size of 100,000 rows. If the published database size (40,000,000 rows) is used the full script of the TPC-C would take around 1300 days to finish.

The study separated and tested the cost of the on-line programs of the TPC-C. When a curve fit using ordinary least square method, it was calculated that the on-line programs of the TPC-C would take 11 hours to complete one loop using table size of 100,000 rows. If the response time is extrapolated to the published database size which is 40,000,000 rows the on-line programs will take around 180 days to complete one loop.

By the end of this chapter it can be concluded that the CITY benchmark has proved a reliable tool in evaluating database performance. The Transaction mix has proved superiority over a range of transaction mixes ranging from simple inexpensive transactions to expensive transactions.

CHAPTER 9

CONCLUSION AND FUTURE WORK

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 Conclusion

For many years two benchmarks dominated the database industry, the first was the Wisconsin benchmark [DeWI85] and the second was the TP/1 [ANON85]. The Wisconsin benchmark has several technical limitations in the database design, row length, table size, types of attributes and transaction mix. Gradually, the Wisconsin benchmark lost ground to the TP/1 that became the database industry standard practice where every database vendor has to market its product in number of TP/1s to be able to sell that product. There has been awareness in the database industry that the TP/1 has several limitations due to the over simplification of its transaction mix and lack of standard guidelines for its implementation. In reliance to those problems the TPC council was form to produce a standard benchmark for the database industry. The TPC produced the TPC-A [TPC 89] that is a standard form of the TP/1 and inherited all of its problems. To correct this problem they issued the TPC-B [TPC 90] which is based on the TPC-A transaction mix in addition to one qualified retrieval. Due to the limitations of both TPC-A and TPC-B the TPC council issued another benchmark, the TPC-C [TPC 91]. The TPC-C implementation rules are not clear and its transaction mix is too complex that since it was launched in 1991 it has not yet been used.

This thesis presented the CITY benchmark. Due to the technical limitations of the existing database benchmarks, the main motivation of this research is to produce a database benchmark that represent the OLTP environments. As discussed before, the TPC council tried to produce several benchmarks such as the TPC-A, the TPC-B and the TPC-C to perform this role. But due to either over simplification of transaction mix in the cases of the TPC-A and TPC-B benchmarks, or due to over complication of the transaction mix in the cases the TPC-C benchmark, the TPC benchmarks have failed to produce performance results that accurately represent transaction behaviour in the high volume transaction environments.

The thesis discussed the most widely used benchmarks in general putting more emphasis on the TPC benchmarks (A, B and C) in particular. It presented and discussed in detail the description of the three benchmarks and the critique directed by several researchers to each one of them explaining why the TPC benchmarks have failed to play their planned role. It also presented an empirical approach to examine the workload of the TPC benchmarks that discovered several technical limitations in there scripts. Revell and Youssef [REVE90, REVE92a, REVE92b, REVE93], traced those limitations to the lack of background study before the benchmark design stage. Those limitations confirmed the critiques directed to the TPC benchmarks.

The thesis presented an investigation of on-line transactions in large database environments. The tested environments were three of the largest organisations in using computer resources and having the largest database environments in the UK, those organisations were different in objectives and activities. Those environments were selected because they represented a good example of high-volume transaction environments due to their large on-line databases and their exceptionally heavy transaction load. That investigation aimed to identify on-line transaction behaviour and define the salient characteristics of databases in high-volume transaction environments. The findings from those studies can establish the basis of a transaction and set of tables that are representative to the studied environments.

The results of the studies have identified the typical behaviour of on-line transactions and defined the main characteristics of large databases in high-volume transaction environments. Those findings have disagreed with the patterns of workloads of the TPC benchmarks A and B. The findings showed that the transaction mix of the TPC benchmarks do not provide a model of the OLTP systems due to the wide difference between typical transaction behaviour in large database environments and the transaction mix of the TPC benchmarks transactions. Moreover, they do not even provide a model of the ATM systems whose performance they are supposed to simulate [REVE92b].

The CITY benchmark design is directly driven from the findings from the empirical studies. The benchmark design took into consideration all the critiques directed towards the TPC benchmarks A, B and C. It is the first benchmark that is designed as a result of studying the behaviour of on-line transactions and databases in large database environments. The CITY benchmark is mainly designed to test and compare database systems performance in high-volume transaction environments (OLTP). The benchmark can play a role in diagnosing systems and detecting bottlenecks provided that the existence of supporting software that can monitor the

execution on different database operations. The CITY benchmark can serve the following objectives.

1. To evaluate the performance of a database management system.
2. To compare performance behaviour of different database management systems.
3. To evaluate the impact of system modifications.
4. To evaluate the effect of system parameters.
5. It can play a role in diagnosing systems and detecting bottlenecks. This is restricted to cases where supporting software exists.

The CITY benchmark verification process has demonstrated that the benchmark is a comprehensive methodology that can evaluate the performance of different DBMS in different architectures under increasing and realistic work load. The CITY benchmark verification process of the CITY benchmark is presented by Revell and Youssef in [REVE94a]. The benchmark verification process showed that the CITY benchmark is characterised by the following.

1. Relevance: the benchmark script is relevant to the OLTP domain because it is derived from in-depth field studies in large OLTP environments.
2. Portability: it has been easily implemented on different systems and architecture ranging from standalone PC environments to large mainframes and multiprocessors computers.
3. Scalability: it is able to strain small and large computer systems.
4. Reproducibility: its are reproducible in large and small environments.
5. Simplicity: it is understandable and well documented.

9.2 Contribution of The Work

This research aimed to achieve several objectives. Those objectives can be summarised in the following:

- test the limitations of the TPC benchmarks;
- define the salient characteristics of large databases and identify typical transactions' behaviour in large database environments;
- build a database evaluation methodology that represent high-volume transactions environments.

This research has fulfilled all of the previous objectives and accomplished several achievements in the database area for the first time [REVE94b]. These achievements are the following:

- This work introduces for the first time the approach of studying systems in action to study the main characteristics of large databases and understand typical transaction behaviour in large database environments. Despite being the most accurate method to get a comprehensive picture of transactions' behaviour, this approach has never been employed before due to the effort, time and cost involved in it.
- The work revealed for the first time the salient characteristics of large database environments and clearly identified a typical behaviour of on-line transaction in OLTP environments.
- This research has clearly shown that the TPC benchmarks are not representative to the domain of high-volume transactions environments (OLTP) and it explained why they could be misleading if used to test database management systems in this domain.
- This thesis presents the first database performance evaluation methodology based on in-depth studies in large database environments. The CITY benchmark was constructed to overcome the limitations of the TPC benchmarks. The benchmark is based on real on-line database transaction operations and takes into consideration all the critiques directed towards the TPC benchmarks. The benchmark design is based on in-depth studies in large database environments and simulates a typical on-line transaction operation at high-volume transactions' environments (OLTP). The CITY benchmark is characterised by being relevant, portable, scalable and simple.

9.3 Limitations of The CITY Benchmark

In chapter two (§2.6) several limitations of the existing benchmarks were discussed. This research tried to overcome those limitations, it managed to solve some and some others were difficult to overcome. May be future work will be able to handle those limitations. The following sections discuss those limitations showing which limitations have been treated and why others were so difficult to overcome.

1. The benchmarks are data model dependent, (e.g., tree dependent or network dependent or relational dependent).

The CITY benchmark is relational model dependent. That is because all new DBMS are based on the relational model.

2. The benchmarks assumed uniform demand for requested records.

The CITY benchmark solved this problem by using a purely random number generator that generates non-uniform distribution of the requested transactions.

3. The benchmarks were tested in a single user mode.

However the CITY benchmark is designed to test multi-user environments, the benchmark test and verification was applied in a single user mode because it was the only way to gain a complete control on the benchmark results. Accordingly, studying concurrency and the effect of increasing the number of database users on the database performance was not studied.

4. The data models in the benchmark experiments did not relay on studying real life database systems.

The CITY data model reflects real life database data models as it relays on studying over 5000 on-line databases in large database environments.

5. The benchmarks database size is too small to be realistic.

The CITY databases size could be increased to virtually infinite size when using response time approach. If the Transactions Per Second (TPS) approach is used the basic database size consists of 4,000,000 rows that increases to 5,000,000 rows by the end of the benchmark run. This kind of load was found to be quit sufficient from this research empirical studies.

6. The benchmarks use only simple transaction types.

The CITY benchmark transaction models the exact behaviour of typical on-line transaction in large database environments as it is based on examining over 4800 applications and 40,000,000 transactions.

7. Transaction results and how they are treated is not clear. Transaction results can be directed to one of three things: the main memory, a log file, and computer terminal, in each case response time or system throughput will be widely different.

The CITY transaction output is directed to a user terminal similar to real life situations.

9.4 Future Work

Benchmarks have several roles to play, may be the most contradicting roles are evaluation and diagnostics. Database evaluation requires a standard benchmark that can run without change from one system to another. Once the user is allowed to modify this kind of benchmark, they lose their standard nature and their results become incomparable between systems. This problem has occurred with the TP/1 where users have implemented the benchmark in different forms and resulted losing its credibility and eventually its was discarded as the market standard. On the other hand benchmarks that are used for diagnosing DBMS should be allowed to be re-configured according to the user requirements. Because the CITY benchmark is mainly designed for evaluating and comparing DBMS, the benchmark users are not allowed to re-configure it. Future work could build a second phase of the benchmark that is re-configurable according to the benchmark user requirements and the benchmark user should have the ability to call and run whichever version of benchmark based on those requirements.

Although the CITY benchmark is written in C programming language which is a high level programming language, available in most environments and known to many users, it is still difficult to understand and to modify by ordinary users. This work can be extended by building a good user interface for the CITY benchmark. That interface would allow the benchmark users to run the program and apply their changes without the need to understand the C language. It will also produce the benchmark results interactively on the users screen associated with all the required graphs.

Two important other areas of future work are decision support systems (DSS) environment and object oriented database (OODB) environment. The approach could be by defining the main characteristics that characterise the real life applications in those environments. Chapters 3 and 4, discussed several guidelines to perform this stage. Then the main factors characterising those environments could be mapped to a model that represents the actual behaviour in those environments.

The main problem with DSS systems is a problem of definition. Database users have a vague idea about the main characteristics DSS transaction and their typical behaviour. Several benchmark designers such as DeWitt [DeWI85] and O'Neil [GRAY91], claimed that they have developed the perfect benchmark to test DSS

environments, but most of that work has an intuitive basis. A proper definition of the salient characteristics of DSS environments is required before having a benchmark that is representative of those environments. A field study, an approach similar to what has been done in this research, could lead to revealing the exact properties of the DSS systems. Until that definition is completed, any work will fall short of realistic test of those systems.

10. REFERENCES

10. REFERENCES

- [AGRA87] Agrawal, R., Carey, M. and Livny, M. "Concurrency Control Performance Modeling Alternatives and Implications", ACM Transaction on Database Systems, Vol 12, No. 1, 1987, pp. 609-654.
- [AHN 86] Ahn, I., et al., "Performance Evaluation of a Temporal Database Management System", SIGMod ACM, 1986.
- [ALAN80] Alanko, T., et. al., "Methodology and Results of Program Behaviour Measurements", ACM Sigmetrus, 7th IFIP W.G., 1980.
- [ALAN84] Alanko, T., et. al., "Virtual Memory Behaviour of Some Sorting Algorithms", IEEE Transactions on Software Engineering, Vol. 10 No. 4, July 1984.
- [ALLE89] Allen, A. "Benchmarking for Beginners", EDP Performance Review, January 1989.
- [AMER86] American National Standard "Database Language SQL", Report No. ZX3.135, 1986.
- [AMER87] American National Standard "Database Language SQL 2", Report No. X3H2, 1987.
- [ANDE89] Anderson, et al., "The Tektronix HyperModel Benchmark Specification", Technical Report No. 89 05, Tektronix, August 3, 1989.
- [ANON85] Anon et al., "A Measure of Transaction Power", Datamation, April 1, 1985.
- [ASTR80] Astrahan, M., Schkolnick, M. and Kim W. "Performance of System R Access Path Selection Mechanism", Proceedings IFIP Conference, 1980.
- [BARA89] Barakati, N. "Turbo C Bible", Howard W. Sams & Company, Indianapolis, Indiana, USA, 1989.
- [BELL84] Bell, D. A. "Database Performance", Program Infotech State of the Art Report, 1984.
- [BELL92a] Bell, D. A. and Grimson J. "Distributed Database Systems", Addison Wesley, 1992.
- [BELL92b] Bell, D. A., et al., "A flexible Parallel Database System Using Transputers", IEEE International Conference on Computer Systems and Software Engineering, The Hague, May 1992.
- [BENB87] Benbasat, I., et al., "The Case Research Strategy in Studies of Information Systems", MIS Q. 11, March 1987, 369-386.
- [BENI84] Benigni, R., Yao, B., and Hevner, A. "A Guide to Performance Evaluation of Database Systems", Report of U.S. Department of Commerce. National Bureau of Standards. May 1984. NBS Special Publication 500-118.

- [BENI85] Benigni, R., Yao, B., and Hevner, A. "Benchmark Analysis of Database Architecture: A Case Study", Report of U.S. Department of Commerce. National Bureau of Standards. Oct 1985. NBS Special Publication 500-132.
- [BITT83] Bitton, D., DeWitt D., and Turbyfill, C. "Benchmarking Database Systems: A Systematic Approach", Proc. VLDB Conf., IEEE Press, 1983.
- [BITT86] Bitton D., and Turbyfill, C. "Performance Evaluation of Main Memory Database Systems", Internal Report, Department of Computer Science, Cornell University, 1986.
- [BITT87] Bitton D., Hanrahan M. and Turbyfill, C. "Performance of Complex Queries in Main Memory Database Systems", Proc. International Conf. on Third Data Engineering, Feb. 1987.
- [BOGD83] Bogdanowicz, R. J. "Benchmarking the Selection and Projection Operations and Ordering Capabilities of Relational Database Machines", Naval Postgraduate School, Monterey, California, June, 1983.
- [BORA84] Boral, H., and DeWitt, D. "A Methodology for Database System Performance Evaluation", Proc. SIGMOD Conf., Boston, June 1984.
- [BORO79] Borovits, I. and Neumann, S. "Computer System Performance Evaluation", D.C. Heath and Co., Lexington, Mass., 1979.
- [BOUC76] Bouchard, T. J. "Field Research Methods", In Handb. Ind. Organ. Psychol., Rand-McNally, Chicago, 1976.
- [BUCH82] Buchanan, D.A. and Boddy, D. "Advanced Technology and the Quality of Working Life: The Effect of Word Processing On Video Typists", Journal of Occupational Psychology, 1982 (55, 1-11).
- [BUTL87] Butler, M. "Database Performance", 1987 DEC User Conference, London, UK, 6-7 Oct. 1987. (London UK: EMAP Conferences 1987), p. 113-16.
- [BUZE76] Buzen, J. and DeWitt, D. "Fundamental Operational Laws of Computer System Performance", Acta Informatica 7, 1976.
- [CANI88] Caniano, S. "All TP1s Are Not Created Equal", Datamation (USA), Aug. 15, 1988, pp. 51-53.
- [CARD75] Cardenas, A.F. "Analysis & Performance of Inverted Database Structures", Commun. ACM 18, 5 (May, 1975), pp. 540-548.
- [CARD87] Card, D.N. McGarry, F.E., and Page, G.T., "Evaluating Software Engineering Technologies", IEEE Trans. Softw. Engineering, Vol.13, 5 July 1987, pp. 845-851.
- [CATT90] Cattell, R. et al., "Engineering Database Benchmark", Technical Report, The Sun Microsystems Inc., April 1990.
- [CLAR72] Clark, P. A. "Action Research and Organizational Change", Harper & Row, 1972.
- [Codd70] Codd, E.F. "A Relational Model of Data for Large Shared Data Banks", CACM, June 1970.
- [COHE90] Cohen, B. "Benchmark Silliness", Byte, August 1990.

- [CURT80] **Curtis, B.** "Measurements and Experimentation in Software Engineering", Proc. IEEE, Vol. 68, September 1980, pp. 1144-1157.
- [CURT85] **Curtis, B.** "Human Factors in Software Development", second ed., IEEE Computer Society, Wash. D.C., 1985.
- [CURT86] **Curtis, B., et al.**, "Software Psychology: The need for all Interdisciplinary Program", Proc. IEEE, Vol. 75, August 1986, pp. 1902-1106.
- [CURT88] **Curtis, B., et al.**, "A Field Study of the Software Design Process For large Systems", Comm. ACM, Vol. 31, No. 11, November 1988.
- [DATE86] **Date, C. J.** "An Introduction to Database System", 3th ed. Addison Wesley, 1986.
- [DEEN90] **Deen, M.** "Practical Database Techniques", PITMAN, 1990.
- [DEMU84] **Demurjian, S., et al.**, "Performance Evaluation in Multiple Backend Configuration", Naval Postgraduate School, Monterey, California, Oct. 1984.
- [DEMU85a] **Demurjian, S., and Hsiao, D.** "Benchmarking Database Systems in Multiple Backend Configuration", Database Eng., IEEE Computer Society Press, Vol. 4, 1985.
- [DEMU85b] **Demurjian, S., and Hsiao, D., et al.**, "Performance Evaluation in Multiple Backend Configuration", Database Machines, Fourth International Workshop, 1985.
- [DENN80] **Denning, P. J.** "Working Sets Past and Present". IEEE Transactions on Software Engineering, Vol.6, No. 1, January 1980.
- [DeWI85] **DeWitt, D.** "Benchmarking Database Systems: Past Efforts and Future Directions", Database Eng., IEEE Computer Society Press, Vol.4, 1985.
- [DeWI88] **DeWitt, D., Ghandeharizadeh, S., and Schbeider, D.** "A Performance Analysis of the Gamma Database Machine", SIGMOD RECORD, Vol. 17, No. 3, Sep. 1988, pp. 350- 360.
- [DOBS93] **Dobson, R.**, "Downsizing Applications - A Method of Evaluation", Msc Thesis, City University, DBC, 1993.
- [DONG87] **Dongarra, J., et al.**, "Computer Benchmarking: Paths and Pitfalls", IEEE Spectrum., 24:7, July 1987.
- [[FEDO87] **Fedorowicz, J.** "Database Performance Evaluation in Indexed File Environment", ACM Transaction on Database Systems, Vol12, March, 1987.
- [FENT86] **Fenton, G.** "A Computer Aided Design For the Generation of Test Transactions and Test Database For the Benchmarking of Parallel, Multiple-Backend Systems", Naval Postgraduate School, Monterey, California, June, 1986.
- [FERR78] **Ferreri, D.** "Computer System Performance Evaluation", Printice-Hall Inc., Englewood, Cliffs, New Jersey, 1978.

- [FERR83] Ferreri, D., Serazzi, G., and Zeigner, A. "Measurement and Tuning of Computer Systems", Printice-Hall Inc., Englewood, Cliffs, New Jersey, 1983.
- [FOX 89] Fox, R. "Benchmarking or Benchmarketing?", Computer Weekly, April 20, 1989, page 36.
- [GIBS70] Gibson, J. C. "The Cibson Mix", IBM report TR-00 2043, June 18, 1970.
- [GLES81] Gleser, M., Bayard, J., and Lang, D. "Benchmarking For the Best", Datamation (USA), pp. 127-136, May 1981.
- [GRAY87a] Gray, J. "A View of Database System Performance Measures", Perf. Eva. Rev. (USA), Vol. 15, Part 1, 1987, pp. 3-4.
- [GRAY87b] Gray, J., Putzolu, F. "The 5 Minute Rule", Proc. ACM SIGMOD Conf., San Francisco, California, May, 1987.
- [GRAY91] Gray, J. "The Benchmark Handbook", Morgan Kaufmann, San Mateo, California, 1991.
- [GREH88] Grehan, R., et al., "Introducing the New Byte Benchmark", Byte, June 1988.
- [HAMM83] Hammersley, M. and Atkinson, P. "Ethnography Principles in Practice", Tavistock Publications, 1983.
- [HAWN87] Hawng, H. and Yu, Y. "An Analytical Method for Estimating and Interpreting Query Time", Proc. of 13th VLDB Conference, Brighton, 1987, 347-357.
- [HAWT79] Hawthorn, P. and Stonebraker, M. "Performance Analysis in Relational Database Systems", Proc. SIGMOD, New York, 1979.
- [HAWT82] Hawthorn, P. and DeWitt, D. "Performance Analysis of Alternative Database Machine Architectures", IEEE Trans. on Software Eng., Vol. SE-8, No. 1, Jan. 1982.
- [HAWT85] Hawthorn, P. "Variations on A Benchmark", Database Eng., IEEE Computer Society Press, Vol. 4, 1985.
- [HEID84] Heidelberg, P. and Lavenberg, S. "Computer Performance Evaluation Methodology", IEEE Transactions on Computers, Vol. 33 No. 12, 1984.
- [HILL86] Hillyer, B., Shaw, D., and Nigam, A. "NON- VON's Performance on Certain Database Benchmarks", IEEE Trans. on Software Eng., Vol. SE-12, No. 4, April 1986.
- [HOEL82] Hoel, P. "Basic Statistics for Business and Economics", John Wiley & Sons, Third Edition, 1982.
- [HOFF87] Hoffman, L. L. "Benchmarking Blues: The Reincarnation of Diogenes", Computer Science and Statistics: Proceedings of the 19th Symposium on the Interface, Philadelphia, PA, USA, 8-11 March 1987.
- [HULL88] Hull, M., Cai, F. and Bell, D. A. "Buffer Management Algorithms for Relational Database Management Systems", Information and Software Technology, 1988.

- [KANT90] Kantaris, N. "Learning To Program in C", Bernard Babani, 1990.
- [KELB87] Kelbe, F., and Majors, D. "Benchmarking Preparation For AND AGGREGATE AND SORTING Retrievals in the Multi-backend Database System", Naval Postgraduate School, Monterey, California, June, 1987.
- [LANG88] Langa, F. "Our New Benchmark", Byte, June 1988.
- [LAWR86] Lawrence, M. H. and Quilici, A. E. "Programming in C", John Wiley & Sons, 1986.
- [LEUT93] Leutenegger, S. and Dias, D. "A Modeling Study of the TPC-C Benchmark", Proc. ACM SIGMOD, Vol. 22, No. 2, June 1993, pp 22-29.
- [LOFL71] Lofland, J. "Analysing Social Settings", Wadworth, 1971.
- [LOOM92] Loomis, M. "Object Database Technology: Whos's Using It and Why?", Object Databases, October, 1992, pp. 13-18.
- [MACK86] Mackert, L., Lohman, G. "R* Optimizer Validation and Performance Evaluation For Local Queries", ACM SIGMod, 1986.
- [MART83] Martinez, R., Wiley, R., Bitton, D., and Turbyfill, C. "An Experiment In Benchmarking Database Systems", (Los Alamos National Lab., NM (USA)), 1983.
- [MAST91] Mastars, D. "C An Introduction With Advanced Applications", Prentice Hall International, (UK), 1991.
- [McCA69] McCall, G.J. and Simmons, J. L. "Issues in Participant Observation", Addison-Wesley, 1969.
- [McCA92a] McCann, J. A. and Bell, D. A. "A CPU Resource Consumption Model for Database System", A Technical Report, University of Ulster, 1992.
- [McCA92b] McCann, J. A. and Bell, D. A. "A Flexible Benchmarking Tool for Parallel Database Systems", Proc. on Conf. on benchmark assessment of Parallel Processors, March, 1992.
- [McCA92c] McCann, J. A. "A Fine Grained Database System Performance Model", Phd Thesis, University of Ulster, 1992.
- [McKA78] McKaskill, T. "Effective Computer-Based Information Systems Operations : A Complex Organisational Study", London University, Phd thesis, 1978.
- [MOHR84] Mohr, J. "Projecting Workloads for New Systems: A Management Introduction", J Cap Mgt. Vol. 2, No. 2, 1984.
- [NADE90] Nadeau, M. "BYTE'S New Benchmarks", Byte, August 1990.
- [NIED79] Niedereichholz, J. "Performancetests mit einem CODASYL Datenbanksystem", Institut fur Wirtschaftsinformatik, Universitat Frankfurt, 1979.
- [ORAC91] ORACLE Report "TPC B Full Disclosure", 1991.

- [ORD 88] **Ord, J.** "Introduction of Information technology in an Urban Health District", LBS Phd thesis, 1988.
- [OSTL88] **Ostle, B. and Malone, L.** "Statistics in Research", Iowa State University Press, Ames, Iowa, Fourth Edition, 1988.
- [[REVE88] **Revell, N. and Ruff, M.** "A Meta System for the Design of Database Systems", Proc. of EMCSR pp. 1169-1176 1988.
- [REVE90] **Revell, N. and Youssef, M. W.** "An Analysis of Database Performance Measures", Proc. of EMCSR, Apr. 1990.
- [REVE92a] **Revell, N. and Youssef, M. W.** "An Analysis of Transaction Behaviour in Large IMS Environment." Proc. of EMCSR, Apr. 1992. **"Best Paper Award for the Symposium Communication and Computers"**.
- [REVE92b] **Revell, N. and Youssef, M. W.** "TP1: The Fact and The Myth." Proc. of SEARCC, 1992.
- [REVE93] **Revell, N. and Youssef, M. W.** "Database Performance Evaluation A Methodological Approach" Proc. of DEXA93, September, 1993.
- [REVE94a] **Revell, N. and Youssef, M. W.** "Testing And Verification Of The City Database Benchmark." Proc. of EMCSR, Apr. 1994.
- [REVE94b] **Revell, N. and Youssef, M. W.** "Modelling Transaction Behaviour in Large Database Environments" 7th International Conference on Systems Research, Informatics and Cybernetics, Aug., 1994.
- [REVE94c] **Revell, N. and Youssef, M. W.** "Benchmarking The Benchmarks" a paper under publication in Proc. of DEXA94, September, 1994.
- [ROTZ91] **Rotzell, K. and Loomis, M.** "Benchmarking an ODBMS", JOOP, March-April 1991.
- [RUBE87] **Rubenstein, W.B. and Kubicar, M. S.** "Benchmarking Simple Database Operations", SIGMOD Rec.(USA) vol 16 part 3 1987, pp 387-97.
- [RYDE83] **Ryder, C.** "Benchmarking Relational Database Machines. Capabilities in Supporting The Database Administrators Functions and Responsibilities", Naval Postgraduate School, Monterey, California, Sep., 1983.
- [SANT89] **Santos, A.** "The design of information systems: a new methodological approach". London University, LBS Phd thesis, 1989.
- [SERL86] **Serlin, O.** "MIPS, DHRYSTONES, and Other Tales." Datamation (USA), Vol.32, pp. 112-118, June 1986.
- [SILV70] **Silverman, D.** "The Theory of Organisations", Heinemann, 1970.
- [SIVU90] **Sivula, C.** "The Benchmark War in Transaction Processing", Datamation, September 1, 1990, pp. 52-55
- [STON83a] **Stone, V.** "Design of Relational Database Benchmark", Naval Postgraduate School, Monterey, California, June, 1983.

- [STON83b] **Stonebraker M., et al.**, "Enhancements to a Relational Database System", ACM Transaction on Database Systems, Vol. 8, No. 2, June 1983, pages 167-185.
- [STON85] **Stonebraker, M.** "Tips On Benchmarking Database Systems", Database Eng., IEEE Computer Society Press, Vol.4, 1985.
- [STRA84] **Strawser, P.** "A Methodology for Benchmarking Relational Database Machines", Ph.D Dissertation. Ohio State University, Dec. 1984.
- [SWAN88] **Swansen, E. B.** "The Use of Case Study Data in Software Management Research", J. Syst. Software, Vol. 8, January 1988, 63-71.
- [TAND88] **Tandem Performance Group**, "A Benchmark of NonStop SQL on the Debit/Credit Transaction", SIGMOD RECORD, Vol. 17, No. 3, Sep. 1988, pp. 337-341.
- [TAZE88] **Tazelaar, J. M.** "Benchmarks", Byte, June 1988.
- [TPC 89] **Transaction Processing Performance Council**, "TPC Benchmark A Standard", November 1989.
- [TPC 90] **Transaction Processing Performance Council**, "TPC Benchmark B Standard", 1990.
- [TPC 91] **Transaction Processing Performance Council**, "TPC Benchmark C Standard", December., 1991.
- [TPC 92] **Transaction Processing Performance Council**, TPC Press Background, March, 1992.
- [TURB88] **Turbyfill, C.** "Comparative Benchmarking of Relational Database Systems", Ph.D Dissertation. Cornell University, Jan. 1988.
- [ULLM82] **Ullman, J.** "Principles of Database Systems", 2nd ed. Computer science press, 1982.
- [VAN 82] **Van maanen, J., et al.**, "Varieties of Qualitative Research", Sage Publications, 1982.
- [VINC85] **Vincent J.R.** "A Performance Measurement Methodology for a Multi-backend Database System", Naval Postgraduate School, Monterey, California, June, 1985.
- [WIED83] **Wiederhold, G.** "Database Design", 2nd ed. McGraw-Hill, New York, 1983.
- [WILL90] **Williams, J. M.** "Thanks for the Unix Benchmarks", Byte, July 1990.
- [WICH82] **Wichmann, B.A. and Hill, I.D.**, NPL Report DITC 6/82, 1982.
- [WOOD81] **Woodfill, J., Siegal, P., Rånstrom, J., Meyer, M. and Allman, E.**, INGRES Reference Manual. Version 7 ed. 1981.
- [WONN85] **Wonnacott, R. and Wonnacott, T.** "Introductory Statistics", John Wiley & Sons, Fourth Edition, 1985.

10. References.

- [YAO 84] Yao, B. and Hevner, A. "Performance Evaluation of Database System. A Benchmark Methodology", Report of U.S. Department of Commerce. National Bureau of Standards. May 1984.
- [YAO 87] Yao, B. and Hevner, A. and Yo, H. "Analysis of Database System Architectures Using Benchmarks", IEEE Trans. on Software Eng., Vol. SE-13, No. 6, June 1987.
- [YOUS86a] Youssef, M. W. and Riad, M. "A Software Monitor For Database Performance Evaluation", The Egyptian Computer Journal, Jan 1986.
- [YOUS86b] Youssef, M. W. and Riad, M. and Selim, S. "Data Base Performance: Measures and Empirical Results", The 20th Annual Conference In Statistics, Computer Science and Operations Research, 1986.
- [YOUS86c] Youssef, M. W. "A Software Monitor For Database Performance Evaluation", Msc. Thesis. Cairo University, May. 1986.
- [ZELK84] Zelkowitz, M., et al, "Software Engineering Practices in US and Japan", IEEE Comput., Vol. 17, June 1984, pp. 57-66.

i

APPENDIX A

APPENDIX A: Create The CITY Benchmark Tables

```

/* ----- */
/* THE CITY BENCHMARK TEST. */
/* DROP AND CREATE THE CITY BENCHMARK TABLES. */
/* ----- */

#include <STDIO.H>
#include <CTYPE.H>
#include <STRING.H>
#include <SQLCA.H>
#include <PROCESS.H>
#include <STDLIB.H>
#include <GRAPH.H>
#include <TIME.H>
#define BEEP '\a'
#define MAXPATH 64
#define MAXLINE 80
#define DSECONDS 1
#define NLETTERS 10

/* ----- */
/* DECLARE TPS RECORD STRUCTURE TAG */
/* ----- */

STRUCT TPS_REC_LAYOUT
{ /* STRUCTURE BEGINING */
CHAR DBTYPE[15]; /* DATABASE NAME. */
LONG DBSIZE; /* DATABASE SIZE. */
INT NLOOPS; /* NUMBER OF LOOPS. */
INT LOOPTIME; /* LOOP TIME IN SECONDS. */
INT TOTTIME; /* TOTAL TIME IN SECONDS. */
LONG TOT_TRANS; /* TOT. NUMBER OF TRANSACTIONS. */
LONG AVR_TRANS; /* AVR. TRANSACTIONS PER LOOP. */
DOUBLE TRN_PER_SEC; /* NUMBER OF TRANSACTIONS/SEC. */
DOUBLE IT_RESP_TIME; /* ITERATION RESPONSE TIME. */
TIME T_TIME_DATE; /* TIME AND DATE OF TEST. */
}; /* STRUCTURE END */

/* ----- */
/* DECLARE TPS RECORD STRUCTURE */
/* ----- */

STRUCT TPS_REC_LAYOUT TPS_IO_REC;

/* ----- */
/* THIS PART CREATES FIVE TABLES FOR THE CITY BENCHMARK */
/* DB100, DB200, DB300 DBINS AND DBUPD THEN LOAD THEM. */
/* ----- */

EXEC SQL BEGIN DECLARE SECTION;

/* ----- */
/* THE PASSWORD DECLARATION SECTION */
/* ----- */

VARCHAR UID[20];
VARCHAR PWD[20];

/* ----- */
/* THE DATABASES INTERMEDIATE DECLARATION SECTION */
/* ----- */

```

```

CHAR DBFC1 [8];
INT DBFI;
CHAR DBFC2 [30];
LONG DBFL;
FLOAT DBFR;
DOUBLE DBFD;
VARCHAR DBFV [40];

UNSIGNED INT KEY_VALUE;

/* ----- */
/* THE PROGRAM VARIABLES DECLARATION SECTION. */
/* ----- */

LONG R_1 = 0; /* UNIQUE NO. OF ROWS COUNTER. */
LONG R_10 = 0; /* NO. OF ROWS COUNTER REPEATED EVERY 10 ROWS. */
LONG R_100 = 0; /* NO. OF ROWS COUNTER REPEATED EVERY 100 ROWS. */

INT VI = 50; /* INTEGER VALUE */
LONG VL = 50; /* LONG VALUE */
FLOAT VF = 50; /* FLOAT VALUE */
FLOAT VR = 50; /* FLOAT VALUE */
DOUBLE VD = 50; /* DOUBLE VALUE */

/* CHARACTER FIELDS CHAR1FIELD1, CHAR2FIELD1 AND VARFIELD1 */

STATIC CHAR CHAR1FIELD1 [8] = {"0000001"};
STATIC CHAR CHAR2FIELD1 [30] = {"ABABA ABABA ABABA ABABA ABABA"};
STATIC CHAR VARFIELD1 [40] = {"ABABA ABABA ABABA ABABA ABABA ABA AAAAA"};

EXEC SQL END DECLARE SECTION;

/* ----- */
/* INCLUDE SQLCA COMMAND (SQL COMM AREA). */
/* ----- */

EXEC SQL INCLUDE SQLCA;

/* ----- */
/* THE MAIN PROGRAM START. */
/* ----- */

MAIN()
{ /* PROGRAM START */

FILE *TPS_FP;
LONG RANDP0;

LONG X, Y, Z;

STATIC CHAR PATH_NAME_1[MAXPATH] = {"C:\CITYCRT.TXT"};

/* RANDOM NUMBER GENERATION VARIABLES */
LONG SEED, RT_VALUE, N_ROWS, R_LIMIT10;
INT R_LIMIT = 30000; /* RANDP MAXIMUM VALUE */
UNSIGNED LONG ROWS; /* DATABASE SIZE */

UNSIGNED LONG ROWS_START; /* NUMBER OF ROWS START VALUE */
ROWS_END; /* NUMBER OF ROWS END VALUE */
ROWS_STEP; /* NUMBER OF ROWS STEP VALUE */

```

INT SECONDS;

```
INT I1 = 0; /* LOOP1 INDEX */
I2 = 0; /* LOOP2 INDEX */
I3 = 0; /* LOOP3 INDEX */
```

```
INT I, /* MAIN LOOP INDEX */
J = 7; /* END OF 8 CHAR FIELD POINTER */
K = 29; /* END OF 30 CHAR FIELD POINTER */
L = 0; /* START OF VAR_FIELDS POINTER */
M = 38; /* END OF VAR_FIELDS POINTER */
```

```
UNSIGNED LONG I_LOOPS; /* INDEX OF NUMBER OF RECORDS. */
UNSIGNED LONG ROWS_LOOP = 0; /* DATABASE LOAD LOOPS INDEX */
UNSIGNED LONG START = 0; /* TPS START TIME. */
END = 0; /* TPS END TIME. */
G_TIME1 = 0; /* USED TO GET START VALUE FROM TIME. */
G_TIME2 = 0; /* USED TO GET END VALUE FROM TIME. */
```

```
UNSIGNED LONG TRANS_COUNT = 0; /* LOOP TRANS_COUNTER. */
TOT_TRANS_COUNT = 0; /* TOTAL OF LOOP TRANS_COUNTERS. */
AVR_TRANS_COUNT = 0; /* AVERAGE OF LOOP TRANS_COUNTERS. */
```

```
DOUBLE RESP_TIME; /* AVERAGE TRANSACTIONS RESPONSE TIME. */
RESP_TIME_T; /* TOTAL TRANSACTIONS RESPONSE TIME. */
RESP_TIME_I; /* ONE TRANSACTIONS RESPONSE TIME. */
RESP_PER_STEP; /* RESPONSE TIME PER STEP. */
TRANS_PER_SEC; /* TRANSACTIONS/SEC. */
```

```
TIME_T RESP_TIME_START; /* RESPONSE TIME START. */
RESP_TIME_STOP; /* RESPONSE TIME END. */
```

UNSIGNED LONG RESP_COUNT;

```
STATIC CHAR DB_TYPE[15] = {"ORACLE. "};
```

```
LONG QS=0;
LONG QU1=0;
LONG QU2=0;
LONG QF=0;
LONG IN_REC=0;
```

```
DOUBLE TR_PER_SECOND= 0; /* AVERAGE NUMBER OF TRANSACTIONS/SEC. */
```

```
STATIC CHAR ERROR_MESSAGE[71] =
{" "};
```

```
STATIC CHAR LINE[71] =
{"#####"};
```

```
STATIC CHAR CONN_ERR[71] =
{"AN ERROR OCCURRED DURING CONNECTION TO THE DBMS. "};
```

```
STATIC CHAR F_MSG1[47] = {"....."};
STATIC CHAR F_MSG1[32] = {"TOTAL NUMBER OF INSERTED ROWS: "};
STATIC CHAR F_MSG2[47] = {"THE PROGRAM HAS BEEN TERMINATED SUCCESSFULLY. "};
```

```
STATIC CHAR TABL100_DROP_ERR[71] =
{"AN ERROR OCCURRED DURING TABLE DB100 DROP. "};
STATIC CHAR INDX100_ERR[71] =
{"AN ERROR OCCURRED DURING INDEX DB100_IX CREATION. "};
STATIC CHAR TABL100_CR_ERR[71] =
{"AN ERROR OCCURRED DURING TABLE DB100 CREATION. "};
```

```

STATIC CHAR TABL200_DROP_ERR[71] =
{"AN ERROR OCCURRED DURING TABLE DB200 DROP.           "};
STATIC CHAR INDX200_ERR[71] =
{"AN ERROR OCCURRED DURING INDEX DB200_IX CREATION.      "};
STATIC CHAR TABL200_CR_ERR[71] =
{"AN ERROR OCCURRED DURING TABLE DB200 CREATION.       "};

STATIC CHAR TABL300_DROP_ERR[71] =
{"AN ERROR OCCURRED DURING TABLE DB300 DROP.           "};
STATIC CHAR INDX300_ERR[71] =
{"AN ERROR OCCURRED DURING INDEX DB300_IX CREATION.      "};
STATIC CHAR TABL300_CR_ERR[71] =
{"AN ERROR OCCURRED DURING TABLE DB300 CREATION.       "};

STATIC CHAR TABLINS_DROP_ERR[71] =
{"AN ERROR OCCURRED DURING TABLE DBINS DROP.           "};
STATIC CHAR INDXINS_ERR[71] =
{"AN ERROR OCCURRED DURING INDEX DBINS_IX CREATION.      "};
STATIC CHAR TABLINS_CR_ERR[71] =
{"AN ERROR OCCURRED DURING TABLE DBINS CREATION.       "};

STATIC CHAR TABLUPD_DROP_ERR[71] =
{"AN ERROR OCCURRED DURING TABLE DBUPD DROP.           "};
STATIC CHAR INDXUPD_ERR[71] =
{"AN ERROR OCCURRED DURING INDEX DBUPD_IX CREATION.      "};
STATIC CHAR TABLUPD_CR_ERR[71] =
{"AN ERROR OCCURRED DURING TABLE DBUPD CREATION.       "};

/* ----- */
/* CLEAR SCREEN COMMAND */
/* ----- */

_CLEARSCREEN(_GCLEARSCREEN);

/* ----- */
/* LOG INTO THE DBMS USERID AND PASSWORD. */
/* ----- */

PRINTF("PLEASE ENTER USERID: ");
SCANF("%S",UID.ARR);
PRINTF("PLEASE ENTER PASS WORD: ");
SCANF("%S",PWD.ARR);
UID.LEN = STRLEN(UID.ARR);
PWD.LEN = STRLEN(PWD.ARR);

/* ----- */
/* LOG INTO THE DBMS EVERY TIME THE PROGRAM START A NEW LOOP. */
/* ----- */

STRCPY(ERROR_MESSAGE,CONN_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL CONNECT :UID IDENTIFIED BY :PWD;

PRINTF("*****\n");
PRINTF(" CONNECTED TO THE SYSTEM USER: %S \n",UID.ARR);
PRINTF("***** \n\n");

/* ----- */
/* DROP TABLES SECTION. THIS SECTION ENDS WITH COMMIT DROP TABLES. */
/* ----- */

```

```

/* ----- */
/* DROP TABLE DB100 SECTION. */
/* ----- */

STRCPY(ERROR_MESSAGE,TABL100_DROP_ERR);
EXEC SQL WHENEVER SQLERROR GOTO DROP_200;
EXEC SQL DROP TABLE DB100;
PRINTF(" ----- \n");
PRINTF(" TABLE DB100 HAS BEEN DROPPED. \n");
PRINTF(" ----- \n");

/* ----- */
/* DROP TABLE DB200 SECTION. */
/* ----- */

DROP_200:
STRCPY(ERROR_MESSAGE,TABL200_DROP_ERR);
EXEC SQL WHENEVER SQLERROR GOTO DROP_300;
EXEC SQL DROP TABLE DB200;
PRINTF(" TABLE DB200 HAS BEEN DROPPED. \n");
PRINTF(" ----- \n");

/* ----- */
/* DROP TABLE DB300 SECTION. */
/* ----- */

DROP_300:
STRCPY(ERROR_MESSAGE,TABL300_DROP_ERR);
EXEC SQL WHENEVER SQLERROR GOTO DROP_INS;
EXEC SQL DROP TABLE DB300;
PRINTF(" TABLE DB300 HAS BEEN DROPPED. \n");
PRINTF(" ----- \n");

/* ----- */
/* DROP TABLE DBINS SECTION. */
/* ----- */

DROP_INS:
STRCPY(ERROR_MESSAGE,TABLINS_DROP_ERR);
EXEC SQL WHENEVER SQLERROR GOTO DROP_UPD;
EXEC SQL DROP TABLE DBINS;
PRINTF(" TABLE DBINS HAS BEEN DROPPED. \n");
PRINTF(" ----- \n");

/* ----- */
/* DROP TABLE DBUPD SECTION. */
/* ----- */

DROP_UPD:
STRCPY(ERROR_MESSAGE,TABLUPD_DROP_ERR);
EXEC SQL WHENEVER SQLERROR GOTO COMM_DROP;
EXEC SQL DROP TABLE DBUPD;
PRINTF(" TABLE DBUPD HAS BEEN DROPPED. \n");
PRINTF(" ----- \n\n");

/* ----- */
/* COMMIT DROP SECTION */
/* ----- */

COMM_DROP:
EXEC SQL COMMIT WORK;
PRINTF(" ----- \n");
PRINTF(" DROP TABLES HAVE BEEN COMMITTED SUCCESSFULLY. \n");

```



```

PRINTF("***** \n");

/* ***** */
/* CREATE TABLES SECTION. THIS SECTION ENDS WITH COMMIT CREATE TABLES. */
/* ***** */

/* ***** */
/* CREATE TABLE DB100 SECTION. */
/* ***** */

EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;

STRCPY(ERROR_MESSAGE,TABL100_CR_ERR);
EXEC SQL CREATE TABLE DB100
(DB1FC1 CHAR(8), /* NO-KEY FIELD. */
DB1FI1 NUMBER(8) NOT NULL, /* UNIQUE KEY. */
DB1FC2 CHAR(30), /* NO-KEY FIELD. */
DB1FL1 NUMBER(4), /* NO-KEY FIELD. */
DB1FR1 NUMBER(4,2), /* NO-KEY FIELD. */
DB1FD1 NUMBER(6,3), /* NO-KEY FIELD. */
DB1FV1 CHAR(40)); /* NO-KEY FIELD. */

_CLEARSCREEN(_GCLEARSCREEN);
PRINTF("***** \n");
PRINTF("TABLE DB100 HAS BEEN CREATED. \n");
PRINTF("***** \n\n");

/* ***** */
/* CREATE INDEX DB1FC1_IX. SECTION. */
/* ***** */

STRCPY(ERROR_MESSAGE,INDX100_ERR);

EXEC SQL CREATE UNIQUE INDEX DB1FI1_IX ON DB100(DB1FI1);
PRINTF("***** \n");
PRINTF("UNIQUE INDEX DB1FI1_IX HAS BEEN CREATED. \n");
PRINTF("AN INDEX ON DB100 TABLE FIELD DB1FI1. \n");
PRINTF("***** \n");

/* ***** */
/* CREATE TABLE DB200 SECTION. */
/* ***** */

STRCPY(ERROR_MESSAGE,TABL200_CR_ERR);
EXEC SQL CREATE TABLE DB200
(DB2FC1 CHAR(8), /* NO-KEY FIELD. */
DB2FI1 NUMBER(8) NOT NULL, /* UNIQUE KEY. */
DB2FC2 CHAR(30), /* NO-KEY FIELD. */
DB2FL1 NUMBER(4), /* NO-KEY FIELD. */
DB2FR1 NUMBER(4,2), /* NO-KEY FIELD. */
DB2FD1 NUMBER(6,3), /* NO-KEY FIELD. */
DB2FV1 CHAR(40), /* NO-KEY FIELD. */
DB2FC3 CHAR(8), /* NO-KEY FIELD. */
DB2FI2 NUMBER(8) NOT NULL, /* SECONDARY KEY 10 TIMES. */
DB2FC4 CHAR(30), /* NO-KEY FIELD. */
DB2FL2 NUMBER(4), /* NO-KEY FIELD. */
DB2FR2 NUMBER(4,2), /* NO-KEY FIELD. */
DB2FD2 NUMBER(6,3), /* NO-KEY FIELD. */
DB2FV2 CHAR(40)); /* NO-KEY FIELD. */

```

```

_CLEARSCREEN(_GCLEARSCREEN);
PRINTF("*****\n");
PRINTF("TABLE DB200 HAS BEEN CREATED.\n");
PRINTF("*****\n\n");

/* ***** */
/* CREATE INDEXES FOR DB200 SECTION. */
/* ***** */

STRCPY(ERROR_MESSAGE,INDX200_ERR);

EXEC SQL CREATE UNIQUE INDEX DB2FI1_IX ON DB200(DB2FI1);
PRINTF("*****\n");
PRINTF("UNIQUE INDEX DB2FI1_IX HAS BEEN CREATED.\n");
PRINTF("AN INDEX ON DB200 TABLE FIELD DB2FI1.\n");
PRINTF("*****\n");

/* SECONDARY KEY 10TIMES */
EXEC SQL CREATE INDEX DB2FI2_IX ON DB200(DB2FI2);
PRINTF("SECONDARY INDEX DB2FI2_IX HAS BEEN CREATED.\n");
PRINTF("AN INDEX ON DB200 TABLE FIELD DB2FI2.\n");
PRINTF("*****\n");

/* ***** */
/* CREATE TABLE DB300 SECTION. */
/* ***** */

STRCPY(ERROR_MESSAGE,TABL300_CR_ERR);
EXEC SQL CREATE TABLE DB300
(DB3FC1 CHAR(8), /* NO-KEY FIELD.*/
DB3FI1 NUMBER(8) NOT NULL, /* UNIQUE KEY.*/
DB3FC2 CHAR(30), /* NO-KEY FIELD.*/
DB3FL1 NUMBER(4), /* NO-KEY FIELD.*/
DB3FR1 NUMBER(4,2), /* NO-KEY FIELD.*/
DB3FD1 NUMBER(6,3), /* NO-KEY FIELD.*/
DB3FV1 CHAR(40), /* NO-KEY FIELD.*/
DB3FC3 CHAR(8), /* NO-KEY FIELD.*/
DB3FI2 NUMBER(8) NOT NULL, /* SECONDARY KEY 10 TIMES.*/
DB3FC4 CHAR(30), /* NO-KEY FIELD.*/
DB3FL2 NUMBER(4), /* NO-KEY FIELD.*/
DB3FR2 NUMBER(4,2), /* NO-KEY FIELD.*/
DB3FD2 NUMBER(6,3), /* NO-KEY FIELD.*/
DB3FV2 CHAR(40), /* NO-KEY FIELD.*/
DB3FC5 CHAR(8), /* NO-KEY FIELD.*/
DB3FI3 NUMBER(8) NOT NULL, /* SECONDARY KEY 100TIMES.*/
DB3FC6 CHAR(30), /* NO-KEY FIELD.*/
DB3FL3 NUMBER(4), /* NO-KEY FIELD.*/
DB3FR3 NUMBER(4,2), /* NO-KEY FIELD.*/
DB3FD3 NUMBER(6,3), /* NO-KEY FIELD.*/
DB3FV3 CHAR(40)); /* NO-KEY FIELD.*/

_CLEARSCREEN(_GCLEARSCREEN);
PRINTF("*****\n");
PRINTF("TABLE DB300 HAS BEEN CREATED.\n");
PRINTF("*****\n\n");

/* ***** */
/* CREATE INDEXES FOR DB300 SECTION. */
/* ***** */

```

STRCPY(ERROR_MESSAGE,INDX300_ERR);

```
EXEC SQL CREATE UNIQUE INDEX DB3FI1_IX ON DB300(DB3FI1);
PRINTF(" ***** \n");
PRINTF(" UNIQUE INDEX DB3FI1_IX HAS BEEN CREATED. \n");
PRINTF(" AN INDEX ON DB300 TABLE FIELD DB3FI1. \n");
PRINTF(" ***** \n");
```

```
/* SECONDARY KEY 10 TIMES */
EXEC SQL CREATE INDEX DB3FI2_IX ON DB300(DB3FI2);
PRINTF(" SECONDARY INDEX DB3FI2_IX HAS BEEN CREATED. \n");
PRINTF(" AN INDEX ON DB300 TABLE FIELD DB3FI2. \n");
PRINTF(" ***** \n");
```

```
/* SECONDARY KEY 100 TIMES */
EXEC SQL CREATE INDEX DB3FI3_IX ON DB300(DB3FI3);
PRINTF(" SECONDARY INDEX DB3FI3_IX HAS BEEN CREATED. \n");
PRINTF(" AN INDEX ON DB300 TABLE FIELD DB3FI3. \n");
PRINTF(" ***** \n");
```

```
/* ***** */
/* CREATE TABLE DBINS SECTION. */
/* ***** */
```

```
STRCPY(ERROR_MESSAGE,TABLINS_CR_ERR);
EXEC SQL CREATE TABLE DBINS
(DBIFC1 CHAR(8), /* NO-KEY FIELD. */
DBIFI1 NUMBER(8) NOT NULL, /* SECONDARY KEY. */
DBIFC2 CHAR(30), /* NO-KEY FIELD. */
DBIFL1 NUMBER(4), /* NO-KEY FIELD. */
DBIFR1 NUMBER(4,2), /* NO-KEY FIELD. */
DBIFD1 NUMBER(6,3), /* NO-KEY FIELD. */
DBIFV1 CHAR(40), /* NO-KEY FIELD. */
DBIFC3 CHAR(8), /* NO-KEY FIELD. */
DBIFI2 NUMBER(8) NOT NULL, /* NO-KEY FIELD. */
DBIFC4 CHAR(30), /* NO-KEY FIELD. */
DBIFL2 NUMBER(4), /* NO-KEY FIELD. */
DBIFR2 NUMBER(4,2), /* NO-KEY FIELD. */
DBIFD2 NUMBER(6,3), /* NO-KEY FIELD. */
DBIFV2 CHAR(40)); /* NO-KEY FIELD. */
```

```
_CLEARSCREEN(_GCLEARSCREEN);
PRINTF(" ***** \n");
PRINTF(" TABLE DBINS HAS BEEN CREATED. \n");
PRINTF(" ***** \n\n");
```

```
/* ***** */
/* CREATE INDEXES FOR DBINS SECTION. */
/* ***** */
```

```
STRCPY(ERROR_MESSAGE,INDXINS_ERR);
/* SECONDARY KEY N TIMES DEPENDING ON NO. OF TRANSACTIONS. */
EXEC SQL CREATE INDEX DBIFI1_IX ON DBINS(DBIFI1);
PRINTF(" ***** \n");
PRINTF(" SECONDARY INDEX DBIFI1_IX HAS BEEN CREATED. \n");
PRINTF(" AN INDEX ON DBINS TABLE FIELD DBIFI1. \n");
PRINTF(" ***** \n");
```

```
/* ***** */
/* CREATE TABLE DBUPD SECTION. */
```

```

/* ----- */

STRCPY(ERROR_MESSAGE,TABLUPD_CR_ERR);
EXEC SQL CREATE TABLE DBUPD
(DBUFC1 CHAR(8), /* NO-KEY FIELD. */
DBUFI1 NUMBER(8) NOT NULL, /* UNIQUE KEY. */
DBUFC2 CHAR(30), /* NO-KEY FIELD. */
DBUFL1 NUMBER(4), /* NO-KEY FIELD. */
DBUFR1 NUMBER(4,2), /* NO-KEY FIELD. */
DBUFD1 NUMBER(6,3), /* NO-KEY FIELD. */
DBUFV1 CHAR(40), /* NO-KEY FIELD. */
DBUFC3 CHAR(8), /* NO-KEY FIELD. */
DBUFI2 NUMBER(8) NOT NULL, /* SECONDARY KEY.*/
DBUFC4 CHAR(30), /* NO-KEY FIELD. */
DBUFL2 NUMBER(4), /* NO-KEY FIELD. */
DBUFR2 NUMBER(4,2), /* NO-KEY FIELD. */
DBUFD2 NUMBER(6,3), /* NO-KEY FIELD. */
DBUFV2 CHAR(40)); /* NO-KEY FIELD. */

_CLEARSCREEN(_GCLEARSCREEN);
PRINTF("-----\n");
PRINTF(" TABLE DBUPD HAS BEEN CREATED.\n");
PRINTF("-----\n\n");

/* ----- */
/* CREATE INDEXES FOR DBUPD SECTION. */
/* ----- */

STRCPY(ERROR_MESSAGE,INDXUPD_ERR);

EXEC SQL CREATE UNIQUE INDEX DBUFI1_IX ON DBUPD(DBUFI1);
PRINTF("-----\n");
PRINTF(" UNIQUE INDEX DBUFI1_IX HAS BEEN CREATED. \n");
PRINTF(" AN INDEX ON DBUPD TABLE FIELD DB2FI1. \n");
PRINTF("-----\n");

/* ----- */
/* NORMAL PROGRAM TERMINATION SECTION. */
/* LOG OUT AND TERMINATE PROCESSING. */
/* ----- */

EXEC SQL WHENEVER SQLERROR CONTINUE; /* DON'T TRAP ERRORS. */
EXEC SQL COMMIT WORK RELEASE; /* LOG OFF DATABASE. */
PRINTF("\n\n-----\n");
PRINTF(" THE PROGRAM HAS TERMINATED NORMALLY. \n");
PRINTF("-----\n");

EXIT(0);

/* ***** */
/* THE ERROR HANDLING TRAPS AND ROUTINES. */
/* ..... */

SQL_ERROR_ROUTINE:
PUTCHAR(BEEP);/* BEEP */
PRINTF("%70S\n",LINE);
PRINTF(" DATABASE SIZE : %LU\n",ROWS);
PRINTF(" LAST KEY VALUE IS: %D\n",KEY_VALUE);
PRINTF("%70S\n",LINE);
PRINTF("%70S\n",ERROR_MESSAGE);
PRINTF("%70S\n",SQLCA.SQLERRM.SQLERRMC);
PRINTF("%70S\n",LINE);

```

```
PUTCHAR(BEEP);/* BEEP */  
EXEC SQL WHENEVER SQLERROR CONTINUE;  
EXEC SQL ROLLBACK WORK RELEASE;  
EXIT(1);  
  
} /* PROGRAM END */
```

APPENDIX B

APPENDIX B: The CITY Benchmark Tables Load

```

/* ===== */
/* THE CITY BENCHMARK TEST.      */
/* DATA LOAD PROGRAM.           */
/* ===== */

#include <STDIO.H>
#include <CTYPE.H>
#include <STRING.H>
#include <SQLCA.H>
#include <PROCESS.H>
#include <STDLIB.H>
#include <GRAPH.H>
#include <TIME.H>
#define BEEP '\A'
#define MAXPATH 64
#define MAXLINE 80
#define DSECONDS 1
#define NLETTERS 10

/* ===== */
/* DECLARE TPS RECORD STRUCTURE TAG */
/* ===== */

STRUCT TPS_REC_LAYOUT
{ /* STRUCTURE BEGINING */
CHAR DBTYPE[15]; /* DATABASE NAME.      */
LONG DBSIZE; /* DATABASE SIZE.          */
INT NLOOPS; /* NUMBER OF LOOPS.          */
INT LOOPTIME; /* LOOP TIME IN SECONDS.    */
INT TOTTIME; /* TOTAL TIME IN SECONDS.   */
LONG TOT_TRANS; /* TOT. NUMBER OF TRANSACTIONS. */
LONG AVR_TRANS; /* AVR. TRANSACTIONS PER LOOP. */
DOUBLE TRN_PER_SEC; /* NUMBER OF TRANSACTIONS/SEC. */
DOUBLE IT_RESP_TIME; /* ITERATION RESPONSE TIME. */
TIME T TIME_DATE; /* TIME AND DATE OF TEST. */
}; /* STUCTURE END */

/* ===== */
/* DECLARE TPS RECORD STRUCTURE */
/* ===== */

STRUCT TPS_REC_LAYOUT TPS_IO_REC;

/* ===== */
/* THIS PART CREATES FIVE TABLES FOR THE CITY BENCHMARK */
/* DB100, DB200, DB300 DBINS AND DBUPD THEN LOAD THEM.*/
/* ===== */

EXEC SQL BEGIN DECLARE SECTION;

/* ===== */
/* THE PASSWORD DECLARATION SECTION */
/* ===== */

VARCHAR UID[20];
VARCHAR PWD[20];

/* ===== */
/* THE DATABASES INTERMEDIATE DECLARATION SECTION */
/* ===== */

```

```

CHAR    DBFC1 [8];
INT     DBFI;
CHAR    DBFC2 [30];
LONG    DBFL;
FLOAT   DBFR;
DOUBLE  DBFD;
VARCHAR DBFV [40];

UNSIGNED INT KEY_VALUE;

/* ===== */
/* THE PROGRAM VARIABLES DECLARATION SECTION. */
/* ===== */

LONG   R_1    = 0; /* UNIQUE NO. OF ROWS COUNTER. */
LONG   R_10   = 0; /* NO. OF ROWS COUNTER REPEATED EVERY 10 ROWS. */
LONG   R_100  = 0; /* NO. OF ROWS COUNTER REPEATED EVERY 100 ROWS. */

INT     VI = 50; /* INTEGER VALUE */
LONG    VL = 50; /* LONG VALUE */
FLOAT   VF = 50; /* FLOAT VALUE */
FLOAT   VR = 50; /* FLOAT VALUE */
DOUBLE  VD = 50; /* DOUBLE VALUE */

LONG DB_SIZE; /* DATABASE SIZE */

/* CHARACTER FIELDS CAR1FIELD1, CHAR2FIELD1 AND VARFIELD1*/

STATIC CHAR CHAR1FIELD1 [8] = {"00000001"};
STATIC CHAR CHAR2FIELD1 [30] = {"ABABA ABABA ABABA ABABA ABABA"};
STATIC CHAR VARFIELD1 [40] = {"ABABA ABABA ABABA ABABA ABABA ABA
AAAAA"};

EXEC SQL END DECLARE SECTION;

/* ===== */
/* INCLUDE SQLCA COMMAND (SQL COMM AREA). */
/* ===== */

EXEC SQL INCLUDE SQLCA;

/* ===== */
/* THE MAIN PROGRAM START. */
/* ===== */

MAIN()
{ /* PROGRAM START */

FILE *TPS_FP;
LONG RANDP();

LONG X, Y, Z;

STATIC CHAR PATH_NAME_1[MAXPATH] = {"C:\CITYLOAD.TXT"};

/* RANDOM NUMBER GENERATION VARIABLES */
LONG SEED, RT_VALUE, N_ROWS, R_LIMIT10;
INT R_LIMIT = 30000; /* RANDP MAXIMUM VALUE */
UNSIGNED LONG ROWS; /* DATABASE SIZE */

```


APPENDIX B: The CITY Benchmark Tables Load.

```
UNSIGNED LONG ROWS_START, /* NUMBER OF ROWS START VALUE */
ROWS_END, /* NUMBER OF ROWS END VALUE */
ROWS_STEP; /* NUMBER OF ROWS STEP VALUE */

INT SECONDS;

INT I1 = 0, /* LOOP1 INDEX */
I2 = 0, /* LOOP2 INDEX */
I3 = 0; /* LOOP3 INDEX */

INT I, /* MAIN LOOP INDEX*/
J = 7, /* END OF 8 CHAR FIELD POINTER */
K = 29, /* END OF 30 CHAR FIELD POINTER */
L = 0, /* START OF VAR_FIELDS POINTER */
M = 38; /* END OF VAR_FIELDS POINTER */

UNSIGNED LONG I_LOOPS; /* INDEX OF NUMBER OF RECORDS. */
UNSIGNED LONG ROWS_LOOP = 0; /* DATABASE LOAD LOOPS INDEX */
UNSIGNED LONG START = 0, /* TPS START TIME. */
END = 0, /* TPS END TIME. */
G_TIME1 = 0, /* USED TO GET START VALUE FROM TIME. */
G_TIME2 = 0; /* USED TO GET END VALUE FROM TIME. */

UNSIGNED LONG TRANS_COUNT = 0, /* LOOP TRANS_COUNTER.*/
TOT_TRANS_COUNT = 0, /* TOTAL OF LOOP TRANS_COUNTERS. */
AVR_TRANS_COUNT = 0; /* AVERAGE OF LOOP TRANS_COUNTERS. */

DOUBLE RESP_TIME, /* AVERAGE TRANSACTIONS RESPONSE TIME. */
RESP_TIME_T, /* TOTAL TRANSACTIONS RESPONSE TIME. */
RESP_TIME_1, /* ONE TRANSACTIONS RESPONSE TIME. */
RESP_PER_STEP, /* RESPONSE TIME PER STEP. */
TRANS_PER_SEC; /* TRANSACTIONS/SEC. */

TIME_T RESP_TIME_START, /* RESPONSE TIME START. */
RESP_TIME_STOP; /* RESPONSE TIME END. */

UNSIGNED LONG RESP_COUNT;

STATIC CHAR DB_TYPE[15] = {"ORACLE. "};

LONG QS=0;
LONG QU1=0;
LONG QU2=0;
LONG QF=0;
LONG IN_REC=0;

DOUBLE TR_PER_SECOND= 0; /* AVERAGE NUMBER OF TRANSACTIONS/SEC. */

STATIC CHAR ERROR_MESSAGE[71] =
{
    "
    "};

STATIC CHAR LINE[71] =
{
    "#####
    "};

STATIC CHAR CONN_ERR[71] =
{
    "AN ERROR OCCURRED DURING CONNECTION TO THE DBMS.
    "};

STATIC CHAR QL_COUNT_ERR[71] =
{
    "AN ERROR OCCURRED DURING COUNTING ROWS IN DB100.
    "};

STATIC CHAR ROW_INSRT_ERR1[71] =
```

APPENDIX B: The CITY Benchmark Tables Load.

```
{ "AN ERROR OCCURRED DURING TABLE DB100 LOADING.                "};

STATIC CHAR ROW_INSRT_ERR2[71] =
{ "AN ERROR OCCURRED DURING TABLE DB200 LOADING.                "};

STATIC CHAR ROW_INSRT_ERR3[71] =
{ "AN ERROR OCCURRED DURING TABLE DB300 LOADING.                "};

STATIC CHAR ROW_INSRT_ERRI[71] =
{ "AN ERROR OCCURRED DURING TABLE DBINS LOADING.                "};

STATIC CHAR ROW_INSRT_ERRU[71] =
{ "AN ERROR OCCURRED DURING TABLE DBUPD LOADING.                "};

STATIC CHAR F_MSGL[47] =
{ "*****" };
STATIC CHAR F_MSG1[32] = { "TOTAL NUMBER OF INSERTED ROWS: " };
STATIC CHAR F_MSG2[47] = { "THE PROGRAM HAS BEEN TERMINATED SUCCESSFULLY. " };

/* ===== */
/* CLEAR SCREEN COMMAND */
/* ===== */

_CLEARSCREEN(_GCLEARSCREEN);

/* ===== */
/* LOG INTO THE DBMS USERID AND PASSWORD. */
/* ===== */

PRINTF("PLEASE ENTER USERID: ");
SCANF("%s", UID.ARR);
PRINTF("PLEASE ENTER PASS WORD: ");
SCANF("%s", PWD.ARR);
UID.LEN = STRLEN(UID.ARR);
PWD.LEN = STRLEN(PWD.ARR);

/* ===== */
/* LOG INTO THE DBMS EVERY TIME THE PROGRAM START A NEW LOOP. */
/* ===== */

STRCPY(ERROR_MESSAGE, CONN_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL CONNECT :UID IDENTIFIED BY :PWD;

PRINTF("** ===== *\n");
PRINTF("** CONNECTED TO THE SYSTEM USER: %s *\n", UID.ARR);
PRINTF("** ===== *\n\n");

/* ===== */
/* COUNT NUMBER OF ROWS IN TABLES */
/* ===== */

STRCPY(ERROR_MESSAGE, QL_COUNT_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL SELECT COUNT (DISTINCT (DB1FIL))
INTO      :DB_SIZE
FROM DB100;

IF (DB_SIZE < 1)
{ /* IF DB_SIZE < 1 END */
```

APPENDIX B: The CITY Benchmark Tables Load.

```
PRINTF("\n\n* ===== *\n");
PRINTF("** THE TEST DATABASE IS EMPTY. *\n");
PRINTF("** ===== *\n");
} /* IF DB_SIZE < 1 END */

/* ===== */
/* GET NUMBER OF ROWS TO BE INSERTED */
/* ===== */

PRINTF("CURRENT DATABASE SIZE IN ROWS IS: %D.\n", DB_SIZE);

ROWS_START = DB_SIZE;

GET_SIZE:
PRINTF("PLEASE ENTER NUMBER OF ROWS TO BE ADDED TO THE TEST DATABASE: ");
SCANF("%LU", &ROWS_END);
IF (ROWS_END < 1)
{ /* IF DB_SIZE < 1 END */
  PRINTF("\n* ===== *\n");
  PRINTF("** NUMBER OF ADDED ROWS MUST BE GREATER THAN 1. *\n");
  PRINTF("** ===== *\n");
  GOTO GET_SIZE;
} /* IF DB_SIZE < 1 END */

_CLEARSCREEN(_GCLEARSCREEN);

/* ===== */
/* THE MAIN LOOP START. */
/* ===== */
*/

R_100 = ROWS_START;
R_10 = ROWS_START;
R_1 = ROWS_START;

FOR (ROWS = (ROWS_START+1); ROWS < (ROWS_START+ROWS_END+1); ROWS++)
{ /* THE MAIN LOOP START */

  /* ===== */
  /* LOAD TABLES SECTION. THIS SECTION ENDS WITH COMMIT INSERT ROWS. */
  /* ===== */

  /* ===== */
  /* LOOP FOR ROWS REPEATED 100 TIMES */
  /* ===== */

  FOR (I3 = 0; I3 < NLETTERS; ++I3)
  { /* FOR I3 START */

    ++R_100; /* INTEGER VALUE REPEATED 100 TIMES. */

    /* ===== */
    /* LOOP FOR ROWS REPEATED 10 TIMES */
    /* ===== */

    FOR (I2 = 0; I2 < NLETTERS; ++I2)
    { /* FOR I2 START */

      ++R_10; /* INTEGER VALUE REPEATED 100 TIMES. */

      /* ===== */
```

APPENDIX B: The CITY Benchmark Tables Load.

```
/* LOOP FOR ROWS WITH UNIQUE KEY. */
/* ===== */

FOR ( i1 = 0; i1 < NLETTERS; ++i1)
{ /* FOR i1 START */

++R_1;

/* ===== */
/* INSERT A ROW IN DB100 AND CHECK ERROR THEN */
/* COMMIT WORK AND PROGRAM TERMINATION SECTION. */
/* ===== */

EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;

STRCPY(ERROR_MESSAGE, ROW_INSRT_ERR1);
EXEC SQL INSERT INTO
DB100 (DB1FC1, DB1FI1, DB1FC2,
DB1FL1, DB1FR1, DB1FD1, DB1FV1)
VALUES (:CHAR1FIELD1, :R_1, :CHAR2FIELD1,
:VL, :VF, :VD, :VARFIELD1);

PRINTF(" * ===== *\n");
PRINTF(" * ROW NUMBER: %7D HAS BEEN INSERTED IN DB100. *\n", R_1);

/* ===== */
/* INSERT A ROW IN DB200 AND CHECK ERROR THEN */
/* COMMIT WORK AND PROGRAM TERMINATION SECTION. */
/* ===== */

STRCPY(ERROR_MESSAGE, ROW_INSRT_ERR2);
EXEC SQL INSERT INTO
DB200 (DB2FC1, DB2FI1, DB2FC2,
DB2FL1, DB2FR1, DB2FD1, DB2FV1,
DB2FC3, DB2FI2, DB2FC4,
DB2FL2, DB2FR2, DB2FD2, DB2FV2)
VALUES (:CHAR1FIELD1, :R_1, :CHAR2FIELD1,
:VL, :VF, :VD, :VARFIELD1,
:CHAR1FIELD1, :R_10, :CHAR2FIELD1,
:VL, :VF, :VD, :VARFIELD1);

PRINTF(" * ROW NUMBER: %7D HAS BEEN INSERTED IN DB200. *\n", R_1);

/* ===== */
/* INSERT A ROW IN DB300 AND CHECK ERROR THEN */
/* COMMIT WORK AND PROGRAM TERMINATION SECTION. */
/* ===== */

STRCPY(ERROR_MESSAGE, ROW_INSRT_ERR3);
EXEC SQL INSERT INTO
DB300 (DB3FC1, DB3FI1, DB3FC2,
DB3FL1, DB3FR1, DB3FD1, DB3FV1,
DB3FC3, DB3FI2, DB3FC4,
DB3FL2, DB3FR2, DB3FD2, DB3FV2,
DB3FC5, DB3FI3, DB3FC6,
DB3FL3, DB3FR3, DB3FD3, DB3FV3)
VALUES (:CHAR1FIELD1, :R_1, :CHAR2FIELD1,
:VL, :VF, :VD, :VARFIELD1,
:CHAR1FIELD1, :R_10, :CHAR2FIELD1,
:VL, :VF, :VD, :VARFIELD1,
:CHAR1FIELD1, :R_100, :CHAR2FIELD1,
```

```

:VL, :VF, :VD, :VARFIELD1);

PRINTF("** ROW NUMBER: %7D HAS BEEN INSERTED IN DB300. *\n",R_1);

/* ===== */
/* INSERT A ROW IN DBUPD AND CHECK ERROR THEN */
/* COMMIT WORK AND PROGRAM TERMINATION SECTION. */
/* ===== */

STRCPY(ERROR_MESSAGE,ROW_INSRT_ERRU);
EXEC SQL INSERT INTO
DBUPD (DBUFC1,DBUF11,DBUFC2,
DBUFL1,DBUFR1,DBUFD1,DBUFV1,
DBUFC3,DBUF12,DBUFC4,
DBUFL2,DBUFR2,DBUFD2,DBUFV2)
VALUES (:CHAR1FIELD1, :R_1, :CHAR2FIELD1,
:VL, :VF, :VD, :VARFIELD1,
:CHAR1FIELD1, :R_10, :CHAR2FIELD1,
:VL, :VF, :VD, :VARFIELD1);

PRINTF("** ROW NUMBER: %7D HAS BEEN INSERTED IN DBUPD. *\n",R_1);

IF ( R_1 == (ROWS_START+ROWS_END) ) GOTO END_OF_LOAD_ROUTINE;

} /* FOR I1 END */
} /* FOR I2 END */
} /* FOR I3 END */
} /* FOR (ROWS = ROWS_STARTEND */

/* ===== */
/* COMMIT WORK TO COMPLETE INSERT ROWS PART. */
/* ===== */

END_OF_LOAD_ROUTINE:
EXEC SQL COMMIT WORK;
PRINTF("\n* ===== *\n");
PRINTF("** THE DATA LOAD PART TERMINATED SUCCESSFULLY. *\n");
PRINTF("** ===== *\n");
PRINTF("** ===== *\n");
PRINTF("** ===== *\n");
PRINTF("** THE PROGRAM HAS TERMINATED NORMALLY. *\n");
PRINTF("** ===== *\n");

EXIT(0);

/* ***** */
/* THE ERROR HANDLING TRAPS AND ROUTINES. */
/* ***** */

SQL_ERROR_ROUTINE:
PUTCHAR(BEEP);/* BEEP */
PRINTF("** %70s *\n",LINE);
PRINTF("** DATABASE SIZE : %LU\n",ROWS);
PRINTF("** LAST KEY VALUE IS: %D\n",KEY_VALUE);
PRINTF("** %70s *\n",LINE);
PRINTF("** %70s *\n", ERROR_MESSAGE);
PRINTF("** %70s *\n", SQLCA.SQLERRM.SQLERRMC);
PRINTF("** %70s *\n",LINE);
PUTCHAR(BEEP);/* BEEP */
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL ROLLBACK WORK RELEASE;
EXIT(1);

```

```
}    /* PROGRAM END */
```

;

APPENDIX C

APPENDIX C: The CITY Benchmark Transactions

```

/* ===== */
/* THE CITY BENCHMARK TRANSACTIONS. */
/* TO RUN AFTER CITYCRT AND CITYLOAD. */
/* EXECUTABLE FORM IS CITY.EXE AND RESULTS ARE CITY.RES. */
/* ===== */

#include <STDIO.H>
#include <CTYPE.H>
#include <STRING.H>
#include <SQLCA.H>
#include <PROCESS.H>
#include <STDLIB.H>
#include <GRAPH.H>
#include <TIME.H>
#define BEEP '\A'
#define MAXPATH 64
#define MAXLINE 80
#define DSECONDS 1
#define NLETTERS 10

/* ===== */
/* DECLARE TPS RECORD STRUCTURE TAG */
/* ===== */

STRUCT TPS_REC_LAYOUT
{ /* STRUCTURE BEGINING */
CHAR DBTYPE[15]; /* DATABASE NAME. */
LONG DBSIZE; /* DATABASE SIZE. */
INT NLOOPS; /* NUMBER OF LOOPS. */
INT LOOPTIME; /* LOOP TIME IN SECONDS. */
INT TOTTIME; /* TOTAL TIME IN SECONDS. */
LONG TOT_TRANS; /* TOT. NUMBER OF TRANSACTIONS. */
LONG AVR_TRANS; /* AVR. TRANSACTIONS PER LOOP. */
DOUBLE TRN_PER_SEC; /* NUMBER OF TRANSACTIONS/SEC. */
DOUBLE IT_RESP_TIME; /* ITERATION RESPONSE TIME. */
TIME_T TIME_DATE; /* TIME AND DATE OF TEST. */
}; /* STUCTURE END */

/* ===== */
/* DECLARE TPS RECORD STRUCTURE */
/* ===== */

STRUCT TPS_REC_LAYOUT TPS_IO_REC;

/* ===== */
/* DECLARE PRINT PROCEDURE */
/* ===== */

PRN_REC1();

/* ===== */
/* THIS PART CREATES FIVE TABLES FOR THE CITY BENCHMARK */
/* DB100, DB200, DB300 DBINS AND DBUPD THEN LOAD THEM. */
/* ===== */

EXEC SQL BEGIN DECLARE SECTION;

/* ===== */
/* THE PASSWORD DECLARATION SECTION */
/* ===== */

```



```

VARCHAR UID[20];
VARCHAR PWD[20];

/* ===== */
/* THE DATABASES INTERMEDIATE DECLARATION SECTION */
/* ===== */

CHAR    DBFC1 [8];
INT     DBFI;
CHAR    DBFC2 [30];
LONG    DBFL;
FLOAT   DBFR;
DOUBLE  DBFD;
VARCHAR DBFV [40];

UNSIGNED INT KEY_VALUE;

/* ===== */
/* THE PROGRAM VARIABLES DECLARATION SECTION. */
/* ===== */

LONG    R_1    = 0; /* UNIQUE NO. OF ROWS COUNTER. */
LONG    R_10   = 0; /* NO. OF ROWS COUNTER REPEATED EVERY 10 ROWS. */
LONG    R_100  = 0; /* NO. OF ROWS COUNTER REPEATED EVERY 100 ROWS. */

INT     VI = 50; /* INTEGER VALUE */
LONG    VL = 50; /* LONG VALUE */
FLOAT   VF = 50; /* FLOAT VALUE */
FLOAT   VR = 50; /* FLOAT VALUE */
DOUBLE  VD = 50; /* DOUBLE VALUE */

LONG    DB_Size; /* DATABASE SIZE */

/* CHARACTER FIELDS CHAR1FIELD1, CHAR2FIELD1 AND VARFIELD1*/

STATIC CHAR CHAR1FIELD1 [8]  = {"00000001"};
STATIC CHAR CHAR2FIELD1 [30] = {"ABABA ABABA ABABA ABABA ABABA"};
STATIC CHAR VARFIELD1    [40] = {"ABABA ABABA ABABA ABABA ABABA ABA
AAAAA"};

EXEC SQL END DECLARE SECTION;

/* ===== */
/* INCLUDE SQLCA COMMAND (SQL COMM AREA). */
/* ===== */

EXEC SQL INCLUDE SQLCA;

/* ===== */
/* EXTERNAL VARIABLES FOR THE RANDOM NUMBER GENERATION PROGRAM */
/* ===== */
EXTERN INT SRAND1(), RAND2(); /* EXTERN IS OPTIONAL */
    }

/* ===== */
/* THE MAIN PROGRAM START. */
/* ===== */

MAIN()
{
    /* PROGRAM START */

```

```

FILE *TPS_FP;
LONG RANDP();

LONG X, Y, Z;

STATIC CHAR PATH_NAME_1[MAXPATH] = {"C:\CITY.RES"};

/* RANDOM NUMBER GENERATION VARIABLES */
LONG SEED, RT_VALUE, N_ROWS, R_LIMIT10;
INT R_LIMIT = 30000; /* RANDP MAXIMUM VALUE */

UNSIGNED LONG ROWS_START, /* NUMBER OF ROWS START VALUE */
ROWS_END, /* NUMBER OF ROWS END VALUE */
ROWS_STEP; /* NUMBER OF ROWS STEP VALUE */

UNSIGNED LONG ROWS; /* DATABASE SIZE */

INT SECONDS;

INT I1 = 0, /* LOOP1 INDEX */
I2 = 0, /* LOOP2 INDEX */
I3 = 0; /* LOOP3 INDEX */

INT I, /* MAIN LOOP INDEX */
J = 7, /* END OF 8 CHAR FIELD POINTER */
K = 29, /* END OF 30 CHAR FIELD POINTER */
L = 0, /* START OF VAR_FIELDS POINTER */
M = 38; /* END OF VAR_FIELDS POINTER */

UNSIGNED LONG I_LOOPS; /* INDEX OF NUMBER OF RECORDS. */
UNSIGNED LONG ROWS_LOOP = 0; /* DATABASE LOAD LOOPS INDEX */
UNSIGNED LONG START = 0, /* TPS START TIME. */
END = 0, /* TPS END TIME. */
G_TIME1 = 0, /* USED TO GET START VALUE FROM TIME. */
G_TIME2 = 0; /* USED TO GET END VALUE FROM TIME. */

UNSIGNED LONG TRANS_COUNT = 0, /* LOOP TRANS_COUNTER. */
TOT_TRANS_COUNT = 0, /* TOTAL OF LOOP TRANS_COUNTERS. */
AVR_TRANS_COUNT = 0; /* AVERAGE OF LOOP TRANS_COUNTERS. */

DOUBLE RESP_TIME, /* AVERAGE TRANSACTIONS RESPONSE TIME. */
RESP_PER_STEP, /* RESPONSE TIME PER STEP. */
TRANS_PER_SEC; /* TRANSACTIONS/SEC. */

TIME_T RESP_TIME_START, /* RESPONSE TIME START. */
RESP_TIME_STOP; /* RESPONSE TIME END. */
LONG RESP_TIME_T, /* TOTAL TRANSACTIONS RESPONSE TIME. */
RESP_TIME_1; /* ONE TRANSACTIONS RESPONSE TIME. */

UNSIGNED LONG RESP_COUNT;

STATIC CHAR DB_TYPE[15] = {"ORACLE."};

LONG QS=0;
LONG QU1=0;
LONG QU2=0;
LONG QF=0;
LONG IN_REC=0;

DOUBLE TR_PER_SECOND= 0; /* AVERAGE NUMBER OF TRANSACTIONS/SEC. */

STATIC CHAR ERROR_MESSAGE[71] =

```

```

{ "
";

STATIC CHAR LINE[71] =
{ "#####
###" };

STATIC CHAR CONN_ERR[71] =
{ "AN ERROR OCCURRED DURING CONNECTION TO THE DBMS.          " };

STATIC CHAR F_MSGL[47] =
{ "#####" };
STATIC CHAR F_MSG1[32] = { "TOTAL NUMBER OF INSERTED ROWS: " };
STATIC CHAR F_MSG2[47] = { "THE PROGRAM HAS BEEN TERMINATED SUCCESSFULLY. " };

STATIC CHAR QL_COUNT_ERR[71] =
{ "AN ERROR OCCURRED DURING COUNTING ROWS IN DB100.          " };

STATIC CHAR QL_SEL1_ERR[71] =
{ "AN ERROR OCCURRED DURING QUALIFIED RETRIEVAL SELECT 1.    " };

STATIC CHAR QL_SEL2_ERR[71] =
{ "AN ERROR OCCURRED DURING QUALIFIED RETRIEVAL SELECT 2.    " };

STATIC CHAR QL_SEL3_ERR[71] =
{ "AN ERROR OCCURRED DURING QUALIFIED RETRIEVAL SELECT 3.    " };

STATIC CHAR QL_SEL4_ERR[71] =
{ "AN ERROR OCCURRED DURING QUALIFIED RETRIEVAL SELECT 4.    " };

STATIC CHAR QL_SEL5_ERR[71] =
{ "AN ERROR OCCURRED DURING QUALIFIED RETRIEVAL SELECT 5.    " };

STATIC CHAR QL_SEL6_ERR[71] =
{ "AN ERROR OCCURRED DURING QUALIFIED RETRIEVAL SELECT 6.    " };

STATIC CHAR QL_SEL7_ERR[71] =
{ "AN ERROR OCCURRED DURING QUALIFIED RETRIEVAL SELECT 7.    " };

STATIC CHAR QL_SEL8_ERR[71] =
{ "AN ERROR OCCURRED DURING QUALIFIED RETRIEVAL SELECT 8.    " };

STATIC CHAR ROW_INSRT_ERR[71] =
{ "AN ERROR OCCURRED DURING INSERTING A ROW IN DBINS.        " };

STATIC CHAR ROW_UPDATE_ERR1[71] =
{ "AN ERROR OCCURRED DURING UPDATING A ROW IN DBUPD, UPDATE OPERATION 1. " };

STATIC CHAR ROW_UPDATE_ERR2[71] =
{ "AN ERROR OCCURRED DURING UPDATING A ROW IN DBUPD, UPDATE OPERATION 2. " };

STATIC CHAR UPD_CUR_OPN_ERR[71] =
{ "AN ERROR HAS OCCURRED DURING UPDATE CURSOR OPEN.          " };

STATIC CHAR UPD_CUR_DCL_ERR[71] =
{ "AN ERROR HAS OCCURRED DURING UPDATE CURSOR DECLARATION.    " };

STATIC CHAR FT1_CUR_OPN_ERR[71] =
{ "AN ERROR HAS OCCURRED DURING FETCH1 CURSOR OPEN.          " };
STATIC CHAR FT1_CUR_DCL_ERR[71] =
{ "AN ERROR HAS OCCURRED DURING FETCH1 CURSOR DECLARATION.    " };
STATIC CHAR FT1_CUR_FTCH_ERR[71] =

```

```

{"AN ERROR HAS OCCURRED DURING FETCH1 EXECUTION.                };

STATIC CHAR FT2_CUR_OPN_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH2 CURSOR OPEN.              };
STATIC CHAR FT2_CUR_DCL_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH2 CURSOR DECLARATION.      };
STATIC CHAR FT2_CUR_FTCH_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH2 EXECUTION.                };

STATIC CHAR FT3_CUR_OPN_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH3 CURSOR OPEN.              };
STATIC CHAR FT3_CUR_DCL_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH3 CURSOR DECLARATION.      };
STATIC CHAR FT3_CUR_FTCH_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH3 EXECUTION.                };

STATIC CHAR FT4_CUR_OPN_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH4 CURSOR OPEN.              };
STATIC CHAR FT4_CUR_DCL_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH4 CURSOR DECLARATION.      };
STATIC CHAR FT4_CUR_FTCH_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH4 EXECUTION.                };

STATIC CHAR FT5_CUR_OPN_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH5 CURSOR OPEN.              };
STATIC CHAR FT5_CUR_DCL_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH5 CURSOR DECLARATION.      };
STATIC CHAR FT5_CUR_FTCH_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH5 EXECUTION.                };

STATIC CHAR FT6_CUR_OPN_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH6 CURSOR OPEN.              };
STATIC CHAR FT6_CUR_DCL_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH6 CURSOR DECLARATION.      };
STATIC CHAR FT6_CUR_FTCH_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH6 EXECUTION.                };

STATIC CHAR FT7_CUR_OPN_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH7 CURSOR OPEN.              };
STATIC CHAR FT7_CUR_DCL_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH7 CURSOR DECLARATION.      };
STATIC CHAR FT7_CUR_FTCH_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH7 EXECUTION.                };

STATIC CHAR FT8_CUR_OPN_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH8 CURSOR OPEN.              };
STATIC CHAR FT8_CUR_DCL_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH8 CURSOR DECLARATION.      };
STATIC CHAR FT8_CUR_FTCH_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH8 EXECUTION.                };

STATIC CHAR FT9_CUR_OPN_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH9 CURSOR OPEN.              };
STATIC CHAR FT9_CUR_DCL_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH9 CURSOR DECLARATION.      };
STATIC CHAR FT9_CUR_FTCH_ERR[71] =
{"AN ERROR HAS OCCURRED DURING FETCH9 EXECUTION.                };

/* ===== */
/* CLEAR SCREEN COMMAND */
/* ===== */

_CLEARSCREEN(_GCLEARSCREEN);

```

```

/* ===== */
/* OPEN FILES FOR TPS AND RESPONSE TIME. */
/* ===== */

IF ( (TPS_FP=FOPEN(PATH_NAME_1,"AB")) == NULL)
{ /* IF START */
PUTCHAR(BEEP); /* BEEP */
PUTCHAR(BEEP); /* BEEP */
PRINTF(" * ##### *\n");
PRINTF(" * CANNOT OPEN THE FILE, ABNORMAL END OF PROGRAM. *\n");
PRINTF(" * FILE NAME IS: %s\n",PATH_NAME_1);
PRINTF(" * ##### *\n");
GETCHAR();
} /* IF END. */

/* ===== */
/* LOG INTO THE DBMS USERID AND PASSWORD. */
/* ===== */

PRINTF("PLEASE ENTER USERID: ");
SCANF("%s",UID.ARR);
PRINTF("PLEASE ENTER PASS WORD: ");
SCANF("%s",PWD.ARR);
UID.LEN = STRLEN(UID.ARR);
PWD.LEN = STRLEN(PWD.ARR);

/* ===== */
/* LOG INTO THE DBMS EVERY TIME THE PROGRAM START A NEW LOOP. */
/* ===== */

STRCPY(ERROR_MESSAGE,CONN_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL CONNECT :UID IDENTIFIED BY :PWD;

PRINTF(" * ===== *\n");
PRINTF(" * CONNECTED TO THE SYSTEM USER: %s *\n",UID.ARR);
PRINTF(" * ===== *\n\n");

/* ===== */
/* COUNT NUMBER OF ROWS IN TABLES */
/* ===== */

STRCPY(ERROR_MESSAGE,QL_COUNT_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL SELECT COUNT (DISTINCT (DB1FI1))
INTO :DB_SIZE
FROM DB100;

IF (DB_SIZE < 1)
{ /* IF DB_SIZE < 1 END */
PRINTF("\n\n * ===== *\n");
PRINTF(" * THE TEST DATABASE IS EMPTY. *\n");
PRINTF(" * THE PROGRAM HAS TERMINATED ABNORMALLY. *\n");
PRINTF(" * ===== *\n");
EXIT(1);
} /* IF DB_SIZE < 1 END */

ROWS = DB_SIZE;

PRINTF("CURRENT DATABASE SIZE IN ROWS IS: %d.\n", ROWS);

```

```

/* ===== */
/* GET NUMBER OF ROWS TO BE INSERTED */
/* ===== */

GET_TIME:
PRINTF("TIME LOOP LENGTH SHOULD BE 300 SECONDS OR MORE.\n");
PRINTF("PLEASE ENTER TIME LOOP LENGTH IN SECONDS: ");
SCANF("%d",&SECONDS);
IF (SECONDS < 300)
{ /* IF SECONDS START */
PRINTF(" * ===== *\n");
PRINTF(" * TIME LOOP LENGTH SHOULD BE 300 SECONDS OR MORE. *\n");
PRINTF(" * PLEASE ENTER NEW LOOP TIME THEN PRESS ENTER. *\n");
PRINTF(" * ===== *\n");
GOTO GET_TIME;
} /* IF SECONDS END */

_CLEARSCREEN(_GCLEARSCREEN);

/* ===== */
/* THE CITY BENCHMARK TRANSACTIONS. */
/* ===== */
*/

/* ===== */
/* DUMMY LOOPS TO ADJUST TIME. */
/* ===== */

START= TIME(&G_TIME1); /* GET SYSTEM ELAPSED TIME IN SECONDS. */
END= START + DSECONDS; /* CALCULATE END TIME IN SECONDS */
DO
{ /* START OF DO LOOP BODY. */
PRINTF("DUMMY LOOPS TO ADJUST TIME\n");
} /* END OF DO LOOP BODY. */
WHILE (TIME(&G_TIME1) < END);
_CLEARSCREEN(_GCLEARSCREEN);

/* ===== */
/* PROGRAM LOOPS TO CALCULATE TPS AND RESPONSE TIME. */
/* ===== */

TRANS_COUNT = 0; /* SET NUMBER OF TRANSACTIONS/LOOP TO 0 */
RESP_TIME_T = 0; /* SET TIME OF TRANSACTIONS/LOOP TO 0 */

/* ===== */
/* GET TIME START AND TIME END TO CALCULATE TPS. */
/* ===== */

START = TIME(&G_TIME1); /* GET SYSTEM ELAPSED TIME IN SECONDS. */
END = START + SECONDS; /* CALCULATE END TIME IN SECONDS */
R_LIMIT10 = ROWS/10;

DO
{ /* START OF DO LOOP BODY. */

/* ===== */
/* GET START TIME TO CALCULATE RESPONSE/ITERATION */
/* ===== */

TIME(&RESP_TIME_START);

```

```

/* ===== */
/* 1. RETRIEVE 1 ROW FROM DB100. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,ROWS);

STRCPY(ERROR_MESSAGE,QL_SEL1_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL SELECT DB1FC1, DB1FI1, DB1FC2, DB1FL1, DB1FR1, DB1FD1,
DB1FV1
INTO :DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV
FROM DB100
WHERE DB1FI1 = :KEY_VALUE;

/* ===== */
/* 2. RETRIEVE 1 ROW FROM DB200. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,ROWS);

STRCPY(ERROR_MESSAGE,QL_SEL3_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL SELECT DB2FC1, DB2FI1, DB2FC2, DB2FL1, DB2FR1, DB2FD1,
DB2FV1
INTO :DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV
FROM DB200
WHERE DB2FI1 = :KEY_VALUE;

/* ===== */
/* 3.. RETRIEVE 1 ROW FROM DB300. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,ROWS);

STRCPY(ERROR_MESSAGE,QL_SEL7_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL SELECT DB3FC1, DB3FI1, DB3FC2, DB3FL1, DB3FR1, DB3FD1,
DB3FV1
INTO :DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV
FROM DB300
WHERE DB3FI1 = :KEY_VALUE;

/* ===== */
/* 4. RETRIEVE 1 ROW FROM DB100. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,ROWS);

```

```

STRCPY(ERROR_MESSAGE,QL_SEL2_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL SELECT DB1FC1, DB1FI1, DB1FC2, DB1FL1, DB1FR1, DB1FD1,
DB1FV1
INTO      :DBFC1, :DBFI,  :DBFC2, :DBFL,  :DBFR,  :DBFD,  :DBFV
FROM DB100
WHERE DB1FI1 = :KEY_VALUE;

/* ===== */
/* 5. RETRIEVE 1 ROW FROM DB200. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,ROWS);

STRCPY(ERROR_MESSAGE,QL_SEL4_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL SELECT DB2FC1, DB2FI1, DB2FC2, DB2FL1, DB2FR1, DB2FD1,
DB2FV1
INTO      :DBFC1, :DBFI,  :DBFC2, :DBFL,  :DBFR,  :DBFD,  :DBFV
FROM DB200
WHERE DB2FI1 = :KEY_VALUE;

/* ===== */
/* 6. RETRIEVE 1 ROW FROM DB300. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,ROWS);

STRCPY(ERROR_MESSAGE,QL_SEL8_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL SELECT DB3FC1, DB3FI1, DB3FC2, DB3FL1, DB3FR1, DB3FD1,
DB3FV1
INTO      :DBFC1, :DBFI,  :DBFC2, :DBFL,  :DBFR,  :DBFD,  :DBFV
FROM DB300
WHERE DB3FI1 = :KEY_VALUE;

/* ===== */
/* 7. RETRIEVE 1 ROW FROM DB200. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,ROWS);

STRCPY(ERROR_MESSAGE,QL_SEL5_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL SELECT DB2FC1, DB2FI1, DB2FC2, DB2FL1, DB2FR1, DB2FD1,
DB2FV1
INTO      :DBFC1, :DBFI,  :DBFC2, :DBFL,  :DBFR,  :DBFD,  :DBFV
FROM DB200
WHERE DB2FI1 = :KEY_VALUE;

```



```

/* ===== */
/* 8. RETRIEVE 1 ROW FROM DB300. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,ROWS);

STRCPY(ERROR_MESSAGE,QL_SEL8_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL SELECT DB3FC1, DB3FI1, DB3FC2, DB3FL1, DB3FR1, DB3FD1,
DB3FV1
INTO :DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV
FROM DB300
WHERE DB3FI1 = :KEY_VALUE;

/* ===== */
/* 9.UPDATE A SINGLE ROW IN DBUPD USING INTEGER KEY. */
/* ===== */

/* ===== */
/* PRINT BEFORE UPDATE RECORD IMAGE. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,ROWS);

STRCPY(ERROR_MESSAGE,ROW_UPDATE_ERR1);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL UPDATE DBUPD
SET DBUFL1 = :VL,
DBUFR1 = :VR,
DBUFD1 = :VD
WHERE DBUFI1 =:KEY_VALUE;

/* ===== */
/* 10. UPDATE 10 ROWS IN DBUPD. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,R_LIMIT10);

STRCPY(ERROR_MESSAGE,UPD_CUR_DCL_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL DECLARE UPD_CUR CURSOR
FOR SELECT
DBUFC1, DBUFI1, DBUFC2, DBUFL1, DBUFR1, DBUFD1, DBUFV1
FROM DBUPD
WHERE DBUFI2 = :KEY_VALUE
FOR UPDATE OF
DBUFL1, DBUFR1, DBUFD1;

STRCPY(ERROR_MESSAGE,UPD_CUR_OPN_ERR);

```

APPENDIX C: The CITY Benchmark Transactions.

```
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL OPEN UPD_CUR;

STRCPY(ERROR_MESSAGE, ROW_UPDATE_ERR2);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL WHENEVER NOT FOUND GOTO UPD_WORK_DONE;

FOR(;;)
{ /* FOR START */
EXEC SQL FETCH UPD_CUR INTO
:DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV;

EXEC SQL UPDATE DBUPD
SET DBUFL1 = :VL,
DBUFR1 = :VR,
DBUFD1 = :VD
WHERE CURRENT OF UPD_CUR;
} /* FOR END */

UPD_WORK_DONE:
EXEC SQL CLOSE UPD_CUR;

/* ===== */
/* 11. RETRIEVE 10 ROWS FROM DB200. */
/* ===== */

/* ===== */
/* STEP 1 DECLARE THE CURSOR TO BE USED. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,R_LIMIT10);

STRCPY(ERROR_MESSAGE, FT1_CUR_DCL_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL DECLARE CF1 CURSOR FOR
SELECT DB2FC1, DB2FI1, DB2FC2, DB2FL1, DB2FR1, DB2FD1, DB2FV1
FROM DB200
WHERE DB2FI2 = :KEY_VALUE;

/* ===== */
/* STEP 2 OPEN THE CURSOR TO BE USED. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT1_CUR_OPN_ERR);
EXEC SQL OPEN CF1;

/* ===== */
/* STEP 3 USE FETCH TO RETRIEVE ALL REQUIRED RECORDS. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT1_CUR_FTCH_ERR);
EXEC SQL WHENEVER NOT FOUND GOTO WORK_DONE_F1;

FOR(;;)
{ /* FOR START */
EXEC SQL FETCH CF1 INTO
:DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV;
} /* FOR END */
```

```

/* ===== */
/* STEP 4 CLOSE THE CURSOR. */
/* ===== */

WORK_DONE_F1:
EXEC SQL CLOSE CF1;

/* ===== */
/* 12. RETRIEVE 10 ROWS FROM DB300. */
/* ===== */

/* ===== */
/* STEP 1 DECLARE THE CURSOR TO BE USED. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,R_LIMIT10);

STRCPY(ERROR_MESSAGE,FT6_CUR_DCL_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL DECLARE CF6 CURSOR FOR
SELECT DB3FC1, DB3FI1, DB3FC2, DB3FL1, DB3FR1, DB3FD1, DB3FV1
FROM DB300
WHERE DB3FI2 = :KEY_VALUE;

/* ===== */
/* STEP 2 OPEN THE CURSOR TO BE USED. */
/* ===== */

STRCPY(ERROR_MESSAGE,FT6_CUR_OPN_ERR);
EXEC SQL OPEN CF6;

/* ===== */
/* STEP 3 USE FETCH TO RETRIEVE ALL REQUIRED RECORDS. */
/* ===== */

STRCPY(ERROR_MESSAGE,FT6_CUR_FTCH_ERR);
EXEC SQL WHENEVER NOT FOUND GOTO WORK_DONE_F6;

FOR(;;)
{ /* FOR START */
EXEC SQL FETCH CF6 INTO
:DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV;

} /* FOR END */

/* ===== */
/* STEP 4 CLOSE THE CURSOR. */
/* ===== */

WORK_DONE_F6:
EXEC SQL CLOSE CF6;

/* ===== */
/* 13. RETRIEVE 10 ROWS FROM DB200. */
/* ===== */

```

```

/* ===== */
/* STEP 1 DECLARE THE CURSOR TO BE USED. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,R_LIMIT10);

STRCPY(ERROR_MESSAGE,FT2_CUR_DCL_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL DECLARE CF2 CURSOR FOR
SELECT DB2FC1, DB2FI1, DB2FC2, DB2FL1, DB2FR1, DB2FD1, DB2FV1
FROM DB200
WHERE DB2FI2 = :KEY_VALUE;

/* ===== */
/* STEP 2 OPEN THE CURSOR TO BE USED. */
/* ===== */

STRCPY(ERROR_MESSAGE,FT2_CUR_OPN_ERR);
EXEC SQL OPEN CF2;

/* ===== */
/* STEP 3 USE FETCH TO RETRIEVE ALL REQUIRED RECORDS. */
/* ===== */

STRCPY(ERROR_MESSAGE,FT2_CUR_FTCH_ERR);
EXEC SQL WHENEVER NOT FOUND GOTO WORK_DONE_F2;

FOR(;;)
{ /* FOR START */
EXEC SQL FETCH CF2 INTO
:DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV;
} /* FOR END */

/* ===== */
/* STEP 4 CLOSE THE CURSOR. */
/* ===== */

WORK_DONE_F2:
EXEC SQL CLOSE CF2;

/* ===== */
/* 14. RETRIEVE 10 ROWS FROM DB300. */
/* ===== */

/* ===== */
/* STEP 1 DECLARE THE CURSOR TO BE USED. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,R_LIMIT10);

STRCPY(ERROR_MESSAGE,FT7_CUR_DCL_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL DECLARE CF7 CURSOR FOR
SELECT DB3FC1, DB3FI1, DB3FC2, DB3FL1, DB3FR1, DB3FD1, DB3FV1
FROM DB300

```

```

WHERE DB3FI2 = :KEY_VALUE;

/* ===== */
/* STEP 2 OPEN THE CURSOR TO BE USED. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT7_CUR_OPN_ERR);
EXEC SQL OPEN CF7;

/* ===== */
/* STEP 3 USE FETCH TO RETRIEVE ALL REQUIRED RECORDS. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT7_CUR_FTCH_ERR);
EXEC SQL WHENEVER NOT FOUND GOTO WORK_DONE_F7;

FOR (;;)
{ /* FOR START */
EXEC SQL FETCH CF7 INTO
:DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV;

} /* FOR END */

/* ===== */
/* STEP 4 CLOSE THE CURSOR. */
/* ===== */

WORK_DONE_F7:
EXEC SQL CLOSE CF7;

/* ===== */
/* 15. RETRIEVE 10 ROWS FROM DB200. */
/* ===== */

/* ===== */
/* STEP 1 DECLARE THE CURSOR TO BE USED. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,R_LIMIT10);

STRCPY(ERROR_MESSAGE, FT3_CUR_DCL_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL DECLARE CF3 CURSOR FOR
SELECT DB2FC1, DB2FI1, DB2FC2, DB2FL1, DB2FR1, DB2FD1, DB2FV1
FROM DB200
WHERE DB2FI2 = :KEY_VALUE;

/* ===== */
/* STEP 2 OPEN THE CURSOR TO BE USED. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT3_CUR_OPN_ERR);
EXEC SQL OPEN CF3;

/* ===== */
/* STEP 3 USE FETCH TO RETRIEVE ALL REQUIRED RECORDS. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT3_CUR_FTCH_ERR);

```

```

EXEC SQL WHENEVER NOT FOUND GOTO WORK_DONE_F3;

FOR(;;)
{ /* FOR START */
EXEC SQL FETCH CF3 INTO
:DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV;

} /* FOR END */

/* ===== */
/* STEP 4 CLOSE THE CURSOR. */
/* ===== */

WORK_DONE_F3:
EXEC SQL CLOSE CF3;

/* ===== */
/* 16. RETRIEVE 10 ROWS FROM DB300. */
/* ===== */

/* ===== */
/* STEP 1 DECLARE THE CURSOR TO BE USED. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,R_LIMIT10);

STRCPY(ERROR_MESSAGE, FT4_CUR_DCL_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL DECLARE CF4 CURSOR FOR
SELECT DB3FC1, DB3FI1, DB3FC2, DB3FL1, DB3FR1, DB3FD1, DB3FV1
FROM DB300
WHERE DB3FI2 = :KEY_VALUE;

/* ===== */
/* STEP 2 OPEN THE CURSOR TO BE USED. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT4_CUR_OPN_ERR);
EXEC SQL OPEN CF4;

/* ===== */
/* STEP 3 USE FETCH TO RETRIEVE ALL REQUIRED RECORDS. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT4_CUR_FTCH_ERR);
EXEC SQL WHENEVER NOT FOUND GOTO WORK_DONE_F4;

FOR(;;)
{ /* FOR START */
EXEC SQL FETCH CF4 INTO
:DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV;

} /* FOR END */

/* ===== */
/* STEP 4 CLOSE THE CURSOR. */
/* ===== */

WORK_DONE_F4:

```

```
EXEC SQL CLOSE CF4;

/* ===== */
/* 17. RETRIEVE 10 ROWS FROM DB200. */
/* ===== */

/* ===== */
/* STEP 1 DECLARE THE CURSOR TO BE USED. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,R_LIMIT10);

STRCPY(ERROR_MESSAGE,FT5_CUR_DCL_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL DECLARE CF5 CURSOR FOR
SELECT DB2FC1, DB2FI1, DB2FC2, DB2FL1, DB2FR1, DB2FD1, DB2FV1
FROM DB200
WHERE DB2FI2 = :KEY_VALUE;

/* ===== */
/* STEP 2 OPEN THE CURSOR TO BE USED. */
/* ===== */

STRCPY(ERROR_MESSAGE,FT5_CUR_OPN_ERR);
EXEC SQL OPEN CF5;

/* ===== */
/* STEP 3 USE FETCH TO RETRIEVE ALL REQUIRED RECORDS. */
/* ===== */

STRCPY(ERROR_MESSAGE,FT5_CUR_FTCH_ERR);
EXEC SQL WHENEVER NOT FOUND GOTO WORK_DONE_F5;

FOR(;;)
{ /* FOR START */
EXEC SQL FETCH CF5 INTO
:DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV;
} /* FOR END */

/* ===== */
/* STEP 4 CLOSE THE CURSOR. */
/* ===== */

WORK_DONE_F5:
EXEC SQL CLOSE CF5;

/* ===== */
/* 18. RETRIEVE 10 ROWS FROM DB300. */
/* ===== */

/* ===== */
/* STEP 1 DECLARE THE CURSOR TO BE USED. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X,Y,Z,R_LIMIT10);
```

```

STRCPY(ERROR_MESSAGE, FT8_CUR_DCL_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL DECLARE CF8 CURSOR FOR
SELECT DB3FC1, DB3FI1, DB3FC2, DB3FL1, DB3FR1, DB3FD1, DB3FV1
FROM DB300
WHERE DB3FI2 = :KEY_VALUE;

/* ===== */
/* STEP 2 OPEN THE CURSOR TO BE USED. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT8_CUR_OPN_ERR);
EXEC SQL OPEN CF8;

/* ===== */
/* STEP 3 USE FETCH TO RETRIEVE ALL REQUIRED RECORDS. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT8_CUR_FTCH_ERR);
EXEC SQL WHENEVER NOT FOUND GOTO WORK_DONE_F8;

FOR(;;)
{ /* FOR START */
EXEC SQL FETCH CF8 INTO
:DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV;
} /* FOR END */

/* ===== */
/* STEP 4 CLOSE THE CURSOR. */
/* ===== */

WORK_DONE_F8:
EXEC SQL CLOSE CF8;

/* ===== */
/* 19. JOIN DB200 (10 ROWS) AND DB300 (100) RETURNS 100 ROWS. */
/* ===== */

/* ===== */
/* STEP 1 DECLARE THE CURSOR TO BE USED. */
/* ===== */

/* GET A RANDOM KEY */
X = RAND2(R_LIMIT);
Y = RAND2(R_LIMIT);
Z = RAND2(R_LIMIT);
KEY_VALUE = RANDP(X, Y, Z, R_LIMIT10);

STRCPY(ERROR_MESSAGE, FT9_CUR_DCL_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL DECLARE CF9 CURSOR
FOR SELECT
DB2FC1, DB2FI1, DB2FC2, DB3FL1, DB3FR1, DB3FD1, DB3FV1
FROM DB200, DB300
WHERE(DB2FI2 = DB3FI2)
AND(DB2FI2 = :KEY_VALUE);

/* ===== */
/* STEP 2 OPEN THE CURSOR TO BE USED. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT9_CUR_OPN_ERR);

```



```

EXEC SQL OPEN CF9;

/* ===== */
/* STEP 3 USE FETCH TO RETRIEVE ALL REQUIRED RECORDS. */
/* ===== */

STRCPY(ERROR_MESSAGE, FT9_CUR_FTCH_ERR);
EXEC SQL WHENEVER NOT FOUND GOTO WORK_DONE_F9;

FOR(;;)
{ /* FOR START */
EXEC SQL FETCH CF9 INTO
:DBFC1, :DBFI, :DBFC2, :DBFL, :DBFR, :DBFD, :DBFV;

} /* FOR END */

/* ===== */
/* STEP 4 CLOSE THE CURSOR. */
/* ===== */

WORK_DONE_F9:
EXEC SQL CLOSE CF9;

/* ===== */
/* 20. 4.1 INSERT A ROW IN DBINS AND CHECK ERROR THEN COMMIT WORK.*/
/* ===== */

STRCPY(ERROR_MESSAGE, ROW_INSRT_ERR);
EXEC SQL WHENEVER SQLERROR GOTO SQL_ERROR_ROUTINE;
EXEC SQL INSERT INTO
DBINS(DBIFC1,DBIFI1,DBIFC2,DBIFL1,DBIFR1,DBIFD1,DBIFV1,
DBIFC3,DBIFI2,DBIFC4,DBIFL2,DBIFR2,DBIFD2,DBIFV2)
VALUES (:DBFC1,:DBFI,:DBFC2,:DBFL,:DBFR,:DBFD,:DBFV,
:DBFC1,:DBFI,:DBFC2,:DBFL,:DBFR,:DBFD,:DBFV);

IN_REC++;
_CLEARSCREEN(_GCLEARSCREEN);

PRINTF(" * ***** *\n");
PRINTF(" * ROW NUMBER: %7D HAS BEEN INSERTED IN DBINS. *\n", IN_REC);
PRINTF(" * ***** *\n");

/* ***** */
/* COMMIT WORK TO DATABASES. */
/* ***** */

EXEC SQL WHENEVER SQLERROR CONTINUE; /* DON'T TRAP ERRORS. */
EXEC SQL COMMIT WORK; /* COMMIT WORK TO DASES & DONT RELEASE. */
TRANS_COUNT++; /* ADD ONE TO NUMBER OF TRANSACTIONS/ITERATION */

/* ***** */
/* GET END TIME TO CALCULATE RESPONSE TIME. */
/* ***** */

TIME(&RESP_TIME_STOP);
RESP_TIME_1 = RESP_TIME_STOP - RESP_TIME_START;
RESP_TIME_T = RESP_TIME_T + RESP_TIME_1;

} /* END OF DO LOOP BODY. */
WHILE (TIME(&G_TIME1) < END);

/* ===== */

```

```

/* FILE PREPARATION AND WRITING SECTION. */
/* ===== */

RESP_TIME      = (DOUBLE) RESP_TIME_T / (DOUBLE) TRANS_COUNT;
TRANS_PER_SEC  = (FLOAT) TRANS_COUNT / (FLOAT) RESP_TIME_T;

STPCPY(TPS_IO_REC.DBTYPE,DB_TYPE);
TPS_IO_REC.DBSIZE = ROWS;
TPS_IO_REC.NLOOPS = 1; /* NUMBER OF LOOPS.*/
TPS_IO_REC.LOOPTIME = SECONDS; /* LOOP TIME IN SECONDS. */
TPS_IO_REC.TOTTIME  = RESP_TIME_T; /* TOTAL TIME IN SECONDS. */
TPS_IO_REC.TOT_TRANS = TRANS_COUNT; /* TOT. TRANSACTIONS. */
TPS_IO_REC.AVR_TRANS = TRANS_COUNT; /* AVR. TRANSACTIONS/LOOP. */
TPS_IO_REC.TRN_PER_SEC = TRANS_PER_SEC; /* TRANSACTIONS/SEC. */
TPS_IO_REC.IT_RESP_TIME = RESP_TIME;
TIME(&TPS_IO_REC.TIME_DATE); /* TEST DATE & TIME */

IF(FWRITE(&TPS_IO_REC, sizeof(STRUCT TPS_REC_LAYOUT),1,TPS_FP) !=1)
{ /* IF START */
  PUTCHAR(BEEP); /* BEEP */
  PUTCHAR(BEEP); /* BEEP */
  PUTCHAR(BEEP); /* BEEP */
  PRINTF("# #####");
  PRINTF("# ERROR OCCURRED WHILE WRITING TO %S #\n",PATH_NAME_1);
  PRINTF("# #####");
  GETCHAR();
} /* IF END */

/* ===== */
/* NORMAL PROGRAM TERMINATION SECTION. */
/* LOG OUT AND TERMINATE PROCESSING. */
/* ===== */

EXEC SQL WHENEVER SQLERROR CONTINUE; /* DON'T TRAP ERRORS. */
EXEC SQL COMMIT WORK RELEASE; /* LOG OFF DATABASE. */
FCLOSE(TPS_FP);
PRINTF("\n\n* ===== *\n");
PRINTF("* THE PROGRAM HAS TERMINATED NORMALLY. *\n");
PRINTF("* ===== *\n");

EXIT(0);

/* ***** */
/* THE ERROR HANDLING TRAPS AND ROUTINES. */
/* ***** */

SQL_ERROR_ROUTINE:
  PUTCHAR(BEEP); /* BEEP */
  PRINTF("* %70s *\n",LINE);
  PRINTF("* DATABASE SIZE : %LU\n",ROWS);
  PRINTF("* LAST KEY VALUE IS: %D\n",KEY_VALUE);
  PRINTF("* %70s *\n",LINE);
  PRINTF("* %70s *\n", ERROR_MESSAGE);
  PRINTF("* %70s *\n", SQLCA.SQLERRM.SQLERRMC);
  PRINTF("* %70s *\n",LINE);
  PUTCHAR(BEEP); /* BEEP */
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL ROLLBACK WORK RELEASE;
  EXIT(1);

} /* PROGRAM END */

/* ===== */

```

```

/* LINE PRINTING SUBROUTINE. */
/* ===== */

PRN_REC1(QP) LONG QP;
{ /* PRN_REC1() START */
_CLEARSCREEN(_GCLEARSCREEN);
PRINTF ("TRANSACTION NUMBER: %8D\n", QP);
PRINTF ("INTEGER FIELD : %8D \n", DBFI );
PRINTF ("CHARACTER FIELD1: %8S \n", DBFC1);
PRINTF ("CHARACTER FIELD2: %30S \n", DBFC2);
PRINTF ("LONG FIELD : %8D \n", DBFL);
PRINTF ("REAL FIELD : %F \n", DBFR);
PRINTF ("DOUBLE FIELD : %F \n", DBFD);
PRINTF ("VARCHAR FIELD : %40S \n", DBFV.ARR);
} /* PRN_REC1() END */

/* ***** */
/* THIS PROCEDURE GENERATES RANDOM NUMBERS WITHIN A GIVEN RANGE. */
/* ***** */

/* ===== */
/* GENERATE PSEUDO RANDOM NUMBERS. */
/* ===== */

LONG RANDP ( X, Y, Z, LIMIT )
LONG X, Y, Z, LIMIT;
{ /* RANDOM FUNCTION START */
/* THIS RANDOM NUMBER GENERATOR IS BASED UPON NPL REPORT DITC 6/82 BY
B.A.WICHMANN AND I.D.HILL. X,Y,Z ARE GLOBAL VARIABLES TO BE GIVEN
INITIAL RANDOM INTEGER VALUES WHICH SHOULD BE LESS THAN 30000 */

FLOAT W;
LONG LW;
LONG RANDOM;

X = 171*(X % 177) - 2*(X / 177);
IF (X < 0) X = X+30269;

Y = 172*(Y % 176) - 35*(Y / 176);
IF (Y < 0) Y = Y+30307;

Z = 170*(Z % 178) - 63*(Z / 178);
IF (Z < 0) Z = Z+30323;

W = (FLOAT) X/30269 + (FLOAT) Y/30307+ (FLOAT) Z/30323;
LW = W;
RANDOM = (W - LW) * LIMIT /* MAX VALUE */ ;

RETURN (RANDOM);
} /* RANDP FUNCTION END */

/* ***** */
/* THIS PROCEDURE GENERATES RANDOM NUMBERS WITHIN A GIVEN RANGE. */
/* ***** */

#define SCALE 32768.0
EXTERN INT RAND1(); /* EXTERN KEYWORD IS OPTIONAL */
INT RAND2(LIMIT) INT LIMIT;
{
FLOAT K_VALUE;
K_VALUE = ( (FLOAT) RAND1() / SCALE + 1.0) * LIMIT / 2.0 + 1.0;
RETURN ( (INT) K_VALUE);
}

```

```
STATIC INT RANDX = 1; /* EXTERNAL STATIC FUNCTION */
INT RAND1()
{
  RANDX = (RANDX * 25173 + 13849) % 65536; /* MAGIC FORMULA */
  RETURN ( (INT) RANDX);
}

INT SRAND1(X) UNSIGNED X;
{
  RANDX = X;
}
```

APPENDIX D

APPENDIX D: Random Numbers Generators

```

/* ***** */
/* THIS PROCEDURE GENERATES RANDOM NUMBERS WITHIN A GIVEN RANGE. */
/* ***** */

/* ===== */
/* GENERATE PSEUDO RANDOM NUMBERS. */
/* ===== */

LONG RANDP ( X, Y, Z, LIMIT )
LONG X, Y, Z, LIMIT;
{ /* RANDOM FUNCTION START */
/* THIS RANDOM NUMBER GENERATOR IS BASED UPON NPL REPORT DITC 6/82 BY
B.A.WICHMANN AND I.D.HILL. X,Y,Z ARE GLOBAL VARIABLES TO BE GIVEN
INITIAL RANDOM INTEGER VALUES WHICH SHOULD BE LESS THAN 30000 */

FLOAT W;
LONG LW;
LONG RANDOM;

X = 171*(X % 177) - 2*(X / 177);
IF (X < 0) X = X+30269;

Y = 172*(Y % 176) - 35*(Y / 176);
IF (Y < 0) Y = Y+30307;

Z = 170*(Z % 178) - 63*(Z / 178);
IF (Z < 0) Z = Z+30323;

W = (FLOAT) X/30269 + (FLOAT) Y/30307+ (FLOAT) Z/30323;
LW = W;
RANDOM = (W - LW) * LIMIT /* MAX VALUE */ ;

RETURN (RANDOM);
} /* RANDP FUNCTION END */

/* ***** */
/* THIS PROCEDURE GENERATES RANDOM NUMBERS WITHIN A GIVEN RANGE. */
/* ***** */

#define SCALE 32768.0
EXTERN INT RAND1(); /* EXTERN KEYWORD IS OPTIONAL */
INT RAND2(LIMIT) INT LIMIT;
{
FLOAT K_VALUE;
K_VALUE = ( (FLOAT) RAND1() / SCALE + 1.0) * LIMIT / 2.0 + 1.0;
RETURN ( (INT) K_VALUE);
}

STATIC INT RANDX = 1; /* EXTERNAL STATIC FUNCTION */
INT RAND1()
{
RANDX = (RANDX * 25173 + 13849) % 65536; /* MAGIC FORMULA */
RETURN ( (INT) RANDX);
}

INT SRAND1(X) UNSIGNED X;
{
RANDX = X;
}

```

APPENDIX E

APPENDIX E: Analysis of Variance

```

/*
*****
*/
/* A PROGRAM TO CALCULATE ONE AND TWO WAY ANALYSIS OF VARIANCE (ANOVA). */
/*
*****
*/

#include <STDIO.H>
#include <DOS.H>
#include <BIOS.H>
#include <MATH.H>
#include <STRING.H>
#include <GRAPH.H>
#include <TIME.H>
#define BEEP '\A'
#define MAXPATH 64
#define MAXLINE 80

/* ===== */
/* THE MAIN PROGRAM BODY */
/* ===== */

INT MAIN(VOID)
{ /* MAIN PROGRAM START */

INT I=0, /* LOOPS INDICES */
J=0;
INT R,C,RC; /* NUMBER OF ROWS AND COLUMNS */
DOUBLE X[50][50],XI[50],XJ[50]; /* ANOVA ARRAYS */
DOUBLE SUM = 0, /* SUM OF ALL VALUES */
SUMJ = 0, /* SUM OF ROWS */
XBAR = 0, /* SUM OF COLUMNS */
ZXIJ = 0, /* SUM OF ALL ELEMENTS */
XIJVAR = 0, /* ALL ROWS VARIANCE */
ALLVAR = 0, /* ALL ROWS VARIANCE */
ZXI = 0, /* SUM OF ROWS */
CZXI = 0, /* SUM OF ROWS * NO. OF ROWS */
MSROWS = 0, /* ROWS MEAN OF SQUARES */
ZXJ = 0, /* SUM OF COLUMNS */
RZXJ = 0, /* SUM OF COLUMNS * NO. OF COLUMNS */
MSCOLMS = 0, /* COLUMNS MEAN OF SQUARES */
RES = 0, /* RESIDUAL. */
MSRES = 0, /* RESIDUALMEAN OF SQUARES */
FROWS = 0, /* F FOR ROWS */
FCOLMS = 0; /* F FOR COLUMNS */

FILE *FP;

CHAR PATH_NAME_1[MAXPATH];
CHAR LINE[MAXLINE];

/* ===== */
/* OPEN FILE AND CHECK IT */
/* ===== */

_CLEARSCREEN(_GCLEARSCREEN);

PRINTF("PLEASE ENTER DATA FILE NAME: ");
SCANF("%s", PATH_NAME_1);

```



```

IF ( (FP=FOPEN(PATH_NAME_1,"RB")) == NULL)
{ /* IF START */
PRINTF(" * ##### *\n");
PRINTF(" * CANNOT OPEN THE FILE, END OF PROGRAM ENCOUNTERED *\n");
PRINTF(" * FILE NAME IS: %S\n",PATH_NAME_1);
PRINTF(" * ##### *\n");
EXIT(1);
} /* IF END */

/* ===== */
/* READ NUMBER OF ROWS AND COLUMNS */
/* ===== */

IF ( (FSCANF (FP, "%D/%D", &R, &C)) !=1)
{ /* IF FSCANF (FP, "%D/%D", &R, &C) START */
PRINTF(" * ##### *\n");
PRINTF(" * AN ERROR OCCURRED DURING READING %S *\n",PATH_NAME_1);
PRINTF(" * ##### *\n");
FCLOSE(FP);
EXIT(1);
} /* IF FSCANF (FP, &R, &C) END */

IF ( (R > 50) || (C > 50) )
{ /* IF (R > 50) || (C > 50) START */
PRINTF(" * ##### *\n");
PRINTF(" * AN ERROR OCCURRED C OR R IS GREATER THAN 50. *\n");
PRINTF(" * ##### *\n");
FCLOSE(FP);
EXIT(1);
} /* IF FSCANF (FP, &R, &C) END */

/* ===== */
/* THIS ROUTINE IS TO CLEAN ARRAYS */
/* ===== */

FOR (I=0; I<R; I++)
{ /* FOR I START */
XI[I] = 0;
FOR (J=0; J<C; J++)
{ /* FOR J START */
XJ[J] = 0;
X[I][J] = 0;
} /* FOR J END */
} /* FOR I END */

/* ===== */
/* THIS ROUTINE READS VALUES TO X(I,J) AND SUMS ALL VALUES */
/* ===== */

RC = R*C;
SUM = 0;
FOR (I=0; I<R; I++)
{ /* FOR I START */
FOR (J=0; J<C; J++)
{ /* FOR J START */
IF ( (FSCANF (FP, "%F", &X[I][J])) !=1) ;
} /* IF FSCANF (FP, &R, &C) START */
PRINTF(" * ##### *\n");
PRINTF(" * AN ERROR OCCURRED DURING READING %S *\n",PATH_NAME_1);
PRINTF(" * ##### *\n");
FCLOSE(FP);
EXIT(1);
} /* IF FSCANF (FP, &R, &C) END */

```

```

SUM += X[I][J];
} /* FOR J END */
} /* FOR I END */
XBAR = SUM / RC;

/* ===== */
/* THIS ROUTINE CALCULATES SUM(X(I,J)-XBAR)^2 */
/* ===== */

ZXIJ = 0;
FOR (I=0; I<R; I++)
{ /* FOR I START */
FOR (J=0; J<C; J++)
{ /* FOR J START */
ZXIJ += (POW((X[I][J] - XBAR),2.0));
} /* FOR J END */
} /* FOR I END */
XIJVAR = ZXIJ / (RC-1);
ALLVAR = ZXIJ / (RC-1);

/* ===== */
/* THIS ROUTINE CALCULATES C*SUM(XBAR(I)-XBAR)^2 */
/* ===== */

ZXIJ = 0;
FOR (I=0; I<R; I++)
{ /* FOR I START */
FOR (J=0; J<C; J++)
{ /* FOR J START */
XI[I] += X[I][J];
} /* FOR J END */
} /* FOR I END */

FOR (I=0; I<R; I++)
{ /* FOR I START */
XI[I] /= C;
} /* FOR I END */

ZXI = 0;
FOR (I=0; I<R; I++)
{ /* FOR I START */
ZXI += (POW((XI[I] - XBAR),2.0));
} /* FOR I END */
CZXI = ZXI * C;
MSROWS = CZXI / (R-1);

/* ===== */
/* THIS ROUTINE CALCULATES R*SUM(XBAR(J)-XBAR)^2 */
/* ===== */

FOR (J=0; J<C; J++)
{ /* FOR J START */
FOR (I=0; I<R; I++)
{ /* FOR I START */
XJ[J] += X[I][J];
} /* FOR I END */
} /* FOR J END */

FOR (J=0; J<C; J++)
{ /* FOR J START */
XJ[J] /= C;
} /* FOR J END */

```

```

ZXJ = 0;
FOR (J=0; J<C; J++)
{ /* FOR I START */
ZXJ += (POW((XJ[J] - XBAR),2.0));
} /* FOR I END */
RZXJ = ZXJ * R;
MSCOLMS = RZXJ / (C-1);

/* ===== */
/* THIS ROUTINE CALCULATES SUM(X(I,J) - XBAR(I) - XBAR(J)+XBAR)^2 */
/* ===== */

RES = 0;
FOR (I=0; I<R; I++)
{ /* FOR I START */
FOR (J=0; J<C; J++)
{ /* FOR J START */
RES += (POW((X[I][J] - XI[I] - XJ[J] + XBAR),2.0));
} /* FOR J END */
} /* FOR I END */

MSRES = RES / ((C-1)*(R-1));

/* ===== */
/* CALCULATE F VALUES. */
/* ===== */

FROWS = MSROWS / MSRES;
FCOLMS = MSCOLMS / MSRES;

/* ===== */
/* PRINT RESULTS SECTION. */
/* ===== */

_CLEARSCREEN(_GCLEARSCREEN);

PRINTF("MEANS OF ROWS\n");
PRINTF("=====\n\n");
FOR (I=0; I<R; I++)
{ /* FOR I START */
PRINTF("MEAN ROW %D = %8.6F\t",I,XI[I]);
} /* FOR I END */

PRINTF("MEANS OF COLUMNS\n");
PRINTF("=====\n\n");
FOR (J=0; J<C; J++)
{ /* FOR J START */
PRINTF("MEAN COLUMN %D = %8.6F\t",J,XJ[J]);
} /* FOR J END */

PRINTF(" * ===== *\n");
PRINTF("MEAN SQUARE OF ROWS : %8.6F\n",MSROWS);
PRINTF("MEAN SQUARE OF COLUMNS : %8.6F\n",MSCOLMS);
PRINTF("MEAN SQUARE OF RESIDUAL: %8.6F\n",MSRES);
PRINTF(" * ----- *\n");
PRINTF("F ROWS : %8.6F\n",FROWS);
PRINTF("F COLUMNS : %8.6F\n",FCOLMS);
PRINTF(" * ===== *\n");

} /* MAIN PROGRAM END */

```

APPENDIX F

APPENDIX F: The City Benchmark Operation Book

The CITY benchmark is written in C high level language imbedded with SQL commands. That allows the program to be run on several platforms provided that platform supports SQL pre-compiler and C compiler. To run the program the implementor should go through the following steps:

1. Pre-compile the program using the existing pre-compilation tool. In a PC environment:

Example:

```
c:> precomp
```

2. Compile the program using the available C compiler.
3. For some systems, such as SUN SPARC, steps 1 and 2 are done through one integrated procedure that is called MAKEFILE.
4. Run the CITY benchmark tables creation program: CITYCRT.

Example:

```
c:>CITYCRT
```

5. Run the CITY benchmark tables load program: CITYLOAD.

Example:

```
c:>CITYLOAD
```

The program will display the current size of existing tables then ask about the number of rows to be added to the existing size.

Example:

```
Pleas enter number of rows to be added to CITY tables: 5000
```

6. Run the CITY benchmark : CITY.

Example:

c:>CITY

The program will ask about the time required for each test loop in seconds and wait for a value, the user should enter the number of seconds. The recommended number of seconds are 900 seconds per loop.

Example:

Pleas enter test loop time in seconds: 900

7. The program will loop run until completion. When the program finishes normally a message is printed saying:

"The program has terminated normally"

If the program does not finish normally, a message will be printed saying:

"Abnormal program termination"

In association with another message that explains the error.

8. The CITY benchmark produces its results in a file called "CITY.RES". To print those results run a program called CITYRES. The program script is presented in appendix G.

Example:

c:> CITYRES

The program will ask for the results file name:

Example:

Please enter file name: CITY.RES

The program will print the results in the following form:

Database name: ;
Database size:
Number of loops:
Loop time in seconds:
Total time in seconds:
Total number of transactions:
Average number transactions per second:
Transaction response time:

APPENDIX G

APPENDIX G: Printing The City Benchmark Results

```
/* ***** */
/* WRITING CHARACTERS TO A FILE, GET FILE NAME FROM USER */
/* THIS PROGRAM WRITES A RECORD FROM A STRUCTURE */
/* ***** */

#include <STDIO.H>
#include <DOS.H>
#include <BIOS.H>
#include <MATH.H>
#include <STRING.H>
#include <GRAPH.H>
#include <TIME.H>

#define BEEP '\A'
#define MAXPATH 64
#define MAXLINE 80

/* ===== */
/* DECLARE STRUCTURE TAG */
/* ===== */

STRUCT REC_LAYOUT
{ /* STRUCTURE BEGINING */
CHAR DBTYPE[15]; /* DATABASE NAME. */
LONG DBSIZE; /* DATABASE SIZE. */
INT NLOOPS; /* NUMBER OF LOOPS. */
INT LOOPTIME; /* LOOP TIME IN SECONDS. */
INT TOTTIME; /* TOTAL TIME IN SECONDS. */
LONG TOT_TRANS; /* TOT. NUMBER OF TRANSACTIONS. */
LONG AVR_TRANS; /* AVR. TRANSACTIONS PER LOOP. */
DOUBLE TRN_PER_SEC; /* NUMBER OF TRANSACTIONS/SEC. */
DOUBLE IT_RESP_TIME; /* ITERATION RESPONSE TIME. */
TIME_T TIME_DATE; /* TIME AND DATE OF TEST. */
}; /* STUCTURE END */

/* ===== */
/* DECLARE RECORD STRUCTURE */
/* ===== */

STRUCT REC_LAYOUT IO_REC;

/* ===== */
/* THE MAIN PROGRAM BODY */
/* ===== */

INT MAIN(VOID)
{ /* MAIN PROGRAM START */

INT I=0;
INT J=0;
INT K=0;

INT N_BEEP=0;
DOUBLE TOT_TPS = 0; /* TOTAL NUMBER OF TRANSACTIONS/SEC. */
DOUBLE AVR_TPS = 0; /* AVERAGE NUMBER OF TRANSACTIONS/SEC. */

FILE *FP;

CHAR PATH_NAME_1[MAXPATH];
```


APPENDIX G: Printing The City Benchmark Results.

```
CHAR LINE[MAXLINE];

/* ===== */
/* OPEN FILE AND CHECK IT */
/* ===== */

_CLEARSCREEN(_GCLEARSCREEN);

PRINTF("PLEASE LOG FILE NAME: ");
SCANF("%s", PATH_NAME_1);

IF( (FP=FOPEN(PATH_NAME_1, "RB")) == NULL)
{ /* IF START */
    PUTCH(BEEP); /* BEEP */
    PUTCH(BEEP); /* BEEP */
    PRINTF(" * ##### * \n");
    PRINTF(" * CANNOT OPEN THE FILE, END OF PROGRAM ENCOUNTERED * \n");
    PRINTF(" * FILE NAME IS: %s \n", PATH_NAME_1);
    PRINTF(" * ##### * \n");
    GETCHAR();
} /* IF END */

/* ===== */
/* READ FILE RECORD AND CHECK IT. */
/* ===== */

DO
{ /* START OF DO LOOP BODY. */

    IF(FREAD(&IO_REC, sizeof(STRUCT REC_LAYOUT), 1, FP) != 1)
    { /* IF FREAD START */
        AVR_TPS = TOT_TPS / I;
        PUTCH(BEEP); /* BEEP */
        PRINTF("\n * TEST TIME AND DATE : %26s \n", CTIME(&IO_REC.TIME_DATE));
        PRINTF(" * ##### * \n");
        PRINTF(" * %d RECORDS HAVE BEEN READ FROM %s * \n", I, PATH_NAME_1);
        PRINTF(" * AVERAGE NUMBER OF TPS IS: %3.6f * \n",
            AVR_TPS);
        PRINTF(" * ##### * \n");
        FCLOSE(FP);
        RETURN(0);
    } /* IF FREAD END */

    ++I;
    TOT_TPS = TOT_TPS + IO_REC.TRN_PER_SEC;
    PRINTF("RECORD NO          : %8d \n", I);
    PRINTF("DB TYPE                  : %s \n", IO_REC.DBTYPE);
    PRINTF("DB SIZE                   : %8d \n", IO_REC.DBSIZE);
    PRINTF("No. OF LOOPS              : %8d \n", IO_REC.NLOOPS);
    PRINTF("LOOP TIME IN SECONDS      : %8d \n", IO_REC.LOOPTIME);
    PRINTF("TOTAL TIME IN SECONDS     : %8d \n", IO_REC.TOTTIME);
    PRINTF("Tot. No. OF TRANSC.       : %8d \n", IO_REC.TOT_TRANS);
    PRINTF("Avr. No. OF TRANSC.       : %8d \n", IO_REC.AVR_TRANS);
    PRINTF("TRANSACTIONS/SECOND       : %4.6f \n", IO_REC.TRN_PER_SEC);
    PRINTF("TRANSACTIONS RESPONSE     : %4.6f \n", IO_REC.IT_RESP_TIME);
    PRINTF(" * ----- * \n");

} /* DO END */
WHILE (!FEOF(FP));

} /* MAIN PROGRAM END */
```

APPENDIX H

M. W. Youssef
School of Informatics
Department of Business Computing
The City University
London
EC1U 0HB

Our Ref -
Our Ref -
Date 27 September, 1993
Subject **CITY BENCHMARK**

Dear Mr Youssef.

As you know, Abbey National makes use of several hardware platforms, from PC networks, to large mainframe environments, including Unisys and IBM, and database machines, such as Teradata model 3 and 4. This hardware infrastructure suggests that co-existence and integration of the various software environments is of utmost importance to us. As part of our Information Technology Strategy, we have selected Oracle as our strategic database management system, but we also have various legacy applications.

During the last quarter of this year, we will be starting an evaluation of Massively Parallel Processing Platforms, using Oracle V7.1. This will serve for the assessment of new platforms as well as their comparison with our existing systems. One of the aspects of this study will be to test the selected platforms with high volume transaction processing workloads. We do not believe that any of the existing benchmarks, such as the TPC suite, can accurately model our environment. We are confident that the CITY benchmark can address our requirement in this area, including platforms such as Teradata DBC systems and NCR 3600, which we understand you have already benchmarked.

We will be contacting you in the short term to define a suitable way ahead for the benchmarking strategy.

Yours sincerely,



Pete Lazard
Head of Technology Strategy & Research

cc: Neira Benchabane, Senior Consultant



NCR Limited
Large Systems Division
Alwyn House, 31 Windsor Street
Chertsey, Surrey KT16 8AT
Tel: 0932 567777
Fax: 0932 564991

DT/JB

6th October 1993

To whom it may concern

Some time ago Mr Youssef ran some benchmarks at our facility at Chertsey. The benchmark appeared to test a range of SQL functions producing a range of answer sets. These were run against a Teradata DBC1012 Model 3, a DBC1012 Model 4 and a NCR 3600. The performance ratios of the various models produced with Mr Youssef's benchmark were consistent with benchmarks that have been done at our plant in the United States.

David Tomlinson
Systems Administrator