UNIT 3

(*). DDL (Data Definition language)

① Consists of SQL commands used to define database schema.

② Used to create and modify the structure of DB objects.

→ CREATE: Used to create DB object.

eg. create table student (int id, name varchar(

→ DROP: This command is used to delete objects from DB.

eg. drop table student;

→ ALTER: Used to alter the structure of DB. Sometimes we may require to add additional info, in that case we do not require to create whole DB again.

eg: Alter table student

ADD(columnname1. datatype, columnname2 data

→ TRUNCATE → Used to remove all the records from the table along with the spaces allocated for the records.

Syntax: truncate table student;

→ RENAME → Used to rename an object existing in DB.

Syntax: Alter table student;

Rename to student100;

(*). DQL: (Data Query language)

① Used for performing queries on data.

② It allows getting data from database and imposing order on it.

→ SELECT: It is used to retrieve data from database.

Syntax: select column1, column2 from table_name.

(*). DML: (Data Manipulation language)

① Deals with the manipulation of data present in DB.

→ INSERT: Used to insert data into a table.

Syntax: ① INSERT INTO table_name values (value1, value2,...

② INSERT INTO table_name (column1, column2,...) VALUES(

value2,...

→ UPDATE: It is used to update existing data within a table.
Syntax: update table_name SET column1 = value1, column2 = value2
WHERE condition;
eg. UPDATE student SETNAME = 'PRATIK' WHERE Age = 20;

→ DELETE: It is used to delete Records from DB table.
Syntax: DELETE FROM table_name WHERE condition;
eg: DELETE FROM student WHERE age > 18;

(*). DCL: (Data Control language):
① Includes commands such as GRANT & REVOKE that mainly deals with rights, permissions, controls, etc. of DB system.
→ GRANT: Gives user access privilages to DB.
Syntax: GRANT privilage_name ON objectname TO user_name;
privilages → SELECT, Insert, Delete, Create, Alter, Drop, Update, etc.

eg: GRANT SELECT ON Users TO 'Amit'@'localhost';
GRANT SELECT, INSERT, DELETE ON Users To 'Amit'@'localhost';
To grant all privilages: GRANT ALL ON Users To 'Amit'@'localhost';
To grant privilage to all the users, we use '*' before username.
eg. GRANT SELECT ON Users TO [redacted]'*'@'localhost';

→ REVOKE: This command withdraws the user's privilages. It does operations opposite to GRANT command.
Syntax: revoke privilage_name ON object_name.

Example GRANT INSERT, SELECT on Users To Ram
REVOKE INSERT, SELECT on Users To Ram.

(*) TCL (Transaction Control Language)

→ Commit: Commits a transaction. The commit command saves all the transactions to the DB since the last commit or rollback command.

Syntax: COMMIT;

→ Rollback: RollBacks a transaction in case any error occurs. If any error occurs, all changes need to be aborted. This process of reverting changes is called RollBack.

Syntax: ROLLBACK;

→ Savepoint: Sets a save point within a transaction. A savepoint is a point in the transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.

Syntax: Savepoint savepoint_name;

Example: Savepoint SP1;
Delete from Student where age = 20;
savepoint SP2;

(*) SQL Constraints:

① Used to specify rules for the data in the table.
② Used to limit the type of data that can go in a table.
③ It ensures accuracy and reliability of data in a table.
④ Constraints can be column level or table level. Column level apply to a column and table level apply to a table.

→ NOT NULL: Ensures that a column cannot have a NULL value.
Example: Creat table Student (Id int NOT NULL, Name varchar(10) NOT NU

→ UNIQUE: Ensures that all the values in the column are different
Eg: Create table Student(id int NOT NULL UNIQUE, Name varchar(10) NOT NU

→ Primary KEY: Combination of NOT NULL and UNIQUE. Uniquely identifies each row in a table.
Eg. Create table Student(id int NOT NULL PRIMARY KEY, Name varchar(10) NOT N

→ FOREIGN KEY: Prevents actions that would destroy links b/w tables.
eg. Create table Orders (OrderID int NOT Null, OrderNo int NOT NULL,
PersonID int, Primary Key(OrderID) foreign key(PersonID)
REFERENCES(PersonID));

→ CHECK: Ensures that the values in a column satisfies the given conditions.
Eg. Create table person (ID int, Name varchar(10),
Age int, CHECK (Age >= 18));

→ DEFAULT: Sets a default value for a column if not specified.
eg. Create table person(ID int, name varchar(10), city varchar(25)
DEFAULT 'Sandnes');

→ CREATE INDEX: Used to create and retrieve data from DB very quickly.
Eg: Create index idx_name ON person (Name);

(*). Aggregate functions in SQL: Values of multiple rows are grouped together as an IP to form a single value of more significant meaning.
Various aggregate functions are: Count(), Sum(), Avg(), Min(), Max()

eg. Table: Employee

Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

→ Count(*): Returns total no. of Records → 6
→ Count(Salary): Returns no. of non null values of column Salary → 5
→ Count(Distinct Salary): Returns no. of distinct non Null salaries → 4.
→ Sum(Salary): Sum of all salaries → 310.
→ Avg(Salary): Average of all salaries → 310/5
→ Min(Salary): Min of except Null, ie → 40
→ Max(Salary): → 80.

(v) Built-in functions

① String

ASCII - Returns ASCII value of specific character.
CHAR - Returns character based on ASCII code.
CHARINDEX - Returns position of a substring in a string.
CONCAT / CONCAT with + - Adds two or more strings together.
CONCAT_WS - Adds two or more strings with seperator
DATALENGTH - Returns the no. of Bytes.
LEFT - Extracts a no. of characters from string (Starting from left)
LEN - Returns length of string.
LOWER - Converts the string to lower case
LTRIM - Removes leading spaces.
PATINDEX - Returns position of pattern in a string.
REPLACE - Replace a particular substring with another substring.
REVERSE - Reverse a string
RIGHT - Extracts a no. of characters from string (Starting from Right)
RTRIM - Removes trailing spaces.
TRIM - Removes leading & trailing spaces.
UPPER - Converts a string to upper case.

② Math / Numeric functions

ABS - Returns absolute value of a no.
ACOS - Returns arc cosine.
ASIN - Returns arc sine
ATAN - Returns arc Tangent
ATN2 - Returns arc tangent of 2 numbers.
AVG - Returns Average value

CEILING → Returns smallest integer value.
COUNT - Returns the no. of Records
COS - Returns cosine of a number
COT - Returns cotangent of a number.
DEGREES → Converts a radian value to degrees.
FLOOR - Returns largest integer value.
LOG - Returns natural logarithm.
MAX, MIN, POWER, SQUARE, SQRT, etc.

② · DATE functions:

CURRENT_TIMESTAMP → Returns current date & time.
Dateadd - Adds a date to another date & returns the date
datediff → Returns the date difference b/w 2 dates.
datefromparts → Returns a date from specified parts (year, month, day values)
datename → Returns a specified part of date (as String)
datepart → Returns a specified part of date (as integer)
DAY → Returns day of the month.
Getdate → Returns the current DB system date and time.
ISDATE: Checks an expression returns 1 if valid, else 0.
MONTH → Returns the month part.
Year → Returns the year part.

(*) - SQL Set operations: Used to combine 2 or more SQL Select commands.

① · Union ② · Union All ③ Intersect ④ · Minus.

→ UNION → Used to combine result of 2 or more SQL SELECT queries.
Syntax: Select column-name from table1
UNION
select column-name from table2;
→ It removes duplicate sets.

→ Union All: Returns result without removing duplicate sets.

Syntax: Select column_name from table1
UNION ALL
Select column_name from table2;

→ Intersect: Returns common rows from both the select statements.
No of datatypes and columns must be same.
Syntax: Same as UNION.

→ Minus: Display the rows that are present in first Query but absent in 2nd Query.
Syntax: Same.

(*). SubQueries, Co-related Sub-Queries:
① Corelated Sub-Queries are used for row-by-row processing.
Eg: Find all the employees who earn more than avg salary in their department.

```
select name, salary, FROM employees, WHERE salary
                    (select AVG(salary) from
                     employees);
```

(*). Nested SubQueries: The inner select Query runs first and executes once, returning values to be used by the main Query.

(*). Views:
① Views in SQL are considered as a virtual table.
It also contains rows and columns.
② To create a view, we can select fields from one or more tables present in DB.

Table 1 (Student_details)

id	Name	Address
1	A	E
2	B	F
3	C	G
4	D	H

eg:

Table 2 (Student_marks)

Id	Name	Marks	Age
1	A	10	19
2	B	9	20
3	C	8	18
4	D	7	17
5	I	6	20

Creating View: Syntax:

1. Create view view_name AS
2. Select column1, column2...
3. From table_name
4. where condition;

Example.

1. Create view detailview AS
   Select Name, Address
   from student_details
   where id<4;
   Select * from detailview

O/P →

Name	Address
A	E
B	F
C	G

Query from 2 tables:

Create VIEW Marksview AS
Select student_detail.Name, student_detail.address, Student_marks.Marks
FROM student_detail, Student_Marks
where Student_detail.Name = Student_Marks.name.
Select * from MarksView.

(*). Delete View: Using DROP VIEW statement

Syntax: DROP VIEW view_name;

(*). PL/SQL Concepts:

① Block Structured language that has multiple blocks in it.

② The programs are logical blocks that can contain any number of sub blocks.

③ Stands for Procedular language Extension of SQL.

④ PL/SQL is not case sensitive. We are free to use upper/lower case letters except within string & character literals.

⑤ A line of PL/SQL contains → Delimeters
Identifiers
Literals
Comments.

(*). PL/SQL Concepts → Stored Procedure: It is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

→ Procedure contains a header & a body.

Head → Contains name of procedure & parameters passed to procedure.

Body → Contains declaration section, execution section and exception section.

How to Pass parameters in procedure?

3 ways:

(a). In parameters: It can be referenced by the procedure / function. The value of parameter cannot be overwritten.

(b). Out parameter: Can not be referenced by procedure / function but the values of parameter can be overwritten.

(c). In Out parameter: Can be referenced by procedure / function & values can be overwritten.

2023/3/1

Syntax for creating Procedure:

```
or replace.
create Procedure Procedure_name
IS          → declaration section
BEGIN       → executable section.
END.
```

eg: Create or replace procedure Insertuser(id IN NUMBER, NAME IN varchar(2))

```
IS/AS
BEGIN
Insert into user values(id, name);
END;
```

(*) PL/SQL Program to call procedure:

```
EXECUTE Insertuser(101, 'Rahul');
```

```
BEGIN
Insertuser(101, 'Rahul')
END;
```

→

ID	Name
101	Rahul

→ PL/SQL Drop Procedure: Syntax: DROP PROCEDURE procedure_name;

(*) Executing a Standalone Procedure: Standalone procedure can be called in 2 ways: ⓐ. Using EXECUTE keyword.

ⓑ Calling the name of Procedure from PL/SQL block

eg: name of procedure → Greetings: Using execute keyword it can be called as:

```
EXECUTE greetings;
```

ⓑ Calling from another PL/SQL block; →

```
BEGIN
Greetings;
END;
```

Unit-3

* PL/SQL Concepts Functions

→ Similar to PL/SQL procedure.

→ main difference betn procedure and function is, a function must always return a value, on the other hand a procedure may or may not return a value.

→ Syntax to create a function :-

```
1. CREATE [OR REPLACE ] FUNCTION function_name [parameter
2. [(parameter_name [IN | OUT | IN|OUT ] type [,...])]
3. RETURN return_datatype
4. {IS|AS}
5. BEGIN
6. < function_body>
7. END [function_name];
```

* Here,

→ function_name => specifies name of the function

→ OR REPLACE => allows modifying existing function

→ IN => that value will be passed from outside

→ OUT => that this parameter will be used to return a value outside of the procedure.

→ RETURN => data type you are going to return from the function.

→ function body => consists the executable part.

→ AS keyword is used instead of IS keyword for creating standalone function.

e.g,

```
Create or replace function adder (n1 in number,
                                  n2 in number)
Return number
Is
n3 number (8);
```

```
Begin
n3 := n1 + n2;
 Return n3;
End;
/ write another program to call a function.
DECLARE
 n3 number(2);
BEGIN
 n3 := adder(11,22);
 dbms_output.put_line('Addition is :' ||n3);
END;
/
```

Output :
Addition is : 33

* PL/SQL Recursive Functions :

→ A program or sub-program may call another sub-program. When a sub-program call itself, it is referred as recursive call and the process is known as recursion.

→ e.g., Calculate Factorial of a Number

```
DECLARE
 num number;
 factorial number;

FUNCTION fact (x number)
RETURN number
IS
 f number;
BEGIN
 IF x=0 THEN
  f:=1
```

```
ELSE
f := x * fact (x-1);
END IF;
RETURN f;
END;

BEGIN
num := 6;
factorial := fact (num);
dbms_output.put_line ('Factorial ' || num || ' is ' || factorial);
END;
```

Output: Factorial of 6 is 720

* PL/SQL concepts - Cursors :-

→ When an SQL statement is processed, a memory area called context area is created.

→ A cursor is a pointer to this context area, which contains all the info. for processing the statement

→ A cursor is used to refer to a program to fetch and process the rows returned by the SQL statements, one at a time.

→ There are 2-types of cursors :-
① Implicit ② Explicit

① PL/SQL Implicit Cursors :-

→ These are generated by default to process the statements when DML statements like INSERT, UPDATE and DELETE etc are executed

e.g

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

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

Output :- 6 customers updated.

② PL/SQL Explicit Cursors :-

→ Defined by the programmers to gain more control over the context area.

→ It is created on a SELECT statement which returns more than one row.

→ Syntax :-

```
CURSOR cursor_name IS select_statement;
```

→ Steps :-

1. Declare cursor to initialize in the memory.
2. Open the cursor to allocate memory.
3. Fetch the cursor to retrieve data.
4. Close the cursor to release allocate memory.

SYNTAX

→ Declare :-	CURSOR name IS SELECT statement ;
Open the cursor	OPEN cursor_name ;
Fetch the cursor	FETCH cursor_name INTO variable_list ;
Close the cursor	CLOSE cursor_name.

e.g.

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

```
1. DECLARE
2. C_id customers.id%type;
3. C_name customers.name%type;
4. C_addr customers.address%type;
5 CURSOR c_customers is
6.  SELECT id, name, address FROM customers;
```

```
7.  BEGIN
8.  OPEN C_customers;
9.  LOOP
10. FETCH C_customers into C_id, C_name, C_addr;
11. EXIT WHEN C_customers%.not found;
12. dbms_output.put_line (C_id || ' ' || C_name || ' ' || C_addr);
13. END LOOP;
14. CLOSE C_customers;
15. END;
```

o/p :- 1. Ramesh Allahabad
2. Suresh Kanpur
3. Mahesh Ghaziabad
4. Chandan Noida
5. Alex Paris
6. Sunita Delhi

* PL/SQL Concepts - Exceptional Handling

→ An error occurs during the program execution is called exception in PL/SQL.

→ PL/SQL facilitates programmers to catch such cond$ using exception block in the program and an appropriate action is taken against the error condition.

→ 2 - types of exceptions :
① System - defined Exceptions
② User - defined Exceptions.

* Syntax :-

```
1. DECLARE
2.   < declarations section >
3. BEGIN
```

```
4.  <executable commands (S)>
5.  EXCEPTION
6.  <exception handling goes here>
7.  WHEN exception 1 THEN
      exception 1- handling -statements
    ------
8.  WHEN exception 3
    other THEN
9.    exception 3 handling statements
10. END;
```

* PL/SQL concepts - Triggers

→ Trigger is invoked automatically when a specific event occurs.

→ Trigger is stored into database and invoked repeatedly, when specific condition match.

→ Triggers are written to be executed to any of the following events.

① A DML statement [DELETE, INSERT, UPDATE]

② A DDL " [CREATE, ALTER, DROP]

③ A Database operation.

→ Trigger could be defined on table, view, schema or database with which event is associated.

* Adv.

1. Generates some derived columns values automatically
2. Auditing
3. Synchronous replication of tables
4. Imposing security authorizations.
5. Preventing invalid transactions.
6. Integrity.

Syntax:

```
CREATE [OR REPLACE ] TRIGGER trigger-name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE }
[OF col-name]
ON table-name
[REFFERENCING OLD AS o NEW AS n ]
[FOR EACH ROW]
WHEN ( condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
 EXCEPTION
   Exception-handling statements
END;
```

Attribute	Description
%FOUND	Returns TRUE ⇒ if DML statements like INSERT, UPDATE, DELETE affect atleast one row or more rows Otherwise FALSE
%NOT FOUND	opposite of %FOUND
%ISOPEN	always returns FALSE for implicit cursors.
%ROWCOUNT	Returns no. of rows affected by DML statements.

* Raising Exceptions :-
→ In case of any internal database errors, exceptions are raised by database server automatically.
→ It also be raised by programmer by using command RAISE.

Syntax :-
```
DECLARE
  exception_name EXCEPTION;
BEGIN
  IF condition THEN
    RAISE exception_name;
  END IF;
EXCEPTION
  WHEN exception_name THEN
  Statement;
END;
```

* PL/SQL User-Defined Exceptions :-
→ Allows user to define their own exceptions according to need of program.
→ A user-defined exception can be raised explicitly using either a RAISE statement or DBMS_STANDARD.RAISE_APPLICATION_ERROR.

→ Syntax :-
```
1. DECLARE
2. my-exception EXCEPTION;
```

* PL/SQL Pre-Defined Exceptions :-
→ These are executed when any database rule is violated by the programs.

e.g. NO_DATA_FOUND is a pre-defined exception which is raised when a ~~select~~ SELECT INTO statement returns no rows.

Some important pre-defined exceptions are

1. ACCESS_INTO_NULL ⇒ when a NULL object is automatically assigned a value
2. CASE_NOT_FOUND
3. COLLECTION_IS_NULL
4. INVALID_CURSOR ⇒ when attempts are made to make a cursor operation that is not allowed.
5. INVALID_NUMBER
6. LOGIN_DENID
7. NO_DADA_FOUND
8. NOT_LOGGED_IN
9. PROGRAM_ERROR
10. STORAGE_ERROR
11. TOO_MANY_ROWS
12. VALUE_ERROR

Joins:

JOINS in SQL are commands which are used to combine rows from two or more tables, based on a related column between those tables. There are predominantly used when a user is trying to extract data from tables which have one-to-many or many-to-many relationships between them.

Why SQL JOIN is used?

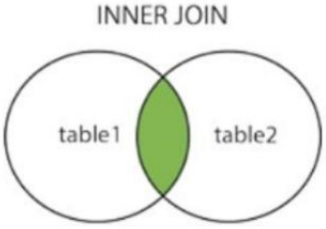
- If you want to access more than one table through a select statement.

- If you want to combine two or more table then SQL JOIN statement is used .it combines rows of that tables in one table and one can retrieve the information by a SELECT statement.
- The joining of two or more tables is based on common field between them.
- SQL INNER JOIN also known as simple join is the most common type of join.

Types of joins:

- Equijoin/innerjoin
- Non-equijoin
- Outer join
- Self join

Equijoin: When the comparison operator is equality, then the join is called an equi-join. Equi-join returns the matching rows from the tables that are being joined.
Equijoins are also called inner join or simple join.



Example:
select employee.emp_id , employee.emp_name , employee.department , department.dept_id , department.location from employee,department where employee.emp_id=department.dept_id;
In this example,
The select clause specifies the column names to retrieve

- emp_id,emp_name,department in employee table
- dept_id,location in department table.

The from clause specifies the two tables that the database must access:

- employee table
- department table

The where clause specifies how the tables are to be joined:
employee.emp_id=department.dept_id here as the dept_id column and emp_id column contain same values so, it must be prefixed by table name to avoid ambiguity.
select employee.emp_id , employee.emp_name , employee.department , department.dept_id , department.location from employee,department where

employee.emp_id=department.dept_id and emp_name='abc';

Non equijoin: A non equi-join is a join condition containing something other than an equality operator i.e the comparison operator is not the equal sign. The relation between employee and grades table is an example of a non equi-join. A relationship between the two tables is that the salary column in employee table must be between the value in the lowsal and highsal column of the grades table. The relationship is obtained using an operator other than (=).

Example:

select e.emp_name,e.salary,g.grade from employee e ,grades g where e.salary between g.low_sal and g.high_sal;

In this example we create non-equi join to find an employee's grade. The salary must be between any pair of the low salary and high salary range. It is important to note that all employees appear exactly once when query is executed. There are two reasons for this:

1. none of the row in the grades table overlaps. That is, the salary value of an employee can lie only between the low_sal and high_sal values of one of the rows in the grades table.
2. All the employees salary lies within the limits provided by the grades table. That is , no employee earns less than the lowest salary value contained in the low_sal column or more than the highest value specified in the high_sal column.

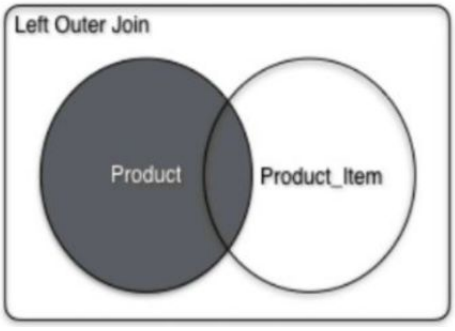
Outer join: When tables are joined , rows which contain matching values in the join predicates, are returned .Sometimes, we want both matching and non matching rows returned for the tables that are being joined. This kind of operation is known as outer join.

You use an outer join to also see rows that do not meet the join condition.

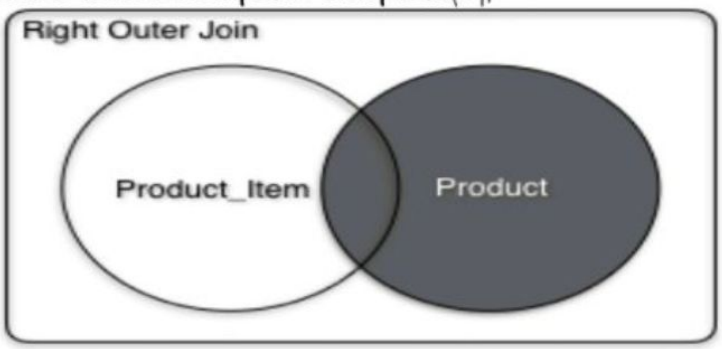
The outer join operator is the plus sign(+).

There are 3 types of Outer Join:

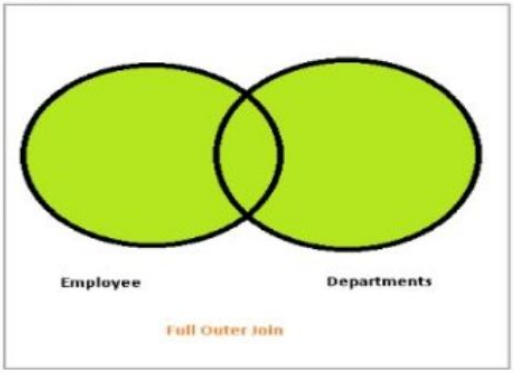
1. Left Outer Join: Returns all the rows from the LEFT table and matching records between both the tables. For example:
   select e.emp_name , d.dept_id , e.department from employee e , department d where e.emp_id(+)= d.dept_id;(left outer join)



2. Right Outer Join: Returns all the rows from the RIGHT table and matching records between both the tables.
   select e.emp_name , d.dept_id , e.department from employee e , department d where e.emp_id= d.dept_id(+);



3. Full Outer Join: It combines the result of the Left Outer Join and Right Outer Join.



Restrictions of outer join: There are some restrictions of outer join. They are:

- The outer join operator can appear only on one side of the expression- the side that has information missing. It returns those rows from the one table
  that has no direct match in the other table.
- A condition involving an outer join cannot use the IN operator or be linked to another condition by the OR operator.

Self joins: A join is called self join when we join a table to itself.To find the name of each employee's manager, we need to join the employee table to itself or perform a self join.
Example: to find the name of the smith's manager we need to:

- find smith in the emp table by looking at the ename column.
- Find the manager number for smith by looking mgr column in emp.
- Find the name of the manager with empno by looking at the ename column.

select worker.last_name || 'works for' || manager.last_name from employee worker, employee manager where worker.manager_id=manager.employee_id;

Types of Views:

There are two types of views, Simple View and Complex View.

Simple View

These views can only contain a single base table or can be created only from one table. Group functions such as MAX(), COUNT(), etc., cannot be used here, and it does not contain groups of data.

By using Simple View, DML operations can be performed. Insert, delete, and update are directly possible, but Simple View does not contain group by, pseudocolumn like rownum, distinct, columns defined by expressions. Simple view also does not include NOT NULL columns from the base tables.

create view empv80 as select emp_id,last_name,salary from employees where dept_id=80;

Complex View

These views can contain more than one base table or can be constructed on more than one base table, and they contain a group by clause, join conditions, an order by clause. Group functions can be used here, and it contains groups of data. Complex views cannot always be used to perform DML operations.

Insert, delete, and update cannot be applied directly on complex views. But unlike Simple Views, Complex Views can contain group by, pseudocolumn like rownum, distinct, columns defined by expressions. NOT NULL columns can be included in complex views while they are not selected by the Simple View.

There are other views, such as Inline View and Materialized View. The inline view is based on a subquery in FROM clause, the subquery creates a temporary table, and this simplifies the complex query.

These views are used to write complex SQL queries without the join and subqueries operations. The materialized view stores the definition and even the data. Replicas of data are created by storing it physically. This view reduces the processing time for regenerating the whole data.

create view dept_sem_v (name,minsal,maxsal,avgsal) as select d.dept_name ,min(e.salary),max(e.salary),avg(e.salary) from employee e department d where e.dept_id=d.dept_id group by d.dept_name;

Differences between Simple and Complex View:

Sr. No.	Key	Simple View	Complex View
1	Definition	Simple View in SQL is the view created by involving only single table. In other words we can say that there is only one base table in case of Simple View in SQL.	On other hand, Complex View is created by involving more than one table i.e., multiple tables get projected in Complex view.
2	Associations	In case of Simple View as only one table is in context hence no major associations need to be applied in case of this view in SQL.	On other hand in case of Complex View multiple tables are in the context hence general associations need to be applied which includes join conditions, a group by clause, a order by clause.
3	Group Functions	In Simple View, due to single table we cannot use group functions like MAX(), COUNT(), etc.	On other hand in case of Complex View due to multiple tables we can use various group functions.
4	Operations allowed	In Simple View, DML operations could easily be performed.	However on other hand in case of Complex view DML operations could not always be performed.
5	Alteration	As mentioned in above point due to DML operations INSERT, DELETE and UPDATE are directly possible.	However on the other hand in case of Complex view, we cannot apply INSERT, DELETE and UPDATE.
6	NULL columns	In Simple View one cannot include NOT NULL columns from base table.	However on other hand in case of Complex view, NOT NULL columns can be included in complex view.