Create Table:

The create table command defines each column of the table uniquely. Each column has a minimum of three attributes a name data type and size. Each table column definition is a single column in the create table syntax. Each table column definition is separated from the other by a comma. Finally the SQL statement is terminated with a semicolon.

Rules of Creating Table:

- A name can have maximum up to 30 characters
- Alphabets from A-Z, a-z and numbers from 0-9 are allowed.
- A name should begin with an alphabet.
- The use of the special character like is allowed & also recommended (#,$).
- SQL reserved words not allowed. (Create, select etc.)

Syntax:

CREATE TABLE <table name>
(Column name1 data type (size), Column name1 data type (size)……);

Eg. Create table Employee (Emp_no varchar2 (10) primary key, Emp_name varcahr2 (15), Emp_city varchar2 (15));

Massage: Table Created.

Finding out the tables created by user:

The command shown below is used to determine the tables to which a user has access. The tables created under the currently selected table space are displayed.

Syntax: SELECT *FROM TAB;

Displaying the table structure:

To display information about the columns defined in a table use the following syntax:

Syntax:

DESCRIBE Table Name;

This command displays the column names, the data types and the special attributes connected to the table.

Example: Desc Employee;

Show the table structure of table Emp.

Modifying the structure of table:

The structure a table can be modified by using by using the ALTER TABLE command ALTER TABLE allows changing the structure of an existing table. With ALTER TABLE it is possible to ADD or DELETE columns, CREATE or DESTROY indexes change the data type of existing columns or rename column of the table itself.

ADDING NEW COLUMN:

Syntax:

ALTER TABLE TableName
ADD (New Column Name Data type Size,..);

Example:

Alter table Emp Add (Commission number (8,2));

Massage: Table Altered.

DROPPING A COLUMN FROM A TABLE:

Dropping any of the columns in entire table

Syntax:

ALTER TABLE Table Name DROP COLUMN Column Name;

Example:

Alter Table Emp Drop Column Commission;

Massage: Table Altered.

MODIFYING EXISTING COLUMNS:

Syntax:

ALTER TABLE Table Name MODIFY (Column Name New Data type Size);

Example:

Alter table Emp Modify (Emp_name Varchar2 (25));

RESTRICTION ON ALTER TABLE

The following tasks can't be performed when using the ALTER TABLE clause:

- Change the name of the table
- Change the name of the column
- Decrease the size of a column if table data exists

Inserting Records into Table:

Once a table is created the most natural things to do is load this table with data to be manipulated latter. When inserting a single row of data into the table the insert operation: [1] create a new row (Empty) in the database table. [2] Load the value passed (by the SQL insert) into the columns specified. [3] Character expressions placed within the insert into statement must be enclosed in single quotes (').

In the INSERT INTO SQL sentence table columns and values have a one to one relationship. Hence in an n INSERT INTO SQL sentence if there are exactly the same numbers of values as there are columns and the values area sequenced in exactly in accordance with the data type of the tables' columns there is no need to indicate the column names. However if there is fewer values begin described than there are columns in the table then it is mandatory to indicate both the table column name and its corresponding value in the INSERT INTO SQL statements.

There are three methods to enter the value in the table.

Method [1]:

INSERT INTO <table name>
(Column1, Column2…ColumnN)Values (value1, value2..valuen);

Eg.

INSERT INTO Employee (emo_no, emp_name) values ('11','abc');

Method [2]

INSERT INTO <table name> Values (value1, value2..valuen);

Eg.

INSERT INTO Employee values ('11','abc');

Method [3]

INSERT INTO <table name>
(&Column1, &Column2…&Column N);

We got prompt for each value.

"&" Sing should be used in string with double quote sign, and number should not allow.

Viewing data in tables:

Once data has been insert into table the next most logical operation would be to vie what has been inserted. The SELECT SQL verb is used to achieve this. This SELECT command is used to retrieve row selected from one or more tables.

If information of a particular client is to be retrieved from a table, its retrieval must be based on a specific condition.

- **Selecting particular columns from table**

Syntax:

Select columnname, columnname from tablename;

Eg.:

Select emp_no,emp_name from emp_master;

- **To select all rows and columns**

Syntax:

Select * from tablename;

Eg.:

Select * from emp_master;

- **To select rows and columns based on some condition**

Syntax:

Select * from tablename where search condition;

Eg.:

Select * from emp_master where emp_no =1;

All rows and all columns

In order to view global table data:

Filtering Table Data:

The ways of filtering table data are:

[1] Selected columns and all rows
[2] Selected Rows and all columns
[3] Selected columns and selected rows

- **Selected columns and all rows:**

The retrieval of specific columns from a table.

Syntax:

SELECT ColumnName1, ColumnName2 FROM Table name;

Example: Select empno, name form emp_master;

The SELECT statement used until now displayed all rows. This is because there was no condition set that informed oracle about how to choose a specific set of rows from any table. Oracle provides the option of using a WHERE Clause in an SQL query to apply a filter pm the rows retrieved.

- **Selected Rows and All Columns:**

Syntax:

SELECT *FROM <Table Name>
WHERE <Condition>;
Example: Select *from emp Where age>=20;

- **Selected columns and Selected Rows:**
  To view a specific set of rows and columns from a table the syntax is:
  SELECT <ColumnName1>, <ColumnName2>FROM <Table Name>
  WHERE <Condition>;
  Example: Select empno, name from emp_master where empno =1;

Renaming Table:

Oracle allows renaming of tables. The rename operation is done atomically, which means that no other thread can access any of the table while the rename process is running.

Syntax: RENAME Table Name TO New Table Name;

Example:

Rename client to product;

Massage: Table Renamed.

Eliminating duplicate rows when using a select statement:

A table could hold duplicate rows, in such a case to view only unique rows the distinct clause can be used.

DISTINCT clause:

The DISTINCT clause allows removing duplicates from the result set. The DISTINCT clause can only be used with select statements. The DISTINCT clause scans through the values of the column/s specified and displays only unique values from amongst them.

Syntax:

SELECT DISTINCT ColumnName1, ColumnName2 FROM Table Name;

Example:

Select Distinct City from emp;

Syntax:

Select Distinct * from tablename;

Example:

Select distinct * from emp_master;

Sorting data in table:

Oracle allows data from a table to be viewed in a sorted order. The rows retrieved from the table will be sorted in either ascending or descending order depending on the condition specified in the SELECT sentence.

Syntax: SELECT * FROM Table Name order by ColumnName1, ColumnName2 Sort Order;

ORDER BY The ORDER BY clause sort the result set based on the columns specified. The ORDER BY clause can only be used in SELECT statement.

Example: Select *from emp order by name;

Select *from emp order by name DESC;

Delete Operations:

The DELETE command deletes row from the table that satisfied the condition provided by its where clause and returns the number of records deleted.

The verb DELETE in SQL is used to remove either: All the rows from a table, A set of rows from a table.

Remove of all Rows:

Syntax: DELETE FROM Table Name;

Example: Delete from emp;

Removal of Specific Row(s):

Syntax: DELETE FROM Table Name where condition;

Example: Delete from emp Where age>=30;

Removal of specific row(s) based on the data held by the other TABLE:

DELETE FROM table name WHERE EXISTS (SELECT Column1 FROM Table Name AND Table Name. <Column Name>=<Expression>);

Example

Delete from Dept Where id=(Select Emp_no from emp where Emp_no=1001);

Updating the content of table:

The UPDATE command is used to change or modify data values in a table. The verb update in SQL is used to either update:

- All the rows from a table
- A select set of rows from a table.

The UPDATE statement updates columns in the existing table's row with new values. The SET clause indicates which column data should be modified and the new values that they should hold. The WHERE clause If given, specifies which rows should be updated. Otherwise all table rows are updated.

Syntax: UPDATE Table Name SET ColumnName1=Expr1, ColumnName2=Expr2;

Example:

Update Emp Set Salary=2000;

Update Record Conditionally:

Syntax:

UPDATE Table Name SET ColumnName1=Expr1, ColumnName2=Expr2

WHERE Condition;

Example:

Update Emp Set Salary=15000 Where Emp_name='abc';

Truncating Table:

TRUNCATE TABLE empties a table completely. Logically this is equivalent to a DELETE statement that deletes all rows, but there are practical difference user some circumstances. The difference between TRUNCATE, DELETE

- Truncate operations drop and re-create the table, which is must faster than deleting rows one of one.
- Truncate operations are not transaction-self
- The number of deleted rows is not returned.

Syntax:

TRUNCATE TABLE<Table Name>;

Example:

Truncate Emp;

DESTROYING TABLE

Sometimes tables within a particular database become obsolete and need to be discarded. In such situation using the DROP TABLE statement with the table name can destroy a specific table.

Syntax:

DROP TABLE <Table Name>;

Example:

Drop table Emp;

OPERATORS:

Arithmetic operator:

- Addition (+): For addition operation.
- Subtraction (-): For subtraction operation.
- Division (/): For division operation.
- Multiplication (*): For multiplication operation.
- Exponentiation (**): For exponentiation operation.
- Enclosed operator (): For Enclosed operation

Eg. Select no, no*no, no*no*no from temp;

By creating an alias

After doing arithmetic operation if we give name to result column that is alias.

Eg.

Select no, no*no square no*no*no cube from temp;

Logical Operator:

AND operator

The oracle engine will process all rows in a table and display the result only when all of conditions specified using the AND operator are satisfied.

Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE; otherwise returns UNKNOWN.

Syntax:

Select column name, column name from table name
Where condition and condition;

Eg.

SELECT * FROM EMP WHERE job='CLERK' AND deptno=10

Or operator:

The oracle engine will process all rows in a table and display the result only when any of the conditions specified using the Or operator are satisfied.

SELECT * FROM emp WHERE job='CLERK' OR deptno=10

Syntax:

Select column name, column name from table name
Where condition or condition;

Eg.

```
Select product_no, name, sell_price
From product_master
Where profit_percent >= 10 or profit_percent <= 20
```

NOT operator:

The oracle engine will process all rows in a table and display the result only when none of the conditions specified using the not operator are satisfied.

Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN

Syntax:

```
Select column name, column name from table name
Where not (condition);
```

Eg.

```
SELECT * FROM EMP WHERE NOT (sal BETWEEN 1000 AND 2000)
```

Range Searching:

Between

In order to select data that is within a range of values, the BETWEEN operator is used. The between operator allows the selection of rows that contain values within a specified lower and upper limit. The range coded after the word between is inclusive.

The lower value must be coded first. The two values in between the range must be linked with the keyword AND. A between operator can be used with both character and numeric data types. However, one cannot mix the data types i.e. the lower value of a range of values from a character column and the other from a numeric column.

Syntax:

```
Select column name, column name from table name
Where column name between lower limit and upper limit;
```

Eg.

```
Select * from product_master
Where profit_percent between 10 and 20;
```

Not between

```
Select * from product_master
Where profit_percent not between 10 and 20;
```

Pattern Matching:

Like Predicate:

The like predicate allows for a comparison of one string value with another string value, which is not identical. This is achieved by using wildcard characters.

That are,

The percent sign (%) matches any string

The underscore (_) matches any single character

Examples:

- Select * from client_master where name like 'ja%';
  Retrieve all information about clients whose names begin with the letters 'ja' from client master.
- Select * from client_master where name like '_r%' or name like '_h%';
  Retrieve all information about clients where the second characters of names are either 'r' or 'h'.

In predicate:

The arithmetic operator (=) compares a single value to another single value. In case a value needs to be compared to a list of values then the in predicate is used. One can check a single value against multiple values by using the in predicate.

Example

Select * from client_master where no in ('c001','c002','c003');

Not In predicate:

The not in predicate is the opposite of in predicate. This will select all the rows where values do not match all of the values in the list.

Example

Select * from client_master where no not in ('c001','c002','c003');

All predicate:

All returns true if all the result of a sub query meet the condition. It can written with the operators like != ,<,< etc.

Example:

Select empno, ename, deptno from emp where deptno > all (12,23,34)

Any predicate:

Any is used to compare column value from the query to the data return by the sub query.

Example:

Select name, city from student
where rollno > any(12,23,34)

Types of data constraint:

There are two types of data constraint that can be applied top data begin inserted into a oracle table.

- I/O constraint
- Business rule constraint.

I/O Constraint:

The I/O data constraints are further divided into two distinctly different constraints:

Oracle allows programmers to define constraints at:

1) Column level
2) Table level

Column level constraints:

If data constraints are defined along with the column definition when creating or altering a table structure the are column level constraints.

Table level constraints:

If data constraints are defined after defining all the table columns when creating or altering table structure is a table level constraints.

The Primary key constraint:

A primary key is one or more column(s) in a table used to uniquely identify each row in the table. A primary key column in a table has special attributes.

- It defines the column as a mandatory column (can't left blank) as not null attributes in active.
- The data held across the column must be UNIQUE.

A single column primary key is called **Simple key**. A multicolumn primary key is called a **composite primary key**. The only function of a primary key in a table is to uniquely identify a row. When a record cannot be uniquely identified using a value in a simple key, a composite key must be defined. A primary key can be defined in either a CREAT TABLE statement or an ALTER TABLE statement.

Feature of Primary key:

- Primary key is a column or a set of columns that uniquely identified a row. It s means purpose is the Record uniqueness.
- Primary key will not allow duplicate values
- Primary key will also not allow null values
- Primary key is not compulsory but it is recommended.
- Primary key helps to identify one record from another record and also helps in relating table with one another.
- Primary key cannot be LONG, LONG RAW data type.
- Only one primary key is allowed per table.
- Unique index is created automatically if there is a primary key.
- One table can combine up to 16 columns in a composite primary key.

A primary key can be defined in either a create table statement or an alter table statement.

Column level primary key:

Syntax: <Column Name> <Data type> <Size> primary key

Example:

```
Create table Employee (Emp_no varchar2 (10) Primary key, name varchar2 (15));
```

Table level primary key:

```
Create table Employee
(Emp_no varchar2 (10), name varchar2 (15), primary key (Emp_no));
```

The Foreign key Constraint:

Foreign key represent relationships between tables. A foreign key is a column (or group of column) whose values are derived from the primary key or unique key of some other table. The table in which the foreign key is defined is called a foreign table or detail table. From which foreign key is called the primary table or master table. A foreign key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Feature of foreign key:

- Foreign key is a column that references a column of a table and it can be the same table also.
- Parent that is being referenced has to be unique or primary key.
- Parent that is being referenced has to be unique or primary key.
- Child may have duplicates and nulls but unless it is specified.
- Foreign key constraint can be specified on child but not on parent.
- Parent record can be deleted provided no child records exist.

- Master table cannot be updated if child records exist.

This constraint established a relationship between records across a master and a detail table. This relationship ensures,

1) Record cannot be inserted into a detail table if corresponding record s in the master table do not exist.

2) Records of the master table cannot be deleted if corresponding records in the detail table actually exist.

Column Level:

Syntax:

<Column Name> <Data type><size> REFERENCES <Table Name>[<column Name>] [ON DELETE CASCADE]

Example:

Create table Department (ID varchar2 (10) primary key, Dept_name varchar2 (15));

Create table employee (EID varchar2 (10) REFERENCES Department (ID), Emp_name varchar2 (15));

Business rules constraints:

Oracle allows the application of business rules to table columns. Business manager determine business rule. They vary from system to system as mentioned earlier. These rules are applied to data; prior the data is being inserted into table columns. This ensures that the data in the table have integrity.

Business rules can be implemented in oracle by using CHECK constraints. Check constraint can be bound to a column or a table using the create table or alter table command.

Command of SQL:

NULL VALUE:

Often there may be records in a table that do not have values for every field. This could be because the information is not available at the time of data entry or because the field is not applicable in every case. If the column was created as null able. Oracle will place a NULL value in the column in the absence of a user-defined value. A NULL value is different from a blank or a zero. A NULL value can be inserted into column of any data type.

Principle of NULL values:

- Setting NULL values is appropriate when the actual value is unknown or when a value would not be meaning full
- A NULL value is not equivalent to a value of zero if the data type is number and in not equivalent to spaces if the data type is character.
- A NULL value will evaluate to NULL in any expression.
- NULL value can be inserted into columns of any data type.
- If the column has a null value oracle ignores any unique, foreign key, check constraint that may be attached to the column.

NOT NULL:

In addition to primary key and foreign key oracle has NOT NULL as column constraint. The NOT NULL as column constraint, the NOT NULL column constraint ensures that a table column cannot be left blank. When a column is defined as not NULL, then that column becomes a mandatory column. The NOT NULL constraint can only be applied at column level.

Syntax:

Create Table Cust_mstr
(cust_no varcahr2(5), Cust_name varchar2(25) NOT NULL, Salary number (8.2));

CHECK Constraint:

Business rule validations can be applied to a table column by using CHECK constraint. CHECK constraint must be specified as a logical expression that evaluates either to TRUE or FALSE

Syntax: <Column name> <data type><size> check (logical expression>;

Example:

Create table Cust_mstr
(Cust_no varcahr2 (5) CHECK (cust_no like 'C%'), Name varchar2 CHECK (Name=UPPER (Name)));

Data value being inserted into the column cust_no must be start with the capital C.
Data values being inserted into the column name should be upper case only.

Limitation of CHECK Constraint:

- The Condition must be a Boolean expression that can be evaluated using the values in the row being inserted or updated.
- The condition cannot contain sub queries or sequences.
- The condition cannot include the SYSDATE, UID, USER or USERENV SQL functions.

DEFAULT VALUES CONCEPT:

At the time of table creation a default values can be assigned to a column. When record is loaded into the table and the column is left empty, the oracle engine will automatically load this column with the default value specified. The data type of the default value should match the data type of the column. The default clause can be used to specify a default value a column

Syntax:

Column name data type (size) default (value)

Example:

Create table student (roll_no number (2), result varchar2 (4) default ('pass'));

UNIQUE KEY Constraint:

The unique column constraint permits multiple entries of NULL into the column. These NULL values are clubbed at the top of the column in the order in which they were entered into the table. This is the essential difference between the primary key and the unique constraint when applied to table column (s).

- Unique key will not allow duplicate value.
- Unique index is created automatically.
- A table can have more then one unique key, which is not possible in primary key.
- Unique key can combine up to 16 columns in a composite unique key.
- Unique key cannot be LONG or LONG RAW data type.

Syntax: <Column Name> <Data type> <Size> UNIQUE

Example:

Create table Employee
(Emp_no varchar2 (15) unique, Name varchar2 (15));

Unique key defined at the table level:

Syntax:

Create table <Table name>
(<Column name1><Data type><Size>, <Column name2><Data type><Size>,
UNIQUE ((<Column name1><Data type><Size>,(<Column name1><Data type><Size>);

Example:

Create table Employee
(Emp_no varchar2 (15), Name varchar2 (15), Salary number(10,2) UNIQUE (Emp_no));

Assigning user defined names to constraints:

When constraints are defined, oracle assigns a unique name to each constraint. The convention used by oracle is, sys_cn where n is a numeric value that makes the constraint name unique. For example sys_c003456.

Referring to the constraints by its name can drop a constraint.

Syntax: constraint <constraint name> <constraint definition>

Example:

Create table student_master(roll_no varchar2(5) constraint p_roll primary key,
Name varchar2(20) not null);

Defining constraints in the alter table command:

Examples:

1) Add a primary key data constraint on the column supplier_no in the supplier_master
   Alter table supplier_master add primary key (supplier_no);
2) Add foreign key constraint on the column oreder_no belonging to the table sales_order which references table order_detail. Modify column client_name to not null.
   Alter table sales_order add constraint order_fkey
   Foreign key (order_no) references order_detail
   Modify (client_name varchar2 (20) not null);

Dropping constraints in the alter table command:

Alter table supplier_master drop primary key;
Alter table sales_order drop constraint order_fkey;

Manipulating dates in sql:

A column of data type date is always displayed in a default format, which is' dd-mon-yy'. If this default format is not used when entering data into a column of the date data type oracle rejects the data and rejects the data and returns an error message. For that to_char and to_date oracle functions are available.

To_char(date[,fmt])

To_char('23-dec-05','dd/mm/yy')

To_date(char [,fmt])

To_date('23/dec/05','dd/mm/yy')

One can use mm, m, day, year, yy, yyyy, month, hh:mm:ss etc format.

Use of th in to_char function:

Select to_char (sysdate,'ddth-mm-yy') from dual;

Output: 12th-05-06

Use of sp in to_char function:

Select to_char (sysdate,'ddsp-mm-yy') from dual;

Output: twelve-05-06

Use of psth in to_char function:

Select to_char (sysdate,'ddspth-mm-yy') from dual;

Output: fourth-06-07

GROUP BY CLAUSE:

The group by clause is another section of the select statement. This optional clause tells oracle to group rows based on distinct values that list for specified columns. The Group By clause creates a data set, containing several sets of records grouped to gather based on a condition.

Syntax:

SELECT<Column1>,<Column2>,<ColumnN>, AGGREGATE_Function<Expr>
FROM TableName WHERE <Condtion> GROUP BY <Column1>,<ColumnN);

Example:

Select Emp_no, avg(salary) From Employee
Group By emp_no;

HAVING CLAUSE:

The having clause can be used in conjunction with the GROUP BY clause. HAVING imposes a condition on the GROUP BY clause. Which further filters the groups created by the group by clause. Each column specification specified in the HAVING clause must occur within a statistical function or must occur in the lust of columns named in the GROUP BY clause.

Syntax:

SELECT<Column1>,<Column2>,<ColumnN>, AGGREGATE_Function<Expr>
FROM TableName WHERE <Condtion> GROUP BY <Column1>,<ColumnN);
Having <condition>;

Example: Select emp_no, avg (salary)from emp
Group By emp_no HAVING emp_no = e01;

ORDER BY CLAUSE:

Oracle allows data from a table to be viewed in a sorted order. The rows retrieved from the table will be sorted in either ASCENDING or DESCENDING order depending on the condition specified in the SELECT statement.

Syntax:

SELECT *FROM <TableName>
ORDER BY <Column Name1>,<ColumnName2> <Sort Order>;

Example:

Select *from from Employee Order by Fname;
Select *from from Employee Order by Fname asc;

Note : it will take default Ascending Order.

Example: Descending

Example: Select *from Employee Order by Name Desc;

SUBQUERIES:

A SUBQUERY is a form of an SQL statement that appears inside another SQL statement. It is also termed as nested query. The statement containing subqueries, the parent statement users the rows returned by the subquery.

It can be used for following:

- To insert records in a target table
- To create tables and insert records in the table created.
- To update records in a target table
- To create views
- To provide value for condition in where, having, in and so on used with Select, Update & Delete statement.

Examples:

1) Select order_no,order_date from sales_order
   Where client_no =(select client_no from client_master where name = 'Rahul');
2) Select order_no, order_date from sales_order
   Where coient_no in (select client_no from client_master whre city = 'Rajkot'
   Or city = 'Jetpur')
   Inside it we can also use any or, and, not, in, not in, group by etc.
3) update student_master
   set name = (select name from student_detail) where roll_no = 3
4) create table student_detail(r_no,name,address) as select
   roll_no, name, add1 from student_master;
5) insert into student_detail select roll_no, name, address from student_master;

Join:

Some time it is to work with multiple tables as through they were a single entity. Then a single SQL sentence can manipulate data from all the tables. Join's are used to achieve this. Tables are joined on columns that have the same data type & data width in the table.

Tables in a database can be related to each other with keys. A primary key is a column with a unique value for each row. The purpose is to bind data together, across tables, without repeating all of the data in every table.

Types of Join:

[1] Inner join [2] Outer Join: [a] Left Join [b] Right Join, [3] Cross join [4] Self Join

[1] Inner Join(Equi join):

Inner joins are also known as equi joins. There are the most common join used in SQL*Plus. They are known as equi joins because the WHERE statement generally compare two columns from two tables with the equivalence operator =. This type of join is by far the most commonly used. In fact many system use this type as the default join. This type of join can be used in situations where selecting any those rows that have values in common in the columns specified, is required. In short, the inner join returns all rows from both tables where there is a match.

Examples:

1) Select order_no, sales_order.product_no, product_name
   From product_master, sales_order
   Where product_master.product_no = sales_order.product_no;

2) Select order_no, sales_order.product_no, product_name, exp_date
   From product_master, sales_order

Where product_master.product_no = sales_order.product_no
Order by exp_date;

3) Select order_no, sales_order.product_no, product_name, sum(sell_price)
From product_master, sales_order
Where product_master.product_no = sales_order.product_no
Group by order_no, sales_order.product_no, product_name;

[2]Outer Join:

Outer joins are similar to inner joins. But give a bit more flexibility when selecting data from related tables. This type of join can be used in situations where it is desired. To select all rows from the table on the left (Or right, both) regardless of whether the other table has values in common and enter NULL where data is missing.

Types

- **Left outer Join:**
  All the fields from left relation will come in output along with matching fields from right relation. The fields from right relation which are not matching are padded with null.

Eg.

Select product_master.pro_no, pname, order_date from product_master, sales_order Where product_master.pro_no = sales_order.pro_no(+);

- **Right Outer Join:**
  All the fields from right relation will come in output along with matching fields from left relation. The fields from left relation which are not matching are padded with null.

Eg.

Select product_master.pro_no, pname, order_date from product_master, sales_order Where product_master.pro_no = sales_order.pro_no(+);

[3] Cross Join:

A cross joins returns what is known as a Cartesian product. This means that the join combines every row from the left table with every row in the right table. This kind of join is usually not preferred as it many run for a very long time & produce a huge result set that may not be useful. Join without where condition.

Eg.

Select deptno, empno from employee, dept;

[4] Self join:

In some situations it may necessary to join the table itself as though you were joining two tables separately this is referred to as self join. In self join two rows from the same table combine to form a result row.

To join a table to itself two copies of very same table opened in memory. Hence in the from clause table has to mentioned twice. Since both table names are same the result will overwrite the first table so we have to open table under alias. Now these table aliases will cause two tables to be opened in different memory locations.

From table name [alias1] table name [alias2]

Example:

Select first.order_no, first.client_no, first.s_no
From sales_order first, sales_order second
Where first.client_no = second.client_no and first.s_no <> second.s_no;

Using the Union, Intersect, Minus clause

Union clause:

Multiple queries can be put together and their output combined using the union clause. The union clause merges the output of tow or more queries into a single set of rows and columns.

The union clause picks up the common records as well as the individual records in both queries.

Restrictions:

- Number of columns in all the queries should be same.
- The data type of the columns in each query must be same.
- Unions cannot be used in sub queries.
- Aggregate functions cannot be used with union clause.

Ex:

```
Select salesman_no “ID”, name from salesman_master where city = ‘rajkot’
Union  Select client_no “ID”, name from client_master where city = ‘rajkot’;
```

Intersect clause:

The intersect clause outputs only rows produced by both the queries intersected i.e. the output in an intersect clause will include only those rows that are retrieved by both the queries or we can say common records from both queries.

Ex.

```
Select salesman_no, name from salesman_master where city = ‘rajkot’
Intersect
Select salesman_master.salesman_no, name from salesman_master,
sales_order  Where salesman_master.salesman_no = sales_order.salesman_no;
```

Minus clause:

The minus clause outputs the rows produced by the first query, after filtering the rows retrieved by the second query.

Ex.

```
Select product_no from product_master
Minus
Select product_no from sales_order;
```

The oracle table Dual:

Dual is a small oracle worktable, which consists of only one row and one column and contains the value x in that column. Besides arithmetic calculations, it also supports date retrieval and it’s formatting.

Often a simple calculation needs to be done, for example, 2*2. The only SQL verb to cause an output is SELECT. However a select must have table name in its from clause, otherwise the select fails.

To facilitate such calculations via a select oracle provides a dummy table called DUAL, against which select statements that are required to manipulate numeric literals can be fired, and output obtained.

Eg.
```
Select 2*4 from dual;
Select sysdate from dual;
```

Oracle functions:

1) Group functions (Aggregate Functions)
2) Scalar functions (Single row functions)

Group functions (aggregate functions)

Functions that act on a set of values are called as group functions.

1) AVG

Syntax: AVG ([DISTINCT/ALL] n)
Purpose: Returns average value of 'n'. Ignoring null values.
Example: select AVG (sellprice) "Average" from product_master;
Output:

```
Average
----------
1234
```

2) Min

Syntax: MIN ([DISTINCT/ALL] expr)
Purpose: Returns minimum value
Example: select min(no) "Minumum No" from client;
Output:

```
Minumum No
---------------
          1
```

2) Max

Syntax: MAX ([DISTINCT/ALL] expr)
Purpose: Returns maximum value
Example: select min (no) "Maximum No" from client;
Output:

```
Maximum No
---------------
         10
```

3) Count (expr)

Syntax: COUNT ([DISTINCT/ALL] expr)
Purpose: Returns no of rows where expr is not null
Example: select count (no) "Total No" from client;
Output:

```
  Total No
---------------
         14
```

4) Count (*)

Syntax: COUNT (*)
Purpose: Returns no of rows in a table including duplicates and Those with nulls.
Example: select count (*) "Total No" from client;
Output:

```
  Total No
---------------
         15
```

5) Sum

Syntax: SUM ([DISTINCT/ALL] n)
Purpose: Returns sum of n values
Example: select sum (no) "Sum" from client;
Output:

```
       Sum
---------------
       140
```

Scalar functions:

Numeric functions:

1) Abs

Syntax: ABS (n)
Purpose: Returns absolute value of 'n'
Example: select abs (-12) "Absolute" from dual;
Output:

```
  Absolute
----------
        12
```

2) Power

Syntax: POWER (m,n)
Purpose: Returns m raised to nth power. N must be an integer
Example: select power (3,2) "Raised" from dual;
Output:

```
   Raised
----------
         9
```

3) Round

Syntax: ROUND (n [,m])
Purpose: Returns n rounded to m places right of the decimal point. If 'm' is ommited n is rounded to 0 places. 'm' can be negative to round off digits left of the decimal point. 'm' must be an integer.
Example: select round (12.35,1) "Round" from dual;
Output:

```
     Round
----------
      12.4
```

4) Sqrt

Syntax: SQRT (n)
Purpose: Returns square root of 'n'. if n<0 null.
Example: select sqrt (4) "Square root" from dual;
Output:

```
  Square root
-------------
           2
```

String functions:

1) Lower

Syntax: LOWER (char)
Purpose: Returns Char with all letters in lower case.
Example: select lower ('ABC') "Lower" from dual;
Output:

```
     Lower
----------
       abc
```

2) Upper

Syntax: UPPER (char)
Purpose: Returns Char with all letters in upper case.
Example: select lower ('abc') "Upper" from dual;
Output:

```
     Upper
----------
       ABC
```

3) Initcap

Syntax: INITCAP (char)
Purpose: Returns string with first char in upper case
Example: select lower ('how r') "Lower" from dual;
Output:

```
     Lower
----------
```

How R

4) Substr

Syntax: SUBSTR (char, m [, n])
Purpose: Returns a portion of char, beginning at character 'm', exceeding upto 'n' characters. If n is omitted, result is returned upto the end char. The first position of char is 1
Example: select substr ('secure', 3 , 4) "Substring" from dual;
Output:
```
Substring
---------
cure
```

5) Length

Syntax: LENGTH (char)
Purpose: Returns length of the string
Example: select substr ('secure') "Length" from dual;
Output:
```
Length
---------
6
```

6) Ltrim

Syntax: LTRIM (char [, set])
Purpose: Removes characters from the left of char with initial characters removed up to the first character not in set. Set is optional default to space.
Example: select ltrim ('Nisha','N') "Ltrim example" from dual;
Output:
```
Ltrim example
---------
isha
```

7) Rtrim

Syntax:: RTRIM (char [, set])
Purpose: Removes char, with final characters removed after the last character not in the set. 'set' is optional and default is space.
Example: select rtrim ('Sunila','a') "Rtrim example" from dual;
Output:
```
Rtrim example
---------
Sunil
```

8) Lpad

Syntax:: LPAD (char1, n [, char2])
Purpose: Returns char1 left padded to length n with the sequence of characters in char2. char2 defaults to blanks.
Example: select lpad ('page1', 10,'*') "Lpad example" from dual;
Output:
```
Lpad example
---------
****page1
```

9) Rpad

Syntax: RPAD (char1, n [, char2])
Purpose: Returns char1 right padded to length n with the sequence of characters in char2.char2 defaults to blanks.
Example: select rpad ('page1', 10,'x') "Rpad example" from dual;
Output:
```
Rpad example
---------
Page1xxxx
```

10) Instr

Syntax:: Instr (string, char, [start,[positon]])

Purpose: Returns the specified char is at which position. If start is not specified then search is start from first position if specified then search will start from that. Position shows repeat of any char.

Example: select Instr ('hello','h',1,1)from dual;

Output:

```
      Instr
----------
         1
```

Conversion functions:

1) To_number

Syntax:: TO_NUMBER (char)

Purpose: Converts char , a character value containing a number, to a value of number data type.

Example: update product_master set dell_price = sell_price + To_number (substr ('$1000', 2 , 3));

2) To_char

Syntax:: TO_CHAR (n [, fmt])

Purpose: Converts a value of number datatype to a value of char datatype, using the optional format string. It accepts a number and a numeric format in which the number has to appear. If fmt is omitted n is converted to a char value exactly long enough to hold significant digits.

Example: select to_char (17145,'$099,999') "char" from dual;

Output:

```
char
---------------
 $017,145
```

3) To_char

Syntax:: TO_CHAR (date [, fmt])

Purpose: Converts a value of date datatype to a value of char. It accepts a date as well as the format in which the date has to appear. 'fmt' must be a date format. If fmt is omitted, date is converted into a character value in the default date format, i.e. 'DD-MON-YY'.

Example: select to_char (sysdate,'Month,dd,yy) "New format" from dual;

Output:

```
New format
---------------
December,26,06
```

4) To_date

Syntax: TO_DATE (char [, fmt])

Purpose Converts a character field to a date field.

Example: insert into sales_order (order_no, order_date) Values('O001', to_date('30-dec-06 10:55 a.m.', 'dd-mon-yy hh:mm a.m.') from dual;

Output:

```
   to_date(
---------------
 01:23
```

Date functions

1) Add_months Syntax: ADD_MONTHS (d, n)
Purpose: Returns date after adding the number of months specified with the function.
Example: select add_months(sysdate,4) from dual;
Output: Add_months

26-apr-07

2) Last_day Syntax: LAST_DAY (d)
Purpose: Returns the last date of the month specified with the function.
Example: select last_day (sysdate) from dual;
Output: last_day

31-dec-06

3) Months_between Syntax: MONTHS_BETWEEN (d1, d2)
Purpose: returns the number of months between date1 and date2.
Example: select months_between ('12-dec-05','23-may-06) from dual;
Output : MONTHS_BETWEEN('12-DEC-05','23-MAY-6')

-5.354839

4). Next_day Syntax: NEXT_DAY (date,char)
Purpose: Returns the the date of the first week day named by char that is after the date named by date. Char must be a day of the week.
Example: select next_day('15-sep-87','tuesday') from dual;
Output: next_day

16-sep-87

Other functions:

1) Translate () Syntax: Translate (st1, st2, st3)
Purpose: Returns the main string st1 by replacing the chars specified in st2 with st3. It replaces single char at a time.
Example: select translate ('12345','24','bb') from dual;
Output: Translate

1b3b5

2) Replace () Syntax: Replace (st1, st2, st3)
Purpose: Returns the main string st1 by replacing char or chars in a string with one or more chars.
Example: select replace ('jack and jue', 'j', 'bl') from dual;
Output: Replace
---
black and blue

4) Chr ()

Syntax: Chr (number)

Purpose: Returns the main string st1 by replacing char or chars in a string with one or more chars.

Example: select chr(65) || chr(66) || chr(67) "character" from dual

Output: cha

---

abc

5) Greatest ()

Syntax: greatest (v1, v2,…vn)

Purpose: Returns the largest value among the given values in function.

Example: select greatest (12, 23, 34) from dual;

Output: greatest

34

6) Least ()

Syntax: Least (v1, v2,…vn)

Purpose: Returns the least value among the given values in function.

Example: select least (12, 23, 34) from dual;

Output: greatest

12

****