UNIT - II

Structured Query Language

Indira College of Engineering Management, Pune

Introduction

- SQL stands for Structured Query Language.
- It is the language of database and almost all companies use databases to store their data.
- It is domain-specific language.
- SQL is declarative.
- Keys and Constraints.
- Join and Nested queries.

Introduction

- SQL makes use of query. A Query is a set of instruction given to the database management system. It tells any database what information we would like to get from the database.
- SQL allows users to query the database in a number of ways, using English-like statements.
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.

Rules

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.

Characteristics and Advantages -

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

Data Types and Literals -

- Numeric data type
- Character-string data type
- Date and Time
- Boolean Data Type

Data Types and Literals -

- Numeric data type –
  - Integer numbers : BIT, INT, TINYINT, SMALLINT, BIGINT.
  - Floating – point(real) Numbers : REAL, DOUBLE, FLOAT.
- Character-string data type –

1. Char -

- It accepts character or string type of data.
- It is Fixed length data type.
- The length of the character string is specified while assigning the data type.
- Example – character(n), n- maximum size of the character string.

Data Types and Literals -

- Character-string data type –

2. Varchar -

- It accepts character or string type of data.
- It is variable length data type.
- The length of the character string is specified while assigning the data type which indicates the maximum number of characters it can accept.
- Example -

Data Types and Literals -

- Boolean data type –

- Accept Values of TRUE or FALSE or NULL.

- No need to declare size while declaring Boolean data type.

- Date and Time – Can store date and time.

- Components are YEAR, MONTH, and DAY in the form YYYY-MM-DD.

DDL, DML, DCL and TCL Structure -

- There are four types of SQL commands –

  DDL

  DML

  DCL

  TCL

Data Definition Language(DDL) -

DDL stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.

It is used to create schema, tables, indexes, constraints, etc. in the database.

Using the DDL statements, you can create the skeleton of the database.

Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Data Definition Language(DDL) -

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete a table from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

Data Manipulation Language(DML) -

DML stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.

Data Control Language(DCL) -

DCL stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.

The DCL execution is transactional. It also has rollback parameters.

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

- CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

Transaction Control Language -

- Its allow you to control and manage transactions to maintain the integrity of data within SQL statements.

-COMMIT – This command is used to save permanently any transaction to database.

-ROLLBACK – This command is used to undo transactions that haven’t saved to database.

TABLES : Creating, Modifying, Deleting

- Creating Table – CREATE TABLE statement is used to create table in database.
- Syntax –

```
CREATE TABLE table_name(
        column1   datatype(size),
        column2   datatype(size),
        column3 datatype(size));
```

Example –

```
CREATE TABLE Students(
        Roll_No. int,
        Stud_Fname VARCHAR(20),
        Stud_Mname VARCHAR(20),
        Stud_Lname VARCHAR(20)
        Address VARCHAR(20));
```

TABLES : Creating, Modifying, Deleting

- Insertion of data into the table – We can insert data into the table using INSERT statement.
- Syntax –

INSERT INTO table_name(column1, column2,….,Columnn) VALUES(value1, value2,….,valuen);

Example – INSERT INTO Students(Roll_No., Stud_Fname, Stud_Mname, Stud_Lname, Address) VALUES(1, ‘AAA’, ‘BBB’, ‘CCC’, ‘Pune’);

TABLES : Creating, Modifying, Deleting

- Modifying the Record from the Table – For modifying the existing of a table, update query is used.
- Syntax –

```
UPDATE table_name
SET column1=value1, column2=value2
WHERE condition;
```

Example – UPDATE Students

```
        SETAddress=‘Gujarat’
        WHERE Roll_No.= 3;
```

TABLES : Creating, Modifying, Deleting

- Deleting Record from the Table – We can delete one or more records based on some condition.
- Syntax –

```
DELETE FROM table_name WHERE condition;
```

Example-

```
DELETE FROM Students WHERE Roll_No. = 3;
```

SQL DML Queries -

- **DML** stands for Data Manipulation Language.
- The basic operations under DML queries are SELECT, INSERT, UPDATE and DELETE.

SQL DML Queries -

1. SELECT Query -

- The select statement is used to fetch the data from the database table.
- The result returns the data in the form of table. These result tables are called resultsets.
- We can use the keyword DISTINCT. It is an optional keyword indicating that the answer should not contain duplicates. Normally if we write the SQL without DISTINCT operator then it does not eliminate the duplicates.

SQL DML Queries -

- Syntax –

```
SELECT col1, col2,….,coln FROM table_name;
```

Example –

```
SELECT Roll_No, Name FROM Students;
```

Select all the records present in the table we make use of * character.

- Syntax –

```
SELECT * FROM table_name;
```

Example –

```
SELECT * FROM Students;
```

SQL DML Queries -

- Use of DISTINCT Keyword : The keyword DISTINCT is used along with the SELECT statements.
- It is used to obtain unique values from the table. This query does not allow duplication of element.
- Syntax –

```
SELECT DISTINCT column_name FROM table_name;
```

Example –

```
SELECT DISTINCTAddress FROM Students;
```

SQL DML Queries -

2. WHERE -

- The WHERE command is used to specify some condition. Based on this condition the data present in the table can be displayed or can be updated or deleted.
- Syntax -

```
SELECT col1, col2
FROM  table_name
WHERE condition;
```

SQL DML Queries -

2. WHERE –

Example –

```
SELECT Roll_No
FROM Students
WHEREAddress=‘Gujarat’;
```

If we want all the record of those person who live in Gujarat then we can write the query using WHERE clause as:

Example –

```
SELECT * FROM Students
WHERE Address=‘Gujarat’;
```

SQL DML Queries -

3. Clause –

Most commonly used clauses in SQL statements are order by, Group by and Having.

1. Order by
2. Group by
3. Having

SQL DML Queries -

1. Order By –

- Many times we need the records in the table to be in sorted order.
- If the records are arranged in increasing order of some column then it is called ascending order.
- If the records are arranged in decreasing order of some column then it is called descending order.
- For getting the stored records in the table we use ORDER BY command.
- The ORDER BY keyword sorts the records in ascending order by default.

SQL DML Queries -

1. Order By –

- Syntax –

  SELECT col1, col2,….,coln

  FROM table_name

  ORDER BY col1, col2,….ASC/DESC;

- Example – SELECT * FROM Students ORDER BY Roll_No DESC;

SQL DML Queries -

2. Group By –

- The GROUP BY clauses is a SQL command that is used to group rows that have the same values.
- The GROUP BY clause is used in the SELECT statement.
- Optionally it is used in conjunction with aggregate functions.
- The queries that contain the GROUP BY clause are called grouped queries.
- This query returns a single row for every grouped item.

SQL DML Queries -

2. Group By –

- Syntax –

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

SQL DML Queries -

2. Group By –

Student

Student_id	Name	Marks	Address
1	AAA	60	Pune
2	BBB	70	Gujarat
3	CCC	90	Pune
4	DDD	55	Gujarat
5	EEE	65	Delhi

SQL DML Queries -

2. Group By –

Find the total marks of each student in each city.

- EXAMPLE –

-
  ```
  SELECT SUM(marks), Address
  FROM Student
  GROUP BYAddress;
  ```

-
  ```
  SELECT count(*),Address
  FROM Student
  GROUP BYAddress;
  ```

SQL DML Queries -

3. Having –

- HAVING filters records that work on summarized GROUP BY results.
- HAVING applies to summarized group records, whereas WHERE applies to individual records.
- Only the groups that meet the HAVING criteria will be returned.
- HAVING requires that a GROUP BY clause is present.

SQL DML Queries -

3. Having –

- Syntax –

```
SELECT column_name
FROM  table_name
GROUP BY column_names
HAVING condition;
```

SQL DML Queries -

3. Having –

- Example –

```
SELECT Address
FROM Student
GROUP BY Address
HAVING count(*)<2 ;
```

View -

- In SQL, a view is a virtual tables.
- A view also has rows and columns as they are in a real table in the database.
- We can create a view by selecting fields from one or more tables present in the database.
- A view can either have all the rows of a table or specific rows based on certain condition.

View -

1. Creating View –

Student

Roll_No	Name	Marks	City
101	AAA	70	Pune
102	BBB	60	Mumbai
103	CCC	65	Pune
104	DDD	75	Gujarat
105	EEE	72	Delhi
106	FFF	74	Pune

View -

i. Creating view having all records and fields from existing table.

- Syntax –

CREATE VIEW view_name AS
SELECT column1, column2,….
FROM table_name;

- Example –

CREATE VIEW Student_view1
AS SELECT * FROM Student;

Student_view1

Roll_No	Name	Marks	City
101	AAA	70	Pune
102	BBB	60	Mumbai
103	CCC	65	Pune
104	DDD	75	Gujarat
105	EEE	72	Delhi
106	FFF	74	Pune

View -

ii. Creating view having specific fields but all the records from existing table.

- Syntax –

```
CREATE VIEW view_name AS
SELECT column1, column2,….
FROM table_name;
```

- Example –

```
 CREATE VIEW Student_view2
AS SELECT Roll_No, Name FROM Student;
```

Student_view2

Roll_No	Name
101	AAA
102	BBB
103	CCC
104	DDD
105	EEE
106	FFF

View -

iii. Creating view having specific records but all the fields from existing table.

- Syntax –

```
CREATE VIEW view_name AS
SELECT * FROM existing_table_name
WHERE condition;
```

- Example –

```
 CREATE VIEW Student_view3
AS SELECT * FROM Student
WHERE Marks>70;
```

Student_view3

Roll_NO	Name	Marks	City
104	DDD	75	Gujarat
105	EEE	72	Delhi
106	FFF	74	Pune

View -

2. Updating View – Update query is used to update the records of view. Updation in view reflects the original table also. Means the same changes will be made in the original table also.

Roll_No	Name	Marks	City
101	AAA	70	Pune
102	BBB	60	Mumbai
103	CCC	65	Pune
104	DDD	75	Gujarat
105	EEE	72	Delhi
106	FFF	74	Pune

View -

- Syntax –

```
UPDATE view_name
SET field_name = new_value
WHERE condition;
```

Roll_No	Name	Marks	City
101	AAA	70	Pune
102	BBB	60	Mumbai
103	CCC	65	Pune
104	DDD	75	Gujarat
105	EEE	72	Delhi
106	FFF	74	Pune

View -

- Example –

```
UPDATE Student_view
SET City = Delhi
WHERE Roll_No = 103;
```

Student_view

Roll_No	Name	Marks	City
101	AAA	70	Pune
102	BBB	60	Mumbai
103	CCC	65	**Delhi**
104	DDD	75	Gujarat
105	EEE	72	Delhi
106	FFF	74	Pune

Student

Roll_No	Name	Marks	City
101	AAA	70	Pune
102	BBB	60	Mumbai
103	CCC	65	**Delhi**
104	DDD	75	Gujarat
105	EEE	72	Delhi
106	FFF	74	Pune

View -

- DROP query is used to delete a view.
- Syntax –

  DROP view view_name;
- Example –

  DROP view Student_view;

SQL Operators -

- The SQL reserved words and characters are called operators, which are used with a WHERE clause in a SQL query. In SQL, an operator can either be a unary or binary operator. The unary operator uses only one operand for performing the unary operation, whereas the binary operator uses two operands for performing the binary operation.

Types of Operator -

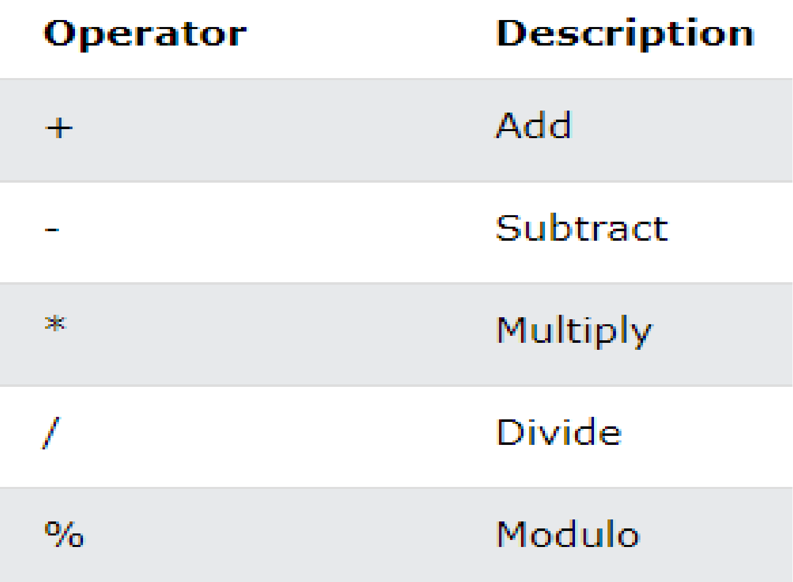
SQL operators are categorized in the following categories:

- SQLArithmetic Operators -
- SQL Comparison Operators -
- SQL Logical Operators -
- SQL Compound Operators -
- SQL Bit-wise Operators -

Types of Operator -

1. SQLArithmetic Operators –

The **Arithmetic Operators** perform the mathematical operation on the numerical data of the SQL tables. These operators perform addition, subtraction, multiplication, and division operations on the numerical operands.



Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

Types of Operator -

- **Syntax for Arithmetic operator -**

- **SELECT** Column_Name_1 Addition_Operator Column_Name2 **FROM** Table_Name;

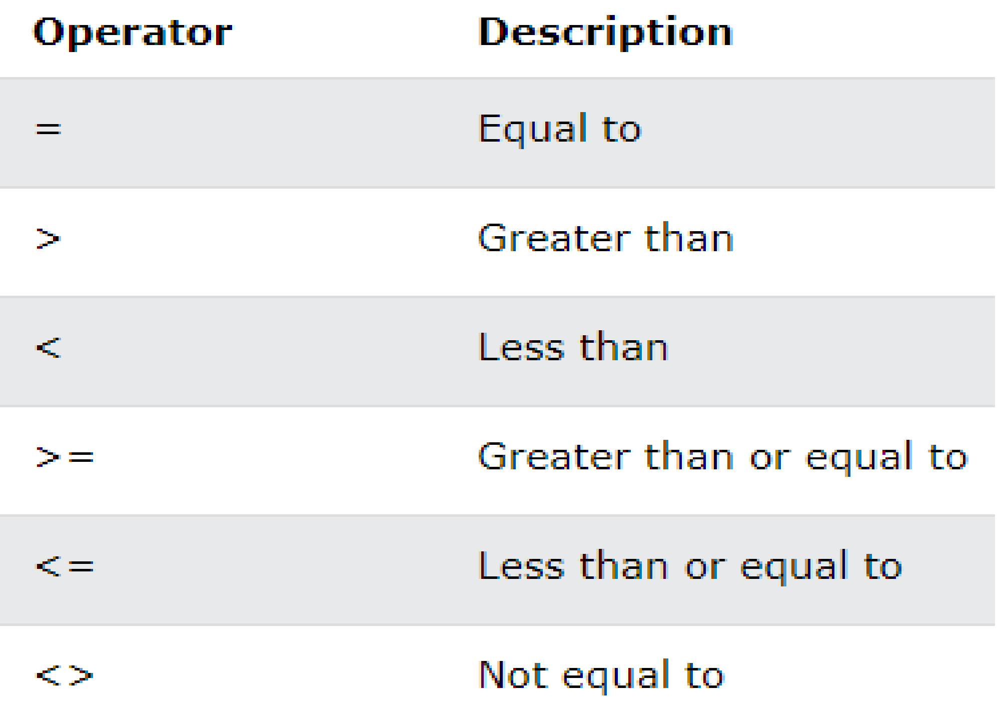
- S**ELECT** Column_Name_1 Subtraction_Operator Column_Name2 **FROM** Table_Name;

- **SELECT** Column_Name_1 Multiplication_Operator Column_Name2 **FROM** Table_Name;

- **SELECT** Column_Name_1 Division_Operator Column_Name2 **FROM** Table_Name;

Types of Operator -

2. SQL Comparison Operator –



Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Syntax –

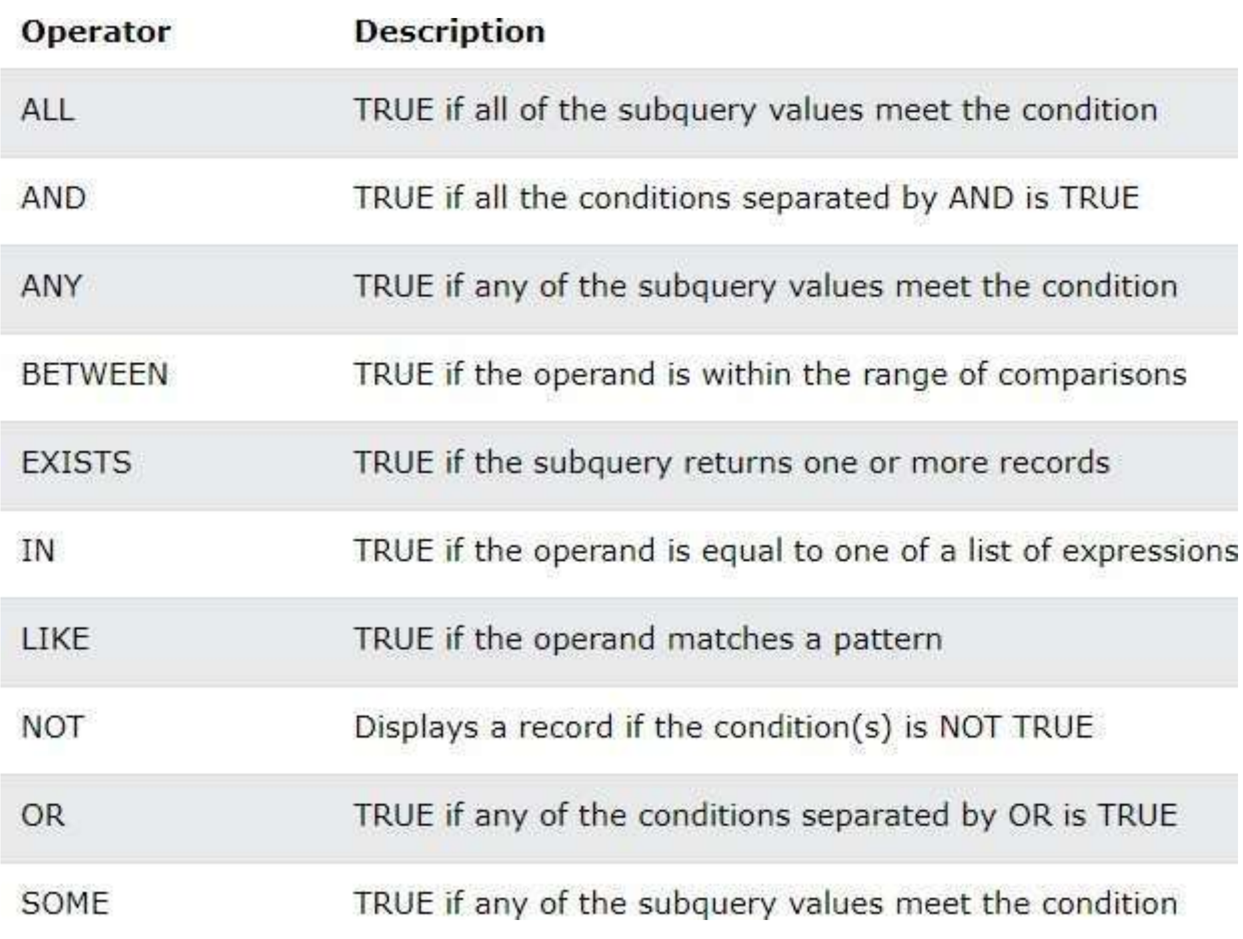
SELECT col1, col2,….,coln FROM table_name WHERE condition;

Example -

SELECT name_emp, salary FROM Employee where salary<20000;

Types of Operator -

3. SQL Logical Operator –



Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

Types of Operator -

3. SQL Logical Operator –

- SELECT column1, Column2, column3,….,columnn

  FROM tableName

  WHERE logical condition ;

- Example –

SELECT name ,salary, age FROM Employee WHERE age>27 AND salary>25000;

Set Operations -

- Set is a collection of elements on which union, all union, intersection and difference operations can be performed.

1. Union
2. All union
3. Intersect

Set Operations -

Employee

Emp_No	EName	Job	DeptNo	Salary
101	King	President	10	5000
102	Blake	Manager	30	2500
103	Clark	Manager	20	2500
104	Jones	Clerk	20	3000
105	Smith	Salesman	30	3000

Department

Dept_no	DeptName	Location
10	Sales	Mumbai
20	Production	Pune
30	Accounts	Nasik
40	Research	Delhi

Set Operations -

1. Union – The UNION operator is used to combine the result-set of two or more SELECT statements.
   - Every SELECT statement within UNION must have the same number of columns
   - The columns must also have similar data types
   - The columns in every SELECT statement must also be in the same order

- Syntax –

```
Select column_name from table1
union
select column_name from table2;
```

- Example –

```
select DeptNo from Employee
union
select Dept_no from Department;
```

10
20
30
40

Set Operations -

2. Union All – The Union All operator returns all rows selected by either query including duplicates.

Syntax –

```
Select column_name from table1
union all
select column_name from table2;
```

- Example –

```
select DeptNo from Employee
union all
select Dept_no from Department;
```

10
30
20
20
30
10
20
30
40

Set Operations -

3. Intersect – The intersect operator returns only those rows which are common in both the queries.

Syntax –

```
Select column_name from table1
intersect
select column_name from table2;
```

- Example –

```
select DeptNo from Employee
intersect
select Dept_no from Department;
```

10
20
30

Predicates and Joins -

- PREDICATES –

A predicate is **an expression that evaluates to TRUE, FALSE**. Predicates are used in the search condition of WHERE clauses and HAVING clauses, the join conditions of FROM clauses, and other constructs where a Boolean value is required.

Predicates –

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
1	King	President	17/11/2000	5000		10
2	Blake	Manager	5/1/2001	2500		30
3	Clark	Manager	6/2/2001	2500	300	20
4	Jones	Clerk	4/3/2002	3000	500	20
5	Smith	Salesman	5/5/2002	3000	0	30
6	James	Clerk	6/5/2002	2000		10

Predicates -

1. Comparison Predicates – This is the comparison of two expressions separated by a comparison operator.

Example –

= Equal to predicate

Select * from Employee

Where EName = ‘King’;

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
1	King	President	17/11/2000	5000		10

Predicates -

Example –

> Greater than predicate

Select * from Employee

Where Salary>3000;

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
1	King	President	17/11/2000	5000		10

Predicates -

Example –

< Less than predicate

Select * from Employee

Where Salary<3000;

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
2	Blake	Manager	5/1/2001	2500		30
3	Clark	Manager	6/2/2001	2500	300	20
6	James	Clerk	6/5/2002	2000		10

Predicates -

Example –

>= Greater than equal to predicate

Select * from Employee

Where Salary >= 3000;

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
1	King	President	17/11/2000	5000		10
4	Jones	Clerk	4/3/2002	3000	500	20
5	Smith	Salesman	5/5/2002	3000	0	30

Predicates -

Example –

<= Less than equal to predicate

Select * from Employee

Where Salary <= 3000;

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
2	Blake	Manager	5/1/2001	2500		30
3	Clark	Manager	6/2/2001	2500	300	20
4	Jones	Clerk	4/3/2002	3000	500	20
5	Smith	Salesman	5/5/2002	3000	0	30
6	James	Clerk	6/5/2002	2000		10

Predicates -

Example –

<> Not equal to predicate

Select * from Employee

Where Salary <> 3000;

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
1	King	President	17/11/2000	5000		10
2	Blake	Manager	5/1/2001	2500		30
3	Clark	Manager	6/2/2001	2500	300	20
6	James	Clerk	6/5/2002	2000		10

Predicates -

Example –

Combination of Predicates can be used with AND operator –

Select * from Employee

Where salary >= 3000 AND salary <= 5000;

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
1	King	President	17/11/2000	5000		10
4	Jones	Clerk	4/3/2002	3000	500	20
5	Smith	Salesman	5/5/2002	3000	0	30

Predicates -

2. Between Predicate – Between predicate is used to specify certain range of values. The AND keyword is used in this predicate.

Example –

```
Select * from Employee
where commission between 300 and 500;
```

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
3	Clark	Manager	6/2/2001	2500	300	20
4	Jones	Clerk	4/3/2002	3000	500	20

Predicates -

Example –

Select * from Employee

where commission not between 300 and 500;

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
5	Smith	Salesman	5/5/2002	3000	0	30

Predicates -

3. In Predicate –

IN Predicate particularly determines whether the value of expression given to test matches any value in specified the list.

Example – Display the records of employee from DeptNo 10 and 20.

```
        Select * from Employee
        Where DeptNo in(10, 20);
```

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
1	King	President	17/11/2000	5000		10
3	Clark	Manager	6/2/2001	2500	300	20
4	Jones	Clerk	4/3/2002	3000	500	20
6	James	Clerk	6/5/2002	2000		10

Predicates -

4. Like Predicate –

Like Operator determines whether a specific character string matches the given pattern or not.

In the pattern we can use regular character and wildcard characters.

Example – Display records of Employee whose names start with letter ‘J’.

```
select * from Employee where EName like ‘J%’;
```

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
4	Jones	Clerk	4/3/2002	3000	500	20
6	James	Clerk	6/5/2002	2000		10

Predicates -

4. Like Predicate –

Example – Display records of Employee whose names ends with letter ‘k’.

```
select * from Employee where EName like ‘%k’;
```

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
3	Clark	Manager	6/2/2001	2500	300	20

Predicates -

4. Like Predicate –

Example – Display records of Employee whose names contains ‘L’ as second character.

select * from Employee where EName like ‘_l%’;

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
2	Blake	Manager	5/1/2001	2500		30
3	Clark	Manager	6/2/2001	2500	300	20

Predicates -

4. Like Predicate –

Example – Display records of Employee whose names contains character anywhere.

select * from Employee where EName like ‘%a%’;

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
2	Blake	Manager	5/1/2001	2500		30
3	Clark	Manager	6/2/2001	2500	300	20
6	James	Clerk	6/5/2002	2000		10

Predicates -

4. Is Null / Is Not Null –

When values for some attributes are not available then, NULL value is assigned. To display records having NULL value, IS NULL predicate is used.

Example – Display records of Employee who never get any commission.

```
select * from Employee where Commission IS NULL;
```

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
1	King	President	17/11/2000	5000		10
2	Blake	Manager	5/1/2001	2500		30
6	James	Clerk	6/5/2002	2000		10

Predicates –

4. Is Null / Is Not Null –

The NOT keyword can be used to get values opposite to given condition.
Example – Display records of Employee who get commission.

```
select * from Employee where Commission IS NOT NULL;
```

Emp_No	EName	Job	HireDate	Salary	Commission	DeptNo
3	Clark	Manager	6/2/2001	2500	300	20
4	Jones	Clerk	4/3/2002	3000	500	20
5	Smith	Salesman	5/5/2002	3000	0	30

Indexing -

- The Index in SQL is a special table used to speed up the searching of the data in the database tables.
- It also retrieves a vast amount of data from the tables frequently.
- The INDEX requires its own space in the hard disk.
- With the help of indexing data retrieval becomes fast and efficient. The concept of index is just similar to the index at the back of the book which contains the keywords.
- Using these keywords it is easy to locate the desired record quickly from the database table.

Indexing -

1. Creating an index –

- Syntax –

```
CREATE INDEX index_name ON table_name(column_name);
```

Example –

```
CREATE INDEX inx_isbn
ON Book(isbn);
```

//Display - show indexes from table_name;

isbn	bname	Author
005	DBMS	XYZ
006	OS	ABC
007	DAA	PQR

Indexing -

2. Creating index on Multiple columns –

- Syntax –

```
CREATE INDEX index_name
ON table_name(column1, column2,….);
```

Example –

```
CREATE INDEX inx_isbn1
ON Book(bname, Author);
```

isbn	bname	Author
005	DBMS	XYZ
006	OS	ABC
007	DAA	PQR

Indexing -

3. Creating index using UNIQUE –

- Syntax –

```
CREATE UNIQUE INDEX index_name
ON table_name(column1, column2,….);
```

Example –

```
CREATE UNIQUE INDEX inx_isbn3
ON Book(bname);
```

isbn	bname	Author
005	DBMS	XYZ
006	OS	ABC
007	DAA	PQR

Indexing -

3. Dropping the Index – The DROP INDEX statement is used to delete an index in a table.

- Syntax –

DROP INDEX index_name on table_name;

Example –

DROP INDEX inx_isbn on Book;

isbn	bname	Author
005	DBMS	XYZ
006	OS	ABC
007	DAA	PQR

Join -

- JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.
- Types of Joins –
  1. Natural Join –
  2. Equi Join –
  3. Left Outer Join –
  4. Right Outer Join –
  5. Self Join

Join -

1. Natural Join –

Employee

Eno	Ename	Address
1	Ram	Delhi
2	Varun	Pune
3	Rani	Pune
4	Amrit	Delhi

Department

Dno	Dname	Eno
D1	HR	1
D2	IT	2
D3	Marketing	4

Find the Ename who is working in department.

Join -

1. Natural Join –

Eno	Ename	Dno	Eno
1	Ram	D1	1
1	Ram	D2	2
1	Ram	D3	4
2	Varun	D1	1
2	Varun	D2	2
2	Varun	D3	4
3	Rani	D1	1
3	Rani	D2	2
3	Rani	D3	4
4	Amrit	D1	1
4	Amrit	D2	2
4	Amrit	D3	4

Join -

1. Natural Join –

- Example –

Select Ename from Employee NATURAL JOIN Department;

Ename
Ram
Varun
Amrit

Join -

2. Equi Join –

= Operator will use.

Employee

Eno	Ename	Address
1	Ram	Delhi
2	Varun	Pune
3	Rani	Pune
4	Amrit	Delhi

Department

Dno	Location	Eno
D1	Delhi	1
D2	Gujarat	2
D3	Patna	4

- Find the Ename who worked in a department having location same as their Address.

Join -

2. Equi Join –

Eno	Ename	Address	Dno	Location	Eno
1	Ram	Delhi	D1	Delhi	1
1	Ram	Delhi	D2	Gujrat	2
1	Ram	Delhi	D3	Patna	4
2	Varun	Pune	D1	Delhi	1
2	Varun	Pune	D2	Gujrat	2
2	Varun	Pune	D3	Patna	4
3	Rani	Pune	D1	Delhi	1
3	Rani	Pune	D2	Gujrat	2
3	Rani	Pune	D3	Patna	4
4	Amrit	Delhi	D1	Delhi	1
4	Amrit	Delhi	D2	Gujrat	2
4	Amrit	Delhi	D3	Patna	4

Join -

2. Equi Join –

Select Ename from Employee, Department

where Employee.Eno = Department.Eno

and

Employee.Address = Department.Location;

Eno	Ename	Address	Dno	Location	Eno
1	Ram	Delhi	D1	Delhi	1

Ename
Ram

Join -

3. Left Outer Join – It gives the matching rows and the rows which are in left table but not in right table.

Eno	Ename	Dno
E1	Ram	D1
E2	Varun	D2
E3	Rani	D1
E4	Amrit	

Dno	Dname	Location
D1	IT	Delhi
D2	HR	Hyderabad
D3	Finance	Pune

Join -

3. Left Outer Join –

select Eno, Ename, Dname, Location from Employee Left outer join Department ON Employee.Dno = Department.Dno;

Eno	Ename	Dname	Location
E1	Ram	IT	Delhi
E2	Varun	HR	Hyderabad
E3	Rani	IT	Delhi
E4	Amrit	NULL	NULL

Join -

4. Right Outer Join – It gives the matching rows and the rows which are in right table but not in left table.

Eno	Ename	Dno
E1	Ram	D1
E2	Varun	D2
E3	Rani	D1
E4	Amrit	

Dno	Dname	Location
D1	IT	Delhi
D2	HR	Hyderabad
D3	Finance	Pune

Join -

4. Right Outer Join –

select Eno, Ename, Dname, Location from Employee Right outer join Department ON Employee.Dno = Department.Dno;

Eno	Ename	Dname	Location
E1	Ram	IT	Delhi
E3	Rani	IT	Delhi
E2	Varun	HR	Hyderabad
NULL	NULL	Finance	Pune

Join -

5. Self Join – In which the table is join with itself.

Study

S_id	C_id	Since
S1	C1	2016
S2	C2	2017
S1	C2	2017

Find Student id who is Enrolled in at least two courses.

Join -

S_id	C_id	Since	S_id	C_id	Since
S1	C1	2016	S1	C1	2016
S1	C1	2016	S2	C2	2017
S1	C1	2016	S1	C2	2017
S2	C2	2017	S1	C1	2016
S2	C2	2017	S2	C2	2017
S2	C2	2017	S1	C2	2017
S1	C2	2017	S1	C1	2016
S1	C2	2017	S2	C2	2017
S1	C2	2017	S1	C2	2017

Join -

S_id	C_id	Since	S_id	C_id	Since
S1	C1	2016	S1	C2	2017
S1	C2	2017	S1	C1	2016

```
Select T1.S_id from Study as T1, Study as T2
Where T1.S_id = T2.S_id
and
T1.C_id <>T2.C_id;
```

Tuple Variables -

- Tuple is a row in a table.
- A field or attribute of a table is a column.
- SQL allows us to define an alias for each occurrence of Relation i.e. Tuple Variable.
- A Tuple Variable is defined in the FROM clause by placing it after the name of the relation separated by space.
- Tuple variables are most useful for comparing two tuples in the same relation.

Ordering of Tuple -

- SQL allows the user to control the order in which tuples are displayed. **order by makes tuples appear in sorted order (ascending order by default). desc specifies descending order. asc specifies ascending order**.
- Syntax –

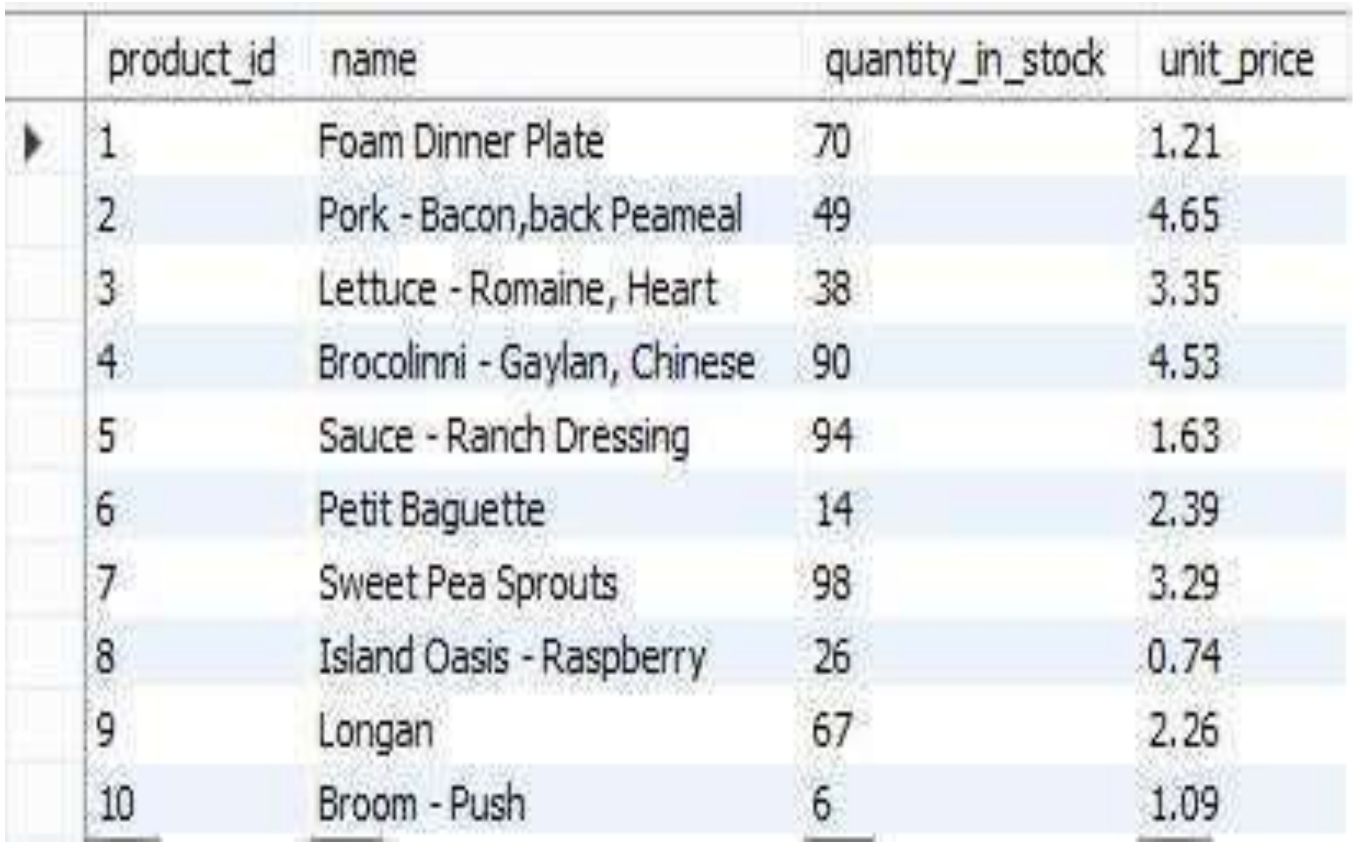
```
SELECT col1, col2,….,coln
FROM table_name
ORDER BY col1, col2,….ASC/DESC;
```

- Example – SELECT * FROM Students ORDER BY Roll_No DESC;

Aggregate Function -

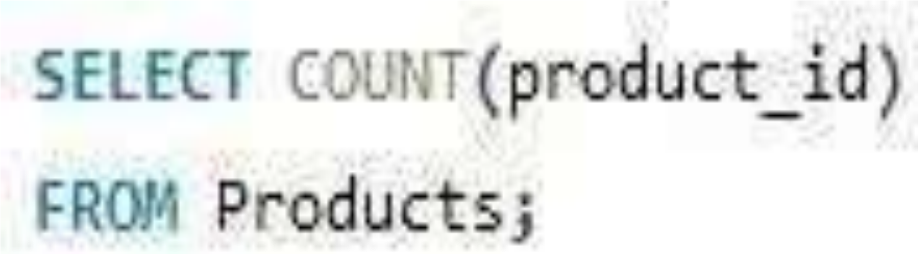
- An aggregate function in SQL **performs a calculation on multiple values and returns a single value**.
- SQL provides many aggregate functions that include avg, count, sum, min, max, etc.
- An aggregate function ignores NULL values when it performs the calculation, except for the count function.

Aggregate Function -



product_id	name	quantity_in_stock	unit_price
1	Foam Dinner Plate	70	1.21
2	Pork - Bacon,back Peameal	49	4.65
3	Lettuce - Romaine, Heart	38	3.35
4	Brocolinni - Gaylan, Chinese	90	4.53
5	Sauce - Ranch Dressing	94	1.63
6	Petit Baguette	14	2.39
7	Sweet Pea Sprouts	98	3.29
8	Island Oasis - Raspberry	26	0.74
9	Longan	67	2.26
10	Broom - Push	6	1.09

Aggregate Function -



```
SELECT COUNT(product_id)
FROM Products;
```

Nested Query -

- Writing a query inside another query is known as nested query or subquery.
- The inner query gets executed first, then the output of inner query is given as input to outer query.

Emp_No	EName	HireDate	Salary	DeptNo
1	King	17/11/2000	5000	10
2	Blake	5/1/2001	2500	30
3	Clark	6/2/2001	2500	20
4	Jones	4/3/2002	3000	20
5	Smith	5/5/2002	3000	30
6	James	6/5/2002	2000	10

Nested Query -

- Example –

1. To display records of employees whose salary is more than the salary of SMITH.

Select * from Employee where salary > (select salary from Employee where EName = ‘Smith’;

2. To display records of employees who are Junior to CLARK.

Select * from Employee where HireDate > (select HireDate from Employee where Ename = ‘Clark’;

PL/SQL -

- Stands for Procedural Language extensions to the Structured Query Language.
- It is the combination of SQL along with the procedural features of programming languages.
- It allows declaration of constant and variables, procedures and functions, types and variable of those types and trigger. It can handle exceptions.

PL/SQL Program Structure -

- PL/SQL Block –

```
DECLARE
        Declaration section
BEGIN
        Execution section
EXCEPTION
        Exception section
END;
```

PL/SQL Program Structure -

- PL/SQL Block –

Declaration section – PL/SQL block has declaration section where we declare variables, allocate memory for cursors and define data types.

Execution section – PL/SQL block has an executable section. An executable section starts with the keyword BEGIN and ends with keyword END. The executable section must have one executable statement, even if it is the NULL statement which does nothing.

Exception section – It starts with the keyword EXCEPTION. The exception-handling section is where we catch and handle exceptions raised by the code in the execution section.

- PL/SQL Data Types – Numeric, Character, Date and Time, Boolean.

Stored Procedures -

- Stored procedure is a type of subprogram in PL/SQL block. It is a group of statements that can be called by its name.
- This is a subprogram that does not return a value directly.
- A procedure is created with the CREATE OR REPLACE PROCEDURE statement.

Stored Procedure -

How to pass parameters in procedure:

Three ways to pass parameters in procedure:

- **IN parameters:** The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
- **OUT parameters:** The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
- **INOUT parameters:** The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

PL/SQL Create Procedure -

- **Syntax**

CREATE [OR REPLACE] **PROCEDURE** procedure_name
[(Parameter_Name [IN | OUT | IN OUT] Type [….])]
IS|AS
[declaration_section]
BEGIN
executable_section
[EXCEPTION
exception_section]
END ;
/
Execute Procedure_Name;

Functions -

- Stored function is a named block or subprogram in PL/SQL.
- In PL/SQL, a function takes one or more parameter and returns one value.
- Syntax

```
CREATE or REPLACE Function Function_Name
[(Parameter_Name [IN | OUT | IN OUT ] Type [….])]
Return Datatype
[IS |AS]
[Declaration Section]
BEGIN
[Execution Section]
END;
/
```

Cursors -

- When an SQL statement is processed, Oracle creates a memory area known as context area.
- A Cursor is a pointer to this context area. It contains all information needed for processing the statement.
- In PL/SQL, the context area is controlled by Cursor.
- A cursor contains information of a select statement and the rows of data accessed by it.
- A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time.

Cursors -

There are two types of cursors:

- Implicit Cursors
- Explicit Cursors

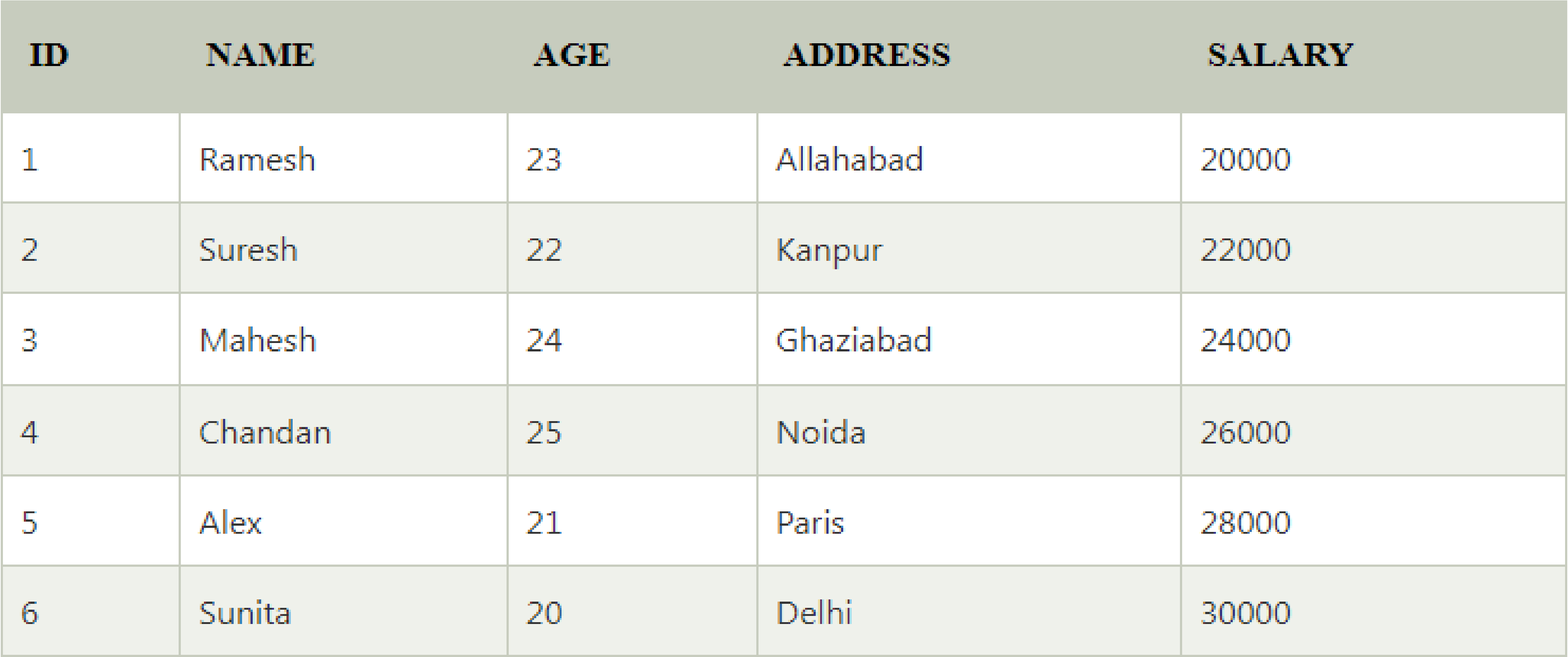
1. PL/SQL Implicit Cursors -

- The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.
- These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

Cursors -

- Orcale provides some attributes known as Implicit cursor's attributes to check the status of DML operations. Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.
- **For example:** When you execute the SQL statements like INSERT, UPDATE, DELETE then the cursor attributes tell whether any rows are affected and how many have been affected. If you run a SELECT INTO statement in PL/SQL block, the implicit cursor attribute can be used to find out whether any row has been returned by the SELECT statement. It will return an error if there no data is selected.

Cursors -



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Cursors -

- Update the table and increase salary of each customer by 5000. Here, SQL%ROWCOUNT attribute is used to determine the number of rows affected:

```
DECLARE
total_rows number(2);
BEGIN
UPDATE  customers
SET salary = salary + 5000;
IF sql%notfound THEN
dbms_output.put_line('no customers updated');
```

Cursors -

```
ELSIF sql%found THEN
total_rows := sql%rowcount;
dbms_output.put_line( total_rows || ' customers updated ');
END IF;
END;
/
```

Output -

```
6 customers updated
PL/SQL procedure successfully completed.
```

Cursors -

2. PL/SQL Explicit Cursors

- The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.
- Syntax for create an explicit cursor -

CURSOR cursor_name **IS** select_statement;

Cursors -

Steps:

You must follow these steps while working with an explicit cursor.

- Declare the cursor to initialize in the memory.
- Open the cursor to allocate memory.
- Fetch the cursor to retrieve data.
- Close the cursor to release allocated memory.

Cursors -

1. Declare the cursor:

- It defines the cursor with a name and the associated SELECT statement.
- **Syntax for explicit cursor declaration:**

  CURSOR cursor_name IS select statement;

2. Open the cursor:

- It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.
- **Syntax for cursor open:**

  OPEN cursor_name;

Cursors -

3. Fetch the cursor –

- It is used to access one row at a time.
- **Syntax for cursor fetch**

FETCH cursor_name **INTO** variable_list;

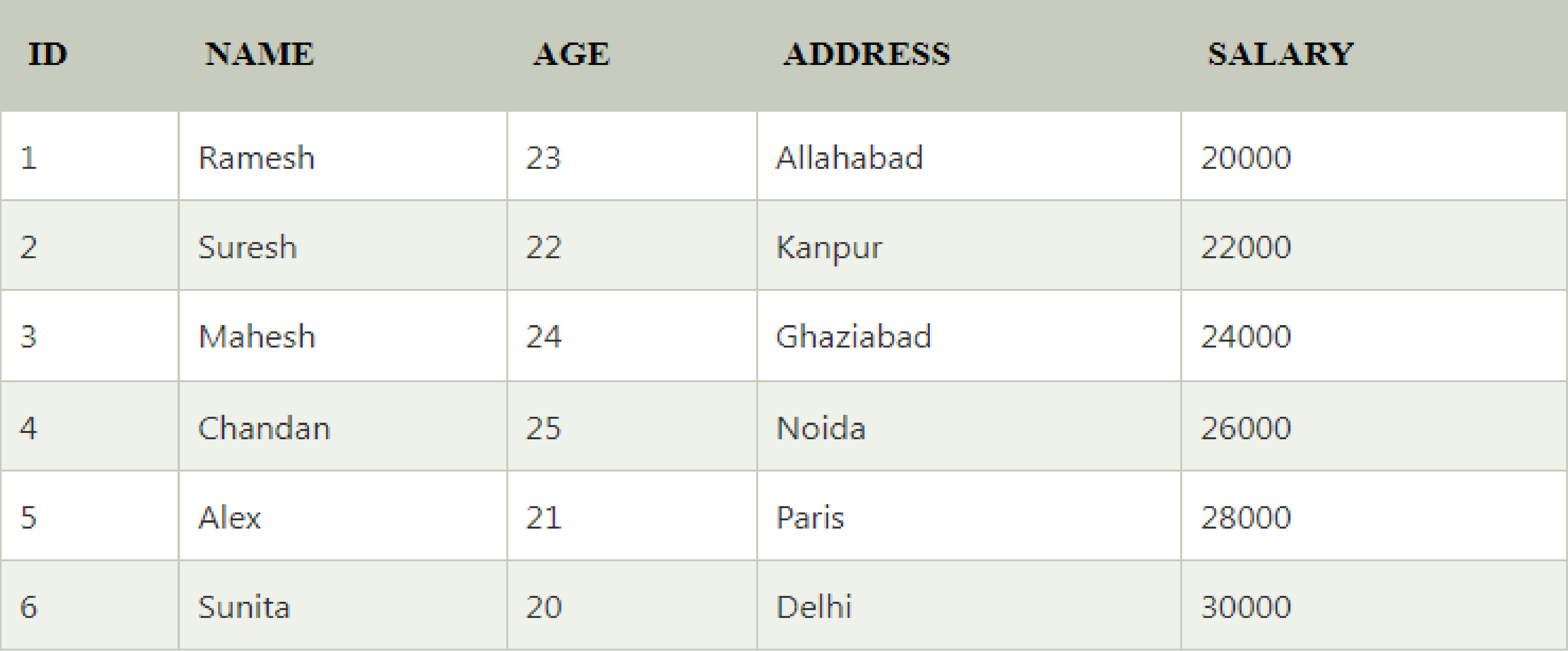
4. Close the cursor:

- It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.
- **Syntax for cursor close:**

Close cursor_name;

Cursors -

Example - Retrieve the name and address from Employee table.



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Cursors -

```
 DECLARE
 CURSOR c_customers is
SELECT name, address FROM customers;
c_name   customers.name%type;
c_addr customers.address%type;
BEGIN
OPEN c_customers;
LOOP
FETCH c_customers into c_name, c_addr;
EXIT WHEN c_customers%notfound;
dbms_output.put_line(c_name || ' ' || c_addr);
END LOOP;
CLOSE c_customers;
END;
/
```

Cursors -

```
Ramesh Allahabad
Suresh Kanpur
Mahesh Ghaziabad
Chandan Noida
Alex Paris
Sunita Delhi
 PL/SQL procedure successfully completed.
```

Trigger -

- Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match.
- Triggers are stored programs, which are automatically executed or fired when some event occurs.
- Triggers are written to be executed in response to any of the following events.

  - A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).

  - A database definition (DDL) statement (CREATE, ALTER, or DROP).

  -A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Creating Trigger -

- **Syntax for creating trigger**

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
Declaration-statements
```

Creating Trigger -

- **Syntax for creating trigger**

```
BEGIN
Executable-statements
EXCEPTION
Exception-handling-statements
END;
/
```

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.

- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Assertion -

- An assertion is a predicate expressing a condition we wish the database to always satisfy.
- When created, the expression must be true.
- DBMS checks the assertion after any change that may violate the expression.
- Syntax –

```
CREATE ASSERTION
<assertion_name>CHECK<predicate>;
```

Assertion -

- Customer(customer_name, customer_street, customer_city)
- If we want that the customer city should not be NULL then the Assertion can be,

```
CREATE ASSERTION city_not_null CHECK
NOT EXIST
(select * from Customer where customer_city is null);
```

Roles and Privileges -

- Roles are names group of privileges that you can assign to users/other roles.
- A privilege is a right allowing the user to run some particular types of SQL commands or access the object of another user. Some of the privileges that are given to users include the rights like connecting to a database or creating a table. There could also be rights to select the rows from the users of another table or execute the stored procedure of another user.

Roles and Privileges -

- Privileges are granted to users in order for them to accomplish the tasks needed for different jobs. Only those privileges should be granted to the user that would allow them to perform the necessary task. Security could be compromised if excessive or unnecessary privileges are granted to a user.
- Privilege is a permission given by DBA.

- Examples of the privileges are –

  Connect to database

  Create a table

  Select a row from another user’s table.

  the right to execute another user's stored procedure

- Types of Privileges –

1. System privilege – allows user to create, alter or drop the database objects such as table, view and so on.
2. Object privilege – allows user to select, insert, update or delete data from database objects.

Roles and Privileges -

Privileges: GRANT and REVOKE

- The user/role can be created in ORACLE using CREATE ROLE command.
- The IDENTIFIED BY clause is used for authentication of the user. It adds the security layer to the role.

```
CREATE ROLE Student
IDENTIFIED BY password1234;
/* Role created*/
//Connect, Username, Password [Error – user Student lacks
CREATE SESSION privilege]
GRANT connect to Student;
/* Grant Succeeded*/
```

Roles and Privileges -

Privileges: GRANT and REVOKE

```
Select * from system.employee; // table doesn’t exist
GRANT select on employee to Student;
/* Grant Succeeded*/
delete from employee where salary>10000; // table doesn’t exist
GRANT delete on employee to Student;
/* Grant Succeeded*/
GRANT all on employee to Student;
/* Grant Succeeded*/
 insert into system.employee values(5, ‘Ram’, 75, ‘Pune’);
```