

A Comparative Study of Indexing Techniques for Relational Database Management Systems

By

Huda Ayesha Mashaan Alrashidi

A Thesis

**Submitted in Partial Fulfillment of the
Requirements for the Master Degree
in Computer Information Systems**

Supervisor

Dr. Hazim A. Farhan

**Department of Computer Information Systems
Faculty of Information Technology
Middle East University**

Amman - Jordan

March, 2011

إقرار تفويض

أنا هدى عايش مشعان الرشيدى , أفوض جامعة الشرق الأوسط بتزويد نسخ من رسالتي للمكتبات أو الهيئات أو الأفراد عند طلبها.



التوقيع:-

٢٠١١ / ٣ / ١٩

التاريخ:-

Authorization Statement

I , Huda Ayesh Mashaan AlRashidi, authorize Middle East University to supply hardcopies and electronic copies of my thesis to libraries, establishments, or bodies and institutions concerned with research and scientific studies upon request, according to the university regulations.

Signature:- 

Date:- 21 / 3 / 2011

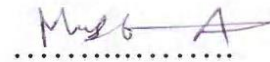
Middle East University Examination Committee Decision

This is to certify that the Thesis entitled "A Comparative Study of Indexing Techniques for Relational Database Management Systems " was successfully defended and approved on 20 March 2011.

Examination Committee Members

Signature

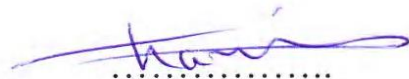
Prof. Musbah M. Aqel



Department of Computer Information Systems

Middle East University (MEU)

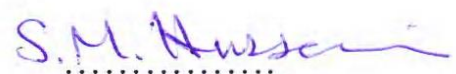
Supervisor: Dr. Hazim A. Farhan



Department of Computer Science

Middle East University (MEU)

Dr. Shakir M. Al-Farraji

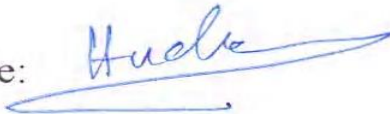


Faculty of Information Technology

University of Petra

DECLARATION

I do declare that the present research work has been carried out by me under the supervision of Dr. Hazim A. Farhan, and this work has not been submitted elsewhere for any other degree, fellowship or any other similar title.

Signature: 

Date: 21 / 3 / 2011

DEDICATION

I dedicate this thesis to my husband, who supported me, and my family, my friends; they were the light in my path and without them nothing of this would have been possible.

ACKNOWLEDGEMENTS

My foremost thank goes to my supervisor Dr Hazim Farhan. Without him, this thesis would not have been possible. I appreciate his vast knowledge his insights, suggestions and guidance that helped to shape my research skills.

I would like to thank Prof. Musbah Aqel and Dr. Hussein Owaied for their advice and help when I started my graduate studies. I want to express my gratitude to all those who gave me the possibility to complete this thesis.

I would like to extend special thanks to my husband Mubarek AlRashidi and my mother for their constant unconditional love and encouragement. It has always been my dream to be able to return their favors for everything that they have given to me. I also thank my sisters, for their genuine love and support. I also would like to thank Eng Imad Saleh for his technical support.

List of Tables

Table 1.1: Summary of indexing types in Oracle and MS SQL Server.....	14
Table 3.1 100k on Oracle.....	39
Table 3.2 1000k on Oracle.....	41
Table 3.3 5000k on Oracle.....	44
Table 3.4 MS SQL Server on 100K.....	46
Table 3.5: MS-SQL Server on 1000K.....	48
Table 3.6: MS-SQL Server on 5000K.....	50
Table 3.7-A: A summary of the best indexing techniques with different data sizes On Oracle.....	52
Table 3.7-B: A summary of the best indexing techniques with different data sizes on MS SQL Server.....	52
Table 3.8: Select Performance using Composite Key in Oracle 1000k.....	57
Table 3.9: Select Performance using Composite Key in MS SQL Server 1000K	58
Table 3.10: Insert on MS SQL Server and Oracle with 1000K.....	60
Table 3.11: Update on MS SQL Server and Oracle with 1000K.....	61
Table 3.12: Delete on MS SQL Server and Oracle with 1000K.....	63
Table 3.13: Select on MS SQL Server and Oracle with 1000K.....	64
Table 3.14: Insert on MS SQL Server and Oracle with 1000K.....	66
Table 3.15: Update on MS SQL Server and Oracle with 1000K.....	67
Table 3.16: Delete on MS SQL Server and Oracle with 1000K.....	69

Table 3.17: Select on MS SQL Server and Oracle with 1000K.....	70
Table 3.18: Insert on MS SQL Server and Oracle with 1000K.....	71
Table 3.19: Update on MS SQL Server and Oracle with 1000K.....	72
Table 3.20: Delete on MS SQL Server and Oracle with 1000K	73
Table 3.21: Possible Rules.....	77
Table 3.22: Actions for each rule.....	78
Table 3.23: Final simplified table of rules and actions on ORACLE environment	78
Table 3.24: Final simplified table of rules and actions on MS-SQL Server environment....	79

.

List of Figures

Figure 1.1 Root node and branch nodes of an index.....	3
Figure 1.2 A search tree showing branch nodes and leaf nodes.....	4
Figure 1.3 Index Levels.....	5
Figure 1.4 B-tree Clustered Index.....	6
Figure 1.5 Non-Clustered index on a table without a Clustered index.....	8
Figure 1.6 Non-clustered indexes on a table with a clustered index.....	9
Figure 1.7 An Oracle B-tree index	10
Figure 1.8 MS-SQL Server B-tree index.....	11
Figure 2.1 Flowchart of methodology procedure on Oracle platform.....	32
Figure2.2 Flowchart of methodology procedure on MS SQL Server platform.....	33
Figure 3.1: Full scan SELECT evaluation time for different indexing with 100K on Oracle.....	38
Figure 3.2: Full scan SELECT executions time for different indexing with 1000K on Oracle.....	40
Figure 3.3: Full scan SELECT executions time for different indexes with 5000K on Oracle 10g.....	43
Figure 3.4: Query execution plan for SELECT statement on table with clustered index.....	45

Figure 3.5: Query execution plan for SELECT operation on table with non-clustered index in MS SQL Server.....	49
Figure 3.6: Comparison between the response times (ms) of B-tree Oracle and MS SQL Server on 100K.....	53
Figure 3.7: Comparison between the response times (ms) of B-tree over Oracle and MS SQL Server on 1000K.....	53
Figure 3.8: Comparison between the response times (ms) of B-tree over Oracle and MS SQL Server on 5000K.....	54
Figure 3.9: Execution result after INSERT one row using B-tree index with 100K.....	55
Figure 3.10: Execution result after INSERT one row using B-tree index with 1000K.....	55
Figure 3.11: Execution result after INSERT one row using B-tree index with 5000K.....	56

Table of Contents

List of Tables	VII
List of Figures	IX
Abstract in English	XVI
Abstract in Arabic	XVIII
Chapter 1: - Introduction.	
1.1 Introduction	1
1.2 Database indexing	2
1.2.1 Indexing techniques for relational databases.....	2
1.2.2 Index Process	3
1.2.3 Types of Indexes.....	5
1.2.3.1 Clustered Indexes	5
1.2.3.2 Non-Clustered Indexes.....	7
1.3 Indexing Examples.....	9
1.3.1 Oracle Indexing.....	11
1.3.2 MS-SQL Server Indexing.....	13
1.4 The Current Indexing Techniques.....	14
1.5 Advantages and Disadvantages of Indexes.....	15
1.6 The Problem Statement.....	16
1.7 Significance of the Study and Motivations.....	17

1.8 Thesis Contributions	17
1.9 Methodology.....	18
1.10 Limitations.....	19
1.11 Literature Review and Related Work.....	19
1.11.1 Literature Review.....	20
1.11.2 Related Work.....	23
1.12 Thesis Organization.....	25

Chapter 2: - The Methodology and Experimental Test

2.1 Introduction.....	26
2.2 Methodology Contributions.....	27
2.3 Criteria for Comparisons.....	28
2.4 Methodology Assumptions.....	28
2.5 Methodology Description.....	29
2.6 Technical Test.....	29
2.7 Test H/W & S/W (Test Environment).....	30
2.8 Test Procedure.....	30

Chapter 3: - Experiments and Test Results

3.1 Introduction.....	34
3.2 Test: SELECT Performance.....	35
3.2.1 Test 1: Oracle 10g and 100K.....	35
3.2.1.1 Test 1-a: Single-row SELECT performance with single row.....	35

3.2.1.2 Test 1-b: Range SELECT performance.....	35
3.2.1.3 Test 1-c: Full scan SELECT performance.....	37
3.2.1.4 Test 1-d: Single-row SELECT performance with non-row.....	38
3.2.2 Test 2: Oracle 10g and 1000K.....	40
3.2.2.1 Test 2-a: Single-row SELECT performance with single row.....	40
3.2.2.2 Test 2-b: Full scan SELECT performance.....	40
3.2.2.3 Test 2-c: Single-row SELECT performance with non-row.....	41
3.2.3 Test 3: Oracle 10g and 5000K.....	42
3.2.3.1 Test 3-a: Single-row SELECT performance with single row.....	42
3.2.3.2 Test 3-b: Range SELECT performance.....	42
3.2.3.3 Test 3-c: Full scan SELECT performance.....	42
3.2.3.4 Test 3-d: Single-row SELECT performance with non-row.....	43
3.2.4 Test 4: MS SQL Server and 100K.....	45
3.2.4.1 Test 4-a: Single-row SELECT performance with single row	45
3.2.4.2 Test 4-b: Full scan SELECT performance.....	46
3.2.4.3 Test 4-c: Single-row SELECT performance with non-row.....	46
3.2.5 Test 5: MS SQL Server and 1000K.....	47
3.2.5.1 Test 5-a: Single-row SELECT performance with single row	47
3.2.5.2 Test 5-b: Full scan SELECT performance.....	47
3.2.5.3 Test 5-c: Single-row SELECT performance with non-row.....	47
3.2.6 Test 6: MS SQL Server and 5000K.....	48

3.2.6.1 Test 6-a: Single-row SELECT performance with single row.....	48
3.2.6.2 Test 6-b: Full scan SELECT performance.....	49
3.2.6.3 Test 6-c: Single-row SELECT performance with non-row.....	49
3.3 Recommendations and Further Analysis.....	50
3.4 Test: INSERT Performance.....	54
3.5 Test: Select Performance using Composite Key in Oracle.....	57
3.6 Test: Select Performance using Composite Key in MS SQL Server.....	58
3.7 Test: Insert Performance using Composite Key in Oracle and MS SQL Server with 1000K.....	59
3.8 Test: Update Performance using Composite Key in Oracle and MS SQL Server with 1000K.....	61
3.9 Test: Delete Performance using Composite Key in Oracle and MS SQL Server with 1000K.....	62
3.10 Test: Select Performance using Composite Key in Oracle and MS SQL Server with 100K.....	64
3.11 Test: Insert Performance using Composite Key in Oracle and MS SQL Server with 100K.....	65
3.12 Test: Update Performance using Composite Key in Oracle and MS SQL Server with 100K.....	67
3.13 Test: Delete Performance using Composite Key in Oracle and MS SQL Server with 100K.....	68
3.14 Test: Select Performance using Composite Key in Oracle and MS SQL Server with 5000K.....	70
3.15 Test: Insert Performance using Composite Key in Oracle and MS SQL Server with 5000K.....	71

3.16 Test: Update Performance using Composite Key in Oracle and MS SQL Server with 5000K.....	72
3.17 Test: Delete Performance using Composite Key in Oracle and MS SQL Server with 5000K.....	73
3.18 Guidelines to Select Indexing Techniques	75
3.18.1 Using Decision Table to Select the Indexing for the DBAs:.....	75
3.18.1.1 Oracle Environment.....	76
3.18.1.2 MS-SQL Server Environment.....	78
3.18.2 Descriptive Guidelines and Advices for the DBAs.....	79
3.19 Observations and Recommendations.....	82

Chapter 4: - Conclusions and Future Work

4.1 Conclusions	84
4.2 Future Work.....	86
References.....	87
Glossary.....	91

Abstract

A Comparative Study of Indexing Techniques for Relational Database Management Systems

By

Huda Ayesha Mashaan AlRashidi

Indexing Represents the essential importance in the databases of all kinds and forms of organization is a method of knowledge and access to different sources of information. Indexes in general are data-structures that were created to speed up the search process and access to data and reduce the number of input and output I / O data, as well as free up system resources for various computer applications.

On the other hand, there is very little published research which presents to comparisons of methodology between the indexing techniques and through the extensive research and deep study shows that there is weakness or lack in previously published research which, did not take adequate criteria that would make clear the comparison process, as well as the absence of a clear methodology and database developers or administrator can follow. In this thesis, established a guidelines in order to advise the database developers to select the best suitable indexing technique (such as B-tree index, organization index, reverse index, clustered index , non-clustered and Bitmap index) for the database tables over two different platforms: Oracle and MS SQL Server with different data sizes starting with 100K, 1000K and ending with 5000K .

There are two directions to build the research methodology in this thesis: mathematical and theoretical, empirical test that relies on the experiments and simulation that are conducted on the same technical environment. In this research, the experimental test is more suitable to the problem statement. Furthermore, we have identified the factors that

need to be considered when database administrator (DBA) or developer wants to establish a proper index on database.

To evaluate the efficiency of indexing technique, we have implemented six indexes (B-tree, reverse, organization, clustered, non-clustered and bitmap) on the Oracle and MS SQL Server. We have conducted several experiments on large databases and recorded the overall performance, CPU consumptions, and I/O cost. Thus, the results obtained are based on criteria of the proposed methodology for selecting the best suitable indexing technique. The DBA then is able to use the guidelines in establishing and retrieving the information from the databases through the indexes.

الملخص

دراسة مقارنة لطرق الفهرسة لنظم إدارة قواعد البيانات العلائقية

هدى عايش مشعان الرشيدى

تمثل الفهرسة أهمية ضرورية في قواعد البيانات بجميع أنواعها وأشكالها فهي طريقة تنظيم للمعرفة والوصول إلى مصادر المعلومات المختلفة . فالفهارس بشكل عام هي عبارة عن هياكل للبيانات التي تم إنشاؤها لتسريع عملية البحث و الوصول إلى البيانات و تخفيض عدد عمليات الإدخال والإخراج I/O للبيانات، وكذلك تحرير موارد النظام للتطبيقات الحاسوبية المختلفة.

من ناحية أخرى، يوجد هناك القليل من الأبحاث المنشورة التي تناولت وتطرق إلى المقارنات المنهجية بين طرق الفهرسة الموجودة ومن خلال البحث والدراسة العميقة تبين انه يوجد ضعف أو عدم دقة في الأبحاث المنشورة سابقا حيث أنها لم تأخذ المعايير الكافية التي من شأنها أن تجعل من عملية المقارنة واضحة ودقيقة وكذلك عدم وجود منهجية واضحة يستطيع مطوري قواعد البيانات إتباعها .

لذا سحاول في هذه الأطروحة أن نقوم بوضع المبادئ المنهجية لتقديم النصح والإرشاد لمطوري قواعد البيانات لكي يقوموا باختيار أفضل طرق الفهرسة وأكثرها ملائمة (مثل B-tree index ، Organization index ،reverse index ،clustered index ،Bitmap index) لجداول قواعد البيانات من خلال نظامين لإدارة قواعد البيانات: أوراكل ومايكروسوفت سيرفر بأحجام بيانات مختلفة تبدأ بـ 100 كيلوبايت، 1000 كيلوبايت وتنتهي بـ 5000 كيلوبايت.

المنهجية المقترحة في هذه الأطروحة تحتوي على مقارنتين: مقارنة تم الحصول عليها من البحوث المنشورة سابقا والاختبارات التجريبية التي تم إجرائها على نفس البيئة التقنية. فضلا عن ذلك، لقد قمنا بتحديد العوامل التي يجب أن تأخذ بعين الاعتبار عندما يريد مشرفي قواعد البيانات (DBA) بناء فهرسة مناسبة لقواعد البيانات.

وقد تم عمل تجارب وتطبيقها على ست طرق فهرسة (B-tree index ، Organization ،reverse ،clustered ،Bitmap ،Non-clustered) على أوراكل ومايكروسوفت سيرفر لإثبات فعالية طرق الفهرسة.

حيث تم إجراء العديد من التجارب على قواعد بيانات كبيرة ومتوسطة وصغيرة ، وقد قمنا أيضا بتسجيل الأداء الكلي وتسجيل الوقت المستغرق في وحدة المعالجة المركزية (CPU) وكذلك تكاليف عمليات الإدخال والإخراج I/O. ،وبالتالي استندت النتائج التي تم الحصول عليها على المعايير المنهجية المقترحة لاختيار أفضل تقنيات الفهرسة المناسبة. ومن هنا أصبح لدى مشرفي قواعد البيانات (DBA) القدرة على استخدام المبادئ الإرشادية في بناء واسترجاع المعلومات من قواعد البيانات من خلال طرق الفهرسة.

Chapter 1

Introduction

1.1 Introduction

Relational Database Management Systems (RDBMS) maintain a collection of huge data files to provide fast and efficient methods in order to access and modify data which is necessary (Martin et al., 1992). Therefore, RDBMS have supported indexing techniques in order to access the data efficiently in static and dynamic manners.

Databases have been in use since the earliest days of computing revolution. In 1960's, it started the development of DBMS that supports manual navigation of a linked data set which was formed into a large network. In addition, DBMS provides boolean queries that required the programmers to go through the entire database and collect the matching records.

In 1970's, Codd outlined a new approach to database construction that was based on a Relational Model of Data for Large Shared Data Banks. He described a new system for storing and working with large databases. Instead of records being stored in some sort of linked list of free-form records as in Codasyl, Codd's idea was to use a "table" of fixed-length records. A linked-list system was inefficient when storing "sparse index" databases where some of the data for any one record could be left empty. The relational model splits the data into a series of normalized tables.

Nowadays, databases have matured and required new technological advancements in order to raise its overall performance, and sustain heavy loads of data. Query processing and optimization

have always been one of the most critical components of database technology. This component deals with efficient and effective processing of user queries against a database. The purpose of the query processing and optimization is to find user-defined data from large database effectively and with an acceptable accuracy (Clement et al., 1997). This sort of optimization is performed mostly by utilizing indexes to facilitate the quick access to user defined queries.

1.2 Database indexing

An index is an identifier, attribute, keyword and their conjunctions using conditional statements. The index is used to retrieve the multimedia information (Ponce , 2002). In databases, indexes allow to quickly identify and locate objects (such as data elements or file objects) according to some criteria which might be based on one or more fields containing key values to which search criteria are applied. In other words, Indexes are data structures designed to make search faster (Wesley, 2008 ; Elbassioni, et el. , 2003).

The nature of index could be an implicit or explicit key. The implicit index is the location of the target object to improve the speed of data retrieval operations on a database table particularly when the storing is costly (Crowe and Chizek, 2004; Mike et al., 2005).

Indexing significantly reduces the number of I/O operations, speeds up access to data as well as frees up the system resources for different applications.(Marcilina et al. , 2000).

1.2.1 Indexing techniques for relational databases

Currently, a database index can be organized in B-tree structure.

In B-tree structure, each page in an index is named and index page or an index node. The index configuration has distinct index structure. This structure begins with a root node at the top level

which marks the beginning of the index-it is the first data accessed when a data lookup occurs. The root node includes a number of index rows, which contain a key value and a pointer to an index page (Marcilina et al., 2000 ; Lin et al. , 2005).

Note that, the B-tree index configuration is necessary because it improves the information retrieving from huge table. B-tree index consists of thousands or millions of index pages. By starting at the root node and traversing the branch nodes, SQL servers can retrieve the data needed for the search criteria.

1.2.2 Index Process

As shown in Figure 1.1, the root node consists of a number of branch nodes. Each branch node contains a number of index rows held in an index page. Each index row points to another branch node or a leaf node. The leaf nodes make up the last level of an index. However, unlike the root node, each branch node also includes a linked list to other branch nodes on the same level. This means, the node recognizes for adjacent nodes as well as lower nodes as illustrated in Figure 1.2 (Sheldon , 2008 ; D. Lin et al., 2005).

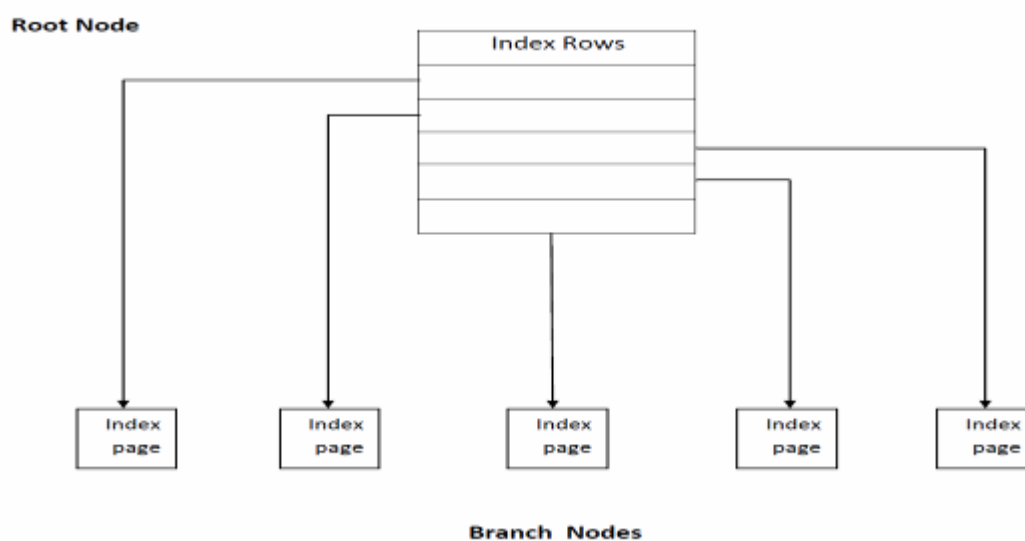


Figure 1.1 Root node and branch nodes of an index

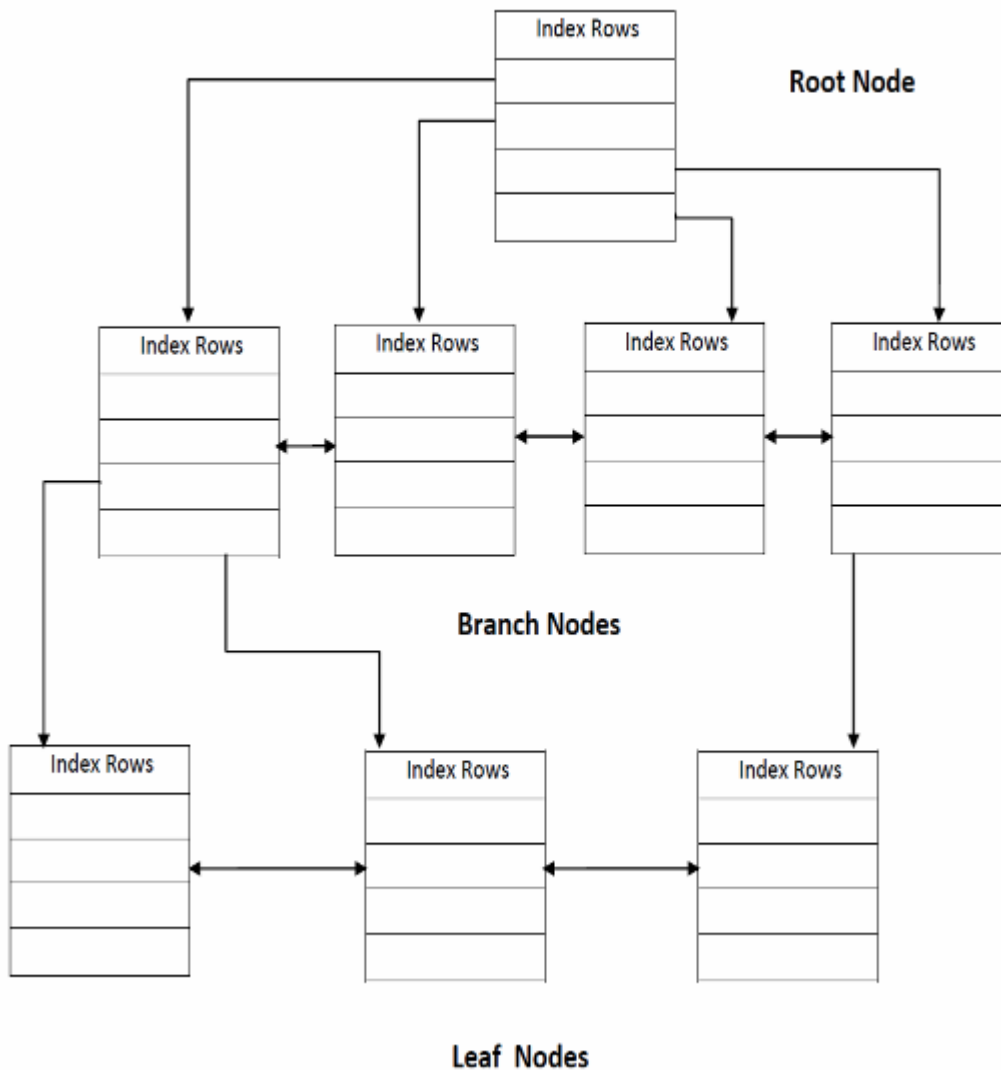


Figure 1.2 A search tree showing branch nodes and leaf nodes

Figure 1.3 illustrates that every group of branch nodes at the same level of the tree structure is known as an *index level*. Thus this thesis will investigate the number of I/O operations that are needed to reach the leaf nodes. Based on capacity of database, if the database table includes only a small amount of data, the root node can point directly to the leaf nodes, and then the index no longer needs to contain any branch nodes (Marcilina et al., 2000).

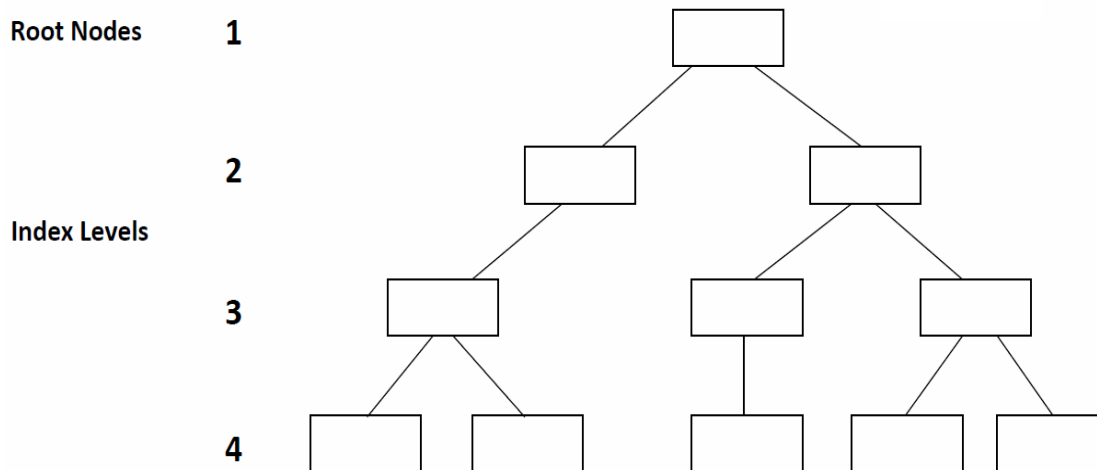


Figure 1.3 Index Levels

The next section describes the type of indexes including clustered indexes and non-clustered indexes.

1.2.3 Types of Indexes

There are two types of B-tree indexes: (i) clustered indexes and (ii) non-clustered indexes. In brief, a clustered index stores the real rows of data in its leaf nodes. Whereas, a non-clustered index is a secondary structure that points to data in a database table (Lin et al. ,2005; Oracle, 2005).

1.2.3.1 Clustered Indexes

As shown in Figure 1.4, a clustered index is a B-tree index that stores the actual row data of the database table in its leaf nodes, in sorted order. Such index offers several advantages and disadvantages.

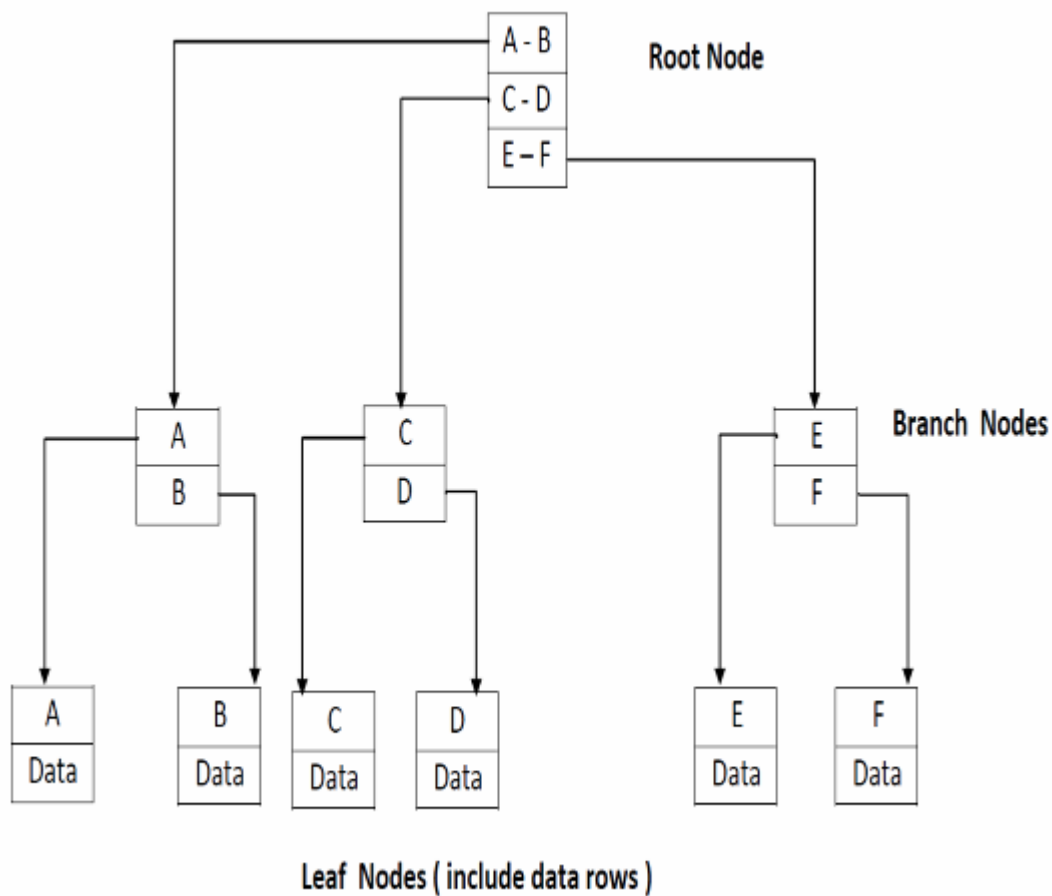


Figure 1.4 B-tree Clustered Index

The first advantage is that the data availability which can result in a minimum number of I/O operations. Consequently, any reduction in these I/O operations will yield better performance for the individual operations and greater overall performance for the B-tree index than other indexes (Marcilina et al., 2000; Oracle, 2005).

Further advantage is that the retrieved data will be in index-sorted order. For instance, if a clustered index is created on the county, and city columns and a query selects all values for which city? Kuwait, the results will be sorted on county and city according to the order in which the index is declared.

As a result, the sorted data feature enhances the time efficiency by reducing the time overhead on database. Use of a clustered index means it will not need to perform the sort after the data retrieval (Marcilina et al., 2000; Oracle, 2005).

On the other hand, the first disadvantage of using a clustered index is that access to the database table can result in extra time overhead. The SQL server begins data access at the root node and traverses the index until it reaches the leaf node including the data. Based on the capacity of the leaf nodes- if many more leaf nodes are built, the number of index levels necessary to support that many leaf nodes is also large. Thus, this involves more I/O operations to pass from the root node to the leaf node (Sheldon , 2008; Oracle, 2005).

1.2.3.2 Non-clustered Indexes

Currently much research relies on clustered indexes; however, some database engines still use the non-clustered indexes. The non-clustered index is different from clustered index at the leaf nodes. The non-clustered index does not include actual table data in its leaf nodes.

Figure 1.5 shows that if there is no clustered index on a table, non-clustered indexes on that table store Row Identifications (Ids) in their leaf nodes. At this stage, each Row ID points to the actual data row in the table. The Row ID comprises from a value that contains: the data file ID, the page number, and the row in the page. The importance of this value is to enable quick access to the actual data by pointing where the data is stored (Sheldon, 2008; Oracle, 2005).

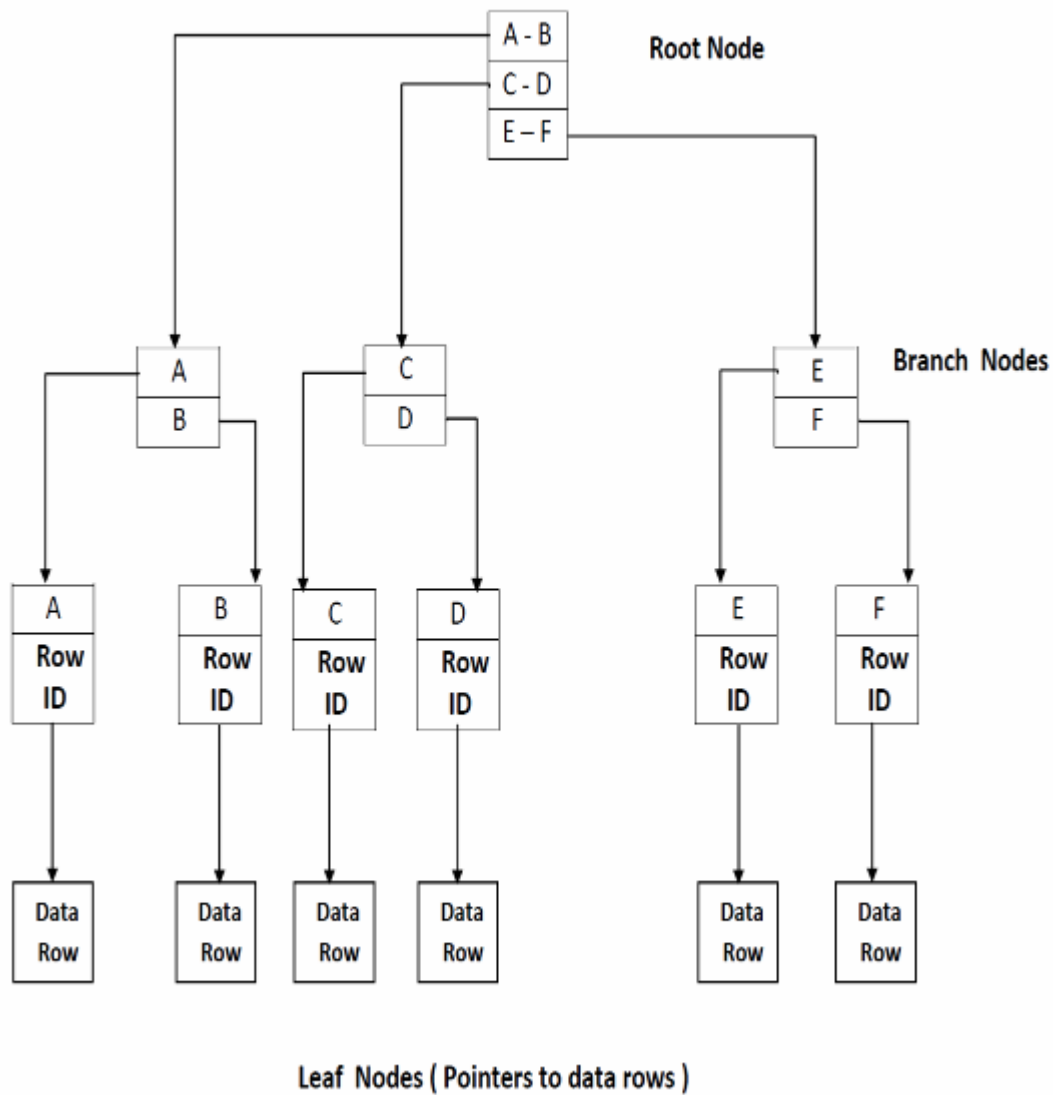


Figure 1.5 Non-Clustered index on a table without a Clustered index.

On the other hand, if there is a clustered index on the table, non-clustered indexes will contain the clustered index key value for that data in the leaf node, as illustrated in Figure 1.6. When the leaf node of the non-clustered index is reached, the clustered key value indicates that it should be used to search the clustered index, when data row will be found in its leaf node (Marcilina et al., 2000).

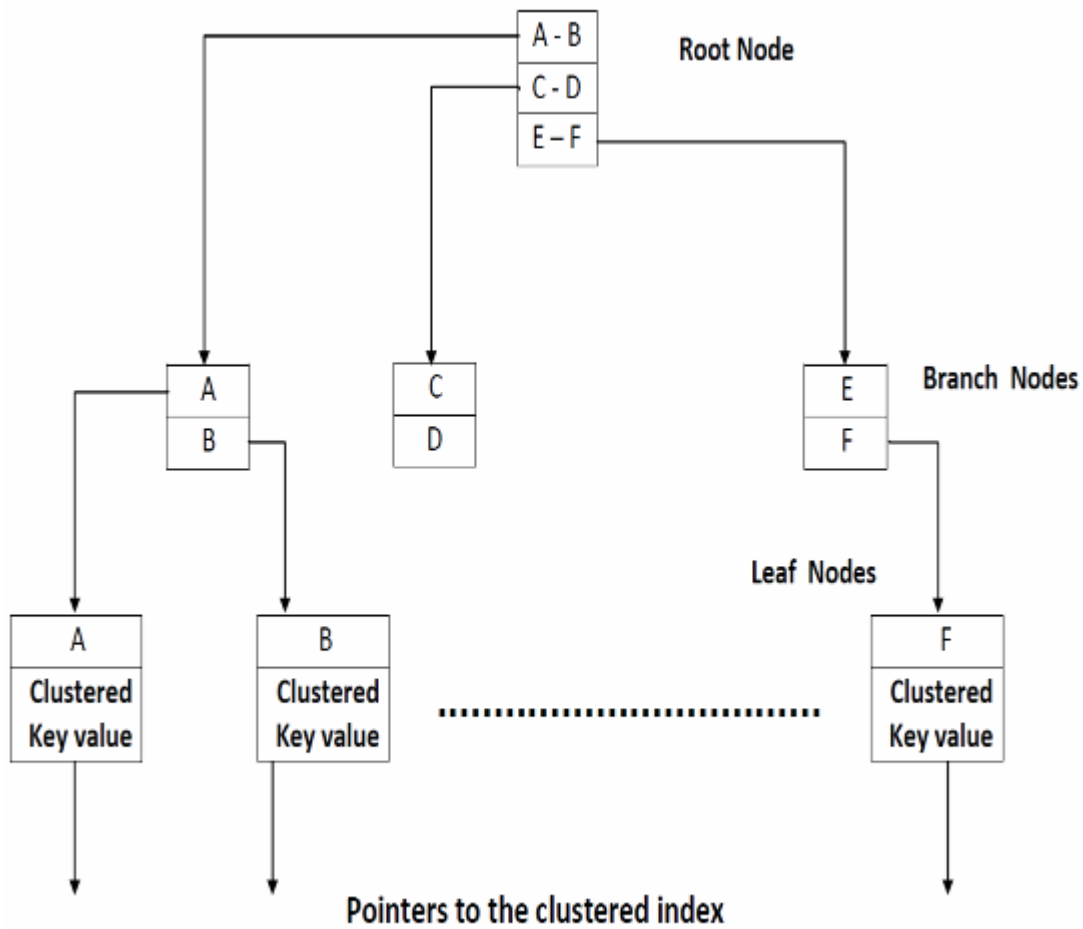


Figure 1.6 Non-clustered indexes on a table with a clustered index.

1.3 Indexing Examples

In this section, the B-tree indexing is described in Oracle and MS-SQL Server Database Management Systems.

The B-tree index in the Oracle and MS-SQL Server is the most popular type for servicing queries and for avoiding large sorting operations (Burleson , 2002; Arzucan ,2006).

In Oracle, a number of choices are provided when creating an index using the default B-tree structure as follows:

1. Indexing on multiple columns (concatenated indexes) to improve access speeds.
2. Allowing for individual columns to be sorted in different orders. For example, it is possible to create a B-tree index on a column called “city” in descending order and have a second column within the index that displays the “country” column in a ascending order (Burleson 2002; Arzucan 2006).

Figure 1.7 describes a B-tree index in a hierarchical manner with a root node at the top and the leaf nodes at the bottom.

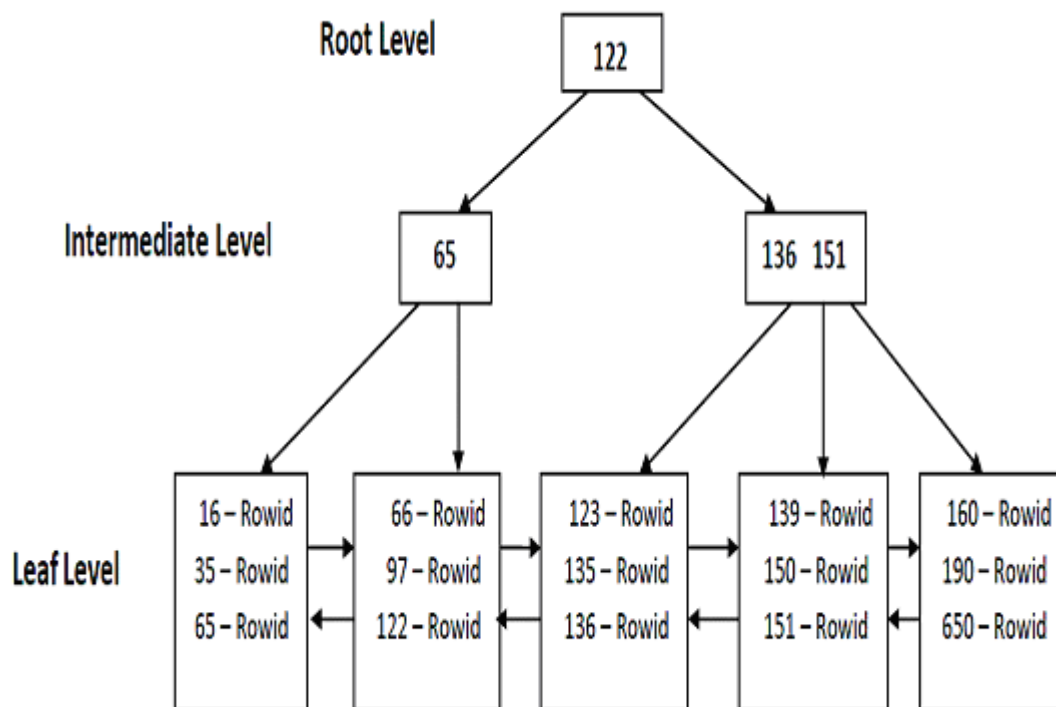


Figure 1.7 An Oracle B-tree index

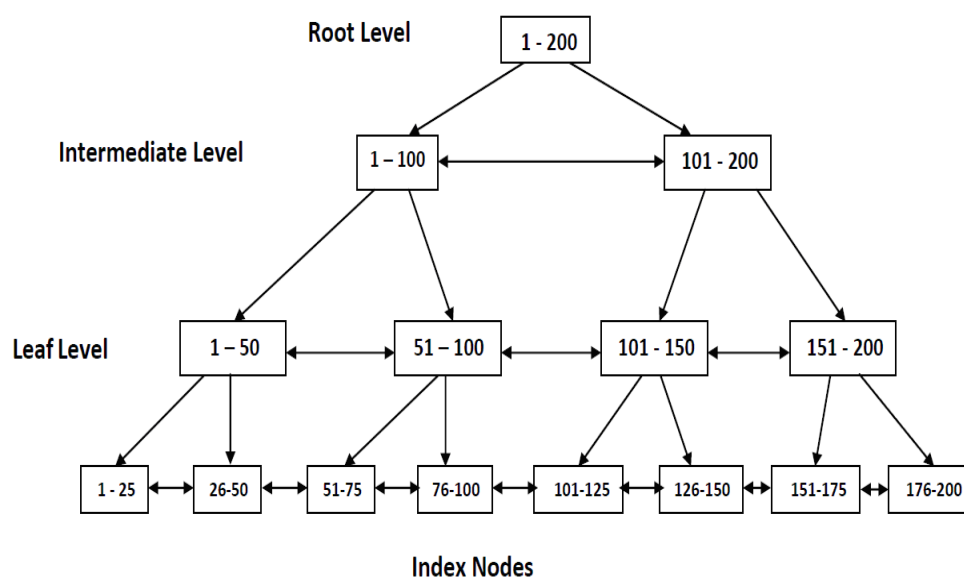


Figure 1.8 MS-SQL Server B-tree index.

Figure 1.8 depicts a practical example which demonstrates the search method in a B-tree. If “Huda” searches for the value 156, then the query engine would start at the root level to determine which page to reference in the top intermediate level. The first page points to the values 1-100, and the second page points to the values 101-200, so the query engine would go to the second page on that level. The query engine would then determine that it must go to the fourth page at the next intermediate level. From there, the query engine would navigate to the leaf node for value 156. The leaf node will contain either the entire row of data, or a pointer to that row depending on whether the index is clustered or non-clustered (Sheldon , 2008; Giugno , 2002) .

1.3.1 Oracle Indexing

Following a description of some indexing techniques on Oracle database.

1. B-Tree Index

B-tree is one of the methods used for searching a specific value on the tree. To implement this method is to invoke the adjacent value of the node and compare it, if the value of the node is less

than the root it will start from the left part of the tree; and if the value is greater, it will start from the right part of the tree.

2. Bitmap Index

Cyran et al. (2005), discovered the Bitmap index. Bitmap is a method containing a two-dimensional matrix and each column stores value in a single bit. The implementation of this method is that each index will have either 1's or 0's, if the value is found, it will specify 1's; but if it is not found it will specify 0's. While searching, the oracle will take the entire row indices that have a value of 1's. In other words, it will be pointed at the index that has been found.

3. Reverse Key Indexes

Cyran et al. (2005), created Reverse Key Index. It is a method that compares a standard index, and reverses the entire bytes in each column except the rowed, in the mean while it will keep the columns in order as arrangements will avoid performance degradation with real application cluster. Oracle would be forced to search for each specific index value separately as each value in the range is likely to be in differing leaf blocks. By reversing the keys of the index, the insertions become distributed across all leaf keys in the index

4. Index-Organized Tables

Index organized tables contain a storage organization that is totally different from B-tree, it is a distinct ordinary, such as heap-organized. The table that is stored as data in an unordered collection will be organized in a table and stored as in B-tree index structure as a primary key stored manner. On the other hand, the primary key will be stored as column values of an index-organized table row. Each index in the not key column values will be entered in the b-tree stores structure.

1.3.2 MS-SQL Server Indexing

Following is a description of some indexing techniques on MS-SQL Server database.

1. Clustered Index

Vassie & Lee (2009), sorted and stored the data rows of the table to perform the table in order. This is based on clustered index key. Clustered index is implemented as same as B-tree index structure which supports fast retrieval of the row. This implementation is based on the value of the clustered index.

2. Non-clustered Index

Vassie & Lee (2009), performed a table with a clustered index or on heap, it can be defined in a non-clustered index. Also, the rows in the non-clustered index will contain the non-clustered key value and a locator for the rows. This locator is a pointer to the data row to the clustered index in order to obtain the value of the key. However, if the data in a clustered index is created in the table will be in order, otherwise it is not guaranteed to be in any particular order.

3. Unique Index

The unique index is one of the techniques that ensures no duplicate values. Therefore, each row on the table should be unique in some way. Both clustered and nonclustered are unique.

4. B-Trees Index

B-trees index is a multi-way trees (forest), each node will contain two parameters set of keys and pointers. The minimum size of B-tree node is a tree which has four keys and five pointers. Also, it contains data pages and it is a dynamic which means as much as the height of the tree is growing the record are added and deleted.

1.4 The Current Indexing Techniques

This thesis is focused on relational database indexing techniques. As a part of future work, the object oriented databases indexing techniques will be discussed. So, RDBMS employ many techniques and methods to enhance the overall performance based on indexes. Thus, this section mentions the most popular indexing techniques used in Oracle and MS-SQL Server, as shown in Table 1.1.

Table 1.1: Summary of indexing types in Oracle and MS SQL Server

Database Platform	Indexing Type
Oracle	B-Tree Index
MS SQL Server	
Oracle	Bitmap Index
MS SQL Server	
Oracle	XML Index
MS SQL Server	
Oracle	Index Organized Tables
Oracle	Domain Index
Oracle	Cluster Index
MS SQL Server	
Oracle	Reverse Key Index

Oracle	B-Tree cluster Index
Oracle	Bitmap Join Index
MS SQL Server	Non-clustered Index

1.5 Advantages and Disadvantages of Indexes:

Indexes have the following advantages and disadvantages:

- **Advantages**

1. Use ordered data to avoid sorts to favor merge joins over nested-loop joins.
2. Speed up reading a row when the right search arguments are known.
3. Index scans are much faster than table scans.
4. Index files are generally small in size and require less time to read than an entire table, particularly as tables grow larger.
5. The entire index may not need to be scanned.
6. The predicates that are applied to the index reduce the number of rows to be read from the data pages.
7. If an ordering requirement on the output can be matched with an index column, then scanning the index in column order will allow the rows to be retrieved in the correct order without sorting.
8. Each index entry contains a search-key value and a pointer to the row containing that value, so values can be searched in an ascending or a descending order.

9. An index can include columns, which are non-indexed columns in the indexed row. Such columns might make it possible for the optimizer to get required information only from the index, without accessing the table itself.

- **Disadvantages**

Although indexes can reduce access time significantly, they can also have adverse effects on performance. Before you create indexes, consider the effects of multiple indexes on disk space and processing time .

1. Each index requires a storage or a disk space. The exact amount depends on the size of the table and the size and number of columns in the index.
2. Each INSERT or DELETE operation performed on a table requires additional updating of each index on that table. This is also true for each UPDATE operation that changes the value of an index key.
3. The LOAD utility rebuilds or appends to any existing indexes.
4. Each index potentially adds an alternative access path for a query for the optimizer to consider, which increases the compilation time.

1.6 The Problem Statement

Indexing is a critical area in the era of academia and industry and it should increase the speed of processing information and enhance accessing the storage areas of hard drives and search engines. A number of researchers and database practitioners have attempted to set guidelines for using the indexing techniques in Oracle and MS SQL Server. However, there is a lack of knowledge because they did not consider part of criteria (such as CPU time, I/O operations, Packet size, and memory factor) which are used to demonstrate the overall performance of indexing types in Oracle and MS

SQL Server. It should be noted that the indexing techniques will be implemented and tested on two platforms: Oracle and MS SQL Server.

Thus the problem question is how to establish a guide to advise the database developers how to select the best indexing techniques that suit their database design. This will be done in this thesis contribution “A Comparative Study of Indexing Techniques for Relational Database Management Systems”.

1.7 Significance of the Study and Motivations

Through our research process in the database indexing field, it is found that there is no systematic approach for the selecting of indexing which relies on the expertise and experience of the user to use it. Therefore, progress should be taken into account to construct a referenced guide to give all database developers who are interested in enhancing the database engine performance and to allow them to understand how an index could affect the performance and use indexing and when not. Moreover, we will cover indexing techniques that help developers and DBAs to make quick, correct decisions in choosing their type of indexing.

1.8 Thesis Contributions

The main contributions of this thesis are summarized as follows:

- 1- We estimated rules (Full Scan SELECT operation, Range SELECT operation, and Single-row SELECT operation) and criteria (I/O cost, CPU consumption, and Performance) that makes the decision of selecting appropriate indexing technique.
- 2- We have done a systematic comparison between the available indexing methods (B-tree, Bitmap, reverse, and organization index) on Oracle platform and the indexing techniques

(Primary key clustered, unique clustered index, and non-unique clustered index) on MS SQL Server platform).

- 3- We constructed a referenced guide to help database developers and DBAs for selecting the indexing method in order to retrieve their data in efficient method.

1.9 Methodology

The research methodology that will be followed is directly connected to our problem statement and contributions of this thesis. Since the thesis purpose and problems may vary, different methods of research can be utilized. A systematic literature review has been performed to analyze all the facts about indexing with the focus on comparing experimental tests and the indexing techniques (B-tree index, Bitmap index, reverse index, and organization index) on Oracle platform and the indexing techniques (Primary key clustered, unique clustered index , and non-unique clustered index) on MS SQL Server platform with different size of data (100K, 1000K, and 5000K) and practical results on table indexing.

After that, testing and evaluation have been conducted which are based on the following criteria:

- a. Performance with various sizes of data for both environments Oracle and MS-SQL Server.
- b. CPU time for both environments Oracle and MS-SQL Server.
- c. Memory factor for both environments Oracle and MS-SQL Server.
- d. Number of I/O operations for for both environments Oracle and MS-SQL Server.

As a result of evaluation, three main points are obtained:

- 1- To build indexing reference for the DBAs and developers; we will assume that indexes can be created well if they have the right selection to evaluate the fitness of an index.

2- To assess the indexing techniques; we will create databases with various sizes ranging from small, medium and large to assess many types of indexing for each database in Oracle and MS-SQL Server.

3- To represent the results we will use special SQL commands to analyze tables; the methods for query optimization include: SQL Trace and Oracle Trace.

The methods adopted in this thesis are in order to clarify indexing by assessing many types of indexes to identify which one is the better for special types of tables by considering and taking into account the size of tables.

1.10 Limitations

Complete and consistent assessment of all indexing methods for all database types is not possible due to difficulties in representing every case scenario to cover all types of databases. Therefore we will take the most common methods of indexing used by Oracle and MS-SQL Server.

1.11 Literature Review and Related Work

This section is drawn according to the recent research papers and technical reports. We survey the recent indexing techniques such as clustered indexes and non-clustered indexes. Also we state the process of these techniques, strengths, and weaknesses. Based on this comparison, the thesis will adopt the methodology for selecting the optimal database indexing techniques.

1.11.1 Literature Review

As mentioned in pervious sections, the thesis is concentrated on the database indexing techniques. Thus, this section discusses the past, and present of indexing techniques.

Martin et al., (1992) presented a novel approach for a tool that assists the database administration in designing an index configuration for the relational database system. A relational database uses indexes to provide a fast access to data repository. However, there is a tradeoff involved in use of indexes for every column. As a result, this tradeoff is referred as the Index Selection Problem (ISP). The ISP denotes to tailoring the configuration of index to the database usage profile. Moreover, Martin's research presented a methodology for run time facility. This methodology for collecting usage statistics at run time was developed which lets the optimizer estimates query execution costs for alternative index configuration. However, this research work has two weaknesses:

- a) Requiring much time for running the queries, and
- b) Defining a workload specification required by existing index design tool,
- c) The architecture of this tool may be very complex for large integrated database system.
- d) The proposed tool automatically derives the workload statistics. These statistics are then used to efficiently compute an index configuration.

Chaudhuri and Narasayya , (1998) introduced algorithms for the index selection tool in Microsoft's SQL Server as part of the AutoAdmin project. The objective of the index selection tools is to generate an index set for a given input workload, obtained by the DBA to be able to perform a quantitative analysis of the existing indexes. In addition, the DBA should have the ability to propose hypothetical ("what-if") indexes and quantitatively analyze their impact on performance of the system. The authors also presented interfaces supported by a Hypothetical Configuration Analysis Engine (HCAE) to conduct significant and powerful analysis studies.

However, the developed HCAE limits the overall performance of query processing and information retrieval systems.

Gaffar, (2001) described how to create B-tree index from independent building blocks that are coherent and decoupled. He expanded the concept to build a complete modular index system. In this design, the index data structure is broken with functionality into container of pages. Each page is built as container of entries where each entry is a pair of <key, reference> .

In the end, the data and the data reference modules are added to complete the system. This allowed constructing a complete index system from modules. In order to adapt the system to different keys/data types, different queries, different access methods, and different storage media, it is needed to locate and modify some modules in the system. This reflects the modifications on the system design and eventually on the interface.

Using a modular design for the index system has the advantage of making it easier to adapt the system to work in different database domains. The analysis of the domain determines the modules that need changes (or replacement), and the sort of changes (or modules) required.

The complexity of modification is also reduced since the developer does not need to know about the details of all modules, but only of those modules to be changed along with an overview of the system. The adoption of Standard Template Library (STL) approach adds great advantage of having a wealth of off-the-shelf standard modules that can be simply used to replace system modules in the process of modifying the system. The modification reduces time and money overheads incurred during the application development process.

Aouiche et al., (2005) presented an automatic strategy for bitmap index selection in data warehouses. In order to improve a response time, data warehouse administrators generally use

indexing techniques such as star join indexes or bitmap join indexes. The proposed model estimated data access cost through indexes, maintenance and storage cost. The experimental evaluation showed that the application of cost models to their index selection strategy decreased the number of selected indexes without a significant loss in performance. Thus, this actually guaranteed a substantial gain in storage space and then a decrease in maintenance cost during data warehouse update.

Graefe, (2010) developed a B-tree locking technique. The process of this technique is dividing B-tree into two sub-topics and exploring each of them in depth. Concurrency control for B-tree indexes in databases can be separated into two levels: (i) concurrent threads accessing in-memory data structures and (ii) concurrent transactions accessing database contents. These two levels are implemented with latches and locks. The functions of latches and locks are explained as follows:

1. Latches support a limited set of modes such as shared and exclusive. They do not provide advanced services such as a deadlock detection or escalation. They can also be embedded in the data structures for protection. Therefore, their acquisition and release can be very fast. Furthermore, they implemented short critical sections in the database system code.
2. Locks provide several modes and multiple advanced services. The management of locks is separated from the protected information, for instance, keys and gaps between keys in the leaf of a B-tree index. Note that the hash table in the lock manager is in fact secure itself by latches.

The conceptual technique for concurrency control among transactions accessing B-tree contents is a key range locking.

The ultimate recent design has the following advantages:

- A) Allowing separate locks on individual key values and on the gaps between key values.

- B) Applying strict multi-granularity locking to each pair of a key and a neighboring gap.
- C) Reducing lock manager invocations by using additional lock modes that can be derived automatically.
- D) Enabling increment locks in grouped summary views.
- E) Exploiting ghost records not only for deletions but also for insertions.

1.11.2 Related Work

Zobel et al. , (1996) discussed the techniques and methods for new indexing which are of common outcome regarding the database research. Moreover, they presented a framework and compared the proposed framework with the existing indexing techniques and schemes. Based on the criteria (namely direct argument, mathematical modeling, simulation, and experimentation), they discussed the principal methods.

The aim of this comparison is to indicate the minimum overall speed, CPU , time and ease of index construction. In a dynamic system should also consider index maintenance in the presence of addition, modification, and deletion of records; and implications for concurrency, transactions, and recoverability.

King, (2001) Oracle8i provided many database tables which have primary or unique keys based on a sequence. These keys are usually indexed by b-tree indexes which, by nature, store the indexed values in order. These types of indexes can become performance bottlenecks on high-volume transactional systems because of serialization that occurs when inserting values into the leaf-blocks of these indexes.

To avoid this serialization, reverse-key indexes can be used. A reverse-key index stores indexed values in reverse-bit order. So, where the values (31, 32, 33) are stored sequentially and

contiguously in a normal b-tree index, they were stored out of sequence and non-adjacent (33, 31, 32). Over a larger set, this reversing of the key distributed the indexed values across the leaf-node blocks of the index, thereby eliminating the serialization on sequential inserts.

King, (2001) Oracle8i presented different types of indexes : (B-tree index , Bitmap index , index-Organized table and reverse-key index), then compared and contrasted the various options available and how to choose from among them . So that, the system helped developers when deciding not just what columns to index but how to index them .

King, (2001) Oracle8i described the index organized tables worked best when there are few columns in the table / index and the size of a row is small compared to the size of block. Index organized columns may not consist LONG columns. Index organized tables may not be used in a cluster.

Madhulatha , (2010) built a new methodology for collecting usage statistics at run time. This methodology developed the optimizer to estimate query execution costs for alternative index configurations that assist the database administrator in designing an index configuration for a relational database system. In addition, the proposed methodology defined the workload specification required by an existing index design tools which may be very complex for a large integrated database system.

However, one need to automatically derive the workload statistics and these are then used to efficiently compute an index configuration.

1.12 Thesis Organization

The reminder of this thesis consists of the following Chapters:

Chapter 2; discuss the thesis methodology. We describe the methodology for selecting indexing techniques for relational databases.

Chapter 3; implements the experimental tests and shows the results. According to these results, the thesis is evaluated.

Chapter 4; draws the conclusions and future work. We identify the real outcomes and compare with the expected contributions. As a result, we have achieved the objectives of this thesis.

Chapter 2

Test Methodology and Experimental Test

2.1 Introduction

The majority of commercial (RDBMS) Relational database management systems performance is relied on I/O operations rather than other computing resources. This is because the performance cost of I/O is expensive and there are other costs such as memory allocations and CPU consumption. The most important factor to consider is whether the I/O subsystem of a given (RDBMS) will support a reliable performance as time passes.

This chapter describes a methodology for evaluating indexing techniques for relational databases. The methodology is based on a number of experiments to test a set of indexing techniques on two different platforms (Oracle and MS-SQL Server) with different data sizes (small, medium, and very large) over the same technical environment (Multiple processors, memory, and I/O devices). This factor is necessary to satisfy the real results on different platforms: Oracle and MS-SQL Server.

To run the experiments, we have taken the following indexing techniques in Oracle: B-tree, Bitmap, Reversed, and organization index. In the meanwhile, we have taken the following indexing techniques in MS-SQL Server: B-tree, Clustered index, and unique non-clustered index and Primary Key Clustered index.

The methodology includes the technical environment, platforms, table schema, table sizes, and a number of indexing techniques. We have also established a number of test scenarios to achieve the real results. The methodology procedure and flowchart include an ordered set of steps that have been taken to run the experiments in this thesis.

The aim of the methodology contribution is to measure overall performance and behavior of indexing techniques that are performed against the same set of data: (i) As a non-clustered index on a specified set of columns, (ii) and as a clustered index on the same set of columns. Note that, we have measured the performance of SELECT operation over Oracle 10g and MS SQL Server on different data sizes (100K, 1000K, and 5000K).

2.2 Methodology Contributions

The contribution of our test methodology is to characterize the performance and behavior of DML operations performed against the same set of table data organized:

- As a non-clustered index on a specified set of columns.
- As a clustered index on the same set of columns.

In this test, a number of questions should be answered through the experimental results as follows:

1. Are clustered indexes necessary for all tables?
2. Are non-clustered indexes necessary for all tables?
3. What are the performance gains or losses for row-by-row SELECT operations executed against three different sizes of tables (100K, 1000K, and 5000K) with a clustered index versus the same tables without a non-clustered index for a high-throughput workload on Oracle 10g and MS-SQL Server 2005 platforms?
4. How does a range query perform on the same tables with a clustered index versus a non-clustered index?
5. What are the effects of having the first column index be monotonically increasing? The purpose of this test is to measure performance.
6. What are the CPU utilization characteristics when rows are selected from a table with a clustered index and from non-clustered index?

2.3 Criteria for Comparison

Each type of index is related to query evaluation algorithms that access the requested information, and update algorithms that maintain it. There are many criteria by which indexing techniques can be compared. We need to consider the overall speed, space requirements, CPU time, memory requirements, measurements of disk traffic such as numbers of seeks and volumes of data transferred, and ease of index construction. All of these considerations will be in the context of assumptions made about the properties of the data and queries (Zobel et al., 1996).

2.4 Methodology Assumptions

To make a contribution to the study of indexing, it is not sufficient to simply describe the current indexing techniques. It is also necessary to provide a demonstration of the value of the method, and place it in the context of other established methods. This demonstration will be based on several constraints and assumptions: the class of data (such as textual and multimedia data), the class of queries (such as SELECT queries), characteristics of the application, for example-and characteristics of the supporting hardware for both MS SQL Server and Oracle.

Database Administrators (DBA) will judge the success of the used technique according to its performance on the basis of the stated. Assumptions should not only be claimed to be reasonable, they should be argued for, and, where possible, demonstrated as being reasonable.

Similarly, assumptions about hardware should associate with the current technology or likely future improvements. The performance of the hardware should be related to common benchmarks, to allow comparison with familiar systems and to convey the impression that the technique will be of value on probable hardware-rather than a machine with limited memory but massive arrays of parallel disks.

2.5 Methodology Description

The main aim of this section is to conduct the tests described in the previous section against throughput workload that represented real-world scenarios. Another aim is to keep the test setup (server configuration, database settings, table schema, computer configuration and Operating System) approximately constant across the tests so that we compare the overall performance between different SELECT operations using different indexes.

After some testing and analysis using real-world measurements, we have noticed that the testing results could not represent the real performance measurements. This is because the experiments are conducted on stand alone machine. Even though, we have obtained results in a way that would make these results meaningful and applicable to a wide variety of other workloads.

Based on our findings, we have drawn a number of recommendations for DBA's and researchers. This methodology is used for the tests explained in Experiments Test and Results section. Our intent is that these individual tests might help the DBA's and researchers to estimate the overall impact of the index choices for a particular application on both platforms: Oracle and MS- SQL Server. Further intent is that the obtained results could ease the selection of the optimal indexing techniques for a certain application and platform.

2.6 Technical Test

We have conducted all tests on computer hardware that was configured with adequate storage. We have used a TOSHIBA Satellite with Core TM i7 CPU processors: 720 @ 1.60GHz and 4-GB memory. The tests are performed on Windows 7 Home Premium for both platforms: MS SQL Server 2005, and Oracle 10g.

2.7 Test H/W & S/W (Test Environment)

We have obtained results for the following test scenarios as shown:

SELECT performance

1- Measure the time taken to select 100K, 1000K, and 5000K rows of data from the table with the primary clustered index, unique clustered index and unique non-clustered index by using individual (row-by-row) select statements in MS SQL Server Platform.

2- Measure the time taken to select 100K, 1000K and 5000K rows of data from the table with the Bitmap index, B-tree index, Reverse index and Organization index by using individual (row-by-row) select statements in Oracle Platform.

2.8 Test Procedure

As shown in Figure 2.1 and Figure 2.2, our methodology procedure was performed on two platforms: Oracle 10g, and MS-SQL Server, respectively. The following steps are used to execute the tests over Oracle platform as described in Test scenarios.

1. The table of size 100k is created and initialized on Oracle platform.
2. The bitmap index is created on the table.
3. The particular test is executed.
4. The table is dropped.
5. The table of size 100k is created and initialized on Oracle platform.
6. The above steps (1-5) have been repeated for other indexing techniques (unique index (B-tree), reverse index and organization index).

7. Note that the procedure that contains steps from 1 to 7 has also repeated for two different data sizes (1000K and 5000 K).

On the MS-SQL Server platform, we have performed the above procedure (steps 1- 7) taking into considerations the following indexing techniques:

1. Primary key clustered index
2. Unique clustered index
3. Unique non-clustered index

Figure 3 illustrates the flowchart of the test procedure. Note that this process of flowchart shows the steps from 1-7 on different platforms with different data sizes. Further note is that the Oracle supports different indexing techniques from MS-SQL Server, as explained in Chapter 1 “Related Work”.

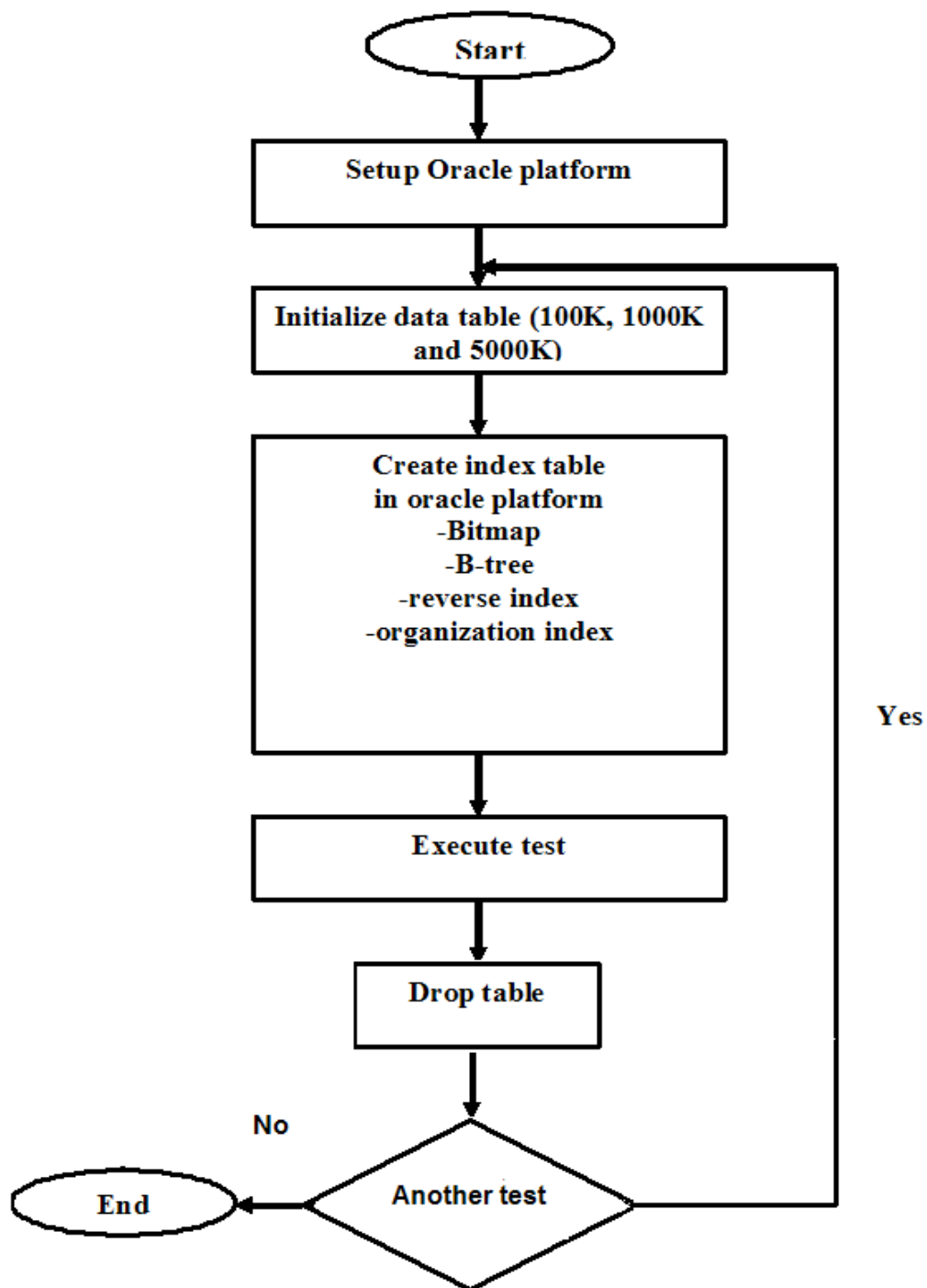


Figure 2.1: Flowchart of index evaluation on Oracle platform.

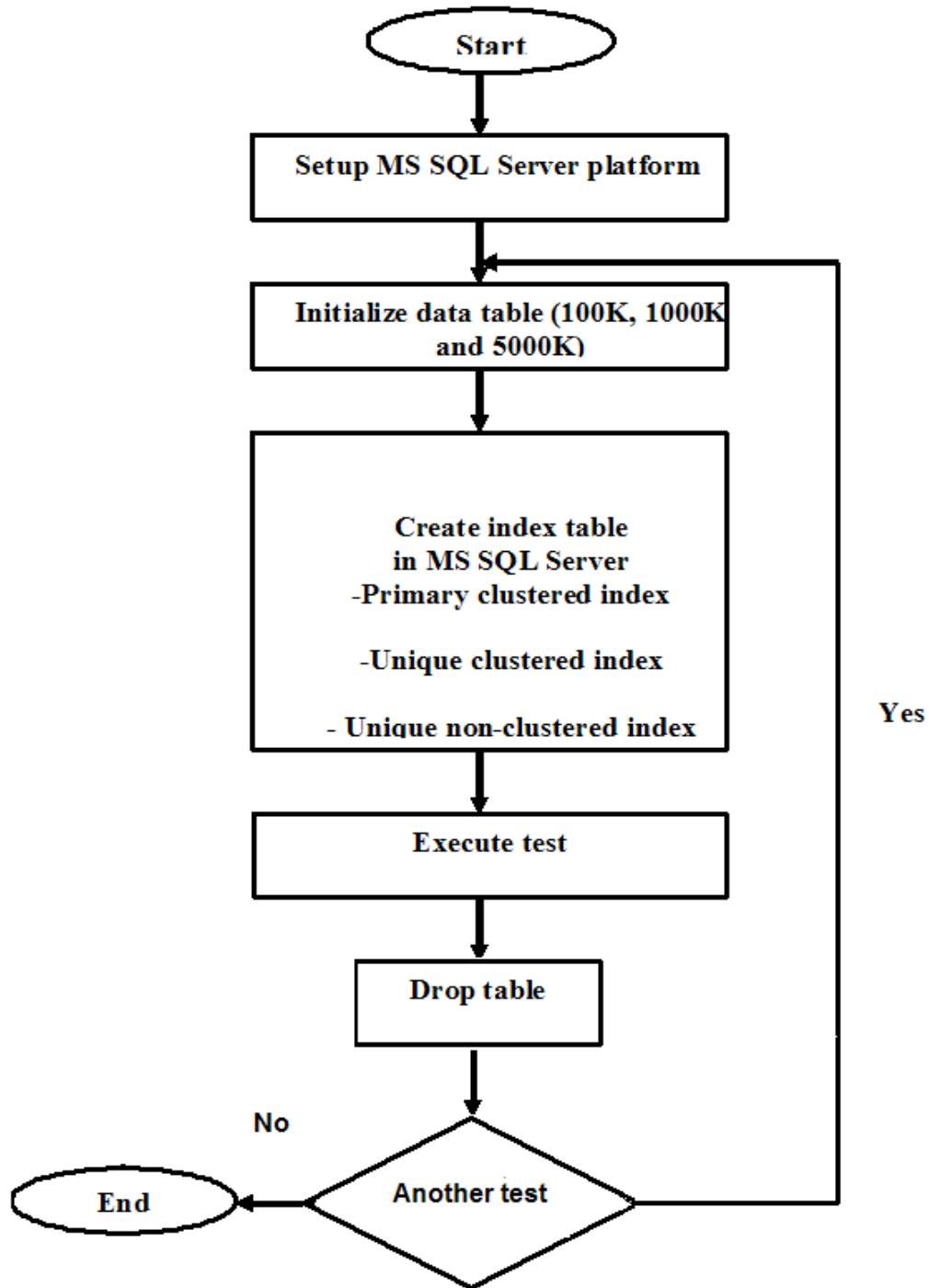


Figure 2.2: Flowchart of the index evaluation on MS-SQL Server platform.

Chapter 3

Experiments and Test Results

3.1 Introduction

This chapter describes the tests for selecting the indexing techniques in detail and presents the results measured. We have evaluated the experimental tests through measuring the performance (response time) of SELECT operation over Oracle 10g and MS SQL Server on different data table sizes (100K, 1000K, and 5000K). In addition, we have mentioned the costs of I/O operations, CPU consumptions and the retrieved rows. Furthermore, a set of guidelines have been added for helping the DBA to select the best indexing techniques.

We have found out the SELECT performance on four different types of SELECT statements including select operation retrieved a single row; select operation retrieved a number of rows depending on the condition in the SELECT statement; SELECT statement retrieved all rows; and SELECT statement retrieved non-row. In order to advise the best indexing technique, this section also contains evaluation analysis for each type of SELECT statement.

Furthermore, we have introduced deeper a analysis through comparing the results on Oracle with MS SQL Server. Even it is very difficult to compare between two different platforms: Oracle and MS SQL Server, we have attempted to run our experimental tests on the identical environment to achieve the possible encouraged results.

Finally it also summarizes observations and includes general recommendations where appropriate.

3.2 Test: SELECT Performance

The SELECT test measured the performance on four different types of SELECT statements:

Test a. Single-row SELECT performance—in this test, each SELECT statement retrieved a single row.

Test b. Rangeⁱ SELECT performance—in this test, each SELECT statement retrieved a number of rows depending on the condition in the SELECT statement.

Test c. Fullⁱⁱ scan SELECT performance—in this test, each SELECT statement retrieved all rows.

Test d. Single-row SELECT performance—in this test, each SELECT statement retrieved non-row.

3.2.1 Test 1: Oracle 10g and 100K

3.2.1.1 Test 1-a: Single-row SELECT performance

The single-row SELECT test has been conducted on Oracle 10g platform with 100K as table size. It should be noted that the row size is 1K and the data attribute is STUD_ID in the SELECT statement. As shown in Table 3.1 the performance of Bitmap index is less costly compared with the other indexing techniques including B-tree (Unique index), Reverse index, and Organization index. The Bitmap index and Organization index consume less response time (2 ms) than B-tree (3 ms) and Reverse index (3 ms).

3.2.1.2 Test 1-b: Range select performance

In this test, we have performed a range select experiment on STUD_GRADE attribute. This attribute has five values from A, B, C, D, and F. On Oracle 10g platform and 100K as file size, the

Table 3.1 illustrates that the Bitmap index consumes less response time (2 ms) than the Reverse index (3 ms).

For example:

```
SELECT STUD_ID FROM STUDENT WHERE STUD_GRADE = B;
```

There are two basic plans the query optimizer could choose:

- Plan 1: Reading all the rows from the "STUDENT" table, for each, check if the predicate is true (STUD_GRADE = B).
- Plan 2: Read the index where STUD_GRADE = B, then access the table based on the ROWIDs returned.

In this test, the "STUDENT" table has 100,000 rows. Also, the values for STUD_GRADE range from A, B, C, D and F.

The cost of Plan 1, which involves a FULL SCAN, will be the cost of reading all the rows in the STUDENT table, which is approximately equal to 100,000; but since Oracle will often be able to read the blocks using multi-block reads, the actual cost will be lower (depending on the database setting up). For example, if the multi-block read count is 1, then the calculated cost of the full scan will be 100,000. Note that the cost has not measured unit.

The cost of Plan 2, which involves an INDEX RANGE SCAN and a table lookup by ROWID, will be the cost of scanning the index, plus the cost of accessing the table by ROWID. The cost of the index range scan is 1 per row; it is expected to find a match in 1 out of 5 cases, so the cost of the index scan is $100,000 / 5 = 20,000$; plus the cost of accessing the table (assuming 1 block read per access) = 20,000; Overall cost = 40,000.

Therefore, the cost of Plan 1 (Full scan) is much greater than the cost of Plan 2 (Index range scan + access by rowid). This means the query optimizer would choose the Index Range scan. Table 3.2 shows that the cost of full scan Bitmap index is 340, whereas the cost of index range Bitmap is 2. This result proves what we have noted above.

3.2.1.3 Test 1-c: Full scan SELECT performance

The full scan test has been conducted on Oracle 10g platform with 100K as table size. It should be noted that the row size is 1K and the data attribute is STUD_ID in the SELECT statement. As shown in Table 3.1, the performance of Bitmap index is the best compared with the other indexing techniques including B-tree (Unique index), Reverse index, and Organization index. The Bitmap index consumes less response time (4 ms) than B-tree (6 ms), Reverse index (6 ms) and Organization index (8 ms) as illustrated in Figure 3.1. On the other hand, another full scan test has been conducted on Oracle 10g platform with 100K as table size. The data attribute is STUD_GRADE in the SELECT statement. As shown in Table 3.1, the performance of Bitmap index is the same compared with the Reverse index. The Bitmap index and Reverse index consume the same response time (6 ms).

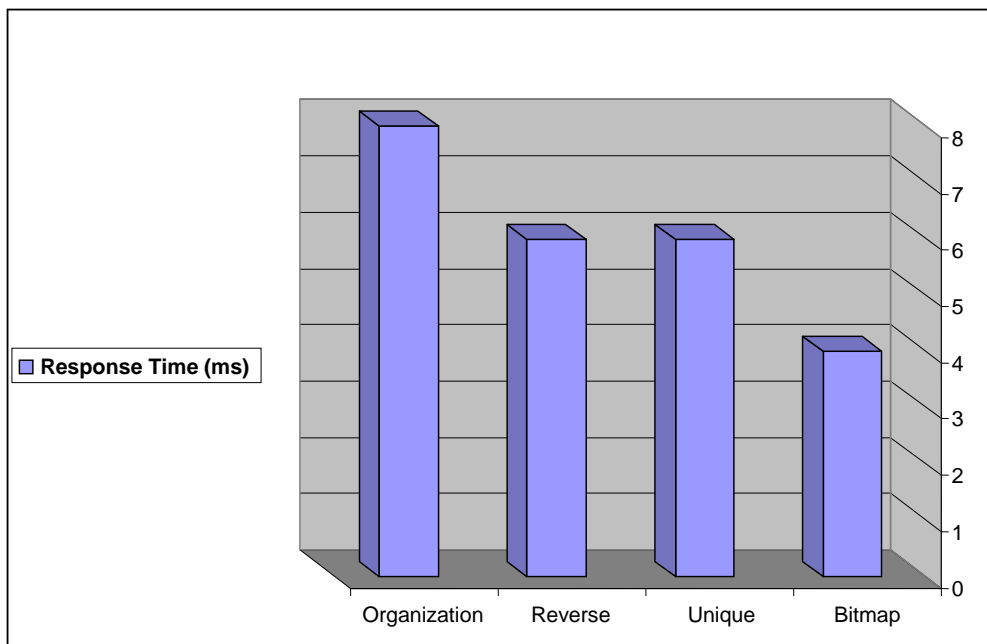


Figure 3.1: Full scan SELECT evaluation time for different indexing with 100K on Oracle

3.2.1.4 Test 1-d: Single-row SELECT performance with non-row

This test has been conducted on Oracle 10g platform with 100K as table size. The data attribute is STUD_ID in the SELECT statement. As seen in Table 3.1, the indexing techniques approximately have the same response time. The Bitmap index, B-tree and Reverse index take 2 ms to complete, whereas Organization index takes 3 ms.

Further testing results

According to the above analysis and Table 3.1, the Bitmap index on 100K has the optimal retrieving time and then it is better performance compared with the other indexing techniques. We have recommended using Bitmap index in the relational Databases.

Table 3.1 100k on Oracle

Indexing Technique	Scan Operation	Data Attribute	CPU Consumption (%) ⁱⁱⁱ	Cost ^{iv}	Response Time (ms)	# of rows retrieved
Bitmap	Full Scan	STU_ID	8	316	4	99996
Bitmap	Index Scan	STU_ID	0 {Big Fraction}	2	2	1
Bitmap	Index Scan	STU_ID	0	2	2	0
Bitmap	Full Scan	STU_GRADE	6	340	6	99996
Bitmap	Index Scan	STU_GRADE	0	2	2	19972
Bitmap	Index Scan	STU_GRADE	0	2	2	0
Unique (B-tree)	Full Scan	STU_ID	6	340	6	99996
Unique (B-tree)	Index Scan	STU_ID	0	5	3	1
Unique (B-tree)	Index Scan	STU_ID	0	5	3	0
Reverse	Full Scan	STU_ID	6	340	6	99996
Reverse	Index Scan	STU_ID	0	5	3	1
Reverse	Index Scan	STU_ID	0	2	2	0
Reverse	Full Scan	STU_Grade	6	340	6	99996
Reverse	Index Range Scan(i)	STU_Grade	0	5	3	19972
Reverse	Index Range Scan	STU_Grade	0	5	3	0
Organization	Index Fast Full Scan(ii)	STU_ID	4	552	8	99996
Organization	Index Unique Scan	STU_ID	0	4	2	0
Organization	Index Unique Scan	STU_ID	0	4	2	1

(i) INDEX RANGE SCAN: Retrieval of one or more rowids from an index. Indexed values are scanned in ascending order.

(ii) INDEX FULL SCAN: Retrieval of all rowids from an index when there is no start or stop key. Indexed values are scanned in ascending order.

(iii) CPU_COST (NUMERIC) CPU cost of the operation as estimated by the query optimizer's approach. The value of this column is proportional to the number of system cycles required for the operation. For statements that use the rule-based approach, this column is null.

(iv) COST (NUMERIC): Cost of the operation as estimated by the optimizer's query approach. Cost is not determined for table access operations. The value of this column does not have any particular unit of measurement; it is merely a weighted value used to compare costs of execution plans. The value of this column is a function of the CPU_COST and IO_COST columns.

3.2.2 Test 2: Oracle 10g and 1000K

3.2.2.1 Test 2-a: Single-row *SELECT* performance

The single-row select test has been conducted on Oracle 10g platform with 1000K as table size. It should be noted that the row size is 1K and the data attribute is STUD_ID in the SELECT statement. As shown in Table 3.2, the performance of Bitmap and Organization indexes are less costly compared with the other indexing techniques including B-tree (Unique index), Reverse index, and Organization index. The Bitmap index and Organization index consume less response time (2 ms) than B-tree (3 ms) and Reverse index (3 ms).

3.2.2.2 Test 2-b: Full scan *SELECT* performance

The full scan test has been performed on the data attribute STUD_ID in the SELECT statement. As shown in Table 3.2, the Bitmap, B-tree and Reverse indexes consume less response time (38 ms) than Organization index (64 ms). Figure 3.2 draws the variances of response times from 38 ms to 64 ms.

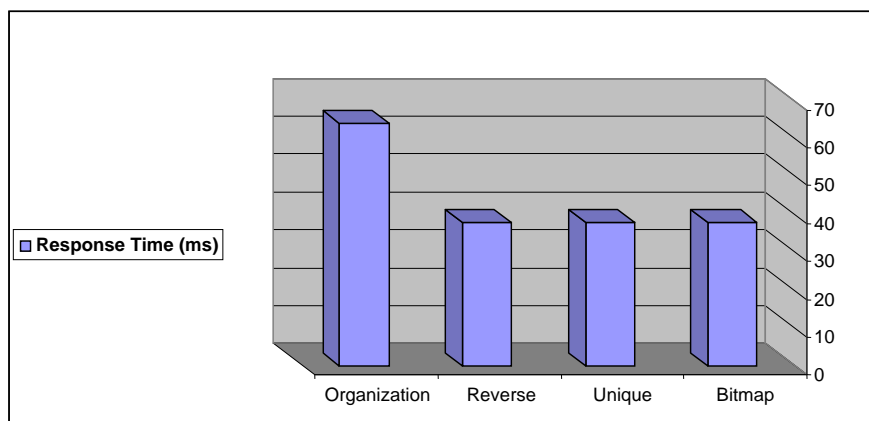


Figure 3.2: Full scan SELECT executions time for different indexing with 1000K on Oracle

3.2.2.3 Test 2-c: Single-row *SELECT* performance with non-row

As seen in Table 3.2, the indexing techniques have approximately the similar response time. The Bitmap index and Organization take 2 ms to complete, whereas B-tree and Reverse index take 3 ms.

Further testing results

In relation with the above analysis and Table 3.2, the Bitmap index on 1000K relatively has the best retrieving time and then better performance compared with the other indexing techniques. It is suggested to use Bitmap index in the relational Databases.

Table 3.2 1000k on Oracle

Indexing Technique	Scan Operation	Data Attribute	CPU Consumption (%)	Cost	Response Time (ms)	# of rows retrieved
Bitmap	Full Scan	STU_ID	8	3018	38	999731
Bitmap	Index Scan	STU_ID	0	6	2	1
Bitmap	Index Scan	STU_ID	0	6	2	0
Bitmap	Full Scan	STU_GRADE	8	3020	38	999731
Bitmap	Index Scan	STU_GRADE	8	3026	38	200720
Bitmap	Index Scan	STU_GRADE	8	3026	38	0
Unique (B-tree)	Full Scan	STU_ID	8	3008	38	999731
Unique (B-tree)	Index Scan	STU_ID	0	8	3	0
Unique (B-tree)	Index Scan	STU_ID	0	8	3	1
Reverse	Full Scan	STU_ID	8	3020	38	999731
Reverse	Index Scan	STU_ID	0	13	3	0

Reverse	Index Scan	STU_ID	0	13	3	1
Organization	Index Fast Full Scan	STU_ID	4	5272	64	999731
Organization	Index Unique Scan	STU_ID	0	4	2	0
Organization	Index Unique Scan	STU_ID	0	4	2	1

3.2.3 Test 3: Oracle 10g and 5000K

3.2.3.1 Test 3-a: Single-row *SELECT* performance

The single-row *SELECT* test has been conducted on Oracle 10g platform with 5000K as table size. As shown in Table 3.3 the performance of Organization index is less costly compared with the other indexing techniques including B-tree (Unique index), Reverse index, and Bitmap index. The Organization index consumes less response time (2 ms) than B-tree (3 ms), Reverse index (3 ms) and Bitmap index (3 ms).

3.2.3.2 Test 3-b: Range *SELECT* performance

In this test, we have performed a range *SELECT* experiment on STUD_GRADE attribute. This attribute has five values A, B, C, D and F. Table 3.3 illustrates that the Bitmap index and Reverse index have the same response time (1840 ms).

3.2.3.3 Test 3-c: Full scan *SELECT* performance

The full scan test has been conducted on Oracle 10g platform with 5000K as table size. As shown in Table 3.3, the performance of B-tree and Reverse indexes are relatively the best compared with the other indexing techniques including Bitmap, Organization indexes. The B-tree and Reverse indexes consume less response time (1840 ms) than Bitmap (1880 ms) and Organization index (3220 ms).

Another full scan test has been conducted on Oracle 10g platform with 5000K as table size. The data attribute is STUD_GRADE in the SELECT statement. Figure 3.3 shows the full scan SELECT performance with 5000K on Oracle 10g. As shown in Table 3.3 and Figure 3.3, the performance of Bitmap index is the same compared with the Reverse index. The Bitmap index and Reverse index consume the same response time (1840 ms).

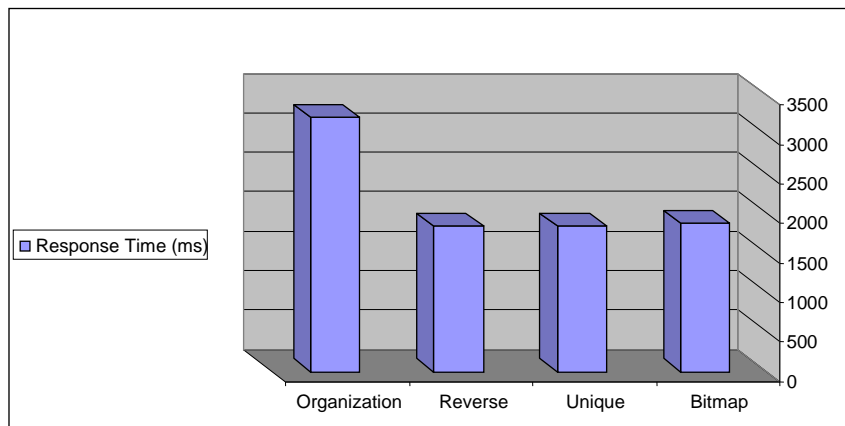


Figure 3.3: Full scan SELECT executions time for different indexes with 5000K on Oracle 10g

3.2.3.4 Test 3-d: Single-row SELECT performance with non-row

This test has been conducted on Oracle 10g platform with 5000K as table size. The data attribute is STUD_ID in the SELECT statement. As seen in Table 3.3, the indexing techniques (such as Bitmap index, B-tree, Reverse index and Organization index) approximately have the same response time (3 ms).

Further testing results

According to the above analysis and Table 3.3, the performance of Bitmap index, B-tree and Reverse index are much better than Organization index especially at full scan select performance. We do not recommend using Organization index in the relational Databases.

Table 3.3 5000k on Oracle

Indexing Technique	Scan Operation	Data Attribute	CPU Consumption (%)	Cost	Response Time (ms)	# of Rows Retrieved
Bitmap	Full Scan	STU_ID	8	15610	1880	4994346
Bitmap	Index Scan	STU_ID	0	10	3	1
Bitmap	Index Scan	STU_ID	0	10	3	0
Bitmap	Full Scan	STU_GRADE	8	15280	1840	4994346
Bitmap	Index Scan	STU_GRADE	8	15310	1840	998391
Bitmap	Index Scan	STU_GRADE	0	2	2	0
Unique (B-tree)	Full Scan	STU_ID	8	15280	1840	4994346
Unique (B-tree)	Index Scan	STU_ID	0	8	3	1
Unique (B-tree)	Index Scan	STU_ID	0	8	3	0
Reverse	Full Scan	STU_ID	8	15280	1840	4994346
Reverse	Index Range Scan	STU_ID	0	11	3	1
Reverse	Index Range Scan	STU_ID	0	2	3	0
Reverse	Full Scan	STU_Grade	8	15280	1840	4994346
Reverse	Index Range Scan	STU_Grade	0	11	3	0
Reverse	Index Range Scan	STU_Grade	8	15310	1840	998391
Organization	Index Fast Full Scan	STU_ID	2	26694	3220	5001165
Organization	Index Unique Scan	STU_ID	0	2	2	1
Organization	Index Unique Scan	STU_ID	0	2	3	0

3.2.4 Test 4: MS-SQL Server and 100K

3.2.4.1 Test 4-a: Single-row *SELECT* performance

The single-row *SELECT* test has been conducted on MS-SQL Server with 100K as table size. As shown in Table 3.4, the Primary Key Clustered and B-tree (Unique Non-clustered) consume the same response time (0.1 ms).

It should be noted that the *SELECT* statement with the clustered index, which is based on the predictions on the indexed columns, results in an index seek operation, which then gives the data values requested. While the related *SELECT* statement without a clustered index results in a seek operation using the non-clustered index, followed by nested loop join with the table in order to extract the columns not in the index definition (assuming the index is not a covering index for the given query), which then gives the requested row.

A *SELECT* statement requires the lookup for one or more rows from a table. In the table with a clustered index, the DBMS engine performs a Clustered Index Seek operation into the table and yields the requested data row, as shown in the query execution plan in Figure 3.4.

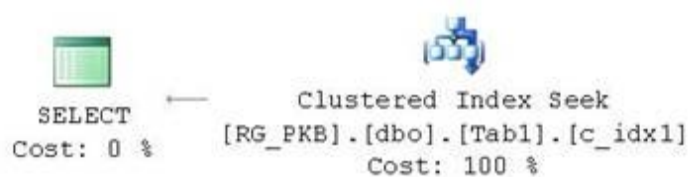


Figure 3.4: Query execution plan for *SELECT* statement on table with clustered index

3.2.4.2 Test 4-b: Full scan SELECT performance

The full scan test has been conducted on MS-SQL Server with 100K as table size. As shown in Table 3.4, the performance of Primary Key Clustered is relatively the faster than B-tree (Unique Non-clustered). The Primary Key Clustered consumes less response time (15 ms) than B-tree (16 ms).

3.2.4.3 Test 4-c: Single-row SELECT performance with non-row

As illustrated in Table 3.4, the Primary Key Clustered and B-tree (Unique Non-clustered) consume the same response time (0.4 ms).

Further testing results

According to the above analysis and Table 3.4, the performance of Primary Key Clustered is relatively better than B-tree (Unique Non-clustered) especially at full scan SELECT performance.

Table 3.4 MS SQL Server on 100K

Indexing Technique	Scan Operation	Data Attribute	CPU cost(%)	Operator Cost	Response Time (ms)	# of Rows Retrieved
Primary Key Clustered	Full Scan	STU_ID	11	69	15	99996
Primary Key Clustered	Index Scan	STU_ID	1	1	0.1	1
Primary Key Clustered	Index Scan	STU_ID	1	3	0.4	0
Unique (B-tree) Non-Clustered	Full Scan	STU_ID	11	69	16	99996
Unique (B-tree) Non-Clustered	Index Scan	STU_ID	1	1	0.1	1
Unique (B-tree) Non-Clustered	Index Scan	STU_ID	1	1	0.4	0

3.2.5 Test 5: MS SQL Server and 1000K

3.2.5.1 Test 5-a: Single-row SELECT performance

The single-row SELECT test has been conducted on MS-SQL Server with 1000K as table size. As shown in Table 3.5, the Primary Key Clustered and B-tree (Unique Non-clustered) consume the same response time (0.4 ms).

3.2.5.2 Test 5-b: Full scan SELECT performance

The full scan test has been conducted on MS-SQL Server with 1000K as table size. As shown in Table 3.5, the performance of Primary Key Clustered is relatively faster than B-tree (Unique Non-clustered). The Primary Key Clustered consumes less response time (11 ms) than B-tree (14 ms).

3.2.5.3 Test 5-c: Single-row SELECT performance with non-row

As illustrated in Table 3.5, the Primary Key Clustered and B-tree (Unique Non-clustered) consume the same response time (0.5 ms).

Further testing results

According to the above analysis and Table 3.5, the performance of Primary Key Clustered is relatively better than B-tree (Unique Non-clustered) especially at full scan SELECT performance.

Table 3.5: MS-SQL Server on 1000K

Indexing Technique	Scan Operation	Data Attribute	CPU cost(%)	Operator Cost	Response Time (ms)	# of Row Retrieved
Primary Key Clustered	Full Scan	STU_ID	110	688	11	999999
Primary Key Clustered	Index Scan	STU_ID	1	1	0.4	1
Primary Key Clustered	Index Scan	STU_ID	1	1	0.5	0
Unique (B-tree) Non-Clustered	Full Scan	STU_ID	110	688	14	999999
Unique (B-tree) Non-Clustered	Index Scan	STU_ID	1	1	0.4	1
Unique (B-tree) Non-Clustered	Index Scan	STU_ID	1	1	0.5	0

3.2.6 Test 6: MS SQL Server and 5000K

3.2.6.1 Test 6-a: Single-row *SELECT* performance

The single-row *SELECT* test has been conducted on MS-SQL Server with 5000K as table size. As shown in Table 3.6, the Primary Key Clustered and B-tree (Unique Non-clustered) consume the same response time (15 ms). The Unique Clustered consumes (16 ms) to complete the scan.

In non-clustered index, the data row has first to be located by using an Index Seek operation with the non-clustered index, followed by Nested Loops with a RID Lookup to extract the set of selected columns that are not a part of the non-clustered index, as illustrated in Figure 3.5.

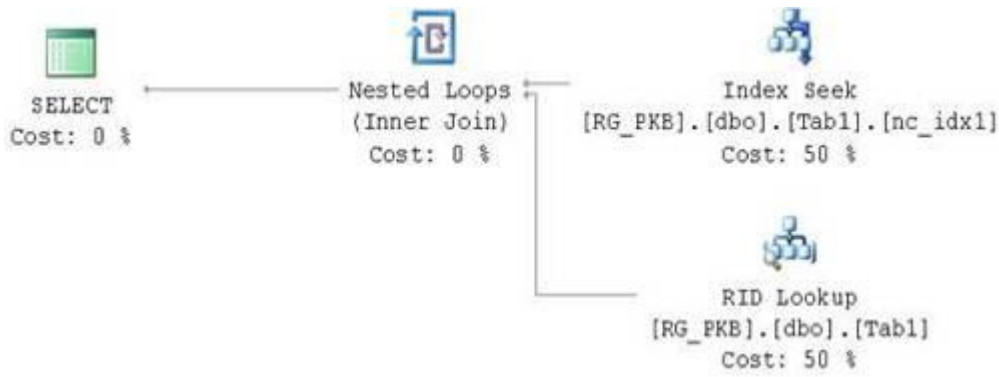


Figure 3.5: Query execution plan for SELECT statement on table with non-clustered index in MS-SQL Server

3.2.6.2 Test 6-b: Full scan SELECT performance

The full scan test has been conducted on MS SQL Server with 5000K as table size. As shown in Table 3.6, the performance of Primary Key Clustered is relatively faster than B-tree (Unique Non-clustered) and Unique Clustered. The Primary Key Clustered consumes less response time (16 ms) than B-tree and Unique Clustered (17 ms).

3.2.6.3 Test 6-c: Single-row SELECT performance with non-row

As illustrated in Table 3.6, the Primary Key Clustered, Unique Clustered and B-tree (Unique Non-clustered) consume the same response time (18 ms).

Further testing results

According to the above analysis and Table 3.6, the performance of Primary Key Clustered is relatively better than B-tree (Unique Non-clustered) and Unique Clustered especially at full scan SELECT performance.

Table 3.6: MS-SQL Server on 5000K

Indexing Technique	Scan Operation	Data Attribute	CPU cost(%)	Operator Cost	Response Time (ms)	# of Rows Retrieved
Primary Key Clustered	Full Scan	STU_ID	560	351	16	5099800
Primary Key Clustered	Index Scan	STU_ID	1	1	15	1
Primary Key Clustered	Index Scan	STU_ID	1	1	18	0
Unique (B-tree) Non-Clustered	Full Scan	STU_ID	560	378	17	5099800
Unique (B-tree) Non-Clustered	Index Scan	STU_ID	1	1	15	1
Unique (B-tree) Non-Clustered	Index Scan	STU_ID	1	1	18	0
Unique (B-tree) Clustered	Full Scan	STU_ID	560	369	17	5099800
Unique (B-tree) Clustered	Index Scan	STU_ID	1	1	16	1
Unique (B-tree) Clustered	Index Scan	STU_ID	1	1	18	0

3.3 Recommendations and Further Analysis

It is very difficult to compare between two different platforms: Oracle and MS-SQL Server because we do not know the background process for each platform. However, we have attempted to run our experiment on the identical environment to achieve the possible encouraging results.

It is clear from the empirical results that the performance of indexing techniques in MS-SQL Server is much faster than Oracle 10g as shown in Figure 3.6, 3.7 and 3.8. Therefore, using the index in the SELECT statement over MS-SQL Server is much less costly in terms of I/O operations, CPU consumptions, and response time than Oracle. However, Figure 3.8 shows that the B-tree index in the SELELCT statement over Oracle is faster in terms of performance than MS-SQL Server in the following cases:

- Index scan SELECT performance when retrieved single row.
- Index scan SELECT performance when retrieved non-row.

A number of technical reports and studies indicate that using index in the retrieval systems over Oracle platform consumes greater response time than MS-SQL Server. Thus, those studies support the results in this thesis.

It should be noted that the B-tree is the common between Oracle and MS-SQL Server. As shown in Table 3.7-A and Table 3.7-B, the B-tree is the best indexing technique and it is more effective on huge data. The results of Bitmap index in Oracle are interesting and significant and unfortunately, this index is not supported by MS-SQL Server.

Table 3.7-A: A summary of the best indexing techniques with different data sizes on Oracle

Data Size	Best Indexing Techniques	Comments
0-100K	Bitmap	Low Cardinality ⁽ⁱ⁾ Medium Cardinality ⁽ⁱⁱ⁾
100-1000K	Bitmap, B-tree	Low Cardinality Medium Cardinality High Cardinality ⁽ⁱⁱⁱ⁾
1000-5000K	Bitmap, B-tree	Low Cardinality Medium Cardinality High Cardinality

Table 3.7-B: A summary of the best indexing techniques with different data sizes on MS SQL Server.

Data Size	Best Indexing Techniques	Comments
0-100K	Non clustered index	Low Cardinality Medium Cardinality
100-1000K	Primary clustered index	Medium Cardinality High Cardinality
1000-5000K	Primary clustered index	Medium Cardinality High Cardinality

(i) Low-cardinality refers to columns with few unique values.

(ii) Medium-cardinality refers to columns with values that are somewhat uncommon. Medium-cardinality column data values such as names, street addresses, or vehicle types

(iii) High-cardinality refers to columns with values that are very uncommon or unique.

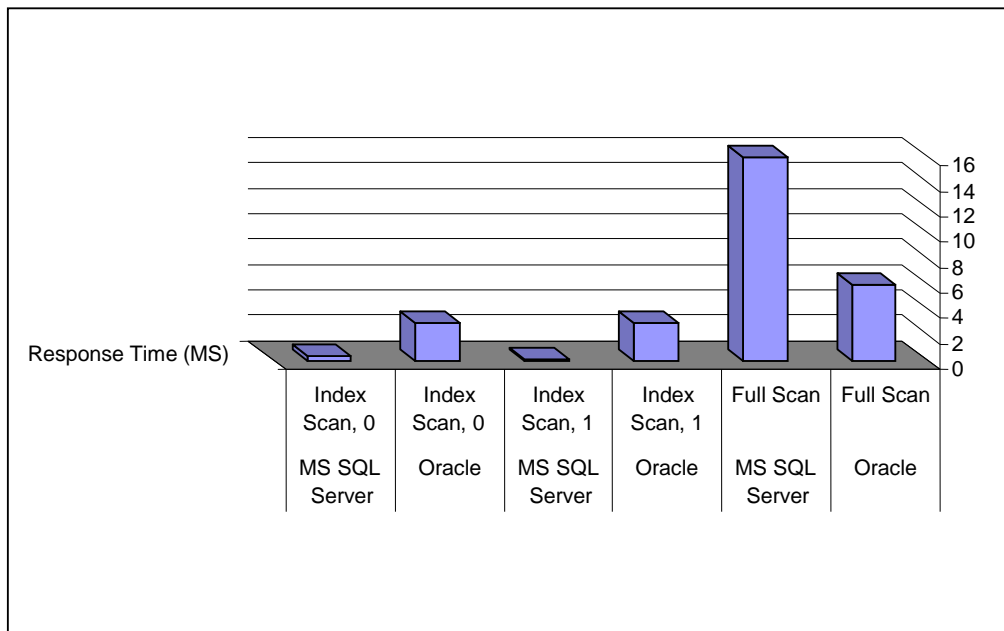


Figure 3.6: Comparison between the response times (ms) of B-tree over Oracle and MS-SQL Server on 100K

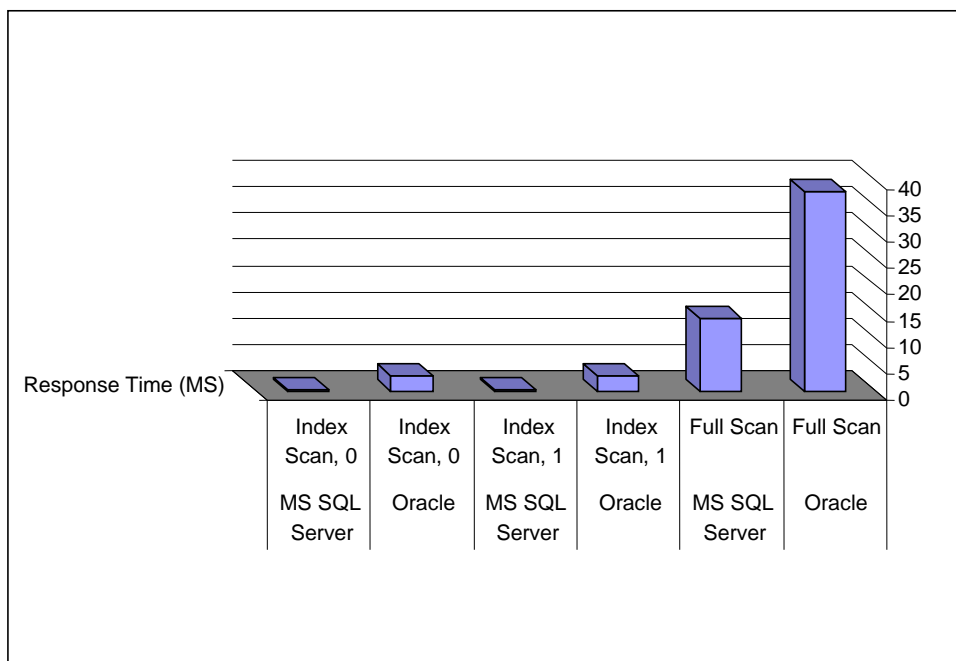


Figure 3.7: Comparison between the response times (ms) of B-tree over

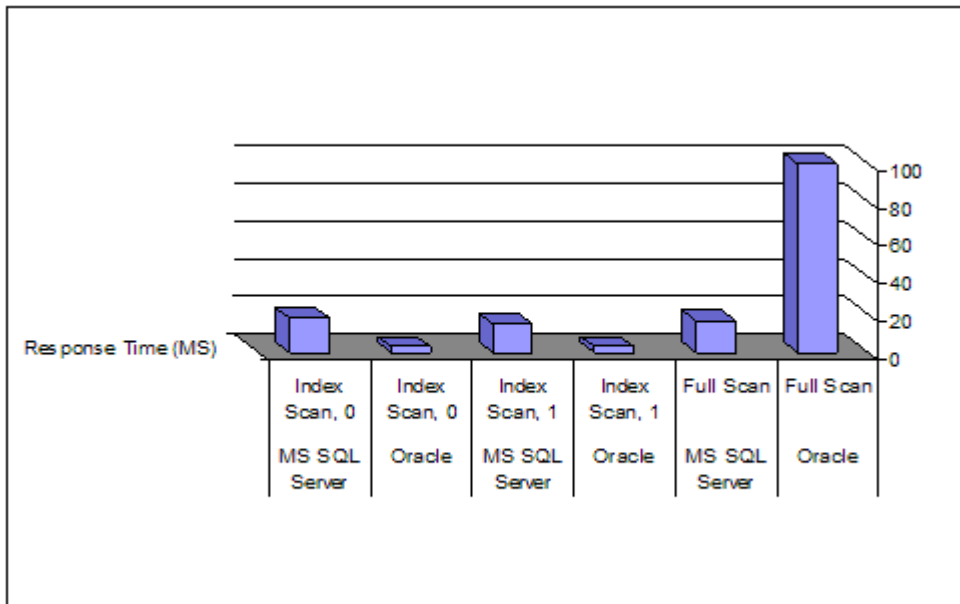


Figure 3.8: Comparison between the response times (ms) of B-tree over Oracle and MS-SQL Server on 5000K

3.4 Test: INSERT Performance

In this section, we have described the effects of INSERT statement on the Oracle platform with 100K, 1000K and 5000K. One row has been added on the database that contains 100K and 1000K with B-tree index technique.

First, we have compiled the following statement on Oracle engine with 100K:

```
insert into student100k (stu_id ,stu_name , stu_gender , stu_address ,stu_seq)
values ('3748528','amman822195','f','meu92289','202255') ;
```

This statement inserts one row on the database. As a result, figure 3.9 shows a snapshot of the execution result. This result ensures that one row will not have an effect on the performance because the response time is zero ms.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL> insert into student100k (stu_id ,stu_name , stu_gender , stu_address ,stu_seq)
  2  values ('3748558','amman837495','f','meu3948489','2304955');
1 row created.

Execution Plan
-----
| Id | Operation          | Name | Rows | Bytes | Cost | (%CPU) | Time |
|----|-----|-----|-----|-----|-----|-----|-----|
|  0 | INSERT STATEMENT   |      |    1 |    42 |    1 |    (0) | 00:00:01 |

Statistics
-----
      17 recursive calls
      13 db block gets
       5 consistent gets
      11 physical reads
     976 redo size
     664 bytes sent via SQL*Net to client
     667 bytes received via SQL*Net from client
        3 SQL*Net roundtrips to/from client
        2 sorts (memory)
        0 sorts (disk)
        1 rows processed

SQL> insert into student1000k (stu_id ,stu_name , stu_gender , stu_address ,stu_seq)
  2  values ('3748558','amman837495','f','meu3948489','2304955');
1 row created.

```

Figure 3.9: Execution result after INSERT one row using B-tree index with 100K

Furthermore, we have compiled the above INSERT statement on 1000K. Figure 3.10 illustrates that there is no effect on the database performance because the response time is too small.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL> insert into student1000k (stu_id ,stu_name , stu_gender , stu_address ,stu_seq)
  2  values ('3748558','amman837495','f','meu3948489','2304955');
1 row created.

Execution Plan
-----
| Id | Operation          | Name | Rows | Bytes | Cost | (%CPU) | Time |
|----|-----|-----|-----|-----|-----|-----|-----|
|  0 | INSERT STATEMENT   |      |    1 |    42 |    1 |    (0) | 00:00:01 |

Statistics
-----
     311 recursive calls
       12 db block gets
       61 consistent gets
       11 physical reads
    4592 redo size
     665 bytes sent via SQL*Net to client
     668 bytes received via SQL*Net from client
        3 SQL*Net roundtrips to/from client
        9 sorts (memory)
        0 sorts (disk)
        1 rows processed

SQL> insert into student5000k (stu_id ,stu_name , stu_gender , stu_address ,stu_seq)
  2  values ('3748558','amman837495','f','meu3948489','2304955')
  3  ;
1 row created.

```

Figure 3.10: Execution result after INSERT one row using B-tree index with 1000K

Further test, we have conducted the same INSERT statement on 5000K. As a result, figure 3.11 shows no change happened to the database performance.

```

Oracle SQL*Plus
File Edit Search Options Help

      3  SQL*Net roundtrips to/from client
      9  sorts (memory)
      0  sorts (disk)
      1  rows processed

SQL> insert into student5000k (stu_id,stu_name , stu_gender , stu_address ,stu_seq)
  2  values ('3748558','amman837495','f','neu3948489','2304955')
  3  ;

1 row created.

Execution Plan
-----
| Id | Operation          | Name | Rows | Bytes | Cost (%CPU)| Time |
|----|-----|-----|-----|-----|-----|-----|
|  0 | INSERT STATEMENT    |      |     1 |    43 |        1 (0)| 00:00:01 |

Statistics
-----
      273  recursive calls
        6  db block gets
       42  consistent gets
        7  physical reads
       536  redo size
      666  bytes sent via SQL*Net to client
      669  bytes received via SQL*Net from client
        3  SQL*Net roundtrips to/from client
        8  sorts (memory)
        0  sorts (disk)
        1  rows processed

SQL>

```

Figure 3.11: Execution result after INSERT one row using B-tree index with 5000K

3.5 Test: Select Performance using Composite Key in Oracle

In this test, we have conducted several experiments on Oracle with data size: 1000K through Select statement that contains composite key (STUD_ID, STUD_NAME).

Table 3.8: Select Performance using Composite Key in Oracle 1000k

Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of rows retrieved
Bitmap	Full Scan	STUD_ID, STUD_NAME	38	999731
Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1
Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	38	999731
Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Reverse	Full Scan	STUD_ID, STUD_NAME	38	999731
Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	66	999731
Organization	Index Unique Scan	STUD_ID, STUD_NAME	3	1
Organization	Index Unique Scan	STUD_ID, STUD_NAME	3	0

The single-row select test has been conducted on Oracle 10g platform with 1000K as table size. Note that we have used composite key (STUD_ID,STUD_NAME) .As shown in Table 3.8, the performance of Bitmap, B-tree index and Organization indexes are less costly compared with the Reverse index. The Bitmap index, Reverse index and B-tree consume less response time (38 ms) than Organization index (66 ms).

In case of non-row retrieved data and in retrieving single row, Table 3.8, the indexing techniques have approximately the similar response time. The Bitmap index takes 2 ms to complete, whereas B-tree, Organization index and Reverse index take 3 ms.

Further testing results

In relation with the above analysis and Table 3.8, the Bitmap index on 1000K relatively has the best retrieving time and then better performance compared with the other indexing techniques.

3.6 Test: Select Performance using Composite Key in MS SQL Server

Table 3.9: Select Performance using Composite Key in MS SQL Server 1000K

Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	33	999999
Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	1
Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.5	0
Unique (B-tree) Non-Clustered	Full Scan	STUD_ID, STUD_NAME	10	999999
Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	1
Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	0.5	0

The single-row SELECT test has been conducted on MS-SQL Server with 1000K using composite key. As shown in Table 3.9, the Primary Key Clustered and B-tree (Unique Non-clustered) consume the same response time (0.4 ms).

On the other hand, the full scan test has been conducted on MS-SQL Server. Table 3.9 shows that the performance of B-tree (Unique Non-clustered) is relatively faster than the Primary Key Clustered. The Primary Key Clustered consumes more response time (33 ms) than B-tree (10 ms).

The final test has been conducted on composite key when the SELECT does not retrieve any data. As illustrated in Table 3.9, the Primary Key Clustered and B-tree (Unique Non-clustered) consume the same response time (0.5 ms).

Further testing results

According to the above analysis and Table 3.5, the performance of B-tree (Unique Non-clustered) is relatively better than Primary Key Clustered especially at full scan SELECT performance.

3.7 Test: Insert Performance using Composite Key in Oracle and MS SQL Server with 1000K

After inserting one row in the table, it is clear from Table 3.10 that the Primary Key Clustered index in MS SQL Server is the best one. This index consumes only 9 ms for full scan using composite key. On the other hand, the Bitmap index is the best in the Oracle Platform for all scan operations including Full scan, Index Scan with one row returned and Index scan without rows returned.

Table 3.10: Insert on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	9	1000001
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.5	0
MS SQL Server	Unique (B-tree) Non-Clustered	Full Scan	STUD_ID, STUD_NAME	26	1000001
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	1
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	0.5	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	38	1000001
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	38	1000001
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	38	1000001
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	68	1000001

Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	1
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.8 Test: Update Performance using Composite Key in Oracle and MS SQL Server with 1000K

After the Update operation applied on one row, there is not any change for the performance values, especially in the Oracle Platform. Bitmap, B-tree and Reverse indexes takes 38 ms for full scan. In the MS SQL Server, a little change happened for the performance after applying Primary Key Clustered index. Table 3.11 shows all the values with different scan operation.

Table 3.11: Update on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	12	1000000
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.5	0
MS SQL Server	Unique (B-tree) Non- Clustered	Full Scan	STUD_ID, STUD_NAME	9	1000000
MS SQL Server	Unique (B-tree) Non- Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	1
MS SQL Server	Unique (B-tree) Non- Clustered	Index Scan	STUD_ID, STUD_NAME	0.5	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	38	1000000
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1
Oracle	Bitmap	Index Scan	STUD_ID,	2	0

			STUD_NAME		
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	38	1000000
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	38	1000000
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	68	1000000
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	1
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.9 Test: Delete Performance using Composite Key in Oracle and MS SQL Server with 1000K

Table 3.12 shows that the Delete performance using composite key in Oracle and MS SQL Server platforms. Indeed, there is no change for the performance values. This means deleting one row from the table, but has no effect on the performance. Bitmap index and others indexes take 38 ms in Oracle Platform.

Table 3.12: Delete on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	10	999999
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.5	0
MS SQL Server	Unique (B-tree) Non- Clustered	Full Scan	STUD_ID, STUD_NAME	10	999999
MS SQL Server	Unique (B-tree) Non- Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	1
MS SQL Server	Unique (B-tree) Non- Clustered	Index Scan	STUD_ID, STUD_NAME	0.5	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	38	999999
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	38	999999
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	38	999999
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	68	999999
Oracle	Organization	Index Unique	STUD_ID,	2	1

		Scan	STUD_NAME		
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.10 Test: Select Performance using Composite Key in Oracle and MS SQL Server with 100K

As shown in Table 3.13, Primary key Clustered and B-tree have the same response time (7 ms) in the MS SQL Server. On the other hand, Bitmap index in Oracle platform is the best for all scans, whereas the Organization scan takes the longest time (8 ms). It is clear from the table that Bitmap and B-tree are recommended for use in the Oracle and MS SQL platform.

Table 3.13: Select on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	7	100000
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.1	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	0
MS SQL Server	Unique (B-tree) Non-Clustered	Full Scan	STUD_ID, STUD_NAME	7	100000
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	4	1
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	4	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	4	100000
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1

Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	4	100000
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	4	100000
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	8	100000
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	1
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.11 Test: Insert Performance using Composite Key in Oracle and MS SQL Server with 100K

In Insert operation using composite key for MS SQL Server and Oracle, no change happened for performance. This confirms that inserting one row will not affect the performance, as shown in Table 3.14.

Table 3.14: Insert on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	7	100001
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.1	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	0
MS SQL Server	Unique (B-tree) Non-Clustered	Full Scan	STUD_ID, STUD_NAME	7	100001
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	6	1
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	4	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	4	100001
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	4	100001
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	4	100001
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	8	100001

Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	1
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.12 Test: Update Performance using Composite Key in Oracle and MS SQL Server with 100K

Table 3.15 illustrates no change took place for the performance. Updating one record on the table does not make sense on the performance values.

Table 3.15: Update on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	7	100000
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.1	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	0
MS SQL Server	Unique (B-tree) Non- Clustered	Full Scan	STUD_ID, STUD_NAME	7	100000
MS SQL Server	Unique (B-tree) Non- Clustered	Index Scan	STUD_ID, STUD_NAME	6	1
MS SQL Server	Unique (B-tree) Non- Clustered	Index Scan	STUD_ID, STUD_NAME	4	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	4	100000
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1

Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	4	100000
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	4	100000
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	8	100000
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	1
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.13 Test: Delete Performance using Composite Key in Oracle and MS SQL Server with 100K

As shown in Table 3.16, only one change happened for the value of the Primary Key Clustered index in the MS SQL Server. On the other hand, all indexes keep the values which have shown the above tests.

Table 3.16: Delete on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	1	99999
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.1	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	0
MS SQL Server	Unique (B-tree) Non-Clustered	Full Scan	STUD_ID, STUD_NAME	7	99999
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	6	1
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	4	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	4	99999
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	4	99999
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	4	99999
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	8	99999
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	1
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.14 Test: Select Performance using Composite Key in Oracle and MS SQL Server with 5000K

In Table 3.17, B-tree is the best for the MS SQL Server and Oracle. This index consumes 49 ms in MS SQL Server and 184 ms in Oracle platform. We are not recommended to use the organization index in the Oracle platform. This index takes 336 ms in Oracle. Further details are shown in Table 3.17.

Table 3.17: Select on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	53	5000000
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	1	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	0
MS SQL Server	Unique (B-tree) Non-Clustered	Full Scan	STUD_ID, STUD_NAME	49	5000000
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	1	1
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	0.4	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	184	5000000
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	184	5000000
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	184	5000000
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0

Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	336	5000000
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	1
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.15 Test: Insert Performance using Composite Key in Oracle and MS SQL Server with 5000K

Table 3.18 shows a very a little bit change in MS SQL Server. This change does not have an effect on the performance values.

Table 3.18: Insert on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	53	5000001
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	6	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	6	0
MS SQL Server	Unique (B-tree) Non- Clustered	Full Scan	STUD_ID, STUD_NAME	57	5000001
MS SQL Server	Unique (B-tree) Non- Clustered	Index Scan	STUD_ID, STUD_NAME	8	1
MS SQL Server	Unique (B-tree) Non- Clustered	Index Scan	STUD_ID, STUD_NAME	6	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	184	5000001
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	184	5000001
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	184	5000001

Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	336	5000001
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	1
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.16 Test: Update Performance using Composite Key in Oracle and MS SQL Server with 5000K

As illustrated in Table 3.19, the only change happened in MS SQL Server. The Primary Key Clustered index takes 142 ms. On the other hand; the values of indexes in the Oracle take 184 ms.

Table 3.19: Update on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	142	5000000
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	6	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	6	0
MS SQL Server	Unique (B-tree) Non- Clustered	Full Scan	STUD_ID, STUD_NAME	49	5000000
MS SQL Server	Unique (B-tree) Non- Clustered	Index Scan	STUD_ID, STUD_NAME	1	1
MS SQL Server	Unique (B-tree) Non- Clustered	Index Scan	STUD_ID, STUD_NAME	1	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	184	5000000
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1

Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	184	5000000
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	184	5000000
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	334	5000000
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	1
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.17 Test: Delete Performance using Composite Key in Oracle and MS SQL Server with 5000K

No change happened here in delete operation.

Table 3.20: Delete on MS SQL Server and Oracle with 1000K

Platform	Indexing Technique	Scan Operation	Data Attribute (Composite Key)	Response Time (ms)	# of Row Retrieved
MS SQL Server	Primary Key Clustered	Full Scan	STUD_ID, STUD_NAME	59	4999999
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	6	1
MS SQL Server	Primary Key Clustered	Index Scan	STUD_ID, STUD_NAME	6	0
MS SQL Server	Unique (B-tree) Non-Clustered	Full Scan	STUD_ID, STUD_NAME	57	4999999

MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	10	1
MS SQL Server	Unique (B-tree) Non-Clustered	Index Scan	STUD_ID, STUD_NAME	6	0
Oracle	Bitmap	Full Scan	STUD_ID, STUD_NAME	184	4999999
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	1
Oracle	Bitmap	Index Scan	STUD_ID, STUD_NAME	2	0
Oracle	Unique (B-tree)	Full Scan	STUD_ID, STUD_NAME	184	4999999
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Unique (B-tree)	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Reverse	Full Scan	STUD_ID, STUD_NAME	184	4999999
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	1
Oracle	Reverse	Index Scan	STUD_ID, STUD_NAME	3	0
Oracle	Organization	Index Fast Full Scan(ii)	STUD_ID, STUD_NAME	334	4999999
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	1
Oracle	Organization	Index Unique Scan	STUD_ID, STUD_NAME	2	0

3.18 Guidelines to Select Indexing Techniques

This section will describe the guidelines to select indexing techniques which helps the DBAs to select indexing techniques that are suitable for databases.

3.18.1 Using Decision Table to Select the Indexing for the DBAs:

1. Decision Tables

Decision tables are precise and compact way to model complicated logic. They are ideal for describing situations in which a number of combinations of actions are taken under varying sets of conditions (Fisher, D.L. 1966). The benefit of using the decision table is to make it easy to observe all possible conditions. Decision tables are also used to analyze a problem. The conditions applying in the particular problems are set out, and the actions to be taken as a result of any combination of the conditions arising, are shown below in this section.

2. Cardinality

The word cardinality is an SQL term to refer to the uniqueness of a data value in a particular column in a database table (*Burleson* 2010). Low cardinality means that the values in the data column of the data table are pretty common. For example, data value such as gender, race, age, hair color or Boolean data represents a kind of low cardinality. On the other hand, Medium-cardinality means that the columns with data values are somewhat uncommon. Normal-cardinality or medium- cardinality column data values include, names, street addresses, or vehicle types. The last type of cardinality a high cardinality data may refer to data in a column which are unique. That data may include identification numbers, user telephones number; email addresses or social security number.

Example Cardinality Attribute

The following samples on the STUDENT table demonstrate the variety of query-processing techniques that are necessary for optimal performance.

```
CREATE TABLE UI.STUDENT_index100K  
  
(  
  STU_ID    NUMBER(10),  
  STU_NAME  VARCHAR2(50 ),  
  STU_GRADE VARCHAR2(1 ),  
  STU_ADDRESS VARCHAR2(100 ),  
  STU_SEQ   NUMBER(10),  
  CONSTRAINT STUDENT_index100K_PK  
  PRIMARY KEY  
  (STU_ID)  
)
```

Example 1: High-Cardinality Attribute

```
select * from student100k where STU_ID = 974166938;
```

Example 2: Medium-Cardinality Attributes

```
select * from student100k where STU_NAME='meu1253398454';
```

Example 3: Low-Cardinality Attributes

```
select * from student100k where STU_GRADE = 'A' ;
```

In the following section a guidelines to select indexing techniques that decision table to help the DBAs to select indexing techniques suited for their databases on MS-SQL Server environment and ORACLE environment:

3.18.1.1 Oracle Environment:

Scenario: A DBA wishes to construct a decision table to decide how to select an indexing techniques to three characteristics: Data size: A (under 100,000), B (between 100,000 and 1,000,000), and C (over 1,000,000). Cardinality type: L (Low cardinality), M (Medium cardinality), and H (High cardinality), and Columns: S (Single columns) and C (Combine

columns). The DBA has two indexes (X,Y) to select. Index X will appeal to Bitmap index. Index Y will appeal to B-Tree index.

1. Identify Conditions & Values

The three data attributes tested by the conditions in this problem are

Data size: with values A, B and C; Cardinality L, M and H; and Columns: S and C as stated in the problem.

2. Compute Maximum Number of Rules

The maximum number of rules is $3 \times 3 \times 2 = 18$.

3. Identify Possible Actions

The two actions are: Bitmap index X, B-Tree index Y.

4. Enter All Possible Rules

The top of the table would look as follows: Note that all combinations of values are present.

Table 3.21: Possible Rules

Process	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Data size	A	A	A	A	A	A	B	B	B	B	B	B	C	C	C	C	C	C
Cardinality	L	L	M	M	H	H	L	L	M	M	H	H	L	L	M	M	H	H
Columns	S	C	S	C	S	C	S	C	S	C	S	C	S	C	S	C	S	C

5. Define Actions for each Rule

The bottom of the table would look as follows:

Table 3.22: Actions for each rule

Indexes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
X	X	X	X	X			X	X	X	X			X	X				
Y					X	X					X	X			X	X	X	X

6. Simplify the table

The revised table is as follows:

Table 3.23: Final simplified table of rules and actions on ORACLE environment

Proccess	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Datasize	A	A	A	A	A	A	B	B	B	B	B	B	C	C	C	C	C	C
Cardinality	L	L	M	M	H	H	L	L	M	M	H	H	L	L	M	M	H	H
Columns	S	C	S	C	S	C	S	C	S	C	S	C	S	C	S	C	S	C
X (Bitmap index)	X	X	X	X			X	X	X	X			X	X				
Y (B-Tree)					X	X					X	X			X	X	X	X

3.18.1.2 MS-SQL Server Environment:

In MS-SQL Server will do the same steps on section 3.18.1.1 on Oracle environments except step 3. That Identify Possible Actions that will be the two actions are: Clustered index X, Non-Clustered index Y.

Table 3.24: Final simplified table of rules and actions on MS-SQL Server environment.

Process	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Datasize	A	A	A	A	A	A	B	B	B	B	B	B	C	C	C	C	C	C
Cardinality	L	L	M	M	H	H	L	L	M	M	H	H	L	L	M	M	H	H
Columns	S	C	S	C	S	C	S	C	S	C	S	C	S	C	S	C	S	C
X(Clustered index)	X		X		X	X	X	X	X	X	X	X	X	X	X	X	X	X
Y (Non-Clustered index)		X		X														

The performance benefits of having a clustered index on a table outweigh the negatives for our sample database table. For the case where the performance was lower (SELECT statements Test4, Test5, Test6), the difference was insignificant. Given this, we recommend creating a clustered index on all SQL Server user tables.

3.18.2 Descriptive Guidelines and Advices for the DBAs

There are several aspects which affect the DBAs performance. However, to select the best indexing technique in Oracle or MS-SQL Server, the DBAs should take into account the following guidelines and pieces of advices.

1. If two database objects are used at the same time, the DBA should store them on different disk drives to minimize disk head contention. For example, if the DBA runs a SELECT statement on two tables or more, the I/O will be at the same time - any two B-tree indexes that show I/O at the same time, or a table that shows I/O at the same time as the B-tree index defined on it.
2. If users are randomly accessing a table and if the total size of the table is much larger than any practical buffer size, then increasing the buffer size is not helpful. For example, if the

table size is 100 MBytes, then a 2000-page buffer does not work much better than a 1000-page buffer.

3. The B-tree indexing is the intersection between Oracle and MS-SQL Server. As a result, the B-tree is the best indexing technique and it is more effective on huge data. But the Bitmap index in Oracle are interesting and significant, unfortunately this index is not supported by MS-SQL Server.
4. Creating an index on a column in any of the following situations:
 - The column is queried frequently.
 - A referential integrity constraint exists on the column.
 - A UNIQUE key integrity constraint exists on the column.
5. It is possible to create an index on any column if the column is not used in any of the above situations. In this case, creating an index on the column does not increase performance and the index takes up resources unnecessarily.
6. Assuming the composite index is on (Col1, Col2), then DBAs cannot use the B-Tree and will either do a complete index scan or a table scan (depending on which it thinks is faster to complete the processing of the SQL statement). Composite index is an index that contains more than one column. In both SQL Server 2005 and 2008, DBA can include up to 16 columns in an index, as long as the index does not exceed the 900-byte limit. Note that both clustered and non-clustered indexes can be composite indexes. In Oracle and MS-SQL Server, composite index could be also a B-tree index, which consists of many columns.
7. A composite index has a significant advantage in the following two cases:
 - a) assumes that the frequent use in the WHERE clause of the following conditions: `STUD_ID = 1` and `STUD_GRADE = 'A'`. If the DBA creates an index for each column, then to search out the value of the two indexes should be read, but if the two have created a

composite index, only an index is read, it certainly demands more than two indexes fewer I/O.

b) Using the same conditions as the previous example: if the DBA creates a composite index, then it will retrieve the line quickly, because the DBA is excluding all STUD_ID not a line, thereby reducing the number of rows STUD_GRADE search.

8. In composite index, B-tree is the preferable indexing technique on both platforms: Oracle and MS SQL Server with different data table sizes (small, medium and large).
9. An experiment is an implementation tested with real data. Experiments should be designed to obtain clear results. Experiments should be reproducible, which means that they should not only be conducted rigorously but their description should be sufficiently comprehensive those others can reproduce the conditions and verify the claimed results. Experiments should be based on benchmarks such as standard sets of data and queries; use of such benchmarks allows easy comparison with other work.
10. DBAs can judge the success of the used technique according to its performance on the basis of the stated. Assumptions should not only be claimed to be reasonable, they should be argued for, and, where possible, demonstrated as being reasonable.
11. DBA should also note that the scaling can relatively change the oval performance. Having larger data table can reduce the chance of sequential seeks to the same block; can increase data fetch costs, even relative to seek costs; and can even affect the proportion of records that are answers.

3.19 Observations and Recommendations

After conducting these tests using our sample database table (100K, 1000K and 5000K), we have made the following general observations and recommendations. Thus, it is suggested that Database DBA's are encouraged to use them as standard recommendations only and validate the applicability of the results to certain target scenario.

1. In Oracle, the Bitmap index on 100K has the efficient retrieving time compared with the other indexing techniques.
2. In Oracle, the Bitmap index on 1000K relatively has the best retrieving time compared with the other indexing techniques.
3. In Oracle with size: 5000K, the performance of Bitmap index, B-tree and Reverse index are much better than Organization index especially at full scan select performance.
4. In MS-SQL Server with size: 100K, the performance of Primary Key Clustered is relatively faster than B-tree (Unique Non-clustered) especially at full scan select performance.
5. In MS-SQL Server with sizes: 1000K and 5000K, the performance of Primary Key Clustered is relatively better than B-tree (Unique Non-clustered) especially at full scan select performance.
6. Some technical studies indicate that using index in the retrieval systems over Oracle platform consumes greater response time than MS-SQL Server. Thus, those studies support our results in this thesis.
7. B-tree is the common between MS-SQL Server and Oracle. The results of Bitmap index in Oracle are interesting and significant and unfortunately, this index is not supported by MS-SQL Server.

Chapter 4

Conclusions and Future Work

4.1 Conclusions

1. The proposed methodology is in thesis driven by our desire to undertake a formal comparison between the current indexing techniques. In this work we have applied the guidelines described in this chapter to one particular problem, and but felt the guidelines themselves are sufficiently interesting to warrant separate description. A set of criteria by which indexing techniques should be compared are discussed.

2. Our methodology is based on a series of experiments to test a set of indexing techniques on two different platforms (Oracle and MS SQL Server) with different data sizes (small – 100K, medium – 1000K, and very large – 5000K) over the same technical environment (Multiple processors, memory, and I/O devices).

3. To run the experiments, we have taken the following indexing techniques in Oracle: B-tree, Bitmap, Reversed, and organization index. In the meanwhile, we have taken the following indexing techniques in MS-SQL Server: B-tree, Clustered index, and unique non-clustered index and Primary Key Clustered index.

4. The empirical results show that the overall performance of indexing techniques (B-tree, reverse, organization, clustered, and bitmap indexes) in MS-SQL Server is much faster than Oracle 10g. Thus using the index in the SELECT statement over MS-SQL Server is less much costly in terms of I/O operations, CPU consumptions, and response time than Oracle. However, the B-tree index in the SELELCT statement over Oracle is faster in terms of performance than MS-SQL Server in the following cases:

- Index scan select performance when retrieved single row.
- Index scan select performance when retrieved non-row.

5. We have attempted to achieve the thesis objectives to be real outcomes at the end of this thesis.

The main outcomes of this thesis are summarized as follows:

a- Building a referenced guide is to help database developers and DBAs for selecting the indexing method in order to retrieve their data in efficient method. This outcome is very clear in Chapter 3 because we have shown the best indexing technique on Oracle and MS-SQL Server with different data sizes (100K, 1000K, and 5000K).

b- Making two types of comparisons between the available indexing methods (Clustered and Non-Clustered indexes) on two platforms: Oracle and MS-SQL Server. The first outcome is shown in chapter 1 where we have offered a cooperative comparison between the indexing techniques in accordance with the previous research. The second outcome is shown in chapter 2 and chapter 3 because we have described experimental results among the indexing techniques on Oracle and MS-SQL Server with different data sizes (100K, 1000K, and 5000K).

c- Finding the rules and criteria that make the decision of selecting appropriate indexing technique. This outcome is shown in chapters 2 and chapter 3 because we have provided a flowchart of methodology procedure and methodology scenarios for conducting the comparisons between the available indexing techniques.

4.2 Future Work

In this research, we have pointed out the following subjects that can be performed in Future Work.

1. Develop a methodology for selecting the indexing techniques for the object oriented databases indexing techniques.
2. The experimental tests can be run on very huge data such as 50,000K. When the data is being huge, the results will be more valid. For example, Google search engine has a very huge retrieval system that contains trillion of documents and files.
3. Apply the indexing techniques on distributed systems.

References:

1. Gaffar A. ,(2001), ' *Design Of A Framework For Database Indexes* ', Thesis , degree of master of computer science ,Concordia Univirsity.
2. An Oracle White Paper,(2005), '*Technical Comparison of Oracle Database 10g vs. SQL Server 2005: Focus on Performance*', an Oracle White Paper October , viewed 10 September 2010 ,
< <http://www.oracle.com/technetwork/database/features/performance/twp-perf-oracle-2.pdf>>
3. Ozgur A., Taflan I. Gundem, (2006), ' Efficient indexing technique for XML-based electronic product catalogs' , *Electronic Commerce Research and Applications*, pp. 56–77.
4. Yu B. and Guoliang L., (2007), 'Effective Keyword-based Selection of Relational Databases', *SIGMOD'07*, Beijing, China. Copyright 2007 ACM .
5. Burleson C., (2010) , '*Bitmapped Index Usage*' , viewed at 1 December 2010 , < http://www.remote-dba.net/t_grid_rac_bitmapped_index_usage.htm> .
6. Fisher D., (1966), 'Data, Documentation and Decision Tables', *Comm ACM* , Vol. 9 No. 1, pp. 26–31.
7. Lin D., Jensen C., Ooi B., and Saltenis S., (2005), 'Efficient indexing of the historical', present and future positions of moving objects. In *MDM*, pp. 59–66.
8. Graefe G., (2010) , 'A survey of B-tree locking techniques' , *published in ACM Transactions on Database Systems (TODS)*, Volume 35, Issue 2.

9. Choi I. , Bongki M. and Hyoung J.,(2007), 'A clustering method based on path similarities of XML data', *Data & Knowledge Engineering* 60 , pp. 361–376.

10. Lo J., Barroso L., Eggers S., Gharachorloo K., Levy H., and Parekh S., (1998), 'An analysis of database workload performance on simultaneous multithreaded processors', in ISCA'98: *Proceedings of the 25th annual international symposium on Computer architecture*, pp. 39–50.

11. King J., (2001), ' ORACLE8I INDEXING CHOICES : BEST OF BREED' ,Designing, Developing and Deploying Applications , viewed 20 June 2010 , < http://www.kingtraining.com/confdownloads/downloads/O8index_paper.pdf>.

12. Zobel, J., Moffat, A. and Ramamohanarao, K. (1996), 'Guidelines for Presentation and Comparison of Indexing Techniques'. *SIGMOD Record*, Vol. 25.,PP 10-15.

13. Aouiche K., Darmont J., Boussaïd O. and Bentayeb F., (2005) , ' Automatic Selection of Bitmap Join Indexes in Data Warehouses', *Data Warehousing And Knowledge Discovery* , Volume 3589, pp. 64-73.

14. Elbassioni K., Elmasry A., and Kamel I., (2003), 'An efficient indexing scheme for multi-dimensional moving objects', *In ICDT*, pp. 422–436.

15. Keeton K., Patterson D., Raphael R. and Baker W., (1998), 'Performance Characterization of a Quad Pentium Pro SMP using OLTP Workloads' , *In Proceedings of the 25th International Symposium on Computer Architecture*, pp. 15 – 26, Barcelona, Spain.

16. Nguyen L., Walid G. Aref and Mohamed F. Mokbel, (2003), 'Spatio-Temporal Access Methods: Part 2 (2003 - 2010)', *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* .

17. Cyran, M, Lane, P & Pol. JP (2005) , Oracle® Database Concepts , 10g Release 2 (10.2) , Oracle , U.S.A, viewed 10 September 2010 , <<http://www.datacons.it/oradoc/26004.pdf>>.

18. Giugno R., (2002) , PhD Thesis , 'Searching Algorithms and Data Structures for Combinatorial, Temporal and Probabilistic Databases' , UNIVERSIT`A DEGLI STUDI DI CATANI.

19. Martin R. Frank, Edward R. Omiecinski and Shamkant B. Navathe , 1992, Adaptive and automated index selection in RDBMS , *Advances In Database Technology* , Volume 580/1992, pp. 277-292, viewed 20 August 2010 , <<http://www.springerlink.com/content/d4327448p5342880/>>.

20. Chaudhuri, S., and Narasayya, V. , (1998),' Autoadmin “what-if” Index Analysis Utility', *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 367–377.

21. Chaudhuri, S., Datar, M., and Narasayya, V. , (2004) , ' Index Selection for Databases', A Hardness Study and a Principled Heuristic Solution. *IEEE Transactions on Knowledge and Data Engineering*, 16(11), pp. 1313–1323.

22. Dessloch S, Mattos N., (1997), 'Integrating SQL Databases with Content-specific Search Engines' , *Proceedings of the 23rd VLDB Conference Athens, Greece*.

23. Halawani¹ S., Albidewi I., Alam J. and Khan Z, (2010),'A Critical Evaluation of Relational Database Systems for OLAP Applications', (*IJCNS*) *International Journal of Computer and Network Security*, Vol. 2, No. 5, PP.122-126.

24. Madhulatha S. , 2010, ' A Study on Index Selection Problem', Dept of Informatics Alluri Institute of management sciences, viewed 2 September 2010, < <http://www.articlesbase.com/print/2930457>>.

25. Ponce S., Vila P. and Hersch R.,(2002), 'Indexing and selection of data items in huge data sets by constructing and accessing tag collections', *19th IEEE Symposium on Mass Storage Systems & Tenth Goddard Conf. on Mass Storage Systems and Technologies Univ. of Maryland*, College Park, Maryland, USA, pp. 181-192.

Glossary

Attribute

Describes the value found in each field in a table. Every field or column in a database table represents a single *attribute* of that table. (An *attribute* is what the data in that field *represents*, while the *value* is the actual data that a specific field contains. See also: *Value*.)

Cardinality

In SQL (Structured Query Language), the term refers to the uniqueness of data values contained in a particular column (attribute) of a database table.

Case; Casing

To designate which characters in an alpha string will be uppercase and which will be lowercase. Common casing methods include: uppercase all characters; lowercase all characters; uppercase first character of the string; uppercase the first character of each “word” (space-separated substrings) contained (aka called “Proper” case); lowercase the entire string, then uppercase the first character; or lowercase the entire string, then uppercase the first character of each “word”.

Case-Sensitive

To be aware of the case of character values. In this context, “SPUD,” “Spud” and “spud” would all be considered as different strings, so the case-sensitivity of a function or query will influence the values they will return.

Client

That part of a DBMS that displays information on the screen and responds to user input (the front-end).

Client/Server System

A multi-user system in which a central processor (server) is connected to multiple intelligent user workstations (clients).

Column

Synonymous with *field*. See also: *Field* and *Attribute*.

Commit

Decision to proceed with the actual posting of a change to the database.

Composite Key

A primary key that consists of two or more attributes is known as composite key

Concurrent Access

Two or more users operating on the same records in the same database table at the same time.

Constraints

Data restrictions specified in a database; rules that determine what values the field to the table can assume.

Data Dictionary

The database stores metadata in an area called the data dictionary, which describes the tables, fields, indexes, constraints, and other related items that make up the database.

Data Model

The logical data structures, including operations and constraints provided by the DBMS to effectively process data; system used for the representation of data (the ERD, or relational model).

Data Redundancy

Having the same data stored in more than one place in a database.

Data Retrieval

It's a data extraction from disparate sources, most operational, some legacy -- typically in different formats.

Data Source

It's the source of data used by a database application. It maybe a DBMS, table or a data file.

Data Structure

Is a logical relationship among data elements that is designed to support specific data manipulation functions (trees, lists, and tables).

Data Type

Every field in every table in a database must be declared as a specific type of data with defined parameters and limitations (e.g. numeric, character or text, date, logical, etc.), known as a data type.

Database

1) A collection of all the data needed by a person or organization to perform their required functions, 2) A collection of related files or tables; 3) Any collection of data organized to answer queries; or, 4) [Informally,] a database management system. (Databases usually consist of both data and metadata [data about the database's data]. When a database contains a description of its own structure, it is said to be self-describing. A database is integrated when it includes its relationships among data items as well as the data items themselves.)

Database Administrator [DBA]

The person who is ultimately responsible for the functionality, integrity, and safety of the database.

Database Engine

That part of the DBMS that directly interacts with the database (part of the back-end).

Database Management System [DBMS]

Also called a database manager. An integrated collection of programs designed to enable people to design databases, enter and maintain data, and perform queries.

Database Manager

1) The person with primary responsibility for the design, construction, and maintenance of a database. 2) [Informally,] a database management system.

Database Warehouse –*Short:* A copy of transaction data specifically structured for query, analysis and reporting. *Long:* The database warehouse, a single repository depicting a logical view of an enterprise's data, accessible to developers and business users alike. Effective database warehousing requires frequent updates and impeccable data quality to insure business end-users and decision makers are using the same data, at the same extraction level, as everyone else when they run queries and reports or formulate analyses.

Distributed Database

Is a database in which resources are stored on more than one computer system, often at different physical locations.

Entity

A real-world object, observation, transaction, or person about which data are to be stored in a database.

Expression

An SQL statement that returns a value.

Extraction

The process of selecting data from one environment and transporting it to another environment.

See also: *Data Transformation*.

Field

Synonymous with *column*. A component of a relation or table that holds a single *attribute* of that relation or table. See also: *Column* and *Attribute*.

File

1) The separately named unit of storage for all data, programs and indexes on most computer systems. For example, a table or a whole database may be stored in one file; 2) Term used as a synonym for relation or table in some database managers [usually smaller or older], like dBase, FoxPro, Alpha Four/Five, etc.

Functional Dependency

Is a relationship between or among fields where one field is functionally dependent on another if the value of the second field determines the value of the first. (If you know the value of the second, you can determine the value of the first.)

High-cardinality

It refers to columns with values that are very uncommon or unique. High-cardinality column values are typically identification numbers, email addresses, or user names. An example of a data table column with high-cardinality would be a STUDENT table with a column named STU_ID. This column would contain unique values of 1-*n*. Each time a new user is created in the STUDENT table, a new number would be created in the STU_ID column to identify them uniquely. Since the values held in the STU_ID column are unique, this column's cardinality type would be referred to as high-cardinality.

Implementation

A particular relational DBMS running on a specific hardware platform.

Index

1) A method used to reorder display or output records in a specific order; 2) A data structure of pointers used to provide rapid, random access to rows in the table.

Information Schema- See: *Schema, Information.*

Integrity

The property of the database that ensures that the data contained in the database is as accurate and consistent as possible.

Join

A relational operator (query) that combines data from multiple tables into a single result table. Tables must have at least one field (sometimes called the join or linking field) in common, so that values from corresponding records in each table are matched up correctly.

Join, Cross

Cross joins return all rows from the left table, each row from the left table is combined with all rows from the right table. Cross joins are also called Cartesian products.

Join, Inner

An inner join discards all records from the result table that don't have corresponding records in both source tables, while an outer join preserves unmatched records.

Key

A key is a field, or combination of fields, that uniquely identifies a record in a table. See also: Key, Primary.

Key, Candidate

1) One or more fields that will uniquely identify one record in a table; 2) A potential primary key.

Key, Composite

A key made up of two or more table columns that, together, guarantee uniqueness, when there is no single column available that can guarantee uniqueness by itself.

Key, Database

The unique value that exists for each record in a database. The value is often indexed.

Key, Foreign

A column or group of columns in a table that corresponds to or references a primary key in another table in the database. A foreign Key need not itself be unique, but must uniquely identify the field or fields in the table that the key references.

Key, Primary

A field or combination of fields that uniquely identifies each record in a table, so that each record can be uniquely distinguished from every other occurring in the table. A table cannot have more than one primary key, and a primary key, by definition may not contain a null value.

Key, Secondary

A key that is not the primary key for a table.

Low-cardinality

That refers to columns with few unique values. Low-cardinality column Boolean values, or major classifications such as gender. An example of a data table column with low-cardinality would be a STUDENT table with a column named NEW_STUDENT. This column would contain only 2 distinct values: Y or N, denoting whether the student was new or not. Since there are only 2 possible values held in this column, its cardinality type would be referred to as low-cardinality.

Medium-cardinality

That refers to columns with values that are somewhat uncommon. Normal-cardinality column values are typically names, street addresses, or vehicle types. An example of a data table column with normal-cardinality would be a STUDENT table with a column named LAST_NAME, containing the last names of customers. While some people have common last names, such as Mohammed, others have uncommon last names. Therefore, an examination of all of the values held in the LAST_NAME column would show "clumps" of names in some places (e.g.: a lot of Mohammed) surrounded on both sides by a long series of unique values. Since there are a variety of possible values held in this column, its cardinality type would be referred to as normal-cardinality.

Normal Form

1) A condition of tables and databases intended to reduce data redundancy and improve performance; 2) Rules and processes for putting tables and databases into normal form.

Normalization

1) The process of breaking up a table into multiple tables, each of which has a single theme, thereby reducing data redundancy; 2) The technique that reduces or eliminates the possibility that a database is subject to modification anomalies. See also: *Data Redundancy*.

Query

1) Literally, a question you ask about data in the database in the form of a command, written in a query language, defining sort order and selection, that is used to generate an ad hoc list of records; 2) The output subset of data produced in response to a query.

Record

Synonymous with *row* and *tuple*. An instance of data in a table, a record is a collection of all the facts related to one physical or conceptual *entity*; often referring to a single object or person, usually represented as a *row* of data in a table, and sometimes referred to as a tuple in some, particularly older, database management systems.

Schema

1) The database's metadata -- the structure of an entire database, which specifies, among other things, the tables, their fields, and their domains. In some database systems, the linking or join fields are also specified as part of the schema 2) The description of a single table. Also called a Logical Schema.

Select; Selection

A query in which only some of the records in the source table appear in the output.

Sort; Sorting

The act of putting records in a particular order.

SQL

Pronounced “Sequel”, it stands for Structured Query Language, the standard format for commands that most database software understands. There are different dialects, since every program handles certain types of data differently, but the core commands are always the same. ODBC uses SQL as the "Lingua Franca" to transfer information between databases. Currently accepted ANSI standard is SQL-92.

Table

Synonymous with *relation*. A collection of data organized into *records* and *fields* (aka *rows* and *columns*), with fields being descriptions of the kinds of information contained in each record (*attributes*); and records being specific instances usually referring to specific objects or persons (*entities*). See also: *Relation* and *Attribute*.

Transaction

- 1) The fundamental unit of change in many (transaction-oriented) databases. A single transaction may involve changes in several tables, all of which must be made simultaneously in order for the database to be internally consistent and correct
- 2) A real-life event which is modeled by the changes to the database; 3) The sequence of SQL statements whose effect is not accessible to other transactions until all of its statements have been executed.

