SQL

SQL-Structured Query Language

- **SQL (Structured Query Language):-**

It is a computer language aimed to store, manipulate data stored in relational databases.

Some of the basic functions of SQL are inputting, modifying, and dropping data from databases.

Data types in SQL

Data type	Description
CHARACTER(n)	Character string, fixed length n.
CHARACTER VARYING(n) or VARCHAR(n)	Variable length character string, maximum length n.
BINARY(n)	Fixed length binary string, maximum length n.
BINARY VARYING(n) or VARBINARY(n)	Variable length binary string, maximum length n.
INTEGER(p)	Integer numerical, precision p.
SMALLINT	Integer numerical precision 5.
NTEGER	Integer numerical, precision 10.
BIGINT	Integer numerical, precision 19.
DECIMAL(p, s)	Exact numerical, precision p, scale s.

FLOAT(p)	Approximate numerical,mantissa precision p.
REAL	Approximate numerical mantissa precision 7.
FLOAT	Approximate numerical mantissa precision 16.
DOUBLE PRECISION	Approximate numerical mantissa precision 16.
INTERVAL	Composed of a number of integer fields, represents a period of time, depending on the type of interval.
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values

Data Type Abbreviations

Abbreviation	Character
CHAR(n)	CHARACTER(n)
CHAR	CHARACTER
CHAR VARYING(n)	CHARACTER VARYING(n)
VARCHAR(n)	CHARACTER VARYING(n)
VARBINARY(n)	BINARY VARYING(n)
INT(p)	INTEGER(p)
INT	INTEGER
DEC(p, s)	DECIMAL(p, s)
DEC	DECIMAL
NUM(p, s)	NUMERIC(p, s)
NUM	NUMERIC

Operators in SQL

- SQL support 3 types of operators

✔ **Arithmetic Operator:**

- Used for basic calculations like +,-,* & /. It can be used in the expression.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division

SQL Operators

- Next two type of Operators are Comparison Operators and Logical Operators. These operators are used mainly in the WHERE clause, HAVING clause to filter the data to be selected.

✔ **Comparison Operators:**

- Comparison operators are used to compare the column data with specific values in a condition.
- Comparison Operators are also used along with the SELECT statement to filter data based on specific conditions.

Comparison Operators

Comparison Operators	Description
=	equal to
<>, !=	is not equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to

Operators in SQL

✔ Logical Operator:

- Used to evaluate the expression depending on combination of conditions.
- Logical operators are,

Operator	Meaning
AND	Logical AND
OR	Logical OR
NOT	Logical NOT

ALL	**The ALL operator is used to compare a value to all values in another value set.**
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.

NOT	**The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL Comparison Keywords

- There are other comparison keywords available in sql which are used to enhance the search capabilities of a sql query.

- They are "IN", "BETWEEN...AND", "IS NULL", "LIKE".

Comparison Operators	Description
LIKE	column value is similar to specified character(s).
IN	column value is equal to any one of a specified set of values.
BETWEEN...AND	column value is between two values, including the end values specified in the range.
IS NULL	column value does not exist.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL LIKE Operator

- The LIKE operator is used to list all rows in a table whose column values match a specified pattern. It is useful when you want to search rows to match a specific pattern, or when you do not know the entire value. For this purpose we use a character '%'.
- Example: To select all the students whose name begins with ‘A'

```
SELECT first_name, last_name
FROM student_details
WHERE first_name LIKE ‘A%’ ;
```

- The previous select statement searches for all the rows where the first letter of the column first_name is ‘A' and rest of the letters in the name can be any character.

- There is another character you can use with LIKE operator. It is the underscore character, ' _ ' . In a search string, the underscore signifies a single character.

- Example: To display all the names with 'a' second character,
- SELECT first_name, last_name
  FROM student_details
  WHERE first_name LIKE '_a%';

SQL BETWEEN ... AND Operator

- The operator BETWEEN and AND, are used to compare data for a range of values.
- Example: to find the names of the students between age 20 to 25 years, the query would be like,

```
SELECT first_name, last_name, age
FROM student_details
WHERE age BETWEEN 20 AND  25;
```

SQL set operators

- SQL set operators combine results from two or more SELECT statements.
- SQL set operators combine rows from different queries with strong preconditions - all involved SELECTS must.
- They retrieve the same number of columns and the data types of corresponding columns.

- According to SQL Standard there are following Set operator types:

- UNION [DISTINCT];
- UNION ALL;
- EXCEPT [DISTINCT];
- EXCEPT ALL;
- INTERSECT [DISTINCT];
- INTERSECT ALL.

Union and UNION ALL

- In SQL the **UNION** clause combines the results of two SQL queries into a single table of all matching rows. The two queries must result in the same number of columns and compatible data types in order to unite. Any duplicate records are automatically removed unless UNION ALL is used.
- A simple example would be a database having tables sales2005 and sales2006 that have identical structures but are separated because of performance considerations. A UNION query could combine results from both tables.
- UNION does not guarantee the order of rows. Rows from the second operand may appear before, after, or mixed with rows from the first operand. In situations where a specific order is desired, ORDER BY clause must be used.

person	amount
A	1000
B	2000
C	5000

Sales2005

person	amount
A	2000
B	2000
D	35000

Sales2006

person	amount
A	1000
B	2000
C	5000
A	2000
D	35000

SELECT * FROM sales2005 **UNION** SELECT * FROM sales2006;

- SELECT * FROM sales2005 UNION ALL SELECT * FROM sales2006;

person	amount
A	1000
A	2000
B	2000
B	2000
C	5000
D	35000

INTERSECT operator

- The SQL INTERSECT operator takes the results of two queries and returns only rows that appear in both result sets.

- The INTERSECT operator removes duplicate rows from the final result set. The INTERSECT ALL operator does not remove duplicate rows from the final result set.

- The following statement combines the results with the INTERSECT operator, which returns only those rows returned by both queries:
- SELECT product_id FROM inventories **INTERSECT** SELECT product_id FROM order_items;

Minus operator

- The following statement combines results with the MINUS operator, which returns only unique rows returned by the first query but not by the second:

- SELECT product_id FROM inventories **MINUS** SELECT product_id FROM order_items;

EXCEPT operator

- The SQL EXCEPT operator takes the distinct rows of one query and returns the rows that do not appear in a second result set.
- The EXCEPT ALL operator does not remove duplicates.
- For purposes of row elimination and duplicate removal, the EXCEPT operator does not distinguish between NULLs.

The query returns all rows where the Quantity is between 1 and 100, **apart from** rows where the quantity is between 50 and 75.

- SELECT * FROM Orders WHERE Quantity BETWEEN 1 AND 100 EXCEPT SELECT * FROM Orders WHERE Quantity BETWEEN 50 AND 75;

Statement

- **DML**

DML is abbreviation of **Data Manipulation Language**. It is used to retrieve, store, modify, delete, insert and update data in database.

Examples: SELECT, UPDATE, INSERT statements

- **DDL**

DDL is abbreviation of **Data Definition Language**. It is used to create and modify the structure of database objects in database.

Examples: CREATE, ALTER, DROP statements

- **DCL**

DCL is abbreviation of **Data Control Language**. It is used to create roles, permissions. It is also used to control access to database by securing it.

Examples: GRANT, REVOKE statements

- **TCL**

TCL is abbreviation of **Transactional Control Language**. It is used to manage different transactions occurring within a database.

Examples: COMMIT, ROLLBACK statements

What are the difference between DDL, DML and DCL commands

✔ **DDL:- Data Definition Language** (DDL) statements are used to define the database structure or schema. Some examples:

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

- ✔ **DML:-Data Manipulation Language** (DML) statements are used for managing data within schema objects. Some examples:
  - SELECT - retrieve data from the database
  - INSERT - insert data into a table
  - UPDATE - updates existing data within a table
  - DELETE - deletes all records from a table, the space for the records remain
  - MERGE - UPSERT operation (insert or update)

✔ **DCL:-Data Control Language** (DCL) statements.

GRANT - gives user's access privileges to database

REVOKE - withdraw access privileges given with the GRANT command.

✔ **TCL:-Transaction Control** (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

COMMIT - save work done

SAVEPOINT - identify a point in a transaction to which you can later roll back

ROLLBACK - restore database to original since the last COMMIT

Basic commands of DBMS

1. Create table
2. Describe table
3. Insert data into table
4. Select data in table
5. Delete data in table
6. Update table
7. Modifying table structure
8. Rename table
9. Truncate table
10. Delete table
11. View tables created by user

SELECT

- The most commonly used SQL command is SELECT statement. The SQL SELECT statement is used to retrieve data from a table in the database. A query may retrieve information from specified columns or from all of the columns in the table. To create a simple SQL SELECT Statement, you must specify the column(s) name and the table name. The whole query is called SQL SELECT Statement.

- Select queries require two essential parts. The first part is the **"WHAT",** which determines what we want SQL to go and fetch. The second part of any ***SELECT*** command is the **"FROM WHERE".** It identifies where to fetch the data from, which may be from a SQL table.

Where Clause

- SQL offers a feature called WHERE clause, which we can use to restrict the data that is retrieved.

- The condition you provide in the WHERE clause filters the rows retrieved from the table and gives you only those rows which you expected to see.

- WHERE clause can be used along with SELECT, DELETE, UPDATE statements.

- The WHERE clause is used when you want to retrieve specific information from a table excluding other irrelevant data.

- **Example:** To find the name of a student with the specific id then the query would be like:

```
SELECT first_name, last_name FROM
student_details WHERE id = 100;
```

- Comparison Operators and Logical Operators are used in WHERE Clause.

Create table-DDL statement

- The CREATE TABLE Statement is used to create tables to store data.
- Syntax for the CREATE TABLE Statement is:

```
CREATE TABLE table_name
(column_name1 datatype,
column_name2 datatype,
... column_nameN datatype
);
```

- ***table_name*** - is the name of the table.
- ***column_name1, column_name2....*** - is the name of the columns
- ***datatype*** - is the data type for the column like char, date, number etc.

- **For Example:** If you want to create the employee table, the statement would be like,
- CREATE TABLE employee
  (id number(5),
  name char(20),
  dept char(10),
  age number(2),
  salary number(10),
  location char(10)
  );
- In Oracle database, the data type for an integer column is represented as "number".
- Oracle provides another way of creating a table.
- CREATE TABLE temp_employee
  as SELECT * FROM employee
- In the above statement, temp_employee table is created with the same number of columns and data type as employee table.

INSERT –DML statement

- The INSERT Statement is used to add new rows of data to a table. Insert data to a table in two ways,
- **1) Inserting the data directly to a table.**
- **Syntax**

```
INSERT INTO TABLE_NAME
            [ (col1, col2, col3,...colN)]
VALUES (value1, value2, value3,...valueN);
```

- col1, col2,...colN -- the names of the columns in the table into which you want to insert data.

- While inserting a row, if you are adding value for all the columns of the table you need not specify the column(s) name in the sql query. But you need to make sure the order of the values is in the same order as the columns in the table

- The sql insert query will be as follows :
- INSERT INTO TABLE_NAME
  VALUES (value1, value2, value3,...valueN);
- **For Example:** If you want to insert a row to the employee table, the query would be like,

- INSERT INTO employee (id, name, dept, age, salary) VALUES (105, ‘ABC', ‘computer', 27, 33000,);

- When adding a row, only the characters or date values should be enclosed with single quotes.
- If you are inserting data to all the columns, the column names can be omitted. The above insert statement can also be written as,
- INSERT INTO employee
  VALUES (105, ‘ABC', ‘computer', 27, 33000);

Inserting data to a table through a select statement.

- Syntax for SQL INSERT is:

```
INSERT INTO table_name
   [(column1, column2, ... columnN)]
   SELECT column1, column2, ...columnN
   FROM table_name [WHERE condition];
```

- For Example: To insert a row into the employee table from a temporary table, the sql insert query would be like,

```
INSERT INTO employee (id, name, dept, age, salary
   location) SELECT emp_id, emp_name, dept, age,
   salary, location FROM temp_employee;
```

- If you are inserting data to all the columns, the above insert statement can also be written as,

```
INSERT INTO employee
SELECT * FROM temp_employee;
```

- When adding a new row, you should ensure the data type of the value and the column matches
- Follow the integrity constraints, if any, defined for the table.

SQL UPDATE-DML Statement

- The UPDATE Statement is used to modify the existing rows in a table.
- **Syntax:**

```
UPDATE table_name
SET column_name1 = value1,
column_name2 = value2, ...
[WHERE condition]
```

- table_name - the table name which has to be updated.
- column_name1, column_name2.. - the columns that gets changed.
- value1, value2... - are the new values.

- In the Update statement, WHERE clause identifies the rows that get affected. If you do not include the WHERE clause, column values for all the rows get affected.

Continue….

- **For Example:** To update the location of an employee, the sql update query would be like,

  UPDATE employee SET location =‘Pune' WHERE id = 101;

- To change the salaries of all the employees, the query would be,

  UPDATE employee

SET salary = salary + (salary * 0.2);

SQL Delete-DML Statement

- The DELETE Statement is used to delete rows from a table.
- Syntax :

  DELETE FROM table_name [WHERE condition];

- table_name -- the table name which has to be updated.
- The WHERE clause in the sql delete command is optional and it identifies the rows in the column that gets deleted.
- If you do not include the WHERE clause all the rows in the table is deleted, so be careful while writing a DELETE query without WHERE clause.
- **For Example:** To delete an employee with id 100 from the employee table, the sql delete query would be like,

  DELETE FROM employee WHERE id = 100;
- To delete all the rows from the employee table, the query would be like,

  DELETE FROM employee;

SQL TRUNCATE Statement

- The SQL TRUNCATE command is used to delete all the rows from the table and free the space containing the table.
- **Syntax**

  TRUNCATE TABLE table_name;
- **For Example:** To delete all the rows from employee table, the query would be like,

  TRUNCATE TABLE employee;

Difference between DELETE and TRUNCATE Statements

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

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

SQL DROP-DDL Statement:

- The SQL DROP command is used to remove an object from the database. If you drop a table, all the rows in the table is deleted and the table structure is removed from the database.
- **Syntax**

  DROP TABLE table_name;
- **For Example:** To drop the table employee, the query would be like

  DROP TABLE employee;

Difference between DROP and TRUNCATE Statement

- If a table is dropped, all the relationships with other tables will no longer be valid, the integrity constraints will be dropped, grant or access privileges on the table will also be dropped.

- If we want to use the table again it has to be recreated with the integrity constraints, access privileges and the relationships with other tables should be established again.

- But, if a table is truncated, the table structure remains the same, therefore any of the above problems will not exist.

SQL ALTER TABLE-DDL Statement

- The SQL ALTER TABLE command is used to modify the definition (structure) of a table by modifying the definition of its columns.
- The ALTER command is used to perform the following functions.

  1) Add, drop, modify table columns
  2) Add and drop constraints
  3) Enable and Disable constraints

Continue….

- **Syntax to add a column**

  ALTER TABLE table_name ADD column_name datatype;
- **For Example:** To add a column "experience" to the employee table, the query would be like

  ALTER TABLE employee ADD experience number(3);

- **Syntax to drop a column**

  ALTER TABLE table_name DROP column_name;
- **For Example:** To drop the column "location" from the employee table, the query would be like

  ALTER TABLE employee DROP location;

- **Syntax to modify a column**

  ALTER TABLE table_name MODIFY column_name datatype;
- **For Example:** To modify the column salary in the employee table, the query would be like

  ALTER TABLE employee MODIFY salary number(6);

SQL RENAME Command

- The SQL RENAME command is used to change the name of the table or a database object.
- If you change the object's name any reference to the old name will be affected. You have to manually change the old name to the new name in every reference.
- **Syntax to rename a table**

  RENAME old_table_name To new_table_name;

- **For Example:** To change the name of the table employee to my_employee, the query would be like

  RENAME employee TO my_emloyee;

Grant-DCL Statement

- DCL commands are used to enforce database security in a multiple user database environment. Two types of DCL commands are GRANT and REVOKE. Only Database Administrator's or owner's of the database object can provide/remove privileges on a database object.
- **SQL GRANT Command**

  SQL GRANT is a command used to provide access or privileges on the database objects to the users.
- **Syntax is:**

```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

- ***privilege_name*** is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.
- ***object_name*** is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- ***user_name*** is the name of the user to whom an access right is being granted.
- ***PUBLIC*** is used to grant access rights to all users.
- ***ROLES*** are a set of privileges grouped together.
- ***WITH GRANT OPTION*** - allows a user to grant access rights to other users.

Example

GRANT SELECT ON employee TO user1;

- This command grants a SELECT permission on **employee** table to **user1**.

- If you GRANT SELECT privilege on employee table to user1 using the WITH GRANT option, then user1 can GRANT SELECT privilege on employee table to another user, such as user2 etc.

- Later, we can REVOKE the SELECT privilege on employee from user1, still user2 will have SELECT privilege on employee table.

Revoke –DCL Statement

- The REVOKE command removes user access rights or privileges to the database objects.

- **Syntax :**

  REVOKE privilege_name
  ON object_name
  FROM {user_name |PUBLIC |role_name}

Example

REVOKE SELECT ON employee FROM user1;

- This command will REVOKE a SELECT privilege on employee table from user1.
- When you REVOKE SELECT privilege on a table from a user, the user will not be able to SELECT data from that table anymore.
- However, if the user has received SELECT privileges on that table from more than one users, he/she can SELECT from that table until everyone who granted the permission revokes it.
- You cannot REVOKE privileges if they were not initially granted by you.

Privileges and Roles

- **Privileges:** Privileges defines the access rights provided to a user on a database object. There are two types of privileges.

  1) System privileges - This allows the user to CREATE, ALTER, or DROP database objects.

  2) Object privileges - This allows the user to EXECUTE, SELECT, INSERT, UPDATE, or DELETE data from database objects to which the privileges apply.

CREATE system privileges

System Privileges	Description
CREATE object	allows users to create the specified object in their own schema.
CREATE ANY object	allows users to create the specified object in any schema.

Object privileges

Object Privileges	Description
INSERT	allows users to insert rows into a table.
SELECT	allows users to select data from a database object.
UPDATE	allows user to update data in a table.
EXECUTE	allows user to execute a stored procedure or a function.

- **Roles:**

  1) Roles are a collection of privileges or access rights. When there are many users in a database it becomes difficult to grant or revoke privileges to users.

  2) Therefore, if you define roles, you can grant or revoke privileges to users, thereby automatically granting or revoking privileges. You can either create Roles or use the system roles pre-defined by oracle.

Some of the privileges granted

System Role	Privileges Granted to the Role
CONNECT	CREATE TABLE, CREATE VIEW, CREATE SYNONYM, CREATE SEQUENCE, CREATE SESSION etc.
RESOURCE	CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER etc. The primary usage of the RESOURCE role is to restrict access to database objects.
DBA	ALL SYSTEM PRIVILEGES

Creating Roles

- **Syntax :**

CREATE ROLE role_name
[IDENTIFIED BY password];

- **For example:** To create a role called "developer" with password as "pwd", the code will be as follows :

```
CREATE ROLE developer
[IDENTIFIED BY pwd];
```

- It's easier to GRANT or REVOKE privileges to the users through a role rather than assigning a privilege directly to every user.
- If a role is identified by a password, then, when you GRANT or REVOKE privileges to the role, you definitely have to identify it with the password.
- We can GRANT or REVOKE privilege to a role as below:
- **For example:** To grant CREATE TABLE privilege to a user by creating a testing role:
- First, create a testing Role

  CREATE ROLE testing
- Second, grant a CREATE TABLE privilege to the ROLE testing. You can add more privileges to the ROLE.

  GRANT CREATE TABLE TO testing;

Continue….

- Third, grant the role to a user.

  GRANT testing TO user1;

- To revoke a CREATE TABLE privilege from testing ROLE, we can write:

  REVOKE CREATE TABLE FROM testing;

- **The Syntax to drop a role from the database is as below:**

  DROP ROLE role_name;

- **For example:** To drop a role called developer, you can write:

  DROP ROLE developer;

Commit-TCL Statement

Commit:

- Used to end the transaction & also make its effect permanent to the database.
- Deletes or removes the savepoint if any.
- Syntax: commit; **OR** commit work;

Rollback-TCL Statement

Rollback:

- Used to undo work done in current transaction.
- Rollback entire transaction or till particular transaction.
- Syntax; rollback; **OR** rollback work;

Save Point-TCL Statement

Savepoint:

- Like marker which marks the lengthy transactions
- It is used with rollback to cancel effect till particular savepoint
- Syntax: savepoint savepointname;

EXAMPLE:

```
SQL> update emp set salary=salary-700;
SQL> savepoint s1;
SQL> delete from emp where job='Worker';
SQL> savepoint s2;
SQL> rollback to savepoint s1;
SQL> rollback
```

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows and columns
View	Logically represents subsets of data from one or more tables
Sequence	Generates primary key value
Index	Improves the performance of some queries
Synonym	Alternative name for an object

SQL Views

- A VIEW is a virtual table, through which a selective portion of the data from one or more tables can be seen.
- Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity.
- A view is stored as a SELECT statement in the database. DML operations on a view like INSERT, UPDATE, DELETE affects the data in the original table upon which the view is based.

- **The Syntax to create a sql view is**

```
CREATE VIEW view_name
AS
SELECT column_list
FROM table_name [WHERE condition];
```

Continue….

- ***view_name*** is the name of the VIEW.
- The SELECT statement is used to define the columns and rows that you want to display in the view.
- **For Example:** to create a view on the product table the sql query would be like
- CREATE VIEW view_product
  AS
  SELECT product_id, product_name
  FROM product;

- **SQL Updating a View**
- **SQL CREATE OR REPLACE VIEW Syntax**

```
CREATE OR REPLACE VIEW view_name AS
  SELECT column_name(s)
  FROM table_name
  WHERE condition
```

- Now we want to add the "Category" column to the "Current Product List" view. We will update the view with the following SQL:

- ```
  CREATE VIEW Current_Product_List AS
  SELECT ProductID,ProductName,Category
  FROM Products
  ```
  ```

- **SQL Dropping a View**
- You can delete a view with the DROP VIEW command.
- **Syntax:**

  DROP VIEW view_name

Syntax for MySQL

- The following SQL statement defines the "ID" column to be an auto-increment primary key field in the “Employee" table:

- CREATE TABLE Employee
  (
  ID int NOT NULL **AUTO_INCREMENT,**
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255),
  PRIMARY KEY (ID)
)

- MySQL uses the **AUTO_INCREMENT** keyword to perform an auto-increment feature.

- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

- To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

  ALTER TABLE Employee AUTO_INCREMENT=100

- To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):

  INSERT INTO Employee (FirstName,LastName)
VALUES (‘abc',‘xyz')

SQL Index

- Index in sql is created on existing tables to retrieve the rows quickly.
- When there are thousands of records in a table, retrieving information will take a long time.
- Therefore indexes are created on columns which are accessed frequently, so that the information can be retrieved quickly.
- Indexes can be created on a single column or a group of columns.
- When an index is created, it first sorts the data and then it assigns a ROWID for each row.
- **Syntax to create Index:**

```
CREATE INDEX index_name
    ON table_name (column_name1,column_name2...);
```

- **Syntax to create SQL unique Index:**

CREATE UNIQUE INDEX index_name
ON table_name(column_name1);

index_name is the name of the INDEX.

- ***table_name*** is the name of the table to which the indexed column belongs.
- ***column_name1*** *name* of the column on which index is created.
- In Oracle there are two types of SQL index namely, implicit and explicit.

Example

- The SQL statement below creates an index named "PIndex" on the "LastName" column in the "Persons" table:

```
CREATE INDEX PIndex
ON Persons (LastName)
```

- If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX PIndex
ON Persons (LastName, FirstName)
```

- **Implicit Indexes:**

✔ Implicit indexes are indexes that are automatically created by the database server when an object is created.

✔ Indexes are automatically created for primary key constraints and unique constraints.

- **Single-Column Indexes:**
- A single-column index is one that is created based on only one table column.

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

- **Unique Indexes:**
- Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table.

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

- **Composite Indexes:**
- A composite index is an index on two or more columns of a table.

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

- **Explicit Indexes:**

They are created using the "create index.. " syntax.

1) Even though sql indexes are created to access the rows in the table quickly, they slow down DML operations like INSERT, UPDATE, DELETE on the table, because the indexes and tables both are updated along when a DML operation is performed. So use indexes only on columns which are used to search the table frequently.

2) It is not required to create indexes on table which have less data.

3) In oracle database you can define up to sixteen (16) columns in an INDEX.

When should indexes be avoided?

- Indexes should not be used on small tables.
- Tables that have frequent, large batch update or insert operations.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed.

MySQL uses indexes for these operations:

- To find the rows matching a WHERE clause quickly.
- To retrieve rows from other tables when performing joins.

Displaying INDEX Information:

- You can use **SHOW INDEX from table_name**

command to list out all the indexes associated with a table.

DROP INDEX

- An index can be dropped using SQL **DROP** command.
- Care should be taken when dropping an index because performance may be slowed or improved.
- **The basic syntax is as follows:**

```
DROP INDEX index_name;
```

Synonyms

- We can simplify access to objects by creating a synonym(another name for an object).
- Syntax:

CREATE [PUBLIC] SYNONYM synonym

FOR object;

Where,

PUBLIC: creates synonym accessible to all users

SYNONYM: It is the synonym to be created

object: identifies the object for which the synonym is created.

- To refer to a table owned by another user, you need to prefix the table name with the name of the user who created it followed by a period.
- Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table,view,sequence,procedure,or other objects.

Creating synonym

- Create a shortened name for the product_sum

  Create SYNONYM p_sum

  FOR product_sum

The database administrator can create a public synonym accessible to all users.

Removing Synonym

- To drop a synonym use the DROP SYNONYM statement. Only the database administrator can drop a public synonym.

```
DROP PUBLIC SYNONYM p_sum;
```

- Suppose the schemas hr and sh both contain tables named customers.

- In the following example, user SYSTEM creates a PUBLIC synonym named customer1 for hr.customers:

CREATE PUBLIC SYNONYM customer1

FOR hr.customers;

Group Functions

- **SQL GROUP Functions**
- Group functions are built-in SQL functions that operate on groups of rows and return one value for the entire group. These functions are: **COUNT, MAX, MIN, AVG, SUM, DISTINCT**

- **SQL COUNT ():** This function returns the number of rows in the table that satisfies the condition specified in the WHERE condition. If the WHERE condition is not specified, then the query returns the total number of rows in the table.

- **For Example:** If you want the number of employees in a particular department, the query would be:

```
SELECT COUNT (*) FROM employee
WHERE dept = 'Electronics';
```

The output would be '2' rows.

- If you want the total number of employees in all the department, the query would take the form:

```
SELECT COUNT (*) FROM employee;
```

The output would be '5' rows.

Continue…

- **SQL DISTINCT():** This function is used to select the distinct rows.

- **For Example:** If you want to select all distinct department names from employee table, the query would be:

```
SELECT DISTINCT dept FROM employee;
```

- To get the count of employees with unique name, the query would be:

```
SELECT COUNT (DISTINCT name) FROM employee;
```

- **SQL MAX():** This function is used to get the maximum value from a column.

- To get the maximum salary drawn by an employee, the query would be:

```
SELECT MAX (salary) FROM employee;
```

Continue…

- **SQL MIN():** This function is used to get the minimum value from a column.
- To get the minimum salary drawn by an employee, the query would be:

```
SELECT MIN (salary) FROM employee;
```

- **SQL AVG():** This function is used to get the average value of a numeric column.
- To get the average salary, the query would be

```
SELECT AVG (salary) FROM employee;
```

- **SQL SUM():** This function is used to get the sum of a numeric column
- To get the total salary given out to the employees,

```
SELECT SUM (salary) FROM employee;
```

Group by clause

- To divide the rows in a table into groups we can use **Group By** clause. Then we can use the group functions to return summary information for each group.

 Syntax:

SELECT column,group_function(column)
FROM table
[WHERE condition]
[GROUP BY expression]

Expression specifies columns whose values determine the basis for grouping rows

Key points related to group by clause

- If you include a group function in a SELECT clause ,you cannot select individual results as well, unless the individual column appears in the GROUP BY clause.
- Using a WHERE clause, you can exclude rows before dividing them into groups.
- You must include the columns in the GROUP BY clause
- By default, rows are sorted by ascending order of the columns included in the GROUP BY list. You can override this by using OREDER BY clause.

Group By clause

- The SQL GROUP BY Clause is used along with the group functions to retrieve data grouped according to one or more columns.
- **For Example:** Consider following table

Employee_id	Job_id	Salary	Department_id
100	ST_Clerk	20000	90
102	Manager	50000	60
103	Accountant	30000	70
104	Assistant	40000	10
105	ST_Clerk	20000	90
106	Assistant	40000	10

Query

- If you want to know the total amount of salary spent on each department, the query would be:

Example

- SELECT Department_id, SUM (salary)
  FROM employee
  GROUP BY Department_id;

Deparment_id	SUM(salary)
90	40000
60	50000
70	30000
10	80000

- The group by clause should contain all the columns in the select list except those used along with the group functions.

- SELECT Job_id, dept, SUM (salary)
  FROM employee
  GROUP BY Job_id, dept;

Having Clause

- Having clause is used to filter data based on the group functions. This is similar to **WHERE** condition but is used with group functions.
- Group functions cannot be used in WHERE Clause but can be used in **HAVING** clause.

- When WHERE, GROUP BY and HAVING clauses are used together in a SELECT statement, the WHERE clause is processed first, then the rows that are returned after the WHERE clause is executed are grouped based on the GROUP BY clause. Finally, any conditions on the group functions in the HAVING clause are applied to the grouped rows before the final output is displayed

Having clause

- Use the having clause to restrict groups:
- 1) Rows are grouped
- 2)The group function is applied
- 3) Groups matching the HAVING clause are displayed.
- Syntax:

SELECT column, group_function
FROM table
[WHERE condition]
[GROUP BY expression]
[HAVING group condition]
[OREDER BY column];

group condition restricts the groups of rows returned to those groups for which the specified condition is true.

Continue….

- The oracle server performs the following steps when you use the HAVING clause:
  - Rows are grouped.
  - The group function is applied to the group
  - The groups that match the criteria in the HAVING clause are displayed.
- Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.

Query

- If you want to select the department that has total salary paid for its employees more than 35000.

Example1

- SELECT Department_id, SUM (salary)
  FROM employee
  GROUP BY Department_id
  HAVING SUM (salary) > 35000

Deparment_id	SUM(salary)
90	40000
60	50000
10	80000

Query

- Display the department id and average salaries for those departments whose maximum salary is greater than 35000

Example 2

- SELECT Department_id, AVG (salary)
  FROM employee
  GROUP BY Department_id
  HAVING MAX(salary) > 35000

Output

Employee_id	Job_id	Salary	Department_id
100	ST_Clerk	20000	90
102	Manager	50000	60
103	Accountant	30000	70
104	Assistant	40000	10
105	ST_Clerk	20000	90
106	Assistant	40000	10

Deparment_id	SUM(salary)
60	50000
10	80000

Deparment_id	AVG(salary)
60	50000
10	40000

Nesting Group functions

- Display maximum average salary

```
SELECT  MAX(AVG(salary) )
FROM  employee
GROUP BY  Department_id
```

Deparment_id	AVG(salary)
60	50000

SQL ORDER BY

- The ORDER BY clause is used in a SELECT statement to sort results either in ascending or descending order. Oracle sorts query results in ascending order by default.

- **Syntax**
- SELECT column-list
  FROM table_name [WHERE condition]
  [ORDER BY column1 [, column2, .. columnN]
  [ASC|DESC]];

Example

- If you want to sort the employee table by salary of the employee, the sql query would be.
- SELECT Employee_id, salary FROM employee ORDER BY salary;

Employee_id	Salary
100	20000
105	20000
103	30000
104	40000
106	40000
102	50000

END

Sequence

- A sequence-

-Automatically generates unique numbers

-Is typically used to create a primary key value

-Speeds up the efficiency of accessing sequence values when cached in memory.

- They are mainly used to create a primary key value which must be unique for each row.

- The sequence is generated and incremented or decremented by an internal oracle routine.

- This can be a time saving object because it can reduce the amount of application code needed to write a sequence-generating routine.

- Sequence numbers are stored and generated independently of tables.

Creating Sequence

- A sequence is a database item that generates a sequence of integers.
- Syntax of CREATE SEQUENCE statement:

```
CREATE SEQUENCE sequence_name
[START WITH start_n
[INCREMENT BY n]
[ { MAXVALUE maximum_n | NOMAXVALUE } ]
[ { MINVALUE minimum_n | NOMINVALUE } ]
[ { CYCLE | NOCYCLE } ]
[ { CACHE cache_n | NOCACHE } ]
[ { ORDER | NOORDER } ];
```

- The minimum information required for getting numbers using a sequence is:
  - The starting number
  - The maximum number that can be generated by a sequence
  - The increment value for generating the next number.
  - This information is provided to oracle at the time of sequence creation.

where

- The default start_n is 1.
- The default increment number is 1.
- The absolute value of n must be less than the difference between maximum_n and minimum_n
- minimum_n must be less than or equal to start_n, and minimum_n must be less than maximum_n.
- NOMINVALUE specifies the maximum is 1 for an ascending sequence or -10^26 for a descending sequence. NOMINVALUE is the default.
- maximum_n must be greater than or equal to start_n, and maximum_n must be greater than minimum_n.
- NOMAXVALUE specifies the maximum is 10^27 for an ascending sequence or -1 for a descending sequence. NOMAXVALUE is the default.

- CYCLE specifies the sequence generates integers even after reaching its maximum or minimum value.
- When an ascending sequence reaches its maximum value, the next value generated is the minimum.
- When a descending sequence reaches its minimum value, the next value generated is the maximum.

- NOCYCLE specifies the sequence cannot generate any more integers after reaching its maximum or minimum value. NOCYCLE is the default.

- CACHE cache_n specifies the number of integers to keep in memory.
- The default number of integers to cache is 20.
- NOCACHE specifies no integers are to be stored.
- ORDER guarantees the integers are generated in the order of the request.
- NOORDER doesn't guarantee the integers are generated in the order of the request.
- NOORDER is the default.

Example

- The following statement creates the sequence customers_seq.This sequence could be used to provide customer ID numbers when rows are added to the customers table.

```
CREATE SEQUENCE customers_seq
START WITH 1000
 INCREMENT BY 1
 NOCACHE
 NOCYCLE;
```

The first reference to customers_seq.nextval returns 1000. The second returns 1001. Each subsequent reference will return a value 1 greater than the previous reference.

Creating a sequence and then get the next value

- SQL> **CREATE** SEQUENCE test_seq;
  Sequence created.
  SQL> **SELECT** test_seq.nextval **FROM** DUAL;
  NEXTVAL

  1
  SQL> **SELECT** test_seq.nextval **FROM** DUAL;
  NEXTVAL

  2
  SQL> **SELECT** test_seq.nextval **FROM** DUAL;
  NEXTVAL

  3
- SQL> DROP SEQUENCE test_seq;
  Sequence dropped.

Test sequence step

```
SQL> CREATE SEQUENCE test_seq
  START WITH 10 INCREMENT BY 5
 MINVALUE 10 MAXVALUE 20
 CYCLE CACHE 2 ORDER;
Sequence created.
```

-
```
SQL> SELECT test_seq.nextval FROM dual;
  NEXTVAL
----------
        10
SQL> SELECT test_seq.nextval FROM dual;
  NEXTVAL
----------
        15
SQL> SELECT test_seq.nextval FROM dual;
  NEXTVAL
----------
        20
SQL> SELECT test_seq.nextval FROM dual;
  NEXTVAL
----------
        10
```

Referencing a sequence

- Once a sequence is created , SQL can be used to view the values held in its cache.
- Use following SELECT statement to view sequence value :

  SELECT <SequenceName>.NextVal FROM DUAL;
- This will display the next value held in the cache . Every time nextval references a sequence its output is automatically incremented from the old value to the new value ready for use.
- To reference the current value of a sequence:

  SELECT <SequenceName>.CurrVal FROM DUAL;

Altering a sequence

- A sequence once created can be altered. This is achieved by using the ALTER SEQUENCE statement.
- Syntax:

ALTER SEQUENCE <sequence_name >

[INCREMENT BY <Integer value>

MAXVALUE <IntegerValue/NOMAXVALUE]

MINVALUE < Integervalue> /NO MINVALUE

CYCLE/NO CYCLE

CACHE <IntegerValue>/NO CACHE

Example

- **Modifying a Sequence:** This statement sets a new maximum value for the customers_seq sequence,

```
ALTER SEQUENCE customers_seq
MAXVALUE 1500;
```

The following statement turns on CYCLE and CACHE for the customers_seq sequence:

```
ALTER SEQUENCE customers_seq CYCLE CACHE 5;
```

Dropping a sequence

- The DROP sequence is used to remove the sequence from the database.
- Syntax:-

  DROP SEQUENCE <SequenceName>;
- Example:
- The following statement drops the sequence customers_seq.

  DROP SEQUENCE customers_seq;

Query

- Display the job_id and total monthly salary for each job with a total payroll exceeding 25000.

Example 3

- SELECT Job_id, SUM (salary) PAYROLL
  FROM employee
  WHERE Job_id NOT LIKE '%tant%'
  GROUP BY Job_id
  HAVING SUM(salary) > 25000 ;

Job_id	Payroll
ST_Clerk	40000
Manager	50000

Constraints

The oracle server uses constraints to prevent invalid data entry into tables. Constraints prevent the deletion of a table if there are dependencies.

Data integrity Constraints

Constraint	Description
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a foreign key relationship between the column and a column of the referred table.
CHECK	Specifies a condition that must be true

Constraint guidelines

- All constraints are stored in data dictionary.

- Create a constraint either:

  -at the same time as the table is created or

  -after the table has been created.

- We can define a constraint at the column or table level.

❖ Constraints can be defined in two ways

- 1) The constraints can be specified immediately after the column definition. This is called column-level definition. If data constraints are defined as an attribute of a column definition when creating or altering a table, they are column level constraints.

  2) The constraints can be specified after all the columns are defined. This is called table-level definition. If data constraints are defined after defining all table column attributes when creating or altering a table structure, it is a table level constraints.

Defining Constraints

```
CREATE TABLE table_name
(column datatype [column_constraint],
……..
[table_constraint][….]);
```

Example:

```
CREATE  TABLE employee(emp_id  NUMBER(6),
        first_name VARCHAR2(20),
        …….
        job_id  VARCHAR2(10) NOT NULL,
        CONSTRAINT emp_id_pk    PRIMARY KEY
    (emp_id));
```

SQL Not Null Constraint

- This constraint ensures all rows in the table contain a definite value for the column which is specified as not null. Which means a null value is not allowed.
- Syntax:

  [CONSTRAINT constraint_ name] NOT NULL
- Example:
- To create a employee table with Null value, the query would be like,

```
CREATE TABLE employee
( id number(5),
name char(20) CONSTRAINT nm NOT NULL,
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

SQL Unique Key

- This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.
- **Syntax to define a Unique key at column level:**

  [CONSTRAINT constraint_name] UNIQUE

- **Syntax to define a Unique key at table level:**

  [CONSTRAINT constraint_name]
  UNIQUE (column_name)

Example

- To create an employee table with Unique key, the query would be like,

Unique Key at column level:

```
CREATE TABLE employee
   ( id number(5) PRIMARY KEY,
   name char(20),
   dept char(10),
   age number(2),
   salary number(10),
   location char(10) UNIQUE
   );
```

Unique Key at table level:

- CREATE TABLE employee
  (id number(5) PRIMARY KEY,
  name char(20),
  dept char(10),
  age number(2),
  salary number(10),
  location char(10),
  CONSTRAINT loc_un UNIQUE(location)
  );

SQL Primary key

- This constraint defines a column or combination of columns which uniquely identifies each row in the table.
- Syntax to define a Primary key at column level:

column_name datatype
[CONSTRAINT constraint_name] PRIMARY KEY

- Syntax to define a Primary key at table level:

[CONSTRAINT constraint_name] PRIMARY KEY (column_name1,column_name2,..)

column_name1, column_name2 are the names of the columns which define the primary Key.

- The syntax within the bracket i.e. [CONSTRAINT constraint_name] is optional.

- Example: To create an employee table with Primary Key constraint, the query would be like,
- **Primary Key at column level:**

```
CREATE TABLE employee
   ( id number(5) PRIMARY KEY,
   name char(20),
   dept char(10),
   age number(2),
   salary number(10),
   location char(10)
   );
```

- **Primary Key at table level:**

```
CREATE TABLE employee
   ( id number(5),
   name char(20),
   dept char(10),
   age number(2),
   salary number(10),
   location char(10),
   CONSTRAINT emp_id_pk PRIMARY KEY (id)
   );
```

SQL Foreign key or Referential Integrity

- This constraint identifies any column referencing the PRIMARY KEY in another table.
- It establishes a relationship between two columns in the same table or between different tables.
- For a column to be defined as a Foreign Key, it should be defined as a Primary Key in the table which it is referring.
- One or more columns can be defined as Foreign key.

- **Syntax to define a Foreign key at column level:**

[CONSTRAINT constraint_name] REFERENCES Referenced_Table_name(column_name)

- **Syntax to define a Foreign key at table level:**

[CONSTRAINT constraint_name] FOREIGN KEY (column_name) REFERENCES referenced_table_name(column_name);

Example:

- Consider "product" table and "order_items".

Foreign Key at column level:

```
CREATE TABLE product
( product_id number(5) CONSTRAINT pd_id_pk PRIMARY KEY,
    product_name char(20),
    supplier_name char(20),
    unit_price number(10)
    );
```

OR

```
CREATE TABLE order_items
( order_id number(5) CONSTRAINT od_id_pk PRIMARY KEY,
    product_id number(5) CONSTRAINT pd_id_fk
    REFERENCES product(product_id),
    product_name char(20),
    supplier_name char(20),
    unit_price number(10)
    );
```

- **Foreign Key at table level:**

```
CREATE TABLE order_items
   ( order_id number(5) ,
   product_id number(5),
   product_name char(20),
   supplier_name char(20),
   unit_price number(10)
   CONSTRAINT od_id_pk PRIMARY KEY(order_id),
   CONSTRAINT pd_id_fk FOREIGN KEY(product_id)
   REFERENCES product(product_id)
   );
```

- If the employee table has a 'mgr_id' i.e, manager id as a foreign key which references primary key 'id' within the same table, the query would be like,

- CREATE TABLE employee
  (id number(5) PRIMARY KEY,
  name char(20),
  dept char(10),
  age number(2),
  mgr_id number(5) REFERENCES employee(id),
  salary number(10),
  location char(10)
  );

SQL Check Constraint

- The constraint can be applied for a single column or a group of columns.

- **Syntax to define a Check constraint:**

[CONSTRAINT constraint_name] CHECK (condition)

- **Example:** In the employee table to select the gender of a person, the query would be like,

- **Check Constraint at column level:**

```
CREATE TABLE employee
    ( id number(5) PRIMARY KEY,
    name char(20),
    dept char(10),
    age number(2),
    gender char(1) CHECK (gender in ('M','F')),
    salary number(10),
    location char(10)
    );
```

- **Check Constraint at table level:**

```
CREATE TABLE employee
   ( id number(5) PRIMARY KEY,
   name char(20),
   dept char(10),
   age number(2),
   gender char(1),
   salary number(10),
   location char(10),
   CONSTRAINT gender_ck CHECK (gender in ('M','F'))
   );
```

Subquery

- Subquery or Inner query or Nested query is a query in a query. A subquery is usually added in the WHERE clause of the sql statement.
- Most of the time, a subquery is used when you know how to search for a value using a SELECT statement, but do not know the exact value.
- Subqueries are an alternate way of returning data from multiple tables.
- Subqueries can be used with the following sql statements along with the comparison operators like =, <, >, >=, <= etc.(these are considered as single-row operators) while IN,ANY,ALL these are considered as multiple row operators.

- A subquery is a SELECT statement that is embedded in a clause of another SELECT statement.

- The subquery (inner query) executes once before the main query.

- The result of the subquery is used by the main query(outer query).

- They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

- The subquery can place in a number of SQL clauses, including:

✔ WHERE clause
✔ HAVING clause
✔ FROM clause

Guidelines for using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition
- Syntax:

```
SELECT select_list
FROM  table
WHERE expr operator (SELECT select_list
                     FROM table);
```

Operator includes a comparison condition such as >,=,or IN

Example: student_details;

id	first_name	last_name	age	subject	games
100	abc	gg	20	Science	Cricket
101	xyz	pp	22	Maths	Football
102	pqr	zz	29	Science	Cricket
103	lmn	aa	28	Maths	Badminton
104	def	bb	25	History	Chess

- **For Example:**
- 1) Usually, a subquery should return only one record, but sometimes it can also return multiple records when used with operators like IN, NOT IN in the where clause. The query would be like,

```
SELECT first_name, last_name, subject
FROM student_details
WHERE games NOT IN ('Cricket', 'Football');
```

- The output would be similar to:

first_name	last_name	subject
lmn	aa	Badminton
def	bb	Chess

- 2) If you know the name of the students who are studying science subject, you can get their id's by using following query:

```
SELECT id, first_name
FROM student_details
WHERE first_name IN (‘abc', ‘pqr');
```

- But, if you do not know their names, then to get their id's you need to write the query like this,

```
SELECT id, first_name
FROM student_details
WHERE first_name IN (SELECT first_name
                     FROM student_details
                     WHERE subject= 'Science');
```

id	first_name
100	abc
102	pqr

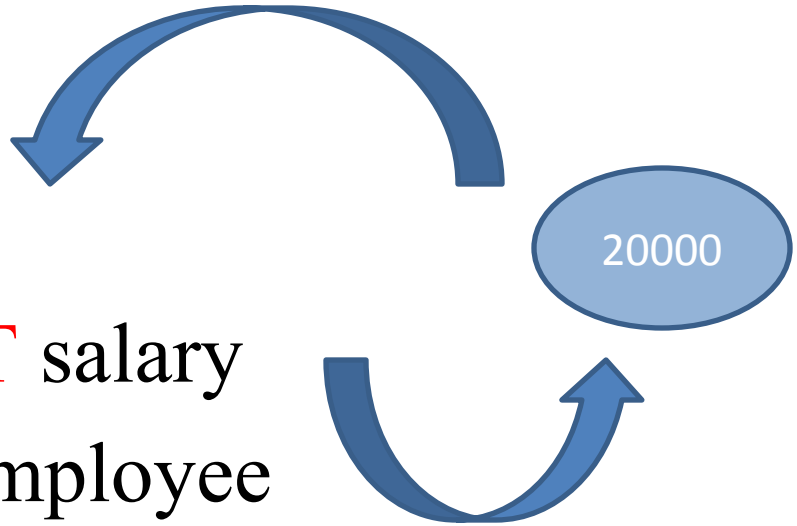
In the above sql statement, first the inner query is processed and then the outer query is processed.

Example-employee table

Employee_id	EMP_Name	Job_id	Salary	Department_id
100	ABC	ST_Clerk	20000	90
102	XYZ	Manager	50000	60
103	PQR	Accountant	30000	70
104	LMN	Assistant	40000	10
105	DEF	ST_Clerk	20000	90
106	JKL	Assistant	40000	10

- List out names of those employees whose salary is greater than employee ‘DEF’

```
SELECT EMP_Name
FROM employee
WHERE salary >
        (SELECT salary
        FROM employee
        WHERE EMP_Name=‘DEF’)
```



Types of subqueries

- Single-row subquery:- Queries that return only one row from the inner SELECT statement.
- Multiple-row subquery:-Queries that return more than one row from the inner SELECT statement.
- Multiple-column subquery:-Queries that return more than one column from the inner SELECT statement.

Single-row subquery

- It returns only one row
- Use single row comparison operators
- Query: Display the employees whose job id is same as that of employee 105

```
SELECT EMP_Name, job_id
FROM employee
WHERE job_id=
        (SELECT job_id
         FROM employee
         WHERE employee_id=105)
```

Employee_id	EMP_Name
100	ABC
105	DEF

Multiple-row subquery

- Return more than one row.
- Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

Using the IN Operator in multiple-row subqueries

- Subqueries that return more than one row are called multiple-row subqueries.
- You use a multiple-row operator, instead of a single-row operator, with a multiple-row subquery.
- The multiple-row operator expects one or more values.

```
SELECT last_name,salary,Depatment_id
FROM employee
WHERE salary IN (SELECT MIN(salary)
                 FROM employee
                 Group BY department_id);
```

- Example:
- Find the employees who earn the same salary as the minimum salary for each department.
- Here, the inner query is executed first, producing a query result. The main query block is then processed and uses the values returned by the inner query to complete its search condition. Hence the query would be:

SELECT EMP_name, salary, Depatment_id

FROM employee

WHERE salary IN (,,,,,,,,,,)

Example

Employee_id	EMP_Name	Job_id	Salary	Department_id
100	ABC	ST_Clerk	20000	90
102	XYZ	Manager	50000	60
103	PQR	Accountant	30000	70
104	LMN	Assistant	15000	10
105	DEF	ST_Clerk	25000	90
106	JKL	Assistant	10000	10
107	UVW	Adviser	22000	30

SQL WHERE, ANY, ALL Clause

- ANY and ALL keywords are used with a WHERE or HAVING clause.
- ANY and ALL operate on subqueries that return multiple values.
- ANY returns true if any of the subquery values meet the condition.
- ALL returns true if all of the subquery values meet the condition.

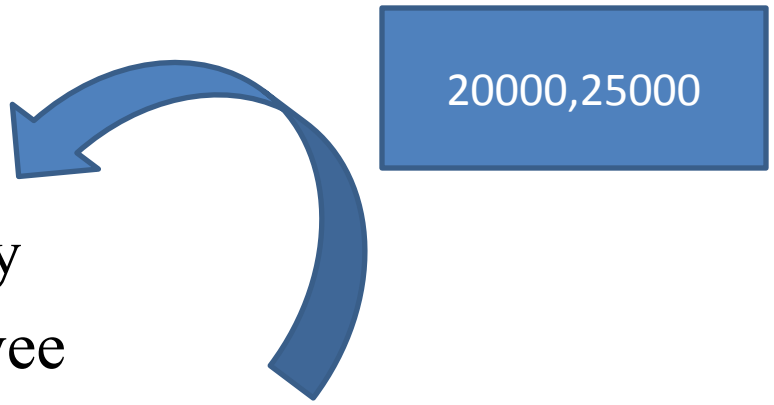
- "x = ANY (...)": The value must match one or more values in the list to evaluate to TRUE.
- "x != ANY (...)": The value must not match one or more values in the list to evaluate to TRUE.
- "x > ANY (...)": The value must be greater than the **smallest** value in the list to evaluate to TRUE.
- "x < ANY (...)": The value must be smaller than the **biggest** value in the list to evaluate to TRUE.
- "x >= ANY (...)": The value must be greater than or equal to the **smallest** value in the list to evaluate to TRUE.
- "x <= ANY (...)": The value must be smaller than or equal to the **biggest** value in the list to evaluate to TRUE.

Query

- Consider example which displays employees who are not clerk and whose salary is less than that of any other clerk.
- The maximum salary that a clerk earn is 25000.

Using the ANY Operator in multiple-row subqueries

```
SELECT Employee_id,Emp_name,Job_id,Salary,
FROM employee
WHERE  salary < ANY
```




```
AND job_id< > ‘ST_Clerk’;
```

Employee_id	EMP_Name	Job_id	Salary
104	LMN	Assistant	15000
106	JKL	Assistant	10000
107	UVW	Adviser	22000

- "x = ALL (...)": The value must match all the values in the list to evaluate to TRUE.
- "x != ALL (...)": The value must not match any values in the list to evaluate to TRUE.
- "x > ALL (...)": The value must be greater than the **biggest** value in the list to evaluate to TRUE.
- "x < ALL (...)": The value must be smaller than the **smallest** value in the list to evaluate to TRUE.
- "x >= ALL (...)": The value must be greater than or equal to the **biggest** value in the list to evaluate to TRUE.
- "x <= ALL (...)": The value must be smaller than or equal to the **smallest** value in the list to evaluate to TRUE.

Query

- Consider example which displays employees whose salary is less than the salary of all employees with a job_id of ST_clerk and whose job is not ST_Clerk.

Using the **ALL** Operator in multiple-row subqueries

```
SELECT Employee_id,Emp_name,Job_id,Salary,
FROM employee
WHERE  salary < ALL
                (SELECT salary
                FROM employee
                WHERE job_id='ST_Clerk')
AND  job_id<> 'ST_Clerk';
```

20000,25000

Employee_id	EMP_Name	Job_id	Salary
104	LMN	Assistant	15000
106	JKL	Assistant	10000

Cartesian Product

- If a sql join condition is omitted or if it is invalid the join operation will result in a Cartesian product.
- The Cartesian product returns a number of rows equal to the product of all rows in all the tables being joined.
- To avoid a cartesian product, always include a valid join condition in the WHERE clause
- For example, if the first table has 20 rows and the second table has 10 rows, the result will be 20 * 10, or 200 rows. This query takes a long time to execute.

Example

employee table

Emp_id	Last_nm	Dept_id
100	Abc	90
200	Def	90
300	Ghi	20
400	Jkl	50
500	mno	60

department table

Dept_id	Dept_nm	Loc_id
90	sales	1700
90	sales	1500
20	accounting	1200
50	contracting	1400
60	admin	1100
70	IT	1200

```
SELECT last_nm,dept_nm
FROM employee,department;
```

Cartesian Product :5*6=30
i.e 30 rows are selected.

Types of Joins

- Equijoin
- Non-equijoin
- Outer join
- Self join

- SQL Joins are used to relate information in different tables. A Join condition is a part of the sql query that retrieves rows from two or more tables.

- A SQL Join condition is used in the SQL WHERE clause of select, update, delete statements.

SQL-JOIN

- The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.

- Tables in a database are often related to each other with keys.

- A primary key is a column (or a combination of columns) with a unique value for each row. Each primary key value must be unique within the table. The purpose is to bind data together, across tables, without repeating all of the data in every table.

- The Syntax for joining two tables is:

```
SELECT table1.coumn,table2.column
FROM table1, table2
WHERE table1.column1=table2.column2
```

Where,

table1.column1=table2.column2; is the condition that joins or relates the tables together.

Different SQL JOINs

- **JOIN/inner join**: Return rows when there is at least one match in both tables.
- **LEFT JOIN**: Return all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN**: Return all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN**: Return rows when there is a match in one of the tables.

- **SQL INNER JOIN Keyword**

The INNER JOIN keyword return rows when there is at least one match in both tables.

- **SQL INNER JOIN Syntax**

```
SELECT column_name(s)
FROM table_name1
INNER JOIN table_name2
ON
   table_name1.column_name=table_name2.column_name
```

SQL INNER JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	ABC	A	ST1	pune
2	PQR	B	ST2	pune
3	XYZ	C	ST3	mumbai

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the persons with any orders.
We use the SELECT statement:

- SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
  FROM Persons
  INNER JOIN Orders
  ON Persons.P_Id=Orders.P_Id
  ORDER BY Persons.LastName
- The result-set will look like this:

LastName	FirstName	OrderNo
ABC	A	22456
ABC	A	24562
XYZ	C	77895
XYZ	C	44678

The INNER JOIN keyword return rows when there is at least one match in both tables. If there are rows in "Persons" that do not have matches in "Orders", those rows will NOT be listed.

- **SQL LEFT JOIN Keyword**

  The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).

- **SQL LEFT JOIN Syntax**

```
SELECT column_name(s)
FROM table_name1
LEFT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

In some databases LEFT JOIN is called LEFT OUTER JOIN.

SQL LEFT JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	ABC	A	ST1	pune
2	PQR	B	ST2	pune
3	XYZ	C	ST3	mumbai

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1

- Now if we want to list all the persons and their orders - if any, from the tables above.

- We use the following SELECT statement:

```
SELECT Persons.LastName,
Persons.FirstName, Orders.OrderNo
FROM Persons
LEFT JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

- The result-set will look like this:

LastName	FirstName	OrderNo
ABC	A	22456
ABC	A	24562
XYZ	C	77895
XYZ	C	44678
PQR	B	

The LEFT JOIN keyword returns all the rows from the left table (Persons), even if there are no matches in the right table (Orders).

- **SQL RIGHT JOIN Keyword**

  The RIGHT JOIN keyword returns all the rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

- **SQL RIGHT JOIN Syntax**

```
SELECT column_name(s)
FROM table_name1
RIGHT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

- In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

SQL RIGHT JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	ABC	A	ST1	pune
2	PQR	B	ST2	pune
3	XYZ	C	ST3	mumbai

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

- Now if we want to list all the orders with containing persons - if any, from the tables above.
- We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName,
Orders.OrderNo
FROM Persons
RIGHT JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	FirstName	OrderNo
ABC	A	22456
ABC	A	24562
XYZ	C	77895
XYZ	C	44678
		34764

The RIGHT JOIN keyword returns all the rows from the right table (Orders), even if there are no matches in the left table (Persons).

- **SQL FULL JOIN Keyword**

The FULL JOIN keyword return rows when there is a match in one of the tables.

- **SQL FULL JOIN Syntax**

```
SELECT column_name(s)
FROM table_name1
FULL JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

- **SQL FULL JOIN Example**

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	ABC	A	ST1	pune
2	PQR	B	ST2	pune
3	XYZ	C	ST3	mumbai

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

- Now if we want to list all the persons and their orders, and all the orders with their persons.

- We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName,
  Orders.OrderNo
  FROM Persons
  FULL JOIN Orders
  ON Persons.P_Id=Orders.P_Id
  ORDER BY Persons.LastName
```

LastName	FirstName	OrderNo
ABC	A	22456
ABC	A	24562
XYZ	C	77895
XYZ	C	44678
PQR	B	
		34764

The FULL JOIN keyword returns all the rows from the left table (Persons), and all the rows from the right table (Orders). If there are rows in "Persons" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Persons", those rows will be listed as well.

- SQL Joins can be classified into Equi join and Non Equi join.

1) SQL Equi joins

It is a simple sql join condition which uses the equal sign as the comparison operator. Two types of equi joins are SQL Outer join and SQL Inner join.

2) SQL Non equi joins

It is a sql join condition which makes use of some comparison operator other than the equal sign like >, <, >=, <=

1) SQL Equi Joins:

An equi-join is further classified into two categories:

a) SQL Inner Join b) SQL Outer Join

a) SQL Inner Join:

All the rows returned by the sql query satisfy the sql join condition specified.

- The columns must be referenced by the table name in the join condition, because P_id is a column in both the tables and needs a way to be identified. This avoids ambiguity in using the columns in the SQL SELECT statement.

- **database table "product";**

product_id	product_name	supplier_name	unit_price
100	Camera	Nikon	300
101	Television	Onida	100
102	Refrigerator	Vediocon	150
103	Ipod	Apple	75
104	Mobile	Nokia	50

database table "order_items"

order_id	product_id	total_units	customer
5100	104	30	Infosys
5101	102	5	Satyam
5102	103	25	Wipro
5103	101	10	TCS

SQL Outer Join

- This sql join condition returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables. The sql outer join operator in Oracle is (+) and is used on one side of the join condition only.

- The syntax differs for different RDBMS implementation. Few of them represent the join conditions as "sql left outer join", "sql right outer join".

- If you want to display all the product data along with order items data, with null values displayed for order items if a product has no order item, the sql query for outer join would be as shown below:

- SELECT p.product_id, p.product_name, o.order_id, o.total_units
  FROM order_items o, product p
  WHERE o.product_id (+) = p.product_id;

- The output would be like,

product_id	product_name	order_id	total_units
100	Camera		
101	Television	5103	10
102	Refrigerator	5101	5
103	Ipod	5102	25
104	Mobile	5100	30

If the (+) operator is used in the left side of the join condition it is equivalent to left outer join. If used on the right side of the join condition it is equivalent to right outer join.

- **1) SQL Self Join:**

A Self Join is a type of sql join which is used to join a table to itself, particularly when the table has a FOREIGN KEY that references its own PRIMARY KEY. It is necessary to ensure that the join statement defines an alias for both copies of the table to avoid column ambiguity.

To join a table to itself, we can use Self-join.

Employees-worker

Employee_Id	Last_Name	Manager_id
100	ABC	
101	PQR	100
102	XYZ	100
103	LMN	102
104	UVW	103

Employee_Id	Last_Name
100	ABC
101	PQR
102	XYZ
103	LMN
104	UVW

Employees-manager

Manager_id in the worker table is equal to Employee_id in the Manager table.

- ✔ To find the name of each employee's manager, you need to join the employees table to itself , or perform a self join.

The below query is an example of a self join,

```
SELECT worker.Last_name || 'works for' ||  manager.Last_name
FROM employees worker, employees manager
WHERE worker.Manager_id=manager.employee_id;
```

Output

worker.Last_name || 'works for' || manager.Last_name
PQR works for ABC
XYZ works for ABC
LMN works for XYZ
……
……

2) SQL Non Equi Join:

- A Non Equi Join is a SQL Join whose condition is established using all comparison operators except the equal (=) operator. Like >=, <=, <, >
- A non-equijoin is a join condition containing something other than an equality operator.
- **For example:**

  If you want to find the names of students who are not studying either History, the sql query would be like,

- SELECT first_name, last_name, subject
  FROM student_details
  WHERE subject != 'History'

Example: student_details;

id	first_name	last_name	age	subject	games
100	abc	gg	20	Science	Cricket
101	xyz	pp	22	Maths	Football
102	pqr	zz	29	Science	Cricket
103	lmn	aa	28	Maths	Badminton
104	def	bb	25	History	Chess

- The output would be something like,

first_name	last_name	subject
lmn	aa	Maths
xyz	pp	Maths
pqr	zz	Science
abc	gg	Science

SQL Functions

Functions are a very powerful feature of SQL and can be used to do the following:

- Perform calculations on data
- Manipulate output for groups of rows
- Format dates and numbers for display
- Convert column data types.

- Two types of functions in Oracle.

1) Single Row Functions: Single row or Scalar functions return a value for every row that is processed in a query.

2) Group Functions: These functions group the rows of data based on the values returned by the query. The group functions are used to calculate aggregate values like total or average, which return just one total or one average value after processing a group of rows.

Single row functions

- These are used to manipulate data items. They accept one or more arguments and return one value for each row returned by the query. An argument can be one of the following :
- User –supplied constant
- Variable value
- Column name
- Expression

These functions operate on single rows only and return one result per row.

- Four types of single row functions are:,

  1) Numeric Functions: These are functions that accept numeric input and return numeric values.

  2) Character or Text Functions: These are functions that accept character input and can return both character and number values.

  3) Date Functions: These are functions that take values that are of datatype DATE as input and return values of datatype DATE, except for the MONTHS_BETWEEN function, which returns a number.

4) Conversion Functions: These are functions that help us to convert a value in one form to another form. For Example: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE etc.

- You can combine more than one function together in an expression. This is known as nesting of functions.

- **1) Numeric Functions:**

  Numeric functions are used to perform operations on numbers. They accept numeric values as input and return numeric values as output. Few of the Numeric functions are:

Function Name	Return Value
ABS (x)	Absolute value of the number '*x*'
CEIL (x)	Integer value that is Greater than or equal to the number '*x*'
FLOOR (x)	Integer value that is Less than or equal to the number '*x*'
TRUNC (x, y)	Truncates value of number '*x*' up to '*y*' decimal places
ROUND (x, y)	Rounded off value of the number '*x*' up to the number '*y*' decimal places

Function Name	Examples	Return Value
ABS (x)	ABS (1) ABS (-1)	1 -1
CEIL (x)	CEIL (2.83) CEIL (2.49)	3 3
FLOOR (x)	FLOOR (2.83) FLOOR (2.49)	2 2
TRUNC (x, y)	trunc(125.815, 1) trunc(125.815, 2) trunc(125.815, 3) trunc(-125.815, 2)	125.8 125.81 125.815 -125.81
ROUND (x, y)	round(125.315, 0) round(125.315, 1) round(125.315, 2) round(125.315, 3) round(-125.315, 2)	125 125.3 125.32 125.315 -125.32

2) Character or Text Functions:

Character or text functions are used to manipulate text strings. They accept strings or characters as input and can return both character and number values as output.

Function Name	Return Value
LOWER (string_value)	All the letters in *'string_value'* is converted to lowercase.
UPPER (string_value)	All the letters in *'string_value'* is converted to uppercase.

Function Name	Return Value
INITCAP (string_value)	All the letters in *'string_value'* is converted to mixed case.
LTRIM (string_value, trim_text)	All occurrences of *'trim_text'* is removed from the left of *'string_value'*.
RTRIM (string_value, trim_text)	All occurrences of *'trim_text'* is removed from the right of *'string_value'* .

Continue…….

TRIM (trim_text FROM string_value)	**All occurrences of *'trim_text'* from the left and right of *'string_value'* , *'trim_text'* can also be only one character long .**
SUBSTR (string_value, m, n)	Returns *'n'* number of characters from *'string_value'* starting from the '*m'* position.
LENGTH (string_value)	Number of characters in *'string_value'* is returned.

LPAD (string_value, n, pad_value)	**Returns *'string_value'* left-padded with *'pad_value'* . The length of the whole string will be of *'n'* characters.**
RPAD (string_value, n, pad_value)	Returns *'string_value'* right-padded with *'pad_value'* . The length of the whole string will be of *'n'* characters.

Function Name	Examples
LOWER(string_value)	LOWER('Good Morning')
UPPER(string_value)	UPPER('Good Morning')
INITCAP(string_value)	INITCAP('GOOD MORNING')
LTRIM(string_value, trim_text)	LTRIM ('Good Morning', 'Good)
RTRIM (string_value, trim_text)	RTRIM ('Good Morning', ' Morning')
TRIM (trim_text FROM string_value)	TRIM ('o' FROM 'Good Morning')
SUBSTR (string_value, m, n)	SUBSTR ('Good Morning', 6, 7)
LENGTH (string_value)	LENGTH ('Good Morning')
LPAD (string_value, n, pad_value)	LPAD ('Good', 6, '*')
RPAD (string_value, n, pad_value)	RPAD ('Good', 6, '*')

3) Date Functions:

- These are functions that take values that are of datatype DATE as input and return values of datatype as DATE, except for the MONTHS_BETWEEN function, which returns a number as output.
- Few date functions are as given below.

Function Name	Return Value
ADD_MONTHS (date, n)	Returns a date value after adding *'n'* months to the date *'x'*.
MONTHS_BETWEEN (x1, x2)	Returns the number of months between dates x1 and x2.
ROUND (x, date_format)	Returns the date *'x'* rounded off to the nearest century, year, month, date, hour, minute, or second as specified by the *'date_format'*.

Continue…

TRUNC (x, date_format)	**Returns the date *'x'* lesser than or equal to the nearest century, year, month, date, hour, minute, or second as specified by the 'date_format'.**
NEXT_DAY (x, week_day)	Returns the next date of the *'week_day'* on or after the date *'x'* occurs.
LAST_DAY (x)	It is used to determine the number of days remaining in a month from the date *'x'* specified.
SYSDATE	Returns the systems current date and time.
NEW_TIME (x, zone1, zone2)	Returns the date and time in zone2 if date 'x' represents the time in zone1.

Examples of date functions

Function Name	Examples	Return Value
ADD_MONTHS ( )	ADD_MONTHS ('16-Sep-81', 3)	16-Dec-81
MONTHS_BETWEEN( )	MONTHS_BETWEEN ('16-Sep-81', '16-Dec-81')	3
NEXT_DAY( )	NEXT_DAY ('01-Jun-08', 'Wednesday')	04-JUN-08
LAST_DAY( )	LAST_DAY ('01-Jun-08')	30-Jun-08
NEW_TIME( )	NEW_TIME ('01-Jun-08', 'IST', 'EST')	31-May-08

4) Conversion Functions

- These are functions that help us to convert a value in one form to another form.

- For Ex: a null value into an actual value, or a value from one data type to another data type like NVL, TO_CHAR, TO_NUMBER, TO_DATE.

- Few of the conversion functions available in oracle are:

Function Name	Return Value
TO_CHAR (x [,y])	Converts Numeric and Date values to a character string value. It cannot be used for calculations since it is a string value.
TO_DATE (x [, date_format])	Converts a valid Numeric and Character values to a Date value. Date is formatted to the format specified by *'date_format'*.
NVL (x, y)	If *'x'* is NULL, replace it with *'y'*. *'x'* and *'y'* must be of the same datatype.

The below table provides the examples for the above functions

Function Name	Examples	Return Value
TO_CHAR ()	TO_CHAR (3000, '$9999') TO_CHAR (SYSDATE, 'Day, Month YYYY')	$3000 Monday, Aug 2012
TO_DATE ()	TO_DATE ('01-Aug-12')	01-Aug-2012
NVL ()	NVL (null, 1)	1

References

- https://www.studytonight.com/dbms/sql-constraints.php
- https://docs.oracle.com/cd/B19306_01/server.102/b14200/functions001.htm
- https://www.tutorialspoint.com/sql

END