**BASIC DEFINITIONS**

**Data:-** The collection of raw facts that are stored in computer media is called ‘data’.

The data may be represented in the form of numbers, text, audio, video, images etc.

Ex. The Student data contains Roll number, Name, Address, Phone etc.

**Information:-**

The data will be processed on the computer, to increase the knowledge of a person is called ‘information’. (or) The processed data is called ‘Information’.

**Data Vs Information**

* The terms data and information are closely related words.
* Data are raw facts about people, places, and objects etc. whereas information is processed data.
* To get good information good data will be collected.
* Raw materials can’t be useful to the users; hence any raw material requires some processing to make it useful.

Ex. Wood is raw material that is processed to produce useful products like tables, benches, chairs etc.

For example, the Student data is look like as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SNO | SNAME | M1 | M2 | M3 |
| 1 | Ramu | 70 | 65 | 90 |
| 2 | Anil | 56 | 75 | 86 |
| 3 | Mahesh | 75 | 80 | 74 |

From the above, we can produce the information like total and average marks, subject wise toppers, class wise toppers etc.

**Database:-**

* An organized collection of logically related data is called ‘Database’.
* Organized means the data in the database structurally arranged. So, that it can be easily stored, manipulated, and retrieved by the user.
* A Database may be of any size and complexity.

Ex. A college database contains the data about

Student (Rollno, Name, Address, Phone)

Employee (Empno, Name, Job, Salary, Deptno)

**DBMS (Database Management System):-**

* DBMS is one of the application software.
* It is used to create, maintain, and provide controlled access to the database.
* The contents present in the database can be managed through DBMS i.e. DBMS takes care of all physical storage details, data structures, relationships etc.

**RDBMS (Relational Database Management System):-**

* It is also one of the application software.
* In RDBMS, the data is maintained in the form of tables, whereas a table is a collection of rows and columns.
* It also maintains the relationships between the data (tables).
* The relationship between the data is defined by the values stored in the common columns in both tables.

**Meta Data:-**

* The data which describes the properties or characteristics of the data is called ‘Meta data’. It is nothing but data about data.
* The properties like data definitions, data structures, and relationships etc.
* Meta data describes the properties of the actual data, but not include the actual data.
* The properties like column (field) name, type, and size.

Ex. Field Name Type Size

EMPNO NUMBER 4

ENAME TEXT 20

JOB TEXT 15

SALARY NUMBER 7,2

**THE DB SYSTEM  ENVIRONMENT**

The DB system refers to an organization of components that define the collection, storage, management, and use of data within a DB environment. The DB system contains 5 major parts –

1. Hardware
2. Software
3. People
4. Procedures
5. Data
6. **Hardware:-** It refers to all of the systems physical devices, storage devices, printers, network devices and other devices.

*Physical devices* - micro computers, workstations, servers, and super computers etc.

*Network devices* - hubs, switches, routers, fiber optics etc.

*Other devices* - ATMs, ID readers etc

**b) Software:-** The most important software is the DBMS itself. To make the DB system fully function, 3 types of the skills are needed.

**Operating systems:-** It manages all hardware components and makes it possible for all other software to run on the computers.

Eg. Ms Windows, Linux, Mac ada, Unix.

**DBMS Software;-** It manages the DB within the  DB systems.

Eg. My SQL, SQL SERVER, ORACLE and  IBM DB2.

**c) People**

This component includes all users of the DB system identifies 5 types of users.

**System Administrators:-** Oversee the DB systems general operations.

**DB Administrators** also known as DBAs. They manage the DBMS and ensure that the Database functioning properly.

**DB designers** design the DB structure. If the DB design is poor, the best application programmers and DBAs cannot produce a useful DB environment.

**System Analysts and programmers** design and implement the application programs. They design and create the data entry screens, reports and procedures through which end users access and manipulate the db’s data.

**End users** are the people who use the application programs to run the organization’s daily operations.

**Eg.** Sales clerks, supervisors, manager etc.

**d) Procedures:-** Procedures are the instructions and rules that govern the design and use of the DB  system. Procedure can play an important role in a company because they enforce the standards by which business is conducted within the organization and with customers.

**e) Data:-** It is a collection facts stored in the DB. Data are the raw facts from which information is generated.

**DATA MODELS**

A **database model** is a type of **data model** that determines the logical structure of a **database ie in whi**ch manner **data** can be stored, organized, and manipulated

Within DB environment, a data model represents data structures and their characteristics, relations, and constraints.

The data models are classified into mainly 5tyes.

1. Hierarchical Model
2. Network Model
3. Relational Model
4. Entity Relationship Model
5. Object –Oriented Model

**1)Hierarchical Model**

* It was developed in 1960s to manage large amounts of data.
* Its basic structure is look like as inverted tree structure.
* It contains levels or segments. A segment is the equivalent of a record (row).
* The top layer is called root, is the parent of the segment under it.
* It satisfies one-to-many relationships.

Limitations:-

* It was complex to implement.
* It  was difficult to manage
* It lacked structural independence.
* It supports only one-to-many relationships.

Component

Component A Component B Component C

Assembly A Assembly B Assembly C Assembly D

**2) The Network Moel**:-

* It was created to represent complex data relationships.
* In this Model, the user gets the network database as a collection of records in one-to-many (1:M) relationships.
* It also supports the relationships like one-to-one (1: 1) and many-to-many (M:N) etc.

SALESREP CUSTOMER

PRODUCT INVOICE PAYMENT

INV\_LINE

In the above eg INVOICE is owned by both SALESREP and CUSTOMER. Similarly, INV\_LINE has two parents, PRODUCT and INVOICE and also CUSTOMER makes PAYMENT.

3) **The Relational Model**:-

* This model was introduced by E.F.Codd (of IBM) in 1970.
* This model is derived from the mathematical concept called relation.
* A relation is collection of rows and columns, also known as a table.
* Each row in a relation is called a tuple and each column is an attribute.
* The relational data model is implemented through RDBMS.
* The RDBMS performs the same basic functions provided by the hierarchical and network DBMS systems.
* The RDBMS hides the complexities of the relational model from the user.
* Tables are related to each other through the sharing of a common attribute.

    AGENT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A\_CODE | A\_LNAME | A\_FNAME | A\_AREACODE | A\_PHONE |

  CUSTOMER

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C\_CODE | C\_LNAME | C\_FNAME | C\_INSTYPE | C\_INS\_AMT | A\_CODE |

4) **The entity Relationship Model**:-.

* The relational model was a vast improvement over the hierarchical a network models.
* The DB designers prefer to use a graphical tool to represent the relations and their relationships. Thus the entity relationship model (ER) was introduced.
* ER model was introduced by Peter Chen in 1976.
* The graphical representation of an ER model is an Entity Relationship Diagram (ERD).
* The ER model is required the following 3 components.

1. Entities
2. Attributes
3. Relationships

**5) The Object –Oriented Model**:-

* The complex real –world problems required a data model, which is closely related to the real-world. For this object-oriented data model was introduced.
* In OODM, both data and their relationships are contained in a single structure known as an object.
* An Object contain data, relationships and operational procedures etc.
* The ODDM has the following components.
  1. Object:- is a real-world entity.
  2. Attributes describe the properties of an object.

Eg:- Person is an object, it’s attributes are name,job, salary, and Date-of-Birth etc.

Eg:- Object representation.

|  |
| --- |
|  |
| EMPLOYEE |
| EMP\_NUM |
| EMP\_NAME |
| EMP\_SALARY |
| DEPARTMENT  DEPT\_NUM |

**ER- model**

In DBMS, the data can be represented as ER- model ie Entity-Relationship model.

**ER Diagram**:

The graphical representation of an ER-model is called ER- diagram.

**Data model building blocks:**

A data model consists of the following components.

1. Entities
2. Attributes
3. Relationships
   * + - 1. **ENTITIES** :-

* An entity is a real world object, which is used to collect and store the data.
* An entity name is always in capital letters only.
* An entity is represented by the symbol rectangle.

Ex. EMPLOYEE

Types of entities

The entities are classified into 2 types.

1. Strong entity
2. Weak entity

Strong entity – The entity which exist independently in an ER-diagram is called strong entity ie it can’t depend on any other entity.

Weak entity – The entity which can depend on another entity for its existence is called weak entity. It can be represented in double lined rectangle.

Eg.

LOAN PAYMENT

**ATTRIBUTES**:-

* An attribute is the characteristic of an entity.
* An attribute name is initial capital and the remaining are in lower case letters. If it has multiple words each word initial must be capital and combined with underscore.
* Attributes are represented by ellipses (ovals) and are connected to the entity rectangle with a line.

**Eg**. an EMPLOYEE entity has the attributes like Empno, Emp\_Name, Emp\_Job, Empsal etc.

**Types of attributes:-**

The attributes are classified into various types.

1. Simple attributes
2. Composite attributes.
3. single valued attributes
4. Multi valued attributes.
5. Derived attributes.
6. Identifier (or) Key attributes.

**1) Simple attributes:-** An attribute which cannot be broken down into parts is called ‘Simple Attribute’.

**Eg:-** Empno, Job and Salary

**2) Composite Attributes:-** An attribute that can be divided into parts is called ‘Composite attribute’.

**Eg:-**

Address

Street Pincode

City State

**3) Single valued attributes:-**

An attribute which takes only a single value for an entity instance (row) is called “Single valued Attribute”.

**Eg**:- job, Salary, aadhar number etc.

**4)Multi valued attributes**;-

An attribute which contain multiple values for an entity instance is called “Multi valued Attribute”. It is represented as double lined oval.

**Eg.**

Phone Qualification

**5) Derived attributes:-**

A Derived attribute is an attribute whose value is derived (calculated) from other attribute. It is not physically stored in the database.

It is represented as dashed-line oval.

Eg. Bonus Total

**6) key (or) Identifier Attributes**:-

The attribute which is used for uniquely identifying the rows in a table is called ‘key attribute’. The key attribute may contain one or more attributes. It is mainly used for retrieving data. For key attributes, the names are underlined. They are in two types.

* 1. ***Simple key attribute***:- If the simple attribute is defined as key attribute is called simple key attribute.
  2. ***Composite key attribute***:- If the composite attribute is defined as key attribute is called composite key attribute.

Ex.

Empno Ename Job

Salary

EMPLOYEE

Address

Bonus Qualification

**Relationships**

A meaningful association between the entities is called a relationship.

The relationship names are always in lower case. It is identified by the symbol

The relationships are classified into 3 types. They are

* + - 1. One-to-one (1:1)
      2. One-to-Many (1:M)
      3. Many-to-Many (M:N)
  1. One-to-one (1:1) – If an entity instance (row) in one entity is participating with only one entity instance of another entity is called one-to-one relationship.

Ex. COLLEGE PRINCIPAL

A COLLEGE has only one PRINCIPAL

A PRINCIPAL belongs to only one COLLEGE

* 1. One-to-many (1:M) – If an entity instance in one entity is participating with no.of instances of another entity is called one-to-many relationship.

EX. DEPT EMPLOYEE

A DEPT has many EMPLOYEEs.

An EMPLOYEE belongs to one DEPT.

* 1. Many-to-many (M:N) – If no.of instances of one entity are participating with the no.of instances of another entity is called many-to-many relationship.

EX. STUDENT COURSE

A STUDENT may learn many COURSEs

A COURSE is learned by many STUDENTs.

**KEYS:-**

For uniquely identifying the rows in a table, the relational model provides the concept called keys. They are also be used to establish relationship between the tables. A key consists of one or more attributes.

The keys are classified into various types.

1. Super keys.
2. Candidate keys.
3. Primary Keys
4. Foreign keys.
5. Secondary keys.

**1) Super keys:-** .

A super key is a key that uniquely identifies each row in a table.

Eg;- In STUDENT table , a super key may be

STUD\_NUM

STUD\_NUM, STUD\_LNAME

Here, STUD\_NUM is a super key with or without additional attribute.

**2) Candidate keys:-**

A candidate key is also a super key without unnecessary attributes, that is, a minimal super key.

Eg. STUD\_NUM, STUD\_LNAME is a super key, but it is not a candidate key because STUD\_NUM by itself is a candidate key.

**3) Primary keys:-**

* + In a table, each row is uniquely identified by the primary key.
  + The primary key is the candidate key, used to identify a row uniquely. So a primary key is a super key, as well as a candidate key.
  + The table which contains primary key can exhibit ***entity integrity***. To maintain entity integrity, a null value is not permitted in the primary key.
  + A primary key can prevent both null values and duplicate values in a column.

Null value:-

Null value means

- An unknown attribute value.

- A known, but missing attribute value.

- A not applicable value.

**4) Foreign Keys:-**

* In a relational database, we can provide a link between two database tables. For this foreign keys are used.
* A foreign key is an attribute whose values match the primary key values in the related table.
* A foreign key contains either matching primary key values or nulls.
* The table which contains the foreign key which shows the ***referential integrity***.

Eg:- Deptno  is the primary key in the DEPT table and it  occurs as a foreign key in the EMPLOYEE table.

A foreign key in a table can always refer the values of primary key in a related table.

DEPT

EMP

**5) Secondary Key**:-

* A Secondary Key is a key that is also used for uniquely identifying the rows in a table.
* The attribute which is used for retrieving the data (uniqueness) in a table other than a primary key is called ‘Secondary Key’.

Ex. Sometimes the attribute employee name is used for retrieving the data of an employee, is called secondary key. Here, the primary key is employee number.

Note: The secondary key cant give a single result ie multiple rows will be retrieved.

**ORACLE**

* **It** is a programming language used for managing data in relational database management systems (RDBMS).
* **Oracle was** developed by Oracle Corporation.
* Oracle can provide the following 2 tools.
  + - * 1. SQL (Structured Query Language)
        2. PL/SQL (Procedural Language)

**SQL**

* SQL was initially developed at IBM (**International Business Machines**) by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s.
* It can also called as SEQUEL (structured English Query Language).
* SQL is the natural language for all RDBMS softwares such as, teradata Oracle, My-SQL, SQL server, db/2.
* SQL is not a case sensitive language, it is case insensitive language.
* It is declarative language that is it is not a procedural language.
* To operate SQL some of the commands called SQL commands are provided.
* Every command in SQL should end with ;
* Using SQL, we can
* Create Databases
* Create database Objects like Tables, Views, Synonyms etc.
* Insert or Delete or modify the data in a Database Object (table/view/synonym).
* Maintain the constraints called rules on data etc.

**SQL COMMANDS:-**

The SQL commands are classified into 5 types. They are

1. DDL Commands (Data Definition Language)
2. DML commands (Data Manipulation Language)
3. DRL Commands (Data Retrieval Language)
4. DCL Commands (Data Control Language)
5. TCL Commands (Transaction Control Language)

To connect oracle

User Name : scott password : tiger

1. **DDL Commands (Data Definition Language)**

They are used to define the database objects like tables, views, synonym etc. they are used to define the structure of a database object (table).

They are:

CREATE – used to create a table

ALTER - to change the structure of a table

RENAME – to change the name of a table

DROP - to delete a table

TRUNCATE – to delete the data, but not the structure

1. **DML Commands (Data Manipulation Language)**

They are used to manipulate the data in a database object (table).Manipulation means inserting , deleting and modifying the data in a table.

The DML commands are:

INSERT - to insert rows

DELETE – to delete rows

UPDATE – to modify rows

1. **DRL Commands (Data Retrieval Language)**

They are used to retrieve or display the of a database object (table).

Eg. SELECT – to display the rows

1. **DCL Commands (Data Control Language)**

They are used to control the accessing of one user data by another user in db. With this, we can grant or cancel the permission on user data. The DCL commands are:

Eg. GRANT – to grant permission

REVOKE – to cancel the permission

1. **TCL Commands (Transaction Control Language)**

A transaction is a collection of operations. A transaction may contain one or more SQL operations (commands). The TCL commands are:

Eg COMMIT - to save db operations (transaction)

ROLLBACK - to cancel the operations

SAVEPOINT - to group the operations in a transaction

**Datatypes in SQL**

In RDBMS, the data can be stored in the form of tables, whereas a table is a collection rows and columns. The data can be stored underlying a column. To specify the type of data a column can stored, datatypes are used.

A datatype specifies the type of data a column of a table can allowed or stored. Like other languages, SQL also support various types of datatypes. They are:

1. Char
2. Varchar/varchar2
3. Number
4. Date
5. Long
6. Raw
7. Long raw

**char** – It represents a fixed length character data. It allows max. upto 2000 characters.

Syntax: colname datatype(size)

Ex. ename char(20)

***Note:*** if the data size is < col size then spaces will be added to the data to make it fixed length.

* + - * 1. **Varchar/varchar2** – It represents a variable length character data. In this, spaces can’t be added to the data. It allows max. upto 4000 characters.

Ex: job varchar(20)

* + - * 1. **Number** – This datatype represents both integer and floating data.

1. *Number(size)* – represents an integer data

Ex. Eno number(4) – allows from 1 to 9999

1. *Number(size,dec)* – represents a floating value.

Here, size – total no.of digits , dec – no.of decimals

Ex. salary number(7,2) - allows from 1 to 99999.99

1. **Long** – it also represent the character data > varchar range. It is used for storing large amount of character data ie a file

Ex. description long

1. **Date** – This datatype can represent the date and time values (data). By default it is represented as character data.

Ex. dateofjoin date

The default date format in sql is ***: 28-sep-22 (dd-mon-yy***).

1. **Raw –** This datatype can represent the binary data.
2. **Longraw** – It also represent the binary data ie images, photo, audio, video

Ex. photo raw

Note:

* + - * 1. For number datatype size cannot be specified then by default 38 digits can be allowed.
        2. In SQL, the character and date values are enclosed in single quotes.

**DDL commands:-**

DDL stands for Data definition language. The DDL commands are:

CREATE

ALTER

RENAME

DROP

TRUNCATE

**CREATE command:**

This command is used to create the database objects like tables, views, synonyms, sequences, users, etc. with this we can specify the structure of the table.

Syntax: CREATE TABLE tablename(

colname datatype(size),

colname datatype(size), ---- );

EX. SQL> CREATE TABLE EMPLOYEE( ENO NUMBER(4), ENAME CHAR(20),

JOB CHAR(15), SALARY NUMBER(7,2) );

**Describe command**: This command is used to display the structure of a database object (table). It is a sql\*plus command.

Syntax: DESC[RIBE] tablename

Ex. SQL> DESC STUD

Name Null? Type

----------------------------------------- -------- ----------------------------

SNO NUMBER(3)

SNAME CHAR(20)

COURSE CHAR(15)

FEE NUMBER(7,2)

**ALTER command–** This command is used to alter or change the structure of a table. With this, we can add a new column to a table or modify the datatype or size of column.

Syntax: ALTER TABLE tablename

***alteroption*** (colname datatype(size), …);

Where alter option represents

1. ADD – adds a new column to a table
2. MODIFY – modify the datatype or size of a column

Ex. SQL> ALTER TABLE STUD ADD DURATION NUMBER(2);

SQL> DESC STUD

Name Null? Type

----------------------------------------- -------- -------------

SNO NUMBER(3)

SNAME CHAR(20)

COURSE CHAR(15)

FEE NUMBER(7,2)

DURATION NUMBER(2)

SQL> ALTER TABLE STUD MODIFY (SNAME VARCHAR(15), COURSE VARCHAR(10) );

SQL> DESC STUD

Name Null? Type

----------------------------------------- -------- ----------------------------

SNO NUMBER(3)

SNAME VARCHAR2(15)

COURSE VARCHAR2(10)

FEE NUMBER(7,2)

DURATION NUMBER(2)

**RENAME command**

This command is used to change the name of a table (object).

Syntax: RENAME tablename TO newname;

Ex. SQL > RENAME STUD TO STUDENT;

**DROP command**

It is used to delete a table from a database. It deletes both the data (rows) and structure permanently.

Syntax: DROP TABLE tablename;

EX. SQL> DROP TABLE SAMPLE;

**TRUNCATE command**

It is also similar to drop command. It deletes only the data but not the structure.

Syntax: TRUNCATE TABLE tablename;

EX. SQL> TRUNCATE TABLE STUDENT;

***To create a table from existing table***

Syntax: CREATE TABLE tablename as SELECT statement;

Sql> CREATE TABLE EMP11 AS SELECT \*FROM EMP;

***To display all tables in a database***

SQL> SELECT \*FROM TAB;

**DML COMMANDS**

They are used to manipulate the data of a database object (table). With this, we can insert, delete and modify the data in a table. For this, the following commands are provided. They are

1. **INSERT**
2. **UPDATE**
3. **DELETE**

**INSERT:-**

This command is used to insert data (rows) into a table. With this we can insert either a single or multiple rows into a table.

* 1. ***To insert a single row***

Syntax:- INSERT INTO tablename VALUES(value1,’value2’ , … );

Note: the character and date values are enclosed in single quotes.

* 1. ***To insert multiple rows:-***

SYN: INSERT INTO tablename

VALUES(&colname, ‘&colname’ , &colname ,….);

**/ command:**

The command typed at SQL prompt will be stored in SQL buffer. To execute the command in SQL buffer ‘/’ command is used.

* 1. ***To insert specified columns data***

Syntax: INSERT INTO tablename (col1,col2, …)

VALUES (val1,val2, … );

Note: for missing columns NULL VALUE will be inserted.

***To insert null value into a column:***

In SQL, the null values are entered into a table column in 2 ways. They are

* 1. Implicit method
  2. Explicit method

**Implicit method:**

If the null values are inserted into a column automatically by SQL is called implicit method. If the specified columns data are entered then for the missing columns null values are inserted by SQL.

Syn: INSERT INTO tablename( colname, colname) VALUES(val1,val2);

**Explicit method:**

If the null values are given by the user is called explicit method. The null values are-

For character data – blank

For numeric data - null

**SELECT Command:-**

It is one of the data retrieval command. It is used to retrieve data from a database object (table) ie to display the data in a table.

With this, we can display all rows or specified rows or specified columns.

SYN: SELECT \*|col1,col2, …

FROM tablename

[WHERE clause];

Where \* - represents all columns

Col1, col2, … - represents specified columns

Where clause – represents specified rows

Note: if where clause omitted then all rows will be retrieved.

**WHERE clause:-**

It is useful to represent specific data (rows) in a table, that is, either a single row or multiple rows. It can be applied with the SQL commands like SELECT, DELETE or UPDATE.

Syn: sql statement WHERE condition;

Here, sql statement may be SELECT OR UPDATE OR DELETE

EX.

SQL> SELECT \*FROM EMP WHERE ENAME='MARTIN';

SQL> SELECT \*FROM EMP WHERE DEPTNO=30;

SQL> SELECT EMPNO, ENAME, JOB, SAL, DEPTNO FROM EMP WHERE EMPNO=7900;

SQL> SELECT \*FROM EMP WHERE JOB='MANAGER';

SQL> SELECT \*FROM EMP WHERE SAL>1200;

**DISTINCT**

It is used to retrieve unique values in a table column. It is useful to retrieve different no.of jobs or departments in an organization. It is applied in select statement.

Syntax: SELECT DISTINCT colname FROM tablename;

Ex.

SQL> SELECT DISTINCT DEPTNO FROM EMP;

SQL> SELECT DISTINCT JOB FROM EMP;

SQL> SELECT DISTINCT MGR FROM EMP;

**UPDATE command:**

This command is used to update or modify the data in a table. With this we can change the data of one or more columns in a table.

Syntax: UPDATE tablename SET colname=value, colname = value, ..

WHERE condition;

Ex.

SQL> UPDATE EMPLOYEE SET ENAME='CHANDINI', JOB='PROGRAMMER', SALARY=4500

WHERE ENO=104;

SQL>UPDATE EMPLOYEE SET JOB='ANALYST', SALARY=9000 WHERE ENAME='RAKESH';

TO UPDATE MULTIPLE EMPLOYEES

SQL>UPDATE EMPLOYEE SET JOB='&JOB', SALARY = &SALARY WHERE ENO=&ENO;

SQL> UPDATE EMPLOYEE SET SALARY= SALARY+SALARY\*3/100 WHERE DEPTNO=10;

**DELETE command:-**

This command is used to delete the data (ROWS) in a table. With this we can delete either a single row, multiple rows or all rows in a table.

Syntax: DELETE FROM tablename [WHERE clause];

Note:

* 1. If where applied – then condition satisfied rows will be deleted
  2. If where omitted – then all rows will be deleted
  3. To retrieve the deleted rows apply ROLLBACK command

EX.

SQL> DELETE FROM EMPLOYEE WHERE DEPTNO=10;

SQL>SELECT \*FROM EMPLOYEE;

SQL> ROLLBACK;

SQL>SELECT \*FROM EMPLOYEE;

**OPERATORS**

An operator is a symbol which can perform either an arithmetical or logical operations on data. Like other languages SQL also support different types of operators.

They are:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. SQL special operators
5. Arithmetic operators – They can perform the arithmetical operations like addition, subtraction, multiplication etc on data.

They are: +, - ,\*, /, mod ()

EX.

SQL> SELECT EMPNO, ENAME, JOB, SAL, SAL\*3/100 FROM EMP11;

SQL> SELECT EMPNO, ENAME, JOB, SAL"SALARY", SAL\*3/100 "BONUS", SAL+SAL\*3/100"TOTAL" FROM EMP;

SQL> UPDATE EMPLOYEE SET SALARY= SALARY+SALARY\*3/100 WHERE DEPTNO=10;

1. Relational operators

They are used to compare any two values. They can form a condition. So, they returns a Boolean value either true or false when comparing. They are applied in where clause of select or update or delete statements.

They are: > , < , >= ,<= , = , <> - not equal to

SQL> SELECT \*FROM EMP WHERE SAL>=2000;

SQL> SELECT \*FROM EMP WHERE DEPTNO=30;

SQL> SELECT \*FROM EMP WHERE DEPTNO<>30;

SQL> SELECT \*FROM EMP WHERE JOB='MANAGER';

SQL> SELECT \*FROM EMP WHERE MGR=7698;

SQL> SELECT \*FROM EMP WHERE EMPNO=7844;

SQL> SELECT \*FROM EMP WHERE ENAME='TURNER';

1. Logical operators

They are used to evaluate more than one conditions at a time. They also returns a Boolean value either true or false when comparing.

They are: AND, OR, NOT

1. AND:- if all conditions are true then it returns true, otherwise false
2. OR: - if all conditions are false then it returns false, otherwise true
3. NOT: - it evaluates the negation of a condition, ie. If a condition is true then it evaluates to false

EX.

SQL> SELECT \*FROM EMP WHERE SAL>=1500 AND SAL<=3000;

SQL> SELECT \*FROM EMP WHERE DEPTNO=30 AND JOB='CLERK';

SQL> SELECT \*FROM EMP WHERE DEPTNO=20 AND SAL>1500;

SQL> SELECT \*FROM EMP WHERE JOB='CLERK' AND SAL<1200;

Using OR operator

SQL> SELECT \*FROM EMP WHERE JOB='MANAGER' OR DEPTNO=20;

SQL> SELECT \*FROM EMPLOYS WHERE JOB='MANAGER' OR JOB='SALESMAN';

SQL> SELECT \*FROM EMP WHERE DEPTNO=10 OR MGR=7698;

SQL> SELECT \*FROM EMP WHERE DEPTNO=10 OR DEPTNO=30;

SQL> SELECT \*FROM EMP WHERE DEPTNO=20 OR SAL<1500;

Using NOT operator

SQL> SELECT \*FROM EMP WHERE not deptno=30; ( deptno<>30)

SQL> SELECT \*FROM EMP WHERE NOT JOB='MANAGER'; (job<>’MANAGER’)

1. **SQL special operators**

The operators that are used in SQL only are called SQL special operators. They are:

1. BETWEEN … AND…
2. IN
3. IS NULL
4. LIKE
5. ***BETWEEN start AND end***

This operator can retrieve the rows that are available in a given range of values. It is applied on numeric columns (data) only. It is an alternative to logical AND operator.

Syn: colname BETWEEN start AND end

EX. SQL> SELECT \*FROM EMP WHERE SAL BETWEEN 1200 AND 3000;

Using logical and :-

SQL> SELECT \*FROM EMP WHERE SAL>=1200 AND SAL<=3000;

To display the employees whose number lies in between 7500 and 7800

SQL> SELECT \*FROM EMP WHERE EMPNO BETWEEN 7500 AND 7800;

***Not BETWEEN***

It retrieves the rows that are not retrieved by BETWEEN operator.

Ex. SQL> SELECT \*FROM EMP WHERE SAL NOT BETWEEN 1200 AND 3000;

1. ***IN operator:***

It is used to retrieve the rows which column value is available in a given list of values. It is an alternative to the logical OR operator.

Syntax: colname IN(value1,value2,…)

***NOT IN*** – it retrieve the rows that are not retrieved by IN operator

Ex. 1) To display the employees working department 10 or 30

SQL> SELECT \*FROM EMP WHERE DEPTNO IN(10,30);

Using logical or operator:

SQL> SELECT \*FROM EMP WHERE DEPTNO=10 OR DEPTNO=30;

2)To display clerks, managers and salesmans list

SQL> SELECT \*FROM EMP WHERE JOB IN('CLERK','MANAGER','SALESMAN');

SQL> SELECT \*FROM EMP WHERE JOB NOT IN('CLERK','MANAGER','SALESMAN');

3)To display the employees whose manager id is 7782 or 7698

SQL> SELECT \*FROM EMP WHERE MGR IN(7782,7698);

**C) *IS NULL:***

It retrieve the rows which column value contain the NULL values.

Syntax: colname IS NULL

***IS NOT NULL*** –it retrieve the rows that are not retrieved by IS NULL operator

EX.

* 1. Display the employees who are not getting commission

SQL> SELECT \*FROM EMP WHERE COMM IS NULL;

* 1. Display the employees who are getting commission

SQL> SELECT \*FROM EMP WHERE COMM IS NOT NULL;

* 1. Display the employees who are not having manager ID

SQL> SELECT \*FROM EMP WHERE MGR IS NULL;

***d) Like operator***:

It is used to retrieve the rows which column value matches a given pattern ie portion of text. It is applied on character columns only.

Syntax: colname like ‘pattern’

This operator uses the following 2 special symbols for pattern matching.

1. % - represents a group of characters
2. \_ - represents a single character

Ex.

1. Display the employees whose name start with the letter ‘S’

SQL> SELECT \*FROM EMP WHERE ENAME LIKE 'S%';

1. Display the employees whose name contain the second letter is ‘A’

SQL> SELECT \*FROM EMP WHERE ENAME LIKE '\_A%';

1. Display the employees whose name contain the letter ‘A’

SQL> SELECT \*FROM EMP WHERE ENAME LIKE '%A%';

1. Display the employees whose name end with the letter ‘R’

SQL> SELECT \*FROM EMP WHERE ENAME LIKE '%R';

1. Display the employees whose name contain 4 letters only

SQL> SELECT \*FROM EMP WHERE ENAME LIKE '\_\_\_\_';

1. Display the employees who joined in 1981

SQL> SELECT \*FROM EMP WHERE HIREDATE LIKE '%-%-81';

1. Display the employees who joined in DECEMBER 1981

SQL> SELECT \*FROM EMP WHERE HIREDATE LIKE '%-DEC-81';

1. Display the employees whose job contain the letter ‘A’

SQL> SELECT \*FROM EMP WHERE JOB LIKE '%A%';

***NOT LIKE*** – It retrieve the rows that are not retrieved by LIKE operator

Ex.

SQL> SELECT \*FROM EMP WHERE ENAME NOT LIKE '%A%';

SQL> select \*from emp WHERE HIREDATE NOT LIKE '%-%-81';

SQL> SELECT \*FROM EMP WHERE ENAME NOT LIKE ‘S%’;

***ORDER BY clause***

It is used to display the rows in a table in sorting order ie. Either in ascending or descending order based on one or more columns. By default it can display in ascending order. It is applied with SELECT statement.

In a table, the values are arranged in ascending order based on primary key by default.

Syntax: ***SELECT statement ORDER BY col1 [, col2 , …] [ASC|DESC];***

Here, ASC – represents ascending order

DESC – represents descending order

If multiple cols applied then the values are sorted based on col1 first then by col2

**Ex.** SQL> SELECT \*FROM EMP ORDER BY ENAME;

SQL> SELECT \*FROM EMP ORDER BY ENAME DESC;

SQL> SELECT EMPNO, ENAME, JOB, SAL, DEPTNO FROM EMP ORDER BY DEPTNO;

SQL> SELECT \*FROM EMP ORDER BY DEPTNO, SAL;

SQL> SELECT \*FROM EMP ORDER BY JOB,SAL DESC;

SQL> SELECT \*FROM EMP ORDER BY HIREDATE;

**INTEGRITY CONSTRAINTS**

In general a table column can allow the null value and duplicate values. Sometimes we need to eliminate those values in a table column. For this, Oracle SQL provides the feature called ‘Integrity Constraints’. Constraints are nothing but rules. The constraints are defined to place restrictions on data of a column. SQL support the following 5 types of Integrity constraints. They are:

1. NOT NULL
2. UNIQUE
3. PRIMARY KEY
4. FOREIGN KEY
5. CHECK

The constraints are also defined with CREATE TABLE command and ALTER TABLE command. The constraints are defined with/without constraint name.

The constraints are created in 2 ways. They are

1. Column level constraints
2. Table level constraints

***Column Level Constraints***

If a constraint is defined along with the column definition in Create table command is called ‘column level constraint’.

SYN: CREATE TABLE tablename(

Colname datatype(size) [CONSTRAINT constraintname] constraint\_type,

Colname datatype(size) , ….);

***Table level Constraints***

If a constraint is defined at the end of the table creation ie. After defining all columns is called ‘table level constraint’.

Syn:

CREATE TABLE tablename(

Colname datatype(size),

Colname datatype(size) , …. ,

CONSTRAINT constraint\_name constraint\_type(colname)

);

**NOT NULL constraint**

In general a table column can allow null values by default. To eliminate the null values in a table column then NOT NULL constraint is used. This constraint is defined at column level only.

Syntax: CREATE TABLE tablename(

Colname datatype(size) [CONSTRAINT constraintname] NOT NULL,

Colname datatype(size) , ….);

Note: The null values for

character column – left blank

numeric column - NULL

EX. SQL> CREATE TABLE SACHIN (ENO NUMBER(4) NOT NULL,

ENAME VARCHAR(20) NOT NULL, JOB VARCHAR(15), SALARY NUMBER(7,2));

SQL> CREATE TABLE SACHIN11 (ENO NUMBER(4) CONSTRAINT ENO\_NNULL NOT NULL, ENAME VARCHAR(20) CONSTRAINT ENAME\_NT NOT NULL,SAL NUMBER(5));

**UNIQUE constraint**

In general, a table column can allow duplicate values. To prevent duplicate values in a table column UNIQUE constraint is used. It will be created at both column level and table level. It is identified by the keyword UNIQUE.

**Column level**

Syntax: CREATE TABLE tablename(

Colname datatype(size) [CONSTRAINT constraintname] UNIQUE,

Colname datatype(size) , ….);

**EX.**

SQL> CREATE TABLE EMPS(ENO NUMBER(3) UNIQUE, ENAME VARCHAR(10) NOT NULL,

JOB VARCHAR(15), SAL NUMBER(7,2));

Constraint With Name

SQL> CREATE TABLE EMPS1(ENO NUMBER(3) CONSTRAINT ENO\_UQ UNIQUE,

ENAME VARCHAR(10) NOT NULL, JOB VARCHAR(15), SAL NUMBER(7,2));

**Table level:**

Syntax: CREATE TABLE tablename(

colname datatype(size), Colname datatype(size) , ….,

CONSTRAINT constraintname UNIQUE(colname)

);

SQL> CREATE TABLE EMPS2(ENO NUMBER(3), ENAME VARCHAR(10) NOT NULL,

JOB VARCHAR(10), SAL NUMBER(7,2),

CONSTRAINT ENO\_UQ1 UNIQUE(ENO));

**PRIMARY KEY**

In general, a table column can allow both null values and duplicate values. But sometimes we require to eliminate these values. For this, SQL provides Primary key. The primary key can’t allow both null values and duplicate values in a table column. It will be worked like as both NOT NULL and UNIQUE constraints.

The primary keys are mainly used for uniquely identifying the rows in a table. A table can have only one primary key. The primary keys also defined in both column level and table level.

PRIMARY KEY = NOT NULL + UNIQUE

**Column level**

SYN: CREATE TABLE tablename(

Colname datatype(size) [CONSTRAINT constraintname] PRIMARY KEY,

Colname datatype(size) , ….);

EX.

SQL> CREATE TABLE EMP8(ENO NUMBER(4) PRIMARY KEY, ENAME VARCHAR(10) NOT NULL,

JOB VARCHAR(10), SAL NUMBER(7,2), DEPTNO NUMBER(2));

With Constraint Name

SQL> CREATE TABLE EMP81(ENO NUMBER(4) CONSTRAINT ENO\_P PRIMARY KEY,

ENAME VARCHAR(10) NOT NULL, JOB VARCHAR(10), SAL NUMBER(7,2),

DEPTNO NUMBER(2));

**Table level:**

Syntax:

CREATE TABLE tablename(

Colname datatype(size), colname datatype(size) , ….,

CONSTRAINT constraintname PRIMARY KEY(colname)

);

SQL> CREATE TABLE EMP82(ENO NUMBER(4), ENAME VARCHAR(10) NOT NULL,

JOB VARCHAR(10), SAL NUMBER(5),

CONSTRAINT ENO\_P1 PRIMARY KEY(ENO));

**FOREIGN KEY**

The Foreign keys are mainly used to establish a relationship between the database tables. A foreign key in one table can always refer the values in primary key of a related table or NULL value. When we define a foreign key, then it can allow matching primary key values in the related table only. It is also be created in two ways – at column level and at table level.

To create a foreign key as column constraint

Syn: CREATE TABLE tablename(

Colname datatype(size) REFERENCES parent\_tablaname,

Colname datatype(size) , ….);

Ex.

SQL> CREATE TABLE DEPART(DEPTNO NUMBER(2) CONSTRAINT DNO\_PK1 PRIMARY KEY, DNAME VARCHAR(15) NOT NULL, HOD VARCHAR(10));

SQL> CREATE TABLE EMPLOY(ENO NUMBER(4) CONSTRAINT ENOPK PRIMARY KEY,

ENAME VARCHAR(10) NOT NULL,JOB VARCHAR(15), SAL NUMBER(7,2),

DEPTNO NUMBER(2) REFERENCES DEPART);

To create a foreign key with constraint name:

Syn: CREATE TABLE tablename(

Colname datatype(size) CONSTRAINT constraintname REFERENCES primarykeytablaname,

Colname datatype(size) , ….);

SQL> CREATE TABLE EMPLOY1(ENO NUMBER(4) ***constraint enopk1 primary ke****y*,

ENAME VARCHAR(10) NOT NULL,JOB VARCHAR(15), SAL NUMBER(7,2),

DEPTNO NUMBER(2) ***constraint dnofk references depart***);

TO create foreignkey as table constraint

Syntax:

CREATE TABLE tablename(Colname datatype(size), Colname datatype(size) , …. ,

CONSTRAINT constraint\_name FOREIGN KEY(colname) REFERENCES primarykey\_tablename

);

EX.

SQL> CREATE TABLE EMPLOY2(ENO NUMBER(4) ***constraint enopk2 primary ke****y*,

ENAME VARCHAR(10) NOT NULL,JOB VARCHAR(15), SAL NUMBER(7,2),

DEPTNO NUMBER(2),

***constraint dnofk1 FOREIGN KEY(DEPTNO) references depart***);

**CHECK constraint**

**It** is used to validate the data before inserting into a table column. It is mainly used for applying the user constraints on a table ie user specified rules on a column. It is also be created at both column level and table level.

***To create at Column level***

Syn:

CREATE TABLE tablename(

Colname datatype(size) [CONSTRAINT constraintname] CHECK(colname condition) ,

Colname datatype(size), …. );

EX.

SQL> CREATE TABLE EMP10(ENO NUMBER(4) PRIMARY KEY, ENAME VARCHAR(10) NOT NULL, JOB VARCHAR(15), SAL NUMBER(5) CHECK(SAL BETWEEN 5000 AND 50000));

With Constraint Name

SQL> CREATE TABLE EMP11(ENO NUMBER(4) PRIMARY KEY,

ENAME VARCHAR(10) NOT NULL, JOB VARCHAR(10),

SAL NUMBER(5) CONSTRAINT SAL\_CK CHECK(SAL BETWEEN 8000 AND 30000));

***To create at Table Level:***

Syntax: CREATE TABLE tablename(

Colname datatype(size), Colname datatype(size), …. ,

[CONSTRAINT constraintname] CHECK(colname condition);

EX.

SQL> CREATE TABLE EMP12(ENO NUMBER(4) CONSTRAINT ENMPK PRIMARY KEY,

ENAME VARCHAR(10) NOT NULL, JOB VARCHAR(15), SAL NUMBER(7,2),

CONSTRAINT JOBCHK CHECK(JOB IN('CLERK','MANAGER','SALESMAN')),

CONSTRAINT SALCHK CHECK(SAL BETWEEN 5000 AND 30000));

**BUILT-IN FUNCTIONS**

The functions that are already defined or provided in SQL are called ‘Built-in or Pre-defined functions’. The functions are classified into two types. They are:

* 1. Single Row Functions
  2. Multi Row Functions

**Single Row Functions**

The functions which returns a result for each row in a query are called single row functions. They are:

1. Numeric functions
2. Character functions
3. Date functions
4. Conversion functions
5. General functions

**Multi Row Functions**

The functions which returns a single result for a group rows are called multi row functions. They are also called as **‘*Group*’** or ‘***Aggregate***’ functions.

**SINGLE ROW FUNCTIONS**

1. ***Numeric Functions***

The functions which can perform the operations on numeric data only are called Numeric Functions.

1. ABS() - It returns the absolute value of a given column/value.

SYN: ABS(col/value)

Ex. SELECT ABS(-12),abs(12) FROM DUAL; O/P – 12 12

1. SQRT() – It returns the square root of a given col/value

Syn:- SQRT(col/value)

EX. SELECT SQRT(25) FROM DUAL; O/P – 5

1. POWER() – It returns the power value of m raised to n power

SYN: POWER(col/val , n)

EX. SELECT POWER(2,5) FROM DUAL; O/P – 32

1. MOD() - it returns the remainder of a col/value divisible by a no

SYN: MOD(col/value ,n)

EX. SELECT MOD(10,4) FROM DUAL; O/P – 2

1. CEIL() - It returns the smallest integer >= a given col/value

SYN: CEIL(col/value)

EX. SELECT CEIL(10.56) FROM DUAL; O/P – 11

1. FLOOR() – it returns the largest integer <= a given col/value

SYN: FLOOR(col/value)

EX. SELECT FLOOR(10.56) FROM DUAL; O/P – 10

1. Sign() – it returns the sign of a given col/val.

Syn: SIGN(n)

Return description

+1 if n is +ve

-1 if n is –ve

0 if n is 0

EX. SELECT SIGN(10), SIGN(-10), SIGN(0) FROM DUAL;

1. ROUND() – It is used to round a given no to a specified number of decimals.

Syntax: ROUND(col/value , n)

Where ‘n’ may be

1. +ve – rounds the digits in decimal part
2. –ve – rounds the digits in integer part
3. 0 - rounds to zero decimals

Note: if the rounded digit is >5 then +1 will be added to the prev. digit

SQL> SELECT ROUND(156.586723,2) , ROUND(156.586723,-1), ROUND(156.586723) FROM DUAL;

O/P - ROUND(156.586723,2) ROUND(156.586723,-1) ROUND(156.586723)

------------------- -------------------- -----------------

156.59 160 157

1. TRUNC() – it is also similar to ROUND(), but it can’t follow any rule for rounding the digits.

Syntax: TRUNC(col/value , n)

SQL> SELECT TRUNC(156.586723,2) , TRUNC(156.586723,-1), TRUNC(156.586723) FROM DUAL;

O/P - TRUNC(156.586723,2) TRUNC(156.586723,-1) TRUNC(156.586723)

------------------- -------------------- -----------------

156.58 150 156

**Character Functions:**

The functions which can perform the operations on character data only are called ‘Character Functions’. They are:

1. ***Length()*** – It returns an integer representing the length of a given col/value. Length means total number of characters including spaces.

Syntax: length(col/val)

EX. SELECT LENGTH('WELCOME') FROM DUAL; O/P- 7

SELECT ENAME, LENGTH(ENAME) FROM EMP;

1. ***Upper()*** - It converts a given col/value into upper case

Syntax: upper(col/val)

EX. SELECT UPPER('welcome to BDPS') FROM DUAL;

1. ***Lower()*** – it converts a given col/val into lowercase.

EX. SELECT LOWER('welcome to BDPS ') FROM DUAL;

1. ***Initcap***() – It converts initial letter as capital for each word in a given character data.

SYN: INITCAP(col/val)

Ex. SELECT INITCAP('WELCOME TO BDPS') FROM DUAL;

1. ***Concat()*** – it is used to add one string to another.

Syntax: concat(char1, char2)

Ex. SQL> SELECT CONCAT('WELCOME' , ' TO BDPS') FROM DUAL;

SQL> SELECT CONCAT(CONCAT(ENAME,' IS '), JOB) FROM EMP;

1. ***SUBSTR()*** – It returns a sub string in a given col/value ie returns the portion of text from a given position.

Syntax: substr(col/val , pos[,n])

Where pos – represents the starting position of sub string

n – Represents no.of characters from pos, if it is omitted then it returns the substring from the position to end.

Ex. SELECT SUBSTR('WELCOME TO BDPS',4) FROM DUAL;

SELECT SUBSTR('WELCOME TO BDPS',4,4) FROM DUAL;

1. ***Replace()*** – This function replaces char1 with char2 in a given col/value.

Syntax: replace(col/val , char1 , char2)

Ex. SELECT REPLACE('WELCOME SACHIN! WELCOME DHONI','COME','DONE') FROM DUAL;

SELECT REPLACE('JACK AND JUE','J','BL') FROM DUAL;

1. ***Lpad()*** – It add spaces or char in left side of a given col/val when you specify the width.

Syntax: lpad(col/val , width, [char])

EX. SELECT LPAD('WELCOME',15) FROM DUAL;

SELECT LPAD('WELCOME',15,’\*’) FROM DUAL;

1. ***Rpad()-*** It adds spaces or char in right side of a given col/val when you specify the width

Syntax: rpad(col/val , width, [char])

EX. SELECT RPAD('WELCOME',15) FROM DUAL;

SELECT RPAD('WELCOME',15,’\*’) FROM DUAL;

1. ***Ltrim()*** – it removes leading spaces or char in left side of a col/value.

***Syntax: ltrim(col/val ,[char])***

EX. SELECT LTRIM(' WELCOME ') FROM DUAL;

SELECT LTRIM('ABABABWELCOME ',’AB’) FROM DUAL;

SELECT LTRIM('\*\*\*\*WELCOME\*\*\*\*','\*') FROM DUAL;

1. ***Rtrim(***) – It removes trailing spaces or char in right side of a col/val.

***Syntax: rtrim(col/val[,char])***

EX. SELECT RTRIM(' WELCOME ') FROM DUAL;

SELECT RTRIM(‘\*\*\*\*\*WELCOME\*\*\*\*\*',’\*’) FROM DUAL;

1. ***Trim()*** – It removes both leading and trailing spaces in a col/value

***Syntax: trim(col/val [,char])***

EX. SELECT TRIM(' WELCOME ') FROM DUAL;

SELECT TRIM(‘\*\*\*\*\*WELCOME\*\*\*\*\*',’\*’) FROM DUAL;

***Select Ltrim(' Welcome '),Rtrim(' Welcome '),Trim(' Welcome ') From Dual;***

**DATE Functions**

The functions which can perform the operations on the system date and time values are called ‘Date’ functions. The default system date format is dd-mon-yy ie 15-OCT-22

dd – two digits in a day

mm- two digits in a month

mon- 3 letters in a month

month – name of a month

yy – two digits in a year

yyyy – 4 digits in a year

1. Sysdate – It returns the current system date.

Syntax: sysdate

Ex. SELECT SYSDATE FROM DUAL;

1. Last\_day() – It returns the last day of a given month in a date.

Syntax: last\_day(date)

Ex. SELECT LAST\_DAY(SYSDATE) FROM DUAL;

SELECT HIREDATE, LAST\_DAY(HIREDATE) FROM EMP;

1. Add\_months() - It returns the date after adding ‘n’ number of months to a date.

Syntax: add\_months(date,n)

Ex. select add\_months(sysdate,20) from dual;

Select add\_months(’23-may-20’,28) from dual;

1. Months\_between() – It returns the no.of months gap between two dates.

Syntax: months\_between(date1,date2)

Ex. SELECT MONTHS\_BETWEEN(SYSDATE,'25-oct-19') FROM DUAL;

1. Next\_day() – It returns the date of next weekday comes after a given date

Syntax: NEXT\_DAY(date, weekday)

**CONVERSION FUNCTIONS**

The functions that are used to convert one datatype value into another type are called ‘Conversion’ Functions. They are

1. TO\_CHAR() – It converts a given number/date into character format.

Syntax: TO\_CHAR(number/date)

SELECT TO\_CHAR(100) FROM DUAL;

SELECT TO\_CHAR(SYSDATE,'DD-MM-YYYY') FROM DUAL;

SELECT TO\_CHAR(SYSDATE,'DD-MONTH-YYYY') FROM DUAL;

1. TO\_NUMBER() – It converts a character format number to number format.

Syntax: TO\_NUMBER(string)

SELECT TO\_NUMBER('100')+TO\_NUMBER('200') FROM DUAL;

1. TO\_DATE() – It converts a given date value into system date format.

Ie. 17-OCT-22

Syntax: TO\_DATE(datevalue , dateformat)

SELECT TO\_DATE('12,JUN,12','DD,MON,YY') FROM DUAL;

SELECT TO\_DATE('DECEMBER-17-2018', 'MONTH-DD-YYYY') FROM DUAL;

SELECT TO\_DATE('23,9,76','DD,MM,YY') FROM DUAL;

HERE, dd - 2 digits in a date

mm - 2 digits in a month

mon – 3 letters in a month

month – name of the month

yy - 2 digits in a year

yyyy - 4 digits in a year

Ww:- week of year

W:- week of month

Ddd:-day of the year

Dd:-day of the month

D:-day of the week

**General functions:**

The functions which can perform the operations on any type of data ie. Character data or numeric data or date data are called ‘General Functions’. They are

1. NVL()
2. GREATEST()
3. LEAST()
4. NVL() – It is used to assign a default value for null values in a table column.

Syn: NVL(colname,value)

SELECT ENAME,JOB,SAL,NVL(COMM,0) FROM EMP;

SELECT ENAME,NVL(JOB,'PENDING'),SAL,DEPTNO FROM EMP;

1. GREATEST() –It returns the largest value in a given list of values.

Syn: GREATEST( value1, value2, ….)

SELECT GREATEST(12,33,56,23,8,45,7) FROM DUAL;

SELECT GREATEST('ANIL','KUMAR','AKASH','RAHUL','ASHOK') FROM DUAL;

SELECT GREATEST('12-OCT-98','23-DEC-12',SYSDATE) FROM DUAL;

1. LEAST() – It returns the smallest value in a given list of values.

Syn: LEAST(value1,value2, ..)

SELECT LEAST(12,33,56,23,8,45,7) FROM DUAL;

SELECT LEAST('ANIL','KUMAR','AKASH','RAHUL','ASHOK') FROM DUAL;

SELECT LEAST('12-OCT-98','23-DEC-12',SYSDATE) FROM DUAL;

**GROUP FUNCTIONS: (AGGREGATE)**

The functions which can perform the operations on group of values in a table column are called ‘Group functions’. They can give a single result for a group of values. They are

1. SUM()
2. AVG()
3. MAX()
4. MIN()
5. COUNT()
6. Sum() – it returns the sum of values in a table column.

Syntax: sum(col)

1. Avg() – it returns the average value of a given column values.

Syntax: avg(col)

1. Max() – it returns the maximum value in a given column values.

Syntax: max(col)

1. Min() – it returns the minimum value in a given column values.

Syntax: min(col)

1. Count() – it returns the no.of rows in a given column. It counts not null rows in a column.

Syntax: count(col/\*)

Ex.

SELECT SUM(SAL), AVG(SAL), MAX(SAL), MIN(SAL), COUNT(SAL) FROM EMP;

SELECT COUNT(COMM),COUNT(\*) FROM EMP;

**GROUP BY clause**

It is used to group the rows in a table based on a column. It will be applied with the group functions. With this we can find sum of salaries, max of salary etc either job wise or department wise. It is also applied with SELECT statement.

Syntax:

SELECT groupcolname , groupfunction(s) FROM tablename

GROUP BY colname;

Ex.

1. SELECT JOB, COUNT(JOB),SUM(SAL),AVG(SAL),MAX(SAL),MIN(SAL) FROM EMP GROUP BY JOB;
2. SELECT DEPTNO, COUNT(JOB),SUM(SAL),AVG(SAL),MAX(SAL), MIN(SAL) FROM EMP GROUP BY DEPTNO;
3. SELECT MGR, COUNT(JOB),SUM(SAL),AVG(SAL),MAX(SAL), MIN(SAL) FROM EMP GROUP BY MGR;

**HAVING clause**

It is used to select the specified rows retrieved by the group by clause. Ie to filter the rows. Where clause is applied only on select statement to filter the rows in a table.

SYNTAX**:**

SELECT colname, groupfunction(s) FROM tablename

GROUP BY colname HAVING group\_condition;

Ex.

1. SELECT JOB, COUNT(JOB),SUM(SAL),AVG(SAL),MAX(SAL), MIN(SAL) FROM EMP GROUP BY JOB HAVING COUNT(JOB)>=4;
2. SELECT JOB, COUNT(JOB),SUM(SAL),AVG(SAL),MAX(SAL), MIN(SAL) FROM EMP GROUP BY JOB HAVING MAX(SAL)>2000;
3. SELECT DEPTNO,COUNT(DEPTNO),SUM(SAL),MAX(SAL),MIN(SAL) FROM EMP GROUP BY DEPTNO HAVING COUNT(DEPTNO) >=4;

***RELATIONAL SET OPERATORS***

The set operators are derived from the mathematical concept called relations. They are used to combine the results of more than one query statements into a single result. In this, all query statements must have same no.of cols and same type of cols.

Syntax:

SELECT statement1

Setoperator

SELECT statement2;

The relational set operators are:

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS
5. ***UNION***

It combine the results of more than one query statements into a single result without duplicate rows.

Ex. SELECT \*FROM EMP WHERE DEPTNO=30

UNION

SELECT \*FROM EMP WHERE JOB='MANAGER';

1. ***UNION ALL***

It combine the results of more than one query statements into a single result with duplicate rows also.

EX. SELECT \*FROM EMP WHERE DEPTNO=30

UNION ALL

SELECT \*FROM EMP WHERE JOB='MANAGER';

1. ***INTERSECT***

It retrieve the common rows in both query results.

Ex. SELECT \*FROM EMP WHERE DEPTNO=30

INTERSECT

SELECT \*FROM EMP WHERE JOB='MANAGER';

1. ***MINUS***

It can retrieve the rows that are available in first query result and not in second query result.

Ex. SELECT \*FROM EMP WHERE deptno=30

MINUS

SELECT \*FROM EMP WHERE JOB='MANAGER';

**SUB QUERIES**

It means one query statement may contain another query statement. It is also known as nested queries. It contain at least two query statements – main/outer query and sub/inner query. The sub query statements are enclosed in parenthesis ().

In sub queries, the result of a sub query will be passed to the main query, then only the main query will be executed.

In this, the sub query will be executed first, later the main query will be executed by result of sub query ie the sub query results can be send to the main query for its execution.

Syntax:

Main/outer sub/inner

SELECT statement1 (***SELECT statement2***);

Types of sub queries:

The sub queries are classified into 2 types. They are:

1. Single Row subqueries
2. Multi Row subqueries
3. ***Single row subquery:***- If a sub query returns only one row then is called ‘single row sub query’.

Ex.

1. SELECT \*FROM EMP WHERE DEPTNO=(SELECT DEPTNO FROM EMP WHERE ENAME=’SCOTT');
2. SELECT \*FROM EMP WHERE JOB=(SELECT SAL FROM EMP WHERE ENAME='BLAKE');
3. SELECT \*FROM EMP WHERE SAL<(SELECT DEPTNO FROM EMP WHERE EMPNO=7844);
4. ***Multi row sub query:-*** If a subquery returns more than one row is called multi row sub query. For executing multi row sub queries the following two operators are used.
5. ANY - compares any one value in the result ie similar to **or** operator
6. ALL - compares all values returned by sub query ie similar to **and** operator

Ex.

1. SELECT \*FROM EMP WHERE SAL>ANY(SELECT SAL FROM EMP WHERE DEPTNO=10);
2. SELECT \*FROM EMP WHERE JOB=ANY(SELECT JOB FROM EMP WHERE DEPTNO=20);
3. SELECT \*FROM EMP WHERE SAL>ALL(SELECT SAL FROM EMP WHERE JOB=’CLERK’);

**JOINS**

It is one of the most useful feature in SQL. It is mainly used to combine the columns in multiple tables into a single result. For this, the tables must have a relationship ie primary key and foreign key relationship. To combine multiple table columns data the join mechanism uses a join condition.

Syntax:

SELECT col1, col2, col3, ….

FROM tablename1 [aliasname], tablename2 [aliasname]

WHERE join\_condition;

Where col1, col2, … - represent the columns in both tables

Join\_condition – representS the condition used to join

***Types of Joins:***

The join mechanism is classified into 4 types. They are:

1. Self join
2. Equi join
3. Non-Equi join
4. Outer join
5. ***Self join:***

If a table is joined to itself is called self join. It is also known recursive join.

Ex. ***To display the employees with their managers details***

SELECT A.EMPNO, A.ENAME, A.JOB, A.SAL, A.MGR, B.ENAME, B.JOB, B.SAL

FROM EMP A, EMP B

WHERE A.MGR=B.EMPNO;

1. ***Equi join:***

It is used to combine the columns in more than one tables into single result based on a join condition. If the join condition uses = operator then is called ‘Equi join’. Here, the tables must have relationship between them, that is, common column is defined in both tables.

Ex. SELECT E.EMPNO, E.ENAME, E.JOB, E.SAL, D.DEPTNO, D.DNAME, D.LOC

FROM EMP E, DEPT D WHERE E.DEPTNO = D.DEPTNO;

1. ***Non-Equi Join:***

It is also similar to equi join, that is, to combine the columns in multiple tables into a single result. If the join condition uses the operator other than = then is called ‘Non-equi’ join.

Ex.

SQL> SELECT EMPNO, ENAME, JOB, SAL, DEPTNO, GRADE

FROM EMP, SALGRADE

WHERE SAL BETWEEN LOSAL AND HISAL;

SQL> SELECT E.EMPNO, E.ENAME, E.JOB, E.SAL, D.DEPTNO, D.DNAME, D.LOC, GRADE FROM EMP E, DEPT D, SALGRADE

WHERE E.DEPTNO = D.DEPTNO AND SAL BETWEEN LOSAL AND HISAL;

1. ***Outer Join:***

The equi-join combine the columns in multiple tables when the join condition is satisfied. Sometimes, the rows in one table does not have the matching rows in the related table then those row not retrieved by the equi-join. To retrieve such rows also outer-join is used.

The outer join is identified by the symbol +. It is applied in deficient table side of join condition. They are in two types.

1. Left outer join
2. Right outer join
3. Left Outer Join: If a deficient table is in right side of join condition then is called ‘left outer join’.

SQL> SELECT D.DEPTNO, D.DNAME, D.LOC, E.EMPNO, E.ENAME, E.JOB

FROM EMP E, DEPT D

WHERE D.DEPTNO = E.DEPTNO(+);

1. Right Outer join: if a deficient table is in left side of the join condition then is called ‘right outer join’.

SQL> SELECT E.EMPNO, E.ENAME, E.JOB, E.SAL, D.DEPTNO, D.DNAME, D.LOC

FROM EMP E, DEPT D

WHERE E.DEPTNO(+)=D.DEPTNO;

**VIEWS**

It is a logical representation of sub-set of data from one (or) more than one table in a convenient format. A view can store the results of a query statement. It can automatically updated when the source of a view changed. A view can be created for a table, synonym or another view.

Why we need to create the views:-

* View produce the rows & columns by creating views on particular columns of rows.
* View protect the data from DML operations by creating the view as read only
* A user can generate reports without creating complex queries again & again without having complex knowledge in joins.
* View doesn’t occupies any physical storage in database. So that by creating multiple views on the table doesn’t waste the memory in database.
* View integrate the validity of insert & update operations by checking the effects on data.

Syntax:

CREATE VIEW viewname AS SELECT statement;

**Types of Views**

1. Simple view
2. Read only view
3. View with check option
4. Complex view

***Simple view:-***

If a view is created based on single table then is called simple view.

Ex.

* 1. SQL> Create view v1 as select \*from emp;
  2. SQL> Create view v2 as select empno,ename,job,sal,deptno from emp;
  3. SQL> Create view v3 as select \*from emp where deptno=30;
  4. SQL> Create view v4 as select \* from emp where job=’SALESMAN’ or job=’CLERK’;
  5. Create view v5 as select empno, ename, job, deptno, trunc(sal/30) dsal, sal msa, sal\*12 ansal, trunc (months\_between (sysdate,hiredate)/12) exp from emp order by job;

***Read only view :-***

Generally all views allow the DML operations. But you don’t want to allow DML operations then read only views can be used.

SQL>create view rview as select \* from emp where job=’MANAGER’ with read only;

***View with check option :-***

If you create a view with check option then it allows the data entry only, if the given where clause condition is satisfied.

SQL>Create view cview as select \* from emp where job in (‘CLERK’, ’SALESMAN’) and sal between 1000 and 2000 with check option;

***Complex View :-***

If a view is created from more than one table is called complex view.

SQL> Create view compview as select e.empno,e.ename, e.job, e.sal, d.deptno,

d.dname, d.loc from emp e, dept d where e.deptno=d.deptno;

SQL> Create view jview as select empno, ename , job, sal , emp , dept.deptno , dname , loc , grade from emp , dept , salgrade where grade in (2,3,4)

And (loc like ‘D%’ or loc like ‘C%’)

And Emp.deptno=dept.deptno

And Sal between losal and hisal order by dname, grade;

**To Insert a row**

INSERT INTO V1 VALUES (1001,’RAHUL’,’CLERK’,7698,’12-OCT-85’,1450,NULL,10);

**To Display The Data In A View**

SELECT \*FROM V1;

**To Delete A View:**

Syntax: drop view viewname;

Ex. DROP VIEW V1;

**To display all views in a database:**

Ex. SELECT \*FROM USER\_VIEWS;

**SYNONYMS**

A synonym is also one of the database object, which is defined an alternative for a database object. It has the same meaning, that is, it can store the data of a database object.

A synonym is created for a table, a view or another synonym.

It is also be created with the DDL command CREATE.

Syntax:

CREATE SYNONYM synonymname FOR dbobject;

EX.

* 1. For A Table

SQL> CREATE SYNONYM STUDSYN FOR STUDENT;

* 1. For A View

SQL> CREATE SYNONYM SYn1 For Svw;

* 1. For Another Synonym

SQL> CREATE SYNONYM SYN2 FOR STUDSYN;

***To display a synonym***

SQL> SELECT \*FROM STUDSYN;

***To display all synonyms:***

SQL> SELECT \*FROM USER\_SYNONYMS;

***To delete a synonym:***

Syntax: drop synonym synonymname;

Ex. drop synonym syn1;

**SEQUENCES**

A sequence is also one of the database object. It is store sequential numbers into a db object. It can store the sequential values between a given range.

It is mainly used to send the sequential values to a primary key in a table because a primary key cannot allow duplicate and null values.

A sequence is also generates sequential numbers with gap 1 or with user specified gap.

Syntax:

CREATE SEQUENCE sequencename

START WITH integer

[INCREMENT BY integer]

MAXVALUE integer

[MINVALUE integer]

[CYCLE/NOCYCLE]

[CACHE/NOCACHE];

WHERE START WITH – represents the starting no.of the sequence

INCREMENT BY – represents the gap between two no.s

MAXVALUE - represents maximum value of a sequence

MINVALUE – represents minimum value of a sequence

CYCLE - used to repeat when a sequence reaches max.value

CACHE - to allocate memory for a sequence when cycle is applied

**To access (retrieve) the values in a sequence**

1. **NEXTVAL** – returns the next value in a sequence
2. **CURRVA**L – returns the current value in a sequence

Syn: sequencename.NEXTVAL

Ex. Sql> Create sequence seq1

start with 1

maxvalue 10;

sql> select seq1.nextval from dual;

nextval

----------

1

Sql> /

nextval

----------

2

SQL> CREATE SEQUENCE E1

START WITH 1001

MAXVALUE 1100;

***To send sequence values to primary key of a table***

SQL> INSERT INTO STUDENT VALUES(E1.NEXTVAL,'&NAME',&M1,&M2,&M3);

***To display all sequences***

SQL> SELECT \*FROM USER\_SEQUENCES;

***To delete a sequence***

SQL> DELETE SEQUENCE SEQ1;

SQL> CREATE SEQUENCE SEQ2

START WITH 100

INCREMENT BY 2

MAXVALUE 150;

***TO CREATE A USER IN ORACLE:-***

In Oracle database the users are also be created using the DDL command CREATE. It is also be operated by DBA (database administrator).

Syntax:

CREATE USER username IDENTIFIED BY password;

Ex. create user bdps identified by oracle;

Create user sachin identified by oracle;

***To connect the user:***

The DBA can grant permissions to the user for connecting to a database. For this apply the following command.

Ex. SQL> grant connect, resource to bdps;

Sql> grant connect, resource to sachin;

SQL> connect bdps

password : oracle

Note: Create a table in bdps user and insert some rows

SQL> create table emp1 (eno number(4) primary key, ename varchar(10) not null,

job varchar(15), sal number(5));

SQL> insert into emp1 values(&eno,'&name','&job',&sal);

SQL> select \*from emp1;

ENO ENAME JOB SAL

---------- ---------- --------------- ----------

1001 kumar clerk 5000

1002 mahesh manager 16000

1003 ram analyst 6500

1004 ramesh clerk 5400

1005 anil salesman 7500

**DCL COMMANDS**

DCL stands for Data Control Language. The DCL commands are used to control the data between the database users ie. To access the data of one user by another user. For this, the privileges (permissions) can be given by the database administrator. Similarly, the given permissions are also taken back by the database administrator. The DCL commands are:

1. GRANT
2. REVOKE
3. GRANT – This command is used to grant privileges on a specified database object to another user.

Syntax: GRANT privilegename(s)/ALL

ON tablename

TO username;

Where privilege name specifies – SELECT, UPDATE, DELETE, INSERT

ALL - specifies all privileges

1. REVOKE – It is used to revoke or taken back the given privileges from a user.

Syntax: REVOKE privilegename(s)/ALL

ON tablename

FROM username;

Note: The DCL commands are operated by the database administrator only.

**To connect DBA**

SQL> connect system/manager

***To grant permission*** to the user sachin for selecting emp1 table data of bdps user.

SQL> grant select

On bdps.emp1

To sachin;

To test:

SQL> connect sachin/oracle

SQL> select \*from bdps.emp1;

***To cancel the permission***

SQL> Revoke select

on bdps.emp1

from sachin;

**TCL COMMANDS**

TCL – stands for Transaction control language.

The TCL commands are used to control the transactions in a db. A transaction may consist of one or more sql operations. Each operation can be performed by one sql statement. The TCL commands are:

1. **COMMIT** - To save the changes made in a database

Syntax: ***Commit;***

1. **ROLLBACK** – To cancel the changes made in a database (undo) ie pending transactions in the database. (uncommitted transactions)

Syntax: ***Rollback [to savepoint***];

1. **SAVEPOINT** – used to group the transactions

***SAVEPOINT savepointname;***

1. Insert any 2 or 3 rows into a table (emps)
2. Sql> select \*from emps;
3. Sql> rollback;

It cancels the inserted rows

1. Insert 2 rows into a table / delete rows in a table
2. Sql>commit;

Now, the changes made permanently in the database. We can’t rollback.

Note: savepoint is used to group transactions

Insert rows into a table

Sql> savepoint s1;

Delete rows in a table

Sql> savepoint s2;

Update rows in a table

Insert rows into a table

Sql> rollback to s2;

It cancels the transactions upto s2.

**PL/SQL**

**PL – stands for Procedural Language**

It is an extension of SQL. In SQL, we have work with only one sql command at a time and it does not support the procedural capabilities like looping and branching. To overcome these problems PL/SQL was introduced.

## Advantages of PL/SQL

PL/SQL has the following advantages:

* PL/SQL is the standard database language.
* PL/SQL supports both static and dynamic SQL.
* PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
* PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
* PL/SQL provides high security level.

***PL/SQL Block (Program):***

In Pl/SQL we have work with multiple commands. This group of commands are known as PL/SQL block (program). A PL/SQL program contains 3 parts. They are:

1. Declaration part
2. Executable part
3. Exception Handling part
4. ***Declaration part***

It is used to declare the variables, functions, exceptions, cursors etc. it is identified by the keyword declare.

Syntax:

Declare

Declaration of variables, functions, procedures, exceptions, cursors etc.

1. ***Executable par***t

It contain the executable statements like input/output statements, calculations, expressions etc. It can start with the keyword ‘begin’ and end by the keyword ‘end’.

Syntax:

Begin

………

Executable statements;

End;

1. ***Exception Handling Part*** – The runtime errors are known as exceptions. It is used to handle the exceptions raised in the executable part. It is also a part of an executable part and is an optional. It is identified by the keyword ‘Exception’.

Syntax:

Exception

……………..

Exception handling statements;

**Structure of a PL/SQL block (program)**

Syntax:

Declare

Declarations

Begin

…………

Executable statements;

[Exception

………………

Exception handling statements;]

End;

/

**Variable declaration:**

Like other languages, in PL/SQL also the variables are used. A variable is mainly used to store data in memory. It required memory space to store a value. For this, variable declaration is used.

Syntax: variablename datatype(size) [ := value];

Ex.

1. n number;
2. n number:=100;
3. n number(4):=&n;
4. name varchar(20):=’&name’;
5. salary number(7,2);
6. dob date;
7. dob date:=’&date’; (dd-mon-yy ie. 16-aug-22)

**Note**: In PL/SQL ***:= is called assignment operator***

**To display output:**

To display output in PL/SQL the following 2 functions are used. They are:

1. **dbms\_output.put ()** - It is used to display or print a message/value on screen sequentially.

Syn: dbms\_output.put (message/value);

Ex. dbms\_output.put(‘welcome’);

dbms\_output.put (n);

1. **dbms\_output.put\_line ()** – it is also used to display or print a message/value on screen in a single line.

Syn: dbms\_output.put\_line (message/value);

Ex. dbms\_output.put\_line (‘welcome to pl/sql’);

dbms\_output.put\_line (n);

***Concatenation operator (||)*** – It concatenates/adds one string to another or a string and a variable.

Ex. dbms\_output.put\_line (‘Number= ‘||n);

**Note:**

1. For the above functions, we should apply the following command once at

SQL***> set serveroutput on***

1. **Edit command** – used to write pl/sql programs.

Syntax: edit filename

Ex. SQL> edit abc

1. **@ command** – used to run a pl/sql program.

Syntax: @filename

Ex. SQL> @abc

declare

n number;

begin

n:=&n;

dbms\_output.put\_line(n);

dbms\_output.put\_line('N= '||n);

end;

/

declare

a number:=&a;

b number:=&b;

c number;

begin

dbms\_output.put\_line('A= '||a||' B= '||b);

c:=a;

a:=b;

b:=c;

dbms\_output.put\_line('After swap A= '||a||' B = '||b);

end;

/

**To Retrieve SQL data (Table) into PL/SQL**

For this, Oracle provides SELECT … INTO statement.

Syntax:

SELECT col1,col2, … INTO var1,var2, ….

FROM tablename where condition;

Ex.

declare

eno number:=&eno;

ena varchar(10);

job varchar(15);

sal number;

dno number;

begin

select ename, job, sal, deptno into ena,job,sal, dno from emp

where empno=eno;

dbms\_output.put\_line('---Employee details----');

dbms\_output.put\_line('-----------------------');

dbms\_output.put\_line('Number = '||eno);

dbms\_output.put\_line('Name = '||ena);

dbms\_output.put\_line('Job = '||job);

dbms\_output.put\_line('Salary = '||sal);

dbms\_output.put\_line('Deptno = '||dno);

end;

/

**TO INSERT DATA INTO A TABLE FROM PL/SQL**

declare

eno number:=&eno;

ena varchar(10):='&name';

jb varchar(10):='&job';

s number(5):=&sal;

begin

insert into emps values(eno,ena,jb,s);

commit;

dbms\_output.put\_line('---Row inserted successfully----');

end;

/

@filename or run filename

Ex. SQL> @a1

**%type attribute**

It is used to declare a variable with a specified table column type.

Syntax: variable tablename.colname%type;

Ex. eno emp.empno%type;

name emp.ename%type;

declare

num number:=&empno;

ena emps.ename%type;

jb emps.job%type;

sa emps.sal%type;

dno emps.deptno%type;

begin

select ename, job, sal, deptno into ena, jb, sa, dno from

emps where empno=num;

dbms\_output.put\_line('-----EMPLOYEE DETAILS-------');

dbms\_output.put\_line('Number : '||num);

dbms\_output.put\_line('Name : '||ena);

dbms\_output.put\_line('Job : '||jb);

dbms\_output.put\_line('Salary : ' ||sa);

dbms\_output.put\_line('Deptno : '||dno);

end;

/

**%rowtype**

It is used to declare a record variable which refers all columns in a table.

Syntax: recordvariable tablename%rowtype;

Ex. rec emp%rowtype;

To access a column:

Syn: recordvariable.colname

Ex. rec.empno

rec.ename

rec.job

Ex. To display the details of a specified employee by accepting employee number.

declare

rec emps%rowtype;

eno emps.empno%type:=&empno;

begin

select \* into rec from emps where empno=eno;

dbms\_output.put\_line('----------EMPLOYEE DETAILS--------');

dbms\_output.put\_line('Number : '||rec.empno);

dbms\_output.put\_line('Name : '||rec.ename);

dbms\_output.put\_line('Job : '||rec.job);

dbms\_output.put\_line('Salary : '||rec.sal);

end;

/

**: Control Statements :**

The control statements are used to control the execution of a program. They are divided into various types. They are:

1. Sequential statements
2. Conditional / selection statements
3. Repetition / Loop statements
4. ***Sequential statements:***

If the statements in a program are executed one after another sequentially are called sequential statements. They are executed in the same order in which they are written.

Ex.

declare

n number:=&n;

c number;

begin

-- c:=n\*n\*n;

-- dbms\_output.put\_line('cube value of '||n||' : '||c);

c:=power(n,3);

dbms\_output.put\_line('cube value of '||n||' : '||c);

end;

/

Note: In PL/SQL, the comment lines are represented by --.

1. **Conditional/selection statements:-**

If the statements in a program are executed based on a condition are called conditional statements. They are:

1. if statement
2. case .. end case statement
3. ***if statement*** – It can execute a statement or set of statements based on a condition. It is applied in various types.
   * + 1. If … then ... end if
       2. If … then … else … end if
       3. If … then … elsif … end if
4. If … then … end if – It can execute the statements when the test condition is true only, otherwise the statements are skipped.

Syntax: if condition then

Statement(s)

End if;

To find the biggest of two numbers

Ex. declare

a number:=&a;

b number:=&b;

big number;

begin

big:=a;

if big<b then

big:=b;

end if;

dbms\_output.put\_line('Big = '||big);

end;

/

To find a given no. is odd/even

declare

n number:=&n;

begin

if mod(n,2)=0 then

dbms\_output.put\_line(n||' is Even');

end if;

if mod(n,2)<>0 then

dbms\_output.put\_line(n||' is Odd');

end if;

end;

/

1. If … then … else … end if :

It can execute only one block of statements at a time from two alternative blocks ie if the condition is true then one block will be executed otherwise another block will be executed.

syn: if condition then

statement(s)

else

statement(s)

end if;

To find biggest of 2 numbers

declare

a number:=&a;

b number:=&b;

k number;

begin

if a>b then

k:=a;

else

k:=b;

end if;

dbms\_output.put\_line('Big = '||k);

end;

/

To retrieve the details of employ and find bonus and total salary

declare

eno emps.empno%type:=&eno;

r emps%rowtype;

bon emps.sal%type;

tsal emps.sal%type;

begin

select \* into r from emps where empno=eno;

if r.sal>=10000 then

bon:=r.sal\*.05;

else

bon:=r.sal\*.04;

end if;

tsal:=r.sal+bon;

dbms\_output.put\_line(r.empno||' '||r.ename||' '||r.job||' '||r.sal);

dbms\_output.put\_line('Bonus = '||bon);

dbms\_output.put\_line('Total = '||tsal);

end;

/

1. elsif … ladder

It is used to evaluate ‘n’ no.of conditions one by one. From ‘n’ conditions, if any one condition is satisfied then it will execute that corresponding block of statements.

syn: if condition-1 then

statement(s)

elsif condition-2 then

statement(s)

----

----

elsif condition-n then

Statement(s)

end if;

To find a given no is positive or negative or zero

declare

n number:=&n;

begin

if n>0 then

dbms\_output.put\_line(n||' is positve');

elsif n<0 then

dbms\_output.put\_line(n||' is negative');

elsif n=0 then

dbms\_output.put\_line('No. is zero');

end if;

end;

/

To compare any 2 numbers

declare

a number:=&a;

b number:=&b;

begin

if a>b then

dbms\_output.put\_line(a||' is big');

elsif b>a then

dbms\_output.put\_line(b||' is big');

else

dbms\_output.put\_line('Equal');

end if;

end;

/

1. Nested if

It means one if statement may contain another if statement within its block. Similarly one else may also contain another if statement. It is used when one condition is depending on another condition. Here, the if statement may be if / if …else / elsif … ladder.

Syntax:

If cond-1 then

If cond-2 then

Statements-1;

Else

Statements-2;

End if;

Else

If cond-3 then

Statements-3;

Else

Statemebts-4;

End if;

End if;

declare

a number:=&a;

b number:=&b;

c number:=&c;

big number;

begin

if a>b then

if a>c then

big:=a;

else

big:=c;

end if;

else

if b>c then

big:=b;

else

big:=c;

end if;

end if;

dbms\_output.put\_line('A= '||a||' B = '||b||' C = '||c);

dbms\_output.put\_line('Big = '||big);

end;

/

1. Case … end case:

It is similar to switch statement in ‘C’. It has multiple entries, we can select one entry from that.

Syntax:

Case expression

when expression1 then

Statements;

when expression2 then

Statements;

------

when expression-n then

Statements;

Else

Statements;

End Case

declare

a number:=&a;

b number:=&b;

c number;

op varchar(1):='&operator';

begin

dbms\_output.put\_line('a = '||a||' b= '||b);

case op

when '+' then

c:=a+b;

when '-' then

c:=a-b;

when '\*' then

c:=a\*b;

when '/' then

c:=a/b;

when '%' then

c:=mod(a,b);

end case;

dbms\_output.put\_line('A '||op||' B = '||c);

end;

/

1. **Iteration/Looping statements**

The repeated execution of statements is called a loop. The statements used to terminate or stop a loop are called looping statements. Like other languages, oracle also support 3 types of looping statements.

They are:

1. Basic loop
2. while loop
3. for loop

In PL/SQL every loop must be terminated with end loop.

**Basic loop:**

It is used to execute one or more statements repeatedly. To terminate a loop it uses exit statement in the body of loop.

Syntax:

Loop

----

Statements

Exit when condition;

End loop;

Ex.1) To print 1 to n numbers

declare

n number:=&n;

i number:=1;

begin

loop

dbms\_output.put\_line(i);

i:=i+1;

exit when i>n;

end loop;

end;

/

1. to find the factors of a number

declare

n number:=&n;

i number:=1;

begin

dbms\_output.put\_line(' -----Factors ----');

loop

if mod(n,i)=0 then

dbms\_output.put\_line(i);

end if;

i:=i+1;

exit when i>n;

end loop;

end;

/

**While loop**:

It is used to execute a statement or set of statements repeatedly.

It can be called as entry-controlled loop or pre-test loop.

Syntax:

While condition

Loop

----

Statement(s)

End loop;

Programs:

1. To print the multiplication table of a number

declare

n number:=&n;

i number:=1;

begin

while i<=10

loop

dbms\_output.put\_line(n||'\*'||i||'='||n\*i);

i:=i+1;

end loop;

end;

/

1. To find the reverse of a number

declare

n number:=&n;

d number;

rev number:=0;

begin

while n>0

loop

d:=mod(n,10);

rev:=(rev\*10)+d;

n:=trunc(n/10);

end loop;

dbms\_output.put\_line('Reverse = '||rev);

end;

/

**For loop:**

It is also used to execute one or more statements repeatedly. It is very easy to apply than while and do..while.

Syntax:

for counter in [reverse] start..end

loop

---

statements

end loop;

where

start – represents initial value of the counter

end - represents the end value of the counter

reverse – indicates the counter in reverse ie. End to start

declare

i number;

n number:=&n;

c number:=0;

begin

for i in 1..n

loop

if mod(n,i)=0 then

c:=c+1;

end if;

end loop;

if c=2 then

dbms\_output.put\_line(n||' is prime');

else

dbms\_output.put\_line(n||' is not prime');

end if;

end;

/

***Nested Loops:***

It means one loop statement may contain another loop statement within the body of a loop. It is mainly used for executing the repeated statements repeatedly. It required at least loops – an inner loop and an outer loop. In this, for each iteration of an outer loop the inner loop was completely executed.

Syntax: outer loop

While condition inner loop

Loop

For i in 1..10

Loop

Statements

End loop;

End loop;

To print all primes between 1 and n

declare

n number:=&n;

i number;

c number;

x number:=1;

begin

dbms\_output.put\_line('Primes between 1 and '||n||' : ');

while x<=n

loop

c:=0;

for i in 1..x

loop

if mod(x,i)=0 then

c:=c+1;

end if;

end loop;

if c=2 then

dbms\_output.put(x||' ');

end if;

x:=x+1;

end loop;

dbms\_output.put\_line(' ');

end;

/

**Exception Handling**

When we are working with programs in PL/SQL then there may be a chance to errors by typing mistakes or values entered. These errors may causes to abnormal termination of a program. The errors are of two types – compile time errors and runtime errors.

The runtime errors are known as exceptions. To handle these exceptions PL/SQL provides the feature called ‘Exception Handling’. It is one of the part in executable part. It is identified by the keyword **exception**.

Syntax:

Begin

------

Statements;

**Exception**

When exception\_name then

Statements;

End;

**Types of exceptions**:

The exceptions are classified into 2 types. They are:

1. predefined exceptions
2. user defined exceptions
3. ***Pre-Defined Exceptions*** – The exceptions that have been already defined in PL/SQL are called pre-defined exceptions. These exceptions are raised in a program and are handled by the user. For handling these exceptions the following exception handlers are provided. They are:

ex. NO\_DATA\_FOUND – to handle when data (empno) not found

ZERO\_DIVIDE – to handle divisible by 0 applied

DUP\_VAL\_ON\_INDEX – to handle duplicate value given for primary key

TOO\_MANY\_ROWS etc – to handle multiple rows retrieved for single row

They are applied in exception handling part as follows:

Syntax:

Exception

When ***exceptionhandler*** Then

Statement;

To handle Zero\_Divide error

declare

a number:=&a;

b number:=&b;

c number;

begin

c:=a/b;

dbms\_output.put\_line('a = '||a||' b = '||b);

dbms\_output.put\_line('a/b = '||c);

exception

when zero\_divide then

dbms\_output.put\_line('Divisible by zero attempted');

end;

/

declare

j emp.job%type:='&job';

e emp%rowtype;

begin

select \* into e from emp where job=j;

exception

when too\_many\_rows then

dbms\_output.put\_line('---more rows retrieved----');

when no\_data\_found then

dbms\_output.put\_line('----No such job----');

end;

/

***User defined exceptions***

The exceptions that are created, raised and handled by the user are called ‘user defined exceptions’. Whereas pre-defined exceptions are automatically raised and handled by the user.

1. *To create an exception:*

The user defined exception can be created by the user. It can be specified in the declaration part of pl/sql program.

Syntax: exception\_name exception;

Ex. myexp exception;

1. *raise statement*:

It is used to raise a user defined exception based on a condition.

Syntax: raise exception\_name;

Ex. raise myexp;

1. Exception handling part:

The exception handling part is used to handle either a user defined or a pre-defined exception.

To raise and handle an exception when marks given >100

declare

m number:=&maths;

p number:=&physics;

c number:=&chem;

t number;

av number;

marks\_outof\_limit exception;

begin

if m>100 or p>100 or c>100 then

raise marks\_outof\_limit;

else

t:=m+p+c;

av:=t/3;

dbms\_output.put\_line('Total : '||t);

dbms\_output.put\_line('Average : '||av);

end if;

exception

when marks\_outof\_limit then

dbms\_output.put\_line('-----Marks exceeds max marks....');

end;

/

**FUNCTIONS AND PROCEDURES**

Like other languages, PL/SQL also support functions and procedures. They are useful to simplify a program ie to divide a program into smaller parts. Functions and procedures are stored in a database. They can perform a specific task.

**PROCEDURES**

* A procedure is a sub program which can perform a specific task.
* A procedure does not return a value.
* A procedure can be defined with/without arguments.
* A procedure is also be create with the CREATE.
* Every procedure has 2 parts.
  + - * 1. Procedure header
        2. Procedure body

**Procedure Header**

The statement before the keyword is, is called procedure header.

**Procedure Body**

The statements after the keyword is, is called body.

To create a Procedure

Syntax:

CREATE OR REPLACE PROCEDURE procedurename([argslist])

IS/AS

Declarations;

BEGIN

Statements;

END;

/

To call a procedure

To execute the body of a procedure that procedure must be calling.

Syntax: procedurename([args]);

create or replace procedure show

IS

begin

dbms\_output.put\_line('Welcome to Procedures');

end;

/

begin

show;

end;

/

Note: a procedure can be called in the same program or another program

create or replace procedure display(m number)

as

begin

dbms\_output.put\_line(m);

end;

/

begin

display(100);

end;

/

create or replace procedure delrec(eno number)

as

begin

delete from emps where empno=eno;

commit;

dbms\_output.put\_line('--- Employee deleted succuessfully---');

exception

when no\_data\_found then

dbms\_output.put\_line('---Employee not found-----');

end;

/

declare

num emps.empno%type:=&eno;

begin

delrec(num);

end;

/

**FUNCTIONS**

* A function is also similar to a procedure. But returns a value, whereas a procedure does not returns a value.
* A function is also a sub program, which can perform a specific task.
* If a function was developed once and can be used many times ie re-usability of code.

***To create a function:***

Syntax:

CREATE OR REPLACE FUNCTION function\_name([arguments])

RETURN datatype IS|AS

Declarations;

BEGIN

Statements;

Return statement;

END [function\_name];

/

***To call a function:***

To execute the code of a function that function that function must be calling.

***Syntax: variable :=function\_name([args]);***

To create a factorial and prime functions

create or replace function factorial(n number)

return number is

i number;

f number:=1;

begin

for i in 1..n

loop

f:=f\*i;

end loop;

return f;

end factorial;

/

create or replace function prime(n number)

return number is

i number;

c number:=0;

begin

for i in 1..n

loop

if mod(n,i)=0 then

c:=c+1;

end if;

end loop;

if c=2 then

return 1;

else

return 0;

end if;

end prime;

/

To call factorial

declare

n number:=&n;

k number;

begin

k:=factorial(n);

dbms\_output.put\_line('----Factorial : '||k);

end;

/

To call prime

declare

n number:=&n;

k number;

begin

k:=prime(n);

if k=1 then

dbms\_output.put\_line(n||' is prime');

else

dbms\_output.put\_line(n||' is not prime');

end if;

end;

/

**PACKAGES**

A package is a collection of stored functions and stored procedures. It is mainly used to group the related functions and procedures. It is also useful for the re-usability of code ie the code written once and executed many times.

A package is also created using create command. Every package contain 2 parts.

1. Package specification
2. Package body
3. ***Package specification***:- It contain the declarations of the procedures and functions.

Syntax:

Create or replace package packagename IS/AS

Prodecure procedurename;

Function functioname return datatype;

-----

End [packagename];

1. ***Package body:-***

It contain the body of a function or a procedure which is specified in a package specification.

Syntax:

Create or replace package body packagename IS/AS

Procedure procedurename is

Begin

---

End [procedurename];

------

End [packagename];

**To call a procedure:**

Syntax: Packagename.procedurename([args]);

**To call a Fuction**

Syntax: Variable := packagename.fun\_name([args]);

create or replace package maths

is

function add(a number,b number) return number;

function mul(a number,b number) return number;

function sub(a number,b number) return number;

end;

/

create or replace package body maths

is

function add(a number, b number)

return number is

begin

return a+b;

end add;

function sub(a number, b number)

return number is

begin

return a-b;

end;

function mul(a number,b number)

return number is

begin

return a\*b;

end mul;

end maths;

/

To work

declare

a number:=&a;

b number:=&b;

k number;

begin

dbms\_output.put\_line('A = '||a||' B = '||b);

k:=maths.add(a,b);

dbms\_output.put\_line('A+B = '||k);

k:=maths.sub(a,b);

dbms\_output.put\_line('A-B = '||k);

k:=maths.mul(a,b);

dbms\_output.put\_line('A\*B = '||k);

end;

/

**CURSORS**

A cursor is a private sql area. It is used to store the results of a query statement. It is mainly used for processing the rows in one or more tables in pl/sql.

**Types of cursors**

The cursors are classified into two types. They are:

1. Implicit cursors
2. Explicit cursors
3. **Implicit Cursors**

Each query statement can be treated as an implicit cursor. It can be success or fail. An implicit cursor is identified by ***‘SQL’***.

The implicit cursors are automatically opened when the control entered into the executable part and automatically closed before exiting from executable part.

1. **Explicit Cursors**

A cursor which is defined by the user is called an ‘Explicit Cursor’. An explicit cursor can be specified in a pl/sql program. It can store the results of a query statement. These results can be processed in pl/sql.

**Cursor Attributes**

Attributes are nothing but characteristics. For cursors the following 4 attributes are defined in pl/sql. They are:

1. %FOUND
2. %NOTFOUND
3. %ISOPEN
4. %ROWCOUNT
5. **%FOUND –** It returns true when query statement was success ie the rows found in a cursor, otherwise it returns false.

Syntax: cursorname%FOUND

Ex. sql%FOUND

1. **%NOTFOUND** – It returns true when the rows not found in a cursor, otherwise it returns false.

Syntax: cursorname%NOTFOUND

EX. sql%NOTFOUND

1. **%ISOPEN -** It is used to find whether a cursor is opened or not.

Syntax: cursorname%ISOPEN

Ex. c%ISOPEN

Note: It is not applicable on implicit cursors because they are automatically opened and closed.

1. **%ROWCOUNT –** It returns an integer representing the number of rows effected in a cursor.

Syntax; cursorname%ROWCOUNT

Ex. sql%ROWCOUNT

**Example for Implicit cursor**

declare

eno emp.empno%type:=&eno;

e emp%rowtype;

begin

select \* into e from emp where empno=eno;

if sql%found then

dbms\_output.put\_line(e.empno||' '||e.ename||' '||e.job||' '||e.sal);

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('---Employ number not found----');

end;

/

**To work with the explicit cursors the following steps are required.**

1. Cursor declaration
2. Open a cursor
3. Fetch from a cursor
4. Close a cursor
5. **Cursor declaration**:

To define a cursor that cursor must be declared in the declaration part of a pl/sql program. A cursor is declared as follows:

Syntax: Cursor cursorname as select statement;

Ex. cursor c1 as select \*from emp;

1. **Open a cursor**:

To work with the rows in a cursor that cursor must be opened. For this open statement is used.

Syntax: open cursorname;

Ex. open c1;

Note: The implicit cursors are automatically opened and closed.

1. **Fetch statement**

To retrieve the rows from a cursor for processing fetch statement is used. The fetch statement can retrieve only one row at a time from a cursor. To process all rows the fetch statement is applied in a loop statement.

Syntax: Loop

Fetch cursorname into variable;

Exit statement;

End Loop;

1. **Close a cursor**

After completing the process with the opened cursor, finally it must be closed. For this, the close statement is used.

Syntax: close cursorname;

Ex. close c1;

**Explicit Cursor examples**

1. To display the rows in a table

declare

cursor c1 is select \*from emps;

rec emps%rowtype;

begin

open c1;

dbms\_output.put\_line('------------------------------------------');

dbms\_output.put\_line(' EMPLOYEE DATA');

dbms\_output.put\_line('-------------------------------------------');

loop

fetch c1 into rec;

exit when c1%notfound;

dbms\_output.put\_line(rec.empno||' '||rpad(rec.ename,10)||' '||

rpad(rec.job,10)||' '||rec.sal);

end loop;

dbms\_output.put\_line('----'||c1%rowcount||' selected----');

close c1;

end;

/

1. To calculate Bonus and Total salary of an employee using cursors

declare

cursor c1 is select \*from emps;

rec emps%rowtype;

bon emps.sal%type;

tsal emps.sal%type;

begin

open c1;

dbms\_output.put\_line('------------------------------------------');

dbms\_output.put\_line(' EMPLOYEE DATA');

dbms\_output.put\_line('-------------------------------------------');

loop

fetch c1 into rec;

exit when c1%notfound;

if rec.sal>=2000 then

bon:=rec.sal\*.05;

elsif rec.sal>=15000 then

bon:=rec.sal\*.04;

elsif rec.sal>=10000 then

bon:=rec.sal\*.03;

else

bon:=rec.sal\*.025;

end if;

tsal:=rec.sal+bon;

dbms\_output.put\_line(rec.empno||' '||rpad(rec.ename,10)||' '||

rpad(rec.job,10)||' '||rec.sal||' '||bon||' '||tsal);

end loop;

dbms\_output.put\_line('----'||c1%rowcount||' selected----');

close c1;

end;

**TRIGGERS**

Triggers are also stored programs (procedures), which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

* A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
* A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
* A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database etc.

Triggers are mainly used to enforce constraints on a table in a database which can’t be implemented in a database design.

**Advantages of Triggers**

* Generating some derived column values automatically
* Enforcing referential integrity
* Event logging and storing information on table access
* Auditing
* Synchronous replication of tables
* Imposing security authorizations
* Preventing invalid transactions

Syntax:

CREATE OR REPLACE TRIGGER trigger\_name

[BEFORE | AFTER]

[INSERT | UPDATE | DELETE]

ON tablename

[FOR EACH ROW]

[WHEN condition]

DECLARE

Declarations

BEGIN

Executable statements

EXCEPTION

Exception handling statements;

END;

/

Where

Before/after – represents the trigger to be fired before/after trigger event occurs

Insert/delete/update – represents the triggering event

For each row – represents the trigger to be executed for each row or not

**Types of Triggers**

The triggers are of two types.

* + - * 1. Row level triggers
        2. Statement level triggers

***Row level triggers***

The triggers that are triggered or fired for each in an operation then are called ‘Row level triggers’. For this, we can specify **for each row**.

***Statement level triggers***

The triggers that are triggered or fired only once for the entire transaction then are called ‘Statement level Triggers’. For this, **for each row** will be omitted (skipped).

***Raise\_application\_error()***:-

This function is used to raise a user-defined error(s) with user defined code and messages.

Syntax:

Raise\_application\_error(errorcode,error\_message);

:old.colname – represents the value in a table (existing)

:new.colname – represents the new value given

Example:

To raise an error when new salary < old salary

create or replace trigger t1

before insert or update

on emps

for each row

begin

if :new.sal < :old.sal Then

dbms\_output.put\_line('New Salary is always > Old Salary');

raise\_application\_error(-20001,' new salary must be > old salary');

end if;

end;

/

create or replace trigger tr1

after insert or delete or update

on emps

declare

msg varchar(30);

begin

if inserting then

msg:=' row(s) inserted';

elsif deleting then

msg:=' row(s) deleted';

elsif updating then

msg:=' row(s) updated';

end if;

insert into temp values(user,msg,sysdate);

end;

/

**SQL\*Plus commands**

The commands that are used to set the environment in sql are called ‘SQL\*Plus’ commands. They are:

1. Edit – this command is used to open a sql file.

Syntax: ED[IT] filename

Ex. SQL> edit prime

It opens prime.sql file if exists, otherwise opens a new file with prime.sql by prompting.

1. @ - is used to run a sql file

Syntax: @filename

Ex. SQL>@prime

1. / - It is also used to run a command in sql buffer.

Syntax: /

Ex. SQL>/

1. Set - is used to set some properties of sql environment

Syntax: set property value

Ex. SQL>set pagesize 30 - set the no.of lines(rows) per page

SQL>Set linesie 100 - set the no.of cols per a row

SQL>Set serveroutput on/off – set the output on/off

(used to display the output by dbms\_output.put\_line)

1. Show – It is used to display the current user

Ex. SQL>show user

User is “SCOTT”

1. Clear – It is used to clear the screen, cols, breaks etc.

Ex. SQL> cl scr

SQL>cl cols

1. List – It is used display the contents in an sql buffer.

Syntax: List n|\*

N – represents the line number

\*- represents the current line in buffer

Ex. SQL>list - display all in the buffer

SQL>list 3 - display line 3

SQL>list \* - display the current line

1. TTITLE – It is used to set a title for the display

Syntax: TTITLE message [on\off]

On – enable top title

Off – disable top title

Ex. SQL>ttitle ‘Employees List ‘

1. BTITLE – It is used to set bottom title for the display

Syntax: BTITLE message [on|off]

Ex. SQL>btitle ‘----End----‘

1. AUTOCOMMIT – It is used to set a commit command. By default it is on.

Syntax: autocommit on/off

Ex. SQL>autocommit on

It automatically save the changes performed in the current database user.

1. EXIT – It is used to exit from sql.

Syntax: EXIT

Ex. SQL>exit

1. Execute – It is also used to execute/run a sql file

Syntax: execute filename

Ex. SQL>execute prime

It is similar to @ command.

1. Save – It is used to save the contents in sql buffer

Syntax: save filename

Ex. SQL> save abc

It creates abc.sql file

1. Change – This command is used to change a string with another in sql buffer.

Syntax: change/string/newstring

Ex. SQL> select \*from emp;

SQL> change/emp/student

Here, emp will be changed as student

Select \*from student;

**VARRAY**

Varray stands for variable sized array. Like other languages, it is also a homogeneous collection of elements with fixed size. It is similar to structures in ‘C’. for this, oracle provides TYPE statement.

**Declaration**

Syntax: TYPE typename is VARRAY(SIZE)

OF datatype [NOT NULL];

EX. TYPE rec is varray(3)

of varchar(20);

**Initialization:**

Syntax: Variable typename:=typename();

Ex. r rec:=rec();

**Add elements**

Variable.EXTEND;

**Accessing of elements**

syntax:

variable(index)

declare

type NAME is varray(3)

of varchar(20) not null;

x rtype:=rtype('Ramu','Akash');

y rtype:=rtype('Anil','kumar','hari');

i number:=1;

begin

dbms\_output.put\_line('The number of names in x: '||x.count);

loop

dbms\_output.put\_line('Name 1 : '||x(i));

i:=i+1;

exit when i>x.count;

end loop;

dbms\_output.put\_line('The number of names in y: '||y.count);

i:=1;

loop

dbms\_output.put\_line('Name 1 : '||y(i));

i:=i+1;

exit when i>y.count;

end loop;

end;

/

Using loop:

declare

Type Name Is Varray(3)

Of Varchar(20) Not Null;

ename name:=name('Anil','Ramesh','hari');

i number:=1;

begin

dbms\_output.put\_line('No. of names : '||ename.count);

loop

dbms\_output.put\_line('Name '||i||' : '||ename(i));

i:=i+1;

exit when i>ename.count;

end loop;

end;

/

DECLARE

TYPE r\_customer\_type IS RECORD(

customer\_name customers.NAME%TYPE,

credit\_limit customers.credit\_limit%TYPE

);

TYPE t\_customer\_type IS VARRAY(2)

OF r\_customer\_type;

t\_customers t\_customer\_type := t\_customer\_type();

BEGIN

t\_customers.EXTEND;

t\_customers(t\_customers.LAST).customer\_name := 'ABC Corp';

t\_customers(t\_customers.LAST).credit\_limit := 10000;

t\_customers.EXTEND;

t\_customers(t\_customers.LAST).customer\_name := 'XYZ Inc';

t\_customers(t\_customers.LAST).credit\_limit := 20000;

dbms\_output.put\_line('The number of customers is ' || t\_customers.COUNT);

END;

/