|  |  |
| --- | --- |
| Chapter-1 | SQL INTRODUCTION |
| Chapter-2 | ORACLE HISTORY |
| Chapter-3 | TYPES OF SQL STATEMENT |
| Chapter-4 | FUNCTIONS |
| Chapter-5 | JOINS |
| Chapter-6 | CONSTRAINTS |
| Chapter-7 | SUBQUERY |
| Chapter-8 | VIEWS |
| Chapter-9 | DCL (Data Control Language) |
| Chapter-10 | INDEX |
| Chapter-11 | SEQUENCES |
| Chapter-12 | NORMALIZATION |
| Chapter-13 | CONVERSIONS |
| Chapter-14 | SYNONYMS |
| Chapter-15 | TCL(Transaction control language) |
| Chapter-16 | Oracle Server Architecture |
| Chapter-17 | Tables |
| Chapter-18 | Functions |
| Chapter-19 | Procuder |
| Chapter-20 | Trigger |
| Chapter-21 | Cursors |
| Chapter-22 | Analytic Functions |

**Chapter-1**

**SQL INTRODUCTION**

* Oracle is a product from oracle which is used to store data permanently see store device.
* Every organization store same type of data.
* Oracle product of Oracle Corporation.

**DATA:** It is a collection of “**RAW FACTS**”.

**Ex:**  1.Student Marks

2. Customer Names

**INFORMATION:**

When we are processing data are achieving meaning full results it is called as “**INFORMATION**”.

**Ex:** 1.Marks Sheets

2. Information Of Customer

**DATA STORE:**

It is a place where we can store data or information.

1. Books & Papers

2. Flat Files

3. Database

**FLAT FILES:**

Data or Information stored in individual unrelated files these files are also called as “**FLAT FILES**”.

Hard disk

Program-2

Program-1

User-1

File-1

File-2

User-2

**Fig:** Flat file approach

If you want to store data in flat file we are most develop application program in high level language. In flat file approach every application maintains it own file which is separate from another application program.

|  |  |  |
| --- | --- | --- |
| Sno | Sname | class |
|  |  |  |

**Student**

Record application program

|  |  |  |  |
| --- | --- | --- | --- |
| Prof id | Prof name | Time | class |
|  |  |  |  |

**Flat file**

Scheduling application program

**DISADVANTAGE:**

1. Data Retrieval

2. Data Redundancy

3. Data Integrity

4. Data Security

5. Data Indexing

**1. Data Retrieval:**

If you want to retrieval data from flat file we must develop application program in high level language. Whereas if you want to retrieval data from database then we are using SEQUEL(Structural English Query Language).

**2. Data Redundancy:**

Same time we are maintaining multiple copies of the same data in different locations. This data is also called as “**REDUNDANCY DATA**”. In flat file mechanism if we are modifying data in one location. It is not affected in another location. This is called as “**INCONSISTENCY**”. Flat file does not maintain consistence data.

Database automatically maintains consistency data. Because in database every transaction in related having 4 properties. This are also called “**ACID PROPERTIES**”. These properties only maintain consistent data. In database we can also reduce redundancy use in normalization process.

**3. Data Integrity:**

Integrity means to maintain proper data if we want to maintain proper data when we define set of rules this set of rules also called “**BUSINESS RULES**”. In database we are defining business rules using constrains, trigger. Where as in flat file if we want maintain proper data. If we most develop application program in high level language.

**4. Data Security:**

Data stored in flat file cannot be security. Because file does not provides rule based security.

**5. Data Indexing:**

If we want to retrieve data very fastly when database provides indexing mechanism but flat file does not support indexing mechanism.

**What is Database?**

It is organized collection of inter related once data is stored in DB. It can be stored integrated and also simultaneously number of user accesses this data.

* American National Standard Institute (ANSI) as establish three level architecture for DBMS.
* Standards Planning and Requirement Committee (SPARC).

**Architecture contains three levels:**

1. Internal Data Level

2. Conceptual Data Level

3. External Data Level

**DBMS ARCHITECTURE:**

Purchase manager

Customer

**External level**

Item name

Price

Item name

Price

Quality

Logical data independence

**Conceptual level**

Create table item(itemname varchar2(10),

Price number(5),qty number(5));

Physical data independence

Item name type=byte(8) offset=(10);

Price type=byte(10) offset=(15);

Qty type=byte(9) offset=(20);

**Internal level**

All dbms s/w implemented based on data independence.db structure is independent on application program.

**Logical data independence:**

When ever we modifying conceptual level it is not affected external level.

**Physical data independence:**

When ever we modifying internal level it is not affected conceptual level. i.e. if we are moving db from one location to another location then conceptual level does not affected.

Conceptual

External

Internal

EMP

Physical data stored

view-1



view-2

DEPT

**Fig:** ANSI/SPARC Architecture (or) DBMS Architecture

**1. Internal Data Level:**

This level describes how data internally stored in database. Physical representation of the database on the computer database administrators only works in internal level. This level maintain by database administrator (DBA’s).

**2. Conceptual Data Level:**

This level describes organization view of the database. In this level only of database develop creating database and maintain data properly.

**3. External Data Level:**

This level users access or view of the database. Describe that part of database that is relevant to a particular user. In this level database developer create views and giving to the number of members.

**DATA MODEL:**

Data model describe how data layout store in conceptual level.

They are three level model used by organization:

1. Hierarchical Data Model

2. Network Data Model

3. Relational Data Model

**1. Hierarchical Data Model:**

Hierarchical data model organizers data in tree like structure. This hierarchy is also called parent, child hierarchy. This structure employee’s child a record can have repeating information (generally child segments) in this model data is represented by collection of records (record type).

**Relational model**

**Hierarchical model**

|  |
| --- |
| Root |

|  |  |  |
| --- | --- | --- |
| Eid | Ename | Deptname |
| 1000  1001  1002 | Smith  Allen  King | Admin  Hr  sw |

|  |  |  |
| --- | --- | --- |
| 1001 | Allen | Hr |

|  |  |  |
| --- | --- | --- |
| 1000 | Smith | admin |

|  |  |
| --- | --- |
| 2005 | 50000 |

|  |  |
| --- | --- |
| 2005 | 50000 |

|  |  |  |
| --- | --- | --- |
| 1002 | King | S/w |

|  |  |  |
| --- | --- | --- |
| Eid | Ename | Deptname |
| 1000  1001  1002  1002 | 2005  2005  2013  2013 | 50000  50000  70000  60000 |

|  |  |
| --- | --- |
| 2013 | 70000 |

|  |  |
| --- | --- |
| 2013 | 60000 |

In this a record type corresponding to the table in relational model and also recording same as row in a table in relational model. This model is implemented based on ONE-TO-MANY relationship. Based on this restriction number of child on single parent they way in this model more number of time child segment repeated that weighs more redundant data and also this model retrieve data very slowly from the database because this model is search data from root node an wards if want operate hierarchical data model product that we are using procedural language.

In 1960 IBM introduced IMS (information management system) product based on hierarchical model.

**2. Network Data Model:**

In 1970 CODASYL (conference on data system languages) implemented network data model. This model implemented based on MANY-TO-MANY relationships. This model reduces duplicate data. In this model also data stored in collection of record and also record type is same table in relational model and also record are represented also graphs used on this model.

In 1970 IBM implemented IDMS (integrated database management system) products. If want operate network data model product also then we are using procedural language.

|  |  |  |
| --- | --- | --- |
| 1000 | Smith | Admin |

|  |  |  |
| --- | --- | --- |
| 1001 | Allen | Hr |

|  |  |
| --- | --- |
| 2005 | 50000 |

|  |  |
| --- | --- |
| 2013 | 70000 |

|  |  |  |
| --- | --- | --- |
| 1002 | king | sw |

|  |  |
| --- | --- |
| 2013 | 60000 |

**3. Relational Data Model:**

In 1970 E.F.CODD introduced relational data model. This model also implement based on MANY-TO-MANY relationship in this model. We are storing data in two-dimensional table.

Relational model is three components:

1. Collection of objects

2. Set of operations

3. Set of integrity rules

**Chapter-2**

**ORACLE HISTORY**

In 1970 E.F.CODD write a paper “Relational model of data for large stored data banks”.

INGRESS

INGRESS

IBM

Larry Ellison (1977)

(1982)

RSI (1979)

System/R

(Square)

Oracle Corporation

Relational Software, Incorporated (RSI)

Oracle

Oracle

Oracle

DYNAMIC SQL

PL/SQL

**SQL**

**What is Oracle?**

In 1970 E.F CODD described Oracle. Oracle is a database. Oracle database is an Object-Relational Database Management System produced and Marketed by Oracle Corporation. Oracle is a fourth generation Relational Database Management Systems. Oracle is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information.

**ORACLE VERSIONS**

**Oracle 2.0 ----------🡪 1979**

* + - * + First public release
        + Basic sql functionality, Joins

**Oracle 3.0 ---------🡪 1983**

* + - * + Commit, Rollback
        + Virtual c++ (window program) (VC++)
        + Rewritten in ‘c’ language

**Oracle 4.0 ----------🡪 1984**

* Read consistency

**Oracle 5.0 -----------🡪 1985**

* Client server architecture

**Oracle 6.0 -----------🡪 1988**

* + - * + Introduced pl/sql
        + Row level locks

**Oracle 7.0 -----------🡪 1992**

* + - * + Integrity
        + Stored procedures, Stored functions
        + Triggers
        + Truncate table
        + Datatype varchar changed into varchar2
        + View compilation

**Oracle 7.1 ----------🡪 1992**

* + - * + Dynamic SQL
        + ANSI/ISO SQL92

**Oracle 7.2 ----------🡪 1992**

* + - * + Ref cursor
        + Inline views
        + Dbms\_job package

**Oracle 7.3 -----------🡪 1992**

* + - * + Bitmap indexes
        + Utl\_file package

**Oracle 8.0 -----------🡪 1997**

* + - * + Object technology
        + Nested table,Varrays
        + Lobs (large objects)
        + Column increased per a table upto 1000

**Oracle 8i -------------🡪 1999** (I-internet)

* + - * + Analytical functions
        + Materialized views
        + Autonomous transaction
        + Bulk collection clause
        + Case conditional statement
        + Instead of triggers
        + Enable, disable trigger
        + Function based indexes

**Oracle 9i ------------🡪 2001**

* + - * + Renaming a column
        + ANSI joins
        + Flashback query
        + Multi table inserts
        + Merge statement

**Oracle 10g ------------🡪 2003** (G-grid technology)

* + - * + Recycle bin concept
        + Flashback table
        + Wm\_concat()
        + Indices of clause
        + Rename table space
        + Regular expressions

**Oracle 11g ------------🡪 2007**

* + - * + Continuous stmts introduced in pl/sql loops
        + Read only tables
        + Virtual columns
        + Simple\_integer datatype in pl/sql
        + Sequences used in pl/sql without dual table
        + Follows clause in triggers
        + Compound triggers
        + Enable, Disable clause used in trigger specification.
        + Named, Mixed notations used in a sub program executed using select statement.

**SQL (Structured Query Language)**

* Sql is a non-procedural language to operate relational database.
* In 1970 E.F.CODD introduced DSL/ALPHA language and operate relational database.
* IBM System/R team developed simplified version of DSL/ALPHA language this is called “SQUARE”.
* IBM modified DSL SQUARE into SEQUEL.SEQUAL modified into SQL.
* In 1986 ANSI standard are defined in SQL-ANSI SQL.In 1987 ISO statement designed in SQL.
* ANSI/ISO SQL89 ---🡪1989 (Sql I)
* ANSI/ISO SQL92 ---🡪1992 (Sql II)
* ANSI/ISO SQL99 ---🡪1999
* ANSI/ISO SQL03 ---🡪2003 (Sql III)

End User

Oracle db

SQL

QUERY

TOOLS

1.sql plus \*loading data

2.isqlplus \*retrieving data

3.toad \*modifying data

Collection of objects

4.pl/sql developer \*deleting data

5.sql navigator

**Oracle 10g (or) 11g (enterprise edition):**

SCOTT

Username:

TIGER

Password:

Error: Account locked

**Unlocked user:**

\ SYS AS SYSDBA

Username: <┘

SYS

Password: <┘

Sql>Alter user scott account unlock;

Sql>conn scott/tiger; <┘

TIGER

Password:

TIGER

Confirm password:

**To view all table:**

Sql>select \* from tab;

DEPT -------🡪 Master table EMP --------🡪 Child table

**To clear screen and set line proper:**

Sql>cl scr; Sql>set line 100;

**Go to editor:**

Sql>ed <┘

**Get back into sqlplus:**

* + Alt+f+x <┘

Sql> / <┘

**Chapter-3**

**TYPES OF SQL STATEMENT**

**Data Definition Language (DDL):**

- Create

- Alter

- Drop

- Truncate

- Rename (Oracle 9i)

**Data Manipulation Language (DML):**

- Insert

- Update

- Delete

- Merge (Oracle 9i)

**Data Query Language (DQL) or Data Retrieval Language (DRL):**

- Select

**Transaction Control Language (TCL):**

- Commit (Save)

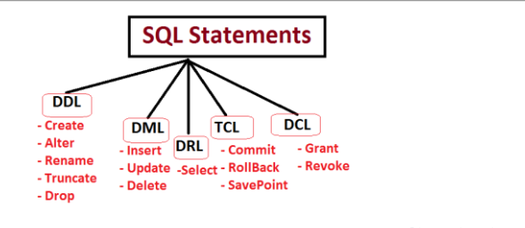
- Rollback (Like undo)

- Savepoint

**Data Control Language (DCL):**

- Grant (Giving Permissions)

- Revoke (Cancel permissions)



**DATA TYPE**

Data type identifier type of data within a table column.

1. Number (P,S)

2. Char -------------🡪 Varchar2 (maxsize)

3. Date

**1. Number:**

It is used to store fixed, floating point number.

**Syntax:**  columnname number (P, S) P-precision S-scale

Maximum limit of precision is upto 38.

**2. Char:**

It is used to store fixed length ALPHA numeric data in bytes. Maximum limit is 2000bytes.By default char data type having 1 bytes.

**Syntax:** columnname char (size)

Long (2 GB) clob (4 GB) blob (image)

**Varchar2 (maximum size):**

It is used to store variable length alpha numeric data maxsize is 4000 bytes.

**Syntax:** columnname varchar2(maxsize)

Sql>select dump(name) from test;

**3. Date:**

It is used to store date in oracle data format. By default in oracle date format “DD-MON-YY”.

**Syntax:** columnname date;

**DATA DEFINITION LANGUAGE (DDL):**

These commands are used to define structure of the table.

- Create

- Alter

- Drop

- Truncate

- Rename

**Create:**

This command is used to create database objects like tables, views, procedures, indexes, sequences………………………

**Syntax:** Create table tablename(col1 datatype(size),col2 datatype(size),…………);

**Ex:** Sql>create table india(sno number(5),name varchar2(10));

**To view structure of the table:**

Sql>desc tablename;

**Ex:** sql>desc india;

**Alter:**

It is used to change existing table structure.

**Alter**

Add Modify Drop

**Add:** It is used to add number of columns into the existing table.

**Syntax:** alter table tablename add(col1 datatype(size),col2 datatype(size),……….);

**Ex:** sql>alter table india add sal number(10);

**Modify:** It is used to change column datatype (or) column datatype size only.

**Syntax:** alter table tablename modify(col1datatype(size),col2 datatype(size),……);

**Ex:** sql>alter table india modify sno date;

**Drop:** It is used to remove column from the table.

Method-1:

If we want drop a single column at a time without using parenthesis then we are using following syntax.

**Syntax:** alter table tablename drop **column** columnname;

Method-2:

If we want drop a single (or) multiple column with using parentheses then we are using following syntax.

**Syntax:** alter table india drop(name);

**NOTE:**  we cannot drop all columns in a table.

**Drop:**  It is used to remove database **object** from the table.

**Syntax:** drop objecttype objectname;

Drop table tablename;

**Ex:** sql>drop table india;

Sql>drop procedure procedurename;

**Dropping a table:**

**Before oracle 10g**

Sql>drop table tablename;

Oracle db

Table

Table

**After oracle 10g**

Sql>drop table tablename;

Oracle db

Recyclebin

Table

Permanently Remove

**Get it back from recycle bin:**

**Syntax:**

Flashback table tablename to before drop;

**To drop permanently:**

Sql>drop table tablename purge;

**Get it back from recycle bin:**

Sql>flashback table india to before drop;

**To drop permanently:**

Sql>drop table india purge;

**Testing:**

Sql>flashback table india to before drop;

Error: object not in recycle bin.

**Recycle bin:**

Recycle bin is a read only which stores dropped store. Whenever we are installation oracle server then automatically so many read only table there. These read only tables are also called as Data Dictionary.

Oracle db

Sql>create table india(sno number(5));

Object\_name

Original\_name

Recycle bin

Sql>drop table india;

**Testing:**

India

Sql>desc recyclebin;

Sql>select original\_name from recyclebin;

**Purge:**

We can also drop object from recycle bin purge command.

**To drop single table from recycle bin:**

**Syntax:** purge table tablename;

**To drop all table from recycle bin:**

**Syntax:** purge recyclebin;

**Truncate:** This command is used to delete all rows permanently from the table.

**Syntax:** Truncate table tablename;

Sql>create table india as select \* from emp;

Sql>select \* from india;

Sql>truncate table india;

**Testing:** sql>select \* from india;

Sql>desc india;

**Rename:** It is used to renaming a table and renaming a column also.

**Renaming a table:**

Sql>rename oldtablename to newtablename;

Sql>rename india to andhra;

**Renaming a column (Oracle 9i):**

**Syntax:** alter table tablename rename column oldcolname to newcolname;

Sql>alter table emp rename column empno to sno;

Sql>select \* from emp;

**NOTE:** In all db system by default all DDL command is automatically commit (save).

**DATA MANIPULATION LANGUAGE (DML):**

DML command is used manipulation data in a table.

- insert

- update

- delete

- merge (Oracle 9i)

**Insert:**

This command is used insert data into a table.

Method-1:

**Syntax:** insert into tablename values(val1,val2,………….);

Sql>insert into india values(1,’SIMHA’);

Sql>insert into india values(2,’REDDY’);

Sql>select \* from india;

Method-2 (using substitutional operator (&)):

**Syntax:** insert into tablename values(&col1,&col2,………………);

Sql>insert into india values(&sno,’&name’);

Enter value for sno: 3

Enter value for name: abc

Sql>/

Enter value for sno: 4

Enter value for name: xyz

Method-3 (skipping column):

**Syntax:** insert into tablename(col1,col2,………..) values(val1,val2,……….)

Sql>insert into india(name)value(‘zzz’);

**Update:** It is used to modify (change) data in a table.

**Syntax:** update tablename set columnname=newvalue **where** columnname= oldvalue;

Sql>update emp set sal=3000 where ename=’SMITH’;

Sql>alter table india add address varchar2(10);

Sql>update emp set sal=NULL where sal=3000;

Remove the sal 3000

**NOTE:** If we want to insert particular cell values also then we are using update stmts.

**Delete:**  It is used to delete all rows (or) particular row from the table.

**Syntax:** Delete from tablename; (To delete all rows)

Delete from tablename where condition; (To delete particular rows)

Sql>create table india **as** select \* from emp;

Sql>delete from india;

**Get it back data:**

Sql>rollback;

Sql>select \* from india;

**Difference b/w Delete and Truncate:**

We are using delete from tablename. Then automatically total data delete from table and also deleted a data automatically store in a buffer. We can get it back data using rollback (without commit).

When we are using truncate from tablename all row delete permanently. It cannot get it back this data. Because truncate is a DDL command automatically commits.

**DATA RETRIEVAL LANGUAGE (DRL) (OR) DATA QUERY LANGUAGE (DQL):**

This command is used to retrieval data from the table.

**Syntax:** select col1,col2,……….

From tablename

Where condition

Group by columnname

Having condition

Order by [asc/desc];

1.select all cols & all rows.

2.select all cols & particular rows.

3.select particular cols & all rows.

4.select particular cols & particular rows.

**Creating a new table from existing table (or) Coping a table from another table:**

**Syntax:** create table newtablename **as** select \* from existingtablename;

Sql>create table india as select \* from emp;

**Coping table without coping a table:**

**Syntax:** create table newtablename **as** select \* from tablename where falsecondition;

Sql>create table kadapa **as** select \* from emp where **1=2**;

Sql>select \* from kadapa;

No rows selected.

Sql>desc kadapa;

**NOTE:** Where we are copping a table from another table constraints are not copied.

**OPERATORS USED IN A SELECT STATEMENT:**

1. Arithmetic operators (\*, +,-,/)

2. Comparison (or) Relational operator (=, <, <=, >, >=, <> (or) != )

Where

Clause

3. Logical operator (AND, OR, NOT)

4. Special operator

Arithmetic operator is used in number, date datatype columns.

**Q) Write a query to display ename, sal, annsal from emp table?**

Ans: sql>select ename, sal, sal\*12 “annsal” from emp;

**Q) Write a query to display the emp except job has clerk from emp table?**

Ans: sql>select \* from emp where job<>’CLERK’;

**Q) Write a query to display EMP’s we are getting more than 2000 salary from emp table?**

Ans: sql>select \* from emp where sal>2000;

Sql>select \* from emp where job=’CLERK’ **AND** sal>1100;

Sql>select \* from emp where job=’CLERK’ **OR** sal>2000;

**Q) Write query to display the emp who are belongs to 20,30,50,70 deptno from emp table?**

Ans: sql>select \* from emp where deptno=20 or deptno=30 or deptno=50 or deptno=90;

Sql>select \* from emp where sal>2000 and sal<5000;

**SPECIAL OPERATOR:**

In not null

Between not between

Is null is not null

Like not like

**In:**

This operator is used to pick the values one by one from list of values. Generally IN operator is used in place of OR operator. Because IN operator performance very high compare to OR operator.

**Syntax:** select \* from tablename where columnname in (list of value);

**NOTE:** Generally in all database system we are using multiple rows subqueries then we must use IN operator.

Sql>select \* from emp where deptno in(20,30,50,70,90);

Sql>select \* from emp where deptno not in(20,30,50,70,90);

**Ex:** sql>select \* from emp where ename in(‘SMITH’,’KING’);

Sql>select \* from emp where not in(20,30,NULL);

**NOTE:** NOT IN operator does not work with null values in all database systems.

**null:**

NULL is an undefined, unknown, unavailable value it is not same as zero. Any arithmetic operation performed with null value again it will becomes null.

Null+50-----------------🡪Null

**Q) Write a query to display ename,sal,comm,sal+comm of the emp SMITH from emp table?**

Ans: sql>select ename,sal,comm,sal+comm From emp where ename=’SMITH’;

**Ename sal comm sal+comm**

SMITH 1000 --------- --------------

To overcome this problem oracle introduced NVL() function.

**nvl():**

It is a predefined function which is used to replace/substitute userdefined value (0) in place of null.

**Syntax:** nvl(exp-1,exp-2)

-here exp-1,exp-2 must belongs to same datatype.

-exp-1 is null then it returns exp-2 otherwise it returns exp-1.

**Ex:** nvl(null,10) ----------🡪10

Nvl(40,10) -----------🡪40

Sql>select ename,sal,comm,sal+nvl(comm,0) from emp where ename=’SMITH’;

**Ename sal comm sal+comm =>**sal+nvl(comm,0)

SMITH 1000 -------- 1000 =>1300+nvl(null,0)

=>1300+0

=>1300

**Between:**

This operator is used to retrieve range of values this operator is also called as BETWEEN ………AND operator.

**Syntax:** select \* from tablename where columnname between lowvalue AND highvalue;

Sql>select \* from emp where sal between 2000 and 5000;

**Is null, is not null:**

These two operator are used in where clause only. Thisused to whether a column having null value (or) not.

**Syntax:** select \* from tablename where columnname is null;

**Syntax:** select \* from tablename where columnname is not null;

**Q) Write a query to display emp who are not getting commission from emp table?**

Ans: sql>select \* from emp where comm is null;

**Q) Write a query display emp does not have manager from table?**

Ans: sql>select \* from emp where mgr is null;

Sql>select \* from emp where comm is not null;

**Like:** This operator is used to search data based on character data. This operator performance is high compare to searching function. Along with LIKE operator we are using 2 special characters.

**LIKE**

**%** **\_** (underscore)

(string or group of character) Single character

**Syntax:** select \* from tablename where columnname **like** ‘character’;

**Q) Write a query to display emp whose ename start with ‘M’ from emp table using like operator?**

Ans: sql>select \* from emp where ename like ‘M%’;

**Q) Write a query to display the emp whose joining in the ‘month dec’ emp table using like operator?**

Ans: sql>select \* from emp where hiredate like ‘%dec%’;

**Q) Write a query to display the emp whose joining in the ‘year 81’ emp table using like operator?**

Ans: sql>select \* from emp where hiredate like ‘%81’;

**Q) Write a query to display the emp whose ename second letter would ‘L’ from emp table using like?**

Ans: sql>select \* from emp where ename like ‘\_L%’;

**If third letter:**

Sql>select \* from emp where ename like ‘\_\_L%’;

**Concatenation operator (||):**

This operator is used to concatenation data column values with literal strings.

Sql>select ‘my employee names are’||ename from emp;

Sql>select ename||’ ‘||sal from emp;

**Chapter-4**

**FUNCTIONS**

Functions are used to solve particular talk in a problem and also functions must return a value oracle having two types of functions.

1. Predefined Functions

2. Userdefined Functions

**1. Predefined functions:**

1. Number Functions

2. Character Functions

3. Date Functions

4. Group of functions (or) Aggregate functions

**Number Functions:** These functions operator an number data.

**1. abs():** It is used to convert negative value into positive value.

**Ex:** sql>select abs(-50) from dual;

Sql>select abs(sal-comm) from emp where comm is not null;

Sql>select ename,sal,comm,abs(sal-comm) from emp where comm is not null;

**Dual:**

Dual is a virtual table which contains one row and one column by default dual table column data type is varchar2.

Sql>select \* from dual;

Sql>desc dual;

Generally dual table is used test predefined userdefined functions functionality. Dual table is also used to generate sequences values and also used to mathematical operations.

**Ex:** sql>select 50+70 from dual;

**2.mod(m,n):**

It returns remaining after M divided by N.

Sql>select mod(30,2) from dual; **o/p:** 0

Sql>select mod(sal,3) from emp;

**3.round(m,n):** It rounds given floating point value number ‘M’ based on ‘N’.

Sql>select round(1.8) from dual; **o/p:** 2

Sql>select round(1.23456,3) from dual; 1.234/56 56 is more than 50%

**o/p:** 1.235 1.235 hence 1 is added.

Sql>select round(1285.786,-1) from dual; 1280 5 replace with 0

**o/p:** 1290 1290 50% above then 1 is added

sql>select ename ,sal,sal/22,round(sal/22),round(sal/22,1),round(sal/22,2) from emp;

**NOTE:**

Round always checks remaining number it means remaining number is above 50% then 1 is added to the rounded number.

**4.trunc(m,n):** It truncate given floating value ‘M’ based on ’N’.

Sql>select trunc(1.9) from dual; **o/p:** 1

Sql>select trunc(1.23456,3) from dual; **o/p:** 1.234

**5.ceil(),floor():**

Ceil() returns nearest greatest integer where are floor() returns nearest lowest integer.

Sql>select ceil(1.4) from dual; **o/p:** 2

Sql>select floor(1.9) from dual; **o/p:** 1

**6.greatest(exp1,exp2………),least(exp1,exp2…………….):**

Greatest() returns maximum value within given expression where as least returns minimum value among given expression.

Sql>select greatest(3,5,9) from dual; **o/p:** 9

Sql>select ename,sal,comm,greatest(sal,comm) from emp where comm is not null;

Sql>select greatest(sal) from emp; **o/p:** 800,1220, 5000,3000 1350…………..

We not show maximum value so there it is the problem. To overcome this value we use max(sal).

**Max():**  This function comparing single column only.

Sql>select max(sal) from emp;

**Character Function:**

**1.upper():**

It is used to convert a string into uppercase or column value into uppercase.

Sql>select upper(‘abc’) from dual;

Sql>select upper(ename) from emp;

**2.lower():**

It is used to convert string into lowercase or column value into lowercase.

Sql>select lower(ename) from emp;

Sql>update emp set ename=lower(ename);

**3.initcap():**

It returns initial letters capital and all remaining letters are small.

Sql>select initcap(‘abcdef’) from dual; **o/p:** Abcdef

Sql>select initcap(‘ab cd ef’) from dual; **o/p:** Ab Cd Ef

Sql>select initcap(ename) from emp; **o/p:** Smith,Allen,Blake,King…………….

**4.length():**

This function always returns number datatype it returns total length of the string including spaces.

Sql>select length(‘abcdef’) from dual; **o/p:** 6

Sql>select length(‘abc def’) from dual; **o/p:** 7

**5.substr():**

It will extract position of the string within the given string based on last two parameters.

Sql>select substr(‘ABCDEF’,2,3) from dual; **o/p:** BCD

Sql>select substr(‘ABCDEF’,-2,3) from dual; **o/p:** EF

**0Syntax:** substr(columnname (or) ‘stringname’,string position, number of char from position);

**Q) Write a query to display the emp’s whose ename second letter would ‘LA’ from emp table using substr() function?**

Ans: sql>select \* from emp where substr(ename,2,2)=’LA’; **o/p:** BLAKE,CLARK

**NOTE:**

In all database systems we are not allowed to use group function in where clause but we are allowed to use number, character, date function in where clauses.

**6.instr():**

It returns number datatype this function returns position of the delimiter (or) position of the character (or) position of the string within given string.

Instr(columnname (or) ‘stringname’,’string’,searching position,no.of occurace from position)

Sql>select instr(‘ABCD’,’C’) from dual; **o/p:** 3

Sql>select instr(‘ABCDEFGHICDJKMLCDHJF’,’CD’,-5,2) from dual; **o/p:** 10

Sql> select instr(‘ABCDEFGHICDJKMLCDHJF’,’CD’,-6,2) from dual; **o/p:** 3

**NOTE:** Always instr() returns position based on last two parameter but oracle server always counts number of characters left side first position on words.

**7.Lpad():**

It will fills specified character on the left side of the given string. Here always second parameters returns total length of the string this parameter must be a number.

**Syntax:** Lpad(columnname (or) ‘stringname’,total length of the string,’filled char);

**Ex:** sql>select lpad(‘ABCD’,10,’&’) from dual; **o/p:** &&&&&&ABCD

**8.Rpad():**

Sql>select rpad(‘ABCD’,10,’&’) from dual; **o/p:** ABCD&&&&&&

Sql>select rpad(ename,20,’-‘)||sal from emp;

**Ans:** **rpad(ename,20,’-‘)||sal**

smith----------------------------------------1800

allen------------------------------------------1250

king-------------------------------------------5000

**9.Ltrim():**

It is used to remove left side specified characters.

**Syntax:** Ltrim(columnname (or) ‘stringname’,{set of characters});

Sql>select Ltrim(‘SSMISSTHSS’,’S’) from dual; **o/p:** MISSTHSS

Sql>select job,Ltrim(job,’lsm’) from emp;

**Ans: job ltrim(job,’CSM’)**

CLERK LERK

SALESMAN ALESMAN

MANAGER ANAGER

**10.Rtrim():**

Sql>select Rtrim(‘SSMISSTHSS’,’S’) from dual; **o/p:** SSMISSTH

Sql>select job,Rtrim(job,’KNR’) from emp;

**Ans: job ltrim(job,’KNR’)**

CLERK CLE

SALESMAN SALESMA

MANAGER MANAGE

**11.Trim():** (Oracle 8i introduced)

It is used to remove left and right side specified character and also leading and trailing spaces.

Syntax: trim(‘character’ from columnname (or) ‘stringname’)

Sql>select trim(‘S’ from ‘SSMISSTHSS’) from dual; **o/p:** MISSTH

Sql>select trim(leading ‘S’ from ‘SSMISSTHSS’) from dual; **o/p:** MISSTHSS

Sql>select trim(trailing ‘S’ from ‘SSMISSTHSS’) from dual; **o/p:** SSMISSTH

**NOTE:** This function also used remove first letter last spaces. Trim set should have only one character.

Sql>select length(trim(‘abcd’)) from dual;

**Ans:** **4**

**12.Translate(), Replace():**

Translate() is used to replaces character by character where as Replace() is used to replaces character by string (or) string by string.

**Syntax:** TRANSLATE( string1, string\_to\_replace, replacement\_string )

Sql>select translate(‘ABCD’,’ACBD’,’@$#&’) from dual; **o/p:** @#$&

Sql>select translate(‘ABCDEF’,’FEDCBA’,’123456’) from dual; **o/p:** 654321

**Ex:** Sql>select replace(‘A B C’,’ ‘,’ ORACLE ’) from dual;

**Ans: A** ORACLE **B** ORACLE **C**

Sql>select job,replace(job,’SALESMAN’,’MARKETING’) from emp;

**Ans: job replace(‘salesman’,’marketing’)**

Clerk clerk

Salesman marketing

King king

Salesman marketing

Sql>select replace(‘SSMISSTHSS’,’S’) from dual; **o/p:** MITH

**Q) Write a query to count number of repeating character from a string?**

Sql>select length(‘ABBBC’)-length(replace(‘ABBBC’,’B’)) from dual; **o/p:** 3

**Date Functions:**

In oracle by default date format is ‘DD-MON-YY’. Oracle having following date functions.

1.sysdate

2.add\_months()

3.last\_day()

4.next\_day()

5.months\_between()

**1.sysdate:** It returns current system date in oracle date format.

Sql>select sysdate from dual; **o/p:** 31-jul-14

**2.add\_months():**

It is used to add (or) subtract number of months to specified date based on second parameter.

**Syntax:** add\_months(date,number)

Sql>select add\_months(sysdate,1) from dual; **o/p:** 31-AUG-14

Sql>select add\_months(sysdate,-1) from dual; **o/p:** 30-JUN-14

**3.last\_day():** It returns last day of the specified months.

**Syntax:** last\_day(data)

Sql>select last\_day(sysdate) from dual; **o/p:** 31-JUL-14

**4.next\_day():**

It returns next day occurrence date from the specified date based on second parameter.

Sql>select next\_day(sysdate,’SUN’) from dual; **o/p:** 03-AUG-14

**5.mon\_between():**

This function returns number datatype it return number of months between two specified dates. It date1>date2 then this function returns negative value.

Sql>select months\_between(sysdate,hiredate) from emp;

**months\_between(sysdate,hiredate)**

403.481436

401.386775

401.328987

Sql>select ename,round(months\_between(sysdate,hiredate)) from emp;

**Ename** **round(months\_between(sysdate,hiredate))**

SMITH 403

ALLEN 401

KING 409

**Date Arithmetic:**

1.date+number

2.date-number

3.date1+date2 (not possible)

4.date1-date2

Sql>select sysdate+1 from dual; **o/p:** 01-AUG-14

Sql>select sysdate-1 from dual; **o/p:** 30-JUL-14

Sql>select sysdate-sysdate from dual; **o/p:** 0

**DATE CONVERSION FUNCTION:**

DD YY DAY

D YYYY DY

DDD HH Number MONTH Character

MM MI MON

SS YEAR

1.TO\_CHAR()

2.TO\_DATE()

**1.TO\_CHAR():**

It is used to convert date type into char type i.e it is used to convert date type into date string.

Sql>select to\_char(sysdate,’DD/MM/YY’) from dual;

**o/p:** 31/07/14

day-------🡪thursday

dy---------🡪thu

DY---------🡪THU

**NOTE:** to\_char() character is an case sensitive .

Sql>select to\_char(sysdate,’DAY’) from dual;

**o/p:** THURSDAY

Sql>select to\_char(sysdate,’DD’) from dual; **o/p:** 31

D----🡪6 (Day of the week sun-1,mon-2,tue-3…………)

DDD------🡪212 (Day of the year) DD--------🡪31 (Day of the month)

Sql>select to\_char(sysdate,’DDSPTH’) from dual; **o/p:** THIRTY-FIRST

DDSPTH-----🡪SP-spellout (thirty-first) DDTH--🡪(31ST)

Sql>select to\_char(sysdate,’HH:MI:SS’) from dual; **o/p:** 01:20:45

HH24:MI:SS----------🡪13:20:45

--🡪by default 12 hours display

Sql>select to\_char(’23-jun-05’,’DD/MONTH/YY’) from dual;

ERROR: Invalid Number

**NOTE:** Whenever we are using to\_char always first parameter must be date type. Otherwise oracle server returns an error.

Sql>select to\_char(sysdate,’DD/MONTH/YY’) from dual; **o/p:** 31/JULY/14

**2.TO\_DATE():**

It is used to convert date string into date type (Oracle date format).

Sql>select to\_date(‘15/june/04’) from dual;

**o/p:**  15-JUN-04 DD-MON-YY

Sql>select to\_date(‘15/06/04’) from dual; num-char-num

**ERROR:** not a valid month 15/06/04

Sql>select to\_date(‘15/06/04’,’DD/MM/YY’) from dual; **o/p:** 15-JUN-04

**NOTE:**

Whenever we are using to\_date passed parameter return values match with default date format return date type otherwise oracle side returns error. To overcame this problem use a second parameter as same as first parameter format. Then oracle server only automatically converts date string into date type.

Sql>select to\_char(to\_date(’15-JUN-05’),’DAY/FMMONTH/YY’) from dual;

**o/p:** WEDNESDAY/JUNE/05

Sql>select to\_date(’08-jul-05’)+6 from dual; **o/p:** 14-JUL-05

Sql>select to\_date(’08-07-05’,’DD-MM-YY’)+6 from dual; **o/p:** 14-JUL-05

**NOTE:** When to\_char function if we are specifying full length format then oracle server returns gaps if we passed string return values occupy less size than the default date format return values to overcome this problem we must use fill mode(FM).

**Q) Write a query to display emp who are joining in month DECEMBER from emp table using to\_char function?**

Sql>select \* from emp where to\_char(hiredate,’MM’)=’12’;

MON=DEC MM=12 (working) MONTH=DECEMBER (not working)

**Q) Write a query to display the emp who are joining in the year ‘81’ from emp table using to\_char function?**

Sql>select \* from emp where to\_char(hiredate,’yy’)=’81’;

Sql>select hiredate,to\_char(hiredate,’yyyy’) from emp;

**NOTE:**

Whenever we are passing date string into oracle date functions then oracle server internally convert date string into date type that why we are not allowed to\_date explicitly. In this case passed parameter must be default date format otherwise oracle server returns an error.

Sql>select last\_day(’12-06-05’) from dual;

**ERROR:** not a valid month

Sql>select last\_day(to\_date(’12-06-05’,’DD-MM-YY’) from dual; **o/p:** 30-JUN-05

**Round,Truncate function used in date:**

Whenever we are using round,truncate function date part can be changed based on time portion. Whenever we are using round function oracle server internally check time portion≥12 or noon. It time portion≥12 then one day added to the given date and also automatically time portion set to zero.

Wherever we are using truncate also oracle server internally time portion set zero.

**TESTING:**

Sql> select to\_char(round(sysdate),’DD/MM/YY HH24:MI:SS’) from dual;

01/08/2014 00:00:00

Sql> select to\_char(trunc(sysdate),’DD/MM/YYYY HH24:MI:SS’) from dual;

31/07/2014 00:00:00

**Q) Write a query to display the emp who are joining today from emp table?**

Sql>insert into emp(empno,ename,hiredate)values(1,’narasimha’,sysdate);

Sql>select \* from emp where hiredate=sysdate;

No Rows Selected

Sql>select \* from emp where trunc(hiredate)=trunc(sysdate);

**GROUP FUNCTION (or) AGGREGETING:**

All this group function operator over group of data and return a single value.

1. max()

2. min()

3. avg()

4. sum()

5. count(\*)

6. count(columnname)

**1. max():** It returns maximum values within a column.

Sql>select max(sal) from emp; **Ans:** 5100

Sql>select max(hiredate) from emp; **Ans:** 23-MAY-87

Sql>select min(ename) from emp; **Ans:** WARD

**NOTE:** We are not allowed to use group function in where clause.

Sql>select \* from emp where sal=max(sal);

**2. min():** Sql>select min(sal) from emp; **Ans:** 850

Sql>select min(hiredate) from emp; **Ans:** 17-DEC-80

**3. avg():** sql>select avg(sal) from emp; **Ans:** 2188.48114

Sql>select avg(comm) from emp; **Ans:** 550

**NOTE:** In oracle by default all group function ignores null values except count(\*) function.

Sql>select avg(nvl(comm)) from emp; **Ans:** 157.142857

**4. Sum():** It returns total within a column.

Sql>select sum(sal) from emp; **Ans:** 29025

Sql>select sum(sal) from emp where deptno=10; **Ans:** 8750

**5. Count (\*):** It count number of row in a table.

Sql>select count(\*) from emp; **Ans:** 14

**6. Count (columnname):** In count number of not null values within a column.

Sql>select count(comm) from emp; **Ans:** 4

Sql>select count(distinct(deptno) from emp; **Ans:** 3

**GROUP BY:** This clause is used to divide similar data items into set of logical group. This clause is used to select statement.

**Syntax:** select columnname from tablename group by columnname;

**Q) Write a query to display no.of emp’s department wise from emp table using group by?**

Sql>select deptno,count(\*) from emp group by deptno;

**DEPTNO** **COUNT(\*)**

10 3

20 5

30 6

**Q) Write a query to display number of emp’s job wise from emp table using group by?**

Sql>select job,count(\*) from emp group by job;

**JOB COUNT(\*)**

CLERK 4

MANAGER 3

ANALYST 2

SALESMAN 4

PRESIDENT 1

**Q) Write a query to display no.of emp’s dept wise max(),min() of sal from emp table using group by?**

Sql>select deptno,max(sal),min(sal) from emp group by deptno;

**DEPTNO MAX(SAL) MIN(SAL)**

10 5000 1300

20 3000 800

30 2850 950

**NOTE:** We can also use group by clause without using group function.

Sql>select deptno from emp group by deptno; **Ans:** 10,20,30

**RULES:**

Sql>select deptno,max(sal),ename from emp group by deptno;

**ERROR:** Not a GROUP FUNCTION Expression

Other than group function columns specification after select those all columns must be use after group by. Otherwise oracle server returns ERROR: not a group function expression.

Sql>select deptno,max(sal),ename from emp group by deptno,ename;

Sql>select deptno,job from emp group by deptno,job;

**Ex:** sql>select deptno from emp group by deptno,ename;

**NOTE:**

Whenever we are using group function with another column with using after select then database server returns an error. To overcome this problems column use group by clause.

**Step-1:** sql>select sum(sal) from emp; **Ans:** 29025

**Step-2:** sql>select job,sum(sal) from emp;

**ERROR:** Not a SINGLE-GROUP Group Function

**Solution:** sql>select job,sum(sal) from emp group by job;

**JOB SUM(SAL)**

CLERK 4150

SALESMAN 5600

MANAGER 8275

ANALYST 6000

PRESIDENT 5000

**Q) Write a query to display deptno,count(\*) and also those dept having more than 4 emp’s from emp table using group by clause?**

Sql>select deptno,count(\*) from emp group by deptno where count(\*)>4;

**ERROR:** Sql Command not properly

Sql> select deptno,count(\*) from emp group by deptno having count(\*)>4;

Ans: Deptno- 20 30 count(\*)- 5 6

**Having:**

In all database systems after GROUP BY clause we are not allowed to WHERE clause. In place of this ANSI/ISO sql introduced using HAVING clause. Generally if you want to restrict the rows in a table then we are using WHERE clause. Where as if we want to restrict the groups then after we are using HAVING clause. Generally we are not allowed to use GROUP function in WHERE clause. Where has in HAVING clause we can also use GROUP function.

Sql>select deptno,sum(sal) from emp group by deptno having sum(sal)>10000;

**DEPTNO SUM(SAL)**

20 10875

**Q) Write a query to display year, number of emp’s in that year which more than one emp was hide from emp table using group by clause?**

Sql>select to\_char(hiredate,’YYYY’) “YEAR”, count(\*) from emp

group by to\_char(hiredate,’YYYY’)having count(\*)>1;

**YEAR COUNT(\*)**

1982 2

1981 10

**ORDER BY:**

This clause is used to arrange data in shorting order along with ORDER BY clause oracle provided two keywords ASC/DESC. By default ORDER BY clause having ASC order.

**ORDER BY**

ASC DESC

**Syntax:** select \* from tablename order by columnname[asc/desc];

Sql>select sal from emp order by sal asc; [optional]

Sql>select sal from emp order by sal desc; [mandatory]

**NOTE:** we can also use column position on ORDER BY clause to improve performance.

Sql>select \* from emp order by 5 desc; [columnnumber]

**NOTE:**

We can also use more than one column ORDER BY clause. In this based ORDER BY clause arrange data in first column then value are same in first column those column second column.

Sql>select ename,deptno,hiredate from emp order by deptno asc,hiredate desc;

Sql>select deptno,count(\*) from emp where sal>1000 group by deptno having count(\*)>3 order by deptno desc;

**DEPTNO COUNT(\*)**

30 5

20 4

**Rollup,Cube:**

Oracle 8i introduced rollup,cube. These clauses are used along with GROUP BY clause only. This clauses are used to calculate sub total,grand total **automatically**.

**Syntax:** select col1,col2........ from emp tablename group by rollup(col1,col2……);

Select col1,col2........ from emp tablename group by cube(col1,col2……);

Sql>select deptno,job,sum(sal),count(\*) from emp group by rollup(deptno,job);

Sql> select deptno,job,sum(sal),count(\*) from emp group by cube(deptno,job);

ROLLUP is used to calculate subtotal based on a single column. If we want to calculate subtotal.Grandtotal based on number of columns than we are using CUBE.

**Chapter-5**

**JOINS**

Joins are used to retrieve data from more than one table. If we are joining ‘N’ tables then we are using N-1 joining condition oracle server having following types of joining. This 4 joins are also called as 8i joins.

1. Equi join (or) Inner join

2. non-equi join

3. Self join

4. Outer join

**ANSI joins (or) 9i joins:**

1. Inner join

2. Left outer join

3. Right outer join

4. Full outer join

5. Natural outer join

We can also retrieve data from multiple tables without using joins. In this case server internally uses cross join. Cross joins must default data. Cross join internal uses catigear product. If we want to display accurate data from multiple tables then we must use explicit join.

Sql>select ename,sal,dname,loc from emp,dept;

**1. Equi join (or) Inner join:**

Based on equality condition we are retrieving data from multiple tables. Hare joining conditional column must belongs to same datatype. When table having common columnname then only we are using this join.

**Syntax:** select col1,col2……………..

From tablename1,tablename2

Where tabblename1.commom columnname=tablename2.common columnname;

Sql>select ename,sal,deptno,daname,loc

From emp,dept

Where emp.deptno=dept.deptno;

**ERROR:** column ambiguously defined

Sql>select ename,sal,dept.deptno,daname,loc

From emp,dept

Where emp.deptno=dept.deptno;

**NOTE:**

To avoid feature ambiguously generally we are specified columnname along with tablename using DOT operator after SELECT.

**Using Alias name:**

We can also create deferent names for the table in FROM clause of the joining condition.

**Syntax:** from tablename1 aliasname1,tablename2 aliasname2;

Sql>select ename,sal,d.deptno,dname,loc

From emp e,dept d

Where e.deptno=d.deptno;

**NOTE:**

Equi join always return **matching rows** only (here deptno 40 does not display if we are using deptno also).

**Q) Write a query to display we are working Chicago from emp,dept tables using joins?**

Sql>select ename,loc from emp,dept

Where emp.deptno=dept.deptno;

**Only Chicago display:**

Sql>select ename,loc from emp,dept

Where emp.deptno=dept.deptno

And loc=’CHICAGO’;

**NOTE:**

If we want to retrieve the rows after joining condition then we are using AND operator in 8i joins where has in 9i join we are using either ‘AND’ (or) ‘WHERE’.

**Q) Write a query display dname,sum(sal) from emp,dept tables using equi joins?**

**Dname**  **sum(sal)**

ACCOUNTING 8750

RESEARCH 10875

SALES 9400

Sql>select dname,sum(sal) from emp,dept

Where emp.deptno=dept.deptno group by dname;

Sql>select d.deptno,dname,sum(sal) from emp e,dept d

Where e.deptno=d.deptno group by d.deptno,dname;

**DEPTNO** **DNAME** **SUM(SAL)**

10 ACCOUNTING 8750

20 RESEARCH 10875

30 SALES 9400

SQL> select dname,sum(sal) from emp,dept

where emp.deptno=dept.deptno group by dname having sum(sal)>10000;

**DNAME** **SUM(SAL)**

RESEARCH 10875

SQL> select dname,sum(sal) from emp,dept

where emp.deptno=dept.deptno group by dname having count(\*)>3;

**DNAME** **SUM(SAL)**

RESEARCH 10875

SALES 9400

SQL> select dname,sum(sal) from emp,dept

where emp.deptno=dept.deptno group by rollup(dname);

**DNAME** **SUM(SAL)**

ACCOUNTING 8750

RESEARCH 10875

SALES 9400

**29025**

**2. Non-equi join:**

Based on other than equality condition (<>,>,>=,<,<=,in,between……….) we can retrieving multiple tables. This join is also called as “between……and” join. Generally table does not have same common columns then only using this join.

SQL> select ename,sal,losal,hisal from emp,salgrade where sal between losal and hisal;

SQL> select ename,sal,losal,hisal from emp,salgrade where sal>=losal and sal<=hisal;

**3. Self join:**

Join in a table to itself is called as “self join”. Here joining conditional columns must belong to same datatype. Generally if we want to compare different column in a single table (or) if we want to single column values then we are using self join.

**Syntax:** from tablename aliasname1,tablename aliasname2

Whenever we are using self join we must create an alias name in from clause this alias names must by different name. Otherwise data oracle server returns ambiguously error. This reference name used by data based on server in query execution time only.

**Q) Write a query to display the working has same job as SCOTT from emp table using self join?**

SQL> select e2.ename,e1.job from emp e1,emp e2

where e1.job=e2.job and e1.ename='SCOTT';

**ENAME** **JOB** e1 e2

Ename Job

? Z

Ename Job

SCOTT Z

SCOTT ANALYST

FORD ANALYST

**Q) Write a query to display ename,manager from emp table using self join?**

Sql>select e1.ename “Emaployee”,e2.ename “Manager” from emp e1,emp e2

Where e1.mgr=e2.empno;

Sql>select e1.empno,e1.ename,e1.mgr,e2.ename “Manager” from emp e1,emp e2

Empno Ename Mgr

7369 SMITH 7902

Empno Ename Mgr

7902

Where e1.mgr=e2.empno;

**Q) Write a query to display the emp and their manager in 10th department from emp table using self join?**

SQL>select e1.empno,e1.ename,e1.mgr,e1.deptno,e2.ename "Manager" from emp e1, emp e2 Where e1.mgr=e2.empno AND e1.deptno=10;

**EMPNO ENAME MGR DEPTNO Manager**

7934 MILLER 7782 10 CLARK

7782 CLARK 7839 10 KING

Sql>select e1.ename "Employee",e1.sal,e2.ename "Manager",e2.sal

from emp e1,emp e2

where e1.mgr=e2.empno and e1.sal>e2.sal;

**Employee SAL Manager SAL**

FORD 3000 JONES 2975

SCOTT 3000 JONES 2975

**Q) Display the emp who are before joining their MGR from emp using self join?**

SQL> select e1.ename "Employee",e1.hiredate,e2.ename "Manager",e2.hiredate

from emp e1,emp e2 where e1.mgr=e2.empno and e1.hiredate<e2.hiredate;

**Q) Write a query to display emp and their mgr who are working in the sales emp,dept tables using self join?**

Sql>select e1.ename "Employee",e1.sal,e2.ename "Manager",e2.sal from emp e1,dept d,emp e2 Where e1.mgr=e2.empno AND e1.deptno=d.deptno and d.dname='SALES';

**Q) Write a query to display the emp who are getting same salary in different dept from emp table using self join?**

Sql>select e1.ename,e1.sal,e1.deptno,e2.sal,e2.ename,e2.deptno from emp e1,emp e2

Where e1.sal=e2.sal and e1.deptno<>e1.deptno;

**Q) Write a query to display emp who joining same month from emp table using self join?**

Sql> select e1.ename,e1.hiredate,e2.ename,e2.hiredate from emp e1,emp e2

where to\_char(e1.hiredate,'MON')=to\_char(e2.hiredate,'MON')

and e1.empno<e2.empno;

**4. Outer join:**

This join is used to retrieve all rows from one table and matching rows for another table. Generally when we are using equi join database server returns matching rows only. In oracle if you want retrieve non-matching rows also then we are using outer join (+) operator in equi join this is called “8i outer join”.

**NOTE:** In this case join operator can be used only one side at a time in joining condition of the equi join.

Sql>select ename,sal,d.deptno,dname,loc from emp e,dept d

**+** Not allowed

**(+)** Allowed

Where e.deptno(+)=d.deptno;

Sql>select ename,sal,d.deptno,dname,loc from emp e,dept d

**+** Allowed

**(+)** Not Allowed

Where e.deptno=+d.deptno;

**Olden days both side using (+) operator:**

Sql> select ename,sal,d.deptno,dname,loc from emp e,dept d

Where e.deptno(+)=d.deptno; Left only use (+)

UNION Right only use +

select ename,sal,d.deptno,dname,loc from emp e,dept d

Where e.deptno=+d.deptno;

**9i joins (or) ANSI joins:**

1. Inner join

2. Left outer join

3. Right outer join

4. Full outer join

5. Natural join

**1. Inner join:**

This joins returns matching rows only. When table having common columnname then only we are using this join this join performance very high compare to 8i equi join. Here joining conditional column same datatype.

Sql>select ename,sal,d.deptno,dname,loc from emp e join dept d

On e.deptno=d.deptno;

**Q) Write a query to display the emp who are working in the loc Chicago from emp,dept using 9i equi join?**

Sql>select ename,loc from emp join dept on emp.deptno=dept.deptno where loc='CHICAGO';

WHERE in this place AND also using.

**Using clause:**

ANSI join we are also use USING clause in place of ON clause. USING clause performance very high compare to ON clause and also when we are using USING clause data we returns database server returns one time only.

**Syntax:** select \* from tablename1 join tablename2 USING(common columnname);

Sql>create table z1(a varchar2(5),b varchar2(5),c varchar2(5));

Sql>insert into z1 values(‘x’,’y’,’z’);

SQL> select \* from z1;

**A B C**

x y z

Sql>create table z2(a varchar2(5),b varchar2(5));

Sql>insert into z1 values(‘x’,’y’);

SQL> select \* from z2;

**A B**

x y

Sql>select \* from z1,z2;

**A B C A B**

x y z x y

Sql>select \* from z1,z2 where z1.a=z2.a and z1.b=z2.b;

Sql> select \* from z1 join z2 on(z1.a=z2.a and z1.b=z2.b);

**A B C A B**

x y z x y

SQL> select \* from z1 join z2 using(a,b);

**A B C**

x y z

**NOTE:**

We are using USING clause we are not allowed to use aliasname and joining condition.

Sql>select ename,sal,deptno,dname,loc from emp e join dept d using(deptno);

**ENAME SAL DEPTNO DNAME LOC**

SMITH 800 20 RESEARCH DALLAS

ALLEN 1600 30 SALES CHICAGO

WARD 1250 30 SALES CHICAGO

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

**2. Left outer join:**

This join all rows from left side table and matching rows from right side table and also return null values in place of non-matching rows display another table.

SQL> insert into z1 values('p','q','r');

SQL> select \* from z1;

**A B C**

x y z

p q r

SQL> insert into z2 values('s','t');

SQL> select \* from z2;

**A B**

x y

s t

SQL> select \* from z1 left outer join z2 on(z1.a=z2.a and z1.b=z2.b);

**A B C A B**

x y z x y

p q r

**3. Right outer join:**

This join always returns all rows from right side table and matching rows from left side table and also returns null values in place of non-matching rows.

Sql>select \* from z1 right outer join z2 on z1.a=z2.a and z1.b=z2.b;

**A B C A B**

x y z x y

s t

**4. Full outer join:**

This join always returns all rows from right side table and matching rows left side table and also returns null values in place of non-matching rows.

SQL> select \* from z1 full outer join z2 on z1.a=z2.a and z1.b=z2.b;

**A B C A B**

x y z x y

p q r

s t

Sql> select ename,sal,d.deptno,loc,dname from emp e natural join dept d;

**ERROR:** Column used in NATURAL join cannot have qualifier

**5. Natural join:**

Natural join return matching rows only in natural we are not allowed to joining condition When ever table having common column then only we are using this join. This join internally uses USING clauses. That why this join always returns common column one time only and also we are not allowed using aliasname and joining conditional column.

Sql> select ename,sal,deptno,loc,dname from emp e natural join dept d;

SQL> select \* from z1 natural join z2;

**A B C**

x y z

**Cross join:**

Sql>select ename,sal,dname,loc from emp cross join dept;

**Joining more than two tables:**

**8i joins**

**Syntax:**

Select col1,col2………………

From table1,table2,table3

Where table1.commoncol=table2.commoncol

And table2.commoncol=table3.commoncol;

**9i joins/ANSI joins**

**Syntax:**

Select col1,col2……………..

From table1 join table2

On table1.commoncol=table2.commoncol

Join table3

On table2.commoncol=table3.commoncol;

**Chapter-6**

**CONSTRAINTS**

Constraints are used prevents (or) invalid data entry into out table. Generally constraints are created on table columns oracle server having following type of constraints.

1. Not null

2. Unique

3. Primary key

4. Foreign key

5. Check

All the above constraints are defining two levels

1. Column Level

2. Table Level

**1. Column Level:**

In this method we are defining constraining on individual columns that is whenever we defining column then only we are using constraint type.

**Syntax:** create table tablename(col1 datatype(size) constraints type,col2 datatype(size) constraint type……………);

**2. Table Level:**

In this method we are defining constraints on group of column that is first we are defining all the columns and last only we are specifying constraint type along with group of column.

**Syntax:** create table tablename(col1 datatype(size),col2 datatype(size)…………….

constraint type(col1,col2………………));

**1. Not null:**

Not null constraints does not support table level this constrains does not accept null values but it accept duplicate value.

Sql>Create table c1(sno number(5) not null,name varchar2(15));

SQL> insert into c1 values(null,'a');

**ERROR:** ORA-01400: cannot insert NULL into SNO

**2. Unique:**

This constrains defining two level that is does not accept duplicate values but if accept null values. Whenever we are creating unique constrains oracle server automatically create on btree index on those column.

**Column level:** Sql>create table c2(sno number(5) unique,name varchar2(15));

SQL> insert into c2 values(1,'abc');

SQL> insert into c2 values(2,'xyz');

SQL> select \* from c2;

**SNO NAME**

1 abc

2 xyz

**Table level:** sql> create table c2(sno number(5),name varchar2(15),unique(sno,name));

SQL> select \* from c3;

**SNO NAME**

1 abc

2 xyz

SQL> insert into c3 values(2,'xyz');

**ORA-00001:** unique constraint violated

1 abc, 2 xyz this is not allowed.

**3. Primary key:**

Primary key uniquely identifier record in a table and also where can be only primary in a table and also primary key does not accept null values and duplicate values. Whenever we are creating primary key internally oracle server create and btree index on those column.

**Column level:** create table c4(sno number(5) primary key,name varchar2(15));

**Table level:** create table c5(sno number(5),name varchar2(15),primary key(sno,name));

This is a composite primary key i.e it is combination column as single primary key.

**4. Foreign key:**

We want to establish relationship between tables then we are using referential integrity constraints foreign key. One table foreign key must belong to another table primary key. Here these two columns must belong to same datatype. Always foreign key values based on primary key only.

Generally primary key does not excepts duplicate,null values where has foreign key accepts duplicates,null value.

**Column level (References):**

**Syntax:** create table tablename(col1 datatype(size),references mastertablename(primary key columnname),col2 datatype(size)……);

Sql>create table f1(sno number(5) primary key);

Sql>create table ff1(sno number(5) references f1);

Sql>create table b1(a number(5) references f1(sno));

**Table level (Foreign key,References):**

**Syntax:** create table tablename(col1 datatype(size),col2 datatype(size)…….

Foreign key(col1,col2………),reference mastertablename(pk colname);

Sql>create table f2(sno number(5),name varchar2(15) primary key(sno,name));

Sql>create table ff2(sno number(5),name varchar2(15)col3 number(5),foreign key(sno,name),references f2);

SNO

SNO

1

1

2

2

3

Child

Master

Primary key Foreign key

References

On delete cascade

On delete set null

**Rule:**

Whenever we are establishing relationship between two tables oracle server violates following two rules.

1. Deletion in master table

2. Insertion in child table

**1. Deletion in master table:**

if we are try to delete a record in master table than oracle server returns a error ORA-2292: to reference this problem first we are deleting children table records in a table. Than only we are deleting those record master table otherwise used on delete cascade clause.

**On delete cascade:**

Whenever we are using this clause in child table record oracle server automatically delete appropriate master,child table record on those table.

**Syntax:** create table tablename(col1 datatype(size) references mastertablename(primary key columnname) on delete cascade………);

**Ex:** sql>create table mas(sno number(5) primary key);

Sql>insert into mas(1);

Sql>select \* from mas; **o/p:** sno:1 2 3 4……………..

Sql>create table child(sno number(5) references mas on delete cascade);

Sql>insert into mas(1);

Sql>select \* from mas; **o/p:** sno:1 1 1 2 2 2 3 3 3 4……………..

**Testing (deletion in master):**

Sql>delete from mas where sno=1;

Sql>select \* from child;

**On delete set null:**

Whenever we are using this clause in child table if we are deleting primary key value in master table then automatically foreign key value set to null in child table.

**Syntax:** create table tablename(col1 datatype(size) references mastertablename(primary colname) on delete set null………);

**2. Insertion in child table:**

If we try to insert other than primary key value into foreign key then oracle server returns on **ERROR:** ora-2291: Because always foreign key values primary key values only.

**5. Check:**

This constraint is used to define logical condition according to our business rules.

**Syntax:** create table tablename(col1 datatype(size) check(logical condition ),col2 datatype(size),………………..);

Sql>create table nr(name varchar2(15) check(name=upper(name));

Sql>insert into nr values(‘NARASIMHA’);

**Name**

NARASIMHA

SQL> insert into nr values('narasimha');

**ERROR:** ORA-02290: check constraint violated

**Assign user defined names to constraints:**

In all database systems whenever we are creating constraint internally database server automatically generates unique identification number in oracle. Whenever we are creating constraint then oracle server automatically generates an unique identification number in the format of “SYS\_CN”. This is called predefined constraint name we can also create our own name in place of SYS\_CN using constraint keyword at the time of table creation. This is called user defined constraint name.

**Syntax:** constraint userconstraintname constrainttype;

**Ex:** sql>create table test(sno number(10) primary key);

SQL> insert into test values(1);

SQL> insert into test values(1);

**ERROR:** ORA-00001:unique constraint(SYSTEM.SYS\_C004274)violated

Sql>create table test1(sno number(10) constraint p\_abc primary key);

Sql> insert into test1 values(1);

Sql>insert into test1 values(1);

**ERROR:**ORA-00001: unique constraint (SYSTEM.P\_ABC) violated

Whenever we are installing oracle server automatically creating read only table those read only table are also called as data dictionary.

Oracle Database

User\_Constraints

User\_Constraints\_columns

Constraint\_name column\_name

Owner constraint\_name constraint\_type

All constraints information stored under in user\_constraints data dictionary.

Sql>desc user\_constraints;

Sql>select constraint\_name,constraint\_type from user\_constraints

where table\_name=’EMP;

**cons\_name con\_type**

pk\_emp p

fk\_deptno R

**NOTE:** If we want to view column names along with constraint name then we are using user\_constraint\_columns.

Sql>select constraint\_name,column\_name from user\_constraints\_column

where table\_name=’EMP;

**cons\_name col\_name**

pk\_dept deptno

fk\_emp empno

**NOTE:** If we want view logical condition of the check constraints then we are using search\_condition property user\_constraints data dictionary.

Sql>select search\_condition from user\_constraints where table\_name=’g5’;

**search\_condition**

name=upper(name)

**NOTE:** All column information stored under user\_tab\_column data dictionary.

**Ex:** sql>desc user\_tab\_columns

Sql>select column\_name from user\_tab\_col where table\_name=’EMP’;

**Column\_name**

Empno

Ename

-----------

**Q) Write a query to display no.of columns numbers from emp using data dictionary?**

Sql>select count(\*) from user\_tab\_column where table\_name=’EMP’;

**Count(\*)**

8

**Adding (or) dropping constructions on existing table:**

Using ALTER we can also add (or) remove existing table.

**NOTE:** If we want add constraints into existing table existing columns then we are using table level syntax method.

X1

SNO

Sql>alter table x1 add primary key(sno);

**NOTE:** If we want to add a new column along with constraints then we are using column level syntax method.

X2

|  |  |
| --- | --- |
| SNO | NAME |
|  |  |

Sql>alter table x1 add name varchar(15) unique;

Sql>alter table x2 add foreign key(sno) reference x1;

**NOTE:** If we want add not null constraints existing table columns then we are using alter with modify.

**Syntax:** alter table tablename modify colname not null;

Sql>alter table x3 modify sno not null;

Sql>alter table x3 add name varchar2(15) not null;

**Dropping constraints:**

Method-1: sql>alter table tablename drop constraintname;

Method-2: sql>alter table tablename drop primary key;

Sql>alter table tablename drop unique(col1,col2,……….);

**Ex:** sql>create table test(sno number(5) primary key);

Sql>alter table test drop primary key;

Sql>alter table x1 drop primary key;

**ERROR:** This unique/primary key is reference by some foreign key’s.

**NOTE:** If we want drop primary key along with foreign key then we are using cascade clause along with alter drop.

**Syntax:** alter table tablename drop primary key cascade;

Sql>alter table x1 drop primary key cascade;

**Ex:** sql>alter table x3 drop constraint sys\_coo5358

**NOTE:** Oracle also support default constraints for a column

**Syntax:** columnname datatype(size) default actual value

Sql>create table test(sno number(5) default 10,name varchar(15));

SQL> insert into test values(1,'A');

SQL> insert into test values(2,'B');

SQL> insert into test(name) values('C');

SQL> select \* from test;

**SNO NAME**

1 A

2 B

10 C

**Ex:** SQL> create table dept1 as select \* from dept;

SQL> create table emp1 as select \* from emp;

SQL> alter table dept add primary key(deptno);

SQL> alter table emp add primary key(empno);

SQL> alter table emp add foreign key(deptno) references dept(deptno) on delete cascade;

**Chapter-7**

**SUBQUERY**

Query within another query is called nested query (or) subquery. Generally child query are called subquery. Subqueries are used to retrieve data from single and multiple tables by using more than one step process. All database systems having two type of subqueries.

1. Non-correlated

2. Correlated

In non-correlated subquery child query is executes first then only parent query executed. Where as in correlated subquery parent query executed first then only child query executed.

**1. Non-correlated:**

1. Single rows subquery

2. Multiple rows subquery

3. Multiple column subquery

4. Inline view (or) subquery used in from clause.

**Q) Write a query to display the emp’s who are getting more than the average salary from emp table?**

Sql>select \* from emp where sal>(select avg(sal) from emp);

This are single row subquery because there child query returns single value. In single row subquery we are using =,<,<=,>,>=,<> operator.

**Execution:**

**Step-1:-** sql>select avg(sal) from emp;

**AVG(SAL)**

2073.21429

**Step-2:-** Sql>select \* from emp where sal>2073.21429

**Child query:** A query which provides value to anther query is called “CHILD QUERY”.

**Parent query:** A query which receives values from another query is called “PARENT QUERY”.

**Q) Write a query to display the emp’s who are working salesman department from dept,emp table using subquery?**

Sql>select \* from emp where deptno=(select deptno from dept where dname=’SALES’);

**NOTE:**

Generally using subquery we are not allowed to display child query column into parent query to overcome this problem we must use join concept.

SQL> select ename,dname from emp e,dept d where e.deptno=d.deptno and d.deptno=(select deptno from dept where dname='SALES');

**ENAME** **DNAME**

ALLEN SALES

WARD SALES

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

**Q) Write a query to display the emp’s who are working SMITH dept table.**

Sql>select \* from emp where deptno=(select deptno from emp where ename=’SMITH’);

**Q) Write a query to display senior most emp details from emp table?**

Sql>select \* from emp where hiredate=(select min(hiredate) from emp);

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7369 SMITH CLERK 7902 17-DEC-80 800 20

**Q) Write a query to display the emp who are working under BLAKE from emp table using empno,mgr?**

Sql>select \* from emp where mgr=(select empno from emp where ename=’BLAKE’);

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300 30

7521 WARD SALESMAN 7698 22-FEB-81 1250 500 30

7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400 30

7844 TURNER SALESMAN 7698 08-SEP-81 1500 0 30

7900 JAMES CLERK 7698 03-DEC-81 950 30

**Q) Write a query to display lowest average salary job from emp table?**

Sql>select job,avg(sal) from emp group by job having avg(sal)=(select min(avg(sal)) from emp);

**ERROR:** ORA-00978: nested group function without GROUP BY

**NOTE:** Whenever child query contains nested group function then we must using group by clause.

Sql>select job,avg(sal) from emp group by job having avg(sal)=(select min(avg(sal)) from emp group by job);

**JOB AVG(SAL)**

CLERK 1037.5

Sql>select deptno,min(sal) from emp group by deptno having min(sal)>(select min(sal) from emp where deptno=20);

**DEPTNO MIN(SAL)**

30 950

10 1300

**Q) Write a query to display the emp delete who are getting maximum sal in each dept from emp table?**

Sql>select \* from emp where sal=(select max(sal) from emp group by deptno);

**ERROR:** ORA-01427: single-row subquery returns more than one row

This are multiple row subquery because here child query returns multiple value in multiple rows subquery we are using IN,ALL,ANY operator.

**NOTE:**

We can also use IN operator in single row subquery.

Sql>select \* from emp where sal in(select max(sal) from emp group by deptno);

Sql> select deptno,sal from emp where sal in(select max(sal) from emp group by deptno);

**Q) Write a query to display the emp’s who are working in ACCORDING,SALES department using emp,dept table?**

Sql>select \* from emp where deptno in(select deptno from dept where dname=’ACCORDING’ or dname=’SALES’);

**TON-N Analysis:**

1. Inline view

2. Rownum

**1. Inline view: (Oracle 7.2 introduced)**

Generally we are not allowed to use order by clause in child query to overcome this problem oracle 7.2 introduced subqueries in from clause this is called as “INLINE IEW”.

**Syntax:** select \* from (subquery)

In inline views we are using subquery in place tablename in parent query this subquery execute to prier executed.

**2. Rownum:**

Rownum is a pseudo column which behavior table column. This pseudo used in all table. It cannot perform DML operators. Rownum automatically assign numbers to each row in a table at the time of selection. By default rownum having temporarily values.

Sql>select rownum,ename from emp;

**ROWNUM ENAME**

1 SMITH

2 ALLEN

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

Sql>select rownum,ename from emp where deptno=10;

**ROWNUM ENAME**

1 CLARK

2 KING

3 MILLER

**Q) Write a query to display first row from emp table using rownum?**

Sql>select \* from emp where rownum=1;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7369 SMITH CLERK 7902 17-DEC-80 800 20

**Q) Write a query to display second row from emp table using rownum?**

Sql>select \* from emp where rownum=2;

No Row Selected.

**NOTE:** Rownum does not work with more than one positive integer. Then is it work’s with **<,<=** operators.

**Q) Write a query to display first 5 row from emp table using rownum?**

Sql>select \* from emp where rownum<=5;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7369 SMITH CLERK 7902 17-DEC-80 800 20

7499 ALLEN SALES 7698 20-FEB-81 1600 300 30

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

**Q) Write a query to display first 5 highest salary from emp table using rownum?**

SQL> select ename,job,sal,deptno from (select ename,job,sal,deptno from emp order by sal desc) where rownum<=5;

**ENAME JOB SAL DEPTNO**

KING PRESIDENT 5000 10

SCOTT ANALYST 3000 20

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

**Q) Write a query to display 5th highest salary employee from emp table using rownum?**

SQL> select ename,job,sal,deptno from (select ename,job,sal,deptno from emp order by sal desc) where rownum<=5

minus

select ename,job,sal,deptno from (select ename,job,sal,deptno from emp order by sal desc) where rownum<=4;

**ENAME** **JOB** **SAL** **DEPTNO**

BLAKE MANAGER 2850 30

**Q) Write a query to display second row from emp table assigning rownum?**

Sql>select \* from emp where rownum<=2

Minus

select \* from emp where rownum<=1;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7499 ALLEN SALES 7698 20-FEB-81 1600 300 30

**Q) Write a query to display last two rows from emp table using rownum?**

Sql>select \* from emp

Minus

Select \* from emp where rownum<=(select count(\*) -2 from emp);

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7902 FORD ANALYST 7566 03-DEC-81 3000 20

7934 MILLER CLERK 7782 23-JAN-82 1300 10

**Q) Write a query to display the rows between 1 and 5 from emp table using rownum?**

Sql>select \* from emp where rownum between 1 and 5;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7369 SMITH CLERK 7902 17-DEC-80 800 20

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

7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400 30

**Q) Write a query to display the rows between 4 and 7 from emp table using rownum?**

Sql>select \* from emp where rownum<=7

Minus

Select \* from emp where rownum<=4;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7369 MARTIN CLERK 7902 17-DEC-80 800 20

7499 CLARK SALESMAN 7698 20-FEB-81 1600 300 30

7654 BLAKE SALESMAN 7698 28-SEP-81 1250 1400 30

Sql>select \* from emp where rownum>=1;

Rows selected

Sql>select \* from emp where rownum>1;

No rows selected

**NOTE:**

Wherever we are using aliasname for rownum in inline view that aliasname works with all sql operators.

**Q) Write a query to display 5th row from emp table using rownum aliasname?**

Sql> select \* from (select rownum r,ename,sal from emp) where r=5;

**R ENAME SAL**

5 MARTIN 1250

Sql> select \* from (select rownum r,ename,sal from emp) where r in(2,3,5,6,10,14);

**R ENAME SAL**

2 ALLEN 1600

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

14 MILLER 1300

Sql>select \* from (select rownum r,emp.\* from emp) where r=5;

**R EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

5 7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400 30

**Q) Write a query to display first and last row from emp table using rownum aliasname?**

Sql> select \* from (select rownum r,ename,sal from emp) where r=1 or r=(select count(\*) from emp);

**R** **ENAME** **SAL**

1 SMITH 800

14 MILLER 1300

**Q) Write a query to display even no.of rows from emp table using rownum aliasname?**

Sql> select \* from (select rownum r,ename,sal from emp) where mod(r,2)=0;

**R** **ENAME** **SAL** odd number mod(r,2)=1

2 ALLEN 1600

4 JONES 2975

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

12 JAMES 950

14 MILLER 1300

**Q) Write a query to display 5th highest salary from emp table using rownum aliasname?**

Sql>select \* from (select rownum r,sal,ename from (select \* from emp order by sal desc)) where r=5;

**R SAL ENAME**

5 2850 BLAKE

**Analytical Function used in inline views: (Oracle 8i introduced)**

Analytical functions also similar to GROUP functions and also these functions display each row in a group because these functions are calculate results number of rows wise.

1. row\_number()

2. rank()

3. dense\_rank()

These three analytical functions automatically assigns numbers to each row in table either group wise (or) rows wise in a table.

**Syntax:** analyticalfunctionname() optional clause

over(**partition by colname** order by colname[asc/desc];

ROW\_NUMBER() analytical function assigns different rank numbers when values same where as RANK(),DENSE\_RANK() analytical function assign same rank number then value are same and also RANK() scrip next consecutive rank number where has DENSE\_RANK() does not scripts next consecutive number.

**ROW\_NUMBER():**

**DEPTNO ENAME SAL R**

20 SMITH 3000 1

20 SCOTT 3000 2

20 ADAMS 1200 3

**RANK():**

**DEPTNO ENAME SAL R**

20 SMITH 3000 1

20 SCOTT 3000 1

20 ADAMS 1200 3

**DENSE\_RANK():**

**DEPTNO ENAME SAL R**

20 SMITH 3000 1

20 SCOTT 3000 1

20 ADAMS 1200 2

Sql>select \* from (select deptno,ename,sal,row\_number() over(partition by deptno

order by sal desc)r from emp) where r<=5;

RANK()

DENSE\_RANK() also use

**DEPTNO ENAME SAL R**

10 KING 5000 1

10 CLARK 2450 2

10 MILLER 1300 3

20 SCOTT 3000 1

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

20 SMITH 800 5

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

30 MARTIN 1250 5

**Q) Write a query to display 2nd highest salary employee in each department from emp table using analytical function?**

Sql> select \* from (select deptno,ename,sal,dense\_rank() over(partition by deptno order by sal desc)r from emp) where r=2;

**DEPTNO ENAME SAL R**

10 CLARK 2450 2

20 JONES 2975 2

30 ALLEN 1600 2

**Q) Write a query to display 5th highest salary employee from emp table using analytical function?**

Sql> select \* from (select deptno,ename,sal,dense\_rank() over(order by sal desc)r

from emp) where r=5;

**DEPTNO** **ENAME** **SAL** **R**

10 CLARK 2450 5

**Q) Write a query to display 2nd row from emp using analytical function?**

SQL> select \* from (select ename,sal,dense\_rank() over(order by rowid)r from emp) where r=2;

**ENAME SAL R**

ALLEN 1600 2

**NOTE:**

We are also use analytical function without use inline view always analytical function returns row wise group data.

**LEAD(),LAG():**

These two analytical functions compare next row values and returns into the current row.

**Syntax:** lead(expression,offset,default value) over(partition by colname order by colname [asc/desc]);

Here offset returns positive integer if we are specifying 1 it returns next row immediate.

Sql>select deptno,ename,sal,lead(sal,1,0) over(partition by deptno

order by sal desc) lead\_sal from emp;

**DEPTNO** **ENAME** **SAL** **LEAD\_SAL**

10 KING 5000 2450

10 CLARK 2450 1300

10 MILLER 1300 **0**

**NOTE:**

Lag() analytical function returns previous value into it is also same syntax as lead function but offset returns previous row.

Sql>select deptno,ename,sal,lag(sal,1,0) over(partition by deptno order by sal desc) lag\_sal from emp;

**DEPTNO** **ENAME** **SAL** **LAG\_SAL**

10 KING 5000 **0**

10 CLARK 2450 5000

10 MILLER 1300 2450

**FIRST\_VALUE:**

First value analytical function returns first value in each group. This function excepts only one parameter.

Sql> select deptno,ename,sal,first\_value(sal) over(partition by deptno order by sal desc) first\_sal from emp;

**DEPTNO** **ENAME** **SAL** **FIRST\_SAL**

10 KING 5000 5000

10 CLARK 2450 5000

10 MILLER 1300 5000

**Q) Write query to display year number of employees in that year and in which year more than one employee hide?**

SQL>select to\_char(hiredate,'yyyy') "YEAR",count(\*) from emp group by to\_char(hiredate,'yyyy');

**YEAR** **COUNT(\*)**

1980 1

1983 1

1982 2

1981 10

SQL>select to\_char(hiredate,'yyyy') "YEAR",count(\*) from emp group by to\_char(hiredate,'yyyy') having count(\*)>1;

**YEAR** **COUNT(\*)**

1982 2

1981 10

**Q) Write a query to display duplicate data from a table?**

SQL> select deptno,count(\*) from emp group by deptno having count(\*)>1;

**DEPTNO** **COUNT(\*)**

30 6

20 5

10 3

**ROWID:**

Rowid is a pseudo column it’s behaviors like a table column pseudo column are not stored in tables. Whenever we are inserting data oracle server automatically generates unique identifier number identifying each row in a table in hexadecimal format this is called as “ROWID” (or) “ROWADDRESS”.

Generally rownum having temporarily values where as rowid having fixed value. Using rowid we are identifying a record uniquely in a table. Using rowid we can also data retrieve very fastly from the database.

Sql>select rownum,rowid,ename from emp;

**ROWNUM** **ROWID** **ENAME**

1 AAADVOAABAAAIGCAAA SMITH

2 AAADVOAABAAAIGCAAB ALLEN

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

14 AAADVOAABAAAIGCAAN MILLER

SQL> select rownum,rowid,ename from emp where deptno=10;

**ROWNUM** **ROWID** **ENAME**

1 AAADVOAABAAAIGCAAG CLARK

2 AAADVOAABAAAIGCAAI KING

3 AAADVOAABAAAIGCAAN MILLER

**NOTE:** We can also use max(),min() function in rowid.

Sql>select min(rowid) from emp;

Sql>select max(rowid) from emp;

In subquery we can also use rowid to delete to subquery duplicate rows in a table. i.e if you want to delete rows except one row each.

**Q) Write a query to display 1st record from the emp using rowid?**

SQL> select \* from emp where rowid in(select min(rowid) from emp);

**EMPNO** **ENAME** **JOB** **MGR** **HIREDATE** **SAL** **COMM** **DEPTNO**

7369 SMITH CLERK 7902 17-DEC-80 800 20

**Q) Write a query to display last record from the emp using rowid?**

SQL> select \* from emp where rowid in(select max(rowid) from emp);

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7934 MILLER CLERK 7782 23-JAN-82 1300 10

**Q) Write a query to display 2nd record from emp table using analytical function?**

Sql>select \* from (select empno,ename,sal,dense\_rank() over(order by rowid) r

from emp) where r=2;

**EMPNO** **ENAME** **SAL** **R**

7499 ALLEN 1600 2

**Q) Write a query to display each dept from the 2nd record from emp table using analytical function?**

SQL>select \* from (select deptno,ename,sal,dense\_rank() over(partition by deptno

order by rowid) r from emp) where r=2;

**DEPTNO** **ENAME** **SAL** **R**

10 KING 5000 2

20 JONES 2975 2

30 WARD 1250 2

**Q) Write a query to display duplicate data from the table?**

SQL> select deptno,count(\*) from emp group by deptno having count(\*)>1;

**Q) Write a query to delete duplicate rows except one row in each group from table using rowid?**

SQL> delete from test where rowid not in(select max(rowid) from test group by sno);

4 rows deleted.

Sql>select \* from test;

**MULTIPLE COLUMN SUBQUERY:**

We can also compare multiple column of the child query with the multiple column of the parent query. This type of query is “MULTIPLE COLUMN SUBQUERY”. In multiple column subqueries we must specify parent query when conditionals with in parenthesis.

**Syntax:** select \* from tablename where(col1,col2……….)

In(select col1,col2…….. from tablename where condition);

**Q) Write a query to display the emp’s who are job,mgr match with the job,mgr of the emp’s SCOTT from emp table?**

Sql>select \* from emp where (job,mgr) in(select job,mgr from emp

where ename='SCOTT');

**EMPNO** **ENAME** **JOB** **MGR** **HIREDATE** **SAL** **COMM** **DEPTNO**

7902 FORD ANALYST 7566 03-DEC-81 3000 20

7788 SCOTT ANALYST 7566 09-DEC-82 3000 20

**Q) Write a query to display senior most emp’s from each job from emp table using multiple column subquery?**

Sql>select deptno,sal,ename,job,hiredate from emp where(job,hiredate) in(select job,min(hiredate) from emp group by job);

**DEPTNO** **SAL** **ENAME** **JOB** **HIREDATE**

20 800 SMITH CLERK 17-DEC-80

30 1600 ALLEN SALESMAN 20-FEB-81

10 5000 KING PRESIDENT 17-NOV-81

20 2975 JONES MANAGER 02-APR-81

20 3000 FORD ANALYST 03-DEC-81

**Q) Write a query to display the emp’s who are getting maximum sal in each dept from emp table using multiple row subquery?**

Sql>select deptno,ename,sal from emp where sal in(select max(sal) from emp group by deptno);

**DEPTNO** **ENAME** **SAL**

30 BLAKE 2850

20 FORD 3000

20 SCOTT 3000

10 KING 5000

**Q) Write a query to display the emp’s who are getting maximum sal in each dept from emp table using multiple row subquery?**

Sql>select deptno,ename,sal from emp where (deptno,sal) in(select deptno,max(sal) from emp group by deptno);

**DEPTNO** **ENAME** **SAL**

30 BLAKE 2850

20 SCOTT 3000

10 KING 5000

**Q) Write a query to display ename,dname,sal of the emp who sal,comm match with sal,comm of the emp’s working in location “DALLAS” from emp,dept tables?**

SQL>select dname,ename,sal from emp e,dept d where e.deptno=d.deptno

and (sal,nvl(comm,0)) in(select sal,nvl(comm,0) from emp e,dept d

where e.deptno=d.deptno and loc='DALLAS' );

**DNAME** **ENAME** **SAL**

RESEARCH SMITH 800

RESEARCH JONES 2975

RESEARCH FORD 3000

RESEARCH SCOTT 3000

RESEARCH ADAMS 1100

**Q) Write a query to display emp’s who are getting more than the highest paid emp in 20th dept from emp table?**

Sql>select \* from emp where sal>(select max(sal) from emp where deptno=20);

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7839 KING PRESIDENT 17-NOV-81 5000 10

**Q) Write a query to display emp’s who are getting more than the lowest paid emp in 20th dept from emp table?**

Sql>select \* from emp where sal>(select min(sal) from emp where deptno=20);

**ALL, ANY OPERATOR:**

When ever resource table having child amount of data and also child query contain max(),min() function and if we are comparison more number of value using relational operator. These type of queries degrades performance of the applications to overcome these problems ANSI/ISO sql provided subquery special operators these are “ALL,ANY”. This operator is used along with relational operator. These types of operators improve performance.

Sql>select \* from emp where sal>all(select sal from emp where deptno=20);

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7839 KING PRESIDENT 17-NOV-81 5000 10

SQL> select \* from emp where sal>any(select sal from emp where deptno=10);

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7839 KING PRESIDENT 17-NOV-81 5000 10

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

7844 TURNER SALESMAN 7698 08-SEP-81 1500 0 30

ALL---------🡪max(sal) ANY-------🡪min(sal)

**NOTE:**

When ever subquery special operator internally optimizer uses when we are using ANY operator internally optimizers uses ‘OR’ logical operator.

**ALL, ANY operator used in multiple row subqueries:**

IN -----🡪 It returns same value in the list

ALL -----🡪 It satisfies all values in the list

ANY-----🡪 It satisfies any value in the list

**Q) Write a query to display the emp’s who are getting more than the salaries of the ‘CLARK’ from emp table using subquery special operator?**

Sql>select \* from emp where sal>all(select sal from emp where job='CLERK')

order by sal desc;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7839 KING PRESIDENT 17-NOV-81 5000 10

7902 FORD ANALYST 7566 03-DEC-81 3000 20

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

7844 TURNER SALESMAN 7698 08-SEP-81 1500 0 30

**NOTE:**

‘OR’ operator internally uses logical operator for ‘AND’ where has ‘ANY’ operator logical use operator ‘OR’.

Sql>select empno,ename,sal,deptno from emp where deptno>any(10,20);

**EMPNO** **ENAME** **SAL** **DEPTNO**

7369 SMITH 800 20

7499 ALLEN 1600 30

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

Sql>select empno,ename,sal,deptno from emp where deptno>all(10,20);

**EMPNO** **ENAME** **SAL** **DEPTNO**

7499 ALLEN 1600 30

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

**NOTE:** ‘=any’ is same as ‘IN’ operator ‘=any’ performance very high compare to IN operator. Whereas ‘<>any’ not same as ‘NOT IN’ operator.

SQL>select empno,ename,sal,deptno from emp where deptno <>any(10,20);

**EMPNO** **ENAME** **SAL** **DEPTNO**

7369 SMITH 800 20

7499 ALLEN 1600 30

7934 MILLER 1300 10

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

SQL> select empno,ename,sal,deptno from emp where deptno not in(10,20);

**EMPNO** **ENAME** **SAL** **DEPTNO**

7499 ALLEN 1600 30

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

SQL> select empno,ename,sal,deptno from emp where deptno in(10,20);

**EMPNO** **ENAME** **SAL** **DEPTNO**

7369 SMITH 800 20

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

7934 MILLER 1300 10

SQL> select empno,ename,sal,deptno from emp where deptno=any(10,20);

**EMPNO** **ENAME** **SAL** **DEPTNO**

7369 SMITH 800 20

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

7934 MILLER 1300 10

**2. CORRELATED SUBQUERY:**

Generally in non-correlated subquery child query is execute first then only parent query executed. Where as in correlated subquery parent query is executed first then only child query executed.

In correlated subquery we must create on aliasname for the parent query table in parent queries use the aliasname in child query where condition

**Syntax:** select \* from tablename aliasname where tablename operator

(select \* from tablename where tablename=aliasname.colname);

**Execution process:**

Whenever we are subquery having correlated subquery database server get a condition row from parent query table and then control passed into child query where condition based on evaluation value of where clause it compares query value with parent query.

Generally non-correlated subquery child query executed only once for parent table. Where has in correlated subqueries child query is executed **for each row** for a parent table.

Generally when resource table having large amount of data then we have not allowed to subqueries because in this case correlated is low performance. Generally correlated subquery used in denormalization process in this process we are modify one table column values based on another table columns in this use we are using correlated update.

**Syntax:** update tablename1 aliasname1 set columnname=(select columnname from tablename2 aliasname2 where aliasname1.commoncolname = aliasname2.commoncolname);

**Ex:** sql>alter table emp add dname varchar2(15);

Sql>update emp e set dname=(select dname from dept d

where e.deptno=d.deptno);

Sql>select \* from emp;

**Difference b/w non-correlated and correlated:**

**Non-correlated**

Outer query

Inner query

Inner query

**Correlated**

Result

Result

Outer query

**Q) Write a query to display 1st highest salary from emp table?**

Sql> select \* from emp e1 where 1=(select count(\*) from emp e2 where e2.sal>=e1.sal);

**EMPNO** **ENAME** **JOB** **MGR** **HIREDATE**  **SAL** **COMM** **DEPTNO**

7839 KING PRESIDENT 17-NOV-81 5000 10

**Q) Write a query to display 2nd highest salary from emp following table using correlated subquery?**

Sql>create table emp1(ename varchar2(5),sal number(5));

Sql>insert into emp1 values(………………………………..);

SQL> select \* from emp1;

EMP1

|  |  |
| --- | --- |
| **Ename** | **Sal** |
| A  B  C  D | 100  150  200  300 |

**ENAME** **SAL**

A 100 Candidate row

B 150

C 200

D 300

SQL>select \* from emp1 e1 where 2=(select count(\*) from emp1 e2 where e2.sal>=e1.sal);

**ENAME** **SAL**

C 200

**Phase-1:**

Step-1: get a candidate row (first row) ---------🡪 (A 100)

Step-2: select count(\*) from emp1 e2 where e2.sal>=100;

Step-3: select \* from emp1 e1 where 2=4 (False)

**Phase-1:**

Step-1: get a candidate row (second row) ---------🡪 (B 150)

Step-2: select count(\*) from emp1 e2 where e2.sal>=150;

Step-3: select \* from emp1 e1 where 2=3 (False)

**Phase-1:**

Step-1: get a candidate row (third row) ---------🡪 (C 200)

Step-2: select count(\*) from emp1 e2 where e2.sal>=100;

Step-3: select \* from emp1 e1 where 2=2 (True)

**Q) Write a query to display 2nd highest sal employee from emp above using correlated subquery N-1 method?**

Sql>select \* from emp1 e1 where (2-1)=(select count(\*) from emp1 e2

where e2.sal>e1.sal);

**ENAME SAL**

C 200

**NOTE:**

|  |  |
| --- | --- |
| **Ename** | **Sal** |
| A  B  C  D  C | 100  150  200  300  200 |

Whenever resource table having duplicate data then we are using N-1 method in case of N method it will be returns No Row Selected as a message to overcame this problem we must use DISTINCT clause.

EMP1

Sql> insert into emp1(ename,sal)values('c',200);

Sql>select \* from emp1 e1 where 2=(select count(\*)

from emp1 e2 where e2.sal>=e1.sal);

No Rows Selected (not current results)

Sql>select \* from emp1 e1 where 2=(select count(distinct(sal)) from emp1 e2

where e2.sal>=e1.sal);

**ENAME** **SAL**

C 200

C 200

**Q) Write a query to display Nth sal employee from emp table using correlated subquery?**

Sql>select \* from emp e1 where &n=(select count(distinct(sal)) from emp e2 where e2.sal>=e1.sal);

**Enter value for N:** 1

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7839 KING PRESIDENT 17-NOV-81 5000 10 **Enter value for n:** 2

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7788 SCOTT ANALYST 7566 19-APR-87 3000 20

7902 FORD ANALYST 7566 03-DEC-81 3000 20

**Q) Write a query to display the emp’s who are getting more than the avg salary of the job’s from emp the using correlated subquery?**

Sql>select \* from emp e where sal>(select avg(sal) from emp where job=e.job);

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300 30

7566 JONES MANAGER 7839 02-APR-81 2975 20

7698 BLAKE MANAGER 7839 01-MAY-81 2850 30

7844 TURNER SALESMAN 7698 08-SEP-81 1500 0 30

7876 ADAMS CLERK 7788 23-MAY-87 1100 20

7934 MILLER CLERK 7782 23-JAN-82 1300 10

**EXISTS OPERATOR:**

EXISTS operator used in correlated subquery. EXISTS operator performance very high compare to IN operator. EXISTS operator always returns Boolean value either True (or) False. EXISTS operator is used to test whether a given set is Empty (or) Non-empty.

EXISTS operator returns true if set having value non-empty and also this operator returns false if set is empty. These operator use WHERE clause only. Whenever we are use EXISTS operator then we are not allowed to use colname along with these operator.

**Syntax:** select \* from tablename aliasname where exists(select \* from tablename where colname=aliasname.colname);

**Ex:** Exists{1,2,3}=True Exists{ }=False

**NOTE:** Generally if we want CLERK one table column value into another table column values then we for EXISTS operator.

**Q) Write a query to display those dept details from dept table having emp’s in emp table assign correlated subquery from emp,dept tables?**

Sql> select \* from dept d where exists(select \* from emp where deptno=d.deptno);

**DEPTNO** **DNAME** **LOC**

10 ACCOUNTING NEW YORK

20 RESEARCH DALLAS

30 SALES CHICAGO

**NOTE:** Whenever we are using EXISTS operator we can also use constant after select in child query because EXISTS operator check child query returns Value (or) Not.

Sql> select \* from dept d where exists(select 1 from emp where deptno=d.deptno);

**DEPTNO** **DNAME** **LOC**

10 ACCOUNTING NEW YORK

20 RESEARCH DALLAS

30 SALES CHICAGO

**Q) Write a query to display dept table does not having emp’s in emp table from emp,dept table using correlated?**

Sql>select \* from dept d where not exists(select \* from emp

where deptno=d.deptno);

**DEPTNO** **DNAME** **LOC**

40 OPERATIONS BOSTON

**Q) Write a query to display dept table does not having emp’s in emp table from emp,dept table using correlated?**

Sql>select \* from dept where deptno not in(select deptno from emp);

**DEPTNO** **DNAME** **LOC**

40 OPERATIONS BOSTON

**NOTE:** Generally not in operator does not work with null values to overcome this problem we must use not EXISTS operator along with correlated subquery.

Sql>insert into emp(empno,ename,deptno)values(1,’murali’,null);

Sql> select \* from dept where deptno not in(select deptno from emp);

No Rows Selected

Sql> select \* from dept d where not exists(select \* from emp where deptno=d.deptno);

**DEPTNO** **DNAME** **LOC**

40 OPERATIONS BOSTON

**Q) Write a query to display who are getting same sal as emp’s SCOTT sal from emp table using correlated subquery in exists operator?**

Sql>select empno,ename,job,sal,deptno from emp e1 where exists(select \* from emp e2 where e2.ename='SCOTT' and e1.sal=e2.sal);

**EMPNO** **ENAME** **JOB** **SAL** **DEPTNO**

7902 FORD ANALYST 3000 20

7788 SCOTT ANALYST 3000 20

**MERGE STATEMENT:**

Oracle 9i introduced MERGE statement. MERGE is a DML statement which is used to transfer data from source table into target table if those two table’s structures are same. MERGE statement are also called as “UPSERT”. Because in MERGE we are using UPDATE, INSERT statements. Generally MERGE is used **data warehousing** applications. When we are using MERGE statement then we must create **aliasname**.

**Syntax:** merge into targettablename using sourcetablename

On (join condition)

When matched then

Update set targettablecolname=sourcetablecolname1, ………………

When not matched then

Insert (targettablecolnames)values(sourcecolvalues);

In MERGE statement we are using UPDATE, INSERT statement but through MERGE statement we cannot modify ON clause.

Sql> select \* from dept; (Target Table)

**DEPTNO** **DNAME** **LOC**

10 ACCOUNTING NEW YORK

20 RESEARCH DALLAS

30 SALES CHICAGO

40 OPERATIONS BOSTON

Sql> create table dept1 as select \* from dept;

Sql> insert into dept1 values(1,'NARASIMHA','YSR');

Sql> select \* from dept1; (Source Table)

**DEPTNO** **DNAME** **LOC**

10 ACCOUNTING NEW YORK

20 RESEARCH DALLAS

30 SALES CHICAGO

40 OPERATIONS BOSTON

1 NARASIMHA YSR

Sql> merge into dept D using dept1 N

on(D.deptno=N.deptno)

when matched then

update set D.dname=N.dname,D.loc=N.loc

when not matched then

insert(D.deptno,D.dname,D.loc)values(N.deptno,N.dname,N.loc);

sql>select \* from dept;

**Chapter-8**

**VIEWS**

View is a database object which is used to provide authority level of security. View does not store data. View is also called as “VIRTUAL TABLE” (or) “WINDOW OF THE TABLE”.

Generally views are created from base table’s database administrator. Generally in all database we want to restrict table columns from one user into another user then also we are using views. Generally views created base tables based on base tables. Views are categories into TWO views.

1. Simple View

2. Complex View (or) Join View

SIMPLE VIEW is a view which is created from only **one** base table. Where as COMPLEX VIEW is a view which is created from **number of** base tables.

1. **Simple View:**

**Syntax:** create or replace view viewname

As

Select statement;

Sql> create or replace view v1

as

select \* from emp where deptno=10;

**Error:** insufficient privileges

Sql>conn sys as sysdba;

Enter password: sys

Sql>grant create any view to SCOTT;

Sql>conn scott/tiger;

Sql> create or replace view v1

as

select \* from emp where deptno=10;

sql>select \* from v1;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7782 CLARK MANAGER 7839 09-JUN-81 2450 10

7839 KING PRESIDENT 17-NOV-81 5000 10

7934 MILLER CLERK 7782 23-JAN-82 1300 10

**DML operations through simple view to base table:**

1. When a simple view contains Group functions, Group by clause, Distinct, Rownum, Set operator, Joins then we cannot perform DML operations through simple view to base table.
2. We must include base table not null column into the view then only we can perform INSERT operation through simple view to base table.

**Ex-1:** Sql> create or replace view v1

as

select \* from emp where deptno=10;

Sql>select \* from v1;

Sql>insert into v1(empno,ename,deptno)values(1,’NARASIMHA’,30);

Sql>select \* from emp;

**Ex-2:** Sql> create or replace view v2

as

select ename,job,sal,deptno from emp where deptno=10;

sql>select \* from v2;

**ENAME** **JOB** **SAL** **DEPTNO**

CLARK MANAGER 2450 10

KING PRESIDENT 5000 10

MILLER CLERK 1300 10

SQL> insert into v2(ename,job,sal,deptno)values('SIMHA','MANAGER',20000,20);

**ERROR:** Cannot insert NULL into ("SCOTT"."EMP"."EMPNO")

**NOTE:** Primary key include in views then insert otherwise not inserted data.

Sql> update v2 set ename=upper(ename);

Generally when we are creating views then automatically database server storage VIEW definition in database. In oracle if we want to VIEW definition then we are using USER\_VIEWS data dictionary.

Sql> desc USER\_VIEWS;

Sql> select TEXT from USER\_VIEWS where VIEW\_NAME='V1';

**TEXT**

select "EMPNO","ENAME","JOB","MGR","HIREDATE","SAL","COMM", "DEPTNO" from emp where deptno=10;

**NOTE:**

Whenever we are using group functions or expressions in simple view then we must create column aliasname for those expressions otherwise oracle server given error.

Sql>create or replace view v3

As

Select deptno,max(sal),min(sal) from emp group by deptno;

**ERROR:** must name this expression with a column alias

Sql> create or replace view v3

As

Select deptno,max(sal) maxsal,min(sal) minsal from emp group by deptno;

Sql> select \* from v3;

**DEPTNO** **MAXSAL** **MINSAL**

30 2850 950

20 3000 800

10 5000 1300

Sql> create or replace view v3 (deptno,maxsal,minsal)

As

Select deptno,max(sal),min(sal) from emp group by deptno;

Sql> select \* from v3;

**DEPTNO** **MAXSAL** **MINSAL**

30 2850 950

20 3000 800

10 5000 1300

Sql> create or replace view v4

as

select rownum,deptno from emp where deptno=10;

**ERROR:** must name this expression with a column alias

Sql> create or replace view v4

as

select rownum “rownum”,deptno from emp where deptno=10;

Sql> select \* from v4;

**rownum** **DEPTNO**

1 10

2 10

3 10

In all database systems whenever we are creating a view then automatically view definition are stored in database. If we want view definition then we must use USER\_VIEWS data dictionary.

**With check option:**

If we want to create constraint type mechanism on views then only we are using WITH CHECK OPTION clause. Then we are using this clause only INSERT. This clause visibility values through base table.

**Syntax:** create or replace view viewname

As

Select \* from tablename

Where condition with check option;

Sql> create or replace view v5

as

select \* from emp where deptno=10 **with check option**;

Sql> select \* from v5;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7782 CLARK MANAGER 7839 09-JUN-81 2450 10

7839 KING PRESIDENT 17-NOV-81 5000 10

7934 MILLER CLERK 7782 23-JAN-82 1300 10

Sql> insert into v5(empno,ename,deptno)values(11,'SIMHA',30);

**ERROR:** View WITH CHECK OPTION where-clause violation

Sql>insert into v5(empno,ename,deptno)values(11,’SIMHA’,10);

1 Row Inserted

(Here **30** not in v5 that’s why not inserted **10** are in v5 that’s why inserting)

**FORCE VIEW:** We can also create view without base table these types of view are called as “FORCE VIEW”.

**Syntax:** create or replace force view viewname

As

Select \* from tablename;

**Ex:** Sql>create or replace force view v6

as

select \* from fan;

**Warning:** View created with compilation errors.

Sql> create table fan(sno number(5));

Sql> alter view v6 compile;

Sql> desc v6;

**Name** **Null?** **Type**

SNO NUMBER(5)

**2. Complex View (or) Join View:**

Sql> create or replace force view v7

as

select ename,sal,dname,loc from emp,dept

where emp.deptno=dept.deptno;

Sql> select \* from v7;

**ENAME SAL DNAME LOC**

SMITH 800 RESEARCH DALLAS

ALLEN 1600 SALES CHICAGO

JONES 2975 RESEARCH DALLAS

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

Whenever we are try to performance DML operator through compare view table some tables column are affected some table column are not affected. Generally we can not modify non-key preserved table columns. Complex views are created multiple base tables.

**DML operations on complex views:**

Sql>update v1 set ename=’abc’ where ename=’SMITH’;

Sql>update v1 set dname=’xyz’ where dname=’SALES’;

**ERROR:** Can not modify a column which map to a non-key preserved table.

In oracle if we want to view modify table, non modify table then we using USER\_UPDATABLE\_COLUMNS data dictionary.

**Ex:** Sql>desc

Sql>select column\_name,updatable from user\_updatable\_columns

where table\_name='V7';

**COLUMN\_NAME** **UPD**

ENAME YES

SAL YES

DNAME NO

LOC NO

Whenever we are performing DML operation through complex view to base table some table column are not affected to overcome this problem oracle introduced instead of triggers in pl/sql. Instead of trigger are created on views by default instead of triggers are row level trigger.

**TRIGGERS (PL/SQL):**

Triggers are also same as stored procedure and it will automatically invoked whenever DML operation performed on Table (or) View. All database system having two types of triggers.

1. Statement Level Trigger

2. Row Level Trigger

In statement level triggers trigger body is executed once for DML statement where as in row level triggers trigger body executed for each row for DML statement.

**Syntax:** create or replace trigger triggername

Trigger

Specification

Before/after insert/delete/update on tablename

[for each row]

Begin

Trigger

Body

--------

--------

End;

**Difference b/w Statement Level, Row Level Triggers:**

Sql>create test(col1 date);

Statement Level Trigger:

Sql> create or replace trigger sl

after update on emp

begin

insert into test values(sysdate);

end;

**Testing:** Sql> update emp set sal=sal+100 where deptno=10;

Sql> select \* from test;

**COL1**

05-SEP-14

Row Level Trigger:

Sql> create or replace trigger rl

after update on emp

for each row

begin

insert into test values(sysdate);

end;

**Testing:** Sql> update emp set sal=sal+100 where deptno=10;

Sql> select \* from test;

**COL1**

05-SEP-14

05-SEP-14

05-SEP-14

05-SEP-14

**Row Level Trigger:**

In row level trigger body executed for each row for done. That way we must use for each row clause in trigger specification and also data internally stored in to rollback segment qualifiers. These are qualifiers in trigger body when we are using this qualifier trigger body then we must use [**:]** then we must qualifier trigger.

**Syntax: :**old.colname

**:**new.colname

delete

update

insert

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

NEW

OLD

**Q) Write a pl/sql row level trigger on emp table whenever user deleting data those deleted that data stored in another?**

Sql>create table backup as select \* from emp where **1=2**;

Sql>desc backup; here any false condition

Sql>select \* from backup;

Sql> delete from emp where sal>2000; 6 rows deleted

Sql> select \* from backup;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7566 JONES MANAGER 7839 02-APR-81 2975 20

7698 BLAKE MANAGER 7839 01-MAY-81 2850 30

7782 CLARK MANAGER 7839 09-JUN-81 2450 10

7788 SCOTT ANALYST 7566 19-APR-87 3000 20

7839 KING PRESIDENT 17-NOV-81 5000 10

7902 FORD ANALYST 7566 03-DEC-81 3000 20

6 rows selected

Sql> select \* from emp;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7369 SMITH CLERK 7902 17-DEC-80 800 20

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300 30

7521 WARD SALESMAN 7698 22-FEB-81 1250 500 30

7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400 30

7844 TURNER SALESMAN 7698 08-SEP-81 1500 0 30

7876 ADAMS CLERK 7788 23-MAY-87 1100 20

7900 JAMES CLERK 7698 03-DEC-81 950 30

7934 MILLER CLERK 7782 23-JAN-82 1300 10

8 rows selected

**INSTEAD OF TRIGGER:**

Oracle 8i introduced instead of triggers. Instead of triggers are created on views. By default instead of triggers are row level triggers.

Generally we cannot performance DML operations through complex view to base table to overcome this problem oracle introduced instead of triggers.

**Syntax:** create or replace trigger triggername

Instead of insert/delete/update on viewname

For each row

Begin

---------

---------

End;

**Solution (using instead of trigger):**

Sql> create or replace trigger t1

instead of update on v7

for each row

begin

update dept set dname=:new.dname where dname=:old.dname;

update dept set loc=:new.loc where loc=:old.loc;

end;

**Testing:** Sql>update v7 set dname='GNSR' where dname='SALES';

Sql> select column\_name,updatable

from user\_updatable\_columns where table\_name='V7';

**COLUMN\_NAME** **UPD**

ENAME YES

SAL YES

DNAME YES

LOC YES

**MATERIALIZED VIEW:**

Oracle 8i introduced materialized views. Generally views are **does not store data** where as Materialized Views are **store data**. Generally MV is handling by database administrator. MV is used to **improve performance** of the Joins (or) Aggregatable queries. MV stores result of the query. MV is store replication of the remote database into local server (local node).

Materialized view also store’s data same like a table. But whenever we are refreshing MV if synchronous of data based on base table.

**Syntax:** create materialized view viewname

As

Select statement;

Before we are creating materialized view database administrator must give create any privileges otherwise oracle server returns an **Error: Insufficient Privileges.**

**Syntax:** Sql>grant create any materialized view to username;

**Ex:** Sql> create materialized view mv1

as

select \* from emp;

**ERROR:** Insufficient Privileges

Sql> conn sys as sysdba;

Enter password: SYS

Sql> grant create any materialized view to SCOTT;

Sql> conn scott/tiger;

Sql> create materialized view mv1

as

select \* from emp;

**NOTE:** In oracle one of the materialized view base table must have primary key. Otherwise oracle server returns on error.

Sql> create table test(sno number(10),name varchar2(15));

Sql> create materialized view mv2

as

select \* from test;

**ERROR:** Table 'TEST' does not contain a primary key constraint

**Difference b/w Views, Materialized Views:**

Sql>create table base(sno number(5) primary key,varchar2(15));

Sql>insert into base values(…………………….);

Sql>select \* from base;

**SNO** **NAME**

1 a

2 b

3 c

4 d

Sql> create or replace view v8

as

select \* from base;

Sql> create materialized view mv3

as

select \* from base;

Here materialized view also behaves likes a view. Whenever we are creating materialized view automatically MV definition permanently stores in a database same like a view definitions. In oracle if we want MV definition then we are using USER\_MVIEWS data dictionary.

Sql>desc USER\_MVIEWS;

Sql>select query from user\_mviews where mview\_name='MV3';

**QUERY**

SELECT "BASE"."SNO" "SNO","BASE"."NAME" "NAME" FROM "BASE" "BASE"

Sql> select rowid,sno,name from base;

**ROWID** **SNO** **NAME**

AAADwEAAEAAAAFPAAA 1 a

AAADwEAAEAAAAFPAAB 2 b

AAADwEAAEAAAAFPAAC 3 c

AAADwEAAEAAAAFPAAD 4 d

Sql> select rowid,sno,name from v8;

**ROWID** **SNO** **NAME**

AAADwEAAEAAAAFPAAA 1 a

AAADwEAAEAAAAFPAAB 2 b

AAADwEAAEAAAAFPAAC 3 c

AAADwEAAEAAAAFPAAD 4 d

Here view rowid’s are same as base table rowid’s. That why view does not store data. Through the view we are viewing base table that’s why view are called as “VIRTUAL TABLE”.

Sql>select rowid,sno,name from mv3;

**ROWID** **SNO** **NAME**

AAADwHAAEAAAAFcAAA 1 a

AAADwHAAEAAAAFcAAB 2 b

AAADwHAAEAAAAFcAAC 3 c

AAADwHAAEAAAAFcAAD 4 d

Here MV rowid’s are called different from base table rowid’s. that’s why materialized view store data.

Sql> update base set name=upper(name);

Sql> select \* from v8;

**SNO** **NAME**

1 A

2 B here affected that’s why capital letters

3 C

4 D

Sql> select \* from mv3;

**SNO** **NAME**

1 a

2 b here not affected that’s why small letters

3 c

4 d

Sql> select \* from base;

**SNO** **NAME**

1 A

2 B

3 C

4 D

When we are refreshing MV it synchronous data based on base table. In oracle if we want to refresh MV then we are using **refresh** method from **DBMS\_MVIEW** package.

**Syntax:** dbms\_mview.refresh(‘materialized viewname’);

Sql> EXEC DBMS\_MVIEW**.**REFRESH('MV3');

Sql>select \* from mv3;

**SNO** **NAME**

1 A

2 B

3 C

4 D

Oracle having two types of Materialized Views

1. Complete Refresh Materialized View

2. Fast Refresh Materialized View

**1. Complete Refresh Materialized View:**

In oracle by default MV’s are complete materialized view. CRMV when we are refreshing internally rowid’s are Re-Created. This process does not improve more performance. If we are called modifying data in base table also that why when we are refreshing MV frequently then this type MV are ‘D’ grade performance.

**Syntax:** create materialized view viewname

Refresh complete

As

Select statement;

Sql>create materialized view mv4

refresh complete

as

select ename,job,sal,deptno from emp where deptno=10;

Sql>EXEC DBMS\_MVIEW.REFRESH('MV4');

Sql> select rowid,sal,deptno from mv4;

**ROWID** **SAL** **DEPTNO**

AAADwUAAEAAAAFsAAD 2450 10

AAADwUAAEAAAAFsAAE 5000 10

AAADwUAAEAAAAFsAAF 1300 10

Sql> EXEC DBMS\_MVIEW.REFRESH('MV4');

Sql> select rowid,sal,deptno from mv4;

**ROWID** **SAL** **DEPTNO**

AAADwUAAEAAAAFsAAA 2450 10

AAADwUAAEAAAAFsAAB 5000 10

AAADwUAAEAAAAFsAAC 1300 10 (here rowid’s are changed)

**2. Fast Refresh Materialized View:**

Fast refresh materialized view are also called as “Incremental Refresh Materialized View”. When we are refreshing FRMV it will **improve performance** of the MV’s. In FRMV **rowid are Not Changed** if you are refreshing MV no.of times also. In FRMV data only synchronous.

**Syntax:** create materialized view viewname

Refresh fast

As

Select statement;

Before we are creating FRMV database administrator must create MV log on base table. These MV create log stores states of the base table into appropriate data dictionary.

**Syntax:** create materialized view **log on** basetablename;

**Ex:** sql> create materialized view log on base;

Sql> create materialized view mv5

refresh fast

as

select \* from base;

Sql> select \* from mv5;

**SNO** **NAME**

1 A

2 B

3 C

4 D

Sql> EXEC DBMS\_MVIEW.REFRESH('MV5');

Sql> select rowid,sno,name from mv5;

**ROWID** **SNO** **NAME**

AAADwYAAEAAAAF8AAA 1 A

AAADwYAAEAAAAF8AAB 2 B

AAADwYAAEAAAAF8AAC 3 C

AAADwYAAEAAAAF8AAD 4 D

Sql> EXEC DBMS\_MVIEW.REFRESH('MV5');

Sql> select rowid,sno,name from mv5;

**ROWID** **SNO** **NAME**

AAADwYAAEAAAAF8AAA 1 A

AAADwYAAEAAAAF8AAB 2 B

AAADwYAAEAAAAF8AAC 3 C

AAADwYAAEAAAAF8AAD 4 D (Here rowid’s are not changed)

**ON DEMAND/ON COMMIT:**

Generally refresh materialized views are two ways

1. Manually

2. Automatically

**1. Manually:**

We can also refresh materialized views either Manually (or) Automatically. In manually method we are using dbms\_mview package to refresh MV these method is called as “On Demand”. By default method is On Demand method.

**2. Automatically:**

If you want to refresh MV without using dbms\_mview package this is called as “On Commit”. When we are using On Commit method in this case explicitly we must specify On Commit key word clause in MV.

**Syntax:** create materialized view viewname

**Refresh complete**/refresh fast **on demand**/on commit

As default

Select statement;

Sql> select \* from base;

**SNO** **NAME**

1 A

2 B

3 C

4 D

Sql> create materialized view mv6

refresh fast on commit

as

select \* from base;

Sql> select rowid,sno,name from mv6;

**ROWID** **SNO** **NAME**

AAADwbAAEAAAAGMAAA 1 A

AAADwbAAEAAAAGMAAB 2 B

AAADwbAAEAAAAGMAAC 3 C

AAADwbAAEAAAAGMAAD 4 D

Sql> update base set name='XYZ' where sno=2;

Sql> select rowid,sno,name from base;

**ROWID** **SNO** **NAME**

AAADwEAAEAAAAFPAAA 1 A

AAADwEAAEAAAAFPAAB 2 XYZ

AAADwEAAEAAAAFPAAC 3 C

AAADwEAAEAAAAFPAAD 4 D

Sql> select rowid,sno,name from mv6;

**ROWID** **SNO** **NAME**

AAADwbAAEAAAAGMAAA 1 A

AAADwbAAEAAAAGMAAB 2 B

AAADwbAAEAAAAGMAAC 3 C

AAADwbAAEAAAAGMAAD 4 D

Sql> commit; (Here explicitly give commit after affected data)

Sql> select rowid,sno,name from mv6;

**ROWID** **SNO** **NAME**

AAADwbAAEAAAAGMAAA 1 A

AAADwbAAEAAAAGMAAB 2 XYZ

AAADwbAAEAAAAGMAAC 3 C

AAADwbAAEAAAAGMAAD 4 D

**View**

1. It does not store data.

2. Security purpose

3. Hiding some information from the user i.e it is also called “Window Of The Table”.

4. When we are modifying data in base table then view is immediately affected.

**Materialized View**

1. It store data.

2. Improve performance purpose.

3. It stores result of the query i.e it is called as “Copy Of The Table”.

4. When we are modifying data in base table then MV not affected immediately because here rowid’s are different.

**Chapter-9**

**DCL (Data Control Language)**

DCL are two types these are

1. GRANT

2. REVOKE

**1. Grant:**

Sql GRANT is a command used to provide access or privileges on the database objects to the users.

Data security point of view oracle provided two types of admin user’s these are

1. SYS

2. SYSTEM

**Creating a user:**

**Syntax:** create user username identified by password;

**Syntax:** grant connect,resource to username;

**Syntax:** conn username/password;

**Ex:** sql>conn sys as sysdba;

Enter password: SYS

Sql> create user NARASIMHA identified by NARASIMHA;

Sql>grant connect,resource to NARASIMHA; ---🡪connect,resource(dba)

Sql> conn NARASIMHA/NARASIMHA;

Sql> select \* from emp;

**Error:** table or view does not exit

Sql> conn scott/tiger;

Sql> grant all on emp to NARASIMHA;

Sql> select \* from emp;

**Error:** table or view does not exit

Sql> select \* from scott.emp;

Sql> conn narasimha/narasimha;

Sql> create synonym gnsr for scott.emp;

**ERROR:** insufficient privileges

Sql> conn sys as sysdba;

Enter password: SYS

Sql> grant create any synonym to NARASIMHA;

Sql> conn narasimha/narasimha;

Sql> create synonym gnsr for scott.emp;

Sql> select \* from gnsr;

**PRIVILEGES:**

Privilege is an write given to the user to perform action to the database.

**Ex:** Grant table,create procedure

Data security point of view two types of privileges

1. System Privileges

2. Object Privileges

**1. System Privilege:**

System privileges are given by database administrator oracle having to system privilege. System privileges are used to create object, alter table, create procedure, create trigger, create any materialized, create session …………………etc.

**Syntax:** grant system privileges to username1,username2……………;

**Ex:** Sql> conn sys as sysdba;

Enter password: SYS

Sql> grant create table,create procedure,create any materialized view,create trigger,

create session,create synonym to scott,narasimha;

**ROLE:**

Role is nothing but collection of system privileges (or) collection of object privileges. Userdefined role are create by database administrator only.

Generally in multi user environment number of user’s works on same project in this case some user require common set of privileges in this case only database administrator creating a userdefined role and assigns common set of privileges in to role and then only that role given to the number of user’s.

**2 (SYSTEM PRIVILEGES) 1 (ROLE) 3 (USER’S)**

User-1

User-2

User-3

User-4

User-5

**Creating a userdefined ROLE:**

Connect, Resource, Dba ----🡪ROLES

Step-1:- (create a role)

**Syntax:** create role rolename;

Step-2:- (assign system privileges to role)

**Syntax:** grant system privileges to rolename;

Step-3:- (assign role to no.of users)

**Syntax:** grant rolename to username1,username2 ……………..;

Sql> conn sys as sysdba;

Enter password: SYS

Sql> create role r1;

Sql>grant create table,create procedure,create any materialized view,create trigger,create session,create synonym to r1;

In oracle system privileges related to role stored under ROLE\_SYS\_PRIVS data dictionary.

Sql> desc role\_sys\_privs;

**Name** **Null?** **Type**

ROLE NOT NULL VARCHAR2(30)

PRIVILEGE NOT NULL VARCHAR2(40)

ADMIN\_OPTION VARCHAR2(3)

Sql> select role,privilege from role\_sys\_privs where role='R1';

**ROLE** **PRIVILEGE**

R1 CREATE SESSION

R1 CREATE TABLE

R1 CREATE SYNONYM

R1 CREATE TRIGGER

R1 CREATE ANY MATERIALIZED VIEW

R1 CREATE PROCEDURE

**PREDEFINED ROLE:**

Whenever we are installing server then automatically **three ROLES** are created. This are called as “Predefined Roles”.

1. CONNECT -------------🡪 End User

2. RESOURCE -------------🡪 Developer

3. DBA -------------🡪 Database Administrator

Sql>conn sys as sysdba;

Enter password: SYS

Sql>desc role\_sys\_privs;

Sql>select role,privilege from role\_sys\_privs where role in('connect','resource');

Sql>select role,privilege from role\_sys\_privs where role in('dba');

**2. Object Privileges:**

Object privileges are used to performance. Some privileges object privileges is given by either developers (or) dba. In oracle insert, update, delete, select, execute, read, write,………. this are also called as “ALL” (or) “OBJECT PRIVILEGES”.

**Syntax:** grant object privilege on objectname to username/rolename/public;

Sql> grant all on emp to narasimha,r1;

Sql> grant all on emp to public;

Sql> desc user\_tab\_privs;

**WITH GRANT OPTION:**

Who receive with grant option clause those users given same object privileges to another user’s.

**Syntax:** grant object privilege on objectname to username/public with grant option;

Sql>grant all on emp to narasimha with grant option;

Sql>grant all on emp to r1 with grant option;

**Error:** Cannot GRANT to a role WITH GRANT OPTION

Sql>grant all on emp to public with grant option;

Sql>grant all on emp to narasimha,public with grant option;

**NOTE:**

In all database systems ROLES does not work WITH GRANT OPTION clause. All object privileges related to user stored under USER\_TAB\_PRIVS data dictionary.

Sql>desc user\_tab\_privs;

**2. REVOKE:**

REVOKE command is used to cancel either system (or) object privileges.

**Syntax:** revoke system privilege from username1,username2,……………..;

Revoke object privilege on objectname from username/rolename/public;

**NOTE:**

Generally if we want to restrict table columns from one user into another user then we creating view with required column’s then only that views given to the no.of users.

**Syntax:** grant all on viewname to username1,username2,………………..;

Sql> create or replace view v9

as

select empno,ename,sal,deptno from emp where deptno=10;

Sql> select \* from v9;

**EMPNO** **ENAME** **SAL** **DEPTNO**

7782 CLARK 2450 10

7839 KING 5000 10

7934 MILLER 1300 10

Sql> grant all on v9 to narasimha;

Sql> conn narasimha/narasimha;

Sql> select \* from scott.v9;

**EMPNO** **ENAME** **SAL** **DEPTNO**

7782 CLARK 2450 10

7839 KING 5000 10

7934 MILLER 1300 10

**NOTE:**

We can also drop view using drop view viewname. Whenever we are dropping base table then view is invalid and also view definition are available in database in oracle. If you want view definition then we are using USER\_VIEWS data dictionary.

Sql> drop table base;

Sql> select \* from v8;

**ERROR:** view "SCOTT.V8" has errors

Sql> desc user\_views;

Sql> select text from user\_views where view\_name='V8';

**TEXT**

select "SNO","NAME" from base

**READ ONLY VIEW:**

We can also create read only view using with read only option clause. Whenever we are using this clause we cannot perform DML operation view to base table.

**Syntax:** create or replace view viewname

As

Select \* from tablename where condition with read only;

Sql> create or replace view v10

as

select \* from emp where deptno=10 with read only;

Sql> select \* from v10;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7782 CLARK MANAGER 7839 09-JUN-81 2450 10

7839 KING PRESIDENT 17-NOV-81 5000 10

7934 MILLER CLERK 7782 23-JAN-82 1300 10

Sql> delete from v10 where deptno=10;

**ERROR:** cannot delete from view without exactly one key-preserved table

**Chapter-10**

**INDEX**

Oracle 8i introduced indexes. INDEX is a database object which is used retrieves data fast from the database. This process automatically improves performance of the application. Become INDEX columns retrieve data very fastly from the table. Generally INDEX created on table columns. INDEX is handling by database administrator.

**Index created two ways:**

1. Automatically

2. Manually

**1. Automatically:**

In oracle whenever we are creating unique key (or) primary key then oracle server automatically creates BTREE indexes on those columns.

**2. Manually:**

We can also creates INDEX explicitly by using create INDEX method.

**Syntax:** create index indexname on tablename(colname);

Whenever we are using select query contain WHERE clause (or) ORDER BY clause then only oracle server searching for INDEX in the database. If INDEX are available oracle server retrieve data very fastly the table by using INDEX SCAN() method. If INDEX not available then oracle server uses full table scan method to retrieve data from the table.

**NOTE:**

Whenever WHERE clause contain <> (or) is null (or) is not null operators then oracle server does not search for INDEX if those column already having INDEX also.

**Oracle having two types of index:**

1. Btree Index

2. Bitmap Index

High cardinality bitmap low cardinality

1. **Btree Index:**

Btree index are created on high cardinality columns. In oracle by default index are btree index whenever we are creating btree index automatically oracle server stores actual data along with rowid’s in leaf blocks of the btree structure. Whenever WHERE condition value available in leaf blocks is create some retrieve data very fastly from the table along index scan method.

**DEVALOPER** **DBA**

Sql> select \* from emp where ename=’KING’; Sql> select \* from emp where ename=’KIMG’;

A ---------Z

Root

Oracle database

K----------Z

A-----------J

Branch

Data Dictionary

Rowid’s King

Rowid’s Scott

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

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

Rowid’s Allen

Rowid’s Blake

------ -----

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

Ename index1 normal

Leaf blocks

If we want view performance of the query we must create **plan** table using **explain plan** clause.

**Syntax:** explain plan for select statement;

If we want view **plan** table then we must use display method from DBMS\_XPLAN method.

Sql> select \* from table(dbms\_xplan.display());

**Without using Index:**

Sql> select \* from emp where ename=’KING’;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7839 KING PRESIDENT 17-NOV-81 5000 10

Sql> explain plan for select \* from emp where ename=’KING’;

Sql> select \* from table(dbms\_xplan.display());

**PLAN\_TABLE\_OUTPUT**

| Id | Operation | Name | Rows | Bytes | **Cost (%CPU)** |Time |

| 0 | SELECT STATEMENT | | 1 | 37 | **3 (0)** | 00:00:01 |

|\* 1 | TABLE ACCESS FULL | EMP | 1 | 37 | **3 (0)** | 00:00:01 |

**Using index:**

Sql> create index index1 on emp(ename);

Sql> select \* from emp where ename=’KING’;

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7839 KING PRESIDENT 17-NOV-81 5000 10

Sql> explain plan for select \* from emp where ename=’KING’;

Sql> select \* from table(dbms\_xplan.display());

**PLAN\_TABLE\_OUTPUT**

| Id | Operation | Name | Rows | Bytes | **Cost (%CPU)**| Time |

| 0 | SELECT STATEMENT | | 1 | 37 | **2 (0)** | 00:00:01 |

| 1 | TABLE ACCESS BY INDEX ROWID | EMP | 1 | 37 | **2 (0)** | 00:00:01 |

|\* 2 | INDEX RANGE SCAN | INDEX1 | 1 | **| 1 (0)**  | 00:00:01 |

All index information stored under USER\_INDEXES data dictionary.

Sql> desc user\_indexes;

Sql> select index\_name,index\_type from user\_indexes

where table\_name='EMP';

**INDEX\_NAME INDEX\_TYPE**

PK\_EMP NORMAL --------------------🡪 Automatically

INDEX1 NORMAL --------------------🡪 Manually

**NOTE:**

If we want view column names along with index name then we are using **user\_ind\_columns** data dictionary.

Sql> desc user\_ind\_columns;

Sql> select COLUMN\_NAME,INDEX\_NAME from user\_ind\_columns

where table\_name='EMP';

**COLUMN\_NAME** **INDEX\_NA**ME

EMPNO PK\_EMP

ENAME INDEX1

**FUNCTION BASED INDEX:**

Oracle 8i introduced function based index. Function based index are by default btree index. Generally whenever **WHERE** clause contain function (or) expressions then oracle server **does not search for index** if already those columns having also to overcome this problem oracle 8i introduced extension of the btree index this are also called as “function based index”. In function based column along with function (or) expression index.

**Syntax:** create index indexname on tablename(function/expression(colname));

Sql> select \* from emp where upper(ename)='KING';

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7839 KING PRESIDENT 17-NOV-81 5000 10

Sql> explain plan for select \* from emp where upper(ename)='KING';

Sql> select \* from table(dbms\_xplan.display());

**PLAN\_TABLE\_OUTPUT**

| Id | Operation | Name | Rows | Bytes | **Cost (%CPU)** |Time |

| 0 | SELECT STATEMENT | | 1 | 37 | **3 (0)** | 00:00:01 |

|\* 1 | TABLE ACCESS FULL | EMP | 1 | 37 | **3 (0)** | 00:00:01 |

Sql> create index index2 on emp (upper(ename));

Sql> explain plan for select \* from emp where upper(ename)='KING';

Sql> select \* from table(dbms\_xplan.display());

**PLAN\_TABLE\_OUTPUT**

| Id | Operation | Name | Rows | Bytes | **Cost (%CPU)**| Time |

| 0 | SELECT STATEMENT | | 1 | 37 | **2 (0)** | 00:00:01 |

| 1 | TABLE ACCESS BY INDEX ROWID | EMP | 1 | 37 | **2 (0)** | 00:00:01 |

|\* 2 | INDEX RANGE SCAN | INDEX2 | 1 | **| 1 (0)**  | 00:00:01 |

Sql> select index\_name,index\_type from user\_indexes where table\_name='EMP';

**INDEX\_NAME INDEX\_TYPE**

PK\_EMP NORMAL

INDEX1 NORMAL

INDEX2 FUNCTION-BASED NORMAL

**VIRTUAL COLUMN:**

Oracle 11g introduced virtual columns using virtual column we can store expressions directly into database. Generally in oracle if we want to stored expressions then we are using either function based index (or) view these two methods in directly stores stored expression in expression to overcome this problem oracle 11g introduced virtual column to stores stored expression directly into database using generated always as.

**Syntax:** columnname datatype(size) generated always

as(function/expression(colname);

Sql> create table t1(a number(5),b number(5),c number(10));

Sql> insert into t1(a,b)values(10,20);

Sql> select \* from t1;

**A B C**

1020 30

**NOTE:**

If you want view virtual column expression then we are using DATA\_DEFAULT property from USER\_TAB\_COLUMNS data dictionary.

Sql> desc user\_tab\_columns;

Sql> select column\_name,data\_default from user\_tab\_columns where table\_name=’T1’;

**COLUMN\_NAME DATA\_DEFAULT**

C “Á+B”

**Oracle having two btree indexes:**

1. Unique Btree Index

2. Non-Unique Btree Index

In oracle by default pk,unique key constrained columns are unique btree index and to give explicitly this are non-unique btree index. We can also create unique btree index explicitly by using following index.

**Syntax:** create unique index indexname on tablename(colname);

We are not allowed to create unique index on duplicate value columns.

Sql> create unique index index3 on emp(ename);

Sql> select index\_name,index\_type from user\_indexes where table\_name='EMP';

**INDEX\_NAME INDEX\_TYPE**

PK\_EMP NORMAL

INDEX3 NORMAL

Sql> create unique index index4 on emp(job)

**ERROR:** cannot CREATE UNIQUE INDEX; duplicate keys found

**2. Bitmap:**

Oracle 7.3 introduced bitmap index. Generally bitmap index are used in data warehousing application. Bitmap index are created in **low cardinality** columns (more duplicate data) low cardinality columns more duplicate data.

**Syntax:** create bitmap index indexname on tablename(colname);

Sql> create bitmap index index4 on emp(job);

Whenever we are creating bitmap index oracle server automatically create a bitmap table on the specified columns and using bits. Whenever user requesting using logical operator. Then only oracle server directly operates bits within bitmap table and also resultant bitmap automatically convert into rowid using internal bitmap function.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **JOB** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| CLERK | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| SALESMAN | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| MANAGER | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ANALYST | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| PRESIDENT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**Chapter-11**

**SEQUENCES**

Sequences are database object which is used to generate **sequence number** automatically. Generally sequence is used to generate primary key values automatically. Sequence is an independent database object. Generally database administrator creates sequence. Once sequence access that sequence.

**Syntax:** create sequence sequencename

Start with n

Increment by n

Minvalue n

Maxvalue n

Cycle/nocycle

Cache/nocache;

If you want access sequence values then we are using 2 pseudo columns.

1. Currval

2. Nextval

**Syntax:** sequencename.currval

**Syntax:** sequencename.nextval

If you want generate sequence values using select statement then we must use DUAL table.

**Syntax:** select sequencename.currval from dual;

**Syntax:** select sequencename.nextval from dual;

If you want generate first sequence numbers then we must nextval pseudo column. In this case nextval pseudo column returns starting sequence number currval pseudo column return current value of the sequence session already having a value.

Sql> create sequence s1

start with 5

increment by 2

maxvalue 15;

Sql> select s1.currval from dual;

**ERROR:** sequence S1.CURRVAL is not yet defined in this session

Sql> select s1.nextval from dual;

**NEXTVAL**

5

Sql> **/**

**NEXTVAL**

7

Sql> select s1.currval from dual;

**CURRVAL**

7

**NOTE:**

We can also return currval along with nextval pseudo then we are generating first sequence value.

Sql> create sequence s2

start with 1

increment by 1

maxvalue 10;

Sql> select s2.nextval from dual;

**NEXTVAL**

1

Sql> **/**

**NEXTVAL**

2

Sql> select s2.nextval,s2.currval from dual;

**NEXTVAL** **CURRVAL**

3 3

**NOTE:**

Sequence are used in insert,update,delete,select statements. We can also modify sequence property using ALTER but we can not starting with sequence number (start with value).

**Syntax:** alter sequence sequencename

Propertyname newvalue;

Sql> alter sequence s2

Increment by -1;

**Sequence alter**

Sql> alter sequence s2

Start with 4

Maxvalue 10;

**ERROR:** cannot alter starting sequence number

**NOTE:**

Whenever we decrement the sequence we cannot instantiate below minimum value and also start with cannot be less minimum value.

Sql> create sequence s3

Start with 3

Increment by 1

Minvalue 5

Maxvalue 10;

**Error:** START WITH cannot be less than MINVALUE

**CYCLE/NOCYCLE:**

When we are defined cycle whenever reach value max value next values start with starting value.

**Nocycle:** Sql> create sequence s3

start with 6

increment by 1

minvalue 6

maxvalue 10

nocycle

nocache;

Sql> select s3.nextval from dual;

**NEXTVAL**

6

Sql> /

**NEXTVAL**

7

Sql> /

**NEXTVAL**

8

Sql> /

**NEXTVAL**

9

Sql> /

**NEXTVAL**

10

Sql> /

**ERROR:** sequence S3.NEXTVAL exceeds MAXVALUE and cannot be instantiated

**Cycle:** Sql> create sequence s4

start with 6

increment by 1

minvalue 6

maxvalue 8

cycle

nocache;

Sql> select s4.nextval from dual;

NEXTVAL

6

Sql> /

NEXTVAL

7

Sql> /

NEXTVAL

8

Sql> /

NEXTVAL

6

Sql> /

NEXTVAL

7

Sql> /

NEXTVAL

8

Sql> /

NEXTVAL

6

Sql> select \* from user\_sequences;

**Sequence\_Name Min\_Value Max\_Value Increment\_By C O Cache\_Size Last\_Number**

S2 1 10 -1 N N 20 0

S1 1 15 2 N N 20 17

S3 6 10 1 N N 0 11

S4 6 8 1 Y N 0 7

**CACHE/NOCACHE:**

Cache is an optional clause used in sequence we access sequence number very fastly i.e whenever we defined sequence if we want access then we run set of sequence value in cache memory then only application access sequence values from cache not from the disk that why cache is memory area which stores set of sequence and improve performance that why cache value by defined by database administrators.

**NOTE:**

In oracle by default cache value is 20 and also cache minimum value is 2. Whenever system caches then automatically generates because cache is a memory area whenever system crush cache values are lost.

Sql> create sequence s5

start with 2

increment by 1

cache 1;

**ERROR:** the number of values to CACHE must be greater than 1

**NOTE:**

Whenever we are defining cache value that is less than one cycle when we are defining cycle option otherwise oracle server returns on error to overcome this problem we can also use no cache when no cache when we are defining cycle option. We must create primary key column otherwise not properly inserted data and duplicate value inserting.

Sql> create sequence s5

start with 5

minvalue 4

maxvalue 6

cycle

cache 40;

**ERROR:** number to CACHE must be less than one cycle

Sql> create table test(sno number(5) primary key,name varchar2(15));

Sql> create sequence s5

Start with 1;

Sql> insert into test(sno,name) values(s5.nextval,’&name’);

Enter value for name: ABC

SQL> /

Enter value for name: XYZ

SQL> select \* from test;

**SNO** **NAME**

1 ABC

2 DEF

3 GHI

4 JKL

5 XYZ

All sequence information stored under USER\_SEQUENCES data dictionary.

Sql> desc USER\_SEQUENCES;

To view all sequence table information’s

Sql> select \* from USER\_SEQUENCES;

**Chapter-12**

**NORMALIZATION**

Normalization is a scientific process **decompose table** into number of tables. These processes automatically **reduce redundancy.**

Normalization process automatically avoids insert, delete and update problems. In design place of the SDLC(Software Development Life Cycle) database designers design logical model of the database. In these logical model only database designer uses normalization process through normal forms.

In 1970 E.F CODD had written a paper “**Relational Model Of Data For Shared Data Banks**” in this paper only E.F CODD introduced first three normal forms.

**Normal Forms:**

1. First Normal Form

2. Second Normal Form

3. Third Normal Form

4. BCNF

5. Fourth Normal Form

6. Fifth Normal Form

**1. First Normal Form:**

If a table is in 1NF then that table does not contain multi table i.e every cell contain atomic (single value) and also identifier record uniquely using a key.

Candidate Key

1NF

Not In 1NF

|  |  |  |  |
| --- | --- | --- | --- |
| **Item name** | **Color** | **Price** | **Tax** |
| Marker | Black, red | 20 | 0.2 |
| Pen | Blue, green | 30 | 0.3 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Item name** | **Color** | **Price** | **Tax** |
| Marker | Blake | 20 | 0.2 |
| Marker | Red | 20 | 0.2 |
| Pen | Blue | 30 | 0.3 |
| Pen | Green | 30 | 0.3 |

**Process:**

Identified repeating groups and putting into separate table in more atomic forms by default 1NF process table is an child table because here one column having duplicate data.

**Electronic Shop**

PK

Order no:

Order Master Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Order no** | **Order date** | **Customer**  **Address** | **Customer**  **phone** |
| 1 | - | - | - |
|  |  |  |  |
|  |  |  |  |

Order date:

Customer name:

Customer address:

Customer phone:

1NF

FK

Order item details

|  |  |  |
| --- | --- | --- |
| **Order no** | **Item name** | **Amount** |
| 1 | Refrigerator | 30000 |
| 1 | Led | 40000 |
| 1 | A/C | 50000 |
|  |  |  |

Item names:

Refrigerator

Led

A/C

Add another item

√

Amount:

**SUBMIT**

1. **Second Normal Form:**

A table is a 1NF and also all non-key attribute fully functionally dependent total candidate key.

Generally 1NF deals with atomicity where as 2NF deals with relationship b/w key,non-key attribute. When a table is in 1NF and also table contain partial non-key attribute then that table not in 2NF.

**Process:**

Identified partial non-key attributes which depends on partially key attributes putting into separate table these are called as “**2NF**”. In these tables only all attribute fully dependency. By default this is on master table.

|  |  |  |
| --- | --- | --- |
| **Item name** | **Price** | **Tax** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

key attributes **1NF** p Non-key attributes Fk Pk 2NF Master table

|  |  |
| --- | --- |
| **Item name** | **color** |
|  |  |
|  |  |
|  |  |
|  |  |

**Item name color price tax**

Master black 20 0.2

Master red 20 0.2

Pen blue 30 0.3

Pen green 30 0.3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sno** | **Sname** | **Actvity1** | **Cost1** | **Activity2** | **Cost2** |
| 101 | ABC | CRICKET | $10 | FOOTBALL | $20 |
| 102 | XYZ | GOLF | $30 | SWIMMING | $40 |
| 103 | ZZZ | CRICKET | $10 | GOLF | $30 |
| 104 | AAA | CRICKET | $10 | SWIMMING | $40 |
| 105 | BBB | FOOTBALL | $20 |  |  |

ACTIVITY TABLE

|  |  |
| --- | --- |
| **Sno** | **Sname** |
| 101 | ABC |
| 102 | XYZ |
| 103 | ZZZ |
| 104 | AAA |
| 105 | BBB |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sno** | **Actvity1** | **Cost1** | **Activity2** | **Cost2** |
| 101 | CRICKET | $10 | FOOTBALL | $20 |
| 102 | GOLF | $30 | SWIMMING | $40 |
| 103 | CRICKET | $10 | GOLF | $30 |
| 104 | CRICKET | $10 | SWIMMING | $40 |
| 105 | FOOTBALL | $20 |  |  |

|  |  |  |
| --- | --- | --- |
| **Sno** | **Act** | **cost** |
| 101 | Ckt | $10 |
| 101 | Fb | $20 |
| 102 | Golf | $30 |
| 102 | Swim | $40 |
| 103 | Ckt | $10 |
| 103 | Golf | $30 |
| 104 | Ckt | $10 |
| 104 | Swim | $40 |
| 105 | Fb | $20 |

|  |  |
| --- | --- |
| **Sno** | **Sname** |
| 101 | ABC |
| 102 | XYZ |
| 103 | ZZZ |
| 104 | AAA |
| 105 | BBB |

|  |  |
| --- | --- |
| **Sno** | **Act** |
| 101 | Ckt |
| 101 | Fb |
| 102 | Golf |
| 102 | Swim |
| 103 | Ckt |
| 103 | Golf |
| 104 | Ckt |
| 104 | Swim |
| 105 | Fb |

|  |  |
| --- | --- |
| **Sno** | **Sname** |
| 101 | ABC |
| 102 | XYZ |
| 103 | ZZZ |
| 104 | AAA |
| 105 | BBB |

pk

Non-key attribute

P

2NF (Master table)

Candidate key

**Not in 2NF**

Fk

Fk

Pk

**2NF (Master table)**

Child table

**Act cost**

Ckt $10

Fb $20

Golf $30

Swim $40

**Project table: p p**

Candidate key key attributes non-key attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ecode** | **Projcode** | **Dept** | **Depthead** | **Hours** |
| E101 | P1 | Systems | D1 | 5 |
| E102 | P1 | Sales | D2 | 3 |
| E101 | P2 | Systems | D1 | 6 |
| E103 | P2 | Research | D3 | 4 |
| E102 | P3 | Sales | D2 | 7 |
| E101 | P3 | Systems | D1 | 8 |

F **Phase-1:-**

1. Fd

Ecode hours ( )

Value more than 1 hour

5

6

8

E101

**2. fd 3. fd**

Projcode hours ( 🗶 ) ecode+Projcode hours ( √ )

5

3

P1

**Phase-2:- fd fd**

Ecode dept ( √ ) Projcode dept ( 🗶 )

System

Sales

E101----🡪systems p1

**Phase-3:-**

1. Ecode dept ( √ ) 2. Projcode dept ( 🗶 )

(Master table) (Child table)

Pk

**2NF**

Fk

|  |  |  |
| --- | --- | --- |
| **Ecode** | **Dept** | **Depthead** |
| E101 | Systems | D1 |
| E102 | Sales | D2 |
| E103 | Research | D3 |

|  |  |  |
| --- | --- | --- |
| **Ecode** | **Projcode** | **hours** |
| E101 | P1 | 5 |
| E102 | P1 | 3 |
| E101 | P2 | 6 |
| E103 | P2 | 4 |
| E102 | P3 | 7 |
| E101 | P3 | 8 |

Before normalization process above resource table having insertion, deletion, updation problems occurred.

**Insertion problem:**

In the above resource table we cannot insert particular dept emp’s then we must assigning a projects this is called as “Insertion Problem”.

**Updation problem:**

In the above resource table Ecode,Dept,Depthead attribute values are repeated. Whenever on employee transferred one dept to another dept all this three attributes value modified currently otherwise inconsistency problems occurred this called as “Updation Problem”.

**Deletion problem:**

In the resource table when we are tried to delete particular emp automatically dept delete also delete this is called as “Deletion Problem”.

All this problems are automatically inserting,deleting,updating avoided then we are using normalization process.

**Order form:**

Order no: PK

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Order no** | **Order date** | **Cust name** | **Cust address** | **Cust phone** |
|  |  |  |  |  |

Order date:

Cust name:

Cust address:

Cust phone:

Item name:

Item no:

Candidate key (**not in 2NF**) P P F F

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Order no** | **Item no** | **Item name** | **Amount** | **Quantity** | **Discount** |
|  |  |  |  |  |  |

Quantity: Discount:

Amount:

**SUBMIT**

**Repository items**

Item no:

Item name:

Amount:

Frontend application

|  |  |  |  |
| --- | --- | --- | --- |
| **Order no** | **Item no** | **Quantity** | **Discount** |
| 1  1  1  1 |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Item no** | **Item name** | **amount** |
| 111 | LED | 30000 |
| 112 | A/C | 20000 |
|  |  |  |
|  |  |  |

**Order details table** FK FK

**OK**

PK

**3.** **Third Normal Form:**

If a table in 2NF and also all non-key attributes **only dependent** on candidate key **(or)** primary key. If non-key attributes which depends on another non-key attributes then that table not in 3NF.

**Process:**

Identified non-key attributes which depends on another non-key attributes putting into separate table this is called as “3NF Table”. By default this is an master table in this table only all non-key attributes only dependent on Candidate key (or) Primary key.

PK

Not in 3NF

**2NF Master table**

Candidate key **(not in 2NF)** non-key attribute

|  |  |
| --- | --- |
| **Item no** | **color** |
|  |  |

|  |  |  |
| --- | --- | --- |
| **Item no** | **Price** | **tax** |
|  |  |  |

**1NF** P P

**Item name color price tax**

Master black 20 0.2

Master red 20 0.2

Pen blue 30 0.3

Pen green 30 0.3

2NF

**3NF**

PK

|  |  |
| --- | --- |
| **Item no** | **color** |
|  |  |

|  |  |
| --- | --- |
| **Item no** | **Price** |
|  |  |

|  |  |
| --- | --- |
| **Price** | **Tax** |
|  |  |

PK

**3NF Master table**

**Order form:**

Order no:

PK

Order date:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Order no** | **Order date** | **Cust name** | **Cust address** | **Phone** | **Cust no** |
|  |  |  |  |  |  |

Cust no:

Cust name:

Cust address:

Cust phone:

**Item no:**

Refrigerator

Led tv

A/C

Item name:

PK

**2NF item mater table**

|  |  |  |
| --- | --- | --- |
| **Item no** | **Item name** | **Amount** |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Order no** | **Item no** | **quantity** | **discount** |
|  |  |  |  |

Quantity Discount

FK

**Order details table**

Amount:

**SUBMIT**

**Customer enquiry** PK **3NF mater table** PK **order mater details** FK

Cust no:

Cust name:

Cust address:

Cust phone:

|  |  |  |
| --- | --- | --- |
| **Order no** | **Order date** | **Cust no** |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Cust no** | **Cust name** | **Cust address** | **Cust phone** |
| **OK** |  |  |  |

**Functional Dependency:**

If any given two tuples in a relation **r** then if then x(on attribute set) agrees then y(another attribute set) cannot agrees the x->y is called as “Functional Dependency”.

x--🡪y Here y is functional dependent on x (or) x is functional determines on y

Candidate key **(not in 3NF)** non-key attribute **child table** **3NF master table**

PK

PK FK

Student table 2NF table

|  |  |  |  |
| --- | --- | --- | --- |
| **Sname** | **Curse id** | **Grade** | **Grade value** |
| Aaa | Cs111 | A | 4.00 |
| Bbb | Cs111 | B | 3.00 |
| Ccc | Cs112 | C | 2.00 |
| Ddd | Cs112 | A | 4.00 |

|  |  |  |
| --- | --- | --- |
| **Sname** | **Curse id** | **Grade** |
| Aaa | Cs111 | A |
| Bbb | Cs111 | B |
| Ccc | Cs112 | C |
| Ddd | Cs112 | A |

|  |  |
| --- | --- |
| **Grade** | **Grade value** |
| A | 4.00 |
| B | 3.00 |
| C | 2.00 |

**Logical Diagram:**

Grade value

Sname

Course id

Grade

1NF--------------🡪 Remove repeating groups

2NF--------------🡪 Remove partial attributes

3NF---------------🡪 Remove attributes which are not dependent on candidate key.

**(or)**

Remove non-key attribute which are dependent on another non-key attributes.

Course id: Course name:

Tutor id: Tutor name:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sno** | **Sname** | **DOB** | **Gender** | **Last attendance** |
|  |  |  |  |  |

**UNF 1NF 2NF 3NF**

Course id **√course id** √**course id** **√course id**

Course name course name not in course name course name

Tutor id tutor id 3NF tutor id tutor id FK

Tutor name tutor name tutor name

Candidate key 3NF Tutor id PK

Sno **√course id**  **√sno** tutor name

Sname **√sno** 2NF sname

DOB not in sname P DOB **√sno**

Gender 2NF DOB P gender sname

Last attendance 1NF gender P 2NF DOB

Last attendance F **√course id** gender

**√Sno**

Last attendance **√course id**

**√Sno**

Last attendance

**CLUSTER:**

Cluster is a database object which contains no.of tables together and also tables share same data block. Clusters are created by database administrator. Clusters are used to **improve performance** of JOINED query.

Cluster tables are created based on a common columnname this common columnname is called as “Cluster key”. Clusters are created **at the time of table creation only** table create complete after cluster creation not possible. In those table are available in cluster then database server **very fastly** retrieve data from cluster table. In cluster **rowid’s** are same.

**Step-1(creating a cluster):**

**Syntax:** create cluster clustername (commoncolumnname datatype(size));

**Step-2 (creating an index on cluster):**

**Syntax:** create index indexname on cluster clustername;

**Step-3 (create actual table)**

**Syntax:** create table tablename (commoncolumnname datatype(size),col1 datatype(size),…………….) cluster clustername (common columnname);

Sql> create cluster emp\_dept(deptno number(5));

Sql> create index index4 on cluster emp\_dept;

Sql> create table c1(deptno number(5),empno number(10),ename varchar2(15), sal number(10)) cluster emp\_dept(deptno);

Sql> create table c2(deptno number(5),dname varchar2(15),loc varchar2(15)) cluster emp\_dept(deptno);

Sql>insert into c1(deptno,empno,ename,sal )values(&deptno,&empno,'&ename', &sal);

Enter value for deptno: **10**

Enter value for empno: **7000**

Enter value for ename: **NARASIMHA**

Enter value for sal: **90000**

Sql> **/**

Enter value for deptno: **20**

Sql> select \* from c1;

**DEPTNO** **EMPNO** **ENAME** **SAL**

30 7002 REDDY 80000

10 7000 NARASIMHA 90000

20 7001 SIMHA 99000

Sql> insert into c2(deptno,dname,loc)values(&deptno,'&dname','&loc');

Enter value for deptno: **10**

Enter value for dname: **NARASIMHA**

Enter value for loc: **YSR**

Sql> **/**

Enter value for deptno: **20**

Sql> select \* from c2;

**DEPTNO** **DNAME** **LOC**

30 REDDY INDIA

10 NARASIMHA YSR

20 SIMHA AP

Sql> select rowid from c1;

**ROWID**

AAAD0AAAEAAAAHEAAA

AAAD0AAAEAAAAHHAAA

AAAD0AAAEAAAAHIAAA

Sql> select rowid from c2;

**ROWID**

AAAD0AAAEAAAAHEAAA

AAAD0AAAEAAAAHHAAA

AAAD0AAAEAAAAHIAAA

Sql> drop cluster emp\_dept;

**ERROR:** cluster not empty

We cannot drop cluster if cluster having tables to overcome this problem oracle 8i introduced INCLUDING TABLES clause to drop cluster along with table.

**Syntax:** drop cluster clustername including tables;

Sql> drop cluster emp\_dept including tables;

All cluster information stored under USER\_CLUSTER data dictionary.

Sql> desc user\_clusters;

**Super key:**

It is a single (or) combination of column’s uniquely identified record in a table.

**Candidate key:**

A candidate key is a combination of attributes that can be uniquely used to identify a database record without any extraneous data. Each table may have one or more candidate keys. One of these candidate keys is selected as the table primary key.

(or)

Minimal super key is called “candidate key”. That is a super key does not subset of a super key.

**Primary key:**

The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain unique values. A primary key column cannot contain NULL values. If any one of the candidate key which uniquely identified record in a table.

**Foreign key:**

A foreign key is a key used to link two tables together. This is sometimes called a referencing key. Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table. You can create a foreign key by defining a FOREIGN KEY constraint when you create or modify a table.

**Alternate key:**

Other than primary key, candidate key which uniquely identified those keys are “alternate key”.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Empno** | **Ename** | **Skill id** | **Skills** | **Voter id** |
| 111 | Narasimha | 11 | Oracle | V1 |
| 111 | Narasimha | 22 | Java | V1 |
| 111 | Narasimha | 33 | C++ | V1 |
| 222 | Simha | 44 | C | V2 |
| 222 | Simha | 11 | Oracle | V2 |
| 333 | Shobha | 55 | Pl/sql | V3 |
| 444 | Rani | ---- | ----- | V4 |
| 555 | Shobharani | 66 | Sql | V5 |
| 666 | Narasimha Reddy | 444 | C | V6 |

**Super key:**

1. empno+skill --------------------------🡪 ck-1
2. empno+ename+skill
3. empno+skill+voter id
4. ename+skill --------------------------🡪 ck-2
5. ename+skill+voter id
6. skill+voter id -------------------------🡪 ck-3
7. empno+ename+skill+voter id

**Candidate key:**

1. empno+skill ---------🡪 ck-1
2. ename+skill ---------🡪 ck-2
3. skill+voter id --------🡪 ck-3

**Primary key:**

1. empno+skill -------------🡪 pk

**Alternate key:**

1. ename+skill
2. skill+voter id

**4. BCNF (Boyce-Codd Normal Form):**

A table is in BCNF the every determinate is a candidate key when a table contain multiple candidate keys.

When a table contain multiple candidate key and also candidate key are overlap and also one candidate keys non-key attribute which depends on another candidate key non-key attribute then only process BCNF process.

If a table contain single candidate key the BCNF is also same as 3NF when a table contain more than one candidate key then deletion problem occurred to overcome this problem we are using BCNF process. BCNF specifies relationship between candidate key it self.

Identifies one candidate key non-key attributes which depends on another candidate key attributes non-key attributes then putted those entire attributes separate table this table is called as “BCNF”. In this table only every determinate is a candidate key.

**NOTE:** In this BCNF table does not have non-key attributes.

**Project table** non-key attributes **BCNF**

|  |  |
| --- | --- |
| **Ecode** | **Ename** |
| E1 | Narasimha |
| E2 | Simha |
| E3 | Reddy |
| E4 | Shobha |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ecode** | **Ename** | **Projcode** | **Hours** |
| E1 | Narasimha | P1 | 10 |
| E2 | Simha | P1 | 7 |
| E3 | Reddy | P2 | 5 |
| E4 | Shobha | P2 | 6 |
| E4 | Shobha | P1 | 8 |

FD

1. ecode+Projcode Hours 1.Ecode Ename

Determinate

1. ename+projectcode Hours ck FD

2.Ename Ecode

Determinate ck

|  |  |  |  |
| --- | --- | --- | --- |
| **Projcode**  Non-key attribute  **University table** | **Dept** | **Hod** | **Hours** |
| P1 | Physics | Narasimha | 5 |
| P1 | Computers | Simha | 7 |
| P2 | Chemistry | Reddy | 9 |
| P3 | Zoology | Shobha | 8 |
| P4 | Botany | Rani | 10 |

**BCNF**

|  |  |  |
| --- | --- | --- |
| **Projcode** | **Dept** | **Hours** |
| P1 | Physics | 5 |
| P1 | Computer | 7 |
| P2 | Chemistry | 9 |
| P3 | Zoology | 8 |
| P4 | Botany | 10 |

|  |  |
| --- | --- |
| **Dept** | **Hod** |
| Physics | Narasimha |
| Computer | Simha |
| chemistry | Reddy |
| Zoology | Shobha |
| Botany | Rani |

1. Projcode+dept ---------🡪 ck-1

1. Projcode+hod ----------🡪 ck-2

FD

1. projcode+dept dept

FD

FD

1. Dept Hod

Determinate

2. projectcode+dept Hours ck FD

2. Hod dept

Determinate ck

1. **Fourth Normal Form:**

If a table is in 4NF then that table does not contain more than **one independent** multi valued attribute.

If a table contains more than two attribute and also identifying a record uniquely combination attributes and also that table does contain and also one attribute set of values depend on another attribute set of value and also same attribute set of value which depend on another attribute set of values and also some attribute are not logically related then only database designers uses 4NF process.

**Process:**

Identifies attributes which dependence on another attribute those attributes putted into another table this table is called as “4NF table”. This table does not contain more than one independent multi valued attribute. These tables automatically reduce redundancy.

**Movie table**

Candidate key

|  |  |
| --- | --- |
| **Movie** | **Producer** |
|  |  |

|  |  |  |
| --- | --- | --- |
| **Movie** | **Star** | **Producer** |
| M1 | S1 | P1 |
| M1 | S2 | P1 |
| M1 | S3 | P1 |
| M1 | S4 | P2 |
| M1 | S5 | P2 |
| M2 | S6 | P3 |
| M2 | S3 | P3 |
| M2 | S2 | P3 |

|  |  |
| --- | --- |
| **Movie** | **Star** |
|  |  |

**1** Before 4NF process in the down table whenever on get a new project then we have to supply null values for the skill and also whenever having new skill we have to supply null values for the project id. Whenever we are using 4NF process this problems are avoided.

**Assumption:**

Each employee has multiple projects and also each employee has multiple skills.

**Project table**

Candidate key

|  |  |
| --- | --- |
| **Ecode** | **Skills** |
| 111 | C |
| 111 | Java |
| 111 | Db2 |
| 111 | Oracle |
| 111 | Php |
| 111 | Pl/sql |

|  |  |
| --- | --- |
| **Ecode** | **Projcode** |
| 111 | Railway reservation |
| 111 | Library management |
| 111 | University management |

|  |  |  |
| --- | --- | --- |
| **Ecode** | **Projcode** | **Skills** |
| 111 | Railway reservation | C |
| 111  4NF | Railway reservation | Java |
| 111 | Railway reservation | Db2 |
| 111 | Library management | Oracle |
| 111 | Library management | Php |
| 111 | University management |  |
| 111 |  | Pl/sql |

**6. Fifth Normal Form:**

A table is in 5NF then that table does not contain cyclic dependency. If a table contain s more than two attribute and also identified record uniquely using combination of all attribute and also all attribute set of values related to one another then only using 5NF.

Generally in 4NF some attributes are not logical related where as in 5NF all attribute are related. 5NF is also called as “Projection-join normal form”.

|  |  |
| --- | --- |
| **Bayer** | **Company** |
|  |  |

|  |  |
| --- | --- |
| **Product** | **company** |
|  |  |

In this normal form we are splitting a table into no.of tables and also when we are joining these tables resultant record must be available on resource table.

|  |  |
| --- | --- |
| **Bayer** | **product** |
|  |  |

**Bayer table**

|  |  |  |
| --- | --- | --- |
| **Bayer** | **Product** | **company** |
| B1 | Shirt | Levis |
| B1 | Jeans | Arrow |
| B1 | Shirts | Peeps |

**Chapter-13**

**CONVERSIONS**

**Set operators:**

Set operators are used to retrieve data from single (or) multiple tables. This operator is also called as “Vertical joins”.

1. Union -------------🡪 It returns values one time
2. Union all ---------🡪 unique+duplicate
3. Intersect ----------🡪 Common values
4. Minus -------------🡪 Values are in first query those values are not in

second query.

Sql> select job from emp where deptno=10

union

select dname from dept where deptno=20;

**JOB**

CLERK

MANAGER

PRESIDENT

RESEARCH

Sql> select job from emp where deptno=10

union

select job from emp where deptno=20;

**JOB**

ANALYST

CLERK

MANAGER

PRESIDENT

**NOTE:**

In all database we are using set operators always corresponding expression must belong to same datatype and also set operator returns first query columnname as column heading’s.

Sql> select deptno from emp

union

select dname from dept;

**ERROR:** expression must have same datatype as corresponding exp

Sql> select ename from emp

union

select dname from dept;

**ENAME**

ACCOUNTING

ADAMS

-----------

WARD

18 rows selected.

**NOTE:**

In oracle we can also retrieve data using set operators if corresponding expressions not belongs to same datatype also in this case we must use corresponding type conversion function.

Sql> select deptno "deptno",to\_char(null)"deptname" from emp

union

select to\_number(null),dname from dept;

**DEPTNO** **DEPTNAME**

10

20

30

ACCOUNTING

GNSR

OPERATIONS

RESEARCH

Sql> select deptno "deptno",to\_char('wwwww')"deptname" from emp

union

select to\_number(99),dname from dept;

**DEPTNO DEPTNAME**

10 wwwww

20 wwwww

30 wwwww

99 ACCOUNTING

99 GNSR

99 OPERATIONS

99 RESEARCH

**CONVERSIONS:**

Converting one datatype into another datatype is called as “Conversions”. Conversions are two types.

1. Implicit Conversion (or) Automatic Conversion

2. Explicit Conversion

**1. Implicit Conversion:**

**NOTE-1:** When a sql expression contains a string representing pure numeric values then oracle server automatically convert into string type into number type.

Sql> select sal+’100’ from emp;

**SAL+'100'**

900

1700

--------

Sql> select sal+’100Z’ from emp;

**ERROR:** ora-1722: invalid number

**NOTE-2:**

Whenever we are passing number into character function then oracle server internally uses implicit conversion to convert number type into string type.

Sql> select length(111) from dual;

**LENGTH (111)**

3

Sql> select length('abcd') from dual;

**LENGTH ('ABCD')**

4

**NOTE-3:**

Whenever we are passing date string into data function then oracle server automatically converts date string into date type but here passed parameter must be default date (number-character-number) format.

Sql> select last\_day('15-aug-05')+4 from dual;

**LAST\_DAY (**

04-SEP-05

Sql> select last\_day('15-07-05') from dual;

**ERROR:** not a valid month

Implicit conversion

|  |  |  |  |
| --- | --- | --- | --- |
| **From** | **To** | **Assignment** | **Evaluation of expression** |
| Varchar2 (or) char | Number | Yes | Yes |
| Varchar2 (or) char | Date | Yes | Yes |
| Number | Varchar2 | Yes | No |
| Date | Varchar | Yes | No |

**2. Explicit Conversion:**

To\_number () to\_date ()

Number char date

To\_char () to\_char ()

1. To\_number()
2. To\_char()
3. To\_date()
4. **To\_number():**

It is used to converting a string representing a numeric value with format into a numeric value without format.

Sql> select '$24.6'+3 from dual;

**ERROR:** ORA-01722: invalid number

Sql> select to\_number ('$24.6') +3 from dual;

**ERROR:** ORA-01722: invalid number

Sql> select to\_number ('$24.6','$99.9’) +3 from dual;

**TO\_NUMBER ('$24.6','$99.9') +3**

27.6

**NOTE:** Only format models working otherwise your own way given not working.

Sql> select to\_number ('d24.6','d99.9') +3 from dual;

**ERROR:** ORA-01481: invalid number format model

1. **To\_char():**

This is an overloading function which is used to convert number type into char type and also date type into date string.

Sql> select to\_char(20000,'99,999.99') from dual;

**TO\_CHAR(20**

20,000.00

Sql> select to\_char(20000,'99g999d99') from dual;

**TO\_CHAR(20**

20,000.00

Sql> select to\_char(123456.78,'$99,99,99,999.99') from dual;

**TO\_CHAR(123456.78**

$1,23,456.78

Sql> select ename,sal,to\_char(sal,'L99,99,999.99','nLS\_currency=Rs')

"INDIAN CURRENCY" from EMP;

**ENAME SAL INDIAN CURRENCY**

SMITH 800 Rs 800.00

ALLEN 1600 Rs 1,600.00

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

G -------------🡪 Group separator

D -------------🡪 Decimal indicator

L --------------🡪 Local currency

Sql> select to\_char(20000,'9,999') from dual;

**TO\_CHA**

######

Sql> select to\_char(20000,'99,999') from dual;

**TO\_CHAR**

20,000

Sql> select ename,to\_char(sal,'L99,99,99,999') "CURRENCY" from emp;

**ENAME** **CURRENCY**

SMITH $800

ALLEN $1,600

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

**NOTE:**

By default oracle server returns local currency ‘$’ if you want return our own currency format we are using ‘NLS\_CURRENCY’ format in third format of the to\_char function.

Sql> select ename,sal,to\_char(sal,'L99,99,999.99','nLS\_currency=Rs')

"INDIAN CURRENCY" from EMP;

Sql> select ename,sal,nvl(to\_char(comm),' No Commission') from EMP;

**ENAME SAL NVL(TO\_CHAR(COMM),'NOCOMMISSION')**

SMITH 800 No Commission

ALLEN 1600 300

WARD 1250 500

JONES 2975 No Commission

**NOTE:**

If we want add leading zero’s then we must use zero in the format. If we want add last only zero’s then we are using 9 in the format.

Sql> select to\_char(345.6,'0999.9999') from dual;

**TO\_CHAR(34**

0345.6000

**NOTE:**

If we want to display number datatype values into char format then we are using to\_char with format value.

Sql> select to\_char(sysdate,'dd/mm/yyyy') from dual;

**TO\_CHAR(SY**

18/09/2014

1. **To\_date():**

It is used to convert date string into date type.

Sql> select to\_date('12/sep/14') from dual;

**TO\_DATE('**

12-SEP-14

**NOTE:**

Whenever we are using to\_date passed parameter return value match with the default date datatype return value otherwise oracle server returns an error to overcome this problem use a one parameter formation.

**NOTE:**

Whenever we ar5e passing date string into date function then oracle server automatically convert into date string into date type. But in this case passed parameter format number-character-number otherwise use in to\_date() function explicit.

Sql> select last\_day('11/JAN/06') from dual;

**LAST\_DAY(**

31-JAN-06

Sql> select last\_day(to\_date(‘11/03/06’,’DD/MM/YY’)) from dual;

**LAST\_DAY(**

31-MAR-06

**DECODE:**

Decode is a conversion function which is used to convert one type into type. Decode internally used equality (=) operator. It is also called same as if-----then-----elseif construct.

**Syntax:** decode(colname,value1,stmt1,value2,stmt2,……………stmtN)

If colname=value1 then stmt;

Elseif colname=value2 then stmts;

Sql> select ename,deptno,decode(deptno,10,'ten',20,'twenty','other') from emp;

**ENAME** **DEPTNO** **DECODE**

SMITH 20 twenty

ALLEN 30 other

WARD 30 other

JONES 20 twenty

MARTIN 30 other

we can also use decode function to display row values into column values.

Sql> select job,sum(decode(deptno,10,sal)) “deptno 10”,sum(decode(deptno,20,sal)) “deptno 20” ,sum(decode(deptno,30,sal)) “deptno 30” from emp group by job;

**JOB** **deptno 10** **deptno 20** **deptno 30**

CLERK 1300 1900 950

SALESMAN 5600

PRESIDENT 5000

MANAGER 2450 2975 2850

ANALYST 6000

Sql> select dname,sum(decode(job,’CLERK’,1,0)) “CLERK”,

Sum(decode(job,’SALESMAN’,1,0)) “SALESMAN”,

Sum(decode(job,’MANAGER’,1,0)) “MANAGER”,

Sum(decode(job,’ANALYST’,1,0)) “ANALYST”,

Sum(decode(job,’PRESIDENT’,1,0)) “PRESIDENT”

From emp e,dept d where e.deptno=d.deptno group by dname;

**DNAME CLERK SALESMAN MANAGER ANALYST PRESIDENT**

ACCOUNTING 1 0 1 0 1

RESEARCH 2 0 1 2 0

SALES 1 4 1 0 0

Sql> select dname,sum(decode(job,’CLERK’,1,0)) “CLERK”,

Sum(decode(job,’SALESMAN’,1,0)) “SALESMAN”,

Sum(decode(job,’MANAGER’,1,0)) “MANAGER”,

Sum(decode(job,’ANALYST’,1,0)) “ANALYST”,

Sum(decode(job,’PRESIDENT’,1,0)) “PRESIDENT”

From emp e,dept d where e.deptno(+)=d.deptno group by dname;

**DNAME CLERK SALESMAN MANAGER ANALYST PRESIDENT**

ACCOUNTING 1 0 1 0 1

OPERATIONS 0 0 0 0 0

RESEARCH 2 0 1 2 0

SALES 1 4 1 0 0

**NOTE:**

We can also use decode function in update statement for modifying data conditionally.

**Q) Write a query to modifying commission column data of employee from emp table based on following condition?**

1. if job=’CLERK’ then comm ------🡪 10% of sal

2. if job=’SALESMAN’ then comm ----🡪 20% of sal and all other job’s comm -------🡪 30% of sal

Sql> update emp set comm=decode (job,’CLERK’,sal\*0.1, ’SALESMAN’,sal\*0.2, sal\*0.3);

**EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO**

7369 SMITH CLERK 7902 17-DEC-80 800 80 20

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 320 30

7521 WARD SALESMAN 7698 22-FEB-81 1250 250 30

7566 JONES MANAGER 7839 02-APR-81 2975 892.5 20

7654 MARTIN SALESMAN 7698 28-SEP-81 1250 250 30

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

**NOTE:**

If we want count different values from a columns then we are using decode function within group by clause. Whenever we are using decode function within group by clause then automatically rows values convert into column values.

Sql> select job,sum(decode(deptno,10,sal)) "deptno 10",

sum(decode(deptno,20,sal)) "deptno 20",

sum(decode(deptno,30,sal)) "deptno 30"

from emp group by job;

**JOB** **deptno 10** **deptno 20** **deptno 30**

CLERK 1300 1900 950

SALESMAN 5600

PRESIDENT 5000

MANAGER 2450 2975 2850

ANALYST 6000

**CASE STATEMENT:**

Oracle 8.0 introduced CASE statement and also 8i introduced case conditional stmt. Case is a statement which is used to decoding the values case stmt is **performance very high** compare to decode function.

Method-1: case columnname

When value1 then stmt1

When value2 then stmt2

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

Else stmt end

Sql> select ename,sal,deptno,case deptn

when 10 then 'TEN'

when 20 then 'TWENTY'

else 'OTHERS' end from emp;

**ENAME** **SAL** **DEPTNO** **CASEDE**

SMITH 800 20 TWENTY

ALLEN 1600 30 OTHERS

CLARK 2450 10 TEN

KING 5000 10 TEN

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

Method-2: (case conditional statement – oracle 8i)

Case

When colcondition1 then stmt1

When colcondition2 then stmt2

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

Else stmt end

Sql> select ename,sal,case

when sal<1000 then 'LOW SALARY'

when sal between 1000 and 2500 then 'MEDIUM SALARY'

when sal in(3150,3000,3500,2850) then 'SPECIAL SALARY'

else 'OTHER SALARY' end from emp;

**ENAME** **SAL** **CASEWHENSAL<10**

SMITH 800 LOW SALARIES

ALLEN 1600 MEdIUM SALARY

WARD 1250 MEdIUM SALARY

JONES 2975 OTHER SALARY

MARTIN 1250 MEdIUM SALARY

BLAKE 2850 SPECIAL SALARY

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

**Chapter-14**

**SYNONYMS**

Synonym is a database object. Synonym is an reference (or) aliasname for the original object. Synonyms is an guide another designer,username,object name,database link synonym provides security for another object. Synonyms are two types.

1. Public Synonyms

2. Private Synonyms

By default synonyms are private synonyms. If we want to create public synonyms then we are using “create public synonym privileges”.

Sql> grant create public synonym to username;

**Syntax:** create public synonym Synonymname

for [username.objectname@databaselink](mailto:username.objectname@databaselink);

Scott/tiger Simha/Simha reddy/reddy

Sql>grant all on emp sql>select \* from scott.emp; (√) sql>select \* from scott.emp; (√)

To Simha,reddy;

Sql>create synonym AAA for

Scott.emp;

Sql>select \* from AAA; (√) sql>select \* from AAA; (√)

**Error:** table or view does not exit

Sql>crate public synonym ABC

For scott.emp;

**Error:** insufficient privileges

Sql>conn sys as sysdba

Enter password: SYS

Sql>grant create public synonym

to Simha;

Sql>conn Simha/Simha;

Sql>create public synonym ABC

For scott.emp;

Sql>select \* from ABC; (√) sql>select \* from ABC; (√)

All synonyms information stored undered USER\_SYNONYMS data dictionary.

Sql> desc user\_synonyms;

**Chapter-15**

**TCL (Transaction Control Language)**

**Transaction:**

It is a logical unit of work in between two points Oracle having two transaction commands.

1. Commit
2. Savepoint
3. **Commit:**

This command is used to **save** the transaction permanently hard disk.

**Syntax:** COMMIT

1. **Savepoint:**

It is a logical mark between transactions.

**Syntax:** SAVEPOINT

**Rollback to particular Savepoint:**

**Syntax:** rollback to savepointname;

Sql> create table test(sno number(5));

Sql> insert into test values(&sno);

Sql> select \* from test;

**SNO**

1

2

3

4

5

Sql> insert into test values(&sno);

Enter value for sno: 6

Enter value for sno: 7

Sql> update test set sno=999 where sno=3;

Sql> delete from test where sno=2;

Sql> Savepoint s1;

Sql> select \* from test;

**SNO**

1

999

4

5

6

7

Sql> insert into test values(&sno);

Enter value for sno: 8

Enter value for sno: 9

Sql> delete from test where sno=9;

Sql> Savepoint s2;

Sql>select \* from test;

**SNO**

1

999

4

5

6

7

8

Sql> insert into test values(&sno);

Enter value for sno: 9

Enter value for sno: 10

Sql> delete from test where sno=10;

Sql> select \* from test;

**SNO**

1

999

4

5

6

7

8

9

Sql> rollback to s2;

Sql> select \* from test;

**SNO**

1

999

4

5

6

7

8

Sql> rollback to s1;

Sql> select \* from test;

**SNO**

1

999

4

5

6

7

**Chapter-16**

**Oracle Server Architecture**

Oracle server mainly consist of two parts

1. Storage Area
2. Instance
3. **Storage Area:**

Whenever we are installing oracle server automatically three files are created in a hard disk. These three files are also called as ‘Physical Database’. This files are stored ‘Physical Storage Area’.

1. Data files (**.**dbf)
2. Control files (**.**ctl)
3. Log files (**.**log)

RAM Memory

**Sqlplus client tool** **Oracle Server Architecture**

**I**

**N**

**S**

**T**

**A**

**N**

**C**

**E**

User name:

Password:

**SGA**

Java pool

Large pool

Redo log buffer

Database buffer cache

Library cache

Dictionary cache

**OK**

**PGA**

Data files (**.**dbf)

Control files (**.**ctl)

Redo log files (**.**log)

Process

Hard disk

Storage Area

Database

G2

G1

M2

M1

M2

M1

Member

Groups

SGA-----------🡪 System Global Area (or) Shared Global Area

PGA-----------🡪 Private Global Area LGWR--------🡪 Log Writer

DBWR--------🡪 Database Writer SMON-----------🡪 System Monitor

PMON--------🡪 Process Monitor CKPT------------🡪 Checkpoint

SGA-----------🡪 System Global Area (or) Shared Global Area

**STORED FILES:**

Data files stores database objects physically that is file stores index, procedures, tables, function, …………….etc. Data files also rollback segment in log onwards this segment’s also called as “Undo Segment”. All data files information stored under DBA\_DATA\_FILE data dictionary. If you want view path of the data file then we are using FILE\_NAME property from DBA\_DATA\_FILE data dictionary;

Sql> desc dba\_data\_files;

Sql> select file\_name from dba\_data\_files;

**FILE\_NAME**

C:\ORACLEXE\ORADATA\XE\USERS.DBF

C:\ORACLEXE\ORADATA\XE\SYSAUX.DBF

C:\ORACLEXE\ORADATA\XE\UNDO.DBF

C:\ORACLEXE\ORADATA\XE\SYSTEM.DBF

**CONTROL FILES:**

Control files controls all other files in storage area this file extension is **.ctl**. Oracle 9i introduced minimum three control files in storage area control files also stores database. All control files information stores under V$CONTROLFILE data dictionary. Control files are used by database administrator in recovery process.

Sql> desc V$CONTROLFILE;

Sql> select name from v$controlfile;

**NAME**

C:\ORACLEXE\ORADATA\XE\CONTROL1.DBF

C:\ORACLEXE\ORADATA\XE\CONTROL2.DBF

C:\ORACLEXE\ORADATA\XE\CONTROL3.DBF

**REDO LOG FILE:**

Redo log file store committed information redo log buffer. This files also used by database administrator. In recovery process all files information log file stored under V$LOGFILE data dictionary.

If we want view path of the log file then we are using MEMBER property from V$LOGFILE data dictionary.

Sql> desc v$logfile;

Sql> select member from v$logfile;

**MEMBER**

C:\ORACLEXE\APP\ORACLE\FLASH\_RECOVERY\_AREA\XE\ONLINELOG\O1\_MF\_2\_9VHP6GDM\_.LOG

C:\ORACLEXE\APP\ORACLE\FLASH\_RECOVERY\_AREA\XE\ONLINELOG\O1\_MF\_1\_9VHP6DQX\_.LOG

1. **INSTANCE:**

Whenever we are connect into the server using a client oracle server automatically on instance created in memory area this instance having two parts.

1. SGA
2. PGA

**SGA:**

SGA is also called as System Global Area (or) Shared Global Area. SGA consists of set buffer these buffers are

1. Database Buffer Cache

2. Shared Pool

3. Java Pool

4. Large Pool

5. Redo Log Buffer

Whenever we are subheading SQL, PL/SQL codes into the database server that code store in library cache within shared pool library cache. Always reduces passing that is library cache checks syntax and semantic checking of the SQL, PL/SQL code. Whenever user requesting data from the table oracle server process checks request table available in database buffer cache (or) Not If request object is not available in database buffer cache the DBWR process checks request table available in data files. When it is available then **copy of the table** created in database buffer cache.

**Oracle server**

**SGA**

11111111111111111111111111111111111111111111111111111111

User

DBWR

Select \* from emp;

Data files

Emp

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Whenever we are subheading update, delete stmt those transaction are performing redo log buffer that is redo log of buffer copy of the table from database buffer cache after modification whenever we are using commit,rollback those values are stored in data files,redo log files and also redo log changes effected in database buffer cache.

**Redo**

**SGA**

Database buffer cache Redo log buffer

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

User

Update

Log wr

DBWR

Delete

Data files Redo log

Sql> commit

**- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -**

Emp

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Library cache also stores cache value defined in sequence object. Shared pool also contains dictionary cache which executed DCL related data dictionary property. Java pool executes java related object and also large pool used by database administrator in recovery process. Server process also contains PGA which uniquely identifies each client connect to the server. Whenever we are using DML transaction new values for the transaction also stored in redo log buffer this values are also called as “ONOFF”. Whenever user using commit (or) 1/3 fill of redo log buffer then automatically redo log buffer data transfer into redo log files this files are used by database administrator in recovery process.

**sqlplus client** Ram memorydirty reader **UNDO**

Undo area

User name:

Password:

Redo log buffer

5 1

Database

Buffer cache

**SGA**

**Undo file**

Update

Delete

Commit

Redo log files

1

1 5

2

3

T1

Data files

Hard disk

Whenever we are using DML transaction old data stored in undo area within database buffer cache and also new data stored in dirty reader within database buffer cache whenever user using commit automatically new data permanently stored in data files and also old data stored in undo files within hard disk. This undo file used in FLASHBACK query. Flashback query are used by database administrator content of a table can be query based on a specific point of time. Oracle 9i introduced flashback query to retrieve accidental data after commit transaction also.

**Oracle server having three structures:**

1. Physical Structure

2. Logical Structure

3. Logical Storage Structure

**1. Physical Structure:**

Physical structure handle database administrator only. Physical structure contain these are data files, control files, redo log files. This structure is visible in the operating system.

**2. Logical structure:**

Logical structure not visible to the operating system this structure contains database object like Tables, Views, Index, Sequence, Procedure, Trigger, Function, Package,…………etc. These structures handle by either Developer (or) Administrator.

**3**. **Logical Storage Area:**

This structure is handling by database administrator only.

1. Tablespace

2. Segment

3. Extent

4. Block

**Tablespace:**

Tablespace Tablespace Tablespace

**- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -**

**- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -**

**- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -**

Logical storage

Not visible in OS

Data

Files

Data

Files

Data

Files

Physical Structure

Visible in OS

Generally without starting database also we are viewing physical structure of the database. Whereas whenever starting database then only we doing logical storage structure. Tablespace is nothing but collection of data files. One data file belongs to one tablespace only. Whenever we are installing oracle server automatically SIX tablespace are created. If you want run oracle minimum TWO tablespace are required. These are SYSAUX tablespace, SYSTEM tablespace. System tablespace contains metadata that is system tablespace all data dictionary. Database administrator also creating tablespace explicitly uses.

**Syntax:** create tablespace tablespacename data file ‘path of data file’

All tablespace information is stored under DBA\_TABLESPACES data dictionary.

Sql> conn sys as sysdba

Enter password: SYS

Sql> desc dba\_tablespaces;

Sql> select tablespace\_name from dba\_tablespaces;

If we want view data files and related tablespace then we are using DBA\_DATA\_FILES data dictionary.

Sql> desc dba\_data\_files;

Sql> select tablespace\_name,file\_name from dba\_data\_files;

**TABLESPACE\_NAME FILE\_NAME**

USERS C:\ORACLEXE\ORADATA\XE\USERS.DBF

SYSAUX C:\ORACLEXE\ORADATA\XE\SYSAUX.DBF

UNDO C:\ORACLEXE\ORADATA\XE\UNDO.DBF

SYSTEM C:\ORACLEXE\ORADATA\XE\SYSTEM.DBF

Users---🡪logical storage structure physical storage structure

C:\ORACLEXE\ORADATA\XE\USERS.DBF

System tablespace (metadata) Bits🡨----bytes🡨----block **Segment**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

[all data dictionary] Extent 1

Oracle Database

Extent 2

Segment1 Segment2

**------------- --------------**

**------------- --------------**

**-------------- -------------**

**-------------- --------------**

Segment1 Segment2

Tablespace data files sysaux tablespace undo tablespace

**Segment, Extent, Block:**

Sql> create table T1(sno number(5),varchar2(10));

Sql> insert into T1 values(1,’aaa’);

Sql> insert into T1values(2,’bbb’); **Block**

**Header**

**Data**

Segment 2mb

Extent1 1mb Extent2 1mb

Row address row

**. .**

Dictionary

Name of table

Table

1 aaa

Dictionary

2 bbb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Tablespace Blocks

Segment is an collection of extents that forms a database object likes tables, views, index, …………..etc.

**LOCKS**

Locks are a mechanism which prevents unauthorized access for our resources. All database systems having two types lock.

1. Row Level Lock

2. Table Level Lock

**1. Row Level Lock:**

Oracle 6.0 introduced row level lock. In this method we are locking set of rows using **for update** clause. This clause used in select statement only using locks only that perform use DML statement not using another user.

**Syntax:** select \* from tablespace where condition for update [nowait];

Whenever we are forming lock another user query the data but they cannot perform DML operation. Whenever we are use COMMIT, ROLLBACK when lock are release.

**Scott/tiger Simha/simha**

Sql> select \* from emp where deptno=10

for update; Sql> select scott.emp set sal=sal+100

where deptno=10;

sql> commit; [for release locks] [we cannot perform DML]

**NOWAIT:**

Nowait is a optional clause is used in FOR UPDATE clause. Wherever we are using this clause oracle server automatic get control into current user if another user not releasing lock in this case oracle server returns an ERROR: ora-0054: Resource busy.

**Scott/tiger Simha/simha**

Sql> select \* from scott.emp where deptno=10 for update;

Sql> select \* from emp where dept=10

For update nowait;

**ERROR:** ora-0054: Resource busy

Sql> **/**

**NOTE:**

In all database whenever we are using DML statement automatically database server internally uses exclusive lock. Wherever we are using COMMIT,ROLLBACK then only these lock are released.

**2. Table Level Lock:**

In this method we are locking a table. These are 2 types of table level lock using by database developer.

1. Share Lock

2. Exclusive Lock

**1. Share Lock:**

Whenever we are using this lock another uses query the data but they cannot perform DML operations and also at a time **no.of user** lock resource.

**Syntax:** lock table tablename in share mode;

**Scott/tiger Simha/simha**

Sql> lock table emp in share mode; Sql> select \* from scott.emp; ( √ )

Sql> lock table scott.emp in share mode; ( √ )

Sql> update scott.emp set sal=sal+100;

Sql> commit; (for release lock) [we cannot perform DML]

**2. Exclusive Lock:**

Whenever we are using this lock another user query the data but we cannot perform DML operations and also at a time only **one user** lock the resource.

**Scott/tiger Simha/simha**

Sql> lock table emp in exclusive mode; Sql> select \* from scott.emp; ( √ )

Sql> lock table scott.emp in share mode; ( 🗶 )

Sql> commit; (for release lock)

**NOTE:**

In all database procedure language when we are using cursor locking mechanism internally database server uses exclusive lock.

**DEADLOCK:**

In oracle whenever we are try to perform same operation in a single resource through different session then automatically deadlock occurred. In this case oracle server returns an **ERROR:** ora-0060: Deadlock detected while waiting for resource.

**Scott/tiger Simha/simha**

Sql> update emp set sal=sal+100 Sql> update emp set sal=sal+100 where deptno=20;

Where deptno=10; Sql> update emp set sal=sal+100 where deptno=10;

Sql> update emp set sal=sal+100 [deadlock detected]

where deptno=20;

[we cannot perform DML]

**ERROR:** ora-0060: Deadlock detected

while waiting for resource

Sql> commit; (for release lock)

**Chapter-17**

**TABLES**

**Global Temporary Tables (GTT):**

Global Temporary Tables stores data temporary that is these table data stored in RAM memory area. These tables are used by database administrator to process data **very fastly** retrieve. Oracle having 2 type of global temporary table

1. Transaction Specific

2. Session Specific

**1. Transaction Specific:**

In this type of GTT stores data until we are using commit (or) rollback.

**Syntax:** create global temporary table tablename(col1 datatype(size),………….)

On commit delete rows;

Sql> create global temporary table GTT(sno number(5)) on commit delete rows;

Sql> insert into GTT values(1);

Sql> insert into GTT values(2);

Sql> insert into GTT values(3);

Sql> select \* from gtt;

**SNO**

1

2

3

Sql> commit;

Sql> select \* from gtt;

No Rows Selected

**2. Session Specific:**

This Global Temporary Tables data available in particular sessions if we are giving commit also another session cannot view that data.

**Syntax:** create global temporary table tablename(col1 datatype(size),…………….)

On commit preserve rows;

**Session-1:**

Sql>create global temporary table GTT(sno number(5)) on commit preserve rows;

Sql> insert into GTT values(1);

Sql> insert into GTT values(2);

Sql> insert into GTT values(3);

Sql> select \* from gtt;

**SNO**

1

2

3

Sql> commit;

Sql> select \* from gtt;

**SNO**

1

2

3

**Session-2:**

Sql> select \* from gtt;

No Rows Selected

**NESTED TABLE:**

Oracle 8.0 introduced nested table. Table within another table is called as “Nested Table”. Nested table is a userdefined table. This is used to multiple data store in a single unit. Generally in oracle if we want to store **no.of data** items then we are using index by table in PL/SQL. But these tables not allowed to stores this table permanently in database. To overcome this problem oracle 8.0 introduced extension of the index by table called as “Nested Table”. This is used to store **no.of data items** permanently in database using SQL.

Step-1: creating an objecttype

Step-2: create an nested table type

Step-3: create an actual table

**Creating an object type:**

This is a userdefined datatype. Which is used represent using different datatype in a single unit it is also same as structure in ‘C’ language.

**Syntax:** create or replace type typename as

object(attributename1 datatype(size),attribute2 datatype(size),…..);

**Create a nested table type:**

**Syntax:** create or replace type typename as table of objecttypename;

**Create an actual table:**

**Syntax:** create table tablename (col1 datatype(size),………colN nestedtabletype)

Nested table colN store as anyname;

Sql> create or replace type nt1 as

object(bookid number(5),bookname varchar2(15),price number(10));

**/**

Sql> create or replace type FAN as table of nt1;

**/**

Sql> create table student(sno number(5),sname varchar2(15),col3 FAN)

nested table col3 store as ABC;

Sql> desc student;

**Name** **Null?** **Type**

SNO NUMBER (5)

SNAME VARCHAR2 (15)

COL3 FAN

Nested table type

Sql> desc fan;

Fan TABLE OF NT1

**Name** **Null?** **Type**

BOOKID NUMBER (5)

BOOKNAME VARCHAR2 (15)

PRICE NUMBER (10)

If we want to insert data into nested table columns then we must using constructor. Here constructor name is same as typename.

Sql> insert into student values(1,'NARASIMHA',

FAN(nt1(101,'JAVA',500),nt1(102,'ORACLE',3000)));

Sql> select \* from student;

**SNO** **SNAME** **COL3 (BOOKID, BOOKNAME, PRICE)**

1 NARASIMHA FAN (NT1 (101, 'JAVA', 500), NT1 (102, 'ORACLE', 3000))

Sql> select col3 from student;

**COL3 (BOOKID, BOOKNAME, PRICE)**

FAN (NT1 (101, 'JAVA', 500), NT1 (102, 'ORACLE', 3000))

If we want to view nested table data then we are using table operator.

Number(5) varchar2(15) FAN

|  |  |  |
| --- | --- | --- |
| **Bookid** | **Bookname** | **Price** |
| 101  102 | JAVA  ORACLE | 500  3000 |

|  |  |  |
| --- | --- | --- |
| **Sno** | **Sname** | **Col3** |
| 1 | NARASIMHA |  |

S**yntax:** select \* from table (select nestedtablename from tablename);

Sql> select \* from table( select col3 from student);

**BOOKID** **BOOKNAME** **PRICE**

101 JAVA 500

102 ORACLE 3000

This table operator used in insert,update,delete,select statement all userdefined types stored in USER\_TYPES data dictionary.

Sql> desc user\_types;

Sql> select TYPE\_NAME,ATTRIBUTES,METHODS,TYPEID,PREDEFINED

from user\_types;

**TYPE\_NAME ATTRIBUTES METHODS TYPEID PRE**

FAN 0 0 NO

NT1 3 0 NO

We can also drop type using drop type typename

Sql> drop type typename;

**PARTITIONS:**

Table can be separation into **number of tables** is called ‘Partition’. Partitions table are created by database performance of the application. These tables are used in **backup and recovery process**. Generally partitions table created when database table having large amount of data partition are created based on a column this is called ‘Partition Key’.

1. Range Partition

2. List Partition

3. Hash Partition

**1. Range Partition:**

If we want view partition then we are using following syntax. This method created based on range of values.

**Syntax:** select \* from tablename partition (partitionname1,partitionname2,……);

**Syntax:** create table tablename(col1 datatype(size),col2 datatype(size),………..)

Partition by range(keycolname) (partition partitionname values less than (actual value),

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

Partition partitionname values less than (maxvalue));

Sql> create table test(sno number(10),ename varchar2(15),sal number(10)) partition by range(sal)(partition p1 values less than(1000),partition p2 values less than(2000),partition other values less than(maxvalue));

Sql> insert into test values(……………………………..);

Sql> select \* from test;

Sql> select \* from test partition(p1);

Sql> select \* from test partition(p2);

**2. List Partition:**

Oracle 9i introduced list partition. Using this partition we can also create partition based character datatype columns. Database administrator creates on list of values.

**Syntax:** create table tablename(col1 datatype(size),……………….)

Partition by list(keycolname)

(partition partitionname values(value1,value2,…………..),

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

Partition partitionname values(default));

Sql> create table test(sno number(10),name varchar2(10))

Partition by list(name)

(partition p1 values(‘india’,’pak’),

Partition p2 values(‘usa’,’usk’),

Partition other values(default));

Sql> inert into test values(……………………………);

Sql> select \* from test;

**3. Hash Partition:**

In this method partition when we are specified partition number then automatically created oracle server based on hash partition.

**Syntax:** create table tablename(col1 datatype(size),………)

Partition by hash(keycolname) partitions anynumber;

Sql> create table test(sno number(10),sal number(10)) partition by hash(sal) partitions 5;

If we want views partition information using USER\_TAB\_PARTITIONS data dictionary.

Sql> desc user\_tab\_partitions;

Sql> select partition\_name from user\_tab\_partitions where table\_name=’TEST’;

**HIERARCHICAL QUERIES:**

In relational database we can also stored hierarchical data. If we want to store hierarchical data then that table contains minimum three columns and also two columns must belongs to same datatype and relations. In oracle if we want to retrieve hierarchical data then we are using following clause. Whenever we are using hierarchical relational one column is null value mandatory.

1. Start With

2. Connect By

3. Level

**1. Start with:**

This clause is used to specify searching condition in hierarchy.

**Syntax:** start with condition

**2. Connect by:**

This clause is used to specify relationship between hierarchical (Parent, Child) column using prior operator.

**Syntax:** connect by prior parentcolname=childcolname

**3. Level:**

Oracle provided LEVEL pseudo column. In hierarchical is pseudo column which assign numbers to each level in tree structure.

**Syntax:** select level,colname1,colname2,………………..

From tablename where condition

Start with condition

Connect by prior parentcolname=childcolname;

**Q) Write hierarchical query to display then employee’s who are working under KING from emp table using hierarchy?**

Sql> select level,ename from emp start with ename=’KING’ connect by prior empno=mgr;

**LEVEL** **ENAME**

1 KING

2 JONES

3 SCOTT

4 ADAMS

3 FORD

**--- --------**

**(Or)**

Sql> select level,lpad(' ',level\*2)||ename from emp start with ename=’KING’

connect by prior empno=mgr;

**LEVEL** **LPAD ('', LEVEL\*2)||ENAME**

1 KING

2 JONES

3 SCOTT

4 ADAMS

3 FORD

4 SMITH

2 BLAKE

3 ALLEN

3 WARD

3 MARTIN

3 TURNER

3 JAMES

2 CLARK

3 MILLER

**KING**

**JONES BLAKE CLARK**

**SCOTT FORD ALLEN WARD MARTIN TURNER JAMES MILLER**

**ADAMS SMITH**

**NOTE:**

Oracle 9i introduced SYS\_CONNECT\_BY\_PATH() which returns path of the hierarchy in tree structure.

**Syntax:** SYS\_CONNECT\_BY\_PATH(columnname,’delimitername’)

Sql> select level,sys\_connect\_by\_path(ename,'--->') "emp names tree structure" from emp start with mgr is null connect by prior empno=mgr;

**LEVEL** **employee names tree structure**

1 --->KING

2 --->KING--->JONES

3 --->KING--->JONES--->SCOTT

4 --->KING--->JONES--->SCOTT--->ADAMS

3 --->KING--->JONES--->FORD

4 --->KING--->JONES--->FORD--->SMITH

2 --->KING--->BLAKE

3 --->KING--->BLAKE--->ALLEN

3 --->KING--->BLAKE--->WARD

3 --->KING--->BLAKE--->MARTIN

3 --->KING--->BLAKE--->TURNER

3 --->KING--->BLAKE--->JAMES

2 --->KING--->CLARK

3 --->KING--->CLARK--->MILLER

**Q) Write a query to display the emp’s who are working under BLAKE from emp table using hierarchy?**

Sql> select level,sys\_connect\_by\_path(ename,'--->') "employee names tree structure" from emp start with ename='BLAKE' connect by prior empno=mgr;

**LEVEL** **employee names tree structure**

1 --->BLAKE

2 --->BLAKE--->ALLEN

2 --->BLAKE--->WARD

2 --->BLAKE--->MARTIN

2 --->BLAKE--->TURNER

2 --->BLAKE--->JAMES

SQL> select level,sys\_connect\_by\_path(ename,'--->') "employee names tree structure" from emp start with ename='MILLER' connect by empno=prior mgr;

**LEVEL** **employee names tree structure**

1 --->MILLER

2 --->MILLER--->CLARK

3 --->MILLER--->CLARK--->KING

**NOTE:**

Prior is a unary operator which specifies parent column i.e oracle server searching data.

**NOTE:**

Oracle 10g introduced CONNECT\_BY\_ISLEAF pseudo column which returns 1 to the leaf node and 0 to the all other nodes.

Sql> select level,ename,connect\_by\_isleaf from emp start with mgr is null connect by prior empno=mgr;

**LEVEL** **ENAME CONNECT\_BY\_ISLEAF**

1 KING 0

2 JONES 0

3 SCOTT 0

4 ADAMS 1

3 FORD 0

4 SMITH 1

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

**NOTE:**

Oracle 10g introduced CONNECT\_BY\_ROOT operator which returns root node in the hierarchy.

**Syntax:** connect\_by\_root columnname

Sql> select level,ename,connect\_by\_root ename "RootnodeENAME" from emp start with mgr is null connect by prior empno=mgr;

**LEVEL** **ENAME** **RootnodeENAME**

1 KING KING

2 JONES KING

3 SCOTT KING

4 ADAMS KING

3 FORD KING

**NVL():**

Nvl() is predefined function which is to replace userdefined function in place of null.

**Syntax:** NVL(exp1,exp2)

If expression1 is null then its return expression2 otherwise its returns expression1.

**NVL2():**

Oracle 9i introduced NVL2() function this function excepts three parameter.

**Syntax:** NVL2(exp1,exp2,exp3)

If exp1 is null then its returns exp3 otherwise its returns exp2.

Sql> select nvl2(null,30,60) from dual;

**NVL2 (NULL, 30, 60)**

60

Sql> select nvl2(10,30,60) from dual;

**NVL2 (10, 30, 60)**

30

**Q) Write a query to update emp’s based following condition using NVL2() function?**

**1) if comm is null then update comm ------🡪 500**

**2) if comm is not null then update comm --🡪 200**

Sql> update emp set comm=nvl2(comm,comm+200,500);

Sql> select \* from emp;

**COALESCE():**

Coalesce() is an ASCII function which return first not null value in the given expression coalesce() no.of expression.

**Syntax:** coalesce(exp1,exp2,……………..)

Sql> select coalesce(null,null,10,null,30) from dual;

**COALESCE(NULL,NULL,10,NULL,30)**

10

Sql> select coalesce(null,30,null,10,null,50) from dual;

**COALESCE(NULL,30,NULL,10,NULL,50)**

30

Sql> select coalesce(40,30,null,10,null,50) from dual;

**COALESCE(40,30,NULL,10,NULL,50)**

40

Sql> select ename,comm,sal,coalesce(comm,sal) from emp;

**ENAME** **COMM** **SAL** **COALESCE(COMM,SAL)**

SMITH 800 800

ALLEN 300 1600 300

WARD 500 1250 500

JONES 2975 2975

MARTIN 1400 1250 1400

BLAKE 2850 2850

Nvl() internally uses implicit conversion. Whereas coalesce() does not uses implicit conversion and also coalesce() performance very high compare to nvl().

Sql> select nvl('A',sysdate) from dual;

**NVL ('A',S**

A

Sql> select coalesce('A',sysdate) from dual;

**ERROR:** ORA-00932: inconsistent datatype: expected CHAR got DATE

**ROLLBACK:**

Rollback is nothing but undo from query in oracle when ever using rollback oracle server automatically past undo area into dirty reader within buffer cache.

**Syntax:** rollback;

**FLASHBACK QUERY:**

Oracle 9i introduced flashback query. Flashback query allows content of a table to be retrieved with reference to a specific point of time using “AS OF” clause.

Flashback query are handle by administrator using flashback query then we are retrieving accidental data flashback query works based on undo file to retrieve data using flashback query following two methods.

1. Using timestamp datatype

2. Using SCN (system Change Number)

**1. Using Timestamp:**

**Syntax:** select \* from tablename AS OF timestamp(specifictime)

Sql> create table test(sno number(10),name varchar2(15));

Sql> insert into test values(&sno,'&name');

Sql> select \* from test;

**SNO** **NAME**

100 simha

101 narasimha

102 reddy

103 narasimha reddy

Sql> commit;

Sql> insert into test values(11,'shobha');

Sql> select \* from test;

**SNO** **NAME**

100 simha

101 narasimha

102 reddy

103 narasimha reddy

104 shobha

Sql> commit;

Sql> delete from test;

Sql> select \* from test;

No Rows Selected

Sql> select \* from test as of timestamp(systimestamp-interval '2' minute);

**SNO** **NAME**

100 simha

101 narasimha

102 reddy

103 narasimha reddy

104 shobha

**2. Using SCN:**

Whenever we are using transaction oracle server automatically generates a unique identification number in undo file. This number is called as “SCN”. If we want to view number then we are using CURRENT\_SCN property from V$DATABASE data dictionary.

**Syntax:** select current\_scn from v$database;

To retrieve data:

**Syntax:** select \* from tablename as of scn scnnumber;

Sql> conn sys as sysdba

Enter password: SYS

Sql> create table ABC(sno number(5));

Sql> desc v$database;

Sql> select current\_scn from v$database;

**CURRENT\_SCN**

6048901

Sql> insert into ABC values(50);

Sql> select \* from ABC;

**SNO**

50

Sql> select count(\*) from ABC;

**COUNT (\*)**

1

Sql> select count(\*) from abc as of scn 6048901;

**COUNT (\*)**

**Chapter-18(PL/SQL)**

**Functions**

**Procedural Language extensions to the Structured Query Language**

**Advantage of Using PL/SQL**

1. Better performance, as SQL is executed in bulk rather than a single statement
2. High Productivity
3. Tight integration with SQL
4. Full Portability
5. Tight Security
6. Support Object Oriented Programming concepts.

* Functions are a standalone block that is mainly used for calculation purpose.
* Function use RETURN keyword to return the value, and the datatype of this is defined at the time of creation.
* A Function should either return a value or raise the exception, i.e. return is mandatory in functions.
* Function with no DML statements can be directly called in SELECT query whereas the function with DML operation can only be called from other PL/SQL blocks.
* It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
* It contains declaration part (optional), execution part, exception handling part (optional).
* The values can be passed into the function or fetched from the procedure through the parameters.
* These parameters should be included in the calling statement.
* Function can also return the value through OUT parameters other than using RETURN.
* Since it will always return the value, in calling statement it always accompanies with assignment operator to populate the variables.

CREATE OR REPLACE FUNCTION totalCustomers

RETURN number IS

total number(2) := 0;

BEGIN

SELECT count(\*) into total

FROM customers;

RETURN total;

END;

**Chapter-19**

**PROCUDURE**

* Procedures are standalone blocks of a program that can be stored in the database.
* Call to these procedures can be made by referring to their name, to execute the PL/SQL statements.
* It is mainly used to execute a process in PL/SQL.
* It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
* It contains declaration part (optional), execution part, exception handling part (optional).
* The values can be passed into the procedure or fetched from the procedure through parameters.
* These parameters should be included in the calling statement.
* Procedure can have a RETURN statement to return the control to the calling block, but it cannot return any values through the RETURN statement.
* Procedures cannot be called directly from SELECT statements. They can be called from another block or through EXEC keyword.

CREATE OR REPLACE PROCEDURE welcome\_msg (p\_name IN VARCHAR2)

IS

BEGIN

dbms\_output.put\_line (‘Welcome '|| p\_name);

END;

**Procedure Vs. Function: Key Differences**

| **Procedure** | **Function** |
| --- | --- |
| * Used mainly to a execute certain process | * Used mainly to perform some calculation |
| * Cannot call in SELECT statement | * A Function that contains no DML statements can be called in SELECT statement |
| * Use OUT parameter to return the value | * Use RETURN to return the value |
| * It is not mandatory to return the value | * It is mandatory to return the value |
| * RETURN will simply exit the control from subprogram. | * RETURN will exit the control from subprogram and also returns the value |
| * Return datatype will not be specified at the time of creation | * Return datatype is mandatory at the time of creation |

**Chapter-20**

**Trigger**

**Thas 3 main components**

* **Triggering SQL Statement: –** This is the DML statement which causes the triggers to be invoked. That is it tells when to call the trigger – before or after, on which DML statement of the table – INSERT/ UPDATE/ DELETE and whether to call trigger when whole table is processed or only few columns are processed.  BEFORE and AFTER is used on tables and INSTEAD OF is used on views to create triggers. For example, it tells when to call the trigger to insert the logs – whether to call it on Inserting, deleting or updating MARKS table, before or after updating the MARKS table, whether to call the trigger on updating all columns of MARKS table or on particular columns of MARKS table.
* **Trigger Restriction: –** this is the part of trigger which tells how many times the trigger needs to be executed. It informs, if the trigger has to be called for each row insert /update/delete, or only once for the transaction. In our example, it would be for each row, because, we need to have log for each row of student marks update. There would be some cases where if we perform some DML statement on table, say first and last entry of days’ transaction in the supermarket billing to be logged in some other table. In this case, each INSERT will not call the trigger, instead it will call only twice in a day to log the entry.
* **Trigger Action: –** this part will actually perform set of transaction as result of original DML statement. For example, inserting the records into log tables. It need not be inserting logs always. It can be any transaction. For example, when date of birth of a student is inserted, trigger might calculate his age and insert into the same table.

CREATE OR REPLACETRIGGER tr\_marks\_log -- creates a trigger, if exists replaces it

AFTER UPDATE OF m.sub\_mark ON MARKS – after updating MARKS table’s sub\_mark column

FOR EACH ROW -- for each row update

BEGIN

INSERT INTO marks\_log (STD\_ID, SUBJECT, OLD\_MARK, NEW\_MARK)

VALUES (:old.std\_id,:old. subject, :old.sub\_mark, :new.sub\_mark);

EXCEPTION

When OTHERS THEN

raise\_application\_error (-20015, ‘Error while inserting marks log’);

END;

**Types of Triggers**

There are two types of triggers.

* **Row level trigger: –** Row level trigger is executed when each row of the table is inserted/ updated/ deleted. If it is a row level trigger, then we have to explicitly specify while creating the trigger, as we did in the above example. Also, we have to specify the WHEN (condition) in the trigger.
* **Statement level trigger: –** this trigger will be executed only once for DML statement. This DML statement may insert / delete/ update one row or multiple rows or whole table. Irrespective of number of rows, this trigger will be fired for the statement. If we have not specified the type of trigger while creating, by default it would be a statement level trigger.

In below example, once the Students are inserted, it calculates the age of the student and updates STUDENT table.

CREATE OR REPLACETRIGGER tr\_calc\_age

AFTER INSERT ON STUDENT

BEGIN

UPDATE STUDENT s

SET AGE = SYSDATE – s.DATE\_OF\_BIRTH

WHERE s.STD\_ID =:old.STD\_ID;

EXCEPTION

When OTHERS THEN

raise\_application\_error (-20015, ‘Error while inserting marks log’);

END;

In both the cases, the DML statement may be on particular column(s) or on whole table.

In addition to above types of trigger, we can have triggers which are called so because of the time when they are executed.

* **BEFORE trigger: –** This trigger is called before the execution of the DML statement. This BEFORE trigger can be used for some condition check or it can be used to alter the whole DML statement so that it cannot be executed on the table. For example, if the student age is less than 10, don’t allow to insert the record into the table.
* **After Trigger: –** this trigger is called after once DML statement is executed. It can perform any kind of transaction.
* **Combination of triggers: –** We can have combination of row, statement, BEFORE and AFTER triggers.
  + **BEFORE STATEMENT: –** This trigger is executed only once before executing the DML statement.
  + **BEFORE ROW: –** This trigger is executed for each row of the table, but before the DML execution.
  + **AFTER STATEMENT:-** This trigger is executed only once after the DML execution is complete
  + **AFTER ROW:** – This trigger is executed once the DML statement is complete, but for each row of the table.

**Some of the key points about trigger**

* Triggers do not have commit or rollback or savepoint in them. All the commit and rollback are mentioned in the main DML statement. If DML transaction is committed or rolled back, the transaction in trigger is also committed or rolled back. This is because, if we give commit or rollback in the trigger, it commits or rollbacks the DML statements too. But this is not desirable, as the triggers are meant to perform the transaction as a result of DML statements. Its purpose is not to confirm the DML transaction, but to perform additional transaction as a part of DML statement.
* There is no :OLD values in INSERT triggers since it is totally a new record.
* There is no :NEW values in DELETE triggers, as we are deleting the record.
* If there are multiple triggers defined on the table, there are no specific criteria to execute them. All of them can perform simultaneously. Hence we might have to have another trigger to execute all these triggers in the order.
* A trigger cannot change the table from which it has been called. If there is a trigger called as a result of insert on a table, then the trigger cannot insert/update the same table. This is because we have a lock on the table already and it will create dead lock situation.

**Execution Order**

We can have multiple types triggers on the same table. Since there is BEFORE, AFTER, INSTEAD OF, ROW level, STATEMENT level etc, there should be some order for it to execute. The order of execution of trigger is as follows.

* BEFORE statement trigger is always called first. It is called before executing the whole DML statement and affecting the table.
* BEFORE row level trigger is then called for each row.
* Once the DML statement is executed, AFTER row level trigger is called for each row of the table.
* At the end, AFTER statement level trigger is called.

**Note:**this is the order in which single trigger with different types will be executed.

**Mutating tables**

Suppose we have inserted some records into STUDENT table. We have created a row level trigger on the same table to compute the total number of records after the insertion. What is wrong in this situation?  It looks perfect right? But it will throw a mutating table error. What happens in this case is, STUDENT table is inserted with records. The row level trigger will be fired after each row insertion. But transaction on STUDENT is still not complete and trigger is trying to access STUDENT. There will be INSERT lock on STUDENT which is not released for trigger to select the count. Hence trigger will be waiting for the insert to complete. But the insert is waiting for the trigger to complete to proceed for the next row insertion or to complete the transaction. Hence both will be waiting for each other to complete their transaction. Such a condition on the table is called mutating error.

Similarly, if there are two tables A and B, and two triggers  where one of the trigger will be updating the table B when there is insertion on A and another trigger will be inserting on the table A when there is update on B. In this case both the tables will be waiting for each other. This is also another condition of mutating tables. It is a kind of deadlock situation when triggers are used.

**Solution for mutating tables**

* Avoid triggers: – Triggers are not called by the users. They are implicitly called by DML statements. Hence we will not have control on triggers. In addition, there will be multiple triggers on the same table, which will contradict with one another leading to mutation. Hence better not to use them as much as possible.
* Usage of AFTER or INSTEAD OF triggers will avoid mutation. If we are using AFTER trigger, it will be executed once the DML transaction is complete. Hence lock on the table would have been released. If we are using INSTEAD OF trigger, it will undo the DML transaction and then it will perform trigger transaction. Hence there will not be any mutation.
* Try to avoid DML transaction on the same calling table in the triggers. This will not create any lock on the table and hence no mutation.

**Chapter-21**

# Cursors

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

* Implicit cursors
* Explicit cursors

## Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

## Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

CURSOR cursor\_name IS select\_statement;

**Chapter-21**

analytic-functions

**RANK, DENSE\_RANK, ROW Number**

SELECT deptno, AVG(sal)

FROM emp

GROUP BY deptno

ORDER BY deptno;

SELECT empno, deptno, sal,

AVG(sal) OVER (PARTITION BY deptno) AS avg\_dept\_sal

FROM emp;

SELECT empno, deptno, sal,

AVG(sal) OVER (PARTITION BY deptno ORDER BY sal

RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS range\_avg,

AVG(sal) OVER (PARTITION BY deptno ORDER BY sal

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS rows\_avg

FROM emp;

SELECT empno,

deptno,

sal,

RANK() OVER (PARTITION BY deptno ORDER BY sal) AS myrank

FROM emp;

EMPNO DEPTNO SAL MYRANK

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

7934 10 1300 1

7782 10 2450 2

7839 10 5000 3

7369 20 800 1

7876 20 1100 2

7566 20 2975 3

7788 20 3000 4

7902 20 3000 4

7900 30 950 1

**7654 30 1250 2**

**7521 30 1250 2**

**7844 30 1500 4**

7499 30 1600 5

7698 30 2850 6

SELECT \*

FROM (SELECT empno,

deptno,

sal,

RANK() OVER (PARTITION BY deptno ORDER BY sal) AS myrank

FROM emp)

WHERE myrank <= 2;

EMPNO DEPTNO SAL MYRANK

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

7934 10 1300 1

7782 10 2450 2

7369 20 800 1

7876 20 1100 2

7900 30 950 1

7521 30 1250 2

7654 30 1250 2

SQL>

SELECT empno,

deptno,

sal,

DENSE\_RANK() OVER (PARTITION BY deptno ORDER BY sal) AS myrank

FROM emp;

EMPNO DEPTNO SAL MYRANK

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

7934 10 1300 1

7782 10 2450 2

7839 10 5000 3

7369 20 800 1

7876 20 1100 2

7566 20 2975 3

7788 20 3000 4

7902 20 3000 4

7900 30 950 1

**7654 30 1250 2**

**7521 30 1250 2**

**7844 30 1500 3**

7499 30 1600 4

7698 30 2850 5

SELECT department\_id, last\_name, employee\_id, ROW\_NUMBER()

OVER (PARTITION BY department\_id ORDER BY employee\_id) AS emp\_id

FROM employees;

DEPARTMENT\_ID LAST\_NAME EMPLOYEE\_ID EMP\_ID

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

10 Whalen 200 1

20 Hartstein 201 1

20 Fay 202 2

30 Raphaely 114 1

30 Khoo 115 2

30 Baida 116 3

30 Tobias 117 4

30 Himuro 118 5

30 Colmenares 119 6

40 Mavris 203 1

. . .

100 Popp 113 6

110 Higgins 205 1

110 Gietz 206 2