

Introduction to SQL : Structured Query Language

What is SQL ?

SQL is a standard that specifies how

- a relational schema is created
- data is inserted/updated in the relations
- data is queried
- transactions re started and stopped
- program access data in the relations
- and a host of other things to do.

SQL provides a small and concise set of commands which can be combined with dBase commands to define and access data from a database. Every relational database management system (RDBMS) is required to support/implement the SQL standard.

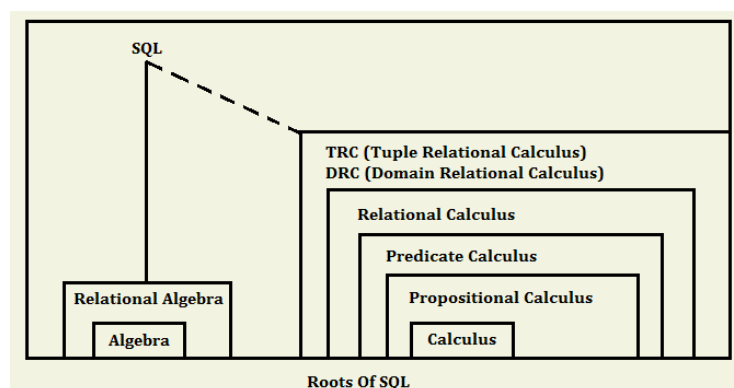
Various SQL Standards are :

- SQL 1 also SQL 86
- SQL 2 also SQL 92
- SQL 3 also SQL 99

There are two types of SQL :

1. *Interactive SQL* - In interactive SQL, we enter a command, it is executed and we may get the output.
2. *Embedded SQL* - In Embedded SQL, the SQL commands is embedded into the programs that are written in other language such as FORTRAN, Pascal, COBOL, C. The output of the SQL commands in embedded SQL is passed off to variables enabled by the program in which it is embedded.

The roots of SQL :



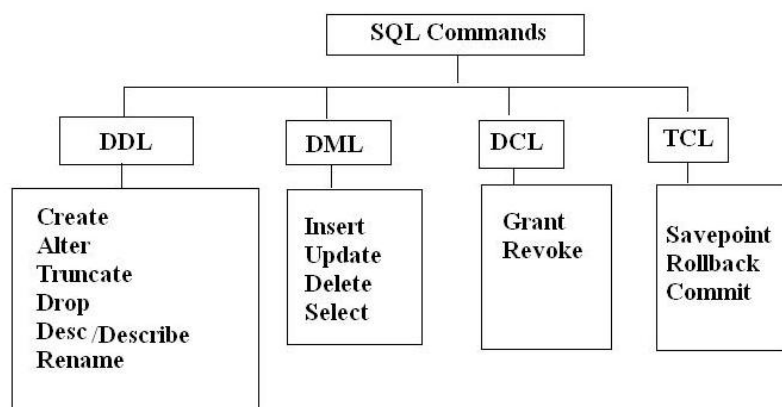
Difference Between SQL and other Programming Languages :

SQL	Conventional Programming Languages
SQL is a non Procedural Language	These Languages are commonly Procedural.
SQL statements specify what data operations should be performed rather than how to perform,	As these follows a procedure, so they need how to perform on data

Components or Elements of SQL

There are three components of SQL -

- DDL (Data Definition Language) - is used for creation and deletion of the database. It provides a set of statements to create/define a database, modify its structure after it has been created, and destroy it after it is no longer needed.
- DML (Data Manipulation Language) - is used for manipulating the database. Manipulation means insert, modify, delete or query the data in the database. It enables users to access or manipulate the data as organised by appropriate data model.
- DCL (Data Control Language) - is used to grant and revoke authorization for database access, auditing the database use and dealing with transactions. It specifies how to restrict a user/se of users to access only certain parts of data, perform only certain types of queries.
- TCL (Transaction Control Language) - is used to manage the changes made by DML statements. It specifies how transactions can be started/stopped, how a set of concurrently executing transactions can be managed.



Data Types in SQL

Data Type	Syntax	Explanation (if applicable)
integer	integer	
smallint	smallint	
numeric	numeric(p,s)	Where p is a precision value; s is a scale value. <i>e.g.</i> , numeric(6,2) is a number that has 4 digits before the decimal and 2 digits after the decimal.
decimal	decimal(p,s)	Where p is a precision value; s is a scale value.
real	real	Single-precision floating point number.
double precision	double precision	Double-precision floating point number.
Float	float(p)	Where p is a precision value.
character	char(x)	Where x is the number of characters to be stored. This data type is space padded to fill the number of characters specified.
character varying	varchar2(x)	Where x is the number of characters to be stored. This data type is NOT space padded.
Bit	bit(x)	Where x is the number of bits to be stored.
Date	date	Stores year, month and day values.
Time	time	Stores the hour, minute and second values.
timestamp	timestamp	Stores year, month, day, hour, minute and second values.

Advantages and Disadvantages of SQL

Advantages :

1. High Speed - SQL Queries can be used to retrieve large amounts of records from a database quickly and efficiently.
2. Well Defined Standards Exist - SQL Databases use long-established standard, which is being adopted by ANSI and ISO. Non SQL databases do not adhere to any clear standard.
3. SQL as a language is independent of the way it is implemented internally. A query returns the same result regardless of whether optimizing has been done with indexes or not.
4. No Coding Required - Using standard SQL, it is easy to manage database systems without having to write substantial amount of code. Also applications written in SQL can be easily ported across systems. Such porting could be required when the underlying DBMS needs to be upgraded because of change in transaction volumes or when a system developed in one environment is to be used on another DBMS.
5. Emergence of ORDBMS - Previously SQL databases were synchronous with relational database. With the emergence of object oriented DBMS, object storage capabilities are extended to relational database.

Disadvantages :

1. Difficulty in Interfacing Interfacing an SQL database is more complex than adding a few lines of code.
2. Scalability - Users have to scale relational database on powerful servers that are expensive and difficult to handle. To scale relational database, it has to be distributed onto multiple servers. Handling tables on different servers is a chaos.

SQL Commands : Data Definition Language Commands

Before describe the SQL Commands, Let us discuss some rules for the commands.

Rules for SQL Commands

- SQL statements are not case sensitive. It can be typed in lower case or upper case or mix of both.
- The statements can be typed in a single line or multiple lines.
- A semicolon is placed to terminate the SQL statements
- The Keywords can't be distributed across the line but statements may be.
- A comma(,) is used to separate parameter without a clause.
- Characters and Date Constants or literals must be enclosed in single quotes('A').
- A command can be type either full or first 4 characters generally.

SQL Commands : DDL Commands (Data Definition Language Commands)

1. The CREATE TABLE Command -

The CREATE TABLE Command is used to create or define the tables in a database.

Syntax

```
CREATE TABLE <table_name>
(
  <Column1_specification>,
  <Column2 specification>,
  <Column3 specification>,
  .....
);
```

The Column Specification includes -

- column name
- data type
- [size or length of column] //[] means optional
- [constraints] // [] means optional

The table_name gives the name of the table you created, Column name gives the name of the column, Data Type specifies what type of data the column can hold and the size is optional, Constraints is a condition or check applicable that is apply on a field or set of fields.

Example -

```
CREATE TABLE Employee
(
  Emp_ID INTEGER,
  Emp_First_Name CHARACTER(25),
  Emp_Last_Name CHARACTER(25),
  Address VARCHAR(255),
  Salary double
);
```

This will create Employee table with five columns namely, Emp_ID, Emp_First_Name, Emp_Last_Name, Address, Salary.

2. The ALTER TABLE Command

The ALTER Command is used to add, delete or modify columns and constraints in the existing table. The ALTER Command performs following functions :

- Add, Drop(Delete), Modify table Columns
- Add and Drop Constraints
- Enable and Disable Constraints

Syntax :

- *To Add a Column*
- ALTER TABLE Table_Name
ADD <Column_specification>

For Example,

```
ALTER TABLE Employee  
ADD Basic_Pay real;
```

The above query will add a column Basic_Pay to the Employee table whose data type is real.

- *Syntax To DROP a Column*
- ALTER TABLE table_name
DROP COLUMN column_name;

Example :

```
ALTER TABLE Employee  
DROP COLUMN Last_Name;
```

- *Syntax To MODIFY a Column*
- ALTER TABLE table_name
MODIFY column_name data_type;

Example :

```
ALTER TABLE Employee  
MODIFY Salary float;
```

- *Syntax to delete a Constraint*
- ALTER TABLE table_name
DROP [CONSTRAINT|INDEX] Constraint_Name;

3. Rename Command

Rename Command is used to change the name of the table or the database object.

Syntax :

```
RENAME old_table_name  
TO new_table_name;
```

Example :

```
RENAME Employee TO  
WIPRO_Employees;
```

4. The DROP TABLE Command

The SQL DROP Command is used to remove the table definition and all data, indexes, triggers, constraints, and permission specified for the table. Once the DROP Command is executed, the table will be deleted and all the data/information available in the table would be lost forever. And we cannot even rollback after dropping a table.

Syntax :

```
DROP TABLE table_name;
```

Example :

```
DROP TABLE Employee;
```

Integrity Constraints in DBMS

- Constraints are the rules that the table must satisfy. These can be specified at the time of creating table or can also be added to table by using alter table statement.
- Constraints prevent the table from deletion, if there is any dependency

Types of Integrity Constraints in DBMS are :

- i. DEFAULT Constraint
- ii. NOT NULL Constraint
- iii. UNIQUE Constraint
- iv. CHECK Constraint
- v. PRIMARY KEY Constraint
- vi. FOREIGN KEY Constraint

Syntax of Constraints :

```
Create table table_name      (  
    column1 datatype column1 constraint,  
    column2 datatype column2 constraint,  
    column3 datatype  
);
```

Constraints can be defined at 2 levels :

- **Column level** - At column level the constraints are referred for a single column and is defined within the specification.
- **Table Level** - At table level when constraints are defined at table level then it can be referred to one or more columns and it can be defined separately from the definitions of the columns in the table.

DEFAULT CONSTRAINT

It is used to define a default value for an attribute.

Example :

```
CREATE TABLE employee  
(  
    EID int,  
    Name varchar(20),  
    Address varchar(50),  
    Salary int,  
    Sex char(1) DEFAULT='M'  
);
```


NOT NULL Constraint

- NOT NULL constraint is used when we do not want null values to be entered. It ensures that null values are not permitted in the column.
- The columns without NOT NULL constraint can have null values. By default, a column can contain NULL.

Example :

```
CREATE TABLE employee
(
    EmpID int NOT NULL,
    Name varchar(30) NOT NULL,
    Address varchar(50),
    Salary int,
    Sex char(1) DEFAULT='M'
);
```

The column "EmpID" and "Name" cannot include NULL, But the column "Address" can.

```
INSERT INTO employee (EmpID, Address, Salary)
VALUES ("1000","Bhiwani",29000)
```

This query would returned an error because it will lead to column Name being NULL, which violates the NOT NULL constraint.

UNIQUE Constraint

- The UNIQUE constraint is used to uniquely identify each record in a database. It requires that each value in a column should be unique,i.e. no 2 rows of same column should have duplicate values.
- It provides you the guarantee for uniqueness in a column or set of columns and allows us to input the NULL values, unless there is not defined NOT NULL constraint for the same column.
- As in UNIQUE constraint, there is a fact that there could be and number of rows which can include NULL for columns without NOT NULL constraints, reason because nulls are not considered equal to anything.
- It can be defined at table level or at column level.

Example :

```
CREATE TABLE employee (
    EID int NOT NULL,
    Name varchar(20) NOT NULL,
    Address varchar(50),
    Salary int,
    Sex char(1) DEFAULT='M',
    UNIQUE (EID) );
```

The column "EmpID" cannot have null values and the values in it are distinct.

CHECK constraint :

- The check constraint is used to check the value which is entered into a record. It is used to define condition which each row must satisfy.
- If the condition value evaluates to fall, then the record violates the constraint and you cannot enter it into the table.
- We cannot define check constraint in SQL view.

Syntax :

```
CREATE TABLE table_name
(
    column1 datatype,
    column2 datatype,
    ....
    CONSTRAINT constraint_name CHECK
    (column-name_contition));
```

The check constraint can be defined by a create table statement or alter table statement. A single column can contain a number of check constraints point it can be defined at the table level or column level

Example :

```
CREATE TABLE employee
(
    EmpID int NOT NULL,
    Name varchar(30) NOT NULL,
    Address char(50),
    Sex char(1) DEFAULT='M',
    Salary int,
    PRIMARY KEY(EmpID),
    CONSTRAINT chk_Salary CHECK (Salary>20000) );
```

The above query will check the chk_Salary constraint from table employee. The constraint will ensure that Salary contains value greater than 20000. We can also define CHECK constraint as :

```
CREATE TABLE employee (
    EmpID int NOT NULL,
    Name varchar(30) NOT NULL,
    Address char(50),
    Sex char(1) DEFAULT='M',
    Salary int CHECK(Salary>20000),
    PRIMARY KEY(EmpID),
);
```

CHECK Constraint using ALTER TABLE Statement :

Add a CHECK Constraint :

Syntax :

```
ALTER TABLE table_name  
ADD CONSTRAINT constraint_name CHECK  
(column_name condition);
```

Example :

```
ALTER TABLE employee  
ADD CHECK (EmpID>999);
```

This CHECK Constraint on EmpID column can be added only when the table is already created.

```
ALTER TABLE employee  
ADD CONSTRAINT chk_EmpID CHECK (EmpID>999 AND Salary>17000);
```

The above CHECK Constraint is defined for multiple columns.

DROP a CHECK Constraint

Syntax :

```
ALTER TABLE table_name  
DROP CONSTRAINT constraint_name;
```

Example :

```
ALTER TABLE employee  
DROP CONSTRAINT chk_EmpID;
```

ENABLE a CHECK Constraint

Syntax :

```
ALTER TABLE table_name  
ENABLE CONSTRAINT constraint_name;
```

Example :

```
ALTER TABLE employee  
ENABLE CONSTRAINT chk_Salary;
```

DISABLE a CHECK Constraint

Syntax :

```
ALTER TABLE table_name  
DISABLE CONSTRAINT constraint_name;
```

Example :

```
ALTER TABLE employee DISABLE CONSTRAINT chk_Salary;
```

Primary Key and Foreign Key Integrity Constraint in DBMS with Example

Primary key constraint

A primary key constraint is a field in a table which uniquely identifies each rows/records in a database table.

Characteristics of Primary Key :

- Primary key constraint must contain unique values.
- Primary key constraint column cannot have null values.
- Table can have only one primary key.
- If multiple fields are used as a primary key, then they are called as composite key.

Example :

```
CREATE TABLE employee
(
    EmpID int NOT NULL,
    Name varchar(30) NOT NULL,
    Address varchar(50),
    Salary int,
    Sex char(1) DEFAULT = 'M',
    PRIMARY KEY (EmpID)
);
```

Primary key applied on EmpID column, therefore now the column cannot be left blank nor duplicate values can exist. **A primary key constraint keeps the attributes of UNIQUE Constraint in itself. A table can contain only one primary key constraint, but can have several many constraints.**

ADD or DROP a PRIMARY KEY using ALTER TABLE Statement :

Alter table statement is used, if we want to add or drop constraint later.

Add a primary key

Syntax

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name PRIMARY
KEY (col1, col2, ...,coln);
```

Example :

```
ALTER TABLE employee
ADD CONSTRAINT employee_pk PRIMARY
KEY (EmpID);
```

By using alter table statement, you can add PRIMARY KEY constraint on the existing table employee named employee_pk.

DROP a Primary Key :

Syntax :

```
ALTER TABLE table_name  
DROP CONSTRAINT constraint_name;
```

Example :

```
ALTER TABLE employee  
DROP CONSTRAINT employee_pk;
```

DISABLE a Primary Key

Syntax :

```
ALTER TABLE table_name  
DISABLE CONSTRAINT constraint_name;
```

Example :

```
ALTER TABLE employee  
DISABLE CONSTRAINT employee_pk;
```

ENABLE a Primary Key :

Syntax :

```
ALTER TABLE table_name  
ENABLE CONSTRAINT constraint_name;
```

Example :

```
ALTER TABLE employee  
ENABLE CONSTRAINT employee_pk;
```

Foreign key constraint

- If foreign key is a field which points to the primary key of another table. Means that, the same value in one table must also appear in another table.
- The reference table, means the table which contains primary key is called the parent table and the table with the foreign key is called the child table.
- If foreign key field cannot contain values, which are not present in the parent tables. In other words, it is used to enforce referential integrity between tables in a relational database.
- Primary key field cannot contain null values but foreign key field can contain null values.
- Foreign key constraint can be defined by a CREATE TABLE statement or an ALTER TABLE statement.

A foreign key constraint can refer to the columns of tables in the same database for within the same table.

Syntax :

```
CREATE TABLE table_name
(
    column1 datatype,
    column2 datatype,
    .....
    CONSTRAINT fk_column
    FOREIGN KEY (column1,column2,...)
    REFERENCES parent_table (column1,column2,.....)
);
```

Example

PRIMARY KEY table : "employee" as defined above.

FOREIGN KEY table :

```
CREATE TABLE Department
(
    Eno int,
    Dno int NOT NULL,
    Dname varchar(30),
    PRIMARY KEY (Eno,Dno),
    FOREIGN KEY (Eno)
    REFERENCES employee(EmpID)
);
```

Points :

- The PRIMARY KEY table contains 5 columns- (EmpID, Name, Address, Salary, Sex) , in which EmpID is the Primary Key.
- The other table, means the FOREIGN KEY table contains 3 columns - (Eno, Dno, Dname), in which Eno and Dno combined is the primary key. Eno has been considered as foreign key as it will take the references from employee table (EmpID).
- Foreign Key accept only those values which are present in primary key table or NULL values. So, while inserting any value in Department table, the column Eno would accept only those values which are already in table Employee, EmpID column or it will accept NULL values.

ADD a FOREIGN KEY using ALTER TABLE Statement

ADD a FOREIGN KEY

Syntax :

```
ALTER TABLE table_name  
ADD CONSTRAINT constraint_name  
FOREIGN KEY (col1, col2,...)  
REFERENCES parent_table (col1, col2,...)
```

Example :

```
ALTER TABLE Department  
ADD CONSTRAINT FK_employee  
FOREIGN KEY (Eno)  
REFERENCES employee(EmpID);
```

DELETION of a Row :

We cannot delete a row that contains a primary key that is used as a foreign key in another table. For example :

Query 1 : DELETE FROM employee where EmpID = 1000; ×

Query 2 : DELETE FROM employee; ×

Query 3 : DELETE FROM Department where Eno = 1002; ✓

- The **Query 1 and Query 2** would return an error, because the column EmpID is a Primary Key and it has been referenced by a Foreign Key table (Department).
- But **Query 3** would successfully executed.
- But it is possible with the use of **ON DELETE CASCADE and ON DELETE SET NULL** commands.

ON DELETE CASCADE and ON DELETE SET NULL

- **ON DELETE CASCADE** - delete the dependent rows in the child table, only when a row in the parent table is deleted.
- **ON DELETE SET NULL**- is used when you want to convert dependent foreign key values to null. Without using ON DELETE CASCADE or ON DELETE SET NULL command, you cannot delete the row of parent table, if it is referenced in the child table.

Guidelines for primary keys and foreign keys

- Primary key does not use duplicate values.
- Foreign key is based on the values from the primary key and purely logical and physical pointers.
- The values of foreign key must match the existing primary key value.
- Foreign key must refer either a primary key or unique key column.

Comparison of primary key and foreign key

- Primary key is unique but foreign key need not be unique.
- Primary key is not null and foreign key can be null, foreign key references a primary key in another table.
- Primary key is used to identify a row, where as foreign key refers to a column or combination of columns.
- Primary key is the parent table and foreign key is a child table.

SQL LIKE Operator

SQL LIKE operator is used to compare a value to similar values using wildcard operators. There are two wildcards:

- % (Percent Symbol) - It indicates sequence of 0 or more characters or numbers.
- _ (Underscore Symbol) - It indicates a single character or number.

Points :

- A% : First Letter is 'A' followed by any number of characters.
- %A : Last letter is 'A' preadded by 0 or more characters.
- %A% : Character 'A' may appear anywhere in the word.
- %ANK% : A sequence 'ANK' may appear anywhere in the word.
- A_ _ _ _ : A five letter word starting with 'A'.

Queries on LIKE Operator (using % and _ Wildcards) :

- Find the student details whose name starts with 'S' and from branch CSE.
- SELECT * FROM Student
- WHERE Name LIKE 'S%' AND Branch = 'CSE';
- Find all the student details who opted 'network protocol' as a subject.
- SELECT * FROM Student
- WHERE Subject LIKE '%network protocol%';
- Find the details of student whose 1st letter is 'A' and 4th letter is C and last letter is 'D'.
- SELECT * FROM Student
- WHERE Name LIKE 'A_ _ C%D';
- Find the details of students who are in 'CSE' branch and coming from Hyderabad and whose 3rd letter is 'A'.
- SELECT * FROM Student
- WHERE Name LIKE '_ _ A%' AND City = 'Hyderabad' AND Branch = 'CSE';

Searching Special Character Using Escape Symbol (/) :

The escape Symbol is used when we want to search some special characters like % (percent symbol) or an _ (Underscore Symbol) in the data. For this, we have to put the escape symbol before the special characters. **For Example :** Find the details of the order, whose ordname starts with 'P' and ends with 'S' and contains a '%' symbol?

```
SELECT * FROM Orderdetails
WHERE Ordname LIKE 'P% /% %S';
```

Complete Format of SELECT Command

Complete Format of SELECT Command is

```
SELECT      [DISTINCT] <attribute or function list>
FROM        <table list>
[WHERE      <Condition (s)>]
[GROUP BY   <grouping attributes (s)>]
[HAVING      <group condition>]
[ORDER BY   <attribute list> [DESC/ASC]
```

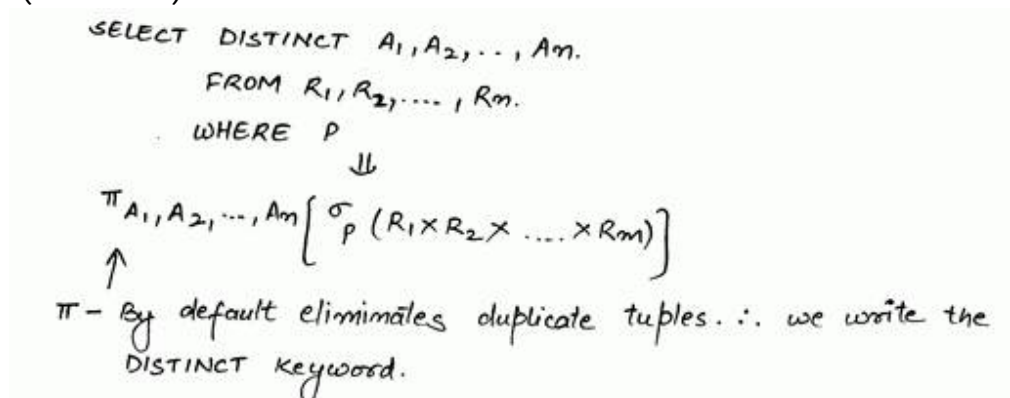
Mandatory Clauses In SQL :

Only the first two clauses SELECT & FROM are mandatory. Other clauses enclosed between square brackets[...] are optional.

Translating SQL into Relational Algebra :

From $\cong \times$ (Cross Product)

- Select $\cong \pi$ (Projection)
- Where $\cong \sigma$ (Condition)



SELECT DISTINCT A_1, A_2, \dots, A_m .
FROM R_1, R_2, \dots, R_m .
WHERE P

\Downarrow

$\pi_{A_1, A_2, \dots, A_m} [\sigma_P (R_1 \times R_2 \times \dots \times R_m)]$

\uparrow
 π - By default eliminates duplicate tuples. \therefore we write the DISTINCT keyword.

- Example :

Order Of Execution in Relational Algebra:

1. \times : Cross Product i.e. FROM
2. σ : Condition i.e. WHERE
3. π : Projection i.e. SELECT

Order Of Execution in SQL:

1. FROM
2. WHERE
3. GROUP BY
4. SELECT

Consider the Following Relation :

Employees				
Eno	Fname	Lname	Address	City
1000	Ankit	Mittal	MC Colony	Bhiwani
1001	Pooja	Garg	JainChowk	Bhiwani
1002	Komal	Gupta	Babra town	Rohtak
1003	Surender	Jain	Model Town	Rohtak

Distinct keyword:

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you may want to listen on a different (distinct) values in a table. The DISTINCT keyword can be used to return only distinct different values.

Example : SELECT DISTINCT City FROM Employees;

Solution : The result will be 'BHIWANI' and 'ROHTAK'.

The WHERE Clause:

The WHERE Clause is used to accept only those records that fulfill the specified criteria.

Example : SELECT Fname FROM Employees WHERE City = 'Bhiwani';

Solution : 'Ankit' and 'Pooja'

Operators Allowed in WHERE Clause

Operator	Description
=	Equal
!=	Not Equal
>	Greater Than
<	Less Than
> =	Greater Than Equal To
< =	Less Than Equal To
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column
AND Operators	Displays record if Both the Conditions are TRUE.
OR Operator	Displays record if Either a Condition is TRUE.

The ORDER BY Clause

The ORDER BY keyword is used to sort the result set along specified column. The result of query can be ordered either in ascending order or in descending order. The ORDER BY keyword sorts the records in ascending order by default. If you want to show the records in a descending order, you can use the DESC keyword. It is applied on Independent columns but not on group of columns.

Example : SELECT Eno FROM Employees
ORDER BY Lname ASC;

Solution : The Result will be 'Garg', 'Gupta', 'Jain', 'Mittal'.

Example of Some Queries that Use ORDER BY Clause

Consider the relation **Student(Rollno, Name, Age, Marks, BranchID, Branch)**.

Query 1 : Show the student details order by name

```
SELECT * FROM Student
ORDER BY Name;
SELECT Rollno, Name, Age FROM Student //Doesn't Produce any error
WHERE age>15 ORDER BY 1,2;           //But applied on Independent
                                     columns but not the group
SELECT Rollno, Name, Age FROM STUDENT //Doesn't Produce any error
where Age>10                          //But applied on Independent
ORDER BY Rolno DESC, Name ASC.         columns but not on the group
```

Query 2: Find the maximum marks of students having rollno 101,110,115.

```
SELECT Rollno, MAX(Marks) FROM Student  
GROUP BY Rollno  
HAVING Rollno IN (101,110,115);
```

Query 3: Find the total number of students in a branch order by their Roll Numbers?

The following Query is written in Wrong Form :

```
SELECT BranchID, COUNT(Rollno) FROM Student ×  
GROUP BY BranchID  
ORDER BY COUNT(Rollno)
```

The Correct Form of the Above Query is :

```
SELECT BranchID, COUNT(Rollno) a FROM Student ✓  
GROUP BY BranchID  
ORDER BY a;
```

Result : We cannot use aggregate functions in ORDER BY Clause.

Query 4 : Find total strength of each branch and display the details of branches where the total number of students is more than 45 and display them in ascending order.

Wrong Query :

```
SELECT BranchID, COUNT(Rollno) a FROM STUDENT ×  
GROUP BY BranchID  
HAVING a>45  
ORDER BY a;
```

Correct Query :

```
SELECT BranchID, COUNT(Rollno) a FROM STUDENT ✓  
GROUP BY BranchID  
HAVING COUNT(Rollno)>45  
ORDER BY a;
```

Result : The main motive of this query is to tell you that we cannot use aliases in HAVING Clause.

The GROUP BY Statement

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns. It allow one to partition the result into a number of groups that hold rows of a group have the same value in some specified column. Whenever group by is used the phrase where is to be replaced by having. The meaning of having the same as where except that the condition is new applicable to each group.

Example : Consider the following Orders table :

Orders			
OID	OrderDate	OrderPrice	Customer
1	2015/08/24	1000	Ankit
2	2015/07/25	1600	Pooja
3	2015/06/24	700	Ankit
4	2015/05/24	300	Ankit
5	2015/12/24	2000	Surender
6	2015/7/24	100	Pooja

Query : SELECT Customer, SUM(OrderPrice) FROM Orders GROUP BY Customer ;

Solution :

Customer	OrderPrice
Ankit	2000
Pooja	1700
Surender	2000

Total number of groups = 3

Some Examples of Queries in which GROUP BY Clause is used :

Consider the relation **Student(Rollno, Name, Age, Marks,BranchID,Branch)**.

- **Query 1** : Find the student details who got second maximum or third maximum or in general n^{th} maximum marks ?
- **Query 2** :Find all the total number of students in each and every branch?
- SELECT BranchID, COUNT(Rollno) FROM Student
- GROUP BY BranchID;

The HAVING clause

- The HAVING clause was added to SQL because the where keyword could not be used with aggregate functions.
- It exists only with GROUP Clause.

Example 1: Consider the table Orders(oid, OrderDate, OrderPrice, Customer)

```
Query : SELECT Customer, SUM (OrderPrice)
        From Orders
        GROUP BY Customer
        HAVING SUM(OrderPrice) < 2000;
```

Solution : The result will be : 'Pooja' and 2000.

Example 2: Retrieve students whose branch average marks is greater than equal to 65.

```
Query : SELECT *
        FROM Student
        GROUP BY (Branch)
        HAVING AVG(Marks) > 65;
```

Solution : It calculates average of the groups and then prints all the attributes of the table for the group whose AVG(Marks) \geq 65.

Restriction for GROUP BY :

If SELECT list consists of non-aggregate column with an aggregate function, then the query associated with the GROUP BY clause must consists of non-aggregate columns in the group by list.

For example :

Wrong Form of Query :

```
SELECT Age,BranchID, COUNT(Rollno) FROM Student
GROUP BY COUNT(Rollno);
```

Wrong Form of Query :

```
SELECT Age,BranchID, COUNT(Rollno) a FROM Student
GROUP BY a;
```

Correct Form of Query :

```
SELECT Age,BranchID, COUNT(Rollno) a FROM Student
GROUP BY BranchID;
```

Whenever we apply GROUP BY, aggregation operator must be there.

We cannot use aggregate function in HAVING Clause rather than GROUP BY clause.

Example : Find out the branch detail whose total strength is more than 50 students?

Query : SELECT BranchID, COUNT(Rollno) FROM Student ×
GROUP BY BranchID
HAVING COUNT(Rollno)>50.

This Query is WRONG due to an aggregation operator in SELECT clause [COUNT(Rollno)].

Group by is applied on Independent rows. Having is applied among groups.

Example :

Query : Find the details of student who paid more than 20 Rupees
as a total fine?

Solution : SELECT Rollno, SUM(Fine) FROM Library
GROUP BY Rollno
HAVING SUM(Fine) > 20;

We can use WHERE with GROUP BY as well.

```
SELECT Branch, MAX(Marks)
FROM Student
WHERE marks>50
GROUP BY(BRANCH);
```

What the query says is :

On the result of WHERE Clause, if any grouping is possible, then it is performed.

Difference Between WHERE Clause and HAVING clause

WHERE Clause	HAVING Clause
It is used to filter rows i.e. at tuple level.	It is used to filter groups i.e. checks a condition for a Group.
Aggregate functions cannot be used.	Aggregate functions can be used.
It is mandatory i.e. There is no alternative for WHERE Clause.	It is optional i.e. the results given by HAVING Clause can also be applied without it.

DML Commands : : Data Manipulation Language Commands

There are four DML Commands :

- The INSERT INTO Command
- The UPDATE Command
- The DELETE Command
- The SELECT Command

The INSERT INTO Command

The INSERT INTO command is to insert a new row in a table.

Syntax :

There are two ways to write the INSERT INTO Commands -

Form 1 : (Mainly used if we want to insert data in all columns)

INSERT INTO table_name
VALUES (value1, value2, value3,...);

Example :

Let the Initial table will be :

Emp_ID	Emp_First_Name	Emp_Last_Name	Address	Salary
1000	Renu	Garg	Bhiwani	31000.00
1001	Pooja	Garg	Bhiwani	32000.00
1002	Komal	Mittal	Rohtak	35000.00
(Table 1)				

INSERT INTO Employee
VALUES (1003,'Surender','Mittal','BHIWANI',50000.00);

Form 2: (Mainly used if we want to insert data in specified columns)

INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...);

Example of Form 2 :

INSERT INTO Employee VALUES (Emp_ID,Emp_First_Name,Emp_Last_Name)
VALUES (1004,'Renu','Mittal');

The Updated table after Insertion Operation :

Emp_ID	Emp_First_Name	Emp_Last_Name	Address	Salary
1000	Renu	Garg	Bhiwani	31000.00
1001	Pooja	Garg	Bhiwani	32000.00
1002	Komal	Mittal	Rohtak	35000.00
1003	Surender	Mittal	Bhiwani	50000.00
1004	Renu	Mittal	NULL	NULL
(Table 2)				

The UPDATE Command

The Update Command is used to modify data values within one or more columns for one or more rows of a table.

Syntax :

```
UPDATE table_name
SET <column_name1> = <value expression1>
  [<column_name2> = <value expression2>
   .....
  ]
[ WHERE <condition>];
```

The columns whose values have to be updated and the expressions to derive these values are included in the SET clause. The WHERE Clause - The WHERE Clause is used to extract only those records that fulfill a specified criteria.

Example of UPDATE Command :

```
UPDATE EMPLOYEE SET Salary = 60000.00
WHERE Emp_ID = 1003;
```

The command will modify the salary of the employee with 60000.00 whose ID is 1004.

SQL UPDATE Warning :

Be careful after removing WHERE Clause during updating records Because it will update all records. For Example :

```
UPDATE Employee
SET ADDRESS = 'ROHTAK', Emp_Last_Name = 'Garg'
```

So, the table data after applying update operation will be as :

Emp_ID	Emp_First_Name	Emp_Last_Name	Address	Salary
1000	Renu	Garg	Rohtak	31000.00
1001	Pooja	Garg	Rohtak	32000.00
1002	Komal	Garg	Rohtak	35000.00
1003	Surender	Garg	Rohtak	60000.00
1004	Renu	Garg	Rohtak	NULL
(Table 3)				

The SELECT Command

The SELECT Command is used to select data from a database. The result is stored in a result table called Result-Set. The SELECT can be used for simple to extremely complex queries. It comes under the Data Query Language because it may also be used in <query specifications> as part of other commands such as INSERT or CREATE VIEW.

Syntax For selecting some specified Columns :

```
SELECT column_name(s)
FROM table_name;
```

Example :

```
SELECT Emp_ID, Emp_First_Name, Salary
FROM Employee;
```

The result will be :

Emp_ID	Emp_First_Name	Salary
1000	Renu	31000.00
1001	Pooja	32000.00
1002	Komal	35000.00
1003	Surender	50000.00
1004	Renu	NULL
(Table 4)		

Syntax for selecting all the rows without specifying column names :

```
SELECT *
FROM table_name;
```

Example :

```
SELECT *
FROM Employee;
```

The result will be (Table 3).

Syntax for selecting specific rows which meet some condition(s) :

```
SELECT column_name(s)
FROM table_name
WHERE <condition(s)>;
```

Example :

```
SELECT Emp_ID, Emp_First_Name, Salary
FROM Employee
WHERE Emp_ID = 1002;
```

The Result will be as :

Emp_ID	Emp_First_Name	Emp_Last_Name	Address	Salary
1002	Komal	Garg	Rohtak	35000.00
(Table 5)				

The DELETE Command

The Delete Command is used to delete rows in a table.

Syntax :

```
DELETE FROM table_name
[WHERE <condition(s)>]
```

Example :

```
DELETE FROM Employee
where Emp_ID = 1003;
```

The Result will be after delete operation as :

Emp_ID	Emp_First_Name	Emp_Last_Name	Address	Salary
1000	Renu	Garg	Rohtak	31000.00
1001	Pooja	Garg	Rohtak	32000.00
1002	Komal	Garg	Rohtak	35000.00
1004	Renu	Garg	Rohtak	NULL
(Table 6)				

DELETE All Rows :

It is possible to delete all rows in a table without affecting the schema of the database or say the table structure, attributes and indexes. Omit the WHERE Clause from the DELETE Command. For Example :

DELETE FROM Employee;
All the table data will be deleted.

Operators in SQL with Example

Various Types of Operators in SQL are : **Arithmetic Operators :**

+ (ADDITION) **-** (DIFFERENCE) ***** (MULTIPLY) **/** (DIVISION)

Relational Operators :

= (EQUAL TO) **<** (LESS THAN) **>** (GREATER THAN)

<> (NOT EQUAL TO) **<=** (LESS THAN EQUAL TO) **>=** (GREATER THAN EQUAL TO)

Conjunction Operators : Conjunction Operators are used to combine the conditions. Operators are :

AND Operator OR Operator

Logical Operators : Logical Operators in SQL are :

Operators	Meaning
Unique	Used to search every row of a specified table for uniqueness (no duplicate)
All	Used to compare a value to all values in another value set.
Any	Used to compare a value to any applicable value in the list according to the condition
Is Null	Used to compare a value with a NULL value
Between	Used to search for values that are within a set of value, given the minimum value and maximum value.
In	Used to compare a value to a list of literals values that have been specified.
Like	Used to compare a value to similar values using wildcard operators. There are two wildcards: (%) - It(percent symbol) indicates sequence of 0 or more characters or numbers. (_) - It(underscore) indicates a single number or character
Exist	Used to search for the presence of a row in a specified table that meets certain criteria.

Negating Conditions with the NOT operator : The NOT operator reverses the meaning of the logical operators with which it is used. For example,

- [NOT] IN
- [NOT] EXIST
- IS [NOT] NULL
- [NOT] BETWEEN
- [NOT] LIKE

The operator enclosed in square brackets [..] is optional.

Aggregate Functions or Summary Functions

Aggregate function is used to provide summarisation information for SQL statement, count , total and averages. Some aggregate functions are:

Function	Meaning
Count ([DISTINCT] attribute)	Used to count rows or values of a column that do not contain null value.
Count (*)	Used to count all the rows of a table including duplicates.
Sum ([DISTINCT] attribute)	Used to return the total awesome on the values of a numeric column for a group of rows.
AVG ([DISTINCT] attribute)	Used to find average is for a group of rows.
Max (attribute)	Used to return the maximum value for the values of a column in a group of rows.
Min (attribute)	Used to return the minimum value for the values of a column in a group of rows.

Null Values are always discarded from aggregation Operators.

Set Operators in SQL

- UNION
- INTERSECT
- MINUS
- UNION ALL
- INTERSECT ALL
- MINUS ALL

Example :

Consider the two relation R(A) and S(A) such that

R	S
A	A
1	1
1	1
2	1
2	2
3	3
	4

R UNION S: Discards Duplicate

Query:-
 SELECT A FROM R
 UNION
 SELECT A FROM S
 ⇒

RESULT:-
1
2
3
4

R UNION ALL S: Do not discard Duplicates

Query:-
 SELECT A FROM R
 UNION ALL
 SELECT A FROM S

Result:-
1
1
2
2
3
1
1
2
3
4

R INTERSECT S: Discards Duplicates from both relations

Query:-
 SELECT A FROM R
 INTERSECT
 SELECT A FROM S

Result:-
1
2
3

R INTERSECT ALL S: Does not Discard Duplicates

Query:-
 SELECT A FROM R
 INTERSECT ALL
 SELECT A FROM S

Result:-
1
1
2
3

R Minus S: Discards Duplicates

$$R - S = \emptyset$$

R MINUS ALL S: Does Not Discard Duplicates

R	S	R-S
A	A	A
1	1	2
1	1	5
1	1	
2	2	
3	3	
5	4	

CONTAIN Operator

The CONTAIN Operator is similar to Division Operator in Relational Algebra.

$$\frac{A(x,y)}{B(y)} = A(x,y) \text{ CONTAIN } B(y)$$

NULL values :

NULL is the set of some randomly generated ASCII value, generated by DBMS engine.

- No two NULL values are equal.
- NULL value is non zero.

NULL VALUE → value don't exists
→ value exists but not known

Empid	Empname	DOB	Spouse
1	A	NULL	Y
2	B	15/7/90	NULL

→ value not known
→ may be unknown
→ may not exist

More About Operators :

The BETWEEN and NOT BETWEEN Operator

The BETWEEN and NOT BETWEEN operators are used in range queries. The BETWEEN operator select a range of data between two values. The values can be numbers, text or dates.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2;
```

The NOT BETWEEN operator select the data outside the range of data between two values. The values can be text, numbers or date.

Syntax:

```
SELECT column_name(s) FROM table_name WHERE column_name NOT BETWEEN value1 AND value2;
```

Combining AND & OR Operator

We can also combine and and or useful and used parenthesis to form complex expressions.

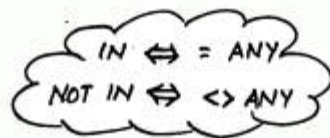
Example :

```
SELECT * FROM Student
WHERE Age=19
AND (Name = 'Ankit' OR Marks = 458)
```

All and any

- If subquery is prefixed with 'ANY' keyword, then the condition will be true, if it is satisfied by any of the values produced by inner query.
- If subquery is prefixed with 'ALL' keyword, then the condition is true, if it is satisfied by all the values produced by inner query.

Relation Between IN and ANY Operator :-



Example 1 :

For Example:-

Consider a set {2,6,8,10,12}

$x >=$ ANY {2,6,8,10,12}

$x=1$: False

$x=5$: True

Condition should satisfy if x is greater than any value from the set

$x >=$ ALL {2,6,8,10,12}

$x=10$: FALSE

$x=51$: TRUE

Condition should satisfy if x is greater than all the values from the set

Difference between $>=$ & $>=$ ANY

$>=$: Compare 2 values

$>=$ ANY: Compares a value with set of values.

Example 2:

Consider the relation Employee(Eno,Ename,Sal,Dno)

Eno	Ename	Sal	Dno
E1	-	5	4
E2	-	10	6
E3	-	15	5
E4	-	20	6
E5	-	25	5

Question : Retrieve employees whose salary is greater than salaries of any employee in department 5.

Query : SELECT Eno FROM Employee
WHERE sal > ANY (SELECT Sal FROM Employee
WHERE Dno=5)

Solution : >ANY(15,25) \Rightarrow E4

The EXIST and NOT EXIST Operator

- If the subquery is prefixed with 'EXIST' operator, then the condition is treated as true, if inner query returns non empty rows.
- If the inner query is prefixed with 'NOT EXIST', then the condition is treated as true, if inner query returns empty rows.
- Always EXIST and NOT EXIST should have SELECT command.

Difference Between EXIST & ALL and ANY

EXIST	ALL and ANY
Any query formulated in ALL and ANY can also be formulated in EXIST.	Any query formulated in EXIST may not be formulated in ALL and ANY.
Exist queries always leads to correlated subquery.	They need not be correlated query
They are less efficient.	They are more efficient.

DCL Commands : Data Control Language Commands

DCL Commands are used for controlling the data. The DCL Commands are -

1. COMMIT Command
2. ROLLBACK Command
3. GRANT Command
4. REVOKE Command

The COMMIT Command

The COMMIT Command is a transactional command which is used to save changes invoked by a transaction to the database. It saves all the transaction to the database since the last COMMIT or ROLLBACK.

Syntax

COMMIT [WORK];

where COMMIT is the keyword and the WORK is optional keyword and it is used to make the command more user friendly.

Example :

```
DELETE FROM Employee WHERE SALARY > 40000.00;  
COMMIT;
```

The first statement deletes the records of the Employee whose salary > 40000.00, a COMMIT command is issued to save the changes to the database, completing the transaction.

The ROLLBACK Command

It is used to undo transactions until the last COMMIT or ROLLBACK command was issued or say that have not already been saved to the database.

Syntax :

ROLLBACK [WORK];

Example :

```
DELETE FROM Employee WHERE Salary > 40000.00;  
ROLLBACK;
```

First statement delete the records of all the employees whose salary is greater than 40000.00. But the second statement remove the effect of first one and rollback/undo the changes made by it. Hence, there will be no effect on Employee table.

The GRANT Command

As a system would have several users, it is necessary to define what may be accessed by each user. Granting of privileges is done with the GRANT command.

Syntax :

```
GRANT <privileges> ON <table/view_name>  
TO <user_name_list>/PUBLIC
```

This SQL Command allows the specified privileges (such as SELECT/INSERT,UPDATE,DELETE on some or all columns or ALL PRIVILEGES) on the specified table/view to the users as specified or to all users (PUBLIC).

The REVOKE Command

The REVOKE Command withdraws granted privileges and is of similar format as GRANT.

Joins in SQL

A Join is a query that combines rows from two or more tables. In a join query, more than one table are listed in FROM clause. The function of combining data from multiple tables is called joining. SQL can obtain data from several related tables by performing either a physical or virtual join on the tables. The WHERE Clause is most often used to perform the JOIN function with two or more tables have columns.

Types of SQL Joins :

There are different kinds of SQL joins :

- Equi join
 - Inner Join
 - Outer Join
 - Left Outer Join
 - Right Outer Join
- Self join
- Non equi join
- Natural join

Joins are used when we have to select data from two or more tables. Joins are used to extract data from two (or more) tables, when we need a relationship between certain columns in these tables. The SQL join condition is always used in the WHERE clause of SELECT, UPDATE and DELETE statements.

Equi join :

Equi join is a simple SQL join condition that uses equals sign as a comparison operator.

Syntax :

```
SELECT col1,col2,col3
FROM table1,table2
WHERE table1.col1 = table2.col2;
```

Cartesian product :

The Cartesian product is also referred as **cross-join**. It is a binary operation and is denoted by X. The degree of the new relation is the sum of the degrees of two relations on which Cartesian product is operated. The number of the tuples of the new relation is equal to the product of the number of tuples, of the two relations on which Cartesian product is performed.

Example 1:

Let $A = \{1,2\}$ and $B = \{a,b,c\}$, Then, $A \times B = \{(1,a),(1,b),(1,c),(2,a),(2,b),(2,c)\} \Rightarrow$ Elements of $A(3) * \text{Elements of } B(2) = \text{Elements of } [A \times B] (6)$

Example 2:

Consider two relations Person and Order such that

PERSON				ORDERS		
P_ID	Last_Name	First_Name	City	O_id	Order_No	P_id
1	Sharma	Abhay	Mumbai	1	10050	3
2	Gupta	Mohan	Delhi	2	25000	3
3	Verma	Akhil	Mumbai	3	5687	1
				4	45000	1
				5	35000	15

```
Query : SELECT Last_Name, First_Name, Order_No
        FROM Person,Orders
        WHERE Person.P_ID = Orders.P_ID
        ORDER BY Person.Last_Name;
```

Solution :

Last_Name	First_Name	Order_No
Sharma	Abhay	5687
Sharma	Abhay	45000
Verma	Akhil	10050
Verma	Akhil	25000

Points

- A Cartesian product is formed when join conditions are omitted and invalid.
- All rows in the first table are joined to all rows in the second table.
- If you want to avoid Cartesian product, then we should use a join condition in WHERE Clause.
- Using Cartesian product, it gives us a large number of rows and is rarely useful.

Types of Equi-Joins

SQL Equi-Joins are further classified into two categories

- Inner join
- Outer join

Inner join

The Inner joins returns us the rows which are common in both the tables. i.e. gives the intersection of two tables.

Syntax :

```
SELECT col1,col2
FROM table1 INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Outer join

The outer join include rows in a joint result even when they have no match in the joint table. And Outer join Returns all rows that must satisfy the join condition and also returns those rows from one table for which no rows from the other satisfy the join condition.

Left outer join:

The left outer join returns all the rows from the left table, even if there are no matches in the right table.

Example :

```
SELECT P.Lname, P.Fname, O.Orderno
FROM PERSON P
LEFT JOIN Orders O
ON P.PID = O.PID
ORDER BY P.Fname
```

Solution :

Last_Name	First_Name	Order_No
Sharma	Abhay	5687
Sharma	Abhay	45000
Verma	Akhil	10050
Verma	Akhil	25000
Gupta	Mohan	NULL

Right outer join:

The right outer join return all rows from the right table, even if there are no matches in the left table.

Syntax :

```
SELECT col1,col2
FROM table1
RIGHT JOIN table2
ON table1.col_name=table2.col_name;
```

Example :

```
SELECT P.Lname, P.Fname, O.Orderno
FROM Person P
RIGHT JOIN Order O
ON P.PID = O.PID
ORDER BY P.Lname;
```

Result :

Last_Name	First_Name	Order_No
Sharma	Abhay	5687
Sharma	Abhay	45000
Verma	Akhil	10050
Verma	Akhil	25000
NULL	NULL	35000

Self join

A self join is a join in which the table is joined with itself to get the appropriate results. In this case, it is necessary to ensure that the join statement defines an ALIAS name for both the of copies of the tables to avoid column ambiguity.

Example :

Consider the relation Course as :

TABLE COURSE		
Course_id	Course_Name	Pre_Course
1	C	NULL
2	C++	1
3	Java	2
4	C#	3
5	VB.NET	4

Query : SELECT A.Coursename AS Course,
 B.Coursename AS Prerequisite_Course
FROM Course A, Course B
WHERE A.Precourse = B.CourseID;

Solution :

Course	Prerequisite_Course
C++	C
Java	C++
C#	Java
VB.NET	Java

Non Equi Join

Non Equi-Join is used to return the result from two or more tables where exact join is not possible. The SQL Non Equi-Join uses comparison operator instead of the equal sign like (>,<,>=,<=) along with conditions.

Syntax :

```
SELECT * FROM table1,table2  
WHERE table1.column > table2.column;
```

Example :

```
SELECT E.Empno, E.Ename, E.Salary, S.Grade  
FROM Emp E, SalaryGrade S  
WHERE E.Salary BETWEEN S.LowSalary AND S.HighSalary;
```

Natural join

The natural join is a type of Equi Join and is structured in such a way that, columns with same name of associated tables will appear once only.

Syntax :

```
SELECT * FROM table1  
NATURAL JOIN table2;
```

Example :

```
SELECT * FROM Person  
NATURAL JOIN Orders;  
Solution :
```

P_ID	Last_Name	First_Name	City	O_ID	Order_No
1	Sharma	Abhay	Mumbai	3	5687
1	Sharma	Abhay	Mumbai	4	45000
3	Verma	Akhil	Mumbai	1	10050
3	Verma	Akhil	Mumbai	2	25000

Union

The union is used to return all the distinct rows selected by either query.

Example :

```
(SELECT Sname FROM Student  
WHERE Stream = 'Science') UNION  
(SELECT Sname FROM Student  
WHERE Stream = 'Arts');
```


Summarization of Joins in SQL : Example :

M

PID	EName
1	A
2	B
3	C

N

OID	PID	PName
1	3	P
2	3	Q
3	1	R
4	5	S

① Equi Join (= Sign)

```

Select *
From M, N
Where M.PID = N.PID
    
```

PID	FName	OID	PID	OName
1	A	3	1	R
3	C	1	3	P
3	C	2	3	Q

①.1 INNER JOIN

```

SELECT PID, FName, OID
From INNER JOIN M
ON M.PID = N.PID
    
```

①.2 Outer Join- Left outer Join

```

SELECT FName, OID
FROM M LEFT JOIN N
ON M.PID = N.PID
    
```

FName	OID
A	NULL
B	NULL
C	1
C	2

①.2.2 Outer Join- Right outer Join

```

SELECT FName, OID
FROM M RIGHT JOIN N
ON M.PID = N.PID
    
```

FName	OID
C	1
C	2
A	3
NULL	4

② SELF JOIN

```

SELECT M1.OName AS OName1,
       M2.OName AS OName2
FROM M M1, M M2
WHERE M1.OID = M2.PID
    
```

OName1	OName2
P	R
R	P
R	Q

③ Non Equi Join

```

SELECT M.PID, OName
FROM M, N
WHERE M.PID BETWEEN 3 AND 4
    
```

M.PID	OName
3	P
3	Q
3	R
3	S

④ Natural Join

```

SELECT * FROM M
NATURAL JOIN N
    
```

PID	FName	OID	OName
1	A	3	R
3	C	1	P
3	C	2	Q

Queries on SQL Operators

QUERIES :

Example of Some Queries related to aggregate functions

Consider the relation **Student(Rollno, Name, Age, Marks1, Marks2, Marks)**. Following Results are obtained from the Queries below :

Result 1: MAX and MIN marks can be used on both numeric and non numeric columns. For Example,

SELECT MAX(Marks) FROM Student; ✓

SELECT MAX(Name) FROM Student; ✓

Result 2: SUM and AVG must be applied only on numeric queries. For Example,

SELECT SUM(Name) FROM Student; ×

SELECT SUM(Marks) FROM Student; ✓

Result 3: DISTINCT has no effect with MIN and MAX but will have a impact on SOME and AVG. For Example,

SELECT SOME(DISTINCT Marks) FROM Student; ✓

SELECT MIN(DISTINCT Name) FROM Student; ✓

Query : Find Out Which Query is Wrong ?

a) SELECT COUNT(Rollno) FROM Student; ✓

b) SELECT SOME(Marks*10) FROM Student; ✓

c) SELECT AVG(Mark1 + mark2) FROM Student; ✓

d) SELECT AVG(Mak1, mark2) FROM Student; ×

The Query (d) is wrong. **Reason :** Aggregate functions operate on a single column of a table or on expressions and returns a single value, but it cannot work on multiple columns simultaneously.

Result 4: Aggregate functions can be used only in the SELECT and HAVING clauses but not in WHERE, ORDER BY and GROUP BY clauses. For Example, The following queries will give you error :

SELECT * FROM Student where MAX(Marks)=500; ×

SELECT * FROM Student ORDER BY MAX(Marks) ×

SELECT * FROM Student GROUP BY COUNT(Rollno); ×

Result 5: If SELECT clause consists of some aggregate functions along with ordinary columns then it must be associated with GROUP BY clause, Otherwise, it will produce incorrect results.

a) SELECT Rollno, MAX(Marks) FROM Student; × // Produces Incorrect Result

SID	Count (Marks)
S1	4
S2	
S3	
S4	

← So, the query is syntactically wrong.

b) SELECT Rollno, MAX(Marks) FROM Student GROUP BY Rollno; ✓

In Example a), in order to get Rollno of Student having Max Marks, we should put Rollno inside GROUP BY.

Point : If GROUP BY exists, then Selection and Aggregation is for every group, not for entire table.

Some Range Queries in which BETWEEN and NOT BETWEEN Operator is used :

Consider the relation **Student(Rollno, Name, Age, Marks)**. ×✓

Query 1: Find the details of students whose marks lies between 500 and 600.

SELECT * FROM Student
WHERE Marks BETWEEN 500 AND 600; ✓

Query 2: Find the details of student whose marks does not lie between 350 and 400.

```
SELECT * FROM Student  
WHERE Marks NOT BETWEEN 350 and 400; ✓
```

Result 1: While executing the BETWEEN query with characters, it will take lower boundary and stops exactly at the upper boundary. For Example,

```
SELECT * FROM Student  
WHERE Name BETWEEN 'A' and 'K'; ✓
```

If any name consists only a single letter K, then that will be included in the BETWEEN case.

Result 2: The following query is in the **wrong form** :

```
SELECT * FROM Student WHERE Marks BETWEEN 400 and 350; ×
```

SQL will give you a syntax error because lower bound must be mentioned first in between query.

Some Examples of NULL Queries

Query 1 : Find the details of students who are free from Fine ?

```
SELECT * FROM Library  
WHERE Fine = NULL; ×
```

We cannot use = operator because value of NULL is random therefore, we can't put Fine = NULL.

```
SELECT * FROM Library  
WHERE Fine IS NULL; ✓
```

Query 2 : Find the details of all the students who are having e-mail addresses.

```
SELECT * FROM Students  
WHERE email IS NOT NULL;
```

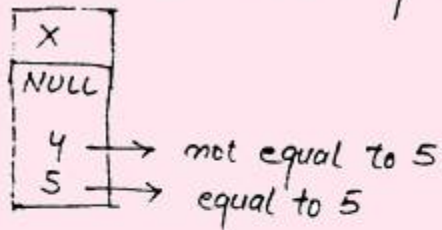
Question 3 : In SQL, relations can contain NULL values, and comparisons with null values are treated as unknown. Suppose all comparisons with a null value are treated as false. Which of the following pairs is not equivalent?

- a) $x=5$ $\text{not}(\text{not}(x=5))$
- b) $x=5$ $x>4$ and $x<6$ where x is an integer
- c) $x\neq 5$ $\text{not}(x=5)$
- d) None of the Above

[GATE 2000]

Solution :

Consider the values of x



a) $x=5$ $\text{not}(\text{not}(x=5))$

$x=5$
 $\text{NULL}=5$
 ~~$\text{NULL}=4$~~
 $4=5$
 $5=5$

$$\begin{array}{c} \text{not}(\text{not}(x=5)) \\ \text{not}(\text{not}(\text{NULL}=5)) \\ \underline{\text{F}} \\ \text{T} \\ \underline{\text{F}} \end{array}$$

$$\begin{array}{c} \text{not}(\text{not}(4=5)) \\ \underline{\text{F}} \\ \text{T} \\ \underline{\text{F}} \end{array}$$

$$\begin{array}{c} \text{not}(\text{not}(5=5)) \\ \underline{\text{T}} \\ \text{F} \\ \underline{\text{T}} \end{array}$$

b) $x > 4$ & $x < 6$ where x is an integer

$$\begin{array}{c} \text{NULL} > 4 \quad \& \quad \text{NULL} < 6 \\ \underline{\text{F}} \end{array} \Rightarrow \text{F}$$

$$\begin{array}{c} 4 > 4 \quad \& \quad 4 < 6 \\ \underline{\text{F}} \end{array} \Rightarrow \text{F}$$

$$5 > 5 \quad \& \quad 5 < 6$$

c) $x \neq 5$
 $\text{NULL} \neq 5 \Rightarrow \text{F}$
 $4 \neq 5 \Rightarrow \text{T}$
 $5 \neq 5 \Rightarrow \text{F}$

$$\begin{array}{c} \text{not}(x=5) \\ \text{not}(\text{NULL}=5) \\ \underline{\text{F}} \\ \text{T} \\ \text{not}(4=5) \\ \underline{\text{F}} \\ \text{T} \\ \text{not}(5=5) \\ \underline{\text{T}} \\ \text{F} \end{array}$$

Some Queries Related to EXIST and NOT EXIST :

Consider the relation Customer(Cno, Cname,) and Order(Cno,Ano). **Query** : Retrieve all customers Name, where customer places an order through Agent A5 ? **Solution** :

1. SELECT Cname FROM Customers
WHERE Cno
IN(SELECT Cno FROM Order
WHERE Ano = 'A5');
OR
2. SELECT Cname FROM Customers
WHERE Cno
ANY (SELECT Cno FROM Order
WHERE Ano = 'A5');
OR
3. SELECT C.Cname FROM Customer C
WHERE EXIST(SELECT * FROM Order O
WHERE C.Cno = O.Cno AND O.Ano = 'A5');
OR
4. SELECT C.Cname FROM Customer C, ORDER O
WHERE C.Cno = O.Cno AND O.Ano = 'A5' ;

The CREATE VIEW Command

Views are the logical Windows for users to see the results of operations on one or more base tables or views. A view is a virtual table, just like a table which is in a 2 dimensional structure of rows and columns. These are derived from stored data structures and computed again for each query referring to them. Views are created using the CREATE VIEW command. The format of the command is

Syntax :

```
CREATE VIEW <view_name>  
[<view column names>]  
AS <query specification>;
```

The view column names are optional. The <query specification> is the SELECT for selecting the rows and columns which will constitute the view. Views are typically used in queries, and are required to provide restricted access, or more relevant window to the user wanting to see the data.

Example :

```
CREATE VIEW DPT100  
(E_Code, Ename, Basic)  
AS SELECT E_Code, Ename, Basic  
FROM Employee  
WHERE Basic > 5000;
```

The view DPT100 will include only the employee having basic greater than 5000 in employee table.

Table ALIASES

Alias means another name. The table alias means to rename a table in a particular SQL statement. The renaming is a temporary change and the actual table name does not change in the database. **Query :** Get the employees name who have some basic pay.

```
SELECT Ename FROM emp First, Emp Second  
WHERE First.Basic = Second.Basic  
AND First.Eno < > Second.Eno;
```

Renaming Using AS :

We can rename attribute and also the table using AS operator. Consider the same example,

```
SELECT Ename AS Employee_Name FROM emp AS First, Emp AS Second  
WHERE First.Basic = Second.Basic  
AND First.Eno < > Second.Eno;
```

SQL Comments

A Comment is the text which appear within the SQL statement such that they do not affect to the SQL statement in execution. A comment can be written in two ways :

- Begin the comment with /* and ended with */.
- Begins with (2 hyphens) and proceed with the text of the comment.

DESCRIBE Command

Describe an Oracle table, view, synonym, package or function.

Syntax :

```
DESCRIBE table_name  
DESC table_name;
```

CLEAR SCREEN

Clear screen is used to clear the screen at SQL* PLUS prompt.

Syntax:

CLEAR SCREEN;

Query and Subquery in SQL

SQL Queries :

A query is an inquiry into the database using the select command. Consider the following relations

Employee			
Eno	Name	Age	Salary
1000	Ankit	24	30000
1001	Pooja	24	20000
1002	Komal	19	10000
1003	Surender	49	40000

Work_IN	
Eno	Dno
1000	10
1001	10
1002	11
1003	10

Department		
Dno	Dname	City
10	Landline	Bhiwani
11	Phone	Rohtak

Example of SQL Queries :

Query 1 : Get full details of all employees.

Solution : SELECT * FROM Employee;

Query 2 : Get the age of all the employees.

Solution : SELECT Age FROM Employee;

Query 3 : Get the age of all the employees with no duplicates.

Solution : SELECT DISTINCT Age FROM Employee;

Query 4 : Get the employee number and name of the employees whose Salary is greater than 25000.

Solution : SELECT Eno, Name FROM Employee WHERE Salary>25000;

Division of SQL Queries :

Based on the filtering Activity, the SQL queries are divided into 5 groups :

- Queries that uses basic search operations.
- Queries that uses range operations. (Using BETWEEN & NOT BETWEEN operator)
- Queries that uses composition operations. (Using AND & OR operator)
- Queries that uses pattern matching. (Using %, _, and /(escape symbol))
- Query that uses null and not null.

Sub-Queries or Nested Queries :

A subquery is a query within another query i. e. A query embedded in another query is called a subquery. It is also known as nested query. Subquery is used to return data that will be used in the main query as a condition, to further restrict the data to be retrieved. Subqueries are of 2 types :

- Independent Nested Query : Inner query doesn't depend on outer query.
- Co-Related Nested Query : Inner query uses attributes of outer query, hence dependent on outer query.

Examples of SubQueries :

Subquery 1: Find names of employees who work in Department number 11.

Solution : SELECT Name FROM Employee WHERE
Employee.Eno = (SELECT Work_IN.Eno FROM Work_IN
WHERE Dno =11);

Subquery 2: Find name of employee whose basic pay is Greater then all basic pay of the employees working in Department number 10.

Solution : SELECT Name FROM Employee WHERE
Salary > ALL(SELECT Salary FROM Employee WHERE Employee.Eno =
(SELECT Work_IN.Eno FROM Work_IN WHERE Dno = 10))

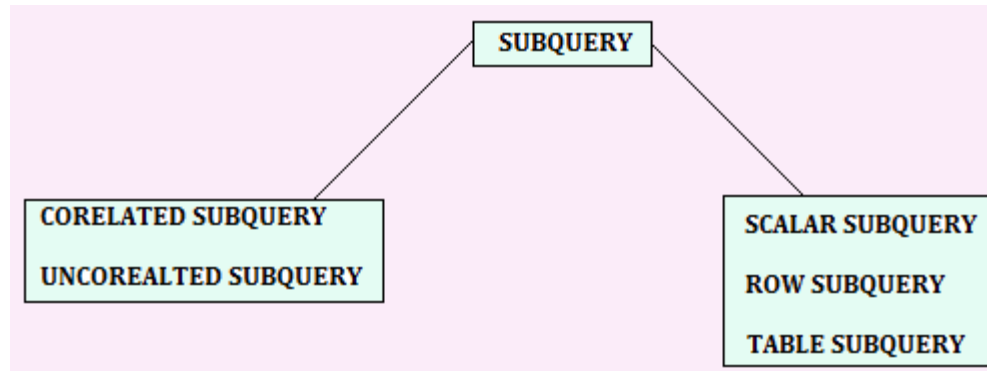
Subquery 3 : Find the names of employees whose basic pay is greater than the average basic pay.

Solution : SELECT Name From Employee WHERE Salary > (SELECT AVG(Salary) FROM Employee);

Characteristics of subquery:

- The inquiry will run first and substitute its results in outer query.
- The variable from outer query can be used in the inner query but reverse is not true.
- The subquery can be used in where and having clauses and sometimes in from clause also.

Classification of sub query



- **Uncorelated subquery** : Inner query will run always only once. If it is independent of outer query then it is called uncorelated subquery. For Example :
SELECT Name FROM Employee WHERE
Employee.Eno = (SELECT Work_IN.Eno FROM Work_IN);
- **Corelated Subquery** : Inner query will run some time as many times as the number of rows are available in the outer query. If it is refer a variable in outer query then it is called corelated subquery. For Example :
SELECT Name FROM Employee WHERE
Employee.Eno = (SELECT Work_IN.Eno FROM Work_IN
WHERE Dno =11);
- **Scalar subquery** : It returns only one row and one column
- **Row subquery** : It returns multiple columns but only one row
- **Table subquery** : It retrive multiple rows and multiple columns

Possible ways of writing sub query

The possible ways of writing sub query are :

- By using IN predicate : (BETWEEN, LIKE IS NULL cannot be used with the subqueries).
- By using quantifiers comparison predicates : (ANY, ALL)
- By using exist and not exist

Restrictions for subquery

The following operators cannot be used in between main query and subquery:

- BETWEEN and NOT BETWEEN operator
- LIKE and NOT LIKE operator
- IS NULL and IS NOT NULL operator

The following operators are normally used between query and subquery:

- IN predicate : IN or NOT IN
- Quantified comparison predicates : ANY, ALL, SOME
- EXIST predicates: EXIST and NOT EXIST

Questions on SQL

Question 1 : Consider the following schema:

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

The Catalog relation lists the prices charged for parts by Suppliers. Write the following queries in SQL:

1. Find the pnames of parts for which there is some supplier.

Solution :

```
SELECT p.pname FROM Parts AS p
WHERE EXISTS (
  SELECT * FROM Catalog AS c
  WHERE c.pid = p.pid)
```

2. Find the snames of suppliers who supply every red part.

Solution : SELECT s.sid, s.sname

FROM Suppliers AS s, Catalog AS c, Parts AS p

WHERE s.sid = c.sid

AND p.pid = c.pid AND p.color = 'red'

GROUP BY s.sid, s.sname

HAVING COUNT(*) = (SELECT COUNT(*) FROM Parts AS p1

WHERE p1.color='red')

3. Find the snames of suppliers who supply every part.

Solution : SELECT S.sname FROM Suppliers S

WHERE NOT EXISTS ((SELECT P.pid FROM Parts P)

EXCEPT

(SELECT C.pid FROM Catalog C

WHERE C.sid = S.sid))

4. Find the pnames of parts supplied by Acme Widget Suppliers and no one else.

Solution : SELECT p.pname FROM Parts AS p

WHERE p.pid IN (

(SELECT c1.pid AS pid FROM Catalog AS c1, Suppliers s1

WHERE c1.pid = s1.sid

AND s1.sname = 'Acme Widget Suppliers')

```
EXCEPT
(SELECT DISTINCT c2.pid AS pid
FROM Catalog AS c2, Suppliers s2
WHERE c2.pid = s2.sid
AND s2.sname != 'Acme Widget Suppliers'))
```

5. Find the sids of suppliers who charge more for some part than the average cost of that part (averaged over all the suppliers who supply that part).

Solution :

```
SELECT DISTINCT c.sid FROM Catalog AS c,
(SELECT c1.pid AS avg pid, AVG(c1.cost) AS avg cost
FROM Catalog AS c1 GROUP BY c1.pid) AS TEMP
WHERE c.pid = TEMP.avg pid AND c.cost > TEMP.avg cost
```

6. Find the sids of suppliers who supply a red part and a green part.

Solution :

```
(SELECT DISTINCT c1.sid AS SID
FROM Parts AS p1, Catalog AS c1
WHERE p1.pid = c1.pid AND p1.color = 'green')
INTERSECT (
SELECT DISTINCT c2.sid AS SID
FROM Parts AS p2, Catalog AS c2
WHERE p2.pid = c2.pid AND p2.color = 'red')
```

Or

```
SELECT DISTINCT c.sid
FROM Catalog c, Parts p
WHERE c.pid = p.pid and p.color = 'red'
INTERSECT
SELECT DISTINCT c.sid
FROM Catalog c, Parts p
WHERE c.pid = p.pid and p.color = 'green'
```

7. For every supplier who supplies at least 1 red part, print the name of the supplier and the total number of red parts that he or she supplies.

Solution :

```
SELECT S.sname, COUNT(*) as PartCount
FROM Suppliers S, Catalog C, Parts P
WHERE C.sid = S.sid and P.pid = C.pid and P.color = 'Red'
GROUP BY S.sname, S.sid
```

8. For each part, find the sname of the supplier who charges the most for that part.

Solution : SELECT P.pid, S.sname
FROM Parts P, Suppliers S, Catalog C
WHERE C.pid = P.pid
AND C.sid = S.sid
AND C.cost = (SELECT MAX (C1.cost)
FROM Catalog C1
WHERE C1.pid = P.pid)

9. Find the sids of suppliers who supply only red parts.

Solution : SELECT DISTINCT C.sid
FROM Catalog C
WHERE NOT EXISTS (SELECT *
FROM Parts P
WHERE P.pid = C.pid AND P.color <> 'Red')

10. Find the sids of suppliers who supply a red part or a green part.

Solution : SELECT DISTINCT C.sid
FROM Catalog C, Parts P
WHERE C.pid = P.pid AND P.color = 'Red'
UNION
SELECT DISTINCT C1.sid
FROM Catalog C1, Parts P1
WHERE C1.pid = P1.pid AND P1.color = 'Green'

11. For every supplier that supplies a green part and a red part, print the name and price of the most expensive part that she supplies.

Solution : SELECT S.sname, MAX(C.cost) as MaxCost
FROM Suppliers S, Parts P, Catalog C
WHERE P.pid = C.pid AND C.sid = S.sid
GROUP BY S.sname, S.sid
HAVING ANY (P.color='green') AND ANY (P.color = 'red')

12. Find the distinct pname of all parts.

Solution : SELECT DISTINCT pname
FROM Parts;

13. Find the distinct pnames of parts for which there is some supplier.

Solution : SELECT DISTINCT P.pname
FROM Parts P, Catalog C
WHERE P.pid = C.pid;

14. Find the distinct pnames of all parts sold (by some supplier) for less than 5.50

Solution : SELECT DISTINCT P.pname
FROM Parts P, Catalog C
WHERE P.pid = C.pid
AND C.cost < 5.50;

15. Find the price of the least expensive red part.

Solution : SELECT MIN(C.cost)
FROM Catalog C, Parts P
WHERE C.pid = P.pid
AND P.color = 'Red';

16. For every supplier, print the name of the supplier and the total number of parts that he or she supplies.

Solution : SELECT S.sname, COUNT(*) as PartCount
FROM Suppliers S, Catalog C
WHERE C.sid = S.sid
GROUP BY S.sname, S.sid

17. Find the pids of the most expensive parts supplied by suppliers named Yosemite Sham.

Solution : SELECT C.pid
FROM Catalog C, Suppliers S
WHERE S.sname = 'Yosemite Sham' AND
C.sid = S.sid AND
C.cost ≥ ALL (SELECT C2.cost
FROM Catalog C2, Suppliers S2
WHERE S2.sname = 'Yosemite Sham' AND
C2.sid = S2.sid)

18. For every supplier, print the name and price of the most expensive part that she supplies.

Solution : SELECT S.sname, MAX(C.cost) as MaxCost
FROM Suppliers S, Parts P, Catalog C
WHERE P.pid = C.pid AND C.sid = S.sid
GROUP BY S.sname, S.sid

QUESTIONS SQL - Part 3

Question 1 : The following relations keep track of airline flight information:

Flights(flno: integer, from: string, to: string,
distance: integer, departs: time,
arrives: time, price: real)

Aircraft(aid: integer, aname: string,
cruisingrange: integer)

Certified(eid: integer, aid: integer)

Employees(eid: integer, ename: string, salary: integer)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly. Write each of the following queries in SQL.

- a) Find the names of aircraft such that all pilots certified to operate them have salaries more than \$80,000.

Solution : SELECT DISTINCT A.aname

FROM Aircraft A

WHERE A.Aid IN (SELECT C.aid

FROM Certified C, Employees E

WHERE C.eid = E.eid AND

NOT EXISTS (SELECT *

FROM Employees E1

WHERE E1.eid = E.eid AND E1.salary < 80000))

- b) For each pilot who is certified for more than three aircraft, find the eid and the maximum cruisingrange of the aircraft for which she or he is certified.

Solution : SELECT C.eid, MAX (A.cruisingrange)

FROM Certified C, Aircraft A

WHERE C.aid = A.aid

GROUP BY C.eid

HAVING COUNT (*) > 3

- c) Find the names of pilots whose salary is less than the price of the cheapest route from Los Angeles to Honolulu.

Solution : SELECT DISTINCT E.ename

FROM Employees E

WHERE E.salary < (SELECT MIN (F.price)

FROM Flights F

WHERE F.from = 'Los Angeles' AND F.to = 'Honolulu')

d) For all aircraft with cruisingrange over 1000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.

Solution : SELECT Temp.name, Temp.AvgSalary
FROM (SELECT A.aid, A.aname AS name,
AVG (E.salary) AS AvgSalary
FROM Aircraft A, Certified C, Employees E
WHERE A.aid = C.aid AND
C.eid = E.eid AND A.cruisingrange > 1000
GROUP BY A.aid, A.aname) AS Temp

e) Find the names of pilots certified for some Boeing aircraft.

Solution : SELECT DISTINCT E.ename
FROM Employees E, Certified C, Aircraft A
WHERE E.eid = C.eid AND
C.aid = A.aid AND
A.aname LIKE 'Boeing%'

f) Find the aids of all aircraft that can be used on routes from Los Angeles to Chicago.

Solution : SELECT A.aid
FROM Aircraft A
WHERE A.cruisingrange > (SELECT MIN (F.distance)
FROM Flights F
WHERE F.from = 'Los Angeles' AND F.to = 'Chicago')

g) Identify the routes that can be piloted by every pilot who makes more than \$100,000.

Solution : SELECT DISTINCT F.from, F.to
FROM Flights F
WHERE NOT EXISTS (SELECT *
FROM Employees E
WHERE E.salary > 100000
AND
NOT EXISTS (SELECT *
FROM Aircraft A, Certified C
WHERE A.cruisingrange > F.distance
AND E.eid = C.eid
AND A.aid = C.aid))

h) Print the enames of pilots who can operate planes with cruisingrange greater than 3000 miles but are not certified on any Boeing aircraft.

Solution : SELECT DISTINCT E.ename
FROM Employees E
WHERE E.eid IN ((SELECT C.eid
FROM Certified C
WHERE EXISTS (SELECT A.aid
FROM Aircraft A


```

WHERE A.aid = C.aid
AND A.cruisingrange > 3000 )
AND
NOT EXISTS ( SELECT A1.aid
FROM Aircraft A1
WHERE A1.aid = C.aid
AND A1.aname LIKE 'Boeing%' ))

```

- i) A customer wants to travel from Madison to New York with no more than two changes of flight. List the choice of departure times from Madison if the customer wants to arrive in New York by 6 p.m.

Solution : SELECT F.departs
FROM Flights F
WHERE F.flno IN ((SELECT F0.flno
FROM Flights F0
WHERE F0.from = 'Madison' AND F0.to = 'New York'
AND F0.arrives < '18:00')
UNION
(SELECT F0.flno
FROM Flights F0, Flights F1
WHERE F0.from = 'Madison' AND F0.to < > 'New York'
AND F0.to = F1.from AND F1.to = 'New York'
AND F1.departs > F0.arrives
AND F1.arrives < '18:00')
UNION
(SELECT F0.flno
FROM Flights F0, Flights F1, Flights F2
WHERE F0.from = 'Madison'
AND F0.to = F1.from
AND F1.to = F2.from
AND F2.to = 'New York'
AND F0.to < > 'New York'
AND F1.to < > 'New York'
AND F1.departs > F0.arrives
AND F2.departs > F1.arrives
AND F2.arrives < '18:00'))

- j) Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots).

Solution : SELECT Temp1.avg - Temp2.avg
FROM (SELECT AVG (E.salary) AS avg
FROM Employees E
WHERE E.eid IN (SELECT DISTINCT C.eid
FROM Certified C)) AS Temp1,
(SELECT AVG (E1.salary) AS avg
FROM Employees E1) AS Temp2

k) Print the name and salary of every nonpilot whose salary is more than the average salary for pilots.

Solution : SELECT E.ename, E.salary
FROM Employees E
WHERE E.eid NOT IN (SELECT DISTINCT C.eid
FROM Certified C)
AND E.salary > (SELECT AVG (E1.salary)
FROM Employees E1
WHERE E1.eid IN
(SELECT DISTINCT C1.eid
FROM Certified C1))

l) Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles.

Solution : SELECT E.ename
FROM Employees E, Certified C, Aircraft A
WHERE C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING EVERY (A.cruisingrange > 1000)

m) Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircrafts.

Solution : SELECT E.ename
FROM Employees E, Certified C, Aircraft A
WHERE C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING EVERY (A.cruisingrange > 1000) AND COUNT (*) > 1

n) Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles and who are certified on some Boeing aircraft.

Solution : SELECT E.ename
FROM Employees E, Certified C, Aircraft A
WHERE C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING EVERY (A.cruisingrange > 1000)
AND ANY (A.aname = 'Boeing')

SQL Based Questions - Part 4

Question 1 : Consider the following relational schema. An employee can work in more than one department; the pct time field of the Works relation shows the percentage of time that a given employee works in a given department.

Emp(eid: integer, ename: string, age: integer, salary: real)
Works(eid: integer, did: integer, pct time: integer)
Dept(did: integer, dname: string, budget: real,
managerid: integer)

Write the following queries in SQL:

1. Print the names and ages of each employee who works in both the Hardware department and the Software department.

Solution :

```
SELECT E.ename, E.age
FROM Emp E, Works W1, Works W2, Dept D1, Dept D2
WHERE E.eid = W1.eid
AND W1.did = D1.did AND D1.dname = 'Hardware'
AND E.eid = W2.eid AND W2.did = D2.did
AND D2.dname = 'Software'
```

2. For each department with more than 20 full-time-equivalent employees (i.e., where the part-time and full-time employees add up to at least that many full-time employees), print the did together with the number of employees that work in that department.

Solution :

```
SELECT W.did, COUNT (W.eid)
FROM Works W
GROUP BY W.did
HAVING 2000 < ( SELECT SUM (W1.pct time)
FROM Works W1
WHERE W1.did = W.did )
```

3. Print the name of each employee whose salary exceeds the budget of all of the departments that he or she works in.

Solution :

```
SELECT E.ename
FROM Emp E
WHERE E.salary > ALL (SELECT D.budget
FROM Dept D, Works W
WHERE E.eid = W.eid AND D.did = W.did)
```

4. Find the managerids of managers who manage only departments with budgets greater than \$1 million.

Solution : SELECT DISTINCT D.managerid
FROM Dept D
WHERE 1000000 < ALL (SELECT D2.budget
FROM Dept D2
WHERE D2.managerid = D.managerid)

5. Find the enames of managers who manage the departments with the largest budgets.

Solution : SELECT E.ename
FROM Emp E
WHERE E.eid IN (SELECT D.managerid
FROM Dept D
WHERE D.budget = (SELECT MAX (D2.budget)
FROM Dept D2))

6. If a manager manages more than one department, he or she controls the sum of all the budgets for those departments. Find the managerids of managers who control more than \$5 million.

Solution : SELECT D.managerid
FROM Dept D
WHERE 5000000 < (SELECT SUM (D2.budget)
FROM Dept D2
WHERE D2.managerid = D.managerid)

7. Find the managerids of managers who control the largest amounts.

Solution : SELECT DISTINCT tempD.managerid
FROM (SELECT DISTINCT D.managerid,
SUM (D.budget) AS tempBudget
FROM Dept D
GROUP BY D.managerid) AS tempD
WHERE tempD.tempBudget = (SELECT MAX (tempD.tempBudget)
FROM tempD)

8. Find the enames of managers who manage only departments with budgets larger than \$1 million, but at least one department with budget less than \$5 million.

Solution : SELECT E.ename
FROM Emp E, Dept D
WHERE E.eid = D.managerid GROUP BY E.Eid, E.ename
HAVING EVERY (D.budget > 1000000)
AND ANY (D.budget < 5000000)

Questions related to SQL - Part 5

Question 1 : Consider the instance of the Sailors relation

Sid	Sname	rating	Age
18	Jones	3	30.0
41	Jonah	6	56.0
22	Ahab	7	44.0
63	Moby	Null	15.0

Table 5.1

1. Write SQL queries to compute the average rating, using AVG; the sum of the ratings, using SUM; and the number of ratings, using COUNT.

Solution :
SELECT AVG (S.rating) AS AVERAGE
FROM Sailors S
SELECT SUM (S.rating)
FROM Sailors S
SELECT COUNT (S.rating)
FROM Sailors S

2. If you divide the sum just computed by the count, would the result be the same as the average? How would your answer change if these steps were carried out with respect to the age field instead of rating?

Solution : The result using SUM and COUNT would be smaller than the result using AVERAGE if there are tuples with rating = NULL.

This is because all the aggregate operators, except for COUNT, ignore NULL values. So the first approach would compute the average over all tuples while the second approach would compute the average over all tuples with non-NULL rating values. However, if the aggregation is done on the age field, the answers using both approaches would be the same since the age field does not take NULL values.

3. Consider the following query: Find the names of sailors with a higher rating than all sailors with age < 21. The following two SQL queries attempt to obtain the answer to this question. Do they both compute the result? If not, explain why.

Under what conditions would they compute the same result?

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS ( SELECT *
FROM Sailors S2
WHERE S2.age < 21
AND S.rating <= S2.rating )
SELECT *
FROM Sailors S
WHERE S.rating > ANY ( SELECT S2.rating
FROM Sailors S2
WHERE S2.age < 21 )
```

Solution : Only the first query is correct. The second query returns the names of sailors with a higher rating than at least one sailor with age < 21. Note that the answer to the second query does not necessarily contain the answer to the first query. In particular, if all the sailors are at least 21 years old, the second query will return an empty set while the first query will return all the sailors.

This is because the NOT EXISTS predicate in the first query will evaluate to true if its subquery evaluates to an empty set, while the ANY predicate in the second query will evaluate to false if its subquery evaluates to an empty set. The two queries give the same results if and only if one of the following two conditions hold :

1. The Sailors relation is empty, or
2. There is at least one sailor with age > 21 in the Sailors relation, and for every sailor s, either s has a higher rating than all sailors under 21 or s has a rating no higher than all sailors under 21.

4. Consider the instance of Sailors shown in Table 5.1 Let us define instance S1 of Sailors to consist of the first two tuples, instance S2 to be the last two tuples, and S to be the given instance.

(a) Show the left outer join of S with itself, with the join condition being sid=sid.

(b) Show the right outer join of S with itself, with the join condition being sid=sid.

(c) Show the full outer join of S with itself, with the join condition being sid=sid.

Solution :

(a)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	<i>null</i>	15.0	63	moby	<i>null</i>	15.0

(b)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	<i>null</i>	15.0	63	moby	<i>null</i>	15.0

(c)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	<i>null</i>	15.0	63	moby	<i>null</i>	15.0

(d) Show the left outer join of S1 with S2, with the join condition being *sid*=*sid*.

(e) Show the right outer join of S1 with S2, with the join condition being *sid*=*sid*.

(f) Show the full outer join of S1 with S2, with the join condition being *sid*=*sid*.

Solution :

(d)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
18	jones	3	30.0	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>
41	jonah	6	56.0	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>

(e)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	22	ahab	7	44.0
<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	63	moby	<i>null</i>	15.0

(f)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
18	jones	3	30.0	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>
41	jonah	6	56.0	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>
<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	22	ahab	7	44.0
<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	63	moby	<i>null</i>	15.0

Questions Based on SQL - Part 6

Question 1 : Consider the following relations:

Student(snum: integer, sname: string, major: string, level: string, age: integer)
Class(name: string, meets at: string, room: string, fid: integer)
Enrolled(snum: integer, cname: string)
Faculty(fid: integer, fname: string, deptid: integer)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

Write the following queries in SQL. No duplicates should be printed in any of the answers.

1. Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach.

Solution :

```
SELECT DISTINCT S.Sname
FROM Student S, Class C, Enrolled E, Faculty F
WHERE S.snum = E.snum
AND E.cname = C.name
AND C.fid = F.fid AND
F.fname = 'I.Teach' AND S.level = 'JR'
```

2. Find the age of the oldest student who is either a History major or enrolled in a course taught by I. Teach.

Solution :

```
SELECT MAX(S.age)
FROM Student S
WHERE (S.major = 'History')
OR S.snum IN (SELECT E.snum
FROM Class C, Enrolled E, Faculty F
WHERE E.cname = C.name AND C.fid = F.fid
AND F.fname = 'I.Teach' )
```

3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.

Solution :

```
SELECT C.name
FROM Class C
WHERE C.room = 'R128'
OR C.name IN (SELECT E.cname
FROM Enrolled E
GROUP BY E.cname
HAVING COUNT (*) >= 5)
```


4. Find the names of all students who are enrolled in two classes that meet at the same time.

```
Solution : SELECT DISTINCT S.sname
           FROM Student S
           WHERE S.snum IN (SELECT E1.snum
                           FROM Enrolled E1, Enrolled E2, Class C1, Class C2
                           WHERE E1.snum = E2.snum AND E1.cname < > E2.cname
                           AND E1.cname = C1.name
                           AND E2.cname = C2.name AND C1.meets at = C2.meets at)
```

5. Find the names of faculty members who teach in every room in which some class is taught.

```
Solution : SELECT DISTINCT F.fname
           FROM Faculty F
           WHERE NOT EXISTS (( SELECT *
                               FROM Class C )
                               EXCEPT
                               (SELECT C1.room
                                FROM Class C1
                                WHERE C1.fid = F.fid ))
```

6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.

```
Solution : SELECT DISTINCT F.fname
           FROM Faculty F
           WHERE 5 > (SELECT COUNT (E.snum)
                     FROM Class C, Enrolled E
                     WHERE C.name = E.cname
                     AND C.fid = F.fid)
```

7. For each level, print the level and the average age of students for that level.

```
Solution : SELECT S.level, AVG(S.age)
           FROM Student S
           GROUP BY S.level
```

8. For all levels except JR, print the level and the average age of students for that level.

```
Solution : SELECT S.level, AVG(S.age)
           FROM Student S
           WHERE S.level < > 'JR'
           GROUP BY S.level
```

9. For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught.

```
Solution : SELECT F.fname, COUNT(*) AS CourseCount
FROM Faculty F, Class C
WHERE F.fid = C.fid
GROUP BY F.fid, F.fname
HAVING EVERY ( C.room = 'R128' )
```

10. Find the names of students enrolled in the maximum number of classes.

```
Solution : SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum IN (SELECT E.snum
FROM Enrolled E
GROUP BY E.snum
HAVING COUNT (*) >= ALL (SELECT COUNT (*)
FROM Enrolled E2
GROUP BY E2.snum ))
```

11. Find the names of students not enrolled in any class.

```
Solution : SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum NOT IN (SELECT E.snum
FROM Enrolled E )
```

12. For each age value that appears in Students, find the level value that appears most often. For example, if there are more FR level students aged 18 than SR, JR, or SO students aged 18, you should print the pair (18, FR).

```
Solution : SELECT S.age, S.level
FROM Student S
GROUP BY S.age, S.level,
HAVING S.level IN (SELECT S1.level
FROM Student S1
WHERE S1.age = S.age
GROUP BY S1.level, S1.age
HAVING COUNT (*) >= ALL (SELECT COUNT (*)
FROM Student S2
WHERE s1.age = S2.age
GROUP BY S2.level, S2.age))
```