Department of Computer Engineering

University of Peradeniya

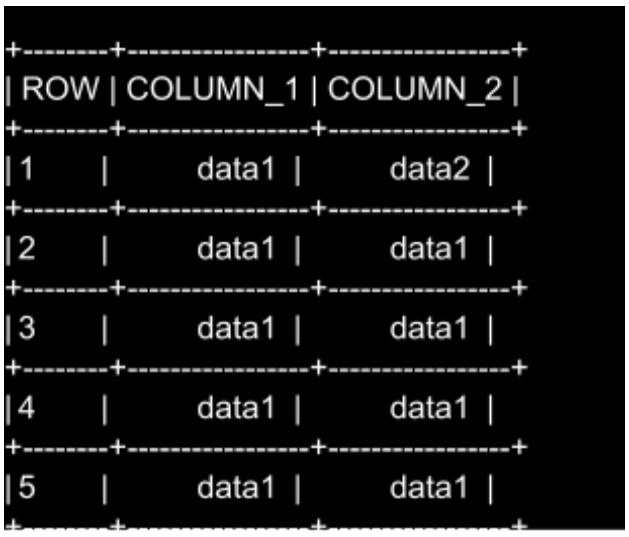
CO527 Advanced Database Systems

Lab Tutorial

Indexing

What is Indexing?

Indexing is a powerful structure in MySQL which can be leveraged to get the fastest response times from common queries. MySQL queries achieve efficiency by generating a smaller table, called an index, from a specified column or set of columns. These columns, called a key, can be used to enforce uniqueness. Below is a simple visualization of an example index using two columns as a key.



ROW	COLUMN_1	COLUMN_2
1	data1	data2
2	data1	data1
3	data1	data1
4	data1	data1
5	data1	data1

An index helps to speed up **SELECT** queries and **WHERE** clauses, but it slows down data input, with the **UPDATE** and the **INSERT** statements. Indexes can be created or dropped with no effect on the data.

Queries utilize indexes to identify and retrieve the targeted data, even if they are a combination of keys. Without an index, running that same query results in an inspection of every row for the needed data. Indexing produces a shortcut, with much faster query times on expansive tables. A textbook analogy may provide another common way to visualize how indexes function.

When to Enable Indexing?

Indexing is only advantageous for huge tables with regularly accessed information. For instance, to continue with our textbook analogy, it makes little sense to index a children's storybook with only a dozen pages. It's more efficient to simply read the book to find each occurrence of the word "turtle" than it would be to set up and maintain indexes, query for those indexes, and then review each page provided. In the computing world, those extra tasks surrounding indexing represent wasted resources which would be better purposed by not indexing.

Without indexes, when tables grow to enormous proportions, response times suffer from queries targeting those obtuse tables. Inefficient queries manifest into latency within application or website performance.

We commonly identify this latency by using the MySQL slow query log feature. Once a colossal table hits its tipping point, it reaches the potential for downtime for applications and websites. Conducting routine evaluations for growing databases establishes optimal database performance and sidesteps long queries' inherent interruptions.

What Information Does One Index?

Selecting what to index is probably the most challenging part to indexing your databases. Determining what is important enough to index and what is benign enough to not index. Generally speaking, indexing works best on those columns that are the subject of the WHERE clauses in your commonly executed queries. Consider the following simplified table:

If your queries rely on testing the WHERE clause using LAST_NAME and FIRST_NAME then indexing by these two columns would significantly increase query response times. Alternatively, if your queries rely on a simple ID lookup, indexing by ID would be the better choice.

There are several types of indexing structures built-in to

MySQL. To refer *How MySQL uses indexes* :

https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html

What is a Unique Index?

Another point for consideration when evaluating which columns to serve as the key in your index is whether to use the UNIQUE constraint. Setting the UNIQUE constraint will enforce uniqueness based on the configured indexing key. As with any key, this can be a single column or a concatenation of multiple columns. The function of this constraint ensures that there are no duplicate entries in the table based on the configured key.

Tip : Adding the UNIQUE constraint to an index further increases the write speed.

What is a Primary Key Index?

As commonly invoked as the UNIQUE constraint the PRIMARY KEY is used to optimize indexes. This constraint ensures that the designated PRIMARY KEY cannot be of a null value. As a result, a performance boost occurs when running on an InnoDB storage engine for the table in question. This boost is due to how InnoDB physically stores data, placing null valued rows in the key out of a contiguous sequence with rows that have values. Enabling this constraint ensures the rows of the table are kept in contiguous order for quicker responses.

Managing Indexes

Now lets' see some of the basics of manipulating indexes using MySQL syntax. In examples, we will include the creation, deletion, and listing of indexes.

Listing/Showing Indexes :

Tables can have multiple indexes. Managing indexes will inevitably require being able to list the existing indexes on a table. The syntax for viewing an index is below.

SHOW INDEX FROM *[tableName]*;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
employees	0	PRIMARY	1	emp_no	A	299290	NULL	NULL		BTREE
employees	1	employeesIndex	1	last_name	A	3218	NULL	NULL		BTREE
employees	1	employeesIndex	2	first_name	A	299290	NULL	NULL		BTREE
employees	1	employeesIndex	3	hire_date	A	299290	NULL	NULL		BTREE

Here we can see an existing index on the **employees** table named **employeesIndex** with an entry for each **Column_name** used as its key : **last_name**, **first_name**, and **hire_date**.

Creating Indexes :

Index creation has a simple syntax. The difficulty is in determining what columns need indexing and whether enforcing uniqueness is necessary. Below we will illustrate how to create indexes with and without a PRIMARY KEY and UNIQUE constraints.

As previously mentioned, tables can have multiple indexes. Multiple indexing is useful for creating indexes attuned to the queries required by your application or website. The default settings allow for up to 16 indexes per table, increasing this number but is generally more than is necessary. Indexes can be created during a table's creation or added onto the table as additional indexes later on. We will go over both methods below.

01) Example: Create a Table with a Standard Index

This example shows how to create a table with an index with the column name **ID** as the key.

```
CREATE TABLE [tableName] (
 ID int,
LName varchar(255),
 FName varchar(255),
 DOB varchar(255),
 LOC varchar(255),
 INDEX ( ID ));
```

02) Example: Create a Table with Unique Index & Primary Key

This example shows how to create a table with an index with the column name **ID** as the index key with both **PRIMARY KEY** and **UNIQUE** constraints enabled.

```
CREATE TABLE [tableName] (
 ID int,
LName varchar(255),
 FName varchar(255),
 DOB varchar(255),
 LOC varchar(255),
 PRIMARY KEY (ID),
UNIQUE INDEX ( ID ));
```

03) Example: Add an Index to Existing Table

Creating an index involves the CREATE INDEX statement, which allows you to name the index to specify the table and which column or columns to index.

CREATE INDEX [name of index] ON [table name] ([attribute-list]);

Example:

```
CREATE INDEX income_ix ON Company.salaries(salary);
```

would create an index on the salary field of the salaries table in the Company database. And the following query uses this income_ix index when returning the related result.

```
SELECT empno
FROM Salaries
WHERE salary>15000;
```

04) Example: Add an Index to Existing Table with Primary Key

Indexes can be created such that they prevent duplicate entries in the indexing field/fields.

CREATE UNIQUE INDEX [name of index] ON [table name] ([attribute-list]);

Example:

```
CREATE UNIQUE INDEX name_ix ON Company.employees (first_name,
last_name);
```

Deleting Indexes :

While managing indexes, you may find it necessary to remove some. Deleting indexes is also a very simple process, see the example below:

You can drop a created index with,

DROP INDEX [index_name] ON [table_name];

Example :

```
DROP INDEX name_ix ON Company.employees;
```

When should indexes be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided.

The following guidelines indicate when the use of an index should be reconsidered.

- ★ Indexes should not be used on small tables.
- ★ Tables that have frequent, large batch updates or insert operations.
- ★ Indexes should not be used on columns that contain a high number of NULL values.
- ★ Columns that are frequently manipulated should not be indexed.

References :

➔ https://www.w3schools.com/sql/sql_create_index.asp