RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model.

Entering SQL Commands

Once you are logged into the database using SQL*Plus, you can enter either SQL*Plus commands or SQL commands. There are a few things you should note before you start typing:-

- Commands may be on a single line, or many lines
- You should place different clauses on separate lines for the sake of readability - also make use of tabs and indents
- SQL Command words cannot be split or abbreviated
- SQL commands are not case sensitive
- All commands entered at the SQL*Plus prompt are saved into a command buffer
- You can execute SQL commands in a number of ways:
    - Place a semicolon (;) at the end of the last clause
    - Place a forward slash (/) at the SQL prompt
    - Issue the SQL*Plus r[un] command

The data in RDBMS is stored in database objects called tables. The table is a collection of related data entries and it consists of columns and rows.

A table is the most common and simplest form of data storage in a relational database. Following is the example of a CUSTOMERS table:

Table Figure

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

A record, also called a row of data, is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table:

Row Figure

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would consist of the following:

Column Figure

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.
It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.

NULL Values

If a row contains a column which has no data in it, then its value is said to be NULL.

NULL is a value that is unavailable, unassigned, unknown or inapplicable.

- NULL is not the same as ZERO
- If NULL is part of an expression, then the result will ALWAYS be NULL

Column Types/Datatypes

All columns on a table must be given a datatype, as this determines what kind of data the column can hold. A few of the more common data types are:

Datatype	Purpose
NUMBER	Holds number data of any precision
NUMBER(w)	Holds number data of w precision
NUMBER(w,s)	Holds number data of w precision and s scale, i.e. 10,2 is a number upto 10 digit in length, with 2 digits after the decimal point.
VARCHAR2(w)	Holds variable length alphanumeric data upto w width.
CHAR(w)	Holds fixed length alphanumeric upto w with.
DATE	Holds data/time data
BLOB	Binary data up to 4 GB

Saving, Loading and Executing SQL

Once you are logged into the database using SQL*Plus, you can enter either SQL*Plus commands or SQL commands.

SQL commands are terminated using semicolon(;).

SQL commands are not case sensitive.

SQL commands can be entered in multiple lines terminated with semi-colon.

As well as using the SQL buffer to store SQL commands, you can also store your SQL in files. These files can be edited with your own editor; you can then re-call and run these files as if you had just typed the SQL commands directly into the SQL buffer. Use the following commands from the SQL*Plus prompt:

Command	Description
`SAVE filename`	Saves the current contents of the SQL buffer to a file
`GET filename`	Loads the contents of a file into the SQL buffer
`START filename`	Runs a file (can also use @file)
`ED filename`	Invokes an editor to edit the file
`EXIT`	Quits SQL*Plus

Using DDL - Tables
The `CREATE TABLE` Command

To create a new table within the database, you use the `CREATE TABLE` command. In its most basic form, the `CREATE TABLE` command has the following form:

`CREATE TABLE`

Command used to create tables Syntax

```
CREATE TABLE table-name
(    column_name type(size),
     column_name type(size), . . . );
```

Example

```
CREATE TABLE dept
 (    deptno     NUMBER,
      dname      VARCHAR2(12),
      loc        VARCHAR2(12));
```

`DESCRIBE` - SQL*Plus command

You can list columns on a table from SQL*Plus using the `describe` command (or `desc`) - for example:

```
SQL> desc  dept;
```

The `DROP TABLE` Command

A table can be removed from the database using the DROP TABLE command.

```
DROP TABLE dept;
```

Be aware that once a table has been dropped, it cannot be recovered. Also, ALL data on the table is removed.

Inserting New Data

```
INSERT INTO table_name [( column1, column2....columnN) ]
VALUES ( value1, value2....valueN);
```

Example:

```
INSERT INTO DEPT VALUES (10,'MARKETING','BANG');

INSERT INTO DEPT (deptno,loc) VALUES (30,'BANG');

INSERT INTO DEPT(dname,loc,deptno) VALUES ('RESEARCH','MANG',
20);
```

Deleting Data

For the time being, type Commit at SQL command prompt before deleting.

```
SQL> COMMIT;
```

If you need to delete data within the database, you use the DELETE statement. This allows you to delete a single or many rows at once (as long as you have the correct privileges).

DELETE

SQL statement used to delete rows from the database

Syntax

```
DELETE [FROM] table
[WHERE condition];
```

Example

```
DELETE emp
WHERE  job = 'MANAGER';
```

The above statement says: delete all rows from the EMP table where the job

column is MANAGER.

If DELETE command is used without WHERE condition then all rows in the table get deleted.

The WHERE condition may be compound condition involving multiple condition concatenated with AND ,OR such as-

WHERE sal>4999 AND Depeno='D1'

It is also possible to use subqueries with the DELETE statement.

The COMMIT command

Whenever you issue a DML statement which changes the data held within the database, you are not actually changing the database. You are effectively putting your changes into a buffer, and to ensure this buffer is flushed and all your changes are actually in the database for others to see, you must first commit the transaction. You can do this with the COMMIT statement.

Syntax

```
COMMIT;
```

The ROLLBACK command

If you have started a transaction by issuing a number of DML statements, but you then decide you want to abort the changes and start again, you need to use the ROLLBACK statement.

Syntax

```
ROLLBACK;
```

Updating Existing Data

If you need to change some data within the database, you use the UPDATE statement. This allows you to change a single row or many rows at the same time (as long as you have the correct privileges).

UPDATE

SQL statement used to update rows in the database

Syntax

```
UPDATE table [alias]
SET    column [,column…] =
       {expression,subquery}
[WHERE condition];
```

Example

```
UPDATE emp
SET    sal = sal * 1.1
WHERE  job = 'CLERK';
```

The above statement says: find all employees whose `job` is `CLERK` and set their salary to itself multiplied by 1.1 - or in other words, give all clerks a 10% pay increase.

Example

```
UPDATE Emp
SET    Sal=40000, comm=3000
WHERE  Empno=111;
```

Updating Existing Data

When using the `UPDATE` statement, you should be aware of the following:

- If the `WHERE` clause is omitted then ALL rows on the table will be updated.
- The `WHERE` clause can contain anything that would normally appear in the `WHERE` clause for the `SELECT` statement.
- It is possible to use subqueries and correlated subqueries in the `SET` clause.

Viewing Data -Querying

Let's now try to write our first SQL statement to query the database.

The basic query block is made up of two clauses:

SELECT which columns?

FROM which tables?

- For example:

```
SELECT ename
  FROM emp;
```

The above statement will select the ENAME column from the EMP table.

You can use a * to specify all columns:

```
SELECT * FROM    emp;
```

Selecting Specific Columns

You can select any columns in any order that appear on the table specified in

the `FROM` clause.

- Use a comma (,) as a column separator
- Specify columns in the order you wish to see them in

For example,

```
SELECT    empno, ename,sal   FROM emp;
```

SQL Integrity Constraints

Rules or regulations imposed to ensure data integrity (correctness of data).

- **Column Constraints-** constraints imposed on single column and in command specified along with column definition.
- **Table Constraints**- constraints imposed on
  - Multiple column (combination of column) ,e.g. Combination of **STDCode** and **PhoneNumber** columns must be unique. **OR**
  - a single column where in the condition we use another column. E.g. Assume there are two columns Date_Birth and Date_Retire , if we want to impose condition on Date_Birth that Date_Birth > Date_Retire

- Assertions (Multiple-table Constraints).
- Triggers.
- Primary Key, Foreign Key, Check, Not Null, Unique, …

Column Definition

Syntax for column definition is as below-

col_name data_type [default value] [column constraints]

column level constraint is defined along with the definition of each column.

Syntax for **column constraints**:

[constraint *constraint_name*]

[not] null | check condition |

unique | primary key |

references table_name [(column)]

[on delete cascade]

Table Constraints

Constraint is defined after the definition of all columns.

Syntax for **table constraints**:

[constraint *constraint_name*]

Check condition |

Unique (column {, column}) |

Primary key (column {, column}) |

Foreign key (column {, column})

References table_name [(column {, column})]

[on delete cascade]

Creating Table with Constraints

Create table *Table-Name* (

Col-Name Type-Deft Col-Constraint, … ,

Col-Name Type-Deft Col-Constraint,

Table-Constraint, … ,

Table-Constraint);

ALL Constraints defined are named by default as – Cxxxxx where xxxxx stands for some number, for example C123445. This constraint name will be Unique.

If constraint violated oracle displays message as – **Constraint Cxxxxx violated**

The constraint which is violated cannot be understood by just looking at code Cxxxxx. So user can give own constraint name and user can understand which constraint is violated whenever constraint violation error message is displayed.

It is best practice to give our own meaningful constraint name.

NOT NULL: A NOT NULL constraint can be created only with the column-level approach. Value to a column defined with NOT NULL constraint is mandatory. While inserting data you cannot put null value to such column.

Example:

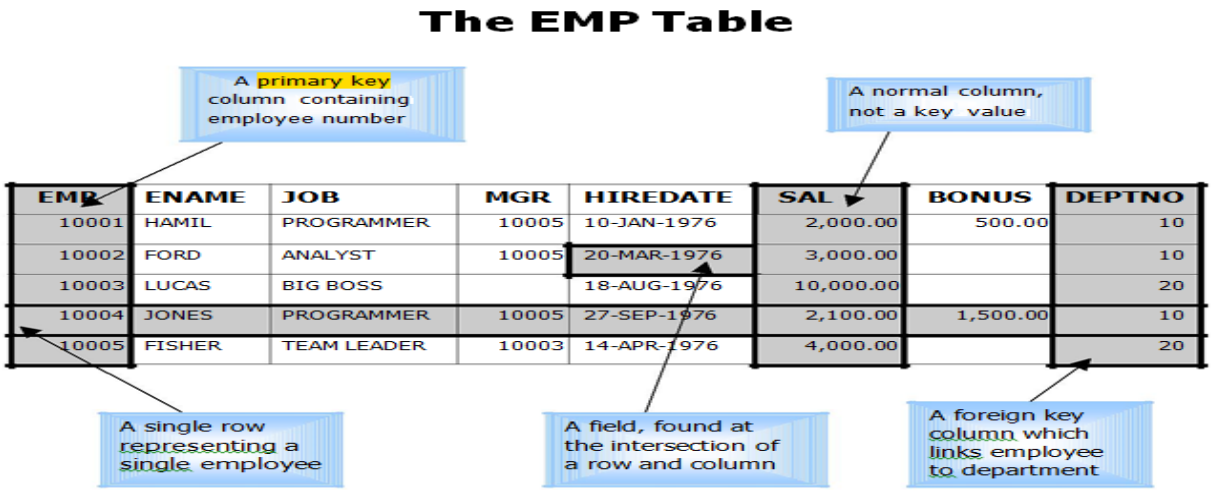
create table Students (SID char (9), Name varchar2(25) not null, Age number(2));

Consider the following Insert command-

INSERT INTO Students (SID, Age) VALUES ('S100', 18);
This results into NOT NULL Constraints Violation Error , because S100 is put to SID column and 18 is put to Age column and NULL value to Name column of a record. NULL to Name results into NOT NULL Constraints Violation Error.

Primary Keys: A primary key is a column/ group of columns that defines the uniqueness of a row. For example, with employee number, you would only ever want one employee with a number of 10001.
Properties:

- A primary key cannot contain duplicate values.
- It cannot be completely or partially NULL.



There can be only one PRIMARY KEY per Table

Primary Key Constraint – Column Level

Example:

CREATE TABLE EMP (

empno NUMBER (6) CONSTRAINT emp_emp_id_PK PRIMARY KEY,

ename VARCHAR2(20), job VARCHAR2(20), hiredate DATE, sal NUMBER(6), bonus NUMBER(5), deptno NUMBER(3));

Primary Key Constraint – Table Level

Example: SID –student ID , CNo-Course Number and Year – Year of Course

```
create table Enrollment
 (SID char(9) not null,
  CNo varchar2(7) not null,
  Year number(2) not null,
  Grade char(2),
  primary key (SID, CNo, Year));
```

Referential Integrity

Need for Referential Integrity Constraint

SID	Name	Age
S101	Ram	
S102	Akshay	
S103	Santosh	

Students (parent table, Students (SID) is parent column)

SID	CNo	Year	Grade
S101	C10	2012	
S101	C11	2013	
S103	C11	2013	
S103	C10	2012	
~~S120~~	~~C13~~	2012	

Enrollment (child)

CNo (Child Column) –Foreign Key referencing CID in Courses Table (parent column)

SID (Child Column) –Foreign Key referencing SID (Parent Column) **of Students Table**

CID	C_Name	Credits	Duration
C10	E.Maths	4	
C11	CSc	4	
C12	Electronics	4	

Courses (parent table, Courses(CID) is Parent column)

Properties:

A Foreign key can contain-

- Only values present in the corresponding Parent Column.
- NULL values, provided Foreign key is not defined with additional NOT NULL constraints.

- Foreign key column can reference to any column (parent column) whose data type, width is same and Parent column has to be defined **with Primary key or Unique constraint**.
- A **Parent Column has to exist before creation of Child Column** with Foreign key Constraint.

Any Insert, Update or Delete, Alter or Drop commands which results into violation of above rules are rejected by the Oracle.

Let us create Parent Tables First.

create table Students (SID char (9) PRIMARY KEY , Name varchar2(25) not null, Age integer);

create table Courses (CID varchar2 (9) UNIQUE , C_Name varchar2(25) not null, Credits number(2), Duration Number(2));

Imposing Referential Integrity – Column Level

create table Enrollment

(SID char (9) NOT NULL References Students,

CNo varchar2 (9) References Courses(CID),

Year number (2) not null,

Grade char (2),

Primary key (SID, CNo, Year));

Imposing Referential Integrity – Table Level

Item_Name	CName	Price
Soap	Godrej	46
Soap	TTK	89
Toothpaste	Dabur	90
Toothpaste	Colgate	75

Items- Parent table

It_Name	CompName	QTy
Soap	Godrej	10
Soap	TTK	20
Soap	TTK	10
Toothpaste	Dabur	5
Soap	Godrej	3
Toothpaste	Dabur	5

Transactions- child table

Example:

create table Items (Item_Name varchar2 (10), CName varchar2 (10), Price Number (5,2), Constraint PK_ITNAME_CNAME **Primary Key(Item_Name, CName)**);

create table Transactions (It_Name varchar2 (10), CompName varchar2(10), Qty Number(2), Constraint FK_It_Name_CpName **Foreign Key(It_Name, CompName) References Items**);

Example:

A table itself can have primary key and foreign key relationship.

EMP table

EMPNO	ENAME	MGRNO
100		103
101		100
103		104
104		104
105		

MGRNO is the Employee number of Manger. Employee with EMpno 103 is the Manger for Employee with Empno 100. Therefore MGRNO is Foreign Key Referencing EMPNO

CREATE TABLE EMP(Empno number(3) PRIMARY KEY, Ename Varchar2(10), MGRNO number(3));

Note: Referential Integrity constraint on MGR_NO has to be defined using **Alter Table** command after creating EMP table

Restrictions on Insert/Update/Delete operations on Columns which are linked with Primary Key (unique), Foreign Key relationship.

1. If you are inserting a value to a foreign key column, the same value must exist previously in the corresponding parent column then Insert is discarded.
   Ex: with reference to Students, Enrollment and Courses table
   INSERT INTO Enrollment VALUES ('S124','C11', 2013,'A');
   Violates Insert restrictions, as S124 does not exist in Parent Column SID in the Students table.
2. If you are updating a foreign key column to a value, the same value must exist previously in the corresponding parent column.
   UPDATE Enrollment SET CNo='C20' WHERE SID='S101';
   This command violates Referential Integrity Constraint, **Why?**
3. Deleting a record from Parent Table when there is Referencing record in the Child table violates the Referential Integrity Constraint.
   DELETE FROM Students WHERE SID='S103';
   Violates DELETE operation, as there exist a child record with value S103 in child column Referencing a Parent record with Parent Column SID value 103 in the Enrollment table.

WHAT IS A FOREIGN KEY WITH CASCADE DELETE IN ORACLE?

A foreign key with cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table with automatically be deleted. This is called a cascade delete in Oracle.

A foreign key with a cascade delete can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Note to students: Find yourself about CASCADE DELETE and CASCADE UPDATE

DEFAULT Constraint

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, SALARY column is set to 5000.00 by default, so in case INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS (
      ID   Number(3)              NOT NULL,
      NAME VARCHAR(20)     NOT NULL,
      AGE  Number(2)              NOT NULL,
      ADDRESS  CHAR (25) ,
      SALARY   Number (18, 2) DEFAULT 5000.00,
      PRIMARY KEY (ID)
);
```

If CUSTOMERS table has already been created, then to add a DFAULT constraint to SALARY column, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS
  MODIFY SALARY NUMBER (18, 2) DEFAULT 5000.00;
```

Check Constraint

A check constraint allows you to specify a condition on the value entering into a column in a row in a table.

Note:

- The check constraint defined on a table must refer to only columns in that table. It cannot refer to columns in other tables.
- A check constraint can NOT include a SQL Sub query.

A check constraint can be defined in either a CREATE TABLE statement or a ALTER TABLE statement.

Using a CREATE TABLE statement

Syntax using a CREATE TABLE statement in Oracle is:

```
CREATE TABLE table_name
```

(
  Column1 datatype null/not null,
  Column2 datatype null/not null,

  ...

  CONSTRAINT constraint_name CHECK (column_name condition) [DISABLE]

);

The DISABLE keyword is optional. If you create a check constraint using the DISABLE keyword, the constraint will be created, but the condition will not be enforced.

Example:

CREATE TABLE Students (SID char (9) PRIMARY KEY , Name varchar2(25) NOT NULL, Age number(2) CONSTRAINT check_Age_Limit CHECK (Age BETWEEN 18 AND 40));

or

CREATE TABLE Students (SID char (9) PRIMARY KEY , Name varchar2(25) not null, Age number(2) CONSTRAINT check_Age_Limit CHECK (Age>=18 and Age<=40));

Any Insert or Update command which try to put a row with Age value less or equal to 18 or Age more than or equal to 40 is rejected.

INSERT INTO Students VALUES ('S103', 'Ganesh', 41); this command will be rejected as Age is more than 40.

CREATE TABLE Students (SID char (9) PRIMARY KEY , Name varchar2(25) CONSTRAINT Capital_Name Check (Name=UPPER(Name)), Age number(2));

Accepts Value only if Name entered is in Capital Letters.

CREATE TABLE Students (SID char (9) CONSTRAINT Start_with_S Check(SID LIKE 'S%') , Name varchar2(25) CONSTRAINT Capital_Name CHECK (Name=UPPER(Name)), Age number(2));

Accepts Value only if SID entered is in starting with **S** letter.

For the time being let us ignore other constraints on Enrollment table and consider only following Check constraint- Grade can take only values A+,A,B,C,D,E,F.

CREATE TABLE Enrollment

(SID char (9),

CNo varchar2 (9),

Year number (2) NOT NULL,

Grade char (2) CONSTRAINT Grade_Letter_A+2F CHECK (Grade

IN ('A+','A','B','C','D','E','F')),

PRIMARY KEY(SID,CNo,Year));

Check(Grade IN('A+','A','B','C','D','E','F')) is equivalent to

Check (Grade='A+' or Grade='A' or Grade='B' or Grade='C' or Grade='D' or Grade='E' or Grade='F'))

ALTER TABLE Statement

The Oracle ALTER TABLE statement is used to add, modify, or drop/delete columns in a table. The Oracle ALTER TABLE statement is also used to rename a table.

Add column in table

Syntax:

To ADD A COLUMN in a table, the Oracle ALTER TABLE syntax is:

ALTER TABLE table_name ADD column_name column-definition;

Example:

ALTER TABLE Students ADD (Last_name varchar2(45), city varchar2(40));

Modify column in table

To MODIFY A COLUMN in an existing table, the Oracle ALTER TABLE

Syntax is:

ALTER TABLE table_name **MODIFY** column_name column_type;

Example:

ALTER TABLE Students **MODIFY (**Last_name **varchar2(100) NOT NULL,**

city varchar2(75));

This command, increases size of Last_Name and imposes NOT NULL constraint , also increases size of city.

Drop column in table

To **DROP A COLUMN** in an existing table, the Oracle ALTER TABLE syntax is:

ALTER TABLE table_name DROP COLUMN column_name;

Example

Let's look at an example that shows how to drop a column in an Oracle table using the ALTER TABLE statement.

For example:

ALTER TABLE Students **DROP COLUMN** Last_name;

Using an ALTER TABLE statement

The syntax for **creating a check constraint in an ALTER TABLE statement** in Oracle is:

ALTER TABLE table_name

ADD CONSTRAINT constraint_name **CHECK** (column_name condition) [DISABLE];

The DISABLE keyword is optional. If you create a check constraint using the DISABLE keyword, the constraint will be created, but the condition will not be enforced.

Example

ALTER TABLE Students
ADD CONSTRAINT check_Stud_Cities
CHECK (City IN ('BNG', 'HYD', 'MNP'));

Create unique contraint - Using an ALTER TABLE statement

Syntax:

ALTER TABLE table_name

ADD CONSTRAINT constraint_name **UNIQUE (column1, column2, ... column_n);**

Example

ALTER TABLE Customer **ADD CONSTRAINT CustId_UNQ UNIQUE(ID);**

Adding Foreign Key Using ALTER TABLE statement

Syntax

The syntax for creating a foreign key in an ALTER TABLE statement is:

ALTER TABLE table_name

ADD CONSTRAINT constraint_name

FOREIGN KEY (column1, column2, ... column_n**)**

REFERENCES parent_table (column1, column2, ... column_n**);**

Example: Assume that you have created a table DEPT (Dno,Dname,City) with Dno being Primary key. Another table say EMP(Empno,Ename,Salary,Deptno) is created with Empno being Primary key.

Assume that we want put foreign key constraint on Deptno Referencing Dno using Alter Table

```
ALTER TABLE Emp
        ADD CONSTRAINT fk_Deptno
                FOREIGN KEY (Deptno)
                        REFERENCES Dept (Deptno);
```

Adding Primary key Using ALTER TABLE Command.

Syntax:

ALTER TABLE table_name

ADD CONSTRAINT constraint_name

Primary Key(Column1,Column2,…,Column_n);

Example:

Assume that You have created EMP table without making Empno as Primary Key.

Make Empno as primary key of EMP.

Example:

```
ALTER TABLE Emp
                ADD CONSTRAINT Empno_PK
```

PRIMARY KEY (Empno);

Removing Constraints

Syntax:

ALTER TABLE tablename **DROP CONSTRAINT** constraint_name**;**

Example: Assume that you want to drop primary constraint defined on Empno column of EMP table above. Note: You must know the name of constraint (it may be system defined constraint like **Cxxxx** or user defined Empno_PK)

ALTER TABLE EMP DROP CONSTRAINT Empno_PK;

DISABLING CONSTRAINT

Syntax:

ALTER TABLE table_name **DISABLE CONSTRAINT** constraint_name**;**

SOME TABLE FROM DATA DICTIONARY

USER_CONS_COLUMNS and USER_CONSTRAINTS

These table can be described to view the columns defined and can be queried to find the information about in which table, which column which constraint is defined.

DESC USER_CONS_COLUMNS;

Or

DESC USER_CONSTRAINTS;

The following SELECT command can be used to find COLUMN name on which Empno_PK constraint is applied.

SELECT COLUMN_NAME FROM USER_CONS_COLUMNS WHERE CONSTRAINT_NAME='Empno_PK' AND TABALE_NAME='EMP';

Oracle RDBMS: Extracting the Table, Index & View Definitions (DDL) and Indexed Columns

By calling the GET_DDL() function of metadata package DBMS_METADATA.

Syntax:
select DBMS_METADATA.GET_DDL('TABLE','<table_name>') from DUAL;

eg.,

SQL> set long 1000

SQL> set pagesize 0

SQL> **select DBMS_METADATA.GET_DDL('TABLE','EMP') from DUAL;**

Sample output

To Extract The Index Definition (DDL Statement) From An Oracle Database For A Given Index Name

Syntax:

select DBMS_METADATA.GET_DDL('INDEX','<index_name>') from DUAL;

eg., Assume EMP is a table with Phone as one column and we want to create an index on Phone column.

SQL> create index Phone_IDX on EMP (PHONE);

Index created.

If already Index is created on the Phone column of some table with index name Phone_IDX , we can get DDL command using command below.

select DBMS_METADATA.GET_DDL ('INDEX','Phone_IDX') from DUAL;

If the interest is only to get the indexed column names for an index, simply query COLUMN_NAME of table USER_IND_COLUMNS.

Syntax:

select COLUMN_NAME from USER_IND_COLUMNS where INDEX_NAME = '<index_name>';

eg.,

column COLUMN_NAME format A15

select COLUMN_NAME from USER_IND_COLUMNS where INDEX_NAME = 'PHONE_IDX';

Returns value as below-
COLUMN_NAME

PHONE

To Get The Corresponding DDL Statement That Was Used To Create The View

There may be privileges on creating view or Querying DBA_VIEWS table. Login with proper **userid** with required privileges and check.

Query the TEXT column of table DBA_VIEWS.

Syntax:

SQL> set long 10000

SQL> SELECT TEXT FROM DBA_VIEWS WHERE OWNER = '<owner_name>'

AND VIEW_NAME = '<view_name>';

SQL> create view EMP_View as Select empno,ename,sal from scott.emp;

View created.

SQL> set long 1000

SQL> **select TEXT from DBA_VIEWS where OWNER = 'SCOTT'**

and VIEW_NAME = 'EMP_VIEW';

TEXT

Select Empno, Ename ,Sal From Scott.Emp;

SQL BASICS-SELECT

The basic query block is made up of two clauses:

- `SELECT` which columns?
- `FROM` which tables?

For example:

```
SELECT ename FROM emp;
```

The above statement will select the `ENAME` column

from the EMP table. Displays all ename column values in all the records in the EMP table.

You can use a * to specify all columns in all the records:

```
SELECT * FROM emp;
```

Selecting Specific Columns

You can select any columns in any order that appear on the table specified in the FROM clause.

- Use a comma (,) as a column separator.
- Specify columns in the order you wish to see them in.

For example,

```
SELECT  empno , ename , sal
FROM emp;
```

Displays empno, ename, sal column values of all the rows in the EMP table.

Arithmetic Operators

We can perform some arithmetic calculations based on the data returned by the SELECT statement. This can be achieved using SQL's arithmetic operators:

- Multiply *
- Divide /
- Add +
- Subtract -

Normal operator precedence applies - you can also use brackets to force precedence.

For example, to find the annual salary of all employees:

```
SELECT  empno, sal * 12 FROM  EMP;
```

Column Aliases

In the above example **sal*12** values are displayed with column name as ***sal*12*** only. We can give meaningful alias name to ***sal*12*** column heading name.

For example:

```
SELECT empno   employee_number
,      sal*12  annual_salary
```

```
FROM    Emp;
```

In the above example now empno values will have heading **employee_number** and sal*12 will have headings **annual_salary.**

Column aliases must not contain any white space and the case is ignored. You can get around this by enclosing the alias in double quotes, as follows:

```
SELECT    empno     "Employee Number"
,         sal*50    "Annual Salary"
FROM emp;
```

Duplicate Rows

When we use SELECT command it displays all the records even though there are duplicate rows.

For example,

```
SELECT deptno FROM
emp;
```

It displays all **deptno** values of all rows even though there are duplicate values in the deptno column.

We can avoid display of duplicate rows using DISTINCT clause as below.

for example,

```
SELECT    DISTINCT  deptno
FROM      emp;
```

The DISTINCT keyword affects ALL columns in the SELECT clause.

Ordering Data

The order of rows returned by a SELECT statement is, by default, undefined. You can use the ORDER BY clause to sort the rows. For example,

```
SELECT    empno
FROM      emp
ORDER BY  empno;
```

The ORDER BY clause is simply added to the end of your SELECT statement.

- Default order is ascending - use DESC after the column name to change order
- There is no limit on the number of sort columns
- There is no need to SELECT sort column

- You can sort using expressions and aliases `NULL` values are sorted

High

ORDER BY Examples

Here are a few examples of using the `ORDER BY` clause:

```
SELECT    empno,
FROM      emp
ORDER BY empno;
```

By **default** sorts in ascending order by empno.

```
SELECT    ename,
          sal*12  renum
FROM      emp
ORDER BY  renum DESC;
```

sorts in ascending order by renum i.e.sal*12 values.

```
SELECT    deptno,
          hiredate ,
          ename  FROM
     emp
ORDER BY  deptno
     ,    hiredate DESC ;
```

Sorts records in **ascending order by deptno** column values and in any rows if deptno value is same then those only records are sorted in **Descending order by hiredate** values. Deptno is considered as **Primary sorting field** and hiredate is considered as **Secondary sorting field**.

Row Restriction

Assume that we only want a list of employees who work in department 10, we would use the WHERE clause. The WHERE clause MUST appear after the FROM clause. Need to specify conditions in the WHERE clause that must be met if the row is to be returned. Conditions are basically comparisons of columns/literals using logical operators and SQL operators.

For Example:

```
SELECT    empno
FROM      emp
WHERE     deptno = 10
ORDER BY  empno;
```

This command displays all employees rows for which deptno =10 (in other words all the employees who are in deptno 10 evaluates to TRUE and all such rows are

sorted in Ascending order by empno.

The following logical operators are available:

Operator	Meaning
=	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

A `WHERE` clause is generally made up of three elements:

- Column name
- Comparison operator (logical operator)
- Column name/literal

Ex: Find the employee names who are working in department 10 and drawing salary more than 50000/-.

```
SELECT    Ename
FROM      Emp
WHERE  deptno = 10 AND sal >50000;
```

Ex: Find the employee names who are working in department 10 or 20 or 30.

```
SELECT    Ename
FROM      Emp
WHERE  deptno = 10 OR deptno=20 OR deptno=30;
```

OR Same can be written using **IN** operator.

```
SELECT    Ename
FROM      Emp
WHERE  deptno IN (10,20,30);
```

Ex: Find the employee names that are from cities – Udupi, Manipal, Mangalore

```
SELECT      Ename
FROM        Emp
WHERE City IN ('Udupi' , 'Manipal', 'Mangalore');
```

OR we can write using OR operator.

```
SELECT      Ename
FROM        Emp
WHERE City='Udupi' OR City='Manipal' OR City='Mangalore';
```

Conditions in WHERE clause can be manipulated by using brackets.

Ex: Find the employee names that are working in department 10 and living in city Udupi or Manipal.

```
SELECT      Ename
FROM        Emp
WHERE deptno = 10 AND (City='Udupi' OR City='Mangalore');
```

RANGE SEARCHING

Ex: Find the employee names that are having commission (COMM column) between 3000 to 5000.

```
SELECT     Ename
FROM       Emp
WHERE   comm >=1000 AND comm <=5000;
```

OR you can write also.

```
SELECT     Ename
FROM       EMP
WHERE    comm BETWEEN 1000 and 5000;
```

Pattern Matching

% used to ignore any number of characters.

_ used to ignore single character

Ex: Find the employee names that are in the city names with first letter 'U'.

```
SELECT     Ename
FROM       EMP
WHERE      City LIKE 'U%';
```

U% is used to indicate that after U ignore all the characters in the City column.

Ex: Find the employee names which are having first letter 'M' , second letter not

known, third letter ‘n’ and remaining letters not known in the City column.

```
SELECT    Ename
FROM      EMP
WHERE     City LIKE 'M_n%';
```

M_n% means picks Enames where City columns are having first character is M and second character is ignored by using _ (underscore) and remaining characters after ‘n’ is ignored by using %.

This query selects Employee names that are in the city Manipal or Mangalore etc. as in first character is M and third character is n.

Using Oracle In-Built Functions

Character Functions

LOWER

Converts all characters to lower case

Syntax

```
LOWER(argument)
```

Example

```
SELECT    LOWER('VINAYAK') FROM DUAL;  returns  vinayak
```

UPPER

Converts all characters to upper case

Syntax

```
UPPER(argument)
```

Example

```
SELECT    UPPER('java') FROM DUAL;  returns  JAVA
```

INITCAP

Forces the first letter of each word to be in upper case

Syntax

```
INITCAP(argument)
```

Example

```
SELECT    INITCAP('manipal university')  FROM DUAL;
```

returns Manipal University

LPAD

Pads string to the left with a specified character until a specified length is reached

Syntax

```
LPAD(string,padstrlen,padstring)
```

string the string to be padded
padstrlen the length of the final string after padding
padstring the string to use for padding
Example:

```
SELECT    LPAD('Two Thousands',18,'*') from dual;
```
returns *******Two Thousands**

RPAD

Pads string to the right with a specified character until a specified length is reached

Syntax:

```
RPAD(string,padstrlen,padstring)
```
Arguments meaning same as LPAD

Example:

```
SELECT    RPAD ('Two Thousands',18,'*') from dual;
```
Returns **Two Thousands*******

SUBSTR

The SUBSTR function is used to extract a portion of a string.

Syntax

```
SUBSTR(string,stratpos,no_of_char)
```

Arguments

string the string to be extracted from.
stratpos starting position from which to extract characters.
no_of_char number of characters to be extracted from startpos.

Example

```
SELECT    SUBSTR('ComPuter',3,5) from Dual;
```
Returns mPuter

LENGTH

The length function returns the number of characters in a string.

Syntax

```
LENGTH(string)
```
Arguments

string The string you want the length of.

Example

1. SELECT LENGTH('Computer') From Dual;

Returns 8

2. Select Length(Ename) from EMP;

Returns string lengths of all values in Ename column.

REPLACE

The REPLACE function searches through a string for another string and replaces all occurrences of it with another string

Syntax

REPLACE(string1,search_str,replace_str)

String1- The string you wish to search for search_str.

search_str – Is the string which you want to Search in the string String1.

Replace_str - The string used to replace search_str in the string String1.

Example

```
SELECT REPLACE ('my name is xyz','xyz','raj') from dual;
```

Returns - my name is raj

Searches for 'xyz' string in the given string 'my name is xyz' and replaces all occurrences of 'xyz' with 'raj'.

Numeric Functions

TRUNC

The TRUNC function truncates a number to a specified number of decimal places.

Syntax

```
TRUNC(number,n)
```

Arguments

number The number you want to truncate

n The number of decimal places

Example

```
SELECT TRUNC(3.142,1)from dual;
```

Returns 3.1 value.

POWER

Syntax:
POWER(m,n) Returns m^n value.
Example:
SELECT POWER (5,3) FROM DUAL;
Returns 125.

ABS
Syntax:
ABS(m) Returns absolute value of m.
Example:
SELECT ABS (-123) FROM DUAL;
Returns 123.

SQRT
Syntax:
SQRT(m) Returns square root of m.
Example:
SELECT SQRT (9) FROM DUAL;
Returns 3.

MOD
Syntax:
MOD(m,n) Returns remainder after dividing m by n.
Example:
SELECT MOD(27,7)FROM DUAL;
Returns 6.

Conversion Functions

TO_CHAR
The TO_CHAR function is used convert a value into a char, with or without a specified format.

Syntax

```
TO_CHAR(number)
TO_CHAR(number,format)
TO_CHAR(date)
TO_CHAR(date,format)
```

Examples:
Convert a number 123 to a char 123

```
SELECT TO_CHAR(123) FROM DUAL;
```
Returns **123** which will be a character value.

Convert a number to a char and display as a 5 digit char:
```
SELECT TO_CHAR(123,'99999') FROM DUAL;
```
Returns **123** as character value.

```
SELECT TO_CHAR (123,'999.99') FROM DUAL;
```
Returns **123.00** as character value.

99999 or 999.99 are said to mask formats, following are the different mask formats

Format Mask	Meaning
9	Numeric position, number of 9's determine width
0	Same as 9 except leading 0's are displayed
$	Floating dollar sign
.	Decimal point position specified

Example:
```
SELECT 'Salary='||TO_CHAR(sal,'9990.00') FROM EMP;
```

TO_NUMBER

The TO_NUMBER function is used convert a char into a number.

Syntax
```
TO_NUMBER(string)
```
Example
```
SELECT TO_NUMBER ('123') FROM DUAL;
```
Returns **123** which will be a NUMERIC value.

TO_DATE

The TO_DATE function is used convert a char into a date with default date format or any other format we wish given as format mask.

Syntax
```
TO_DATE(string)
          OR
TO_DATE(string,format)
```

Arguments

string The string to be converted

format The format mask you wish to apply to the input string: this ensures that the string is in a correct date format. If

format is omitted then the default date format (usually DD-MON-YY) is used.

Date Format Masks

Format Mask	Meaning
YYYY,YYY,YY,Y	Displays year in 4, 3, 2 or 1 digits.
MON,MONTHS,MM	3 Character spelled month (like JAN,FEB etc), full month Spelling (January, February etc.) or 2 digit month number(1,2,..12)
DY,DAY,DDD,DD,D	3 letter spelled day, fully spelled day , day number of year, day of month or day of week
WW,W	Week of month or year

Time Format Mask

Format Mask	Meaning
HH,HH12,HH24	Hour of day, Hours 1-12 or Hours 1-24
MI	Minute
SS	Second
SSSSS	Seconds since midnight

Example

```
SELECT TO_DATE('10-JUL-1999','DD-MONTH-YY') from dual;

SELECT TO_DATE('10-JUL-1999','DD-MONTH-YY') from dual;

SELECT TO_CHAR(SYSDATE,'DD-MM-YY') from dual;
```

Returns

```
TO_CHAR(
--------
21-08-14
```

```
SELECT TO_CHAR(SYSDATE,'MM-YY') from dual;
Returns
TO_CH
--------
08-14

SELECT TO_CHAR(JOINDATE,'DD') from EMP;
If Joindate is 15-Aug-2014 then it returns 15.

SELECT TO_CHAR (JOINDATE,'MONTH') from EMP;
It returns AUGUST

SELECT TO_CHAR (TO_DATE ('15-AUG-14'), 'Day') from DUAL;
TO_CHAR(T
---------
Friday
```

EXTRACT

The **EXTRACT** function can be used in place of the **TO_CHAR** function when you are selecting portions of date values—such as just the month or day from a date.

```
Syntax:
EXTRACT
(YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
 FROM datetime_value_expression )
Example:
SELECT EXTRACT (YEAR FROM TO_DATE ('10-JUL-1999','dd-MON-yyyy'))
from dual;
Returns 1999

SELECT EXTRACT (MONTH FROM TO_DATE ('10-JUL-1999','dd-MON-
yyyy')) from dual;
Returns 7

SELECT EXTRACT (DAY FROM TO_DATE ('10-JUL-1999','dd-MON-yyyy'))
from dual;
Returns 10

SELECT EXTRACT (DAY FROM Birth_Date) from EMP;
Returns column containing Day part of birt_day of all employees.

SELECT EXTRACT (MONTH FROM Birth_Date) from EMP WHERE Empno=101;
```

Returns month number from the date of birth of employee with employee number 101.

Aggregate Functions

Function	Value Returned
AVG(n)	Returns average on n, ignoring nulls
COUNT(n|*)	Returns number on non-null rows using column n. If * is used then all rows are counted
MAX(expr)	Maximum value of expr
MIN(expr)	Minimum value of expr
SUM(n)	Sum of n, ignoring nulls

n- can be prefixed with the keyword DISTINCT - this will make the group function only work on unique values of the column specified by n.

Examples of using group functions

To find total paid in salaries for all employees:

```
SELECT SUM(sal),COUNT(Empno) FROM EMP;
```

To find highest, lowest and average salary:

```
SELECT MAX(sal), MIN(sal), AVG(sal) FROM EMP;
```

To find total paid in salaries for all employees in department 20:

```
SELECT SUM(sal)
FROM EMP
WHERE department = 20;
```

Grouping Data

The **example table EMP** considered below for the discussion of Data Grouping.

```
SQL> select empno,job,deptno,sal from emp;

     EMPNO JOB              DEPTNO        SAL
---------- --------- ---------- ----------
      7369 CLERK              20        800
      7499 SALESMAN           30       1600
      7521 SALESMAN           30       1250
      7566 MANAGER            20       2975
      7654 SALESMAN           30       1250
      7698 MANAGER            30       2850
      7782 MANAGER            10       2450
      7788 ANALYST            20       3000
      7839 PRESIDENT          10       5000
      7844 SALESMAN           30       1500
      7876 CLERK              20       1100

     EMPNO JOB              DEPTNO        SAL
---------- --------- ---------- ----------
      7900 CLERK              30        950
      7902 ANALYST            20       3000
      7934 CLERK              10       1300

14 rows selected.
```

We can split the records in a table into smaller groups; we can then use Group Functions to return **summary information** (such as sum, avg, max, min etc.) about each group. We split a table using the GROUP BY clause.

The GROUP BY clause instructs the query to return rows split into groups determined by the **specified columns** called **GROUP BY Field.**

Example: Find each kind of job and average salary of each kind of job in the entire organization.

```
SELECT    job ,AVG(sal)
FROM   EMP GROUP BY  job;
```

The data has been grouped by the job column; the AVG group function has then returned summary data based on all rows in the table that are in the current group.

Some expected output is as below-

```
SQL> select job, avg(sal) from emp group by job;

JOB          AVG(SAL)
--------- ----------
CLERK          1037.5
SALESMAN         1400
PRESIDENT        5000
MANAGER    2758.33333
ANALYST          3000
```

- **Rows can be omitted from the grouped data by using the WHERE clause**

Example: Selecting some records and forming groups among only those records.

Find average salary of each kind of jobs within the department with number 20.

```
SELECT    job, AVG (sal)
FROM   EMP WHERE Deptno =20 GROUP BY job;
```

```
SQL> select job, avg(sal) from emp where deptno=20 group by job;

JOB         AVG(SAL)
--------- ----------
CLERK             950
MANAGER          2975
ANALYST          3000
```

- **You are not restricted to a single column. You can group by as many columns as you like, as long as the columns you are grouping by are in the SELECT clause (refer EMP table data for the example)**

Example: Find the sum of salary of each kind of jobs within each department.

```
SQL> select job,deptno,sum(sal) from emp group by job,deptno;

JOB           DEPTNO   SUM(SAL)
--------- ---------- ----------
MANAGER           20       2975
PRESIDENT         10       5000
CLERK             10       1300
SALESMAN          30       5600
ANALYST           20       6000
MANAGER           30       2850
MANAGER           10       2450
CLERK             30        950
CLERK             20       1900

9 rows selected.
```

- **Groups can be omitted from the results by using the HAVING clause**

  You can omit groups returned from a query which use a GROUP BY clause by using the HAVING clause.

  Example: Find the Jobs and average salary of each kind of job and list only those for which average salary is more than 10000.

```
SELECT    job , AVG(sal)
 FROM  EMP GROUP BY   job HAVING AVG(Sal) >2000;
```

```
SQL> select job,avg(sal) from emp group by job having avg(sal)>2000;

JOB         AVG(SAL)
--------- ----------
PRESIDENT        5000
MANAGER    2758.33333
ANALYST          3000
```

Retrieving data from More Than One Table Joins

So far, any queries we've seen have been from a single table at a time - but SQL allows you to **query many tables** at the same time through the **use of joins.**

Cross-product

If you construct a SELECT statement which contains information from two or more tables without specifically linking any of the columns from one table to the next, the resulting query would be what is known as a Cross-product (sometimes referred to as a Cartesian join). This basically means that ALL rows from ALL tables are returned in EVERY combination.

```
SQL> select ename,dname from emp, dept;

ENAME      DNAME
---------- --------------
SMITH      ACCOUNTING
ALLEN      ACCOUNTING
WARD       ACCOUNTING
JONES      ACCOUNTING
MARTIN     ACCOUNTING
BLAKE      ACCOUNTING
CLARK      ACCOUNTING
SCOTT      ACCOUNTING
KING       ACCOUNTING
TURNER     ACCOUNTING
ADAMS      ACCOUNTING

ENAME      DNAME
---------- --------------
JAMES      ACCOUNTING
FORD       ACCOUNTING
MILLER     ACCOUNTING
SMITH      RESEARCH
ALLEN      RESEARCH
WARD       RESEARCH
JONES      RESEARCH
MARTIN     RESEARCH
BLAKE      RESEARCH
CLARK      RESEARCH
SCOTT      RESEARCH
```

……..

```
ALLEN          OPERATIONS

ENAME          DNAME
---------- --------------
WARD           OPERATIONS
JONES          OPERATIONS
MARTIN         OPERATIONS
BLAKE          OPERATIONS
CLARK          OPERATIONS
SCOTT          OPERATIONS
KING           OPERATIONS
TURNER         OPERATIONS
ADAMS          OPERATIONS
JAMES          OPERATIONS
FORD           OPERATIONS

ENAME          DNAME
---------- --------------
MILLER         OPERATIONS

56 rows selected.
```

This is all possible combination of EMP records with DEPT records.All these joined records do not represent actual records. For example assume that James is working in Sales department. The record JAMES SALES in the cross product is showing actual fact, but JAMES RESEARCH and JAMES OPERATIONS are the some records that do not represent actual fact. In Joining with equality condition (Equi-Join) yields only records representing actual facts.

Joins - Equi

An equi join is a join which directly links columns from one table to another, or in other words, an equi join joins tables where a column on one table is equal to a column from another table.

```
SELECT     emp.ename,dept.dname
FROM emp,dept
WHERE      dept.deptno = emp.deptno;
```

The above statement joins the emp and dept tables using the deptno column, and in English the statement reads: select the ename column from the emp table and get the dname column from the dept table, only select dept rows where the deptno on dept is the same as the deptno on the emp table.

```
SQL> select ename,dname from emp, dept where emp.deptno=dept.deptno;

ENAME      DNAME
---------- --------------
CLARK      ACCOUNTING
KING       ACCOUNTING
MILLER     ACCOUNTING
JONES      RESEARCH
FORD       RESEARCH
ADAMS      RESEARCH
SMITH      RESEARCH
SCOTT      RESEARCH
WARD       SALES
TURNER     SALES
ALLEN      SALES

ENAME      DNAME
---------- --------------
JAMES      SALES
BLAKE      SALES
MARTIN     SALES

14 rows selected.
```

Joins - Self

By using a self join with table aliases you can join a table to itself. A self join basically allows you to select from the same table more than once within the same SQL statement - this is very useful if a table has rows on it which relate to other rows on the same table. For example, the emp table holds employees, and each employee has a manager (except the big boss). This manager is stored on the same table: so, you would need a self join if you wanted to create a statement that listed all employee names along with their manager name. e and m are the two table aliases for the same table EMP.

```
SELECT    e.ename   employee_name
   ,  m.ename   manager_name
        FROM    emp e,emp   m
        WHERE   m.empno =    e.mgr;
```

The above statement says: select the employee name from emp, and call it employee_name, then select the employee name again and call it manager_name from emp where the employee number (empno) is the same as the manager (mgr) stored on the first record.

Outer Join

An outer join allows you to join tables together and still return rows even if one side of a condition is not satisfied. Depending on it is left/right side we choose to display all the records (even if condition not matched) we call as left/right outer join. **(The syntax for an outer join uses (+) on the side of the join that will be returning additional rows i.e which side we want to display rows even if do not**

match condition) . We can tell Oracle to perform a left, right, or full outer join.

Ex: Assume in DEPT table we have a department 'Operations' in which there are no employees. If we want to display all departments their employees names, along with department names in which there are no employees also.

```
SELECT    e.ename,d.dname
FROM emp e, dept    d
WHERE     d.deptno =     e.deptno(+);
```

Here e and d are the table aliases.

```
SQL> SELECT     e.ename,d.dname
  2  FROM       emp e,dept      d
  3  WHERE      d.deptno =      e.deptno(+);

ENAME      DNAME
---------- --------------
CLARK      ACCOUNTING
KING       ACCOUNTING
MILLER     ACCOUNTING
JONES      RESEARCH
FORD       RESEARCH
ADAMS      RESEARCH
SMITH      RESEARCH
SCOTT      RESEARCH
WARD       SALES
TURNER     SALES
ALLEN      SALES

ENAME      DNAME
---------- --------------
JAMES      SALES
BLAKE      SALES
MARTIN     SALES
           OPERATIONS

15 rows selected.
```

LEFT OUTER JOIN

```
SELECT *
FROM A, B
WHERE A.column = B.column(+)
```

RIGHT OUTER JOIN

```
SELECT *
FROM A, B
WHERE B.column(+) = A.column
```

Subqueries

A sub query is basically a SELECT statement within another

SELECT statement (nested queries); they allow you to select data based on unknown conditional values. A subquery generally takes the form:

```
SELECT column(s) FROM table(s)
WHERE     column(s) = (SELECT column(s)
                       FROM table(s)
                       WHERE condition(s) );
```

The subquery is the part in bold and in brackets: this part of the query is executed first, just once, and its result is used in the main (outer) query.

Ex: Find the employee names who are working in departments in the location (Loc)- 'NEW YORK'.

```
SELECT    Ename
FROM      Emp
WHERE  deptno IN
(Select deptno from Dept where Loc='NEW YORK');
```

Inner query (**Select deptno from Dept where Loc='NEW YORK'); retrives deptno value 10**

```
SQL> Select deptno from Dept where Loc='NEW YORK';

    DEPTNO
----------
        10
```

Equivalently the query is as below-

```
SELECT    Ename
FROM      Emp
WHERE  deptno IN  (10)
```

If inner query returns single value then we can rewrite query by replacing IN with = .

```
SELECT    Ename
FROM      Emp
WHERE  deptno =
Select deptno from Dept where Loc='NEW YORK');
```

Ex: Find the employee names who are working in departments 'RESEARCH' or 'SALES'

```
SELECT    Ename
FROM      Emp
WHERE  deptno IN
(Select deptno from Dept where dname='SALES' OR
dname='RESEARCH');
```

The inner query (Select deptno from Dept where dname='SALES' OR

dname=’RESEARCH’); returns deptno values -20, 30

The query is equivalent to

```
SELECT    Ename
FROM      Emp
WHERE  deptno IN (20,30);
```

Ex:Find the name of employees who are working in job as that of BLAKE.

```
SELECT ENAME FROM EMP WHERE JOB = (SELECT JOB FROM EMP WHERE ENAME='BLAKE');
```

```
ENAME
----------
JONES
BLAKE
CLARK
```

Multiple Row Subqueries

A subquery can return more than one row, but you must use a multi-row comparison operator (such as IN) in the outer query or an error will occur:

```
SELECT ename,sal,deptno
FROM emp WHERE (deptno,sal)
IN (SELECT deptno,MIN(sal)
FROM emp
GROUP BY  deptno);
```

The above statement executes the subquery first to find the lowest salary in each department (by using a GROUP BY), then it uses each row returned from that query to find all employees who earn that amount in each department.

ANY/SOME Operator

The ANY (SOME) operator compares a value to EACH row returned from the subquery.

```
SELECT ename,sal,job,deptno FROM emp
WHERE     sal > ANY
(SELECT DISTINCT SAL FROM emp
WHERE     depton = 30);
```

The above statement executes the subquery first to find all distinct salaries in department 30, and the > ANY part of the outer query says where the sal column in greater than ANY of the rows returned by the subquery. This effectively says: list all employees whose salary is greater than the lowest salary given in the department 30.

Set Operators

- UNION
- INTERSECT
- MINUS

UNION, INTERSECT and MINUS set operators are used when you need to construct two or more queries.

UNION

The UNION set operator combines the results of two or more queries and returns all distinct rows from all queries.

Ex: Display Employee names who are working in deptno 10 , 20.

```
SELECT    Ename
FROM      Emp
WHERE  deptno=10
UNION
SELECT    Ename
FROM      Emp
WHERE  deptno=20;
```

or we can also write as below-

```
SELECT    Ename
FROM      Emp
WHERE  deptno=10 OR deptno=20;
```

INTERSECT

The INTERSECT set operator combines the results of two or more queries and returns only rows which appear in BOTH queries.

Ex: Find the name of employees working in department –'RESEARCH' and drawing salary more than 2500.

```
Select ename, dname from emp,dept where emp.deptno=dept.deptno
and dname='RESEARCH' INTERSECT
Select ename,dname from emp,dept where emp.deptno=dept.deptno
and sal>2500;

ENAME      DNAME
---------- --------------
```

```
FORD       RESEARCH
JONES      RESEARCH
SCOTT      RESEARCH
```

We can view result generated by individual queries query 1

```
select ename,dname from emp,dept where emp.deptno=dept.deptno
and dname='RESEARCH';

ENAME      DNAME
---------- --------------
JONES      RESEARCH
FORD       RESEARCH
ADAMS      RESEARCH
SMITH      RESEARCH
SCOTT      RESEARCH
```

query 2

```
select ename,dname from emp,dept where emp.deptno=dept.deptno
and sal>2500;

ENAME      DNAME
---------- --------------
KING       ACCOUNTING
FORD       RESEARCH
JONES      RESEARCH
SCOTT      RESEARCH
BLAKE      SALES
```

query 1 INTERSECT query 2 GIVES

```
ENAME      DNAME
---------- --------------
FORD       RESEARCH
JONES      RESEARCH
SCOTT      RESEARCH
```

MINUS

The MINUS set operator combines the results of two or more queries and returns only rows that appear in the first query and not the second.

Ex: Find ename and deptno from emp who are not working in deptno 20.

```
Select empno,ename,deptno from emp MINUS select
empno,ename,deptno from emp where deptno=20;
```

```
     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      7499 ALLEN              30
      7521 WARD               30
      7654 MARTIN             30
      7698 BLAKE              30
      7782 CLARK              10
      7839 KING               10
      7844 TURNER             30
      7900 JAMES              30
      7934 MILLER             10

9 rows selected.
```

Indexes

An index is a data structure within the database that allows you to provide quick access to data on a table via a particular column or columns. Its use is similar to the indexes in book, which allows quick search to locate a chapter. Searching with indexes is much faster than sequential index.

CREATE INDEX

You create indexes with the CREATE INDEX command.

Syntax:

```
CREATE [UNIQUE] INDEX index_name ON table_name
(column [,column . . .]);
```

Examples

To create a unique index called empno_idx on the emp table using the empno column:

```
CREATE UNIQUE INDEX empno_idx ON emp (empno);
```

To create a non-unique index on the emp table using the ename and deptno columns:

```
CREATE INDEX ename_deptno_idx ON emp (ename,deptno);
```

Views

A view is basically a virtual table (exist as only query definition) which is made up of the rows that the query returns. For example, if we have a query that lists employee names along with department names, we could create a view (say EMP_DEPT_VW)

and allow the user to have access to view rather allowing users to access EMP and DEPT base tables. User will be aware of only EMP_DEPT_VW containing Ename and Dname columns only. Thus we can hide actual base table from user and allow them to retrieve only those columns which are required to them. This is a kind of data security.

Syntax:

CREATE VIEW view_name AS SELECT … query that retrieves data for the view.

Ex: Create view EMP_DEPT_VW containing Ename and their corresponding department names.

```
CREATE VIEW EMP_DEPT_VW AS
SELECT ename employee_name , dept.dname department_name
FROM emp, dept
WHERE dept.deptno = emp.deptno;
```

View created.

We can query a view as if querying a base table.

```
SELECT * FROM emp_dept_vw;
```

```
EMPLOYEE_N DEPARTMENT_NAM
---------- --------------
CLARK      ACCOUNTING
KING       ACCOUNTING
MILLER     ACCOUNTING
JONES      RESEARCH
FORD       RESEARCH
ADAMS      RESEARCH
SMITH      RESEARCH
SCOTT      RESEARCH
WARD       SALES
TURNER     SALES
ALLEN      SALES

EMPLOYEE_N DEPARTMENT_NAM
---------- --------------
JAMES      SALES
BLAKE      SALES
MARTIN     SALES
```

14 rows selected.

Another advantage of using view is –speed up in execution.
Querying a view is faster than base tables involved in a complex query. This is because –view already exits as compiled DDL. If it is a complex query involving many base tables then every time DDL compiler has to compile and execute which takes more time.

Ex:Create view EMP_SAL containing empno, ename and corresponding salary working in department 10.

```
CREATE VIEW EMP_SAL AS SELECT empno,ename,sal FROM EMP
WHERE Deptno=10;
```

DROP VIEW

A view can be removed with the DROP VIEW command.
Ex:

```
DROP VIEW emp_dept_vw;
```

The above statement would remove a view called emp_dept_vw.

View can be created using any kind of query involving single/multiple tables, involving sub queries, involving aggregate functions, GROUP By, HAVING. View can be created using only single table, involving Primary key unique, Notnull constraints but not involving sub queries/aggregate function/GROUP By/HAVING are said to be Updatable Views otherwise not updatable.
Updatable views means, and insertion/updation on view also affects corresponding Base tables.
Find more about Updatable views and experiment with it.

Sequences

A sequence is an oracle object which always generates Unique value and can be used for primary keys/ unique columns. User need not remember what is the last unique value generated and next what value is to be generated for a column defined as primary key or unique column.
Create a sequence, then reference the sequence in your INSERT statement. A sequence is simply an object within the database that returns a number, usually the next in sequence. To create a sequence, use the CREATE SEQUENCE command.
syntax:

```
CREATE SEQUENCE sequence_name
INCREMENT BY n
```

```
START WITH m
MAXVALUE mx
MINVALUE mn;
```

INCREMENT BY- Specifies the interval between consecutive sequence numbers. It cannot be 0(zero). IF it is +ve sequence ascends. If it is –ve then the sequence descends. By default it is 1.

START WITH – Specifies first sequence to be generated.

Use this clause to start ascending at a value greater than its minimum.

Use this clause to start descending at a value less than its maximum.

MAXVALUE – Specifies maximum value a sequence can generate. MAXVALUE must be equal or greater than STRAT WITH and must be greater than MINVALUE.

MINVALUE- Specifies minimum value a sequence can generate. MINVALUE must be less than or equal to START WITH and must be less than MAXVALUE.

Ex: create a sequence called empno_seq01 which starts at 100 and increases by 10 each time.

```
CREATE SEQUENCE empno_seq01
INCREMENT BY 10
START WITH 100;
```

CREATE SEQUENCE empno_seq01 INCREMENT BY 10 START WITH 100;

Referencing a Sequence

You have created a sequence called empno_seq01: we can reference it in two ways, the first being by using the **NEXTVAL** pseudo column to get the next number:

```
SELECT empno_seq01.NEXTVAL FROM dual;
```

This would return 100; if you ran this SQL again it would return 110, and so on.

You can also retrieve a sequence's current value without increasing its value with the **CURRVAL** pseudo column:

```
SELECT empno_seq01.CURRVAL FROM dual;
```

Ex: Insert value to EMPNO column and other columns using

empno_seq01 sequence created above.

```
INSERT INTO Emp VALUES(empno_seq01.currval,'RAVI','MANAGER',
7900, '31-AUG-14',7000,2000,10);
```

1 row created.
Query the EMP table to check the record inserted.

DROP SEQUENCE

A sequence can be removed with the DROP SEQUENCE command. For example:

```
DROP SEQUENCE empno_seq01;
```

The above statement would remove a sequence called empno_seq01.
For Students-Explore more on sequences