MEASURING RELATIONAL DATABASE SERVER TRANSACTION SPEEDS USING THE AS3AP BENCHMARK

by
Rajiv Zutshi

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science.

Chapel Hill, North Carolina

April, 1999

Approved by:

_____
Advisor

RAJIV ZUTSHI. Measuring relational database server transaction speeds with the AS3AP benchmark. A Master's paper for the M.S. in I.S. degree. April, 1999. 31 pages. Advisor: Gregory B. Newby

This project involves comparing two relational database servers using sections of the AS3AP industry standard benchmark. The two servers tested were Oracle 8.0.3, developed by the Oracle Corporation and MySQL 3.22.14b-gamma, developed by T.c.X DataKonsultAB. The servers were both installed on the same Intel-based hardware platform running the Red Hat Linux 5.2 operating system (kernel 2.0.36).

All tests were conducted over a network on a fixed size database. The complexities of the queries were varied and the transactions per second were measured on the database server side to give an indication of relative performance.

Headings:

Database Benchmark

Database Management System

Transaction

Oracle

MySQL

TABLE OF CONTENTS

INTRODUCTION

This project involved conducting several experiments on two relational database servers in order to compare their performance. The database servers under test were Oracle 8.0.3 and MySQL 3.22.14b-gamma. A database server is a software package that provides the ability to store data and manipulate that data through its database management system (DBMS). The architecture of individual servers varies significantly which accounts for their varying levels of performance.

To compare both systems required implementing a benchmark. A benchmark is essentially a single test or set of tests that are run against an individual server. The industry standard benchmark known as the ANSI SQL STANDARD SCALABLE and PORTABLE (AS3AP) was used to test the Oracle and MySQL servers. The focus of the tests were to determine the fastest transaction processing time achieved by the database servers over a series of pre-defined tests. The units used to measure this processing time were transactions per second (tps).

A series of SQL statements were used to perform the various tests and obtain the transaction speeds for the AS3AP benchmark. All these SQL statements were written in compliance with the ANSI SQL-92 standard, and since both these database servers upheld these standards, they could be tested.

The AS3AP benchmark was chosen in light of other industry standard benchmarks because its results provide the most comprehensive comparison between relational database systems of different architectures. The software packages used in the experiments were chosen because they were free. This is important as it parallels the

mission of the Free Software Foundation which promotes "…the development and use of free software in all areas of computing." (Free Software Foundation, 1998).

The Oracle 8.0.3 server was chosen as it is the most widely used database server in industry today and the 8.0.3 version is the latest for Linux systems. Oracle allows data to be stored in its repository and allows users to share this information seamlessly through its various configurations (Oracle 8: A Beginners Guide, 16-17):

- ***Host-based*** Users are connected directly to the same computer on which the database resides.
- ***Client/server*** Users access the database from their personal computer (client) via a network, and the database sits on a separate computer (server).
- ***Distributed processing*** Users access a database that resides on more than one computer. The database is spread across more than one machine, and the users are unaware of the physical location of the data they work with.
- ***Web-enabled computing*** The ability to access data from an Internet-based application.

The MySQL server supports most of the configurations listed above. However, it is a much smaller system and therefore does not support distributed processing, which is mostly required for high-end computing. The primary reason behind benchmarking MySQL stems from its reputation as being an extremely fast and efficient server.

Red Hat Linux 5.2 is the operating system upon which both database servers were installed and tested. Linux is a flavor of UNIX, which is freely available over the Internet and was chosen because it has also gained a strong reputation for being extremely stable and efficient.

BENCHMARKS AND THE AS3AP

As more software is introduced into the market, there is a great need to establish standardized benchmarks to rate the overall quality of a new product in comparison to a competing product. The Transaction Processing Council (TPC) is the current authority for creating benchmarks to be used in fields such as transaction processing and database applications. The TPC was formed in 1988 and is composed of eight companies. It was formed in order to allow computer vendors to compare their products using a standard set of tests known as benchmarks. The TPC is also involved in creating processes for monitoring and reviewing the validity of their existing benchmarks to prevent corporations from over inflating their results.

In this experimental environment, there is no need to inflate results, therefore we should be able to gain a fair assessment of the individual server performance. With the rapid innovations in both the hardware and software industry, it is apparent that the benchmarks produced by the TPC have a certain shelf life. After a benchmark is published, companies generally optimize their systems to run the benchmark faster. This would naturally improve the quality of the product for the end-user as well as increasing the product's sales potential. However, it is important to understand that after a certain point, the results obtained do not accurately reflect real world situations. Benchmarking is, after all, an experimental science, conducted in a controlled environment. In light of this fact, the results published by a corporation for its benchmarks are generally a best case scenario. This fact also applies to this project. Although we have chosen to implement the AS3AP benchmark, the TPC has created a number of other benchmarks

and they include the Wisconsin, TPC-A, TPC-B, and the TPC-C, to name a few examples.

All these benchmarks work with two fundamental variables which are used in many experimental settings. The first is the set of independent variables called experimental factors that affect the performance of the database system. Experimental factors include the size of the database under test and the complexity of the queries run on the database. The advantage of using the AS3AP benchmark here is the database under test and the queries are all constant as per the AS3AP specifications outlined by the TPC.

The second is the set of dependent variables called performance metrics, which are the data collected during the experiment. The performance metrics we will be concentrating on include the transaction processing time spanned when a query executes on a fixed sized database. The other metric commonly used in evaluation of database systems is the cost per megabyte. As stated earlier, the software used in this experiment is available at no cost and therefore the cost per megabyte for storage for either server is $0/MB. (Benchmarking RDBMS, 1994)

INDUSTRY STANDARD BENCHMARKS

The standard list of benchmarks can be broken down into two essential categories: On-Line Transaction Processing (OLTP) benchmarks and relational query benchmarks. The list of OLTP benchmarks includes the TPC-A, TPC-B, TPC-C, and TPC-D.

OLTP BENCHMARKS

The TPC-A benchmark is "an update-intensive on-line transaction processing … benchmark which simulates one hypothetical bank transaction type in a networking environment." (Benchmarking RDBMS, 1994). The reason a bank transaction was used in the benchmark was due to the fact TPC-A was actually developed around an earlier benchmark known as the DebitCredit, which was designed to measure OLTP in the banking industry.

The TPC-B can be described as "the batch version of the DebitCredit [benchmark], without the network and user interaction (terminals) figured into the workload." (Transaction Processing Council, 1998) The TPC-B benchmark gained mixed reviews. Proponents stated that the TPC-B fairly represented the environment in which these types of transactions occurred and thus were a fair representation of performance. On the other hand, there were those that argued that the TPC-B actually produced over-inflated results, as the benchmark did not test key system resources. Needless to say, the TPC-B did not gain very much support.

The TPC-C benchmark surfaced in 1992 and was later revised in 1993. It is scheduled to be revised again sometime in 1999. The TPC-C "is a complex OLTP benchmark [which] emulates hypothetical order entry and inventory control transactions in a production environment." (Benchmarking RDBMS, 1994) This industry standard, unlike the TPC-A and TPC-B, has become a widely used benchmark for measuring transaction speeds for processing intensive applications.

The TPC-D benchmark is also widely used in industry today. It is designed to test the manipulation of large sets of data by focusing on complex queries that involve joining

several tables, ordering results sets, and computing aggregates. "The base size of the test database is six hundred megabytes. TPC-D basically measures the execution time of a complete query set." (Benchmarking RDBMS, 1994)

RELATIONAL QUERY BENCHMARKS

The other category of benchmarks are the relational query benchmarks which include the Wisconsin benchmark and the AS3AP benchmark. The Wisconsin benchmark was developed in the early 1980s and was an early attempt to establish a series of tests relational database systems could be run against. The main argument against using the Wisconsin benchmark was that it was too simplistic and therefore provided an unrealistic measure of performance. This is attributed to the fact that the four relations present only contain two different data types. Further developments of the Wisconsin benchmark led to the first release of the AS3AP benchmark.

As stated earlier, this project focuses on implementing the AS3AP benchmark. The AS3AP tests are divided into the following two sections (Benchmark Handbook, 1998):

- Single-user tests, including:

  (a) utilities for loading and structuring the database,
  (b) queries designed to test access methods and basic query optimization – selections, simple joins, projections, aggregates, one-tuple updates, and bulk updates.

- Multi-user tests modeling different types of database workloads:

  (a) on-line transaction processing (OLTP) workloads,
  (b) information retrieval (IR) workloads,
  (c) mixed workloads including a balance of short transactions, report queries, relation scan, and long transactions.

The project will concentrate on single-user tests. In the evaluation, considerable emphasis will also be given to functionality issues such as database loading and table creation. These types of operations directly impact the average user of the database server. To summarize, the primary benefits of implementing the AS3AP benchmark are (Benchmark Handbook, 1998):

- AS3AP provides a comprehensive but tractable set of tests for database processing power.
- Human effort in implementing and running benchmark tests is minimized.
- The AS3AP provides a uniform metric… for a straightforward and non-ambiguous interpretation for the benchmark results.

The actual AS3AP database upon which the SQL queries execute consist of the following five relations (Benchmark Handbook, 1998):

A_TINY: a relation used to measure overhead.

A_UNIQUES: a relation where all attributes have unique values.

A_HUNDRED: a relation where most of the attributes have exactly 100 unique values, and are correlated. This relation provides absolute selectivities of 100, and projections producing exactly 100 multi-attribute tuples.

A_TENPCT: A relation where most of the attributes have 10% unique values. This relation provides relative selectivities of 10%.

A_UPDATES: A relation customized for updates. Different distributions are used and three types of indices are built on this relation.

All the relations with the exception of A_TINY have exactly the same ten columns with the same names and data types. The schema of these relations can be found in Appendix A, where a list of the column names and their data types are presented. The actual AS3AP benchmark run sequence can be found in Appendix B.

METHODOLOGY

There were numerous points to consider when setting up the system to perform the benchmark. The major concerns were setting up the actual test bench to make sure it was fully functional and could in fact run the experiments. All the benchmarks were conducted on a constant hardware platform. The specifications of this hardware platform are as follows: Intel Pentium II 400MHz, 256 MB RAM, 10 GB Hard Disk.

The second stage in the setup was to install the operating system on the machine. Red Hat's distribution of Linux (kernel 2.0.36) was chosen arbitrarily over other distributors like S.u.S.E. or Caldera. The operating system was installed via FTP from ftp://metalab.unc.edu/pub/Linux/distributions/redhat/current/i386/.

The third stage was the installation and configuration of the database servers. The MySQL server was installed first. The compressed file *mysql-3.22.14b-gamma.tar.gz* was transferred from ftp://ftp.mysql.com/pub/mysql/Downloads/MySQL-3.22, and at the time was the latest version of MySQL available for Linux based systems. Some configuration of the server was required after the installation. The installation of the Oracle 8.0.3 server was done via CD-ROM. The installation of this software package was fairly complicated and required considerable configuration.

The diagram showing the setup of the experiment can be found in Appendix C. All the tests were conducted over the network from a Windows NT 4.00 workstation using a piece of software called Benchmark Factory 97, developed by Client Server Solutions. This piece of software was composed of two main components called the Visual Control Center and the Agent. The Agent is a tool that can simulate several users trying to process queries on the database server under test. This component however,

was neglected in our experiments as we were focused on single-user tests and not multi-user tests.

The Visual Control Center is the dashboard that provides the tools to run all the experiments over the network segment. This component provides all the timing mechanisms necessary to run the AS3AP benchmark. The connections between the Visual Control Center and the database servers are established through the use of Open Database Connectivity (ODBC) drivers. These drivers were installed on the Windows NT machine.

ODBC is an Application Programming Interface (API) that allows connection to any SQL database. ODBC essentially defines a set of function calls, data types, and error codes that enable queries written at the application level to be run against the database server. ODBC is also responsible for retrieving the resulting relation as the SQL query executes. This functionality is brought about through the database server's proprietary ODBC driver.

When the query was passed to the database via the ODBC driver, the server performed a number of read and/or write operations to the hard disk. Depending upon the specific query that was executed, the number of read and/or write operations to the disk varied considerably. These operations, known as transactions, are defined below in more detail.

A transaction and its various states (Elmasri/Navathe, 534):

BEGIN_TRANSACTION: This marks the beginning of transaction execution.

READ or WRITE: These specify read or write operations on the database items that are executed as part of a transaction.

END_TRANSACTION: This specifies that READ and WRITE transaction operations have ended and marks the end limit of transaction execution…

COMMIT_TRANSACTION: This signals a *successful end* of the transaction so that any changes (updates) executed by the transaction can be safely **committed** to the database and will not be undone.

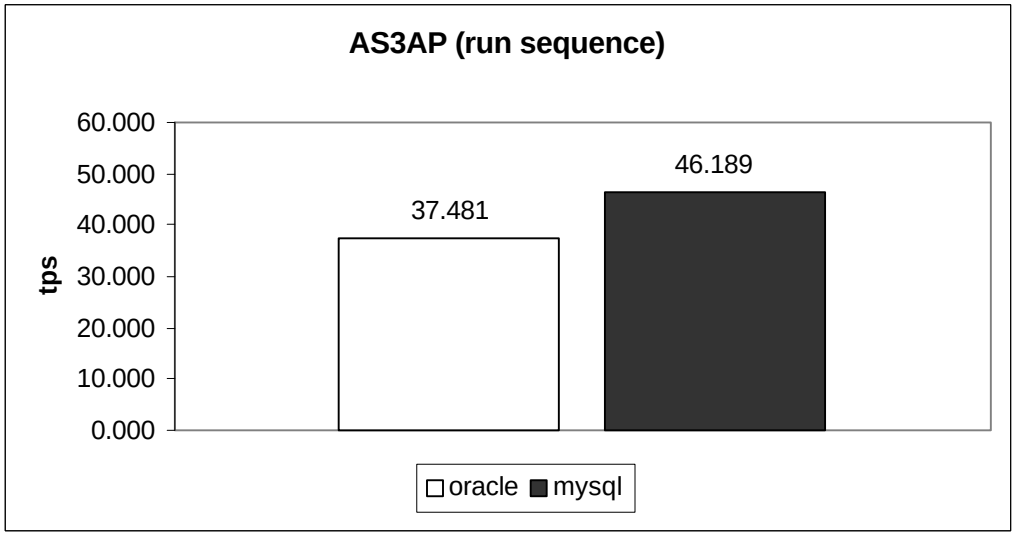
ROLLBACK (or ABORT): This signals that the transaction has *ended unsuccessfully*, so that any changes or effects that the transaction may have applied to the database must be *undone*.

Oracle completes all stages of the transaction listed above. MySQL on the other hand does not support the COMMIT or ROLLBACK stages. The ratio of the number of transactions completed by the database server divided by the time taken to execute those transactions is measured and outputted to the Visual Control Center. The benefit of measuring the power of the database servers on the server side allowed the effects of the network to be neglected. Isolating variables and deciding upon which ones to keep constant and which ones to vary were important considerations in the experimental process.

The test bench kept the hardware and operating system constant and isolated network effects. The size of the database was also kept constant and scaled to 10,000 rows of randomized data. The database servers were varied as were the set of benchmarks. In addition to running the AS3AP sequence, other functional performance criteria were evaluated. These criteria are explained further in the next section.

DATA ANALYSIS

As stated earlier, the suite of benchmarks that were run can be found in Appendix B. The data produced from the numerous runs during the testing phase was averaged, and these average values were then graphed. The transaction per second (tps) on the y-axis, the dependent variable, was plotted against the fixed-size database servers on the x-axis. The database server with the higher level of performance had a higher number of transactions executed per second.



AS3AP - This run sequence of forty-four separate tests is the comprehensive set of SQL statements that evaluates the overall robustness of the database server under test. The run sequence can be divided into five primary categories (Benchmark Handbook, 1998):

Selections: A series of selection queries retrieved relations consisting of one tuple, a range of 100 tuples, or 10 % of the tuples. For each of these queries, there were two versions, which differed in their indexing structure.

Joins: The join queries tested how efficiently the system made use of the available indices and how query complexity affected the relative performance of the DBMS.

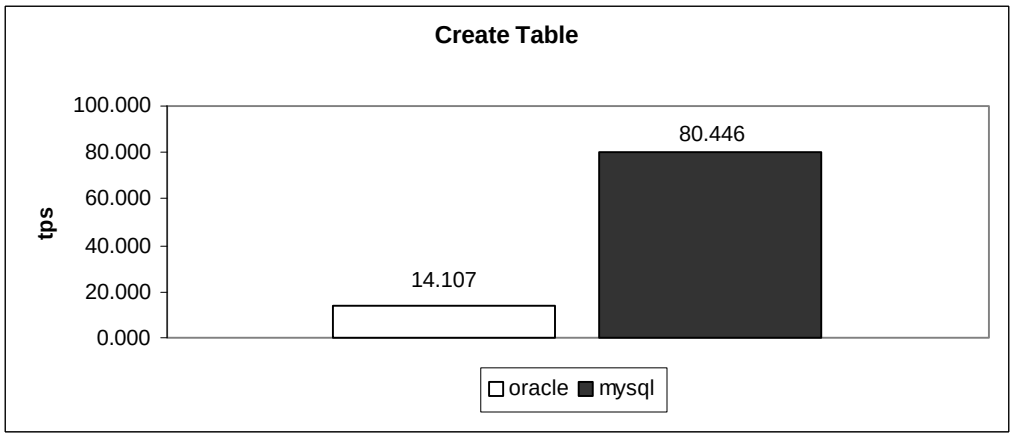
Projections: The projection queries tested the DBMS processing time by eliminating duplicate tuples, which were created on non-key attributes for obvious reasons.

Aggregates: The sequence of aggregate tests involved a combination of simple and complex queries. The simple queries consisted of scalar aggregate functions executing on a key field whereas the complex queries generated a variety of reports.

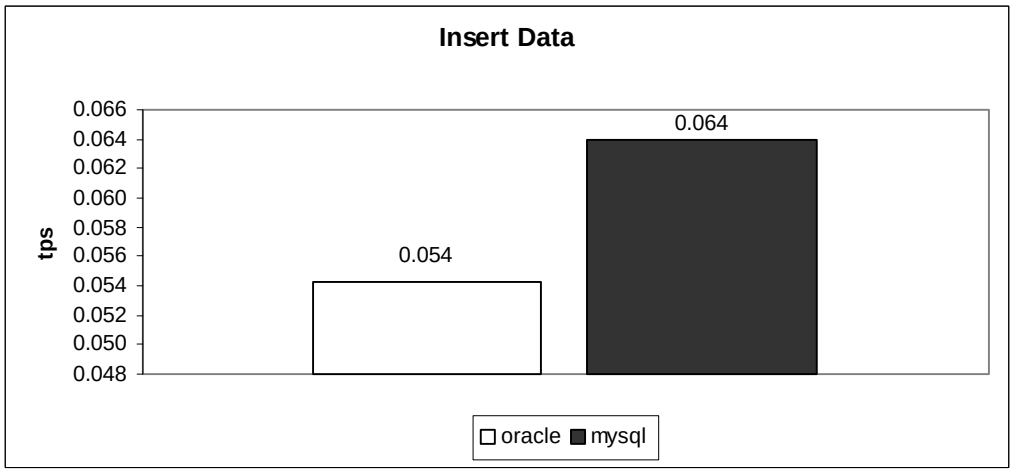
Updates: The update tests were designed to test both integrity and performance. There are two kinds of integrity constraints: entity integrity and referential integrity. The entity integrity constraint states that no primary key in a relation can have a duplicate value or be null. Attempting to append a tuple with a duplicate value tested this constraint. Referential integrity essentially states that a foreign key value in one relation must refer to an existing tuple in another relation where the value is a primary key. Attempting to update a field which was a foreign key, tested this constraint.

MySQL clearly dominated during this sequence of tests. The number of transactions per second executed by the MySQL server was over three times greater than Oracle.

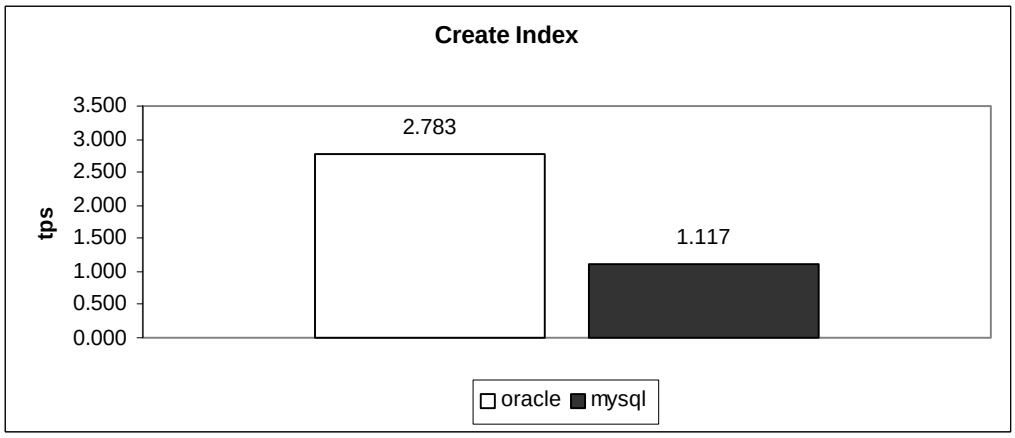
The next set of benchmarks focused on specific operational and functional issues. They do not include a lengthy series of tests like the AS3AP run sequence, but focus on definite criteria: table creation, data insertion, index creation, table deletion, aggregate computation, table joins, and single table selections with variation between low and high selectivity.



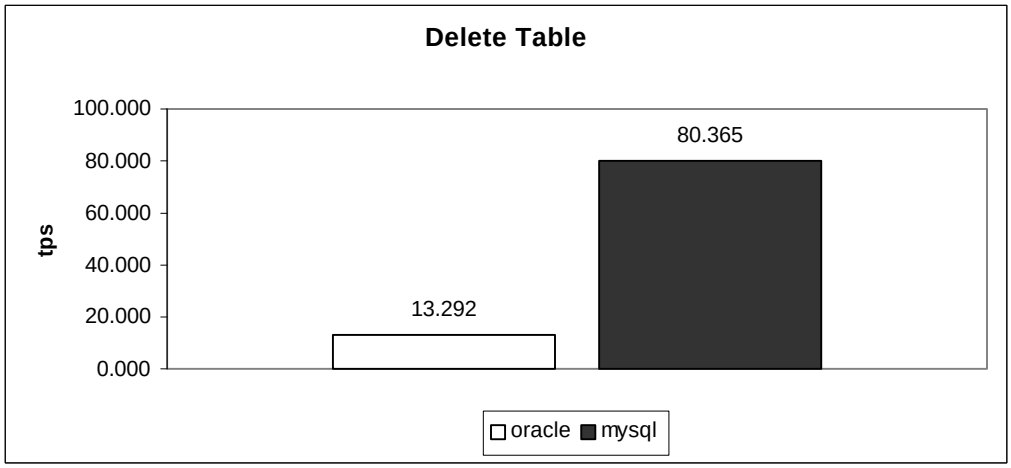
Create Table - This test measured transaction-processing speed when creating empty tables. The MySQL server's performance was over five times greater than Oracle.



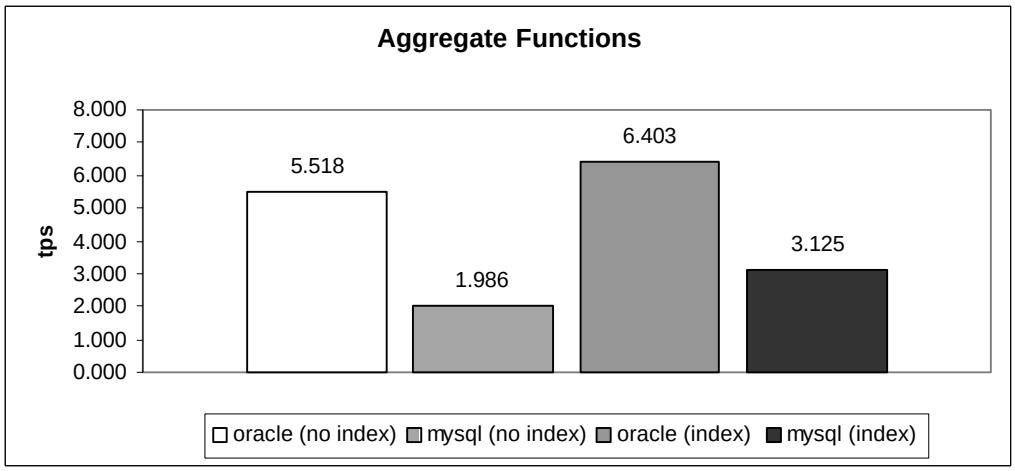
Insert Data - This test measured transaction processing speed when loading each of the four relations A_UNIQUES, A_HUNDRED, A_TENPCT, A_UPDATES with 10000 rows of random data. The MySQL server performed slightly better on this test. Both servers were fairly evenly matched.



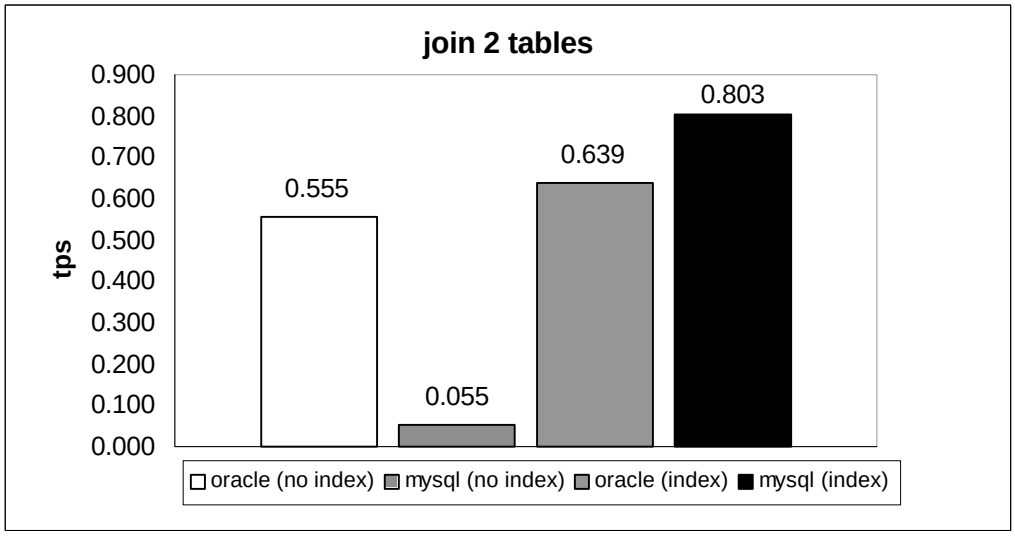
Create Index - This test measured transaction-processing speed when creating B-tree indexes on various attributes. A B-tree is a highly sophisticated indexing structure that allows for faster processing and quicker location of a record. In this scenario, the Oracle server performed over two times faster than the MySQL server.



Delete Table - This test measured transaction-processing speed when deleting a table containing 10,000 rows of data. The MySQL server proved to be over six times faster than the Oracle server.

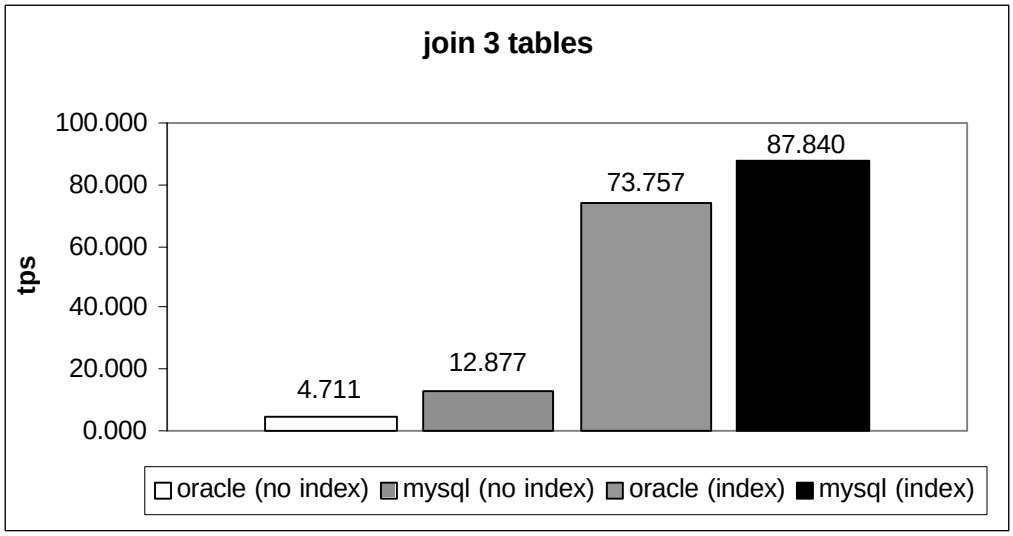


Aggregate Functions - The aggregate function attempted to find the minimum value of a floating-point number from the Afloat column in the A_HUNDRED table. This test was first performed on the A_HUNDRED table without any indexes present. The Oracle server's transaction rate was almost three times faster than the MySQL server. The same benchmark was run after a B-tree index had been created on the Akey field. The resulting effect was higher performance for both servers. However, Oracle performed about twice as fast as MySQL.



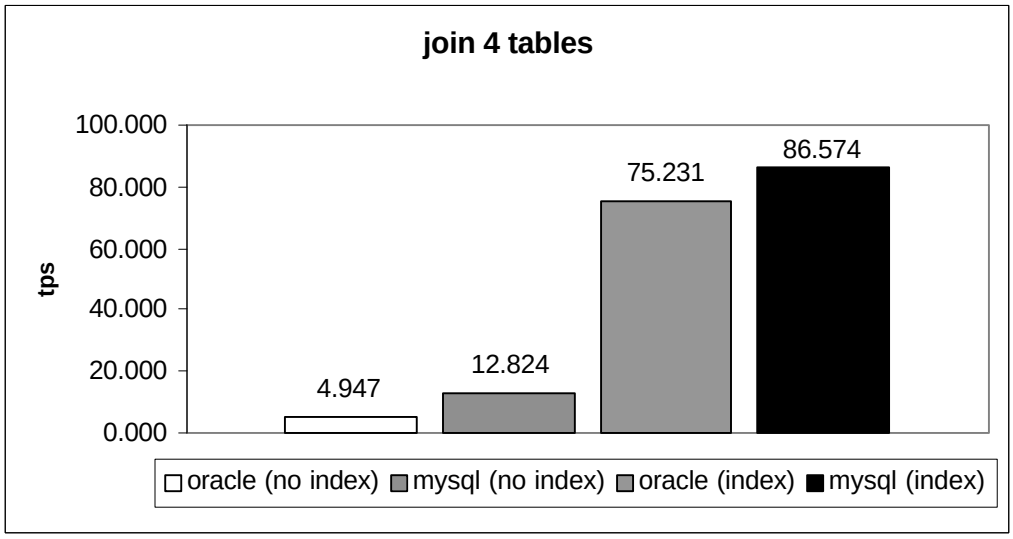
Join 2 Tables - This benchmark performed an inner join between the A_UPDATES and A_HUNDRED table and retrieved data from several attributes. This was a highly selective query, as 999 rows had to be retrieved from the two 10,000 row tables. The benchmark was first run without any indexes on either table. The Oracle server's transaction rates were ten times greater than that of the MySQL server.

The benchmark was run again, but this time an explicit B-tree index was created on the Akey field for both tables. The results were quite astounding. The Oracle server performed only 1.15 times faster with the index than without the index. On the other hand, the MySQL server performed at a significantly higher rate than Oracle, processing all the transactions almost fifteen times faster with the index than without the index.

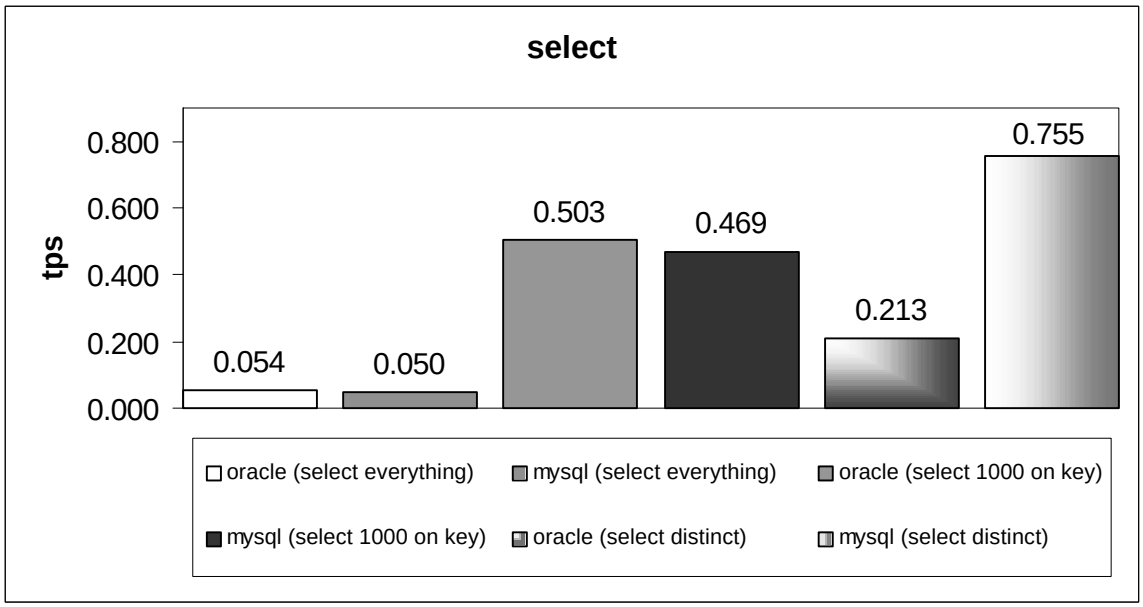


Join 3 Tables - This benchmark performed an inner join between the A_UNIQUES, A_HUNDRED, and A_TENPCT tables. This query retrieved zero rows in the result. The low selectivity designed into the query allowed for much higher transaction rates as displayed in the graph. This query was also executed with and without explicit B-tree indexes created on the Akey field on the three tables mentioned above. Without B-trees,

the MySQL server's transaction times were more than twice as fast as Oracle. After indexing, the Oracle server was only slightly slower than MySQL.



Join 4 Tables - This benchmark performed an inner join between the A_UNIQUES, A_HUNDRED, A_TENPCT, and A_UPDATES tables. This query also retrieved no rows in its result. The benchmark was run with and without explicit B-tree indexes and the results were similar to those when only three tables were joined.



Select - There were three queries executed sequentially on the A_UPDATES relation. All these queries were created to give some indication as to the transaction rates achieved when performing highly selective retrieval requests from the database. An explicit B-tree index was created on the Akey field for all three tests.

The first query required selecting all the information from the table, in other words, all 10,000 rows. The performance for both database servers was just about even and extremely slow. The second query retrieved all the information in all the fields for the first 1,000 rows of the table. Both database servers performed fairly evenly, and their transaction rates increased by a factor of ten. This performance leap allows us to deduce that the number of records retrieved is in fact directly proportional to the transaction rate for a single relation under test. The third query was designed to retrieve non-duplicate values from the Aaddress and Asigned fields. The total number of records retrieved was 7684. There was a fairly large performance gap resulting from this test and MySQL is shown to have processed transactions at over three times the rate than that of Oracle.

CONCLUSION

After executing all the benchmarks, it is clear that the MySQL server out performed the Oracle server in terms of transaction processing speed on the AS3AP database. It should be noted that in the evaluation of the two database servers, neither of the two systems were tuned or optimized in any way to enhance performance. They were both 'out of the box' packages. By tuning the database servers a completely different set of performance data could result.

Furthermore, there are limitations to be taken into account for the MySQL server. For example, basic functionality available on Oracle and other high-end database servers allows for the use of structures known as 'views.' These are important as they increase usability on the system. MySQL has no current support for 'views.' In addition, 'rollback' capabilities of the database system are non-existent in MySQL as stated earlier. Rollbacks provide a high level of fault tolerance required to maintain integrity of the data within databases. However, they also come with a heavy cost. The overhead of the server is increased tremendously when rollback capabilities are introduced, thus leading to the appearance of degraded performance. These examples just touch upon some basic functionality issues that can affect transaction-processing rates. The actual performance of a database server can vary significantly in a production environment, and results may differ considerably from this experimental setting.

APPENDIX A – TABLE SCHEMA

ORACLE: *A_UNIQUES, A_HUNDRED, A_TENPCT, A_UPDATES*

ATTRIBUTE	NULL?	DATA TYPE
AKEY	NOT NULL	NUMBER(38)
AINT	NOT NULL	NUMBER(38)
ASIGNED		NUMBER(38)
AFLOAT	NOT NULL	FLOAT(63)
ADOUBLE	NOT NULL	FLOAT(63)
ADECIM	NOT NULL	FLOAT(63)
ADATE	NOT NULL	DATE
ACODE	NOT NULL	CHAR(10)
ANAME	NOT NULL	CHAR(20)
AADDRESS	NOT NULL	VARCHAR2(80)

ORACLE: *A_TINY*

ATTRIBUTE	NULL?	DATA TYPE
AKEY	NOT NULL	NUMBER(38)

MySQL: *A_UNIQUES, A_HUNDRED, A_TENPCT, A_UPDATES*

ATTRIBUTE	NULL?	DATA TYPE
AKEY		INT(11)
AINT		INT(11)
ASIGNED	YES	INT(11)
AFLOAT		DOUBLE(16,4)
ADOUBLE		DOUBLE(16,4)
ADECIM		DOUBLE(16,4)
ADATE		TIMESTAMP(14)
ACODE		VARCHAR(10)
ANAME		VARCHAR(20)
AADDRESS		VARCHAR(80)

MySQL: *A_TINY*

ATTRIBUTE	NULL?	DATA TYPE
AKEY		INT(11)

APPENDIX B – BENCHMARK RUN SEQUENCES

AS3AP RUN SEQUENCE

o_mode_1k
SELECT * FROM A_updates WHERE Akey <= 10

o_mode_10k
SELECT * FROM A_hundred WHERE Akey <= 100

o_mode_100k
SELECT * FROM A_hundred WHERE Akey <= 1000

join_3_cl
SELECT A_uniques.Asigned, A_uniques.Adate, A_hundred.Asigned, A_hundred.Adate, A_tenpct.Asigned, A_tenpct.Adate FROM A_uniques, A_hundred, A_tenpct WHERE A_uniques.Akey = A_hundred.Akey AND A_uniques.Akey = A_tenpct.Akey AND A_uniques.Akey = 1000

sel_100_ncl
SELECT Akey, Aint, Asigned, Acode, Adouble, Aname FROM A_updates WHERE Aint <= 100

table_scan
SELECT * FROM A_uniques WHERE Aint = 1

func_agg
SELECT min(Akey) FROM A_hundred GROUP BY Aname

scal_agg
SELECT min(Akey) FROM A_uniques

sel_100_cl
SELECT Akey, Aint, Asigned, Acode, Adouble, Aname FROM A_updates WHERE Akey <= 100

join_3_ncl
SELECT A_uniques.Asigned, A_uniques.Adate, A_hundred.Asigned, A_hundred.Adate, A_tenpct.Asigned, A_tenpct.Adate FROM A_uniques, A_hundred, A_tenpct WHERE A_uniques.Acode = A_hundred.Acode AND A_uniques.Acode = A_tenpct.Acode AND A_uniques.Acode = 'BENCHMARKS'

sel_10pct_ncl
SELECT Akey, Aint, Asigned, Acode, Adouble, Aname FROM A_tenpct WHERE Aname = 'THE+ASAP+BENCHMARKS+'

info_retrieval
SELECT count(Akey) FROM A_tenpct WHERE Aname = 'THE+ASAP+BENCHMARKS+' AND Aint <= 100000000 AND Asigned between 1 and 99999999 AND not (Afloat between -450000000 and 450000000) AND Adouble > 600000000 AND Adecim < -600000000

join_2_cl
SELECT A_uniques.Asigned, A_uniques.Aname, A_hundred.Asigned, A_hundred.Aname FROM A_uniques, A_hundred WHERE A_uniques.Akey = A_hundred.Akey AND A_uniques.Akey = 1000

join_2
SELECT distinct A_uniques.Asigned, A_uniques.Aname, A_hundred.Asigned, A_hundred.Aname FROM A_uniques, A_hundred WHERE A_uniques.Aaddress = A_hundred.Aaddress AND A_uniques.Aaddress = 'SILICON VALLEY'

variable_select (low selectivity)
SELECT Akey, Aint, Asigned, Acode, Adouble, Aname FROM A_tenpct WHERE Asigned <= -500000000

variable_select (high selectivity)
SELECT Akey, Aint, Asigned, Acode, Adouble, Aname FROM A_tenpct WHERE Asigned <= -250000000

join_4_cl
SELECT A_uniques.Adate, A_hundred.Adate, A_tenpct.Adate, A_updates.Adate FROM A_uniques, A_hundred, A_tenpct, A_updates WHERE A_uniques.Akey = A_hundred.Akey AND A_uniques.Akey = A_tenpct.Akey AND A_uniques.Akey = A_updates.Akey AND A_uniques.Akey = 1000

proj_100
SELECT distinct Aaddress, Asigned FROM A_hundred

join_4_ncl
SELECT A_uniques.Adate, A_hundred.Adate, A_tenpct.Adate, A_updates.Adate FROM A_uniques, A_hundred, A_tenpct, A_updates WHERE A_uniques.Acode = A_hundred.Acode AND A_uniques.Acode = A_tenpct.Acode AND A_uniques.Acode = A_updates.Acode AND A_uniques.Acode = 'BENCHMARKS'

proj_10pct
SELECT distinct Adecim FROM A_tenpct

sel_1_ncl
SELECT Akey, Aint, Asigned, Acode, Adouble, Aname FROM A_updates WHERE Acode = 'BENCHMARKS'

join_2_ncl
SELECT A_uniques.Asigned, A_uniques.Aname, A_hundred.Asigned, A_hundred.Aname FROM A_uniques, A_hundred WHERE A_uniques.Acode = A_hundred.Acode AND A_uniques.Acode = 'BENCHMARKS'

join_1_1pct
SELECT A_uniques.Akey, A_uniques.Aname, A_tenpct.Akey, A_tenpct.Asigned FROM A_uniques, A_tenpct WHERE A_uniques.Akey >= 1000000000 AND A_tenpct.Akey <= A_uniques.Akey AND A_tenpct.Akey >= 990000000

integrity_test a
SELECT A_uniques.Akey, A_uniques.Aname, A_tenpct.Akey, A_tenpct.Asigned FROM A_uniques, A_tenpct WHERE A_uniques.Akey >= 1000000000 AND A_tenpct.Akey <= A_uniques.Akey AND A_tenpct.Akey >= 990000000

integrity_test b
UPDATE A_hundred SET Asigned = -500000000 WHERE Aint = 0

integrity_restore
DELETE FROM A_hundred WHERE Aint = 0

bulk_save
INSERT INTO A_saveupdates SELECT * FROM A_updates WHERE Akey BETWEEN 5000 AND 5999

bulk_modify
UPDATE A_updates SET Akey = Akey - 100000 WHERE Akey BETWEEN 5000 AND 5999

append_duplicate
INSERT INTO A_updates VALUES (6000, 0, 60000, 39997.90, 50005.00, 50005.00, '1985/11/10', 'CONTROLLER', 'ALICE IN WONDERLAND', 'UNIVERSITY OF ILLINOIS AT CHICAGO')

remove_duplicate
DELETE FROM A_updates WHERE Akey = 6000 AND Aint = 0

app_t_mid
INSERT INTO A_updates VALUES (5005, 5005, 50005, 50005.00, 50005.00, 50005.00, '1988/1/1', 'CONTROLLER', 'ALICE IN WONDERLAND', 'UNIVERSITY OF ILLINOIS AT CHICAGO')

mod_t_mid
UPDATE A_updates SET Akey = -5000 WHERE Akey = 5005

del_t_mid
DELETE FROM A_updates WHERE Akey = -5000

app_t_end
INSERT INTO A_updates VALUES (1000000001, 5005, 50005, 50005.00, 50005.00, 50005.00, '1988/1/1', 'CONTROLLER', 'ALICE IN WONDERLAND', 'UNIVERSITY OF ILLINOIS AT CHICAGO')

mod_t_end
UPDATE A_updates SET Akey = -1000 WHERE Akey = 1000000001

del_t_end
DELETE FROM A_updates WHERE Akey = -1000

app_t_mid
INSERT INTO A_updates VALUES (5005, 5005, 50005, 50005.00, 50005.00, 50005.00, '1988/1/1', 'CONTROLLER', 'ALICE IN WONDERLAND', 'UNIVERSITY OF ILLINOIS AT CHICAGO')

mod_t_cod
UPDATE A_updates SET Acode = 'SQL+GROUPS' WHERE Akey = 5005

del_t_mid
DELETE FROM A_updates WHERE Akey = -5000

app_t_mid
INSERT INTO A_updates VALUES (5005, 5005, 50005, 50005.00, 50005.00, 50005.00, '1988/1/1', 'CONTROLLER', 'ALICE IN WONDERLAND', 'UNIVERSITY OF ILLINOIS AT CHICAGO')

mod_t_int
UPDATE A_updates SET Aint = 50015 WHERE Akey = 5005

del_t_mid
DELETE FROM A_updates WHERE Akey = -5000

bulk_append
INSERT INTO A_updates SELECT * FROM A_saveupdates

bulk_delete
DELETE FROM A_updates WHERE Akey < 0

CREATE TABLE SEQUENCE

Create A_updates Table
CREATE TABLE A_updates (Akey INTEGER NOT NULL, Aint INTEGER NOT NULL, Asigned INTEGER , Afloat REAL NOT NULL, Adouble REAL NOT NULL, Adecim REAL NOT NULL, Adate DATETIME NOT NULL, Acode CHAR (10) NOT NULL, Aname CHAR (20) NOT NULL, Aaddress VARCHAR (80) NOT NULL)

Create A_uniques Table
CREATE TABLE A_uniques (Akey INTEGER NOT NULL, Aint INTEGER NOT NULL, Asigned INTEGER , Afloat REAL NOT NULL, Adouble REAL NOT NULL, Adecim REAL NOT NULL, Adate DATETIME NOT NULL, Acode CHAR (10) NOT NULL, Aname CHAR (20) NOT NULL, Aaddress VARCHAR (80) NOT NULL)

Create A_hundred Table
CREATE TABLE A_hundred (Akey INTEGER NOT NULL, Aint INTEGER NOT NULL, Asigned INTEGER , Afloat REAL NOT NULL, Adouble REAL NOT NULL, Adecim REAL NOT NULL, Adate DATETIME NOT NULL, Acode CHAR (10) NOT NULL, Aname CHAR (20) NOT NULL, Aaddress VARCHAR (80) NOT NULL)

Create A_tenpct Table
CREATE TABLE A_tenpct (Akey INTEGER NOT NULL, Aint INTEGER NOT NULL, Asigned INTEGER , Afloat REAL NOT NULL, Adouble REAL NOT NULL, Adecim REAL NOT NULL, Adate DATETIME NOT NULL, Acode CHAR (10) NOT NULL, Aname CHAR (20) NOT NULL, Aaddress VARCHAR (80) NOT NULL)

Create A_tiny Table
CREATE TABLE A_tiny (Akey INTEGER NOT NULL)

Create A_saveupdates Table
CREATE TABLE A_saveupdates (Akey INTEGER NOT NULL, Aint INTEGER NOT NULL, Asigned INTEGER , Afloat REAL NOT NULL, Adouble REAL NOT NULL, Adecim REAL NOT NULL, Adate DATETIME NOT NULL, Acode CHAR (10) NOT NULL, Aname CHAR (20) NOT NULL, Aaddress VARCHAR (80) NOT NULL)

Create A_sel100seq Table
CREATE TABLE A_sel100seq (Akey INTEGER NOT NULL, Aint INTEGER NOT NULL, Asigned INTEGER , Afloat REAL NOT NULL, Adouble REAL NOT NULL, Adecim REAL NOT NULL, Adate DATETIME NOT NULL, Acode CHAR (10) NOT NULL, Aname CHAR (20) NOT NULL, Aaddress VARCHAR (80) NOT NULL)

Create A_sel100rand Table
CREATE TABLE A_sel100rand (Akey INTEGER NOT NULL, Aint INTEGER NOT NULL, Asigned INTEGER , Afloat REAL NOT NULL, Adouble REAL NOT NULL, Adecim REAL NOT NULL, Adate DATETIME NOT NULL, Acode CHAR (10) NOT NULL, Aname CHAR (20) NOT NULL, Aaddress VARCHAR (80) NOT NULL)

Create A_integrity_temp Table
CREATE TABLE A_integrity_temp (Akey INTEGER NOT NULL, Aint INTEGER NOT NULL, Asigned INTEGER , Afloat REAL NOT NULL, Adouble REAL NOT NULL, Adecim REAL NOT NULL, Adate DATETIME NOT NULL, Acode CHAR (10) NOT NULL, Aname CHAR (20) NOT NULL, Aaddress VARCHAR (80) NOT NULL)

INSERT DATA SEQUENCE

Initialize A_updates Table
INSERT INTO A_updates (Akey,Aint,Asigned,Afloat,Adecim ,Adouble,Adate,Acode,Aname,Aaddress) VALUES (?,?,?,?,?,?,?,?,?,?)

Initialize A_uniques Table
INSERT INTO A_uniques(Akey,Aint,Asigned,Afloat,Adouble, Adecim,Adate,Acode,Aname,Aaddress) VALUES (?,?,?,?,?,?,?,?,?,?)

Initialize A_hundred Table
INSERT INTO A_hundred(Akey,Aint,Asigned,Afloat,Adouble, Adecim,Adate,Acode,Aname,Aaddress) VALUES (?,?,?,?,?,?,?,?,?,?)

Initialize A_tenpct Table
INSERT INTO A_tenpct(Akey,Aint,Asigned,Afloat,Adouble,Adecim,Adate,Acode,Aname,Aaddress) VALUES (?,?,?,?,?,?,?,?,?,?)

Initialize A_tiny Table
INSERT INTO A_tiny(Akey) VALUES (?)

CREATE INDEX SEQUENCE

Create A_updates Index 1
create index PI_A_updates_key on A_updates (Akey)

Create A_updates Index 2
create index SI_A_updates_decim on A_updates (Adecim)

Create A_updates Index 3
create index SI_A_updates_int on A_updates (Aint)

Create A_updates Index 4
create index SI_A_updates_code on A_updates (Acode)

Create A_updates Index 5
create index SI_A_updates_dbl on A_updates (Adouble)

Create A_uniques Index 1
create index PI_A_uniques_key on A_uniques (Akey)

Create A_uniques Index 2
create index SI_A_uniques_code on A_uniques (Acode)

Create A_hundred Index 1
create index PI_A_hundred_key on A_hundred (Akey)

Create A_hundred Index 2
create index SI_A_hundred_code on A_hundred (Acode)

Create A_tenpct Index 1
create index PI_A_tenpct_key on A_tenpct (Akey, Acode)

Create A_tenpct Index 2
create index SI_A_tenpct_int on A_tenpct (Aint)

Create A_tenpct Index 4
create index SI_A_tenpct_double on A_tenpct (Adouble)

Create A_tenpct Index 5
create index SI_A_tenpct_float on A_tenpct (Afloat)

Create A_tenpct Index 6
create index SI_A_tenpct_decim on A_tenpct (Adecim)

Create A_tenpct Index 7
create index SI_A_tenpct_name on A_tenpct (Aname)

Create A_tenpct Index 8
create index SI_A_tenpct_code on A_tenpct (Acode)

DELETE TABLE SEQUENCE

Delete A_updates Table
DROP TABLE A_updates

Delete A_uniques Table
DROP TABLE A_uniques

Delete A_hundred Table
DROP TABLE A_hundred

Delete A_tenpct Table
DROP TABLE A_tenpct

Delete A_tiny Table
DROP TABLE A_tiny

Delete A_saveupdates Table
DROP TABLE A_saveupdates

Delete A_sel100seq Table
DROP TABLE A_sel100seq

Delete A_sel100rand Table
DROP TABLE A_sel100rand

Delete A_integrity_temp Table
DROP TABLE A_integrity_temp

AGGREGATE FUNCTIONS

Scalar aggregate with and without key
SELECT min(Afloat) FROM A_hundred GROUP BY Aname

JOIN 2 TABLES

SELECT A_hundred.Aint, A_hundred.Akey, A_updates.Afloat, A_updates.Asigned, A_updates.Adouble FROM A_updates, A_hundred WHERE A_updates.Akey = A_hundred.Akey AND A_hundred.Akey < 1000

JOIN 3 TABLES

SELECT A_uniques.Asigned, A_uniques.Adate, A_hundred.Asigned, A_hundred.Adate, A_tenpct.Asigned, A_tenpct.Adate FROM A_uniques, A_hundred, A_tenpct WHERE A_uniques.Akey = A_hundred.Akey AND A_uniques.Akey = A_tenpct.Akey AND A_uniques.Akey = 1000

JOIN 4 TABLES

SELECT A_uniques.Adate, A_hundred.Adate, A_tenpct.Adate, A_updates.Adate FROM A_uniques, A_hundred, A_tenpct, A_updates WHERE A_uniques.Akey = A_hundred.Akey AND A_uniques.Akey = A_tenpct.Akey AND A_uniques.Akey = A_updates.Akey AND A_uniques.Akey = 1000

SELECT SEQUENCE

Select everything from table
SELECT * FROM A_updates

Select first 1000 rows
SELECT * FROM A_updates WHERE Akey <= 1000

Select unique values from Aaddress and Asigned
SELECT DISTINCT Aaddress, Asigned FROM A_updates

Appendix C - System Architechure

(unavailable for pdf format)

BIBLIOGRAPHY

Ramez Elmasri and Shamkant Navathe (1994). Fundamentals of Database Systems Menlo Park, CA: Addison-Wesley.

Free Software Foundation (1998). Available: http://www.gnu.org/fsf/fsf.html.

Benchmark Handbook (1998). Available: http://www.benchmarkresources.com/handbook/5-1.html.

Your On-line Linux Resource (1999). Available: http://www.redhat.com.

Oracle 8 Installation Guide for Linux (1998). Available: http://www.oracle.com/download/Linux/oracle8/html/instguide/server805/a66251/toc.htm.

MySQL Reference Manual (1998). Available: http://www.mysql.com/Manual_chapter/manual_toc.html.

Seng, Jia-Lang (1994). Benchmarking Relational Database Systems – Overview. Available: http://www.twnic.net/contest/winner/week3/Paper/sixth/11.html.

Michael Abbey and Michael Corey (1997). Oracle 8: A Beginner's Guide Berkeley, CA: McGraw-Hill.