# DATA MANAGMENT [SQL]

## Creating databases in My SQL:-

**The CREATE DATABASE command is used to create a database in MySQL.**

**Syntax--**

**CREATE DATABASE <database name>;**

**For example:-**

**CREATE DATABASE portal_express ;**

## Viewing Databases in MySQL:-

**If you want to know how many Databases in MySQL then, we need SHOW DATABASES command.**

**Syntax—**

**SHOW DATABASES;**

**For example:**

**SHOW DATABASES;**

## Accessing Database in MySQL:-

**After creating database we need to open it to work in it. We use USE command to access database in MySQL.**

**Syntax--**

**use <database name>;**

**For example:**

**use portal_express ;**

## Delete / Remove Database from MySQL:-

**To delete/remove database from MySQL we need DROP command.**

**Syntax--**

**DROP DATABASE <database name>;**

**For example:**

**DROP DATABASE portal_express;**

## Creating Tables in MySQL:-

**To create table in MySQL database, we need CREATE TABLE statement.**

**Syntax--**

**create table <table name>**

**(<column name > <data type> [<size>] <constraints of you want>,**

**<column name > <data type> [<size>] <constraints of you want>,**

**<column name > <data type> [<size>] <constraints of you want>....)**

**For example:**

**create table student**

**(Name varchar(40),**

**Roll_no integer,**

**Marks integer(10),**

**Class integer);**

## Viewing Table of Database:-

**If you want to see, how many tables has been created in Database, then we need SHOW TABLES command.**

**Syntax--**

**SHOW TABLES;**

**For example:**

**SHOW TABLES;**

## Inserting Data into Table:-

**INSERT INTO command is use to insert data into a table.**

**(A) Inserting data for all the columns into a table:-**

**• In the first method, it does not specify the column names where the data will be inserted, only their values.**

**Please make sure that the order of the values is in the same order as the columns represented in the structure of the table.**

**Syntax--**

**INSERT INTO <Table name>**

**VALUES (value1 for column1, value2 for column2, value3 for column3, ...);**

**For example:**

**INSERT INTO student**

**VALUES (Ravi, 26, 87, 12);**

**• The second method is from specific both the column names and the values to inserted.**

**Syntax--**

**INSERT INTO <Table name> (column1, column2, column3.....)**

**VALUES (value1, value2, value3,.....);**

**For example:**

**INSERT INTO student (Name, Rollno, Marks, Class)**

**VALUES (Rajesh, 2, 75, 12);**

**(B) Inserting Data into specific Columns of a Table:-**

**If you want to insert value in specific columns then we need following command.**

**Syntax--**

**INSERT INTO <Table name> (column1, column3, column7.....)**

**VALUES (value1, value3, value7 ...);**

**For example:**

**INSERT INTO student (Rollno, Class, Name, Marks)**

**VALUES (14, 11, Nilay, 81);**

**(C) Inserting NULL values into a Table:-**

**To insert value NULL in specific columns, you can type NULL without quotes and NULL will be inserted in that column.**

**Syntax-**

**INSERT INTO <Table name> (column1, column3, column7, column4.....)**

**VALUES (value1, NULL, value7, NULL ...);**

**For example:**

**INSERT INTO student (Rollno, Class, Name, Marks)**

**VALUES (14, 11, Nilay, NULL);**

## Inserting Dates into table:-

**Dates are by default entered in 'YYYY-MM-DD' format i.e., first four digits depicting year , followed by a hyphen , followed by 2 digits of month , followed by a hyphen and a two digit day. All this is enclosed in single quotes.**

**For example:**

**'2020-07-01'**

## Modifying Data in a Table:-

**(A) Updating multiple Columns:-**

**Modifying the values in more than one column can be done by separating the columns along with the new values using SET clause, separated by commas.**

**Syntax--**

**UPDATE <table name>**

**SET <column 1>=<new value>,<column 3 >=<new value>,<column 5>=<new value>**

**WHERE <condition>;**

**For example:**

**UPDATE student**

**SET Class=12, Marks=71**

**WHERE Name= 'Rajesh';**

**(B) Updating Single Value:-**

**Syntax--**

**UPDATE <table name>**

**SET <column 1>=<new value>**

**WHERE <condition>**

**For example:**

**UPDATE student**

**SET Class=11**

**WHERE Name= 'Ravi';**

**(C) Updating using an expression or formula:-**

**Like that: --**

**UPDATE <table name>**

**SET <column 1>=<new value> + 1000**

**WHERE <condition>;**

**For example:**

**UPDATE student**

**SET Marks=Marks + 5**

**WHERE Name= 'Rajesh';**

## Removing Data from a Table:-

**• The DELETE statement is used to delete rows from a table.**

**Syntax--**

**DELETE FROM <table name>**

**WHERE <condition>;**

**For example:**

**DELETE FROM student**

**WHERE Name= 'Rajesh';**

**• Truncate Statement: -**

**The MySQL TRUNCATE command is used to delete all the rows from Table and free the space containing the Table.**

**Syntax--**

**TRUNCATE TABLE <table name>;**

**Difference between DELETE and TRUNCATE statement:**

**DELETE Statement: This command deletes only the rows from the table based on the condition given in the where clause or deletes all the rows from the table if no condition is specified. But it does not free the space containing the table.**

**TRUNCATE Statement: This command is used to delete all the rows from the table and free the space containing the table.**

**ALTER TABLE Command:-**

**The ALTER TABLE command is used to modify the definition (structure) of a table by modifying the definition of its columns. The ALTER TABLE command is used to perform the following operations:**

- **To add a column to an existing table.**
- **To rename any existing column.**
- **To change the datatype of any column or to modify its size.**
- **To remove or physically delete a column.**

## (A) Adding a Column to an existing Table:-

**To add new column in existing Table, we need ADD command.**

**Syntax--**

**ALTER TABLE <table name> ADD (<column_name> <datatype> [size]);**

**For example:**

**ALTER TABLE student ADD (School char(50));**

## (B) Adding a Column with default value: -

**Alter table command can be used to add a new column to an existing Table with default values.**

**Syntax--**

**ALTER TABLE <table name> ADD (<column_name> <datatype> [size] default <data>);**

**For example:**

**ALTER TABLE student ADD (School char(50) default "K.V");**

## (C) Modifying an existing Column definition:-

**The MODIFY clause can be used with ALTER TABLE command to change the datatype, size constraints related to any column of table.**

**Syntax--**

**ALTER TABLE <table name>**

**MODIFY (<Column name> <datatype>);**

**For example:**

**ALTER TABLE student**

**MODIFY (Name char(100));**

## (D) Renaming a Column:-

**Syntax---**

**ALTER TABLE <table name>**

**CHANGE <old Column name> <new column name> <datatype [size]>;**

**For example:**

**ALTER TABLE student**

**CHANGE school school_name char(60);**

## (E) Removing a Column:-

**To remove or drop a column in a table. Like that:-**

**Syntax--**

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

**For example:**

**ALTER TABLE student DROP school_name;**

## Viewing Structure of a Table:-

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

**DESCRIBE or DESC <table name>;**

**For example:**

**DESC student;**

**DROP TABLE command:-**

**Sometimes, we may need to physically remove a table which is not in use. DROP TABLE command is used to remove/delete a table permanently. It should be kept in mind that we cannot drop a table if it contains records. That is why all the rows of the table have to be deleted first and only then can the table be dropped.**

**Syntax--**

**DROP TABLE <table name>;**

**For example:**

**DROP TABLE student;**

## MAKING SIMPLE QUERIES THROUGH SELECT COMMAND

**The SELECT command can perform selection as well as projection. It is most extensively used MySQL command. The general form of the statement is:-**

**SELECT what to select**

**FROM which table**

**WHERE condition to satisfy;**

**Before going forward Consider the following table or look the table carefully:-**

| name | species | sex | birth | owner | city |
|---|---|---|---|---|---|
| Ace | DOG | M | 2018-07-16 | RAHUL | MUMBAI |
| Alex | BIRD | F | 2019-09-13 | AMIT | DELHI |
| Ally | CAT | F | 2015-01-01 | RAVI | MUMBAI |
| Axel | DOG | F | 2012-04-04 | KRITI | DELHI |
| Bella | CAT | F | 2013-05-09 | JATIN | CHENNAI |
| Bingo | SNAKE | M | 2016-06-30 | ABHAY | AGRA |
| Bob | DOG | M | 2020-06-12 | NIKHIL | DELHI |
| Chaz | SNAKE | F | 2011-09-13 | AMIT | RAIPUR |
| Chevy | BIRD | F | 2012-11-21 | KRITI | SRINAGAR |
| Max | DOG | M | 2014-12-24 | ELVISH | PATANA |
| Dude | CAT | M | 2015-03-31 | GAURAV | AGRA |

**Selecting all Data:-**

**For example**

**SELECT * FROM pet;**

**Selecting Particular Rows:-**

**1 = Select all pets with gender (sex) as male ("m").**

**SELECT * FROM pet**

**WHERE sex = ‘m’;**

**2 = Select all pets that were born on or after Jan 10, 2016.**

**SELECT * FROM pet**

**WHERE birth >= '2016-1-10’;**

**3 = Select all female-dogs.**

**SELECT * FROM pet**

**WHERE sex = 'f' AND species = ‘dog’;**

**4 = Select all snakes or birds.**

**SELECT * FROM pet**

**WHERE species = 'snake' OR species = ‘bird’;**

**5 = Select all male cats.**

**SELECT * FROM pet**

**WHERE (species = 'cat' AND sex = 'm');**

**Selecting Particular Columns:-**

**1 = Display names and birth-dates of all pets.**

**SELECT name, birth FROM pet ;**

**2 = Display owners of pets born after Dec 2017.**

**SELECT owner FROM pet**

**WHERE birth = '2017-12-31';**

**Eliminating Redundant Data (with keyword DISTINCT):-**

**1 = Display names of all pets owner (non Redundant).**

**SELECT DISTINCT owner FROM pet ;**

**2 = Display distinct species of pets from table pet.**

**SELECT DISTINCT (species) FROM pet ;**

**Select From All the Rows - ALL Keyword:-**

**• It will give values of name column from every row of the table without considering the duplicate entries.**

**SELECT ALL name FROM pet;**

**Performing Simple Calculation: -**

**To perform simple calculations, you can write the expression/formula to be calculated next to keyword SELECT, e.g.**

**1. To calculate 3.14159*6*6**

**SELECT 3.14159 * 6 * 6 ;**

**2. To obtain current system date.**

**SELECT curdate() ;**

**Using Column Aliases:-**

**The columns that you select in a query can be given a different name i.e., column alias name for output purposes. As per following syntax**

**Select <columnname> AS [column alias] [, <columnname> AS [column alias]] ….. From <table name>;**

**For example:**

**SELECT name , species AS "Type of animals" FROM pet ;**

**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 includes both lower value and the upper value. For example, to list the owner, name whose pet's date of birth between 1 Jan 2014 to 1 Jan 2019 (both inclusive), would be:**

**SELECT owner, name**

**FROM pet**

**WHERE birth BETWEEN '2014-01-01' AND '2019-01-01';**

**Condition Based on a List:-**

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

**For example:**

**To display a list of pet from 'DELHI', 'MUMBAI', 'CHENNAI' or 'AGRA' cities, you may give.**

**SELECT * FROM pet**

**WHERE city IN ('DELHI', 'MUMBAI', 'CHENNAI', 'AGRA');**

**• The NOT IN operator finds rows that do not match in the list. So if you write.**

**SELECT * FROM pet**

**WHERE city NOT IN ('DELHI', 'MUMBAI', 'CHENNAI', 'AGRA');**

**It will list members not from the cities mentioned in the list.**

**Condition Based on Pattern Matches:-**

**SQL also includes a string-matching operator, LIKE, for comparisons on character strings using patterns. Patterns are described using two special wildcard characters:**

**• Percent (%):- It matches any string.**

**• Underscore (_):- It matches any one character.**

**• The LIKE keyword is used to select rows containing columns that match a wildcard pattern.**

**For Examples:**

**1 = To list pet in which owner name starting with 'A', then command is:**

```
SELECT name, owner, city
FROM pet
WHERE owner LIKE 'A%';
```

**2 = To list names of pets who have names ending with 'y', the command would be:**

```
SELECT name
FROM pet
WHERE name LIKE '%y';
```

**Searching for NULL:-**

**The NULL value in a column can be searched for in a table using IS NULL in the WHERE clause. (Relational operators like = <> etc. can't be used with NULL).**

**For example:**

**To list details of all pets whose date of birth contain NULL (i.e., novalue), then you use the command:**

```
SELECT name, owner, city
FROM pet
WHERE birth IS NULL;
```

**• Non-NULL values in a table can be listed using IS NOT NULL.**

**INSERTING DATA INTO ANOTHER TABLE**

**You already know that data is added to tables using INSERT INTO command but INSERT command can also be used to take or derive values from one table and place them in another by using it with a query.**

**To do this, simply replace the VALUES clause with an appropriate query as shown in the following example:**

**INSERT INTO branch1**

**SELECT * FROM branch2**

**WHERE gross > 7000.00;**

**It will extract all those rows from branch2 that have gross more than 7000.00 and insert this produced result into the table branch1. But for above command to work, table namely branch1 must be an existing table of the database.**

## CREATING TABLES WITH SQL CONSTRAINTS:-

**You already know that a table is created using CREATE TABLE command. But while creating tables, we may need to apply certain conditions on columns. To apply conditions on columns, SQL constraints are used.**

**Constraint: - A constraint is a condition or check applicable on a field or set of fields.**

## SQL constraints:-

**Some common SQL constraints**

**NOT NULL**

**Ensures that a column cannot have NULL value.**

**DEFAULT**

**Provides a default value for a column when none is specified.**

**UNIQUE**

**Ensures that all values in a column are different.**

**CHECK**

**Makes sure that all values in a column satisfy certain criteria.**

**Primary Key**

**Used to uniquely identify a row in the table.**

**Foreign Key**

**Used to ensure referential integrity of the data.**

## SQL NOT NULL constraints:-

**By default, a column can hold NULL. If you do not want to allow NULL value in a column, you will want to place a constraints on this column specifying that NULL is now not an allowable value.**

**Look the following statement:**

**CREATE TABLE customer**

**(ID integer NOT NULL, First_name varchar (30) NOT NULL, Last_name varchar (30));**

**• Column ID & First_name cannot include NULL, while Last_name can include NULL.**

**SQL DEFAULT Constraints:-**

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

**For example, if we create a table as below:**

**CREATE TABLE Student**

**(Student_ID integer,**

**Last_Name varchar (30),**

**First_Name varchar (30),**

**Score DEFAULT 70);**

**• And execute the following SQL statement:**

**INSERT INTO Student**

**(Student ID, Last_Name, First_Name)**

**VALUES ('10', 'Kumar', 'Rahul');**

**After this SQL query, look the table**

| Student ID | Last_name | First_name | Score |
|---|---|---|---|
| 10 | Kumar | Rahul | 70 |

**SQL 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.**

**For example, in the following CREATE TABLE statement.**

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

**Column ID has a unique constraint, and hence cannot include duplicate values. Such constraint does not hold for columns Last_Name and First_Name. So, if the table already contains the following rows:**

| ID | Last_name | First_name |
|---|---|---|
| 1 | Roa | Ravi |
| 2 | Raj | Shahu |
| 3 | Kumar | Rahul |

**Executing the following SQL statement:**

```
INSERT INTO Customer
VALUES ('3','kumar', 'Rahul');
```

**Will result in an error because the value 3 already exists in the ID column, thus trying to insert another row with that value violates the UNIQUE constraint.**

## SQL CHECK Constraint:-

**The CHECK constraint ensures that all values in a column satisfy certain condition. Once defined, the database will only insert a new row or update an existing row if the new value satisfies the CHECK constraint. The CHECK constraint is used to ensure data quality.**

**For example, in the following CREATE TABLE statement.**

**CREATE TABLE Customer**

**(ID integer CHECK (ID > 0),**

**Last_Name varchar (30),**

**First_Name varchar (30));**

## PRIMARY KEY Constraint:-

**A primary key is used to uniquely identify each row in a table. It can either be part of the actual record itself, or it can be an artificial field (one that has nothing to do with the actual record). A primary key can consist of one or more fields on a table. When multiple fields are used as a primary key, they are called a composite key.**

**Primary keys can be specified either when the table is created (using CREATE TABLE) or by changing the existing table structure (using ALTER TABLE).**

**Defining Primary Key through Create Table Command**

**You can define a primary key in CREATE TABLE command through keywords PRIMARY KEY.**

**Below are examples for specifying a primary key when creating a table:**

**CREATE TABLE Customer**

**(ID integer not null PRIMARY KEY,**

**Last_Name varchar (30),**

**First_Name varchar (30));**

**Or**

**CREATE TABLE Customer**

**(ID integer not null,**

**Last_Name varchar (30),**

**First_Name varchar (30),**

**PRIMARY KEY (ID));**

**• The latter way is useful if you want to specify a composite primary key (i.e., having a group of fields)**

**For example:-**

**CREATE TABLE Customer**

**(Branch integer not null,**

**ID integer not null,**

**Last_Name varchar (30),**

**First_Name varchar (30),**

**PRIMARY KEY (Branch, ID);**

**Defining Primary Key through Alter Table Command**

**You can define a primary key in ALTER TABLE command through keywords.**

**ADD PRIMARY KEY (<key-field>)**

**Below are examples for specifying a primary key by altering a table:**

**ALTER TABLE Customer**

**ADD PRIMARY KEY (ID);**

**• Before using the ALTER TABLE command to add a primary key, you'll need to make sure that the field is defined as 'NOT NULL', in other words, NULL cannot be an accepted value for that field.**

***Foreign key Constraints:-**

**Whenever two tables are related by a common column (or set of columns), then the related column(s) in the parent table (or primary table) should be either declared a PRIMARY KEY or UNIQUE key and the related column(s) in the child table (or related table) should have FOREIGN KEY constraint.**

**For instance, if we have two tables having structures as given below:**

**Table: CUSTOMER**

Table: CUSTOMER

| Column name | Characteristic |
|---|---|
| SID | Primary key |
| Last_name | |
| First_name | |

Table: ORDERS

| Column name | Characteristic |
|---|---|
| Order_ID | Primary key |
| Order_Date | |
| Customer_SID | Foreign key |
| Amount | |

**In the above example, the Customer_SID column in the ORDERS table is a foreign key pointing to the SID column in the CUSTOMER table.**

**Just like primary key, Foreign key can also be created in two ways: through CREATE TABLE and ALTER TABLE commands.**

## Defining Foreign key through Create Table

**In Create Table command, you can add foreign key's definition through following syntax:**

**Foreign Key (<column-to-be-designated-as-foreign-key>) references**

**Master-Table (<primary-key-of master-table>);**

**Following example shows how to specify the foreign key when creating the ORDERS table:**

**CREATE TABLE ORDERS**

**(Order_ID integer,**

**Order_Date date,**

**Customer_SID integer,**

**Amount double,**

**Primary Key (Order ID),**

**Foreign Key (Customer_SID) references CUSTOMER (SID));**

**• The above code will designate Customer_SID field of ORDERS table as foreign key referencing SID field of CUSTOMER table.**

**Defining Foreign key through Alter Table**

**In Alter Table command, you can add foreign key's definition through following syntax :**

**ALTER TABLE <table-name>**

**ADD FOREIGN KEY (<column-to-be-designated-as-foreign-key>)**

**references Master-Table(<primary-key-of master-table>) ;**

**Following example specifies a foreign key by altering a table. This assumes that the ORDERS table has been created, and the foreign key has not yet been put in:**

**ALTER TABLE ORDERS**

**ADD FOREIGN KEY (customer_sid) REFERENCES CUSTOMER (SID);**

**Sorting in SQL - ORDER BY:**

**If you want to sort or order the result set, you can use the ORDER BY clause of SQL, SELECT statement as per following format:**

**Syntax:-**

**SELECT <comma separated select list> FROM <table>**

**[WHERE <condition>]**

**ORDER BY <fieldname> [ASC | DESC] [, <fieldname> [ASC|DESC], ...];**

**• Keywords ASC and DESC denote the order - ASC stands for ascending and the DESC stands for descending.**

**• If you do not specify any order keyword ASC or DESC, then by default ORDER BY clause sorts the result set in ascending order.**

**For example:**

**SELECT * FROM data**

**ORDER BY marks;**

**OR**

**SELECT * FROM data**

**ORDER BY marks ASC;**

**Ordering Data on Multiple Columns:-**

**To order the result set on multiple columns, you can specify the multiple column names in ORDER by clause along with the desired sort order, i.e., as:**

**SELECT <comma separated select list> FROM <table>**

**[WHERE <condition>]**

**ORDER BY <fieldname1> [ASC/DESC], [<field name1> [ASC|DESC], ..... ];**

**For example:**

**The following statement will sort the records firstly on the column name Section and then on the basis of descending order of column marks.**

**SELECT * FROM data**

**ORDER BY section ASC, marks DESC;**

**Ordering Data on the basis of a Mathematical Expression:-**

**The ORDER BY clause allows you to include the mathematical expression to order the result set by it.**

**• Consider the following example statement that arrange the result set on the basis of a calculated result:**

```
SELECT rollno, name, grade, section, marks * 0.35 FROM DATA
WHERE marks > 60
ORDER BY section ASC, marks * 0.35 DESC;
```

**Sorting on Column Alias:-**

**If you want, you can provide a column alias name to the mathematical expression in the select list.**

**For example:**

```
SELECT rollno, name, grade, section, marks * 0.35 AS term1 FROM DATA
WHERE marks > 60
ORDER BY section ASC, term1 DESC;
```

**AGGREGATE FUNCTIONS:-**

**Till now, we have studied about single-row functions which work on a single value. SQL also provides multiple-row functions which work on multiple values. So, we can apply SELECT query on a group of records rather than the entire table. Therefore, these functions are called Aggregate functions or Group functions.**

**Many group function accept the following options:**

**• DISTINCT: - This option causes a group function to consider only distinct value of the argument expression.**

**• ALL: - This option cause a group function to consider all values including duplicates.**

**. AVG: -**

**This function computes the average of given data.**

**Syntax--**

**AVG ([DISTINCT | ALL] n);**

**For example:-**

**SELECT AVG (sal) 'Average' FROM empl;**

**COUNT:-**

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

**Syntax--**

**COUNT ({* [DISTINCT | ALL] expr})**

**• Return the number of rows in the query.**

**• If you specify argument expr, this Function returns rows where expr is not null. You can count either all rows, or only distinct values of expr.**

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

**For example:**

**• Count number of records a table empl.**

**SELECT COUNT (*) "Total"**

**FROM empl;**

**• Count number of jobs in table emp.**

**SELECT COUNT (job) "Job Count"**

**FROM empl;**

**• How many distinct jobs are listed in table empl?**

**SELECT COUNT (DISTINCT job) "Distinct Jobs"**

**FROM empl;**

**MAX:-**

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

**Syntax--**

**MAX ([DISTINCT | ALL] expr)**

**• A Returns maximum value of argument expr.**

**For example:**

**• Display maximum salary from table empl.**

```
SELECT MAX (sal) "Maximum Salary"
FROM empl;
```

**MIN:-**

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

**Syntax--**

**MIN ( [DISTINCT | ALL] expr)**

**• A Returns minimum value of expr.**

**For example:**

**• Display the Joining date of senior most employee.**

```
SELECT MIN (hiredate) as "Minimum Hire Date"
FROM empl;
```

**SUM:-**

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

**Syntax--**

**SUM([DISTINCT | ALL] n)**

**• A Returns sum of values of n.**

**For example:**

**• Display total salary of all employees listed in table empl.**

**SELECT SUM (sal) as "Total Salary"**

**FROM empl;**

## GROUPING RESULT - GROUP BY

**The GROUP BY clause combines all those records that have identical values in a particular field or a group of fields.**

**For example:**

**SELECT job, count (*)**

**FROM empl**

**GROUP BY job;**

## Nested Group - Grouping on Multiple Column --

**With Group By clause, you can create groups within groups. Such type of grouping is called Nested grouping.**

**This can be done by specifying in GROUP BY expression, where the first field determines the highest group level, the second field determines the second group level, and so on. The last field determines the lowest level of grouping.**

**For example:**

**SELECT Deptno, job, count (empno)**

**FROM empl**

**GROUP BY Deptno, job;**

## Placing Condition on Groups - HAVING Clause

**The HAVING clause place conditions on groups in contrast to WHERE clause that places conditions on individual rows. While WHERE conditions can not include aggregate functions, HAVING condition can do so.**

**For example:**

**To calculate the average gross and total gross for employees belonging to 'E4'grade.**

**SELECT AVG(gross) , SUM(gross)**

**FROM employee**

**GROUP BY grade**

**HAVING grade = 'E4'**

**SQL JOINS -**

**SQL join is a query that combines rows from two or more table .**

**Syntax -**

**Select <field list>**

**From <table1>,<table 2>,……**

**Where <join condition for the tables >**

**Example -**

**Select ENAME , LOC**

**From EMPL , DEPT**

**Where ENAME = "Anoop"**

**And EMPL.DEPTNO = DEPT.DEPTNO**

**Types of SQL joins -**

**(i) Cartesian Product :-**

**SQL join query without any join condition return all the records of joined with all records of other table .**

**Syntax -**

**Select * from <table 1>,<table 2>…..**

**(ii) Equi Join :-**

**SQL join query that join two or more tables based on a condition using equality operator.**

**Syntax -**

**Select * from <table 1>,<table 2>**

**Where < column of table 1 > = < column of table 2 >**

**(iii) Inner join :-**

**An inner join implement an equi join . In this join , only those rows are returned from both the table that satisfy the join condition .**

**Syntax -**

**Select <field list>**

**From <table1> inner join <table 2>**

**On <join condition for the tables >;**

**(iv) Natural Join :-**

**The join in which only one of the identical column exits.**

**Syntax -**

**Select ***

**From < table 1 > Natural join < table 2 >**

**(v) Left join:-**

**The Left join is a particular type of join that selects rows from both left and right tables that are matched , plus all rows from left table even with no matching rows found in right table .**

**Syntax -**

**Select <field list>**

**From <table1> Left Join <table 2>**

**On <join condition for the tables >;**

**(v) Right Join:-**

**The Right join is a particular type of join that selects rows from both left and right tables that are matched , plus all rows from right table even with no matching rows found in left table .**

**Syntax -**

**Select <field list>**

**From <table1> Right Join <table 2>**

**On <join condition for the tables >**