UNIT III - SQL & MYSQL NOTES

MYSQL

It is freely available open source Relational Database Management System (RDBMS) that uses **Structured Query Language(SQL).** In MySQL database , information is stored in Tables. A single MySQL database can contain many tables at once and store thousands of individual records.

SQL (Structured Query Language)

SQL is a language that enables you to create and operate on relational databases, which are sets of related information stored in tables.

DIFFERENT DATA MODELS

A **data model refers to a set of concepts to describe the structure of a database**, and certain constraints (restrictions) that the database should obey. The four data model that are used for database management are :

1. **Relational data model** : In this data model, the data is organized into tables (i.e. rows and columns). These tables are called relations.

2. **Hierarchical data model 3. Network data model 4. Object Oriented data model**

RELATIONAL MODEL TERMINOLOGY

1. **Relation :** A table storing logically related data is called a Relation.

2. **Tuple :** A **row of a relation** is generally referred to as a tuple.

3. **Attribute :** A **column** of a relation is generally referred to as an attribute.

4. **Degree :** This refers to the **number of attributes** in a relation.

5. **Cardinality :** This refers to the **number of tuples** in a relation.

6. **Primary Key :** This refers to a set of one or more attributes that can uniquely identify tuples within the relation.

7. **Candidate Key :** All attribute combinations inside a relation that can serve as primary key are candidate keys as these are candidates for primary key position.

8. **Alternate Key :** A candidate key that is not primary key, is called an alternate key.

9. **Foreign Key :** A non-key attribute, whose values are derived from the primary key of some other table, is known as foreign key in its current table.

REFERENTIAL INTEGRITY

- A referential integrity is a system of rules that a DBMS uses to ensure that relationships between records in related tables are valid, and that users don't accidentally delete or change related data. This integrity is ensured by foreign key.

CLASSIFICATION OF SQL STATEMENTS

SQL commands can be mainly divided into following categories:

1. Data Definition Language(DDL) Commands

Commands that allow you to perform task, related to data definition e.g;

- Creating, altering and dropping.
- Granting and revoking privileges and roles.
- Maintenance commands.

2. **Data Manipulation Language(DML) Commands**

   Commands that allow you to perform data manipulation e.g., retrieval, insertion, deletion and modification of data stored in a database.

3. **Transaction Control Language(TCL) Commands**

   Commands that allow you to manage and control the transactions e.g.,

   - Making changes to database, permanent
   - Undoing changes to database, permanent
   - Creating savepoints
   - Setting properties for current transactions.

MySQL ELEMENTS

1. Literals 2. Datatypes 3. Nulls 4. Comments

LITERALS

It refer to a fixed data value. This fixed data value may be of character type or numeric type. For example, 'replay' , 'Raj', '8' , '306' are all character literals.

Numbers not enclosed in quotation marks are numeric literals. E.g. 22 , 18 , 1997 are all numeric literals.

Numeric literals can either be integer literals i.e., without any decimal or be real literals i.e. with a decimal point e.g. 17 is an integer literal but 17.0 and 17.5 are real literals.

DATA TYPES

Data types are means to identify the type of data and associated operations for handling it. MySQL data types are divided into three categories:

- Numeric
- Date and time
- String types

Numeric Data Type

1. int – used for number without decimal.
2. Decimal(m,d) – used for floating/real numbers. m denotes the total length of number and d is number of decimal digits.

Date and Time Data Type

1. date – used to store date in YYYY-MM-DD format.
2. time – used to store time in HH:MM:SS format.

String Data Types

1. char(m) – used to store a fixed length string. **m** denotes max. number of characters.
2. varchar(m) – used to store a variable length string. **m** denotes max. no. of characters.

DIFFERENCE BETWEEN CHAR AND VARCHAR DATA TYPE

S.NO.	Char Datatype	Varchar Datatype
1.	It specifies a **fixed length** character String.	It specifies a **variable length** character string.
2.	When a column is given datatype as CHAR(**n**), then MySQL ensures that all values stored in that column have this length i.e. **n** bytes. If a value is shorter than this length **n** then blanks are added, but the size of value remains **n** bytes.	When a column is given datatype as VARCHAR(**n**), then the maximum size a value in this column can have is **n** bytes. Each value that is stored in this column store exactly as you specify it i.e. no blanks are added if the length is shorter than maximum length **n**.

NULL VALUE

If a column in a row has no value, then column is said to be **null** , or to contain a null. **You should use a null value** when the actual value is not known or when a value would not be meaningful.

DATABASE COMMNADS

1. VIEW EXISTING DATABASE

To view existing database names, the command is : **SHOW DATABASES ;**

2. CREATING DATABASE IN MYSQL

For creating the database in MySQL, we write the following command : **CREATE DATABASE** <databasename> ;

e.g. In order to create a database Student, command is :

CREATE DATABASE Student ;

3. ACCESSING DATABASE

For accessing already existing database , we write :

USE <databasename> ;

e.g. to access a database named Student , we write command as :

USE Student ;

4. DELETING DATABASE

For deleting any existing database , the command is :

DROP DATABASE <databasename> ;

e.g. to delete a database , say student, we write command as ; **DROP DATABASE** Student ;

5. VIEWING TABLE IN DATABASE

In order to view tables present in currently accessed database , command is : **SHOW TABLES ;**

CREATING TABLES IN MYSQL

- Tables are created with the CREATE TABLE command. When a table is created, its columns are named, data types and sizes are supplied for each column.

Syntax of CREATE TABLE command is : CREATE TABLE <table-name>

(<column name> <data type> ,
<column name> <data type> ,
…… **) ;**

E.g. in order to create table EMPLOYEE given below :

ECODE	ENAME	GENDER	GRADE	GROSS

We write the following command :

```
CREATE TABLE employee
(    ECODE integer ,
     ENAME varchar(20) ,
     GENDER char(1) ,
     GRADE char(2) ,
     GROSS integer    ) ;
```

INSERTING DATA INTO TABLE

- The rows are added to relations(table) using INSERT command of SQL. Syntax of
  INSERT is : INSERT INTO <tablename> [<column list>]
  VALUE (<value1> , <value2> , …..) ;

e.g. to enter a row into EMPLOYEE table (created above), we write command as :

```
INSERT INTO employee
VALUES(1001 , 'Ravi' , 'M' , 'E4' , 50000);
```

OR

```
INSERT INTO employee (ECODE , ENAME , GENDER , GRADE , GROSS)
VALUES(1001 , 'Ravi' , 'M' , 'E4' , 50000);
```

ECODE	ENAME	GENDER	GRADE	GROSS
1001	Ravi	M	E4	50000

In order to insert another row in EMPLOYEE table , we write again INSERT command :

```
INSERT INTO employee
VALUES(1002 , 'Akash' , 'M' , 'A1' , 35000);
```

ECODE	ENAME	GENDER	GRADE	GROSS
1001	Ravi	M	E4	50000
1002	Akash	M	A1	35000

INSERTING NULL VALUES

- To insert value NULL in a specific column, we can type NULL without quotes and NULL will be inserted in that column. E.g. in order to insert NULL value in ENAME column of above table, we write INSERT command as :

```
INSERT INTO EMPLOYEE
VALUES (1004 , NULL , 'M' , 'B2' , 38965 ) ;
```

ECODE	ENAME	GENDER	GRADE	GROSS
1001	Ravi	M	E4	50000
1002	Akash	M	A1	35000
1004	NULL	M	B2	38965

SIMPLE QUERY USING SELECT COMMAND

- The SELECT command is used to pull information from a table. Syntax of SELECT command is : SELECT <column name>,<column name>
  FROM <tablename>
  WHERE <condition name> ;

SELECTING ALL DATA

- In order to retrieve everything (all columns) from a table, SELECT command is used as : **SELECT * FROM** <tablename> ;

e.g.

In order to retrieve everything from **Employee** table, we write SELECT command as :

EMPLOYEE

ECODE	ENAME	GENDER	GRADE	GROSS
1001	Ravi	M	E4	50000
1002	Akash	M	A1	35000
1004	NULL	M	B2	38965

SELECT * FROM Employee ;

SELECTING PARTICULAR COLUMNS

EMPLOYEE

ECODE	ENAME	GENDER	GRADE	GROSS
1001	Ravi	M	E4	50000
1002	Akash	M	A1	35000
1004	Neela	F	B2	38965
1005	Sunny	M	A2	30000
1006	Ruby	F	A1	45000
1009	Neema	F	A2	52000

- A particular column from a table can be selected by specifying column-names with SELECT command. E.g. in above table, if we want to select ECODE and ENAME column, then command is :

SELECT ECODE , ENAME
FROM EMPLOYEE ;

E.g.2 in order to select only ENAME, GRADE and GROSS column, the command is :

SELECT ENAME , GRADE ,
GROSS **FROM** EMPLOYEE ;

SELECTING PARTICULAR ROWS

We can select particular rows from a table by specifying a condition through **WHERE clause** along with SELECT statement. **E.g.** In employee table if we want to select rows where Gender is female, then command is :

SELECT * FROM **EMPLOYEE**
WHERE **GENDER = 'F' ;**

E.g.2. in order to select rows where salary is greater than 48000, then command is :

SELECT * FROM **EMPLOYEE**
WHERE **GROSS > 48000 ;**

ELIMINATING REDUNDANT DATA

The **DISTINCT** keyword eliminates duplicate rows from the results of a SELECT statement. For example ,

SELECT GENDER **FROM** EMPLOYEE ;

GENDER
M
M
F
M
F
F

SELECT DISTINCT(GENDER) **FROM** EMPLOYEE ;

DISTINCT(GENDER)
M
F

VIEWING STRUCTURE OF A TABLE

- If we want to know the structure of a table, we can use DESCRIBE or DESC command, as per following syntax :

DESCRIBE | **DESC** <tablename> ;

e.g. to view the structure of table **EMPLOYEE**, command is : **DESCRIBE** EMPLOYEE ; **OR DESC** EMPLOYEE ;

USING COLUMN ALIASES

- The columns that we select in a query can be given a different name, i.e. column alias name for output purpose.

Syntax :

SELECT <columnname> **AS** column alias , <columnname> **AS** column alias …..
FROM <tablename> ;

e.g. In output, suppose we want to display ECODE column as EMPLOYEE_CODE in output , then command is :

SELECT **ECODE AS "EMPLOYEE_CODE"**
FROM **EMPLOYEE ;**

CONDITION BASED ON A RANGE

- The **BETWEEN** operator defines a range of values that the column values must fall in to make the condition true. The range include both lower value and upper value.

e.g. to display ECODE, ENAME and GRADE of those employees whose salary is between 40000 and 50000, command is:

SELECT ECODE , ENAME ,GRADE
FROM **EMPLOYEE**
WHERE GROSS **BETWEEN** 40000 AND 50000 ;

Output will be :

ECODE	ENAME	GRADE
1001	Ravi	E4
1006	Ruby	A1

CONDITION BASED ON A LIST

- To specify a list of values, IN operator is used. The IN operator selects value that match any value in a given list of values. E.g.

SELECT * FROM EMPLOYEE
WHERE GRADE **IN** ('A1' , 'A2');

Output will be :

ECODE	**ENAME**	**GENDER**	**GRADE**	**GROSS**
1002	Akash	M	A1	35000
1006	Ruby	F	A1	45000
1005	Sunny	M	A2	30000
1009	Neema	F	A2	52000

- The **NOT IN** operator finds rows that do not match in the list. E.g.

SELECT * FROM EMPLOYEE
WHERE GRADE **NOT IN** ('A1' , 'A2');

Output will be :

ECODE	**ENAME**	**GENDER**	**GRADE**	**GROSS**
1001	Ravi	M	E4	50000
1004	Neela	F	B2	38965

CONDITION BASED ON PATTERN MATCHES

- LIKE operator is used for pattern matching in SQL. Patterns are described using two special wildcard characters:

1. percent(%) – The % character matches any substring.
2. underscore(_) – The _ character matches any character.

e.g. to display names of employee whose name starts with R in EMPLOYEE table, the command is :

SELECT ENAME
FROM **EMPLOYEE**
WHERE ENAME **LIKE 'R%' ;**

Output will be :

ENAME
Ravi
Ruby

e.g. to display details of employee whose second character in name is 'e'.

SELECT *
FROM **EMPLOYEE**
WHERE **ENAME LIKE '_e%' ;**

Output will be :

ECODE	ENAME	GENDER	GRADE	GROSS
1004	Neela	F	B2	38965
1009	Neema	F	A2	52000

e.g. to display details of employee whose name ends with 'y'.

SELECT *
FROM **EMPLOYEE**
WHERE **ENAME LIKE '%y' ;**

Output will be :

ECODE	ENAME	GENDER	GRADE	GROSS
1005	Sunny	M	A2	30000
1006	Ruby	F	A1	45000

SEARCHING FOR NULL

- The NULL value in a column can be searched for in a table using IS NULL in the WHERE clause. E.g. to list employee details whose salary contain NULL, we use the command :

SELECT ***
FROM **EMPLOYEE**
WHERE GROSS **IS NULL ;**

e.g.

STUDENT

Roll_No	Name	Marks
1	ARUN	NULL
2	RAVI	56
4	SANJAY	NULL

to display the names of those students whose marks is NULL, we use the command :

SELECT **Name**
FROM **EMPLOYEE**
WHERE Marks **IS NULL ;**

Output will be :

Name
ARUN
SANJAY

SORTING RESULTS

Whenever the SELECT query is executed , the resulting rows appear in a predecided order.The **ORDER BY clause** allow sorting of query result. The sorting can be done either in ascending or descending order, the default is ascending.

The **ORDER BY clause is used as :**

```
SELECT <column name> , <column name>….
FROM <tablename>
WHERE <condition>
ORDER BY <column name> ;
```

e.g. to display the details of employees in EMPLOYEE table in alphabetical order, we use command :

```
SELECT *
FROM EMPLOYEE
ORDER BY ENAME ;
```

Output will be :

ECODE	ENAME	GENDER	GRADE	GROSS
1002	Akash	M	A1	35000
1004	Neela	F	B2	38965
1009	Neema	F	A2	52000
1001	Ravi	M	E4	50000
1006	Ruby	F	A1	45000
1005	Sunny	M	A2	30000

e.g. display list of employee in descending alphabetical order whose salary is greater than 40000.

```
SELECT ENAME
FROM EMPLOYEE
WHERE GROSS > 40000
ORDER BY ENAME desc ;
```

Output will be :

ENAME
Ravi
Ruby
Neema

MODIFYING DATA IN TABLES

you can modify data in tables using UPDATE command of SQL. The UPDATE command specifies the rows to be changed using the WHERE clause, and the new data using the SET keyword. Syntax of update command is :

```
UPDATE <tablename>
SET <columnname>=value , <columnname>=value
WHERE <condition> ;
```

e.g. to change the salary of employee of those in EMPLOYEE table having employee code 1009 to 55000.

```
UPDATE EMPLOYEE
SET GROSS = 55000
WHERE ECODE = 1009 ;
```

UPDATING MORE THAN ONE COLUMNS

e.g. to update the salary to 58000 and grade to B2 for those employee whose employee code is 1001.

```
UPDATE EMPLOYEE
SET GROSS = 58000, GRADE='B2'
WHERE ECODE = 1009 ;
```

OTHER EXAMPLES

e.g.1. Increase the salary of each employee by 1000 in the EMPLOYEE table.

UPDATE EMPLOYEE
SET GROSS = GROSS +100 ;

e.g.2. Double the salary of employees having grade as 'A1' or 'A2' .

UPDATE EMPLOYEE
SET GROSS = GROSS * 2 ;
WHERE GRADE='A1' **OR** GRADE='A2' ;

e.g.3. Change the grade to 'A2' for those employees whose employee code is 1004 and name is Neela.

UPDATE EMPLOYEE
SET GRADE='A2'
WHERE ECODE=1004 **AND** GRADE='NEELA' ;

DELETING DATA FROM TABLES

To delete some data from tables, DELETE command is used. **The DELETE command removes rows from a table.** The syntax of DELETE command is :

DELETE FROM <tablename>
WHERE <condition> ;

For example, to remove the details of those employee from EMPLOYEE table whose grade is A1.

DELETE FROM EMPLOYEE
WHERE GRADE ='A1' ;

TO DELETE ALL THE CONTENTS FROM A TABLE

DELETE FROM EMPLOYEE ;

So if we do not specify any condition with WHERE clause, then all the rows of the table will be deleted. Thus above line will delete all rows from employee table.

DROPPING TABLES

The DROP TABLE command lets you drop a table from the database. The **syntax of DROP TABLE** command is :

DROP TABLE <tablename> ;

e.g. to drop a table employee, we need to write :

DROP TABLE employee ;

Once this command is given, the table name is no longer recognized and no more commands can be given on that table. After this command is executed, all the data in the table along with table structure will be deleted.

S.NO.	DELETE COMMAND	DROP TABLE COMMAND
1	It is a DML command.	It is a DDL Command.
2	This command is used to delete only rows of data from a table	This command is used to delete all the data of the table along with the structure of the table. The table is no longer recognized when this command gets executed.
3	Syntax of DELETE command is: **DELETE FROM** <tablename> **WHERE** <condition> ;	Syntax of DROP command is : **DROP TABLE** <tablename> ;

ALTER TABLE COMMAND

The ALTER TABLE command is used to change definitions of existing tables.(adding columns,deleting columns etc.). The ALTER TABLE command is used for :

1. adding columns to a table

2. Modifying column-definitions of a table.
3. Deleting columns of a table.
4. Adding constraints to table.
5. Enabling/Disabling constraints.

ADDING COLUMNS TO TABLE

To add a column to a table, ALTER TABLE command can be used as per following syntax:

ALTER TABLE <tablename>
ADD <Column name> <datatype> <constraint> ;

e.g. to add a new column ADDRESS to the EMPLOYEE table, we can write command as :

ALTER TABLE EMPLOYEE
ADD ADDRESS VARCHAR(50);

A new column by the name ADDRESS will be added to the table, where each row will contain NULL value for the new column.

ECODE	ENAME	GENDER	GRADE	GROSS	ADDRESS
1001	Ravi	M	E4	50000	NULL
1002	Akash	M	A1	35000	NULL
1004	Neela	F	B2	38965	NULL
1005	Sunny	M	A2	30000	NULL
1006	Ruby	F	A1	45000	NULL
1009	Neema	F	A2	52000	NULL

However **if you specify NOT NULL constraint while adding a new column**, MySQL adds the new column with the default value of that datatype e.g. for INT type it will add 0 , for CHAR types, it will add a space, and so on.

e.g. Given a table namely Testt with the following data in it.

Col1	Col2
1	A
2	G

Now following commands are given for the table. Predict the table contents after each of the following statements:

(i) ALTER TABLE testt ADD col3 INT ;
(ii) ALTER TABLE testt ADD col4 INT NOT NULL ;
(iii) ALTER TABLE testt ADD col5 CHAR(3) NOT NULL ;
(iv) ALTER TABLE testt ADD col6 VARCHAR(3);

MODIFYING COLUMNS

Column name and data type of column can be changed as per following syntax :

ALTER TABLE <table name>
CHANGE <old column name> <new column name> <new datatype>;

If **Only data type of column need to be changed**, then

ALTER TABLE <table name>
MODIFY <column name> <new datatype>;

e.g.1. In table EMPLOYEE, change the column GROSS to SALARY.

ALTER TABLE EMPLOYEE
CHANGE GROSS SALARY INTEGER;

e.g.2. In table EMPLOYEE , change the column ENAME to EM_NAME and data type from VARCHAR(20) to VARCHAR(30).

ALTER TABLE EMPLOYEE
CHANGE ENAME EM_NAME VARCHAR(30);

e.g.3. In table EMPLOYEE , change the datatype of GRADE column from CHAR(2) to VARCHAR(2).

ALTER TABLE EMPLOYEE
MODIFY GRADE VARCHAR(2);

DELETING COLUMNS

To delete a column from a table, the ALTER TABLE command takes the following form :

ALTER TABLE <table name>
DROP <column name>;

e.g. to delete column GRADE from table EMPLOYEE, we will write :

ALTER TABLE EMPLOYEE
DROP GRADE ;

ADDING/REMOVING CONSTRAINTS TO A TABLE

ALTER TABLE statement can be used to add constraints to your existing table by using it in following manner:

- **TO ADD PRIMARY KEY CONSTRAINT**

  ALTER TABLE <table name>
  ADD PRIMARY KEY (Column name);

e.g. to add PRIMARY KEY constraint on column ECODE of table EMPLOYEE , the command is :

ALTER TABLE EMPLOYEE
ADD PRIMARY KEY (ECODE) ;

- **TO ADD FOREIGN KEY CONSTRAINT**

  ALTER TABLE <table name>
  ADD FOREIGN KEY (Column name) REFERENCES Parent Table (Primary key of Parent Table);

REMOVING CONSTRAINTS

- To remove primary key constraint from a table, we use ALTER TABLE command as : **ALTER TABLE** <table name>
  DROP PRIMARY KEY ;
- To remove foreign key constraint from a table, we use ALTER TABLE command as : **ALTER TABLE** <table name>
  DROP FOREIGN KEY ;

ENABLING/DISABLING CONSTRAINTS

Only foreign key can be disabled/enabled in MySQL.

To disable foreign keys : **SET** FOREIGN_KEY_CHECKS = 0 ;
To enable foreign keys : **SET** FOREIGN_KEY_CHECKS = 1 ;

INTEGRITY CONSTRAINTS/CONSTRAINTS

- A constraint is a condition or check applicable on a field(column) or set of fields(columns).
- Common types of constraints include :

S.No.	Constraints	Description
1	NOT NULL	Ensures that a column cannot have NULL value
2	DEFAULT	Provides a default value for a column when none is specified
3	UNIQUE	Ensures that all values in a column are different
4	CHECK	Makes sure that all values in a column satisfy certain criteria
5	PRIMARY KEY	Used to uniquely identify a row in the table
6	FOREIGN KEY	Used to ensure referential integrity of the data

NOT NULL CONSTRAINT

By default, a column can hold NULL. It you not want to allow NULL value in a column, then NOT NULL constraint must be applied on that column. E.g.

```
CREATE TABLE Customer
(       SID integer NOT NULL ,
        Last_Name varchar(30)   NOT NULL ,
        First_Name varchar(30)      ) ;
```

Columns **SID** and **Last_Name** cannot include NULL, while **First_Name** can include NULL.

An attempt to execute the following SQL statement,

```
INSERT INTO Customer
VALUES (NULL , 'Kumar' , 'Ajay');
```

will result in an error because this will lead to column SID being NULL, which violates the NOT NULL constraint on that column.

DEFAULT CONSTARINT

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value. **E.g.**

```
CREATE TABLE Student
( Student_ID integer ,
   Name varchar(30) ,
   Score integer DEFAULT 80);
```

When following SQL statement is executed on table created above:

```
INSERT INTO Student
VALUES (10 , 'Ravi' );
```

no value has been provided for score field.

Then table **Student** looks like the following:

Student_ID	Name	Score
10	Ravi	**80**

score field has got the default value

UNIQUE CONSTRAINT

- The UNIQUE constraint ensures that all values in a column are distinct. In other words, no two rows can hold the same value for a column with UNIQUE constraint.

e.g.

```
CREATE TABLE Customer
(       SID integer Unique ,
        Last_Name varchar(30) ,
        First_Name varchar(30) ) ;
```

Column SID has a unique constraint, and hence cannot include duplicate values. So, if the table already contains the following rows :

SID	Last_Name	First_Name
1	Kumar	Ravi
2	Sharma	Ajay
3	Devi	Raj

The executing the following SQL statement,

```
INSERT INTO Customer
VALUES ('3' , 'Cyrus' , 'Grace') ;
```

will result in an error because the value 3 already exist in the SID column, thus trying to insert another row with that value violates the UNIQUE constraint.

CHECK CONSTRAINT

- The CHECK constraint ensures that all values in a column satisfy certain conditions. Once defined, the table will only insert a new row or update an existing row if the new value satisfies the CHECK constraint.

e.g.

```
CREATE TABLE Customer
(       SID integer CHECK (SID > 0),
        Last_Name varchar(30) ,
        First_Name varchar(30) ) ;
```

So, attempting to execute the following statement :

```
INSERT INTO Customer
VALUES (-2 , 'Kapoor' , 'Raj');
```

will result in an error because the values for SID must be greater than 0.

PRIMARY KEY CONSTRAINT

- A primary key is used to identify each row in a table. A primary key can consist of one or more fields(column) on a table. When multiple fields are used as a primary key, they are called a **composite key.**

- You can define a primary key in CREATE TABLE command through keywords PRIMARY KEY. e.g.

```
CREATE TABLE Customer
(       SID integer NOT NULL PRIMARY KEY,
        Last_Name varchar(30) ,
        First_Name varchar(30) ) ;
```

Or

```
CREATE TABLE Customer
(       SID integer,
        Last_Name varchar(30) ,
        First_Name varchar(30),
        PRIMARY KEY (SID) ) ;
```

- The latter way is useful if you want to specify a composite primary key, **e.g.**

```
CREATE TABLE Customer
(       Branch integer NOT NULL,
        SID integer NOT NULL ,
        Last_Name varchar(30) ,
        First_Name varchar(30),
        PRIMARY KEY (Branch , SID) ) ;
```

FOREIGN KEY CONSTRAINT

- Foreign key is a non key column of a table (**child table**) that draws its values from **primary key** of another table(**parent table).**
- The table in which a foreign key is defined is called a **referencing table or child table.** A table to which a foreign key points is called **referenced table or parent table.**

e.g.

TABLE: STUDENT ← **Parent Table**

ROLL_NO ← **Primary key**	NAME	CLASS
1	ABC	XI
2	DEF	XII
3	XYZ	XI

Child Table

TABLE: SCORE

ROLL_NO	MARKS
1	55
2	83
3	90

Here column Roll_No is a foreign key in table SCORE(Child Table) and it is drawing its values from Primary key (ROLL_NO) of STUDENT table.(Parent Key).

```
CREATE TABLE STUDENT
(   ROLL_NO integer NOT NULL PRIMARY KEY ,
    NAME VARCHAR(30) ,
    CLASS VARCHAR(3) );
```

```
CREATE TABLE SCORE
  ( ROLL_NO integer ,
    MARKS integer ,
    FOREIGN KEY(ROLL_NO) REFERNCES STUDENT(ROLL_NO) ) ;
```

**** Foreign key is always defined in the child table.***

Syntax for using foreign key

FOREIGN KEY(column name) REFERENCES Parent_Table(PK of Parent Table);

REFERENCING ACTIONS

Referencing action with ON DELETE clause determines what to do in case of a DELETE occurs in the parent table.
Referencing action with ON UPDATE clause determines what to do in case of a UPDATE occurs in the parent table.

Actions:

1. **CASCADE :** This action states that if a DELETE or UPDATE operation affects a row from the parent table, then automatically delete or update the matching rows in the child table i.e., cascade the action to child table.
2. **SET NULL :** This action states that if a DELETE or UPDATE operation affects a row from the parent table, then set the foreign key column in the child table to NULL.
3. **NO ACTION :** Any attempt for DELETE or UPDATE in parent table is not allowed.
4. **RESTRICT :** This action rejects the DELETE or UPDATE operation for the parent table.

Q: Create two tables

Customer(customer_id, name)
Customer_sales(transaction_id, amount , **customer_id**)

Underlined columns indicate primary keys and bold column names indicate foreign key.
Make sure that no action should take place in case of a DELETE or UPDATE in the parent table.

Sol :
```
CREATE TABLE Customer (
        customer_id int Not Null Primary Key ,
        name varchar(30) ) ;

    CREATE TABLE Customer_sales (
        transaction_id Not Null Primary Key ,
        amount int ,
        customer_id int ,
        FOREIGN KEY(customer_id) REFERENCES Customer (customer_id)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION );
```

Q: Distinguish between a Primary Key and a Unique key in a table.

S.NO.	PRIMARY KEY	UNIQUE KEY
1.	Column having Primary key can't contain NULL value	Column having Unique Key can contain NULL value
2.	There can be only one primary key in Table.	Many columns can be defined as Unique key

Q: Distinguish between ALTER Command and UPDATE command of SQL.

S.NO.	ALTER COMMAND	UPDATE COMMAND
1.	It is a DDL Command	It is a DML command
2.	It is used to change the definition of existing table, i.e. adding column, deleting column, etc.	It is used to modify the data values present in the rows of the table.
3.	Syntax for adding column in a table: ALTER TABLE <tablename> ADD <Column name><Datatype> ;	Syntax for using UPDATE command: UPDATE <Tablename> SET <Columnname>=value WHERE <Condition> ;

AGGREGATE / GROUP FUNCTIONS

Aggregate / Group functions work upon groups of rows , rather than on single row, and return one single output. Different aggregate functions are : COUNT( ) , AVG( ) , MIN( ) , MAX( ) , SUM ( )

Table : EMPL

EMPNO	ENAME	JOB	SAL	DEPTNO
8369	SMITH	CLERK	2985	10
8499	ANYA	SALESMAN	9870	20
8566	AMIR	SALESMAN	8760	30
8698	BINA	MANAGER	5643	20
8912	SUR	NULL	3000	10

1. **AVG( )**

   This function computes the average of given data. e.g. SELECT AVG(SAL)
   FROM EMPL ;

   Output

AVG(SAL)
6051.6

2. **COUNT( )**

   This function counts the number of rows in a given column.

   If you specify the COLUMN name in parenthesis of function, then this function returns rows where COLUMN is not null.

   If you specify the asterisk (*), this function returns all rows, including duplicates and nulls.

   e.g. SELECT COUNT(*)
   FROM EMPL ;

   Output

COUNT(*)
5

   e.g.2 SELECT COUNT(JOB)
   FROM EMPL ;

   Output

COUNT(JOB)
4

3. **MAX( )**

   This function returns the maximum value from a given column or expression.

   e.g. SELECT MAX(SAL)
   FROM EMPL ;

   Output

MAX(SAL)
9870

4. **MIN( )**

This function returns the minimum value from a given column or expression.

e.g. SELECT MIN(SAL)
FROM EMPL ;

Output

MIN(SAL)
2985

5. **SUM( )**

This function returns the sum of values in given column or expression.

e.g. SELECT SUM(SAL)
FROM EMPL ;

Output

SUM(SAL)
30258

GROUPING RESULT – GROUP BY

The GROUP BY clause combines all those records(row) that have identical values in a particular field(column) or a group of fields(columns).

GROUPING can be done by a column name, or with aggregate functions in which case the aggregate produces a value for each group.

Table : EMPL

EMPNO	ENAME	JOB	SAL	DEPTNO
8369	SMITH	CLERK	2985	10
8499	ANYA	SALESMAN	9870	20
8566	AMIR	SALESMAN	8760	30
8698	BINA	MANAGER	5643	20

e.g. Calculate the number of employees in each grade.

SELECT JOB, COUNT(*)
FROM EMPL
GROUP BY JOB ;

Output

JOB	COUNT(*)
CLERK	1
SALESMAN	2
MANAGER	1

e.g.2. Calculate the sum of salary for each department.

SELECT DEPTNO , SUM(SAL)
FROM EMPL
GROUP BY DEPTNO ;

Output

DEPTNO	SUM(SAL)
10	2985
20	15513
30	8760

e.g.3. find the average salary of each department.
Sol:

*** One thing that you should keep in mind is that while grouping , you should include only those values in the SELECT list that either have the same value for a group or contain a group(aggregate) function. Like in e.g. 2 given above, DEPTNO column has one(same) value for a group and the other expression SUM(SAL) contains a group function.*

NESTED GROUP

- To create a group within a group i.e., nested group, you need to specify multiple fields in the GROUP BY expression. e.g. To group records **job wise** within **Deptno wise**, you need to issue a query statement like :

```
SELECT DEPTNO , JOB , COUNT(EMPNO)
FROM EMPL
GROUP BY DEPTNO , JOB ;
```

Output

DEPTNO	JOB	COUNT(EMPNO)
10	CLERK	1
20	SALESMAN	1
20	MANAGER	1
30	SALESMAN	1

PLACING CONDITION ON GROUPS – HAVING CLAUSE

- The **HAVING clause places conditions on groups** in contrast to WHERE clause that places condition on individual rows. While **WHERE conditions cannot include aggregate functions, HAVING conditions can do so.**
- e.g. To display the jobs where the number of employees is less than 2,

```
SELECT JOB, COUNT(*)
FROM EMPL
GROUP BY JOB
HAVING COUNT(*) < 2 ;
```

Output

JOB	COUNT(*)
CLERK	1
MANAGER	1

DATABASE TRANSACTIONS

TRANSACTION

A Transaction is a logical unit of work that must succeed or fail in its entirety. This statement means that a transaction may involve many sub steps, which should either all be carried out successfully or all be ignored if some failure occurs. A Transaction is an atomic operation which may not be divided into smaller operations.

Example of a Transaction

```
Begin transaction
    Get balance from account X
    Calculate new balance as X – 1000
    Store new balance into database file
    Get balance from account Y
    Calculate new balance as Y + 1000
    Store new balance into database file
End transaction
```

TRANSACTION PROPERTIES (ACID PROPERTIES)

1. **A**TOMICITY(All or None Concept) – This property ensures that either all operations of the transaction are carried out or none are.

2. **C**ONSISTENCY – This property implies that if the database was in a consistent state before the start of transaction execution, then upon termination of transaction, the database will also be in a consistent state.

3. **I**SOLATION – This property implies that each transaction is unaware of other transactions executing concurrently in the system.

4. **D**URABILITY – This property of a transaction ensures that after the successful completion of a transaction, the changes made by it to the database persist, even if there are system failures.

TRANSACTION CONTROL COMMANDS (TCL)

The TCL of MySQL consists of following commands :

1. BEGIN or START TRANSACTION – marks the beginning of a transaction.
2. COMMIT – Ends the current transaction by saving database changes and starts a new transaction.
3. ROLLBACK – Ends the current transaction by discarding database changes and starts a new transaction.
4. SAVEPOINT – Define breakpoints for the transaction to allow partial rollbacks.
5. SET AUTOCOMMIT – Enables or disables the default auto commit mode