My SQL

Created	@May 23, 2022 10:56 AM
Class	
Type	
Materials	
Reviewed	☐

For Creating a Database

```
CREATE DATABASE Movie;
```

If we have a doubt that whether the database was already present or a database with a same name exists we can use the following command.

```
CREATE DATABASE IF NOT EXISTS Student;
```

So both of the commands automatically explains what is the intention i.e. these are going to create a database.

For listing down all the databases

```
SHOW DATABASES;
```

To delete a database

```
DROP DATABASE Student;
```

Whatever database we will pass in the above command that will be removed.

Data types of MySql

- Numeric - INT, BIGINT, TINYINT, DECIMAL etc
- Data and time - DATE, TIME, YEAR, TIMESTAMP
- String - CHAR, VARCHAR, SET, BLOB, ENUM
- JSON -

Creating a table in a databse

```
USE Movie; // This command will select the database on which further ops are going to be p
erformed
```

```
CREATE TABLE Actors (FIRSTNAME VARCHAR(20), LASTNAME VARCHAR(20), DoB DATE, GENDER ENUM('M
ale', 'Female', 'Transgender'), MARITIALSTATUS ENUM ('DIVORCES', 'MARRIED', 'SINGLE'), NET
WORTH DECIMAL);
```

To create a table, we use `CREATE TABLE <name>` and then inside a pair of parenthesis we put comma separated column names with their data types.

Listing all the tables in a database

```
SHOW TABLES;
```

Detailed properties of the table

```
DESC Actors;
DESCRIBE Actors;
```

```
+---------------+-------------------------------------+------+-----+---------+-------+
| Field         | Type                                | Null | Key | Default | Extra |
+---------------+-------------------------------------+------+-----+---------+-------+
| FIRSTNAME     | varchar(20)                         | YES  |     | NULL    |       |
| LASTNAME      | varchar(20)                         | YES  |     | NULL    |       |
| DoB           | date                                | YES  |     | NULL    |       |
| GENDER        | enum('Male','Female','Transgender') | YES  |     | NULL    |       |
| MARITIALSTATUS | enum('DIVORCES','MARRIED','SINGLE') | YES  |     | NULL    |       |
| NETWORTH      | decimal(10,0)                       | YES  |     | NULL    |       |
+---------------+-------------------------------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

Creating table with default values, null checks and primary key

```
CREATE TABLE IF NOT EXISTS Actors (ID INT AUTO_INCREMENT, FIRSTNAME VARCHAR(20) NOT NULL,
 LASTNAME VARCHAR(20), DoB DATE, GENDER ENUM('Male', 'Female', 'Transgender') NOT NULL, MA
RITIALSTATUS ENUM ('DIVORCES', 'MARRIED', 'SINGLE', 'UNKNOWN') DEFAULT 'UNKNOWN', NETWORTH
DECIMAL, PRIMARY KEY (ID));
```

If we want to have a default value for a column, we just need to add `default value` as a clause after the column data type. If we want to add extra properties like `NOT NULL` or `AUTO_INCREMENT` then we can just mention them after the data type of the column.

If we want to make any particular column as Primary key then we just add `PRIMARY KEY (column_name)` to add it.

```
mysql> desc Actors;
+---------------+-----------------------------------------------+------+-----+---------+----------------+
| Field         | Type                                          | Null | Key | Default | Extra          |
+---------------+-----------------------------------------------+------+-----+---------+----------------+
| ID            | int                                           | NO   | PRI | NULL    | auto_increment |
| FIRSTNAME     | varchar(20)                                   | NO   |     | NULL    |                |
| LASTNAME      | varchar(20)                                   | YES  |     | NULL    |                |
| DoB           | date                                          | YES  |     | NULL    |                |
| GENDER        | enum('Male','Female','Transgender')           | NO   |     | NULL    |                |
| MARITIALSTATUS | enum('DIVORCES','MARRIED','SINGLE','UNKNOWN') | YES  |     | UNKNOWN |                |
| NETWORTH      | decimal(10,0)                                 | YES  |     | NULL    |                |
+---------------+-----------------------------------------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

Inserting single row in the table

For inserting a single we can use the INSERT INTO command.

```
INSERT INTO Actors (FIRSTNAME, LASTNAME, DoB, GENDER, MaritialStatus, NETWORTH) VALUES ("J
ohnny", "Depp", "1960-10-08", "Male", "Single", 5000);
```

Here after INSERT INTO we mention the table name and then inside a pair of parenthesis we mention the order of the columns in which we will give the data and then we put the keyword `Values` and then again inside a pair of parenthesis we add the actual data in the same column order which we defined before. If you want to put multiple records in single query then you can these records in separate pair of parenthesis comma separated.

`INSERT INTO <TABLE NAME> (Col1, Col2, Col3 ..) VALUES (Value1, value2, value3 …);`

```
INSERT INTO Actors (FIRSTNAME, LASTNAME, DoB, GENDER, MaritialStatus, NETWORTH) VALUES ("A
ngelina", "Jolie", "1969-11-02", "Female", "Married", 5000) , ("Chris", "Hemsworth", "1975
-08-10", "Male", "Married", 7000) ;
```

To insert as key value pair in arbitrary order, we can use `SET`

```
INSERT INTO Actors SET DoB="1970-12-03", FIRSTNAME="Robert", LASTNAME="Downey Jr.", NETWOR
TH=10000, GENDER="MALE";
```

Display data from table

```
SELECT * FROM Actors;
```

The select actually displays data for the table you mention. `*` represents that we want to get all the columns printed for the table.

`SELECT * FROM <TABLE NAME>;`

Displaying particular Columns

```
SELECT FIRSTNAME, LASTNAME, DoB FROM Actors;
```

Instead of * we can put specific column names comma separated to get the desired columns filtered

Displaying particular Rows

So we can filter out rows based on some condition using `WHERE` clause. After the table we put the where clause and write our conditions.

`SELECT * FROM <TABLE NAME> WHERE <CONDITIONS>;`

```
SELECT FIRSTNAME, LASTNAME, DoB, NETWORTH FROM Actors WHERE GENDER = "Male";
```

```
SELECT * FROM Actors WHERE GENDER = "FEMALE" AND  NETWORTH > 5000;
```

Substring matching queries

If we want to check all the actors whose firstname start with `Ch` then we can do

```
select * from Actors WHERE FIRSTNAME LIKE "Ch%";
```

Here Ch% represents that we want to have the Firstname starting with Ch and after Ch anything can be there. So the % represents that we can put any value after Ch

If we want to get all the actors whose firstname end with `r` then we can do

```
select * from Actors WHERE FIRSTNAME LIKE "%r";
```

Here %r represents that string can start with anything but it should end with a r.

If we want to end it with `rt` we can do

```
select * from Actors WHERE FIRSTNAME LIKE "%rt";
```

The above cases are for mainly prefix and suffix matching. But what if instead of just prefix or suffix we want to search for the string anywhere in the data.

```
select * from Actors WHERE FIRSTNAME LIKE "%r%";
```

So here %r% represents that string can start with anything end with anything only it should have r somewhere in the string.

Sorting the data

If we want to sort the data based on any particular column, we can do

```
SELECT * FROM Actors ORDER BY FIRSTNAME; // This will arrange data in INC order of firstname
```

Here we are using the ORDER BY clause, Order by clause takes column names and then rearrange the data for display based on the column name.

If we want to tackle the condition where let's say we have firstname same, then we should consider different column, we can use the following query

```
SELECT * FROM Actors ORDER BY FIRSTNAME, NETWORTH;
```

So for having multiple column for comparison of data, we can put column comma separated after order by. So if we have firstname same then we will try networth. And if we put any more columns, then a case where networth is also same can be handled too.

But If want to arrange the data in Decreasing order of firstname and increasing order of networth.

```
SELECT * FROM Actors ORDER BY FIRSTNAME DESC, NETWORTH ASC;
```

`DESC` stands for decreasing, so when we put DESC in front of any column it sorts them based on the decreasing order. Similarly `ASC` sorts in ascending order.

Mysql ignores case when comparing string in Orderby clause. Which means “chris” and “Chris” are treated equally. If we want to do ascii comparison for the data, we can use `BINARY` Keyword.

```
SELECT * FROM Actors ORDER BY BINARY FIRSTNAME DESC, NETWORTH ASC;
```

Limit and offset

If we dont want all the rows to get printed instead we want the first 3 rows we can do something like

```
SELECT * FROM Actors ORDER BY CAST(NETWORTH AS CHAR) LIMIT 3;
```

The LIMIT Clause helps you to put a number that denotes how many entry you want to fetch. By default the first entry to be fetched starts from the first row. But let’s say we want 3 rows not from the start but instead from the 2 row

```
SELECT * FROM Actors ORDER BY CAST(NETWORTH AS CHAR) LIMIT 3 OFFSET 1;
```

So if we put offset as `x` we get the data from `x+1` th row.

Deleting Data from Table

In order to delete data we can use `DELETE FROM <TABLE NAME> WHERE <CONDITION>;`

```
DELETE FROM Actors WHERE GENDER = "FEMALE";
```

If there are multiple entries satisfying the condition and we want to delete initial 2 of them then we can use

```
DELETE FROM Actors WHERE MARITIALSTATUS = "UNKNOWN" LIMIT 2;
```

If we want to delete all the entries we can do

```
DELETE FROM Actors;
```

A faster way to delete all the rows is `TRUNCATE`

```
TRUNCATE Actors;
```

This actually drops the whole table and recreates it, so it is a bit faster than manually deleting every row.

Updating data

We can use `UPDATE <TABLE NAME> SET <COLUMN NAME> = <VALUE> WHERE <CONDITION>`

```
UPDATE Actors SET MARITIALSTATUS = "Married", NETWORTH=11000 WHERE LASTNAME = "Evans";
```

If we want to update all the rows

```
UPDATE Actors SET MARITIALSTATUS = "Unknown";
```

If instead of all the rows we just want to update the first 2 based on the condition or without a condition also, we can LIMIT.

Indexes

Every time we do a `where` clause based query MySQL has to do a full scan of the tables to find the matching rows. A linear scan or a linear search is not very efficient. That's why we try to add indexes, adding an index to the table can significantly speed up search for the matching rows. An index's primary work is to provide an ordered representation of data. It is similar to book index. If we are looking for a particular word, we can scan the index and find the page with the corresponding word.

```
SHOW INDEX FROM Actors;
```

The cardinality shows the number of unique values for the primary key. It’s also described as an estimate of the number of unique values in the index.

Whenever we do indexing, we try to represent the data in ordered fashion using some data structure. B Tree and B+ Tree are few of them.

```
CREATE INDEX networth_index ON Actors (NETWORTH);
```

Alterations

If we want to some alterations to the table we can use `alter table` command.

```
ALTER TABLE Actors CHANGE FIRSTNAME First_Name VARCHAR(255);
```

for adding a new column

```
ALTER TABLE Actors ADD Middle_Name VARCHAR(20);
```

If we want to remove a column from the table

```
ALTER TABLE Actors DROP Middle_Name;
```

Let’s say, we dont want the new column to be added at last, we can add it in after any column using

```
ALTER TABLE Actors ADD Middle_Name VARCHAR(20) AFTER First_Name;
```

Alias

Using `AS` key word we can add alias

Distinct Clause

If we want to filter out all the distinct entries from the table we can use this clause

```
SELECT DISTINCT First_Name From Actors;
```

If we want multiple columns distinct value we can pass comma separated columns.

Aggregate Functions

In MySQL we can get a lot of aggregate functions that does computation on a bunch of rows together.

Few of those functions are `Count, MAX, MIN, SUM, AVG`

```
SELECT COUNT(*) FROM ACTORS;
SELECT SUM(NETWORTH) FROM ACTORS;
SELECT MIN(NETWORTH) FROM ACTORS;
SELECT MAX(NETWORTH) FROM ACTORS;
```

Group by

As the name suggests, it sorts out the rows together into group. The clause returns one row for each group. Data is organised using a comma separated list of columns as the grouping criteria.

Syntactically the group by clause must appear after FROM and WHERE clauses but before ORDER BY or LIMIT.

```
SELECT COUNT(*) AS COUNT FROM ACTORS GROUP BY FIRST_NAME ORDER BY COUNT DESC;
```

Having Clause

WHERE clause is used to filter rows whereas HAVING clause is used to filter groups. We can using having for filtering rows also but that inefficient. And Having clause should

be used to decide what rows from the group comes as answer.

Joins

The SQL `Join` joins two or more tables based on a common column and selects records that have matching values in these columns.

movieid	Name
1	abc
2	def

movieid	cinemaid
1	2
2	3

cinemaId	Cinama Name
1	a
2	b
3	c

There are different type of joins

- `Inner Join` The common column that we use for joining the tables, in inner join we will only consider those values which are common in both the tables.

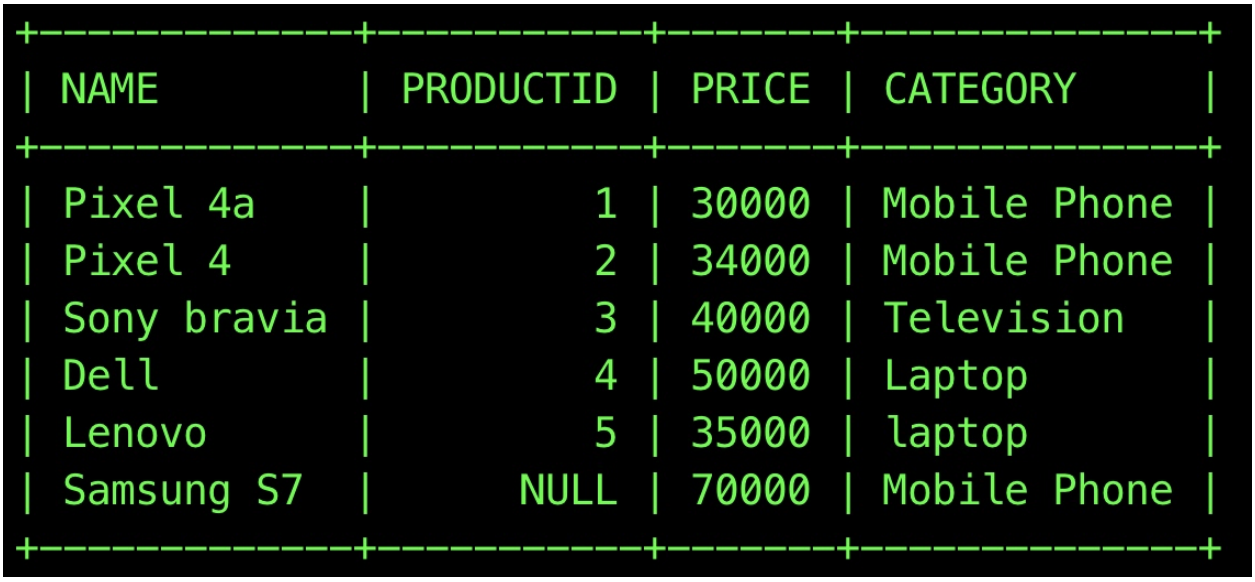
```
select quantity, email, userId , name from Orders INNER JOIN users ON Orders.userId = user
s.id INNER JOIN Products ON Orders.productId = Products.id;
```

- `Left Join` If we want all the data which is common in both the tables as well as the data which is present just in the first table then we use left join.

```
SELECT NAME, PRODUCTID, PRICE, CATEGORY FROM PRODUCTS INNER JOIN ORDERS ON PRODUCTS.ID = O
RDERS.PRODUCTID;
```

- `Right Join` In right join we get all the data which is common in both the joined tables and also the data which is just present in the second table but not first.

```
SELECT NAME, PRODUCTID, PRICE, CATEGORY FROM ORDERS RIGHT JOIN PRODUCTS ON PRODUCTS.ID = O
RDERS.PRODUCTID;
```



NAME	PRODUCTID	PRICE	CATEGORY
Pixel 4a	1	30000	Mobile Phone
Pixel 4	2	34000	Mobile Phone
Sony bravia	3	40000	Television
Dell	4	50000	Laptop
Lenovo	5	35000	laptop
Samsung S7	NULL	70000	Mobile Phone