Batch: A1 **Roll No.:** 1611015

Experiment No. 03

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Database Tuning

Objective: Tuning the database to improve system performance

Expected Outcome of Experiment:

CO1 : Design and tune database.

Books/ Journals/ Websites referred:

1. *Elmasri & Navathe " fundamentals of Database Systems" V edition. PEARSON Education.*
2. *Korth, Silberschatzsu darshan "Database systems, concepts" 5th edition McGraw Hill.*
3. *Raghu Ramkrishnan & Johannes Gehrke "Database Management System" Tata McGraw Hill. III edition.*

Pre Lab/ Prior Concepts: Database, ER diagram, Relation mapping, SQL

Implementation Details:

INDEX TUNING:

The reasons for tuning indexes are as follows:

i) Certain queries may take too long to run for lack of index.

ii) Certain indexes may not get utilized at all.

iii) Certain indexes may be causing excessive overhead because the index is on a attribute that undergoes frequent changes.

How and Effect: Indexes can be created on single as well as a group of attributes:

FOR SIMPLE INDEX (1 attribute):

Before:

Your SQL query has been executed successfully.

```
EXPLAIN SELECT * FROM course_enrolled WHERE whether_completed="No"
```

[Edit inline] [Edit] [Skip Explain SQL] [Analyze Explain at mariadb.org] [Create PHP code]

+ Options

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	course_enrolled	ALL	NULL	NULL	NULL	NULL	3	Using where

After:

Your SQL query has been executed successfully.

```
EXPLAIN SELECT * FROM course_enrolled where whether_completed="No"
```

[Edit inline] [Edit] [Skip Explain SQL] [Analyze Explain at mariadb.org] [Create PHP code]

+ Options

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	course_enrolled	ref	course_status	course_status	7	const	2	Using index condition

Initially, the query was searching the whole table taking more time.
However, after creating an index, the same query uses the index 'course_status' and executes faster.

FOR COMPOSITE INDEX (Multiple Attributes):

Before:

Your SQL query has been executed successfully.

```
EXPLAIN SELECT * FROM COURSE WHERE start_date>='2018-09-01' AND end_date<='2018-09-30'
```

[Edit inline] [Edit] [Skip Explain SQL] [Analyze Explain at mariadb.org] [Create PHP code]

+ Options

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	COURSE	ALL	NULL	NULL	NULL	NULL	2	Using where

After:

Your SQL query has been executed successfully.

```
EXPLAIN SELECT * FROM COURSE WHERE start_date>='2018-09-01' AND end_date<='2018-09-30'
```

[Edit inline] [Edit] [Skip Explain SQL] [Analyze Explain at mariadb.org] [Create PHP code]

+ Options

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	COURSE	range	course_start_end	course_start_end	3	NULL	1	Using index condition

Initially, the query had to check the whole table for combination of both the attributes. However, after creating the index, it can apply binary search on the index since the index is sorted and hence, can reduce the time of execution drastically.

QUERY TUNING:

The reasons for query tuning are:

i) A query issues too many disk accesses.

ii) A query plan shows that the relevant indexes are not being used.

How and Effect:

For Tables:

```
CREATE TABLE course (
  course_id int(11) NOT NULL,
  faculty_id int(11) NOT NULL,
  course_name varchar(100) NOT NULL,
  start_date date NOT NULL,
  end_date date NOT NULL,
  about varchar(1000) NOT NULL,
  syllabus blob NOT NULL
);
CREATE TABLE faculty (
  faculty_id int(11) NOT NULL,
  faculty_fname varchar(50) NOT NULL,
  faculty_lname varchar(50) NOT NULL,
  email varchar(50) NOT NULL
)
```

1. Tuning count(*):

Select count(*)

From faculty,course

Where faculty.faculty_id=course.faculty_id;

```
Select count(*)

From faculty,course

Where faculty.faculty_id=course.faculty_id

--------------------------------------------------------------------------------
ERRORS SECTION
--------------------------------------------------------------------------------
- ORA-00942: table or view does not exist

--------------------------------------------------------------------------------

ALSO CONSIDER ABOUT

Instead of COUNT(*) use COUNT(1)
```

Hence we use:

SELECT count(1)

FROM `faculty` as a,`course` as b

Where a`.faculty_id`=b.`faculty_id`;

2. Tuning *:

SELECT *

FROM course

the sql query becomes faster if we use the actual columns names (required columns instead of all) in SELECT statement instead of than '*'.

Therefore we use:

SELECT course_id,faculty_id,start_date,end_date,syllabus

FROM course

3. Tuning IN:

SELECT *

FROM COURSE c

WHERE faculty_id IN

(SELECT faculty_id FROM FACULTY);

IN has slowest performance,use EXISTS is efficient when most of filter criteria is in main query.

therefore we use:

SELECT *

FROM COURSE c

WHERE EXISTS(

SELECT * FROM FACULTY f WHERE c.faculty_id=f.faculty_id);

For Tables:

```
CREATE TABLE quiz_ans (
  student_id int(11) NOT NULL,
  course_id int(11) NOT NULL,
  quiz_id int(11) NOT NULL,
  ques_num int(11) NOT NULL,
  answer varchar(50) NOT NULL
);
CREATE TABLE student (
  student_id` int(11) NOT NULL,
  student_fname varchar(50) NOT NULL,
  student_lname varchar(50) NOT NULL,
  email varchar(50) NOT NULL)
```

4. Using LIKE instead of SUBSTR:

```
SELECT student_id, student_fname, email
FROM student
WHERE SUBSTR(student_fname,1,3) = 'sur';
```

SUBSTR will have to search through all data, whereas LIKE will make use of indexes and hence will be efficient.

Therefore we use:

```
SELECT student_id, student_fname, email
FROM student
WHERE student_fname LIKE 'sur';
```

5. Using alternative names:

SELECT *

FROM student,quiz_ans

WHERE student.student_id=quiz_ans.student_id;

Instead of using name of tables again and again ,alternative names ie. Aliases can be used for tables

Therefore we use:

SELECT *

FROM student as s,quiz_ans as q

WHERE s.student_id=q.student_id;

6. UNION ALL instead of UNION

SELECT student_id, student_fname, email
FROM student UNION

SELECT quiz_num, student_id
FROM quiz_ans

UNION ALL can perform same task faster than efficiently than UNION

We use:

SELECT student_id, student_fname, email
FROM student UNION ALL

SELECT quiz_num, student_id
FROM quiz_ans

For Tables:

```
CREATE TABLE discussion_forum_ans(
  course_id int(11) NOT NULL,
  thread_id int(11) NOT NULL,
  user_type varchar(10) NOT NULL,
  id int(11) NOT NULL,
  answer text NOT NULL)
CREATE TABLE discussion_forum_ques(
  course_id int(11) NOT NULL,
  student_id int(11) NOT NULL,
  thread_id int(11) NOT NULL,
  question varchar(500) NOT NULL
)
```

7. Using INNER JOIN instead of WHERE

```
SELECT  question,answer
FROM discussion_forum_ques q,discussion_forum_ans a
WHERE q.thread_id=a.thread_id
```

This type of join creates a Cartesian Join, also called a Cartesian Product or CROSS JOIN. In a Cartesian Join, all possible combinations of the variables are created. This is an inefficient use of database resources, as the database has done more work than required. Cartesian Joins are especially problematic in large-scale databases, as a Cartesian Join of two large tables could create billions or trillions of results.

Hence we inner join as:

SELECT question,answer

FROM discussion_forum_ques q

INNER JOIN discussion_forum_ans a

ON q.thread_id=a.thread_id;

8. Avoiding use of DISTICT :

SELECT DISTINCT usertype,ans

FROM discussion_forum_ans

Distinct is used to remove duplicates in table, however for that a large amount of processing power is required. Additionally, data may be grouped to the point of being inaccurate. To avoid using SELECT DISTINCT, more fields should be selected to create unique results.

Hence we use:

SELECT usertype,ans,thread_id

FROM discussion_forum_ans

9. Limiting number of records using LIMIT:

SELECT thread_id,question

FROM discussion_forum_que

Here all the questions with thread ids will be displayed.if we want to see only sample questions we can limit no of records to be displayed by using LIMIT keyword

Hence we can use:

SELECT thread_id,question

FROM discussion_forum_que

LIMIT 10

Now we will see only 10 records as output.

DATABASE TUNING:

Reasons:
Sometimes, the conceptual schema is designed such that certain frequently used queries are exceuted rather slowly. In such cases, it is necessary to tune the schema to increase performance.

How and Effect:

1.
Course :

Course_id	Faculty_id	Name	Start_Date	End_Date

Faculty :

F o r Faculty_id	First_Name	Last_Name	University	email	password

For getting name of faculty who created the course, 'Faculty' and 'Course' tables will be joined frequently on 'faculty_id' column. To simplify this join operation, both tables are de-normalized into one as :

Course_id	Faculty_id	Faculty_fname	Faculty_lname	Course_Name	Start_Date	End_Date

2.

Assignment_Answer :

Assignment_id	Student_id	Answer	Grade

Student :

Student_id	First_Name	Last_Name	email	password

For getting name of student who submitted assignment answer, 'Student' and 'Assignment_ans' tables are joined frequently on 'student_id' column. To simplify this join operation, both tables are de-normalized into one as :

Assignment_id	Student_id	Student_fname	Student_lname	Answer	Grade

Conclusion:

Index tuning, Database tuning and Query tuning related to Virtual Classroom database was successfully done.

Post Lab Descriptive Questions:

1. What are the factors that influence Physical Database Design ?

Following factors affect physical database design:

A. Analyzing the database queries and transactions

For each query, the following information is needed.

1. The files that will be accessed by the query;
2. The attributes on which any selection conditions for the query are specified;
3. The attributes on which any join conditions or conditions to link multiple tables or objects for the query are specified;
4. The attributes whose values will be retrieved by the query

For each update transaction or operation, the following information is needed.

1. The files that will be updated;
2. The type of operation on each file (insert, update or delete);

3. The attributes on which selection conditions for a delete or update operation are specified;
4. The attributes whose values will be changed by an update operation.

B. Analyzing the expected frequency of invocation of queries and transactions

a. The expected frequency information, along with the attribute information collected on each query and transaction, is used to compile a cumulative list of expected frequency of use for all the queries and transactions.
b. It is expressed as the expected frequency of using each attribute in each file as a selection attribute or join attribute, over all the queries and transactions.
c. 80-20 rule
   i. 20% of the data is accessed 80% of the time

C. Analyzing the time constraints of queries and transactions

a) Performance constraints place further priorities on the attributes that are candidates for access paths.
b) The selection attributes used by queries and transactions with time constraints become higher-priority candidates for primary access structure.

D. Analyzing the expected frequencies of update operations

A minimum number of access paths should be specified for a file that is updated frequently.

E. Analyzing the uniqueness constraints on attributes

Access paths should be specified on all candidate key attributes — or set of attributes — that are either the primary key or constrained to be unique.

Date: 11/09/2018