RDBMS Terminology

One of the freely available, reliable and popular RDBMS is MySQL and we will be studying more about same in this series.
Before we proceed to explain the MySQL database system, let us have a look at few definitions related to the database.

Database - A database is a collection of tables, with related data.

Table - A table is a matrix with data. A table in a database looks like a simple spreadsheet.

Column - One column (data element) contains data of one and the same kind, for example, the column postcode.

Row - A row (= tuple, entity or record) is a group of related data, for example, the data of one subscription.

Primary Key - A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row. A table will have only one Primary Key

Unique Key - A unique key is a combination of one or more columns which can uniquely identify a row in the table. That combination cannot occur more than once in one table excluding rows having NULL values in key columns. A table can have more than one unique key.

Compound Key - A compound key (composite key) is a key that consists of multiple columns because one column is not sufficiently unique. Another term for the unique key which has more than one column.

Foreign Key - A foreign key is a linking pin between two tables.

Index - An index in a database resembles an index at the back of a book.

Referential Integrity - Referential Integrity makes sure that a foreign key value always points to an existing row.

MySQL Database:

- MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed, and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons:

- MySQL is released under an open-source license. So you have nothing to pay to use it.

- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.

- MySQL uses a standard form of the well-known SQL data language.

- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.

- MySQL works very quickly and works well even with large data sets.

- MySQL is very friendly to PHP, the most appreciated language for web development.

- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).

- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

Creating Database

Before doing anything else with the data, you need to create a database. A database is a container of data. It stores contacts, vendors, customers or any kind of data that you can think of. In MySQL, a database is a collection of objects that are used to store and manipulate data such as tables, database views, triggers, stored procedures, etc.

To create a database in MySQL, you use the CREATE DATABASE statement as follows:

CREATE DATABASE [IF NOT EXISTS] database_name;

Let's examine the CREATE DATABASE statement in greater detail:
 - Followed by the CREATE DATABASE statement is database name that you want to create. It is recommended that the database name should be as meaningful and descriptive as possible.

- The IF NOT EXISTS is an optional element of the statement. The IF NOT EXISTS statement prevents you from an error of creating a new database that already exists in the database server. You cannot have 2 databases with the same name in a MySQL database server. The IF NOT EXISTS first remove existing database and create new database.

For example, to create vision database, you can execute the CREATE DATABASE statement as follows:

CREATE DATABASE vision;

After executing the statement, MySQL returns a message to notify that the new database has been created successfully or not.

Displaying Databases

The SHOW DATABASES statement displays all databases in the MySQL database server. You can use the SHOW DATABASES statement to check the database that you've created or to see all the databases on the database server before you create a new database, for example:

SHOW DATABASES;

We have thee databases in the MySQL database server. The information_schema and mysql are the default databases that are available when we install MySQL, and the vision is the new database that we have created.

Before working with a particular database, you must tell MySQL which database you want to work with by using the

USE statement.
USE <<database_name>>;

You can select the vision database using the USE statement as follows:

USE vision;

From now all operations such as querying data, create new tables or stored procedures, function which you perform, will take effects on the current database.

Removing Databases

Removing database means you delete the database physically. All the data and related objects inside the database are permanently deleted and this cannot be undone, therefore it is very important to execute this query with extra cautions.

To delete a database, you use the DROP DATABASE statement as follows:

DROP DATABASE [IF EXISTS] <<database_name>>;

Followed the DROP DATABASE is the database name that you want to remove. Similar to the CREATE DATABASE statement, the IF EXISTS is an optional part of the statement to prevent you from removing a database that does not exist in the database server.

If you want to practice with the DROP DATABASE statement, you can create a new database, make sure that it is created and remove it. Take a look at the following queries:

CREATE DATABASE IF NOT EXISTS temp_database;
SHOW DATABASES;
DROP DATABASE IF EXISTS temp_database;

Rename Databases

RENAME {DATABASE | SCHEMA} db_name TO new_db_name;

This statement is added in Mysql 5.1.7 but was found to be dangerous and was remove after this release.
To perform the task of upgrading database names with the new encoding, use
ALTER DATABASE <<DBNAME>>;

MySQL - Data Types

Database table contains multiple columns with specific data types such as numeric or string. MySQL provides more data types other than just numeric or string. Each data type in MySQL can be determined by the following characteristics:

- Kind of values it can represent.
- The space that takes up and whether the values are fixed-length or variable-length.
- Does the values of the data type can be indexed.
- How MySQL compares the value of a specific data type.

Numeric Data Types

You can find all SQL standard numeric types in MySQL including exact number data type and approximate numeric data types including integer, fixed-point and floating point. In addition, MySQL also supports BIT data type for storing bit field values. Numeric types can be signed or unsigned except the BIT type.

Numeric Types	**Description**
TINYINT	A very small integer
SMALLINT	A small integer
MEDIUMINT	A medium-sized integer
INT	A standard integer
BIGINT	A large integer
DECIMAL	A fixed-point number
FLOAT	A single-precision floating-point number
DOUBLE	A double-precision floating-point number
BIT	A bit field

String Data Types

In MySQL, string can hold anything from plain text to binary data such as images and files. String can be compared and searched based on pattern matching by using the LIKE operator or regular expression. The following table shows you the string data types in MySQL:

String Types	**Description**
CHAR	A fixed-length non-binary (character) string
VARCHAR	A variable-length non-binary string
BINARY	A fixed-length binary string
VARBINARY	A variable-length binary string
TINYBLOB	A very small BLOB (binary large object)
BLOB	A small BLOB
MEDIUMBLOB	A medium-sized BLOB

LONGBLOB	A large BLOB
TINYTEXT	A very small non-binary string
TEXT	A small non-binary string
MEDIUMTEXT	A medium-sized non-binary string
LONGTEXT	A large non-binary string
ENUM	An enumeration; each column value may be assigned one enumeration member
SET	A set; each column value may be assigned zero or more set members

Date and Time Data Types

MySQL provides types for date and time as well as a combination of date and time. In addition, MySQL also provides timestamp data type for tracking the changes of a row in a table. If you just want to store the year without date and month, you can use YEAR data type. The following table illustrates the MySQLdate and time data types:

Date and Time Types	**Description**
DATE	A date value in 'CCYY-MM-DD' format
TIME	A time value in 'hh:mm:ss' format
DATETIME	A date and time value in 'CCYY-MM-DD hh:mm:ss' format
TIMESTAMP	A timestamp value in 'CCYY-MM-DD hh:mm:ss' format
YEAR	A year value in CCYY or YY format

Spatial Data Types

MySQL supports many spatial data types that contain various kind of geometrical and geographical values as shown in the following table:

Spatial Data Types	**Description**
GEOMETRY	A spatial value of any type
POINT	A point (a pair of X Y coordinates)
LINESTRING	A curve (one or more POINT values)
POLYGON	A polygon
GEOMETRYCOLLECTION	A collection of GEOMETRY values
MULTILINESTRING	A collection of LINESTRING values
MULTIPOINT	A collection of POINT values
MULTIPOLYGON	A collection of POLYGON values

MySQL - Create Database

MySQL CREATE TABLE syntax

In order to create a new table within a database, you use the MySQL CREATE TABLE statement. TheCREATE TABLE statement is one of the most complex statement in MySQL.

The following illustrates the syntax of the CREATE TABLE statement in the simple form:

```
CREATE TABLE [IF NOT EXISTS] table_name(
    column_list
) engine=table_type
```

Let's examine the syntax in greater detail:

- First, you specify the name of table that you want to create after the CREATE TABLE keywords. The table name must be unique within a database. The IF NOT EXISTS is an optional part of the statement that allows you to check if the table you are creating already exists in the database. If this is the case, MySQL will ignore the whole statement and it will not create any new table. It is highly recommended that you to use IF NOT EXISTS in every CREATE TABLE statement for preventing from an error of creating a new table that already exists.

- Second, you specify a list of columns for the table in the column_list section. Columns are separated by a comma (,).

- Third, you need to specify the storage engine for the table in the engine clause. You can use any storage engine such as InnoDB, MyISAM, HEAP, EXAMPLE, CSV, ARCHIVE, MERGE FEDERATED or NDBCLUSTER. If you don't declare the storage engine explicitly, MySQL will use InnoDB by default.

InnoDB became the default storage engine since MySQL version 5.5. The InnoDB table type brings many benefits of relational database management system such as ACID transaction, referential integrity and crash recovery. In the previous versions, MySQL used MyISAM as the default storage engine.

To define a column for the table in the CREATE TABLE statement, you use the following syntax:

```
column_name data_type[size] [NOT NULL|NULL] [DEFAULT value]
[AUTO_INCREMENT]
```

The most important components of the syntax above are:
- The column_name specifies the name of the column. Each column always associates with a specific data type and the size e.g., VARCHAR(255).
- The NOT NULL or NULL indicates that the column accepts NULL value or not.
- The DEFAULT value is used to specify the default value of the column.

- The AUTO_INCREMENT indicates that the value of column is increased by one whenever a new row is inserted into the table. **Each table has one and only one AUTO_INCREMENT column.**

If you want to set particular columns of the table as the primary key, you use the following syntax:

PRIMARY KEY (col1,col2,...)

Example of MySQL CREATE TABLE statement

```
CREATE TABLE IF NOT EXISTS tasks (
 task_id int(11) NOT NULL AUTO_INCREMENT,
 subject varchar(45) DEFAULT NULL,
 start_date DATE DEFAULT NULL,
 end_date DATE DEFAULT NULL,
 description varchar(200) DEFAULT NULL,
 PRIMARY KEY (task_id)
) ENGINE=InnoDB
```

MySQL - Sequence

MySQL sequence is used to generate unique numbers automatically for ID columns of tables.

In MySQL, a sequence is a list of integers generated in the ascending order i.e., 1,2,3… Many applications need sequences to generate unique numbers mainly for identification e.g., customer ID in CRM, employee number in HR, equipment number in services management system, etc.

To create a sequence in MySQL automatically, you set the AUTO_INCREMENT attribute to a column, which typically is primary key column. The following are rules that you must follow when you useAUTO_INCREMENT attribute:

- Each table has only one AUTO_INCREMENT column whose data type is typically integer or float which is very rare.
- The   AUTO_INCREMENT column must be indexed, which means it can be either PRIMARY KEY or UNIQUE index.
- The AUTO_INCREMENT column must have NOT NULL constraint. When you setAUTO_INCREMENT attribute to a column, MySQL will make it NOT NULL for you in case you don't define it explicitly.

MySQL create sequence example

The following example creates employees table whose emp_no column is AUTO_INCREMENTcolumn:

```
CREATE TABLE employees(
   emp_no INT(4) AUTO_INCREMENT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name  VARCHAR(50)
)ENGINE = INNODB;
```

How MySQL sequence works

The AUTO_INCREMENT column has the following attributes:

- The starting value of an AUTO_INCREMENT column is 1 and it is increased by 1 when you insert NULL value into the column or when you omit its value in the INSERT statement.
- To obtain the last generated sequence number, you use the **LAST_INSERT_ID()** function. You often use the last insert ID for the subsequent statements e.g., insert data into child tables. The last generated sequence is unique across sessions.In other words, if another connection generates a sequence number, from your connection you can obtain it by using the LAST_INSERT_ID()function. For more details on LAST_INSERT_ID() function, check it out the MySQL LAST_INSERT_ID() function tutorial.
- If you insert a new row into a table and specify a value for the sequence column, MySQL will insert the sequence number if the sequence number does not exist in the column or issue an error if it already exists. If you insert a new value that is greater than the next sequence number, MySQL will use the new value as the starting sequence number and generate a unique sequence number greater than the current one for the next use. This creates gaps in the sequence.
- If you use UPDATE statement to update an AUTO_INCREMENT column to a value that already exists, MySQL will issue a duplicate-key error if the column has a unique index. If you update anAUTO_INCREMENT column to a value that is larger than the existing values in the column, MySQL will use the next number of the last insert sequence number for the next row e.g., if the last insert sequence number is 3, you update it to 10, the sequence number for the new row is 4. See the example in the below section.
- If you use DELETE statement to delete the last insert row, MySQL may or may not reuse the deleted sequence number depending on the storage engine of the table. A MyISAM table does not reuse the deleted sequence numbers if you delete a row e.g., the last insert id in the table is 10, if you remove it, MySQL still generates the next sequence number which is 11 for the new row. Similar to MyISAM tables, InnoDB tables do use reuse sequence number when rows are deleted.

Once you set AUTO_INCREMENT attribute for a column, you can reset auto increment value in various ways e.g., by using ALTER TABLE statement.

Let's practice with the MySQL sequence.

First, insert two new employees into the employees table:

```
INSERT INTO employees(first_name,last_name)
VALUES('abc','xyz'),('xyz','abc');
```

Second, select data from the employees table:

```
SELECT * FROM employees;
```

Third, delete the second employee whose emp_no is 2:

DELETE FROM employees WHERE emp_no = 2;

Fourth, insert a new employee:

INSERT INTO employees(first_name,last_name) VALUES('abc','xyz');

Because the storage engine of the employees table is InnoDB, it does not reuse the deleted sequence number. The new row has emp_no 3.

Fifth, update an existing employee with emp_no 3 to 1:

UPDATE employees SET first_name = 'abc', emp_no = 1
WHERE emp_no = 3;

MySQL issued an error of duplicate entry for the primary key. Let's fix it:

UPDATE employees SET first_name = 'abc', emp_no = 10
WHERE emp_no = 3;

Sixth, insert a new employee after updating the sequence number to 10:

INSERT INTO employees(first_name,last_name)
VALUES('abc','xyz');

The next sequence number of the last insert is 4, therefore MySQL use 4 for the new row instead of 11.

MySQL - Primary Key

A primary key is a column or a set of columns that uniquely identifies each row in the table. The following are the rules that you must follow when you define a primary key for a table:

- A primary key must contain unique values. If the primary key consists of multiple columns, the combination of values in these columns must be unique.
- A primary key column cannot contain NULL values. It means that you have to declare the primary key column with NOT NULL attribute. If you don't, MySQL will force the primary key column as NOT NULL implicitly.
- A table has only one primary key.

Because MySQL works faster with integers, the primary key column's type should be an integer type e.g.,INT or BIGINT. You can choose a smaller integer type such as TINYINT, SMALLINT, etc., however you should make sure that the range of values of the integer type for the primary key is sufficient for storing all possible rows that the table may have.

A primary key column often has AUTO_INCREMENT attribute that generates a unique sequence for the key automatically. The the primary key of the next row is greater than the previous one.

MySQL creates an index named PRIMARY with PRIMARY type for the primary key in a table.

Defining MySQL PRIMARY KEY Constraints

MySQL allows you to to create a primary key by defining a primary key constraint when you create or modify the table.

MySQL allows you to create the primary key when you create the table by using the CREATE TABLEstatement. To create a PRIMARY KEY constraint for the table, you specify the PRIMARY KEY in the primary key column's definition.

The following example creates users table whose primary key is user_id column:

```
CREATE TABLE users(
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(40),
  password VARCHAR(255),
  email VARCHAR(255)
);
```

You can also specify the PRIMARY KEY at the end of the CREATE TABLE statement as follows:

```
CREATE TABLE roles(
  role_id INT AUTO_INCREMENT,
  role_name VARCHAR(50),
  PRIMARY KEY(role_id)
);
```

In case the primary key consists of multiple columns, you must specify them at the end of the CREATE TABLE statement. You put a coma-separated list of primary key columns inside parentheses followed the PRIMARY KEY keywords.

```
CREATE TABLE userroles(
  user_id INT NOT NULL,
  role_id INT NOT NULL,
  PRIMARY KEY(user_id,role_id),
  FOREIGN KEY(user_id) REFERENCES users(user_id),
  FOREIGN KEY(role_id) REFERENCES roles(role_id)
);
```

Besides creating the primary key that consists of user_id and role_id columns, the statement also created two foreign key constraints.

Defining MySQL PRIMARY KEY constraints using ALTER TABLE statement

If a table, for some reasons, does not have a primary key, you can use the ALTER TABLE statement to add a column that has all necessary primary key's characteristics to the primary key as the following statement:

```
ALTER TABLE table_name ADD PRIMARY KEY(primary_key_column);
```

The following example adds the id column to the primary key.

First, create t1 table without defining the primary key.

```
CREATE TABLE t1(
  id int,
  title varchar(255) NOT NULL
);
```

Second, add the id column to primary key of the t1 table.

```
ALTER TABLE t1 ADD PRIMARY KEY(id);
```

PRIMARY KEY vs. UNIQUE KEY vs. KEY

A KEY is a synonym for INDEX. You use KEY when you want to create an index for a column or a set of column that is not a part of a primary key or unique key.

A UNIQUE index creates a constraint for a column whose values must be unique. Unlike the PRIMARY index, MySQL allows NULL values in the UNIQUE index. A table can also have multiple UNIQUE indexes.

For example, the email and username of user in the users table must be unique. You can define UNIQUE indexes for the email and username column as the following statement:

Add a UNIQUE index for the username column.

ALTER TABLE users ADD UNIQUE INDEX username_unique (username ASC) ;

Add a UNIQUE index for the email column.

ALTER TABLE users ADD UNIQUE INDEX email_unique (email ASC) ;

MySQL - Foreign Key

A foreign key is a field in a table that matches a field of another table. A foreign key places constraints on data in the related tables that, which enables MySQL to maintain referential integrity.

We have two tables: customers and orders. Each customer has zero or more orders and each order belongs to only one customer. The relationship between customers table and orders table is one-to-many, and it is established by a foreign key in the orders table specified by thecustomerNumber field. The customerNumber field in the orders table relates to thecustomerNumber primary key field in customers table.

The customers table is called parent table or referenced table, and the orders table is known as child table or referencing table.

A foreign key has not only one column but also a set of columns. The columns in the child table often refer to the primary key columns in the parent table.

A table may have more than one foreign key, and each foreign key in the child table can have a different parent table.

A row in the child table must contain values that exist in the parent table e.g., each order record in theorders table must have a customerNumber that exists in the customers table. Multiple orders can refer to the same customer therefore this relationship is called one (customer) to many (orders), or one-to-many.

Sometimes, the child and parent table is the same table. The foreign key refers back to the primary key ofthe table e.g., the following employees table :

The reportTo column is a foreign key that refers to the employeeNumber column which is the primary key of the employees table to reflect the reporting structure between employees i.e., each employee reports to another employee and an employee can have zero or more direct reports.

The reportTo foreign key is also known as recursive or self-referencing foreign key.

Foreign keys enforce referential integrity that helps you maintain the consistency and integrity of the data automatically. For example, you cannot create an order for a non-existent customer.

In addition, you can set up a cascade on delete action for the customerNumber foreign key so that when you delete a customer in the customers table, all the orders associated with the customer are also deleted. This saves you time and efforts of using multiple
DELETE statements or a DELETE JOIN statement.

The same as deletion, you can also define a cascade on update action for the customerNumberforeign key to perform cross-table update without using multiple UPDATE statements or an UPDATE JOIN statement.

In MySQL, the InnoDB storage engine supports foreign keys so that you must create InnoDB tables in order to use foreign key constraints.

MySQL create table foreign key

```
CONSTRAINT constraint_name
FOREIGN KEY foreign_key_name (columns)
REFERENCES parent_table(columns)
ON DELETE action
ON UPDATE action
```

Let's examine the syntax in greater detail:

- The CONSTRAINT clause allows you to define constraint name for the foreign key constraint. If you omit it, MySQL will generate a name automatically.

- The FOREIGN KEY clause specifies the columns in the child table that refer to primary key columns in the parent table. You can put a foreign key name after FOREIGN KEY clause or leave it to let MySQL to create a name for you. Notice that MySQL automatically creates an index with the foreign_key_name name.

- The REFERENCES clause specifies the parent table and its columns to which the columns in the child table refer. The number of columns in child table and parent table specified in the FOREIGN KEY and REFERENCES must be the same.

- The ON DELETE clause allows you to define what happens to the records in the child table when the records in the parent table are deleted. If you omit the ON DELETE clause and delete a record in the parent table that has records in the child table refer to, MySQL will reject the deletion. In addition, MySQL also provides you with actions so that you can have other options such as ON DELETE CASCADE that lets MySQL to delete records in the child table that refer to a record in the parent table when the record in the parent table is deleted. If you don't want the related records in the child table to be deleted, you use the ON DELETE SET NULL action instead. MySQL will set the foreign key column values in the child table to NULL when the record in the parent table is deleted, with a condition that the foreign key column in the child table must accept NULL values. Notice that if you use ON DELETE NO ACTION or ON DELETE RESTRICT action, MySQL will reject the deletion.

- The ON UPDATE clause enables you to specify what happens to the rows in the child table when rows in the parent table are updated. You can omit the ON UPDATE clause to let MySQL to reject any update to the rows in the child table when the rows in the parent table are updated. The ON UPDATE CASCADE action allows you to perform cross-table update, and the ON UPDATE SET NULL action resets the values in the rows in the child table to NULL values when the rows in the parent table are updated. The ON UPDATE NO ACTION or UPDATE RESTRICT actions reject any updates.

MySQL create table foreign key example

The following example creates a dbdemo database and two tables: categories and products. Each category has one or more products and each product belongs to only one category. The cat_idfield in the products table is defined as a foreign key with **UPDATE ON CASCADE and DELETE ON RESTRICT** actions.

```
CREATE DATABASE IF NOT EXISTS dbdemo;
USE dbdemo;

CREATE TABLE categories(
  cat_id int not null auto_increment primary key,
  cat_name varchar(255) not null,
  cat_description text
) ENGINE=InnoDB;

CREATE TABLE products(
  prd_id int not null auto_increment primary key,
  prd_name varchar(355) not null,
  prd_price decimal,
  cat_id int not null,
  FOREIGN KEY fk_cat(cat_id)
  REFERENCES categories(cat_id)
  ON UPDATE CASCADE
  ON DELETE RESTRICT
)ENGINE=InnoDB;
```

MySQL add foreign key syntax

To add a foreign key to an existing table, you use the ALTER TABLE statement with the foreign key definition syntax above:

```
ALTER table_name
ADD CONSTRAINT constraint_name
FOREIGN KEY foreign_key_name(columns)
REFERENCES parent_table(columns)
ON DELETE action
ON UPDATE action
```

Now, let's add a new table named vendors and change the products table to include the vendor id field:
USE dbdemo;

```
CREATE TABLE vendors(
   vdr_id int not null auto_increment primary key,
   vdr_name varchar(255)
)ENGINE=InnoDB;

ALTER TABLE products
ADD COLUMN vdr_id int not null AFTER cat_id;
```

To add a foreign key to the products table, you use the following statement:

```
ALTER TABLE products
ADD FOREIGN KEY fk_vendor(vdr_id)
REFERENCES vendors(vdr_id)
ON DELETE NO ACTION
ON UPDATE CASCADE;
```

Now, the products table has two foreign keys, one refers to the categories table and another refers to the vendors table.

MySQL drop foreign key

You also use the ALTER TABLE statement to drop foreign key as the following statement:

```
ALTER TABLE table_name
DROP FOREIGN KEY constraint_name
```

In the statement above:

- First, you specify the table name from which you want to remove the foreign key.
- Second, you put the constraint name after the DROP FOREIGN KEY clause.

Notice that constraint_name is the name of the constraint specified when you created or added the foreign key to the table. If you omit it, MySQL generates a constraint name for you.

MySQL disable foreign key checks

Sometimes, it is very useful to disable foreign key checks e.g., when you load data into the tables that have foreign keys. If you don't disable foreign key checks, you have to load data into a proper order i.e., you have to load data into parent tables first and then child tables, which can be tedious. However if you disable the foreign key checks, you can load data into any orders.

Another example is that, unless you disable the foreign key checks, you cannot drop a table that is referenced by a foreign key constraint. When you drop a table, any constraints that you defined for the table are also removed.

To disable foreign key checks, you use the following statement:

SET foreign_key_checks = 0

And of course, you can enable it by using the statement below:

SET foreign_key_checks = 1

MySQL - Alter Table

MySQL ALTER TABLE statement that changes existing table structure such as adding or removing column, changing column attribute, etc.

MySQL ALTER TABLE syntax

The ALTER TABLE statement is used to change the structure of existing tables. You can use the ALTER TABLE statement to add or drop column, change data type of column, add primary key, rename table andmany more. The following illustrates the ALTER TABLE statement syntax:

ALTER TABLE table_name action1[,action2,…]

To alter an existing table:

- First, you specify the table name that you want to change after the ALTER TABLE keywords.
- Second, you list a set of actions that you want to apply to the table. An action can be anything such as

adding a new column, adding primary key, renaming table, etc. The ALTER TABLE statement allows you to apply multiple actions in a single ALTER TABLE statement, each action is separated by a comma (,).

Let's create a new table for practicing the ALTER TABLE statement.

We're going to create a new table named tasks in our sample database. The following is the script for creating the tasks table.

```
CREATE  TABLE tasks (
  task_id INT NOT NULL ,
  subject VARCHAR(45) NULL ,
  start_date DATE NULL ,
  end_date DATET NULL ,
  description VARCHAR(200) NULL ,
  PRIMARY KEY (task_id) ,
  UNIQUE INDEX task_id_UNIQUE (task_id ASC)
);
```

Changing columns using MySQL ALTER TABLE statement

Using MySQL ALTER TABLE statement to set auto-increment attribute for a column

Suppose you want the value of the task_id column to be increased automatically by one whenever youinsert a new record into the tasks table. To do this, you use the ALTER TABLE statement to set the attribute of the task_id column to
AUTO_INCREMENT as follows:

```
ALTER TABLE tasks
CHANGE COLUMN task_id task_id INT(11) NOT NULL AUTO_INCREMENT;
```

You can verify the change by adding some records to the tasks table.

```
INSERT INTO tasks(subject,start_date,end_date,description)
VALUES('Learn MySQL ALTER TABLE',Now(),Now(),'Practicing MySQL ALTER TABLE statement');

INSERT INTO tasks(subject,start_date,end_date,description)
VALUES('Learn MySQL CREATE TABLE',Now(),Now(),'Practicing MySQL CREATE TABLE statement');
```

And you can query data to see if the value of the task_id column is increased by 1 each time you insert a new record:

```
SELECT task_id, description FROM tasks
```

Using MySQL ALTER TABLE statement to add a new column into a table

Because of the new business requirement, you need to add a new column called complete to store the percentage of completion for each task in the tasks table. In this case, you can use the ALTER TABLE to add a new column to the tasks table as follows:

```
ALTER TABLE tasks
ADD COLUMN complete DECIMAL(2,1) NULL
AFTER description;
```

Using MySQL ALTER TABLE to drop a column from a table

Suppose you don't want to store the description of tasks in the tasks table and you have to remove it. The following statement allows you to remove the description column of the tasks table:

```
ALTER TABLE tasks
DROP COLUMN description;
```

Renaming table using MySQL ALTER TABLE statement

You can use the ALTER TABLE statement to rename a table. Notice that before renaming a table, you should take a serious consideration to see if the change affects both database and application layers.

The following statement rename the tasks table to work_items:

```
ALTER TABLE tasks
RENAME TO work_items;
```

MySQL - Temporary Table

In MySQL, a temporary table is a special type of table that allows you to store a temporary result set, which you can reuse several times in a single session. A temporary table is very handy when it is impossible or expensive to query data that requires a single SELECT statement with JOIN clauses. You often use temporary tables in stored procedures to store immediate result sets for the subsequent uses.

MySQL temporary tables have some additional features:

- A temporary table is created by using CREATE TEMPORARY TABLE statement. Notice that theTEMPORARY keyword is added between CREATE and TABLE keywords.

- MySQL drops the temporary table automatically when the session ends or connection is terminated. Of course, you can use the DROP TABLE statement to drop a temporary table explicitly when you are no longer use it.

- A temporary table is only available and accessible by the client who creates the table.

- Different clients can create a temporary table with the same name without causing errors because only the client who creates a temporary table can see it. However, in the same session, two temporary tables cannot have the same name.

- A temporary table can have the same name as an existing table in a database. For example, if you create a temporary table named employees in the sample database, the existing employeestable becomes inaccessible. Every query you issue against the employees table refers to theemployees temporary table. When you remove the employees temporary table, the permanentemployees table is available and accessible again. Though this is allowed however it is not recommended to create a temporary table whose name is same as a name of a permanent table because it may lead to a confusion. For example, in case the connection to the MySQL database server is lost and you reconnect to the server automatically, you cannot differentiate between the temporary table and the permanent table. In the worst case, you may issue a DROP TABLE statement to remove the permanent table instead of the temporary table, which is not expected.

Create MySQL temporary table

Like the CREATE TABLE statement, MySQL provides many options to create a temporary table. To create a temporary table, you just add the TEMPORARY keyword to the CREATE TABLE statement.

For example, the following statement creates a top 10 customers by revenue temporary table based on the result set of a

SELECT statement:

CREATE TEMPORARY TABLE top10customers

```
 SELECT p.customerNumber, c.customerName,
       FORMAT(SUM(p.amount),2) total
FROM payments p
INNER JOIN customers c ON c.customerNumber = p.customerNumber
GROUP BY p.customerNumber
ORDER BY total DESC
LIMIT 10
```

Now, you can query data from the top10customers temporary table as from a permanent table:

```
SELECT * FROM top10customers
```

Drop MySQL temporary table

You can use the DROP TABLE statement to remove temporary tables however it is good practice to use the DROP TEMPORARY TABLE statement instead. Because the DROP TEMPORARY TABLE removes only temporary tables, not the permanent tables. In addition, the DROP TEMPORARY TABLE statement helps you avoid the mistake of removing a permanent table when you name your temporary table the same as the name of the permanent table.

For example, to remove the top10customers temporary table, you use the following statement:

```
DROP TEMPORARY TABLE top10customers
```

Notice that if you try to remove a permanent table with the DROP TEMPORARY TABLE statement, you will get an error message saying that the table you are trying drop is unknown.

Note if you develop an application that uses a connection pooling or persistent connections, it is not guaranteed that the temporary tables are removed automatically when your application is terminated. Because the database connection that the application used may be still open and is placed in a connection pool for other clients to reuse it. This means you should always remove the temporary tables that you created whenever you are done with them.

MySQL - Database Index

Database index, or just index, helps speed up the retrieval of data from tables. When you query data from a table, first MySQL checks if the indexes exist, then MySQL uses the indexes to select exact physical corresponding rows of the table instead of scanning the whole table.

A database index is similar to an index of a book. If you want to find a topic, you look up in the index first, and then you open the page that has the topic without scanning the whole book.

It is highly recommended that you should create index on columns of table from which you often query the data. Notice that all primary key columns are in the primary index of the table automatically.

If index helps speed up the querying data, why don't we use indexes for all columns? If you create an index for every column, MySQL has to build and maintain the index table. Whenever a change is made to the records of the table, MySQL has to rebuild the index, which takes time as well as decreases the performance of the database server.

Creating MySQL Index

You often create indexes when you create tables. MySQL automatically add any column that is declared as PRIMARY KEY, KEY, UNIQUE or INDEX to the index. In addition, you can add indexes to the tables that already have data.

In order to create indexes, you use the CREATE INDEX statement. The following illustrates the syntax of the CREATE INDEX statement:

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name

USING [BTREE | HASH | RTREE]

ON table_name (column_name [(length)] [ASC | DESC],...)

First, you specify the index based on the table type or storage engine:

- UNIQUE means MySQL will create a constraint that all values in the index must be unique. Duplicate NULL value is allowed in all storage engine except BDB.

- FULLTEXT index is supported only by MyISAM storage engine and only accepted on column that has data type is CHAR, VARCHAR or TEXT.

- SPATIAL index supports spatial column and is available on MyISAM storage engine. In addition, the column value must not be NULL.

Then, you name the index and its type after the USING keyword such as BTREE, HASH or RTREE also based on the storage engine of the table.

Here are the storage engines of the table with the corresponding allowed index types:

Storage Engine	Allowable Index Types
MyISAM	BTREE, RTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE
NDB	HASH

Third, you declare table name and a list columns that you want to add to the index.

Example of creating index in MySQL

In the sample database, you can add officeCode column of the employees table to the index by using the CREATE INDEX statement as follows:

CREATE INDEX officeCode ON employees(officeCode)

Removing Indexes

Besides creating index, you can also remove index by using the DROP INDEX statement. Interestingly, the DROP INDEX statement is also mapped to ALTER TABLE statement. The following is the syntax of removing the index:

DROP INDEX index_name ON table_name

For example, if you want to drop index officeCode of the employees table, which we have created above, you can execute following query:

DROP INDEX officeCode ON employees

MySQL - SELECT Statement

The MySQL SELECT statement allows you to retrieve zero or more rows from tables or views. The SELECT statement is the one of the most commonly used queries in MySQL.

The SELECT statement returns a result that is a combination of columns and rows, which is also known as a result set.

MySQL SELECT syntax

```
SELECT column_1,column_2...
FROM table_1
[INNER | LEFT |RIGHT] JOIN table_2 ON conditions
WHERE conditions
GROUP BY group
HAVING group_conditions
ORDER BY column_1 [ASC | DESC]
LIMIT offset, row_count
```

The SELECT statement is composed of several clauses:

- SELECT chooses which columns of the table you want to get the data.
- FROM specifies the table from which you get the data.
- JOIN gets data from multiple table based on certain join conditions.
- WHERE filters rows to select.
- GROUP BY group rows to apply aggregate functions on each group.
- HAVING filters group based on groups defined by GROUP BY clause.
- ORDER BY specifies the order of the returned result set.
- LIMIT constrains number of returned rows.

MySQL - Where clause

If you use the SELECT statement to query the data from a table without the WHERE clause, you will get allrows in the table, which sometimes brings more data than you need. The WHERE clause allows you to specify exact rows to select based on given conditions e.g., find all customers in the India

The following query selects all customers whose country is India from the customers table. We use the WHERE clause to filter the customers. In the WHERE clause, we compare the values of the countrycolumn with the India literal string.

```
SELECT customerName, city FROM customers
WHERE country = 'India';
```

You can form a simple condition like the query above, or a very complex one that combines multiple expressions with logical operators such as AND and OR. For example, to find all customers in the India and also in the Ahmedabad city, you use the following query:

```
SELECT customerName, city FROM customers
WHERE country = 'India' AND city    = 'Ahmedabad';
```

You can test the condition for not only equality but also inequality. For example, to find all customers whose credit limit is greater than Rs. 200.000, you use the following query:

```
SELECT customerName, creditlimit FROM customers
WHERE creditlimit > 200000;
```

There are several useful operators that you can use in the WHERE clause to form more practical queries such as:
- BETWEEN selects values within a range of values.
- LIKE matches value based on pattern matching.
- IN specifies if the value matches any value in a list.
- IS NULL checks if the value is NULL

The WHERE clause is used not only with the SELECT statement but also other SQL statements to filter rows such as DELETE and UPDATE.

MySQL - ORDER BY Clause

When you use the SELECT statement to query data from a table, the result set is not sorted in a specific order. To sort the result set, you use the ORDER BY clause. The ORDER BY clause allows you to:

- Sort a result set by a single column or multiple columns.
- Sort a result set by different columns in ascending or descending order.

The following illustrates the syntax of the ORDER BY clause:

```
SELECT col1, col2,...
FROM tbl
ORDER BY col1 [ASC|DESC], col2 [ASC|DESC],...
```

The ASC stands for ascending and the DESC stands for descending. By default, the ORDER BY clause sorts the result set in ascending order if you don't specify ASC or DESC explicitly

Let's practice with some examples of using the ORDER BY clause.

The following query selects contacts from the customers table and sorts the contacts by last name inascending order.

SELECT contactLastname, contactFirstname FROM customers
ORDER BY contactLastname;

If you want to sort the contact by last name in descending order, you specify the DESC after thecontactLastname column in the ORDER BY clause as the following query:

SELECT contactLastname, contactFirstname FROM customers
ORDER BY contactLastname DESC

If you want to sort the contacts by last name in descending order and first name in ascending order, you specify both DESC and ASC in the corresponding column as follows:

SELECT contactLastname, contactFirstname FROM customers
ORDER BY contactLastname DESC, contactFirstname ASC;

In the query above, the ORDER BY clause sorts the result set by last name in descending order first, and then sorts the sorted result set by first name in ascending order to produce the final result set.

MySQL ORDER BY sort by an expression example

The ORDER BY clause also allows you to sort the result set based on an expression. The following query selects the order line items from the orderdetails table. It calculates the subtotal for each line item and sorts the result set based on the order number and subtotal.

SELECT ordernumber, quantityOrdered * priceEach FROM orderdetails
ORDER BY ordernumber, quantityOrdered * priceEach

To make the result more readable, you can use a column alias, and sort the result based on the column alias.

SELECT orderNumber, quantityOrdered * priceEach AS subTotal FROM orderdetails
ORDER BY orderNumber, subTotal;

In the query above, we used subtotal as the column alias for the quantityOrdered * priceEach expression and sorted the result set based on the subtotal alias.

If you use a function that returns a value whose data type is different from the column's and sort the result based on the alias, the ORDER BY clause will sort the result set based on the return type of the function, which may not work as expected.

For example, if you use the DATE_FORMAT function to format the date values and sort the result set based on the strings returned by the DATE_FORMAT function, the order is not always correct. For more information, check it out the example in the DATE_FORMAT function tutorial.

MySQL ORDER BY with customer sort order

The ORDER BY clause enables you to define your own custom sort order for the values in a column using the FIELD() function. For example, if you want to sort the orders based on the following status by the following order:

1. In Process
2. On Hold
3. Cancelled
4. Resolved
5. Disputed
6. Shipped

You can use the FIELD() function to map those values to a list of numeric values and use the numbers for sorting; See the following query:

```
SELECT orderNumber, status FROM orders
ORDER BY FIELD(status, 'In Process', 'On Hold', 'Cancelled',
                'Resolved','Disputed', 'Shipped');
```

MySQL - Distinct

When querying data from a table, you may get duplicate rows. In order to remove the duplicate rows, you use the DISTINCT operator in the SELECT statement. The syntax of using the DISTINCT operator is as follows:

```
SELECT DISTINCT columns
FROM table_name
WHERE where_conditions
```

Let's take a look a simple example of using the DISTINCT operator to select the distinct last names of employees from the employees table.

First, we query the last names of employees from the employees table using the SELECT statementas follows:

```
SELECT lastname
FROM employees
ORDER BY lastname
```

Some employees has the same last name Bondur, Firrelli, etc. To remove the duplicate last names, you use the DISTINCT operator in the SELECT clause as follows:

SELECT DISTINCT lastname
FROM employees
ORDER BY lastname

The duplicate last names are eliminated in the result set when we use the DISTINCT operator.

MySQL DISTINCT and NULL values

If a column has NULL values and you use the DISTINCT operator for that column, MySQL will keep one NULL value and eliminate the other because the DISTINCT operator treats all NULL values as the same value.

For example, in the customers table, we have many rows with state column has NULL values. When we use the DISTINCT operator to query states of customers, we will see distinct states plus a NULL value as the following query:

SELECT DISTINCT state
FROM customers

MySQL DISTINCT with multiple columns

You can use the DISTINCT operator with more than one column. The combination of all columns will be used to define the uniqueness of the row in the result set.

For example, to get the unique combination of city and state from the customers table, you use the following query:

SELECT DISTINCT state, city
FROM customers
WHERE state IS NOT NULL
ORDER BY state, city

Without the DISTINCT operator, you will get duplicate combination state and city as follows:

SELECT state, city
FROM customers
WHERE state IS NOT NULL
ORDER BY state, city

DISTINCT vs. GROUP BY Clause

If you use the GROUP BY clause in the SELECT statement without using aggregate functions, theGROUP BY clause will behave like the DISTINCT operator. The following queries produce the same result set:

```
SELECT DISTINCT state
FROM customers;

SELECT state
FROM customers
GROUP BY state;
```

The difference between DISTINCT operator and GROUP BY clause is that the GROUP BY clause sorts the result set whereas the DISTINCT operator does not.

MySQL DISTINCT and COUNT aggregate function

The DISTINCT operator is used with the COUNT function to count unique records in a table. In this case, it ignores the NULL values. For example, to count the unique states of customers in the U.S., you use the following query:

```
SELECT COUNT(DISTINCT state)
FROM customers
WHERE country = 'USA';
```