Unit – II

SQL (Structured Query Language)

- Introduction to SQL
- SQL sublanguages – DDL, DML, DCL
- Basic data types
- SQL statements: Create, Select, Insert, Delete, Update etc.
- Database constraints
- Built-in functions

SQL (Structured Query Langugar)

- **What is SQL?**
  - SQL is a language that provides an interface to RDBMS.
  - SQL is an ANSI (American National Standards Institute) standard language.
  - RDBMS like SQL Server , Oracle, MS Access use SQL as their standard database language.

- **Features of SQL**
- SQL used to access and manipulate databases. It can
  - Creates a new objects in database
  - Insert records in a database
  - Update records in a database
  - Delete records from a database.
  - Retrieve records from a database.

- **SQL Engine**
  - It is a server side process takes SQL queries from the application and execute.

Rules For SQL

- SQL starts with a verb (i.e. A SQL action word)
  - E.g. CREATE , INSERT , UPDATE , DELETE

- A comma (**,**) separates parameters. E.g. INSERT INTO EMP VALUES (1 **,** 'HARSH');

- A ' **;** ' is used to end SQL statements. E.g. INSERT INTO EMP VALUES (1 , 'HARSH') **;**

- A space separate clauses. E.g. DROP TABLE EMP;

- Reserve words cannot be used as identifier unless enclosed with double quotes.

- Character and Data literals must be enclosed within single quotes.

DDL (Data Definition Language) commands

- It is set of SQL commands used to create, modify and delete database structures but not data.
- DDL commands **implicitly issues a commit** command to the database.
- **CREATE** – To create objects in the database.
- **ALTER** – Alters / Modifies the existing structure of the database.
- **DROP** – Delete an entire object from the database.
- **TRUNCATE** – Remove all records from a table, but the structure of table is present.

DML (Data Manipulation Language) commands

- Commands of SQL that allows inserting, changing or retrieving data within the database.

- **INSERT** – Insert data into a table.

- **UPDATE** – Updates existing data within a table.

- **DELETE** – Deletes records from a table.

- **SELECT** – Retrieve certain / all records from one or more tables.

DCL (Data Control Language) command

- Commands of SQL that control access to data and to the database.
- **COMMIT** – Save work done.
- **ROLLBACK** – Restore database to original since the last COMMIT.
- **GRANT –** Grant permissions to the oracle users.
- **REVOKE** – Revert permissions from the oracle users.

SQL (Structural Query Language)

Components of SQL:

DDL (Data Definition Language)

Create, Alter , Drop

DML (Data Manipulation Language)

Insert, Update , Delete, ...

DCL (Data Control Language)

Commit, Rollback

Data Types

Data Type	Description
CHAR(size)	Is used to store character strings values of fixed length. The size in brackets the number of characters the cell can hold. The maximum number of characters hold is 2000. For example: Name CHAR(50)
VARCHAR(size) / VARCHAR2(size)	Is used to store variable length alphanumeric data. It is a more flexible form of CHAR data type. Varchar can hold maximum 2000 characters. Varchar2 can hold maximum 4000 characters.
DATE	It is used to represent date and time. The standard format is DD-MON-YY as in 21-JUN-04. To enter dates other than standard format use the inbuilt functions. In Oracle 11g standard format is MM-DD-YY or MM-DD-YYYY

Data Types

Data Type	Description
NUMBER(P, S)	It is used to store numbers (fixed or floating point). P : Maximum length of the data. It includes both decimal & fractional part digits. S : Number of places to the right of the decimal point. Precision is maximum 38 digits. If scale is not defined: Maximum no. of digits used in decimal part is P. If scale is defined: Maximum (P – S) no. of digits are used for Decimal part and maximum S number of digits are used for fractional part. Valid values : 0, positive nos. & negative nos. Numbers may be expressed in two ways 1. With the numbers 0 .. 9 , + , - and decimal point (.). 2. Scientific notation like 1.85E3 for 1850
LONG	It is used to store variable length character strings containing up to 2 GB. **Only one LONG** value can be defined per table.
LONG RAW	These data types are used to store binary data, such as digitized picture or image. LONG RAW data type can contain up to 2 GB. Values stored in columns having LONG RAW data type cannot be indexed.

Data Types

Data Type	Description
Rowid	Every record in the database has a physical address or rowid. Fixed length binary data. The format of rowid is : BBBBBBB.RRRR.FFFFF Where **B :** Block in the database file **R :** row in the block **F :** Database file
NCHAR(size)	Is used to store character strings values of fixed length. The characters may be of different language. The maximum number of characters hold is 2000.
NVARCHAR(size) / NVARCHAR2(size)	Is used to store variable length alphanumeric data. It is a more flexible form of NCHAR data type. NVarchar can hold maximum 2000 characters. NVarchar2 can hold maximum 4000 characters.
Blob	Maximum size is 4 GB. Large Binary Objects within the database.
Clob	Maximum size is 4 GB. Large Character Large Objects within the database.
Nclob	Maximum size is 4 GB. Large Character Large Objects within the database.

Primary Key

- A **Primary key** is a key (Attribute/Field) that uniquely identifies a row in each table.
- It is denoted with its first two letters, namely, **PK**.
- Employee table is uniquely identified by Employee_No field than it is called a primary key of a table.
- Project Table Project number is PK.

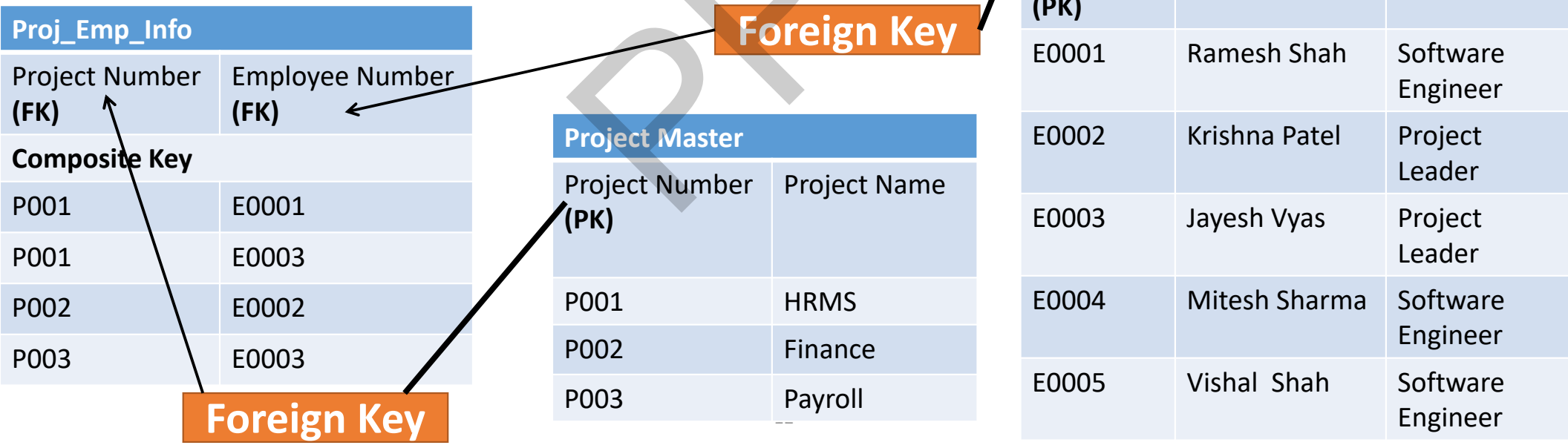
Project Master	
Project Number **(PK)**	Project Name
P001	HRMS
P002	Online Bidding
P003	Payroll

Employee Master		
Employee Number **(PK)**	Employee Name	Designation
E0001	Ramesh Shah	Software Engineer
E0002	Krishna Patel	Project Leader
E0003	Jayesh Vyas	Project Leader
E0004	Mitesh Sharma	Software Engineer
E0005	Vishal Shah	Software Engineer

Foreign Key

- A **Foreign key** is a key borrowed from another related table (that's why its foreign) in order to make the relationship between two tables.
- It is normally denoted with its first two letters, namely, **FK**.
- **Foreign key** must match actual values and data types with the related **Primary Key.**

E.g. Proj_Emp_Info table Project_Number & Employee Number are foreign keys.



Proj_Emp_Info

Project Number **(FK)**	Employee Number **(FK)**
Composite Key	
P001	E0001
P001	E0003
P002	E0002
P003	E0003

Project Master

Project Number **(PK)**	Project Name
P001	HRMS
P002	Finance
P003	Payroll

Employee Master

Employee Number **(PK)**	Employee Name	Designation
E0001	Ramesh Shah	Software Engineer
E0002	Krishna Patel	Project Leader
E0003	Jayesh Vyas	Project Leader
E0004	Mitesh Sharma	Software Engineer
E0005	Vishal Shah	Software Engineer

Alternate Key

- A **key** associated with one or more columns whose values uniquely identify every row in the table, but which is not the primary **key**.
- **E.g.** Employee Master having Employee Name as Alternate key but it is not primary key.

Employee Master		
Employee Number **(PK)**	Employee Name	Designation
E0001	Ramesh Shah	Software Engineer
E0002	Krishna Patel	Project Leader
E0003	Jayesh Vyas	Project Leader
E0004	Mitesh Sharma	Software Engineer
E0005	Vishal Shah	Software Engineer

Create Statement

- It is used to create a new **Table (Entity or Relation)** in database.
- Each table having multiple **Columns (Attributes or Fields)**.
- **Each Column of contains three things :**
  - Column-name,
  - Data-type &
  - Size
- **Rules for Creating Tables**
  - A name can have maximum upto 30 characters.
  - Alphabet from A-Z, a-z and number from 0-9 are allowed.
  - A name should begin with Alphabet.
  - The use of special characters like _ , $ & # signs are allowed.
  - SQL reserved words are not allowed.
- **Syntax**
- **CREATE TABLE** *table_name*
  (
  column_name1 data_type(*size*)
  ……. …….
  column_nameN data_type(*size*) *);*

Create Statement

```
CREATE TABLE Student_Master
   ( Stud_Id        NUMBER (2) PRIMARY KEY ,
     Stud_Name VARCHAR(20),
     Stud_Birth_Date DATE,
     Stud_Mobile   NUMBER(10)
   )
```

Constraints

- **Primary Key**
  - **At column level**
    - Column-name data-type(size) **PRIMARY KEY**
  - E.g.

```
CREATE TABLE Student_Master
( Stud_Id  NUMBER (2) PRIMARY KEY ,
  Stud_Name VARCHAR(20),
  Stud_Birth_Date DATE,
  Stud_Mobile   NUMBER(10) )
/
```

OR

  - **At table level**
    - **PRIMARY KEY** (col-name1, col-name2, ……. col-nameN)

```
CREATE TABLE Student_Master
( Stud_Id  NUMBER (2),
  Stud_Name VARCHAR(20),
  Stud_Birth_Date DATE,
  Stud_Mobile   NUMBER(10),
  PRIMARY KEY (Stud_Id))
/
```

To Display Table Structure

- DESCRIBE Table-name

OR

- DESC Table-name

E.g.

DESCRIBE Student_Master;

DESC Student_Master;

SQL Query Is Not a Case-Sensitive

- Query may be allowed in either
  - Upper case or
  - Lowercase or
  - Combination of uppercase and lowercase

E.g.

```
DESC Student_Master;
desc Student_Master;
DesC Student_Master;
DESC STUDENT_MasTer;
```

Insert Statement

- **Creates a new row in database tables and loads the values passed into the columns specified.**
- **Syntax**
- To insert values of all fields/attributes
  - **INSERT INTO** *table_name* **VALUES** (*value1*,*value2*, ...*valueN*)
  - **For Oracle 10g** INSERT INTO STUDENT_MASTER

    INSERT INTO STUDENT_MASTER **VALUES** (1 , 'Amit' , '**05-DEC-90**', 9422290900)
  - **For Oracle 11g Express Edition**

INSERT INTO STUDENT_MASTER **VALUES** (1 , 'Amit' , '**12-05-1990**', 9422290900)

OR

- To insert values of selected fields/attributes
  - **INSERT INTO** *table_name* (*column1*,*column2*, ..*columnN*) **VALUES** (*value1*,*value2*, ..*valueN*);
  - **INSERT INTO** STUDENT_MASTER (Stud_Id, Stud_Name) **VALUES** (2,'Kalpesh');

To Insert Records In A Table

- To insert multiple records of all fields/attributes
  - **INSERT INTO** *table_name*
    VALUES (&*column1*,&column2, ... &column*N*);
- To insert multiple records of selected fields/attributes
  - **INSERT INTO** *table_name* (*column1*, ... *columnX*)
    VALUES (&*column1,* ... &columnX);

For Oracle 10g

==========

- **INSERT INTO** STUDENT_MASTER **VALUES** (&STUD_ID,'&STUD_NAME', '&STUD_BIRTH_DATE', &STUD_MOBILE);

OR

- **INSERT INTO** STUDENT_MASTER (STUD_ID) **VALUES** (&STUD_ID);

To Insert Records In A Table

- To insert multiple records of all fields/attributes
  - **INSERT INTO** *table_name* (*column1*,*column2*, ... *columnN*)
    VALUES (&*column1*,&column2, ... &column*N*);

E.g. For Oracle 11g Express Edition in Browser.

```
INSERT INTO STUDENT_MASTER
(STUD_ID,STUD_NAME,STUD_BIRTH_DATE,STUD_MOBILE)
 WITH names AS (
   SELECT 4, 'Ratna',     '10-20-98', '111111111'    FROM dual UNION ALL
   SELECT 6, 'Prakash',     '5-25-97', '3333333311'    FROM dual UNION ALL
   SELECT 5, 'Kartik', '09-23-86', '222222222'   FROM dual
 )
SELECT * FROM Names
```

Select Statement

- **It is used to retrieve rows select from one or more tables.**

- **Syntax**

SELECT [**DISTINCT**]

{ *|*column_name1*, *column_name2, .. column_nameN* **}**

FROM *table_name*

[**Where** conditions]

[**ORDER BY {** column_name1.., column_nameN [**ASC / DESC**]
}] **;**

Select Statement

- **All rows and all columns**
  - SELECT * FROM table-name;

  E.g.
  - SELECT * FROM Student_Master;

- **Filtering Data**
  - **Select columns and all rows**
    - SELECT { **Selected COLUMN-LIST** } FROM table-name;

    E.g.
    - SELECT Stud_Id, Stud_Name FROM Student_Master;
    - SELECT Stud_Name FROM Student_Master;
    - SELECT Stud_Id FROM Student_Master;

Select Statement

- **Selected rows and all columns**
  - SELECT * FROM table-name [WHERE conditions]

  E.g.
  - SELECT * FROM Student_Master **WHERE** Stud_Id > 2;
  - SELECT * FROM Student_Master **WHERE** Stud_Name = 'Amit';

- **Selected columns and selected rows**
  - SELECT {Selected COLUMN-LIST } FROM table-name [WHERE conditions]

  E.g.
  - SELECT Stud_Name FROM Student_Master WHERE Stud_Id > 2;
  - SELECT Stud_iD FROM Student_Master WHERE Stud_Id > 2;

Select Statement

- **Eliminating Duplicate Rows**
  - SELECT DISTINCT * FROM table-name;
  - SELECT DISTINCT { Selected Column-list } FROM table-name;

  E.g.
  - SELECT DISTINCT * FROM Student_Master;
  - SELECT DISTINCT Stud_Name FROM Student_Master;

Select Statement :: Ordering

- **Sorting Data**
  - SELECT * FROM table-name **ORDER BY**

    { column_name1.., column_nameN [**ASC / DESC**] }

  E.g.
  - SELECT * FROM Student_Master ORDER BY Stud_Name
  - SELECT * FROM Student_Master ORDER BY Stud_Id
  - SELECT * FROM Student_Master ORDER BY Stud_Name DESC
  - SELECT * FROM Student_Master ORDER BY Stud_Name ASC
  - SELECT * FROM Student_Master ORDER BY Stud_Birth_Date DESC

Select Statement :: Grouping

- **Sorting Data**
  - SELECT * FROM table-name **GROUP BY**

    { column_name1.., column_nameN [**ASC / DESC**] }

  E.g.

  - CREATE TABLE Emp_Master (Emp_id Number(3), Emp_name Varchar(40), Desg Varchar(40));

  - SELECT Desg, Count(Emp_Id) FROM Emp_Master GROUP BY Desg

Exercise

- **Product_Master (Product_Id, Product_Name, Price, Quantity)**
- Show the structure of Product_Master table.
- Insert at least five records in Product_Master table.
- Retrieve all records from Product_Master table.
- Show all products name and price.
- Display product names whose price is equal to 200.
- Display product name & price whose price greater than 100 and less than 500.
- Display all product names and It's price sorted by price.
- Display all product names and price whose price greater than 500.

Update Statement

- **Syntax**

  UPDATE *table_name*
  SET
  { *column1=value1,column2=value2,...*
  columnN=valueN }
  [**WHERE** *Conditions*]

- ***Update single column of single row***

  UPDATE *table_name* **SET** *column1=value1* ***WHERE*** *condition*

  E.g. Change the name of student id = 2.

  – ***UPDATE*** *Student_Master* ***SET*** *Stud_Name='MAHESH'*
  WHERE *Stud_Id = 2*

Update Statement

- **Syntax**

  UPDATE *table_name*
  SET
  { *column1=value1,column2=value2,...*
  columnN=valueN }
  [**WHERE** *Conditions*]

- ***Update more than one column of single row.***

  UPDATE *table_name* **SET** *column1=value1, ……..*
  columnN=valueN ***WHERE*** *condition*

  E.g. Change the mobile & name of student with id = 2.

  UPDATE Student_Master
  SET Stud_Mobile = 8811223355 , Stud_name ='Maulik'
  WHERE Stud_Id = 2

Update Statement

- **Syntax**

  UPDATE *table_name*
  SET
  { *column1=value1,column2=value2,...*
  columnN=valueN}
  [**WHERE** *Conditions]*

- ***Update single column of selected rows.***

  UPDATE *table_name* **SET** *column1=value1 WHERE condition*

  E.g.

ALTER TABLE Student_Master **ADD** Graduation VARCHAR2(10);

  - **UPDATE** Student_Master
    SET Graduation = 'Science'
    WHERE Stud_id > 3

Update Statement

- **Syntax**

  UPDATE *table_name*
  SET
  { *column1=value1,column2=value2,...*
  columnN=valueN }
  [**WHERE** *Conditions*]

- ***Update single column of all rows.***

  UPDATE *table_name* **SET** *column1=value1*

  E.g.

  UPDATE Student_Master
  SET Graduation = 'Science'

Delete Statement

- **Syntax**

  DELETE FROM *table_name* [**WHERE** Condition];

- **Remove selected rows from table**

  DELETE FROM *table_name* **WHERE** Condition;

  E.g.

  DELETE FROM Student_Master **WHERE** Stud_Id = 5

  DELETE FROM Student_Master **WHERE** Stud_Id > 5

  DELETE FROM Student_Master **WHERE** Stud_Name = 'AMIT'

- **Remove all rows from table**

  DELETE FROM *table_name.*

  E.g. **DELETE FROM** Student_Master

Alter Table Syntax

- **ALTER TABLE** table_name **ADD** column_name datatype;

E.g. ALTER TABLE Student_Master **ADD** Email VARCHAR2(100);

- **ALTER TABLE** table_name **DROP COLUMN** column_name;

E.g. ALTER TABLE Student_Master **DROP COLUMN** Email ;

- **ALTER TABLE** table_name **MODIFY COLUMN** column_name datatype;

E.g. ALTER TABLE Student_Master **MODIFY** Email VARCHAR2(150);

- **Restriction on the Alter table**

  The following tasks cannot be performed.
  - Change the name of the table
  - Change the name of the column
  - Decrease the size of a column if any value with more than specified size.

- **Integrity constraints through Alter table**
  - **ALTER TABLE** table_name **ADD PRIMARY KEY** (Col_name1, …..Col_name N)
  - **ALTER TABLE** table_name **DROP PRIMARY KEY**

- **Objects created by user**
  - **SELECT * FROM TAB;**

- **RENAMING Table :** Change the name of a table
  - **RENAME** table-name **TO** new-table-name

- **TRUNCATING Table :** Remove only records, not a Structure of table.
  - **TRUNCATE TABLE** table-name

- **DESTROYING Table :** Remove the structure of the table also.
  - **DROP TABLE** table-name

- **SYNONYMS Table :** Alternative name of a table
  - **CREATE SYNONYM** synonym-name **FOR** table-name

- **DROPPING SYNONYM Table :** Remove alternative name of a table
  - **DROP SYNONYM** synonym-name

GROUP BY Clause

- **Syntax**

  SELECT <columnName1> <columnName2> ……<columnNameN>

  AGGREGATE FUNCTION (<Expression>)

  FROM Table Name

  WHERE <Condition >

  GROUP BY <columnName1> <columnName2> ……<columnNameN>

  - The **Columns1 …. Columns N** are not used within an aggregate function and **must be included in the GROUP BY clause**
  - **Aggregate functions** are **AVG(), COUNT(), MAX(), MIN()** etc.

Select Statement :: Grouping

- **Sorting Data**
  - SELECT * FROM table-name **GROUP BY**
    { column_name1.., column_nameN [**ASC / DESC**] }

    E.g.

    Add City attribute in Student_Master table.

    ALTER TABLE Student_Master **ADD** City VARCHAR2(50);

  - SELECT city, count(stud_id) FROM Student_Master GROUP BY City

HAVING CLAUSE

- It impose a condition on the GROUP BY clause, which further filters the groups created by the GROUP BY clause.
- It filters on fields that used with aggregate function.

  - SELECT city, count(stud_id) FROM Student_Master **GROUP BY** City

  - SELECT city, count(stud_id) FROM Student_Master **GROUP BY** City **HAVING** count(stud_id) > 2;

WHERE clause & HAVING clause

- WHERE clause impose a condition on all rows in table.
- GROUP BY clause is written after WHERE clause. It makes a group on rows selected by WHERE clause.
- HAVING clause impose a condition on group. It filters on fields that used with aggregate function.
  - SELECT city, count(stud_id) FROM Student_Master **GROUP BY** City **HAVING** count(stud_id) > 2;
  - SELECT city, count(stud_id) FROM Student_Master **WHERE** city **like 'A%'** **GROUP BY** City **HAVING** count(stud_id) > 2;

Creating a table from a table

- **CREATE TABLE** *table_name*
  (
  column_name1
  ……. …….
  column_nameN)

AS

SELECT

(column_name1
……. …….
column_nameN)

FROM *table-name;*

Inserting data into a table from another table

- **INSERT**

  INTO *table_name*
  SELECT
  (column_name1
  …… ……
  column_nameN)
  FROM table-name
  [**WHERE** conditions]

;

Exercise

Client_Master (Client_No, Client_Name, Client_*City*, Gender, BirthDate, Pincode)


```
CREATE TABLE Client_Master
(Client_No NUMBER(3) PRIMARY KEY,
Client_Name VARCHAR2(40),
Client_City VARCHAR2(40),
Gender CHAR(1),
BirthDate DATE ,
Pincode NUMBER(6))
```

Exercise

- Retrieve all records of Client_Master table
- Display names of all the clients.
- List all the clients who are located in Mumbai.
- Find the names of a client who have birthdate = '20-Oct-1980'
- Find the names of a client who have Gender ='Male' and City = 'Anand'
- Display all records in alphabetical order of Client_Name;
- Add a column called 'Mobile_No' of data type 'number' and size = 10 to the Client_Master;
- Change the size of City_Name column in Client_Master to 50.
- Change the city of client_no = 10 to 'Bombay';
- Change the mobile number of client_name='Harsh' to 8989989898.
- Delete all the clients whose city belongs to 'Banglore'.
- Backup the Client_Master table in B_Client_Master.
- Change the name of Client_Master table to Client_Mst.
- Destroy the B_Client_Master table with its data.

Computations on Data

- Arithmetic Operators
  - +, -, *, /, ( )
- Logical Operators
  - OR, AND, NOT
- Range Searching
  - BETWEEN
- Pattern Matching
  - LIKE
- Predicates
  - IN, NOT IN
- DUAL Table

Emp Table

- CREATE TABLE Emp_Master
  (Eno NUMBER(3),
  Ename VARCHAR(50),
  Dept VARCHAR(20),
  Post VARCHAR(50),
  Basic NUMBER(6), PRIMARY KEY (Eno))

Operators:

SQL supports various arithmetic, logical , character, and relational r operators. **Arithmetic Operators:**

Arithmetic operators in SQL will return numeric values.

Operators	Function	Example
+	Add	UPDATE Emp_Master SET basic = basic + 500;
-	Subtract	UPDATE Emp_Master SET basic = basic - 500;
*	Multiply	UPDATE Emp_Master SET basic = basic + basic * 5/100;
/	Divide	UPDATE Emp_Master SET basic = basic + basic * 5/100;

Logical Operators:

Logical operators in SQL will return either true or false value.

Operators	Function	Example
AND	Check two conditions are true	SELECT * FROM Emp_Master WHERE basic >= 10000 AND basic <= 20000;
OR	Check any of the two conditions are true	SELECT * FROM Emp_Master WHERE basic >= 10000 OR dept = 'Sales';
NOT	Reversed the result of logical expression	SELECT * FROM Emp_Master WHERE NOT(basic >= 10000 OR dept = 'Sales');

Character Operators:

Character operators are used to manipulate character strings.

Operators	Function	Example
||	Concatenates character strings	SELECT 'Name is ' || ename FROM Emp_Master; Output: Name is Kumar Name is Umar Name is Vino Name is Raj

Comparison Operators:

Comparison operators in SQL will compare two quantity

Operators	Meaning/Function
=	Equal to
!=, <>	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Operators	Meaning/Function
IN	Equal to any member of a given set
NOT IN	Not equal to any member of a given set
BETWEEN	In between two values
NOT BETWEEN	Not in between two values
EXISTS	True if sub query returns at least one row
ANY	Compares a value to each list of value (must be prefaced with =,!=,<,<=,>,>=)
ALL	Compares a value to every list of value (must be prefaced with =,!=,<,<=,>,>=)

Example:

1) To find the employees who are either clerk or manager.

[I] SELECT * FROM Emp_Master

WHERE post **IN** ('CLERK','MANAGER');

[ii] SELECT * FROM Emp_Master

WHERE post = 'CLERK' **OR** post = 'MANAGER'

2) To find the employees who are neither clerks nor anager

SELECT * FROM Emp_Master

WHERE post **NOT IN** ('CLERK','MANAGER');

Example:

3) To find the employees whose salary lies in between 10000 and 20000

[1] SELECT * FROM Emp_Master

WHERE basic **BETWEEN** 10000 AND 20000;

[2] SELECT * FROM Emp_Master

WHERE basic >= 10000 AND basic <= 20000;

4) To find the employees whose salary not lies in between 10000 and 20000

SELECT * FROM Emp_Master

WHERE basic **NOT BETWEEN** 10000 AND 20000;

LIKE Operator: (Matching a character pattern)

When one does not know the exact value for the search conditions and wishes to use a replaceable parameter a LIKE operator can be used.

The replaceable parameters used are shown bellow.

Symbol	Represents
• **%**	Any sequence or more characters
• **_**	Any single character

Example:

1) To list employees whose names begin with 'P'

```
SELECT * FROM Emp_Master
        WHERE ename LIKE 'P%' ;
```

Example:

2) To list employees whose name start with 'P', third character must be 'R', fourth character must be 'A' and length must be 5.

E.g. Values : ('PARAS',PARAM','PAVAN', 'PARAMVEER')

Selected Values : ('**PARA**S',**PARA**M')

```
SELECT * FROM Emp_Master
   WHERE ename LIKE 'P_RA_';
```

3) To list employees whose name start with 'P' , third character must be 'R', fourth character must be 'A' and length may be any number.

E.g. **Values :** ('PARAS',PARAM','PAVAN','PARAMVEER')

Selected Values : ('**PARA**S',**PARA**M','**PARA**MVEER')

```
SELECT * FROM Emp_Master
   WHERE ename LIKE 'P_RA%';
```

Tables For Examples of
Exists, Not Exists, All & Any Operators

Client_Master (Client_No, Client_Name,
Client_*City*, Gender, BirthDate, Pincode)

Bill_Details (bill_no, client_no, bill_amt)

EXIST Operator

- The Oracle EXISTS operator is a Boolean operator that returns either true or false.
- The EXISTS operator is often used with a subquery to test for the existence of rows:
- SELECT * FROM table_name WHERE EXISTS (subquery);
- **E.g. To find all customers who have the order.**

```
SELECT client_no, client_name FROM client_master c
    WHERE EXISTS (
                SELECT 1 FROM bill_details
                WHERE client_no = c.client_no )
                ORDER BY client_name;
```

NOT EXIST Operator

- The Oracle **NOT EXISTS** operator is a Boolean operator that returns either true or false.
- The NOT EXISTS operator is often used with a subquery to test for not existence of rows:
- SELECT * FROM table_name WHERE EXISTS (subquery);
- **E.g. To find all customers who have not given an order.**

```
SELECT client_no, client_name FROM client_master c
    WHERE NOT EXISTS (
            SELECT 1 FROM bill_details
            WHERE client_no = c.client_no )
            ORDER BY client_name;
```

ANY Operator

- The **ANY** operator returns true if any of the subquery values meet the condition.
- SELECT *column_name(s)* FROM *table_name*
  WHERE *column_name operator* **ANY**
  (SELECT *column_name* FROM *table_name* WHERE *condition*);
- **E.g. Lists the customer names if it finds ANY records in the OrderDetails table that bill amount <= 3000**
- SELECT client_no, client_name FROM client_master

  WHERE CLIENT_NO = **ANY** (SELECT CLIENT_NO FROM bill_details
  WHERE bill_amt <= 3000)

  ORDER BY client_no, client_name;

ALL Operator

- The **ALL** operator returns true if all of the subquery values meet the condition.
- SELECT *column_name(s)* FROM *table_name*
  WHERE *column_name operator* **ANY**
  (SELECT *column_name* FROM *table_name* WHERE *condition*);
- **E.g. Lists the customer names whose bill amount not less than 3000.**
- SELECT client_no, client_name FROM client_master

  WHERE CLIENT_NO **!= ALL** (SELECT CLIENT_NO FROM bill_details
  WHERE bill_amt<=3000)

  ORDER BY client_no, client_name;

Constraints

- **Foreign Key**
- It defines the relationship between the tables.
- It is a column / columns 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.
- The table that defined s the primary key or unique key and is referenced by the foreign key is called Primary Table or Master Table.
- Features:
  - Child may have duplicates and nulls but unless it is specified.
  - Parent record can be delete provided no child record exist.
  - Master table can not be updated if child record exist.
  - Record can not be inserted in detail table if corresponding record does not exist in master table.

Constraints

- **Foreign Key**
  - **At column level**
    - Column-name data-type(size) **REFERENCES** table-name [column-name] [**ON DELETE CASCADE / ON DELETE SET NULL**]

```
CREATE TABLE bill_details
  (bill_no NUMBER (5) PRIMARY KEY ,
  client_no NUMBER(3) REFERENCES
  client_master(client_no),
  bill_amt number(6))
  /
```

Constraints

- **Foreign Key**

– **At table level**

- **FOREIGN KEY** (column-name1, column-name2, …….. column-nameN) **REFERENCES** table-name (column-name1, column-name2, …….. column-nameN) [**ON DELETE CASCADE / ON DELETE SET NULL**]

```
CREATE TABLE bill_details
(bill_no NUMBER (5) PRIMARY KEY ,
client_no NUMBER(3),
bill_amt number(6),
FOREIGN KEY (client_no) REFERENCES
client_master(client_no))
/
```

Constraints

- When constraints are defined, Oracle assigns a unique name to each constraint.
- The conversion is used by Oracle is SYS_C**n** : n is numeric value that makes the constraint name unique.
- **System table : User_Constraints**
  - It is used to store the constraints (like primary key , foreign key, check constraints, default constraints) of the tables
  - **DESC User_Constraints**
    - OWNER
    - CONSTRAINT_NAME
    - TABLE_NAME
    - CONSTRAINT_TYPE
    - R_OWNER
    - R_CONSTRAINT_NAME

    E.g. SELECT Owner, Constraint_Name, Table_Name, Constraint_Type FROM user_constraints WHERE table_name = 'CLIENT_MASTER'

Constraints

- **Unique Key**
- It allows store unique values in column as primary key.
- But it allows multiple entries of NULL into the column.
- **Features**
  - It will not allow duplicate values.
  - Unique index is created automatically.
  - A table can have more than one Unique Key.
  - Unique key can combine upto 16 columns.
  - Unique key can not be LONG or LONG RAW Data type.
  - **Column-level**
    - Column-name data-type(size) [**UINIQUE KEY / UNIQUE]**
  - **Table – level**
    - **[UNIQUE KEY / UNIQUE] (** Column1,….. Column N)

Constraints

- **Unique Key**
  - **Column-level**
    - Column-name data-type(size) [**UINIQUE KEY / UNIQUE]**
    - **E.g.**
    - CREATE TABLE Client_Master (Client_No NUMBER(3) ,
      Client_Name VARCHAR(50), Client_City VARCHAR(50) ,
      Gender CHAR(1), BirthDate DATE, Pincode NUMBER(6),
      Mobile_No NUMBER(10) **UNIQUE**)
  - **Table – level**
    - **[UNIQUE KEY / UNIQUE] (** Column1,….. Column N)
    - **E.g.**
    - CREATE TABLE Client_Master (Client_No NUMBER(3) ,
      Client_Name VARCHAR(50), Client_City VARCHAR(50) ,
      Gender CHAR(1), BirthDate DATE, Pincode NUMBER(6),
      Mobile_No NUMBER(10), **UNIQUE (Mobile_No , BirthDate))**

Constraints

- **NOT NULL Constraint**
  - **It is a column level constraint.**
  - It ensure that a table column cannot be left empty.
  - it used for mandatory columns.
  - **Column-level**
    - Column-name data-type(size) **[NOT NULL]**

    E.g.
    - **CREATE TABLE** STUDENT_MASTER
      (s_id NUMBER(2),
      s_name VARCHAR(50) **NOT NULL,**
      PRIMARY KEY (s_id));

Constraints

- **DEFAULT Constraint**
  - **It is a column level constraint.**
  - It specify the default value of a table column.
  - **Column-level**
    - Column-name data-type(size) **DEFAULT (Value)**

E.g. Majority of students city = 'ANAND'.

```
CREATE TABLE STUDENT_MASTER
  (s_id NUMBER(3),
  s_name VARCHAR(50) NOT NULL,
    s_city VARCHAR(50) DEFAULT 'ANAND' ,
  PRIMARY KEY (s_id) );
```

Constraints

- **CHECK Constraint**
  - Business rules are applied to data prior the data is being inserted into table columns.
  - This ensures that the data (records) in the table have integrity.
    - E.g.
      - Client Name should be inserted with Upper Case letters.
      - Employee Id should be start with 'E'.
      - Student ID between 1000 to 1100
      - No employee in company get salary less than Rs. 5000
  - It can be applied at a column level & table level by using CHECK clause.
  - It is a logical expression that evaluates either TRUE or FALSE.
  - It takes longer time to execute as compared to NOT NULL, Primary Key, Foreign Key & Unique Key.

Constraints

- **CHECK Constraint**
  - **Column-level**
    - Column-name data-type(size) **CHECK (** logical-expression**)**

E.g.

```
CREATE TABLE STUDENT_MASTER
(s_id NUMBER(3) CHECK (s_id BETWEEN 1000 and 1100),
 s_name VARCHAR(50) CHECK ( s_name = UPPER(s_name)),
 s_gender  CHAR(1) CHECK (s_gender IN ('M','F')),
 s_age NUMBER(2) CHECK ( s_age >= 18 ) ,
 s_city VARCHAR(50) DEFAULT  'ANAND' ,
 PRIMARY KEY (s_id)  )
```

Constraints

- **CHECK Constraint**
  - **Table-level**
    - **CHECK (** logical-expression1**), … CHECK (** logical-expressionN**)**

```
CREATE TABLE STUDENT
(s_id NUMBER(3),
 s_name VARCHAR(50) CHECK ( s_name = UPPER(s_name)),
 s_gender  CHAR(1),
 s_age NUMBER(2) CHECK ( s_age >= 18 ) ,
 s_city VARCHAR(50) DEFAULT  'ANAND' ,
CONSTRAINT c_sid CHECK (s_id BETWEEN 1000 and 1100),
CONSTRAINT c_sgender CHECK (s_gender IN ('M','F')),
 PRIMARY KEY (s_id)  )
```

Multiple Check Constraints

- **CHECK Constraint**
  - **E.g.**

```
CREATE TABLE STUDENT
(s_id NUMBER(3),
 s_name VARCHAR(50) CHECK ( s_name = UPPER(s_name)),
 s_gender  CHAR(1),
 s_age NUMBER(2) ,
 s_city VARCHAR(50) DEFAULT  'ANAND' ,
CONSTRAINT c_sga CHECK
        (s_id BETWEEN 1000 and 1100  AND
        s_gender IN ('M','F')  AND
        s_age >= 18 ),
 PRIMARY KEY (s_id)  )
```

Functions

- **Aggregate / Group Function**

It return results based on set of rows.

Aggregate function accepts two options:([DISTINCT | ALL] < EXPR >)

DISTINCT – It consider only distinct values of the argument

ALL – It consider all values including the duplicates also.

Note: if neither option is specified , the default is ALL

Group functions are statistical functions such as

- SUM : It returns total of the supplied field values.
- AVG : It returns an average of the field values.
- MIN : It returns the maximum value within the defined data set.
- MAX :It returns the maximum value within the defined data set.
- COUNT : It returns the number of rows within the defined data set

Aggregate Functions (Group Functions)

1. AVG

- Returns the average value of specified field n.

Syntax: AVG([DISDINCT | ALL] n)

Example:

SELECT AVG (basic) FROM emp;

SELECT AVG (DISTINCT basic) FROM emp;

SELECT AVG (ALL basic) FROM emp;

Functions

- **Scalar / Numeric Function**

  ABS(n) : Returns the absolute value of a number (n)

  – SELECT ABS (-32) FROM DUAL;

  – SELECT ABS (-3 + 6) FROM DUAL;

  – SELECT ABS (-3 * 2 + 6) FROM DUAL;

  – SELECT ABS (-3 + 2 * 6 + 3) FROM DUAL;

  POWER (n, m) : Returns the value of one expression raised to the power of another expression n^m

  – SELECT POWER (2 , 3) FROM DUAL;

  – SELECT POWER (16 , 1/2) FROM DUAL;

  – SELECT POWER (3 , 3) FROM DUAL;

  – SELECT POWER (27 , 1/3) FROM DUAL;

Functions

- **Scalar / Numeric Function**

  ROUND [n [,m] :

  Returns a number rounded to a certain number of decimal places. If decimal places is 0, then the result will have no decimal point.

  - SELECT ROUND (139.237,0) FROM DUAL;
  - SELECT ROUND (139.537,1) FROM DUAL;
  - SELECT ROUND (139.537,2) FROM DUAL;
  - SELECT ROUND (139.537,-1) FROM DUAL;
  - SELECT ROUND (139.537,-2) FROM DUAL;
  - SELECT ROUND (139.237) FROM DUAL;
  - SELECT ROUND (139.537) FROM DUAL;

Functions

- **Scalar / Numeric Function**

TRUNC (number, [decimal places]) :

Returns number truncated to decimal places. If decimal places is 0, then the result will have no decimal point.

- SELECT TRUNC (139.237,0) FROM DUAL;

- SELECT TRUNC (139.587,1) FROM DUAL;
- SELECT ROUND(139.587,1) FROM DUAL;

- SELECT TRUNC (139.537,-1) FROM DUAL;
- SELECT ROUND(139.537,-1) FROM DUAL;

- SELECT TRUNC (169.537,-2) FROM DUAL;
- SELECT ROUND (169.537,-2) FROM DUAL;

Functions

Scalar / Numeric Function

SQRT (n) : Returns a square root of a number n

- SELECT SQRT (4) FROM DUAL;
- SELECT SQRT (100) FROM DUAL;
- SELECT SQRT (169) FROM DUAL;

EXP (n) :Returns the base of the natural logarithm (e) raised to the power of passed numeric expression.

Hear e = 2.71828183.

- SELECT EXP (2) FROM DUAL

MOD (m , n) : Returns the remainder of one expression by diving by another expression. Reminder when m divide by n.

- SELECT MOD (10 , 2) FROM DUAL
- SELECT MOD (36 , 8) FROM DUAL

Functions

- **String Functions**

  ASCII(string) : Returns numeric value of left-most character

  – SELECT ASCII ('A') FROM DUAL

  – SELECT ASCII ('a') FROM DUAL

  – SELECT ASCII ('z') FROM DUAL

  – SELECT ASCII ('APPLE') FROM DUAL

  CHR(n) : Returns the character for each n integer passed

  SELECT CHR (97) FROM DUAL;

  SELECT CHR (65) FROM DUAL;

Functions

- **String Functions**

LENGTH (string) : Returns number of characters in argument

- SELECT LENGTH ('HELLO') FROM DUAL;
- SELECT LENGTH ('HELLO HOW ARE YOU?') FROM DUAL;

CONCAT(string1, string2) : Returns concatenation of two strings

- SELECT CONCAT ('HELLO' , 'HELLO HOW ARE YOU?') FROM DUAL;

Functions

- **String Functions**

LOWER(String) : Converts all letters in the specified string to lowercase.

- SELECT LOWER ('HELLO') FROM DUAL;
- SELECT LOWER ('HeLlO') FROM DUAL;

UPPER (String) : Converts all letters in the specified string to uppercase.

- SELECT UPPER('hello') FROM DUAL;
- SELECT UPPER ('HeLlO') FROM DUAL;

INITCAP(String) : Sets the first character in each word to uppercase and the rest to lowercase.

- SELECT INITCAP ('HELLO') FROM DUAL;
- SELECT INITCAP ('heLlO') FROM DUAL;

Functions

String Functions

LTRIM (string [, set]) :

Removes all specified characters from the left-hand side of a string till the first character not belongs to a set.

- SELECT LTRIM('AAAGDCST','A') FROM DUAL
- SELECT LTRIM('BBAAAGDCST','AB') FROM DUAL
- SELECT LTRIM('BBAACAGDCST','AB') FROM DUAL
- SELECT LTRIM('   AAGDCSTAAA ',' ') FROM DUAL

RTRIM (string [, set]) :

Removes all specified characters from the right-hand side of a string till the last character not belongs to a set.

- SELECT RTRIM('AAAGDCSTAAA','A') FROM DUAL
- SELECT RTRIM('BBAAAGDCSTBBAA','AB') FROM DUAL
- SELECT RTRIM('BBAACAGDCSTABCCAA','AB') FROM DUAL
- SELECT RTRIM('   AAGDCSTAAA   ',' ') FROM DUAL

TRIM (leading | trailing | both [<trim_char > FROM]] < string >) :

Removes all specified characters either from the beginning or the end or both of a string till the first or last character not belongs to a set

- SELECT TRIM(LEADING 'A' FROM 'AAGDCSTAAA') FROM DUAL
- SELECT TRIM(TRAILING 'A' FROM 'AAGDCSTAAA') FROM DUAL
- SELECT TRIM(BOTH 'A' FROM 'AAGDCSTAAA') FROM DUAL
- SELECT TRIM(BOTH ' ' FROM '   AAGDCSTAAA ') FROM DUAL

Functions

String Functions

LPAD (String , n [, chars]) :

Pads the left-side of a String with a specific set of characters. n indicate total size of string after padding.

- SELECT LPAD('GDCST',6, '*') FROM DUAL
- SELECT LPAD('GDCST',9, '*#') FROM DUAL
- SELECT LPAD('GDCST',8, '*#') FROM DUAL
- SELECT LPAD('GDCST',8,' ') FROM DUAL

RPAD (char , n [, char2]) :Pads the right-side of a string with a specific set of characters (when *string1* is not null). n indicates total size of string after padding.

- SELECT RPAD('GDCST',6, '*') FROM DUAL
- SELECT RPAD('GDCST',9, '*#') FROM DUAL
- SELECT RPAD('GDCST',8, '*#') FROM DUAL
- SELECT RPAD('GDCST',8,' ') FROM DUAL

Functions

- **String Functions**
  - **SUBSTR (String , Str_Position , [Length]) :**
    - **String :** Original String
    - **Str_Position:** Position to start extracting from original string.
    - **Length:** How many characters to be extract from original string
    - **Allows you to extract a substring from a string.**
    - If *start_position* is 0, It means start from the first position in the String
    - If *start_position* is a positive number, starts from specified position in the String.
    - If *start_position* is a negative number, starts from specified position from the end of the String.

Functions

SUBSTR (String , Str_Position , [Length]) :

- SELECT SUBSTR(SYSDATE,1,2) FROM DUAL;
- SELECT SUBSTR(SYSDATE,4,3) FROM DUAL;
- SELECT SUBSTR(SYSDATE,8,2) FROM DUAL;
- SELECT SUBSTR(SYSDATE,8,4) FROM DUAL;

- SELECT SUBSTR('MCA SEM-1 BATCH', 1, 3) FROM DUAL;
- SELECT SUBSTR('MCA SEM-1 BATCH', 5, 5) FROM DUAL;
- SELECT SUBSTR('MCA SEM-1 BATCH', 1) FROM DUAL;
- SELECT SUBSTR('MCA SEM-1 BATCH', 5) FROM DUAL;
- SELECT SUBSTR('MCA SEM-1 BATCH', -1) FROM DUAL;
- SELECT SUBSTR('MCA SEM-1 BATCH', -5) FROM DUAL;
- SELECT SUBSTR('MCA SEM-1 BATCH', -5,1) FROM DUAL;
- SELECT SUBSTR('MCA SEM-1 BATCH', -5,2) FROM DUAL;
- SELECT SUBSTR('MCA SEM-1 BATCH', -5,5) FROM DUAL;

Functions

- **Conversion Functions**
  - **TO_NUMBER (char [,fmt])**
    - Converts char, a CHARACTER value expressing a number, to a NUMBER datatype.
    - Formatting characters are
      - L for currency symbol
      - D or ' . ' for decimal marker
      - G or ' , 'thousands group separator
    - E.g.
      - UPDATE EMP SET basic = basic + '$1000' WHERE eno = 2;
      - UPDATE EMP SET basic = basic + TO_NUMBER('$1000','L9999') WHERE eno = 2;

Functions

- **Conversion Functions**
  - **TO_NUMBER (char [,fmt])**
- SELECT TO_NUMBER ('$1234','L9999') FROM DUAL
- SELECT TO_NUMBER ('$10,234','L99G999') FROM DUAL
- SELECT TO_NUMBER ('$10,234','L99,999') FROM DUAL
- SELECT TO_NUMBER ('$10,78,234','L99G99G999') FROM DUAL
- SELECT TO_NUMBER ('$10,78,234','L99,99,999') FROM DUAL
- SELECT TO_NUMBER ('$10,234.90','L99G999D90') FROM DUAL
- SELECT TO_NUMBER ('$10,234.90','L99,999.90') FROM DUAL

- UPDATE EMP SET basic = basic + '$1000' WHERE eno = 2;
- UPDATE EMP SET basic = basic + TO_NUMBER('$1000','L9999') WHERE eno = 2;

Functions

- **Conversion Functions**
  - **TO_CHAR (n [, fromat])**
    - Number conversion
    - Converts a value of a NUMBER datatype to a CHARACTER datatype.
    - Format characters for number may be **$, 0, 9, , , .**


```
SELECT TO_CHAR(1234,'9999') FROM DUAL;
SELECT TO_CHAR(1234,'9,999') FROM DUAL;
SELECT TO_CHAR(12.34,'99.99') FROM DUAL;
SELECT TO_CHAR(12.34, '$99.99') FROM DUAL;
SELECT TO_CHAR(12.34, '$0099.99') FROM DUAL;
```

Functions

- **Conversion Functions**
  - **TO_CHAR (date [, fromat])**
    - Date conversion
    - Converts a value of DATE datatype to a CHARACTER datatype.
    - Format characters for date may be ,MON, YY, MONTH, DAY, YEAR , HOUR, MINUTE, SECOND, HH, MM, SS etc.

E.g.

```
SELECT TO_CHAR(SYSDATE,'DD-MON-YY') FROM DUAL;
SELECT TO_CHAR(SYSDATE,'DDTH-MON-YY') FROM DUAL;
SELECT TO_CHAR(SYSDATE,'DDSP-MON-YY') FROM DUAL;
SELECT TO_CHAR(SYSDATE,'DDSPTH-MON-YY') FROM DUAL
```

Functions

- **Date Conversion Functions**
  - By default DATE datatype column/field store value in the following format.
    - 'DD – MON – YY HH:MI:SS '
  - When insert the date filed/column value in other format TO_DATE function is used.
  - TO_DATE (char [, fmt]) : Converts a character values into a date values.
    - E.g.
    - INSERT INTO d VALUES ('25-01-90')
    - INSERT INTO d VALUES (TO_DATE ('25-01-90','DD-MM-YY'))
    - INSERT INTO d VALUES (TO_DATE ('25-JAN-90','DD-MON-YY'))
    - INSERT INTO d VALUES (TO_DATE ('25-JAN-90','DD-MONTH-YY'))
    - INSERT INTO d VALUES
      (TO_DATE ('25-JAN-1980 10:55 A.M.','DD-MON-YY HH:MI A.M.'))
    - INSERT INTO d VALUES
      (TO_DATE ('25-JANUARY-80 10:55 A.M.','DD-MONTH-YY HH:MI A.M.'))
    - INSERT INTO d VALUES
      (TO_DATE ('25-01-80 10:55 A.M.','DD-MM-YY HH:MI P.M.'))
    - INSERT INTO d VALUES
      (TO_DATE ('25-JANUARY-1980 10:55 A.M.','DD-MON-YYYY HH:MI P.M.'))
    - INSERT INTO d VALUES
      (TO_DATE ('25-JAN-1980 10:55:30','DD-MON-YYYY HH:MI:SS'))

Functions

- **Miscellaneous Function**
  - Uid : Userid of the logged in user. It is unique for each user.
  - User : User name of the user who has logged in.
  - UserEnv : Returns the information about the current session.
  - Parameters
    - language
      - SELECT USERENV('language') FROM DUAL
    - SessionID
      - SELECT USERENV('sessionid') FROM DUAL
    - Terminal
      - SELECT USERENV('terminal') FROM DUAL
    - IsDBA
      - SELECT USERENV('isdba') FROM DUAL
    - Instance
      - SELECT USERENV('instance') FROM DUAL

GROUP BY Clause

- **Syntax**

  SELECT <columnName1> <columnName2> ……<columnNameN>

  AGGREGATE FUNCTION (<Expression>)

  FROM Table Name

  WHERE <Condition >

  GROUP BY <columnName1> <columnName2> ……<columnNameN>

  - The **Columns1 …. Columns N** are not used within an aggregate function and **must be included in the GROUP BY clause**
  - **Aggregate functions** are **AVG(), COUNT(), MAX(), MIN()** etc.

Select Statement :: Grouping

- **Sorting Data**
  - SELECT * FROM table-name **GROUP BY**
    { column_name1.., column_nameN [**ASC / DESC**] }

    E.g.

    Add City attribute in Student_Master table.

    ALTER TABLE Student_Master **ADD** City VARCHAR2(50);

  - SELECT city, count(stud_id) FROM Student_Master GROUP BY City

HAVING CLAUSE

- It impose a condition on the GROUP BY clause, which further filters the groups created by the GROUP BY clause.
- It filters on fields that used with aggregate function.

  - SELECT city, count(stud_id) FROM Student_Master **GROUP BY** City

  - SELECT city, count(stud_id) FROM Student_Master **GROUP BY** City **HAVING** count(stud_id) > 2;

WHERE clause & HAVING clause

- WHERE clause impose a condition on all rows in table.
- GROUP BY clause is written after WHERE clause. It makes a group on rows selected by WHERE clause.
- HAVING clause impose a condition on group. It filters on fields that used with aggregate function.
  - SELECT city, count(stud_id) FROM Student_Master **GROUP BY** City **HAVING** count(stud_id) > 2;
  - SELECT city, count(stud_id) FROM Student_Master **WHERE** city **like 'A%'** **GROUP BY** City **HAVING** count(stud_id) > 2;