Index in MySQL and Normalization

INDEX in MySQL

Suppose you have a company of 100000 employees and a phone book that contains all the names and phone numbers of employees is maintained. Let’s say you want to find Rajesh Kumar's phone number. Knowing that the names are alphabetically ordered, you first look for the page where the first name is Rajesh, then you look for Kumar and his phone number.

Now, if the names in the phone book were not sorted alphabetically, you would need to go through all pages, reading every name on it until you find Rajesh Kumar. This is called sequential searching. You go over all the entries until you find the person with the phone number that you are looking for.

Relating the phone book to the database table, if you have the table phonebooks and you have to find the phone number of Rajesh Kumar, you would perform the following query:

```
SELECT phone_number
FROM phonebooks
WHERE first_name = 'Rajesh' AND
last_name = 'Kumar';
```

It is pretty easy. Although the query is fast, the database has to scan all the rows of the table until it finds the row. If the table has millions of rows, without an index, the data retrieval would take a lot of time to return the result.

*Is there any way to enhance performance of the system? Yes, using **index in MySQL.***

An **index** is a data structure such as B-Tree that improves the speed of data retrieval on a table at the cost of additional writes and storage to maintain it. The query optimizer may use indexes to quickly locate data without having to scan every row in a table for a given query.

- Primary Key and Unique key are types of index.
- Index can be applied on a single column or combination of more than column.
- By default, MySQL creates the B-Tree index if you don’t specify the index type.

```
CREATE TABLE t(
   col-name data-type(size) constraint,
   col-name data-type(size) constraint,
   ...
   col-name data-type(size) constraint,
```

```
   INDEX (col-name-1, col-name-1)
);
```

MySQL CREATE INDEX statement: To add an index for a column or a set of columns, you use the CREATE INDEX statement as follows:

```
CREATE INDEX index_name ON table_name (column_list);
```

To view list of existing index in the table-

```
SHOW INDEX FROM table-name;
```

To drop index-

```
ALTER TABLE table-name DROP INDEX index-name;
```

For the sake of brevity, let use take example of same table st. Take a look at the existing indexes-

```
mysql> SHOW INDEX FROM st;
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+-
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+-
| st    |          0 | PRIMARY  |            1 | rollNo      | A         |           9 |     NULL | NULL   |      | BTREE      |         |
| st    |          0 | email    |            1 | email       | A         |        NULL |     NULL | NULL   | YES  | BTREE      |         |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+-
```

Say we want to find student whose name is AED, but to view performance of the query precede query with EXPLAIN keyword

```
mysql> EXPLAIN SELECT * FROM st WHERE name = 'AED';
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | st    | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    9 |    11.11 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
```

Here you can see that all 9 records are scanned to find desired record.

Let us create index on the column name.

```
CREATE INDEX name_index ON st (name);
```

Again search for student whose name is AED, but to view performance of the query precede query with EXPLAIN keyword

```
mysql> EXPLAIN SELECT * FROM st WHERE name = 'AED';
+----+-------------+-------+------------+------+---------------+------------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key        | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | st    | NULL       | ref  | name_index    | name_index | 22      | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+------------+---------+-------+------+----------+-------+
```

See total indexes in the table st

```
mysql> SHOW INDEX FROM st;
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------
| Table | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------
```

```
| st    |          0 | PRIMARY    |            1 | rollNo      | A         |           9 |     NULL | NULL   |      | BTREE      |
| st    |          0 | email      |            1 | email       | A         |        NULL |     NULL | NULL   | YES  | BTREE      |
| st    |          1 | name_index |            1 | name        | A         |        NULL |     NULL | NULL   |      | BTREE      |
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------
```

Normalization

Consider the following student_and_exam table-

```
st_id  student_name   stream   exam_name   max_marks   result_date   headquater    description
1      Ramesh         Science  NEET        720         01-01-2022    Delhi         This exam is for Medical
2      Yogesh         Science  JEE         300         01-02-2022    Kanpur        This exam is for Engineering
3      Suresh         Arts     CLAT        150         20-03-2022    Bangalore     This exam is for Law
4      Usman          Commerce CA          400         24-03-2022    Ajmer         This exam is for Accountants
5      Joseph         Science  NEET        720         01-01-2022    Delhi         This exam is for Medical
```

For this table do following operations-

1. Change the maximum marks for NEET to 750 from 720.
2. Add a new student Gurpreet for CLAT or Add a new student Harpreet but we have no information about his exam.
3. Say all participants wants to quit CA course. Remove their details.

While doing the changes, You find problems like changing same data or repeating same data or losing some information etc.

- **Insertion Anamoly:** Add fake/repeated entries to maintain some information and it leads to data inconsistency.
- **Updation Anamoly:** To update a small information, one has to update multiple records.
- **Deletion Anamoly:** Deletion of one information leads to deletion of another type of information.

See a Solution, How to make the table such that insertion, updation & deletion anamolies can be eliminated

```
Table: student
st_id    student_name
1        Ramesh
2        Yogesh
3        Suresh
4        Usman
5        Joseph

Table: Exam
Exam_id  stream   exam_name   max_marks   result_date   headquater    description
E001     Science  NEET        720         01-01-2022    Delhi         This exam is for Medical
E002     Science  JEE         300         01-02-2022    Kanpur        This exam is for Engineering
E003     Arts     CLAT        150         20-03-2022    Bangalore     This exam is for Law
E004     Commerce CA          400         24-03-2022    Ajmer         This exam is for Accountants

Table: student_exam
st_id    Exam_id
1        E001
2        E002
3        E003
4        E004
5        E004
```

For these tables do following operations-

1. Change the maximum marks for NEET to 750 from 720.
2. Add a new student Gurpreet for CLAT or Add a new student Harpreet but we have no information about his exam..
3. Say all participants wants to quit CA course. Remove their details.

A well structured relations is a relation, which contains minimum data redundancy and allow users to insert, delete and update rows without causing data inconsistencies. To create such efficient tables, normalization is used.

> 💡 **Caution:** Using normalization it is not guaranteed that redundancy & anamolies will be eliminated completely but they can be minimized so summarization is; When tables are normalized then-
> Complexity & Access time: High
> Redundancy & Anamolies: Low

Difference between candidate key and primary key

Primary Key	Candidate Key
Primary Key is a unique and non-null key which identify a record uniquely in table. A table can have only one primary key.	Candidate key is also a unique key to identify a record uniquely in a table but a table can have multiple candidate keys.
Primary key column value can not be null.	Candidate key column can have null value.
Primary key is most important part of any relation or table.	Candidate key signifies as which key can be used as Primary Key.
Primary Key is a candidate key.	Candidate key may or may not be a primary key.

Consider the following table structure

```
Student (EnrollmentNumber, Email, Name, Phone, City, Age)
Candidate Keys: EnrollmentNumber, Email, Phone
Primary Keys: EnrollmentNumber
```

The columns in a candidate key are called prime attributes, and a column that does not occur in any candidate key is called a non-prime attribute.

```
Non-Prime Attributes: Name, Age, City
Prime Attributes: EnrollmentNumber, Email, Phone
```

We are having following normal forms.

1-NF
2-NF
3-NF

- As the level of normalization increases then the number of tables also increase and the same is for complexity.
- To achive the nth normal form the table must be in the (n-1)th normal form so we can say that if table is in the nth normal form then it will be in the (n - 1)th normal form or we can say that nth normal form is superset of (n - 1)th normal form.

1-NF (First Normal Form)

- No multi-values attributes are allowed
- Each attribute value must be atomic (Not further decomposible)

Consider the following table

```
Table: Trainee (Unnormalized)
trainee_id  name    hobbies
1           Rohan   Dance
2           Joseph  Travel, Music
3           Roshesh Reading, Surfing
```

To make this table in 1-NF, we can create separate row for each value of hobbies as this is only multi-values attribute

```
Table: Trainee (1-NF)
trainee_id  name      hobbies
1           Rohan     Dance
2           Joseph    Travel
2           Joseph    Music
3           Roshesh   Reading
3           Roshesh   Surfing
```

Another way to do the same is by creating another table

```
Table: Trainee
trainee_id  name
1           Rohan
2           Joseph
3           Roshesh

Table: hobbies
trainee_id  subject
1           Dance
2           Travel
2           Music
3           Reading
3           Surfing
```

Pre-requisute for second Normal Form (Functional dependency)

What is functional dependency?

In a relation R, we have two attributes that are A and B, the attribute B is said to be functionally dependent on attribute A if B can be uniquely identified using value of A or we can say that each value of A is associated with exactly one value of B. Functional dependency between A and B is represented as A ----> B. A is called determinant and B is called dependent.

e.g. Student name is functionally dependent on roll no [roll no ---> Student name]
e.g. tax payer details are functionally dependent on PAN [PAN ---> tax payer details]

Fully Functional Dependency
If X and Y are attribute set in a relation, Y is fully functional dependent on X, if Y is functionally dependent on X not on any proper subset of X

Example ABC -> D

D is fully functionally dependent on combination of A,B and C not on any subset of A,B or C
A -> D [False]
B -> D [False]
C -> D [False]
AB -> D [False]
BC -> D [False]
CA -> D [False]

An example

```
Table: cbse_scores_for_last_three_years
rollNo   session       cgpa
10253    2020-21       7.8
11698    2021-22       8.9
15986    2021-22       5.6
11698    2022-23       7.6
10253    2022-23       9.9

(rollNo, session) ---> cgpa [True]
(rollNo) ---> cgpa [False]
(session) ---> cgpa [False]
```

Partial Functional Dependency
If X and Y are attribute set in a relation, Y is partially functional dependent on X, if Y is functionally dependent on any proper subset of X

An Example

```
Table: orders
product_id  customer_id order_date  price
1           1           2022-01-01  2000
1           2           2023-01-01  2000
2           2           2021-02-02  3000
2           1           2020-02-02  3000

(product_id, customer_id) --> price [True]
(product_id) --> price [True]
(customer_id) --> price [False]
```

2-NF (Second Normal Form)

- The table must be in 1-NF
- No partial dependency will be there i.e. Non prime attributes should not be dependent on the subset of a candidate key.

We have to decompose the table in two parts such that

```
Table: orders
product_id  customer_id order_date
1           1           2022-01-01
1           2           2023-01-01
2           2           2021-02-02
2           1           2020-02-02

Table: product
product_id  price
1           2000
2           3000
```

Another example

```
Book(title, pubId, auId, price, auAddress)
(title, pubId, auId) -> price [true]
(title, pubId, auId) -> auAddress [true]
auId -> auAddress [true]

Decomposing the table to make them in second normal form
Book(title, pubId, auId, price)
Author(auId, auAddress)
```

Pre-requisute for third Normal Form (Transitive Functional dependency)

What is transitive functional dependency?
In transitive functional dependency, dependent is indirectly dependent on the determinant i.e. a -> b and b -> c the according to the axiom of trasitivity a -> c this is called transitive functional dependency.
Say we have a relation

```
score(score_id, student_id, subject_id, marks, exam_name, max_marks)
(student_id, subject_id) -> exam_name [true]
exam_name -> max_marks [true]
(student_id, subject_id) -> max_marks [true]
```

Here exam_name is a non-key attribute that is used to determine other attribute i.e. max_marks so we can say that one non-key attribute is determining another non key attribute such that the former non key attribute is dependent of key attribute. We have

transitivity dependency here to overcome this issue, we have to decompose the tables again

3-NF (Third Normal form)

- The table must be in the 2-NF
- Must not have transitive dependency

```
score (score_id, student_id, subject_id, marks, exam_name)
exam (exam_name, max_marks)
```

An Example

```
building(buildingid, contractor, fee)
buildingid -> contractor [True]
contractor -> fee [True]
buildingid -> fee [True]

Decompose the table to eliminate the transitive dependency
building (buildingid, contractor)
contractor (contractor, fee)
```