**Difference Between ROWID AND ROWNUM**

**Difference between ROWID and ROWNUM.**

|  |  |
| --- | --- |
| **Row id** | **RowNum** |
| 1.Physical address of the rows.  2.Rowid is permanent  3.Rowid is 16-bit hexadecimal  4.Rowid gives address of rows or records  5. Rowid is automatically generated unique id of a row and it is generated at the time of insertion of row.  6. ROWID is the fastest means of accessing data.  7. They are unique identifiers for the any row in a table. | 1. Rownum is the sequential number, allocated to each returned row during query execution.  2. Rownum is temporary.  3.Rownum is numeric  4.Rownum gives count of records  5. Rownum is an dynamic value automatically  retrieved along with select statement output.  6. It represents the sequential order in which Oracle has retrieved the row. |

**Difference between VIEW and SYNONYM.**

**Difference between VIEW and SYNONYM.**

|  |  |
| --- | --- |
| view | synonym |
| 1.   It doesn’t contain data itself.       2.view occupies no space for data       3.view has only query | 1. It contain data itself.  2.synonym occupy space for data  3.synonym can be used as an alias name for the table name |

**Difference Between TRIGGER AND PROCEDURE**

  **Trigger Vs Procedure**

|  |  |
| --- | --- |
| **Trigger** | **Procedure** |
| 1.   Trigger does not accept parameters.  2.   A trigger is executed implicitly by the oracle engine. | 1. Procedure accepts the parameters.  2.Procedure is execute explicitly, called by a user |

**Difference Between FUNCTION AND PROCEDURE**

**Procedure Vs Function**

|  |  |
| --- | --- |
| **Procedure** | **Function** |
| 1. Procedure doesn’t contain return clause in header & executable part.  2. Procedure may be return or may not be return a value.  3. Procedure can’t be called into select statements.  4. Procedure are used to implement business logic. | 1. Function contain return clause in header & executable part.  2. Function always returns a value.  3. Function can be called into select statement.  4. Function are used for numeric calculation. |

**Difference Between WHERE AND HAVING**

**WHERE Vs HAVING**

|  |  |
| --- | --- |
| **Where** | **Having** |
| 1.Without group by clause  2. Where clause select Row before grouping.  3. Where clause can’t contain aggregate function. | 1.With group by clause  2.Having clause select after grouping  3.Having clause contain aggregate  Function. |

**Difference Between JOIN & UNION**

**Difference between JOIN and UNION**

|  |  |
| --- | --- |
| **Join** | **Union** |
| 1.   Join the columns.  2.   Duplicate are allowed  3.   Combine the column based on condition. | 1.Merge the row  2.Duplicates are not allowed  3. Combine the result of two select statements. |

**Difference Between DROP, DELETE & TRUNCATE**

**Difference between DROP, DELETE and TRUNCATE**

|  |  |  |
| --- | --- | --- |
| **Drop** | **Delete** | **Truncate** |
| 1. Whole structure and data will drop.  2.can’t  Rollback  3.can’t use where clause  4.All data and structure  drop simultaneously | 1. Structure will not drop, data will drop.  2.can Rollback  3.can use WHERE clause  4. Row by Row data Delete | 1. Structure will not drop data will drop.  2.can’t Rollback  3.can’t use WHERE clause  4. Only all data will drop, truncate operation drop and recreate the table. |

**Difference Between Star schema and Snowflake Schema**

|  |  |
| --- | --- |
| **Star schema** | **Snowflake Schema** |
| 1. De-Normalized Data Structure  2. Category wise Single Dimension Table  3. More data dependency and redundancy  4. No need to use complicated join  5. Query Results Faster  6. No Parent Table  7. Simple DB Structure | 1. Normalized Data Structure  2. Dimension table split into many pieces  3. less data dependency and No redundancy  4. Complicated Join  5. Some delay in Query Processing  6. It May contain Parent Table  7. Complicated DB Structure |

############################################# TOP 2 SAL FROM EMP TABLE #################################################  
  
  
**select** \* **from** (**select** \* **from** emp **order** **by** sal **desc**) **where** rownum <3;

Nth max sal (<http://www.oratable.com/nth-highest-salary-in-oracle/>)

#### Scenario 1: DENSE\_RANK () for Nth highest row

**select** ename ,sal ,dense\_rank() over (**order** **by** sal **desc**) ranking

**from** emp;

O/P:

ENAME             SAL    RANKING

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

KING             5000          1

FORD             3000          2

SCOTT            3000          2

JONES            2975          3

CLARK            2850          4

BLAKE            2850          4

ALLEN            1600          5

**select** \*

**from**

(

**select** ename

,sal

,dense\_rank() over (**order** **by** sal **desc**) ranking

**from** emp

)

**where** ranking = 4 *-- Replace 4 with any value of N*

ENAME             SAL    RANKING

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

BLAKE            2850          4

CLARK            2850          4

#### Scenario 2: RANK () for Nth highest row

**select** ename

,sal

,rank() over (**order** **by** sal **desc**) ranking

**from** emp;

ENAME             SAL    RANKING

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

KING             5000          1

FORD             3000          2

SCOTT            3000          2

JONES            2975          4

CLARK            2850          5

BLAKE            2850          5

ALLEN            1600          7

TURNER           1500          8

**select** \*

**from**

(

**select** ename

,sal

,rank() over (**order** **by** sal **desc**) ranking

**from** emp

)

**where** ranking = 4 *-- Replace 4 with any value of N*

ENAME             SAL    RANKING

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

JONES            2975          4

Scenario\_3

**select** ename

, sal

**from** emp a

**where** 3 = ( **select** count(\*) *-- Replace 3 with any value of (N - 1)*

**from** emp b

**where** b.sal > a.sal)

ENAME             SAL

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

JONES            2975

**select** \* **from** emp r1

**where** &n = (**select** count(\*) **from** emp

**where** sal >= r1.sal)

Remove the duplicate records

**Method 1:**

**DELETE** **FROM** table\_name A **WHERE** **ROWID** > (

**SELECT** min(**rowid**) **FROM** table\_name B

**WHERE** A.key\_values = B.key\_values);

**Method 2:**

**create** **table** table\_name2 **as** **select** **distinct** \* **from** table\_name1;

**SQL**> **drop** **table** table\_name1;

**SQL**> **rename** table\_name2 **to** table\_name1;

**Method 3:**

**delete** **from** my\_table t1

**where** **exists** (**select** 'x' **from** my\_table t2

**where** t2.key\_value1 = t1.key\_value1

**and** t2.key\_value2 = t1.key\_value2

**and** t2.**rowid** > t1.**rowid**);

[**Using dense\_rank()**](http://www.orafaq.com/faq/how_does_one_eliminate_duplicates_rows_from_a_table#comment-1076)

**delete** **from** table\_name

**where** **rowid** **in**

(**select** rn **from**

(**select** **rowid** rn, dense\_rank() over (**partition** **by** col1, col2, ..**order** **by** **rowid**) **from** table\_name ) **where** rn <> 1

)

Triggers:

**SQL**> **create** **or** **replace** **trigger** emp\_delete\_trigger **before** **delete** **on** emp

2 **for** **each** **row**

3 **begin**

4 **insert** **into** emp\_delete(empno,ename,sal,deptno) **values**(:**old**.empno,**user**,:**old**.sal,:**old**.deptno);

5 **end** emp\_delete\_trigger;

6 /

**Trigger** created.

**SQL**> ed

Wrote file afiedt.buf

1 **create** **or** **replace** **trigger** emp\_delete\_trigger **before** **insert** **on** emp

2 **for** **each** **row**

3 **begin**

4 **if** :**new**.sal >6000 **then**

5 raise\_application\_eroor(-20888,'the sal is too high in this organization');

6\* **end** emp\_delete\_trigger;

**SQL**> /

**Warning**: **Trigger** created **with** compilation **errors**.

**SQL**> **show** **errors**

**Errors** **for** **TRIGGER** EMP\_DELETE\_TRIGGER:

LINE/COL **ERROR**

*-------- -----------------------------------------------------------------*

4/5 PLS-00103: Encountered the symbol "EMP\_DELETE\_TRIGGER" **when**

expecting one **of** the following:

**if**

**SQL**> ed

Wrote file afiedt.buf

1 **create** **or** **replace** **trigger** emp\_insert\_trigger **before** **insert** **on** emp

2 **for** **each** **row**

3 **begin**

4 **if** :**new**.sal >6000 **then**

5 raise\_application\_eroor(-20888,'the sal is too high in this organization');

6\* **end** emp\_delete\_trigger;

**SQL**> /

**Warning**: **Trigger** created **with** compilation **errors**.

**SQL**> ed

Wrote file afiedt.buf

1 **create** **or** **replace** **trigger** emp\_insert\_trigger **before** **insert** **on** emp

2 **for** **each** **row**

3 **begin**

4 **if** :**new**.sal >6000 **then**

5 raise\_application\_eroor(-20888,'the sal is too high in this organization');

6 **end if**;

7\* **end** emp\_insert\_trigger;

**SQL**> /

**Warning**: **Trigger** created **with** compilation **errors**.

**SQL**> sho erros

SP2-0158: **unknown** **SHOW** **option** "erros"

**SQL**> **show** **errors**

**Errors** **for** **TRIGGER** EMP\_INSERT\_TRIGGER:

LINE/COL **ERROR**

*-------- -----------------------------------------------------------------*

3/1 PLS-00201: identifier 'RAISE\_APPLICATION\_EROOR' must be declared

3/1 PL/**SQL**: **Statement** ignored

Constrains:

**DATA** INTEGRITY

**Data** Integrity means **data** validation **or** **data** checking process **or** **Type** Checking process. **Before** storing **user** supplied information **into** the **table** server performs **data** integrity process **in** **order** **to** verify whether **user** supplying valid information **or** **not**.

**If** **user** supplies valid information **then** **only** it will stored **into** the **table** otherwise server raises an **error** message **like** ‘**Data** **Type** Mismatch’.

We can achieve this **Data** Integrity **in** Three ways

1. **Data** Types (see previous concepts)

2. Constraints

3. Triggers

Constraints: **Constraint** **is** nothing **but** condition **on** **column**. **If** we perform **any** operation against **to** **constraint** server raises an **error** message.

**OR**

It **is** a mechanism automatically activated **when** **user** performs DML operations **on** the **table**.

We can place constraints (**with** **constraint** names **or** **with** **out** **constraint** names) **on** **columns** **while** creating the **table** **or** **after** creating the **table**.

TYPES **OF** CONSTRAINTS:

1) **Unique** **Constraint**: **When** we place **UNIQUE** **constraint** **on** **any** **column** ,It will **not** allow duplicate **values** **but** it allows single **null** **value**

2) **Not** **null** **Constraint**: **When** we place **NOT** **NULL** **constraint** **on** **any** **column** , it will **not** allow **any** **null** **values**. Entering **value** **for** that **column** **is** mandatory.

3) **Check** **Constraint**: It **is** used **for** evaluating **range** condition **on** numeric **columns**. It will **check** **values** provided **for** **column**. **Like** salary should greater than 5000 **and** less than 40000

It **is** used **for** evaluating character comparison conditions **on** character **columns**

4) **Primary** **Key** **Constraint**:

It **is** a combination **of** **UNIQUE**+ **NOT** **NULL** + CLUSTERED **INDEX**.

It means **when** we place **PRIMARY** **KEY** **constraint** **on** **any** **column** **then** it will **not** **any** duplicate **values** **and** it does **not** accept **any** **null** **values** mean time the **data** **in** that **column** will be arranged **in** ascending **order** due **to** CLUSTERED **INDEX**.

5) **Foreign** **key** **Constraint**:

a. **Foreign** **Key** must be **Primary** **Key**

b. **Foreign** **Key** can accept duplicate **values** **and** **Null** **values**

c. **Foreign** **Key** has **to** take the **values** **from** its corresponding **Primary** **Key**.

6) **Default** **Constraint**: It **is** useful **to** provide **default** **value** **into** a **column** **when** **user** will **not** provide **any** **value** **while** inserting the **data** **into** the **table** **then** **default** **value** will be arranged.

Note:

1. **Only** one **PRIMARY** **KEY** **is** allowed per **table**.

2. **PRIMARY** **KEY** **table** **is** **called** parent **table** **and** **FOREIGN** **KEY** **table** **is** **called** child **table**.

3. **PRIMARY** **KEY** **column** **is** **KEY** **COLUMN** **and** the rest **of** the **columns** **in** the same **table** are **called** NON-**KEY** **COLUMNS**.

4. **While** providing **FOREIGN** **KEY** we should have **to** **give** the reference **of** its corresponding **PRIMARY** **KEY**.

5. one **PRIMARY** can be placed **on** more than one **column** **then** that **primary** **is** **called** COMPOSITE **PRIMARY** **KEY**

Constraints can be added **to** **table** **in** two levels

1. **Column** **Level**

2. **Table** **Level**

1.**Column** **level** constraints: Here constraints are going **to** placed **on** **columns**, **after** the definition **of** **each** **and** every individual **column** **and** their corresponding **data** **type**.

Syntax: (**With** **out** **Constraint** names)

**CREATE** **TABLE** TABLENAME

(**COLUMN** 1 **DATA** **TYPE** **CONSTRAINT** **TYPE**,

**COLUMN** 2 **DATA** **TYPE** **CONSTRAINT** **TYPE**,

*--------------------------------------------------------)*

E.g.:

**CREATE** **TABLE** DEPT

(DEPTNO INT **PRIMARY** **KEY**,

DNAME VARCHAR (20) **UNIQUE**,

LOC VARCHAR (10) **DEFAULT** ‘HYD’)

E.g.:

**CREATE** **TABLE** EMP

(EMPNO INT **PRIMARY** **KEY**,

ENAME VARCHAR (20) **NOT** **NULL**,

SAL MONEY **CHECK** (SAL>=1000),

DEPTNO INT **FOREIGN** **KEY** **REFERENCES** DEPT (DEPTNO)

**ON** **DELETE** CASCADE

**ON** **UPDATE** CASCADE)

Advantage **of** **ON** **DELETE** CASCADE:

**With** **out** specifying the **ON** **DELETE** CASCADE it **is** **not** possible **to** **delete** the record **in** the PARENT **table** **if** there are dependent records **from** the child **table** **for** that record. **ON** **DELETE** CASCADE **if** used **when** the record **in** the PARENT **table** **is** deleted **all** the dependent records **in** the child **table** will be also be deleted.

Advantage **of** **ON** **UPDATE** CASCADE:

**With** **out** specifying the **ON** **UPDATE** CASCADE it **is** **not** possible **to** **update** the record **in** the PARENT **table** **if** there are dependent records **from** the child **table** **for** that record. **ON** **UPDATE** CASCADE **if** used **when** the record **in** the PARENT **table** **is** updated **all** the dependent records **in** the child **table** will be also be updated

Syntax: (**With** **Constraint** names)

**CREATE** **TABLE** TABLENAME

(**COLUMN** 1 **DATA** **TYPE** **CONSTRAINT** CONSTRAINTNAME **CONSTRAINT** **TYPE**,

**COLUMN** 2 **DATA** **TYPE** **CONSTRAINT** CONSTRAINTNAME **CONSTRAINT** **TYPE**, *--------------------------------------------------------)*

E.g.:

**CREATE** **TABLE** DEPT

(DEPTNO INT **CONSTRAINT** PK **PRIMARY** **KEY**,

DNAME VARCHAR (20) **CONSTRAINT** UQ **UNIQUE**,

LOC VARCHAR (10) **CONSTRAINT** DF **DEFAULT** ‘HYD’)

**In** the above example **constraint** have been placed **on** **columns** **with** **constraint** names **like**

PK name **of** **Primary** **Key** **Constraint** **on** Deptno **column**

UQ name **of** **Unique** **Constraint** **on** Dname **column**

DF name **of** **Default** **Constraint** **on** Loc **column**

E.g.:

**CREATE** **TABLE** EMP

(EMPNO INT CONSTRAITN PRK **PRIMARY** **KEY**,

ENAME VARCHAR (20) **CONSTRAINT** NN **NOT** **NULL**,

SAL MONEY **CONSTRAINT** CK **CHECK** (SAL>=1000),

DEPTNO INT **FOREIGN** **KEY** **CONSTRAINT** FK **REFERENCES** DEPT (DEPTNO)

**ON** **DELETE** CASCADE

**ON** **UPDATE** CASCADE)

**In** the above example **constraint** have been placed **on** **columns** **with** **constraint** names **like**

PRK name **of** **Primary** **Key** **Constraint** **on** Empno **column**

NN name **of** **Not** **Null** **Constraint** **on** Ename **column**

CK name **of** **Check** **Constraint** **on** Sal **column**

FK name **of** **Foreign** **Key** **Constraint** **on** Deptno **column**

2.**Table** **level** constraints: Here constraints are going **to** be placed **on** **columns** **after** the definition **of** **all** **columns** **and** their corresponding **data** types. It means **at** the **end** **of** the **table** definition constraints will be placed **on** **columns**.

Note: **In** **Table** **Level** constraints **DEFAULT** **and** **NOT** **NULL** constraints are **not** allowed.

Syntax: (**With** **out** **Constraint** names)

**CREATE** **TABLE** TABLENAME

(COLUMN1 DATATYPE, COLUMN2 **DATA** **TYPE**,…………….,

**CONSTRAINT** **TYPE** (COLUMN1),

**CONSTRAINT** **TYPE** (COLUMN2),……………….)

E.g.:

**CREATE** **TABLE** DEPT

(DEPTNO INT, DNAME VARCHAR (20), LOC VARCHAR (20),

**PRIMARY** **KEY** (DEPTNO), **UNIQUE** (DNAME))

E.g.:

**CREATE** **TABLE** EMP

(EMPNO INT, ENAME VARCHAR (20), SAL MONEY, DEPTNO INT,

**PRIMARY** **KEY** (EMPNO), **CHECK** (SAL>=1000),

**FOREIGN** **KEY** (DEPTNO) **REFERENCES** DEPT (DEPTNO))

Syntax: (**With** **Constraint** names)

**CREATE** **TABLE** TABLENAME

(COLUMN1 DATATYPE, COLUMN2 **DATA** **TYPE**,…………….,

**CONSTRAINT** CONSTRAINTNAME **CONSTRAINT** **TYPE** (COLUMN1),

**CONSTRAINT** CONSTRAINTNAME **CONSTRAINT** **TYPE** (COLUMN2),

…………………………………)

E.g.:

**CREATE** **TABLE** DEPT

(DEPTNO INT, DNAME VARCHAR (20), LOC VARCHAR (20),

**CONSTRAINT** PK **PRIMARY** **KEY** (DEPTNO),

**CONSTRAINT** UQ **UNIQUE** (DNAME))

E.g.:

**CREATE** **TABLE** EMP

(EMPNO INT, ENAME VARCHAR (20), SAL MONEY, DEPTNO INT,

**CONSTRAINT** PRK **PRIMARY** **KEY** (EMPNO),

**CONSTRAINT** CK **CHECK** (SAL>=1000),

**CONSTRAINT** FK **FOREIGN** **KEY** (DEPTNO) **REFERENCES** DEPT (DEPTNO))

Adding constraints **for** the existing **table** **with** **out** **any** **constraint** names:

We can place constraints **on** **columns** **after** creating the **table**

Step1: **Create** **any** **table** **with** **out** **any** constraints

E.g.: **CREATE** **TABLE** DEPT

(DEPTNO INT, DNAME VARCHAR (20), LOC VARCHAR (20))

E.g.: **CREATE** **TABLE** EMP

(EMPNO INT, ENAME VARCHAR (20), SAL MONEY, DEPTNO INT)

Step2: Make a single **column** **as** **NOT** **NULL** **for** which we want provide **Primary** **Key** **constraint**

Syntax:

**ALTER** **TABLE** TABLENAME **ALTER** **COLUMN** COLUMNNAME DATATYPE **NOT** **NULL**

Note: **if** we want **to** provide **only** **NOT** **NULL** **constraint**, **then** we have **to** that process **in** this step itself.

E.g.:

**ALTER** **TABLE** DEPT **ALTER** **COLUMN** DEPTNO INT **NOT** **NULL**

E.g.:

**ALTER** **TABLE** EMP **ALTER** **COLUMN** EMPNO INT **NOT** **NULL**

Step3: **Add** your required constraints **to** **columns**

Syntax:

**ALTER** **TABLE** TABLENAME **ADD** CONSTRAINTTYPE (COLUMN1),

CONSTRAINTTYPE (COLUMN2),………..

E.g.:

**ALTER** **TABLE** DEPT **ADD** **PRIMARY** **KEY** (DEPTNO),

**UNIQUE** (DNAME),

**DEFAULT** ‘HYD’ **FOR** LOC

E.g.:

**ALTER** **TABLE** EMP **ADD** **PRIMARY** **KEY** (EMPNO),

**CHECK** (SAL>=1000),

**FOREIGN** **KEY** (DEPTNO) **REFERENCES** DEPT (DEPTNO)

Adding constraints **for** the existing **table** **with** **constraint** names:

Step1 **and** Step2 are common here also

Step3: **Add** your required constraints **to** **columns**

Syntax:

**ALTER** **TABLE** TABLENAME **ADD** **CONSTRAINT** CONSTRAINTNAME CONSTRAINTTYPE (COLUMN1), **CONSTRAINT** CONSTRAINTNAME CONSTRAINTTYPE (COLUMN2),………………..

E.g.:

**ALTER** **TABLE** DEPT **ADD** **CONSTRAINT** PK **PRIMARY** **KEY** (DEPTNO),

**CONSTRAINT** UQ **UNIQUE** (DNAME),

**CONSTRAINT** DF **DEFAULT** ‘HYD’ **FOR** LOC

E.g.:

**ALTER** **TABLE** EMP **ADD** **CONSTRAINT** PRK **PRIMARY** **KEY** (EMPNO),

**CONSTRAINT** CK **CHECK** (SAL>=1000),

**CONSTRAINT** FK **FOREIGN** **KEY** (DEPTNO) **REFERENCES** DEPT (DEPTNO)

Dropping The Constraints:

Syntax:

**ALTER** **TABLE** TABLENAME **DROP** **CONSTRAINT** CONSTRAINTNAME

E.g.: **ALTER** **TABLE** DEPT **DROP** **CONSTRAINT** PK

E.g.: **ALTER** **TABLE** EMP **DROP** **CONSTRAINT** PRK

Note: **If** you **create** **any** **table** **with** **out** **any** **constraint** names **then** server will arrange the **constraint** names **in** its own format. Those **constraint** names will be displayed **when** you **execute** a stored **procedure** SP\_HELPCONSTRAINT Table\_Name.

SP\_HELPCONSTRAINT:

This Stored **Procedure** **is** used **to** display the description **of** constraints which have been placed **on** different **columns** **of** a **specific** **table**.

Syntax:

SP\_HELPCONSTRAINT **Table**-Name

Ex:

SP\_HELPCONSTRAINT EMP

Indexes

INDEXES

Indexes **in** **SQL** server **is** similar **to** **index** **in** **text** book.. Indexes are used **to** improve the performance **of** queries.

INDEXES ARE GENERALLY CREATED **FOR** FOLLOWING **COLUMNS**

**Primary** **key** **column**

**Foreign** **key** **column**: frequently used **in** **join** conditions.

**Column** which are frequently used **in** **where** clause

**Columns**, which are used **to** **retrieve** the **data** **in** sorting **order**.

INDEXED CANNOT BE CREATED **FOR** FOLLOWING **COLUMNS**:

The **columns** which are **not** used frequently used **in** **where** clause.

**Columns** containing the duplicate **and** **null** **values**

**Columns** containing images, **binary** information, **and** **text** information.

TYPES **OF** INDEXES:

• CLUSTERED **INDEX**

• NON-CLUSTERED **INDEX**

CLUSTERED **INDEX**: **only** one clustered **index** **is** allowed per **table**. The **order** **of** **values** **in** a **table** **order** **of** **values** **in** **index** **is** also same. **When** **cluster** **index** **is** created **on** **table** **data** **is** arranged **in** ascending **order** **cluster** **index** will occupy 5% **of** the **table**.

Syntax:

**CREATE** [**UNIQUE**] CLUSTERED **INDEX** INDEXNAME **ON** TABLENAME (**COLUMN**)

E.g.:

**CREATE** CLUSTERED **INDEX** CI **ON** EMP (EMPNO)

Note: **if** we want **to** maintain **unique** **values** **in** clustered/non clustered indexed **column** **then** specify **UNIQUE** keyword along **with** CLUSTERED **INDEX**/NONCLUSTERD **INDEX**

NONCLUSTERED **INDEX**: It **is** the **default** **index** created **by** the server the physical **order** **of** the **data** **in** the **table** **is** different **from** the **order** **of** the **values** **in** **index**.

Max **no**. **Of** non-clustered indexed allowed **for** **table** **is** 249

Syntax:

**CREATE** [**UNIQUE**] NONCLUSTERED **INDEX** INDEXNAME

**ON** TABLENAME ( COLUMN1,…)

E.g.:

**CREATE** NONCLUSTERED **INDEX** NCI **ON** EMP (ENAME, SAL)

Ex:

**CREATE** **UNIQUE** NONCLUSTERED **INDEX** UI **ON** DEPT (DNAME)

COMPOSITE **INDEX**: **If** a **Unique** NonClustered **index** **is** created **on** more than one **column** **then** that concept **is** **called** composite **index**.

**CREATE** **UNIQUE** NONCLUSTERED **INDEX** COI **ON** DEPT (DEPTNO, DNAME)

DEPTNO DNAME

10 SALES

20 HR

30 IR

10 HR (Accepted)

20 SALES (Accepted)

30 IR (Repeated, **Not** accepted)

SP\_HELPINDEX: This stored **procedure** **is** used **to** display the list **of** indexes, which have been placed **on** different **columns** **of** a **specific** **table**.

E.g.: SP\_HELPINDEX EMP

Syntax **to** **drop** the **index**:

**DROP** **INDEX** TABLENAME.**INDEX**.NAME

E.g.:

**DROP** **INDEX** DEPT.UI

JOINS

JOINS

Joins **in** **SQL** Server are used **to** **select** the **data** **from** multiple tables **using** a single **select** **statement**.

T-**SQL** provides the **join** concept, which allows retrieval **of** **data** **from** more than one **table**. This concept **is** probably the most important **for** RDBMS, because it allows **data** **to** be spread over many tables.

**In** **SQL** Server there existed three types **of** joins which includes

1. **INNER** **JOIN**

2. OUER **JOIN**

3. **CROSS** **JOIN**

**Sample** Tables

EMP

EMPNO ENAME SAL DEPTNO

*----------- ---------- --------------------- -----------*

11 RAGHU 10000.0000 10

22 RAZ 20000.0000 20

33 AMAR 10000.0000 10

44 MANI 15000.0000 20

55 CHARN 15000.0000 40

66 ANIL 20000.0000 50

DEPT

DEPTNO DNAME LOC

*----------- ---------- ----------*

10 SALES HYD

20 HR CHE

30 IR BAN

1.**INNER** **JOIN**: **Inner** **join** selects the **data** **from** multiple tables based **on** the equality condition It means it selects **only** **matched** records **from** the multiple tables. **For** doing this **Inner** **join** operation we should have **to** maintain one common valued **column** **in** the multiple tables.

Syntax:

**SELECT** TABLE1.COLUMN1, TABLE1.COLUMN2,………,

TABLE2.COLUMN1, TABLE2.COLUMN2,……

**FROM** TABLE1 **INNER** **JOIN** TABLE2

**ON** TABLE1.COMMON **COLUMN**=TABLE2.COMMON **COLUMN**

E.g.:

**SELECT** EMP.EMPNO, EMP.ENAME,

DEPT.DEPTNO, DEPT.DNAME **FROM** EMP **INNER** **JOIN** DEPT

**ON** EMP.DEPTNO=DEPT.DEPTNO

EMPNO ENAME DEPTNO DNAME

*----------- ---------- ----------- ----------*

11 RAGHU 10 SALES

22 RAZ 20 HR

33 AMAR 10 SALES

44 MANI 20 HR

2. **OUTER** **JOIN**: It **is** the extension **of** **Inner** **Join** operation because **Inner** selects **only** **matched** records **from** multiple tables **where** **Outer** **join** selects **matched** records **as** well **as** unmatched records. It includes

a. **Left** **Outer** **Join**

b. **Right** **Outer** **Join**

c. **Full** **Outer** **Join**

a. **Left** **Outer** **Join**: It selects **matched** records **from** both the tables **as** well **as** unmatched records **from** **Left** side **table**. **For** doing this operation we have **to** keep a special symbol ‘\*’ **at** the **left** side **of** the equality condition.

Syntax:

**SELECT** TABLE1.COLUMN1, TABLE1.COLUMN2,………,

TABLE2.COLUMN1, TABLE2.COLUMN2,……

**FROM** TABLE1 **LEFT** **OUTER** **JOIN** TABLE2

**ON** TABLE1.COMMON **COLUMN** =TABLE2.COMMON **COLUMN**

E.g.:

**SELECT** EMP.EMPNO, EMP.ENAME,

DEPT.DEPTNO, DEPT.DNAME

**FROM** EMP **LEFT** **OUTER** **JOIN** DEPT

**ON** EMP.DEPTNO =DEPT.DEPTNO

EMPNO ENAME DEPTNO DNAME

*----------- ---------- ----------- ----------*

11 RAGHU 10 SALES

22 RAZ 20 HR

33 AMAR 10 SALES

44 MANI 20 HR

55 CHARN **NULL** **NULL**

66 ANIL **NULL** **NULL**

b. **Right** **Outer** **Join**: It selects **matched** records **from** both the tables **as** well **as** unmatched records **from** **Right** side **table**. **For** doing this operation we have **to** keep a special symbol ‘\*’ **at** the **right** side **of** the equality condition.

Syntax:

**SELECT** TABLE1.COLUMN1, TABLE1.COLUMN2,………,

TABLE2.COLUMN1, TABLE2.COLUMN2,……

**FROM** TABLE1 **RIGHT** **OUTER** **JOIN** TABLE2

**ON** TABLE1.COMMON **COLUMN** = TABLE2.COMMON **COLUMN**

E.g.:

**SELECT** EMP.EMPNO, EMP.ENAME,

DEPT.DEPTNO, DEPT.DNAME

**FROM** EMP **RIGHT** **OUTER** **JOIN** DEPT

**ON** EMP.DEPTNO = DEPT.DEPTNO

EMPNO ENAME DEPTNO DNAME

*----------- ---------- ----------- ----------*

11 RAGHU 10 SALES

22 RAZ 20 HR

33 AMAR 10 SALES

44 MANI 20 HR

**NULL** **NULL** 30 IR

c. **Full** **Outer** **Join**: It **is** just combination **of** **Left** **outer** **Join** + **Right** **outer** **join**. It selects **matched** records **as** well **as** unmatched records **from** the given tables.

Syntax:

**SELECT** TABLE1.COLUMN1, TABLE1.COLUMN2,………,

TABLE2.COLUMN1, TABLE2.COLUMN2,……

**FROM** TABLE1 **FULL** **OUTER** **JOIN** TABLE2

**ON** TABLE1.COMMON **COLUMN** =TABLE2.COMMON **COLUMN**

E.g.:

**SELECT** EMP.EMPNO, EMP.ENAME,

DEPT.DEPTNO, DEPT.DNAME

**FROM** EMP **FULL** **OUTER** **JOIN** DEPT

**ON** EMP.DEPTNO =DEPT.DEPTNO

EMPNO ENAME DEPTNO DNAME

*----------- ---------- ----------- ----------*

11 RAGHU 10 SALES

22 RAZ 20 HR

33 AMAR 10 SALES

44 MANI 20 HR

55 CHARN **NULL** **NULL**

66 ANIL **NULL** **NULL**

**NULL** **NULL** 30 IR

**CROSS**-**JOIN**:

It **is** also known **as** **CROSS** PRODUCT **or** CARTESIAN PRODUCT because it produces the product **of** multiple tables. Every **row** **from** **first** **table** **is** multiplied **with** **all** rows **of** another **table**. Simply it **is** the multiplication **of** two tables.

Syntax:

**SELECT** TABLE1.COLUMN1, TABLE1.COLUMN2,…….

TABLE2.COLUMN1,TABLE2.COLUMN2,…….

**FROM** TABLE1 **CROSS** **JOIN** TABLE2

Ex:

**SELECT** EMP.EMPNO, EMP.ENAME,

DEPT.DEPTNO, DEPT.DNAME

**FROM** EMP **CROSS** **JOIN** DEPT

EMPNO ENAME DEPTNO DNAME

*----------- ---------- ----------- ----------*

11 RAGHU 10 SALES

22 RAZ 10 SALES

33 AMAR 10 SALES

44 MANI 10 SALES

55 CHARN 10 SALES

66 ANIL 10 SALES

11 RAGHU 20 HR

22 RAZ 20 HR

33 AMAR 20 HR

44 MANI 20 HR

55 CHARN 20 HR

66 ANIL 20 HR

11 RAGHU 30 IR

22 RAZ 30 IR

33 AMAR 30 IR

44 MANI 30 IR

55 CHARN 30 IR

66 ANIL 30 IR

**SELF** **JOIN**:

**Join** a **table** **with** itself **by** providing two **table** alias names **is** **called** **SELF**-**JOIN**.

**Select** \* **from** DEPT, DEPT

The above **statement** shows an **error** message because it **not** possible **to** multiply a **table** **by** itself **with** the same name, so that we have **to** project the same **table** DEPT **as** two tables **to** the **SQL** Server. **To** **show** a single DEPT **table** **as** two tables **to** server we have **to** use the concept **of** **table** Alias Names.

**SELECT** \* **FROM** DEPT A, DEPT B

**In** the above example we provided two **table** alias names **for** the single **table** DEPT those are A **and** B. **Then** server identifies that there are two tables available **and** it performs **join** operation **in** a normal way. According **user** point **of** **view** there existed **only** one **table** **but** according **to** Server point **of** **view** there are two tables available those are A **and** B.

DEPTNO DNAME LOC DEPTNO DNAME LOC

*----------- ---------- ---------- ----------- ---------- ---------- ----------------*

10 SALES HYD 10 SALES HYD

20 HR CHE 10 SALES HYD

30 IR BAN 10 SALES HYD

10 SALES HYD 20 HR CHE

20 HR CHE 20 HR CHE

30 IR BAN 20 HR CHE

10 SALES HYD 30 IR BAN

20 HR CHE 30 IR BAN

30 IR BAN 30 IR BAN

9) Odd number of records?

sql> select \* from emp where (rowid,1) in (select rowid, mod(rownum,2) from emp);

10) Even number of records?

sql> select \* from emp where (rowid,0) in (select rowid, mod(rownum,2) from emp);

**Imp\_queries**

How to find schema name and database name in oracle

select sys\_context('userenv','db\_name'), sys\_context('userenv','session\_user') from dual;

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

select trunc(trunc(sysdate,'MM')-1,'MM') "First Day of Last Month",trunc(sysdate,'MM')-1 "Last Day of Last Month" from dual

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

drop table tablename;

we can see drop table in recyclebin by using below query.

SELECT \* FROM RECYCLEBIN;

if we want to get back the drop table we can use below query

FLASHBACK TABLE tablename TO BEFORE DROP;

or

FLASHBACK TABLE emp TO BEFORE DROP RENAME TO emp\_2;

Examples

Drop a table:

SQL> DROP TABLE t1;

Undrop the table:

SQL> FLASHBACK TABLE t1 TO BEFORE DROP;

if we want to delete table from recylebin u can use below query

purge table tablename;

ALTER SYSTEM SET recyclebin = ON;

ALTER SYSTEM SET recyclebin = OFF;

ALTER SESSION SET recyclebin = ON;

ALTER SESSION SET recyclebin = OFF;

Clear recyclebin

To remove all dropped objects from the recyclebin (current user):

PURGE RECYCLEBIN;

To remove all dropped objects from the recyclebin (system wide):

PURGE DBA\_RECYCLEBIN;

Tables can also be droped without sending them to the recyclebin. Example:

DROP TABLE t1 PURGE

Show recyclebin contents

To see the objects in the recyclebin:

SHOW RECYCLEBIN

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

--To know used sapce in particular table

SELECT OWNER,SEGMENT\_NAME,SEGMENT\_TYPE,(BYTES/(1024)/1024) "size in MB" FROM DBA\_SEGMENTS WHERE SEGMENT\_NAME='PRODUCT' AND OWNER='LGDW';

select trunc(trunc(sysdate,'MM')-1,'MM') "First Day of Last Month",trunc(sysdate,'MM')-1 "Last Day of Last Month" from dual

SELECT COLUMN\_NAME,DATA\_TYPE,DATA\_LENGTH,column\_id FROM DBA\_TAB\_COLUMNS

WHERE TABLE\_NAME='CAP\_MBR\_PRICE\_DETAIL\_WRK'

minus

SELECT COLUMN\_NAME,DATA\_TYPE,DATA\_LENGTH,COLUMN\_ID FROM DBA\_TAB\_COLUMNS

WHERE TABLE\_NAME='CAP\_MBR\_PRICE\_DETAIL\_WRKBACKUP';

synonym find:

select \* FROM ALL\_SYNONYMS where owner='INFORMGA' and synonym\_name like 'GADW\_GROUP%'

select \* from dba\_objects where trunc(CREATED)=trunc(sysdate)

select \* from dba\_tables where table\_name like 'PS\_WP\_LOB\_TRNS';

SELECT \* FROM dba\_objects WHERE object\_name = 'WP\_LOB\_TRNS' AND object\_type = 'TABLE';

select \* from dba\_tab\_privs where table\_name='PS\_WP\_LOB\_TRNS';

select \* from dba\_source where upper(text) like '%PS\_WP\_LOB\_TRNS%';

SELECT \* FROM sysadm.PS\_WP\_LOB\_TRNS ;

select \* from dba\_tab\_privs where table\_name='PS\_WP\_LOB\_TRNS';

select \* from dba\_source where upper(text) like '%PS\_WP\_LOB\_TRNS%';

SELECT \* FROM sysadm.PS\_WP\_LOB\_TRNS ;

SELECT COUNT(\*) INT01 FROM SYS.DBA\_DATA\_FILES

Find column in all tables:

select owner, table\_name from all\_tab\_columns where column\_name = 'TRAN\_SQNC\_NBR';

select table\_name from dba\_tab\_columns where column\_name='THE\_COLUMN\_YOU\_LOOK\_FOR';

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

to check table space size

select df.tablespace\_name "Tablespace",

totalusedspace "Used MB",

(df.totalspace - tu.totalusedspace) "Free MB",

df.totalspace "Total MB",

round(100 \* ( (df.totalspace - tu.totalusedspace)/ df.totalspace))

"Pct. Free"

from

(select tablespace\_name,

round(sum(bytes) / 1048576) TotalSpace

from dba\_data\_files

group by tablespace\_name) df,

(select round(sum(bytes)/(1024\*1024)) totalusedspace, tablespace\_name

from dba\_segments

group by tablespace\_name) tu

where df.tablespace\_name = tu.tablespace\_name ;

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

QL query to find long running queries...

select a.\*,sofar/totalwork from v$session\_longops a where time\_remaining <> 0 and totalwork is not null and sid=635;

select b.sql\_text from v$session a ,V$SQLAREA b where 1=1 and a.SQL\_ADDRESS=b.ADDRESS and a.SQL\_HASH\_VALUE =b.HASH\_VALUE and a.sid=502

select b.sql\_text,a.sid from v$session a ,V$SQLAREA b where 1=1 and a.SQL\_ADDRESS=b.ADDRESS and a.SQL\_HASH\_VALUE =b.HASH\_VALUE and upper(b.sql\_text) like '%SELECT%';

to kill long running query

ALTER SYSTEM KILL SESSION 'sid,serial#';

ALTER SYSTEM KILL SESSION '586,31228';

go to toad you will find the running process find the serial no for respective process and kill it.

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

to delete huge number of records

set timing on

set serveroutput on

begin

LOOP

DELETE FROM EDW.MMBR\_CVRG PARTITION (P1012) where rownum<=10000;

exit when sql%rowcount=0;

commit;

end LOOP;

commit;

end;

DELETE FROM EDW.MMBR\_CVRG PARTITION (P1012) where rownum<=10000;

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

SELECT UTL\_INADDR.get\_host\_name('30.130.37.235') FROM dual;

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

Below are the table space details for the Informed schema.

QUERY: select df.tablespace\_name "Tablespace", totalusedspace "Used MB", (df.totalspace - tu.totalusedspace) "Free MB",

df.totalspace "Total MB" from

(select tablespace\_name,

round(sum(bytes) / 1048576) TotalSpace from dba\_data\_files

group by tablespace\_name) df,

(select round(sum(bytes)/(1024\*1024)) totalusedspace, tablespace\_name from dba\_segments

group by tablespace\_name) tu

WHERE DF.TABLESPACE\_NAME = TU.TABLESPACE\_NAME

and df.tablespace\_name='TRANSFORM';

Table space Used MB Free MB Total MB

TRANSFORM 243840 160 244000

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

SELECT 'DELETE FROM '||table\_name FROM TABS WHERE upper(TABLE\_NAME) like '%/\_AN' escape '/';

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

to delete huge number of records

set timing on

set serveroutput on

begin

LOOP

DELETE FROM EDW.MMBR\_CVRG PARTITION (P1012) where rownum<=10000;

exit when sql%rowcount=0;

commit;

end LOOP;

commit;

end;

DELETE FROM EDW.MMBR\_CVRG PARTITION (P1012) where rownum<=10000;

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

to check table space size

select df.tablespace\_name "Tablespace",

totalusedspace "Used MB",

(df.totalspace - tu.totalusedspace) "Free MB",

df.totalspace "Total MB",

round(100 \* ( (df.totalspace - tu.totalusedspace)/ df.totalspace))

"Pct. Free"

from

(select tablespace\_name,

round(sum(bytes) / 1048576) TotalSpace

from dba\_data\_files

group by tablespace\_name) df,

(select round(sum(bytes)/(1024\*1024)) totalusedspace, tablespace\_name

from dba\_segments

group by tablespace\_name) tu

where df.tablespace\_name = tu.tablespace\_name ;

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

select trunc(trunc(sysdate,'MM')-1,'MM') "First Day of Last Month",trunc(sysdate,'MM')-1 "Last Day of Last Month" from dual

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

**Question 1: Oracle version 9.2.0.4.0 what does each number refers to?**

Answer :oracle version number refers

9-Major database release number

 2-Database Maintenance release number  
 0-Application server release number

 4-Component Specific release number  
 0-Platform specific release number

Link: <http://dwhlaureate.blogspot.in/2012/08/joins-in-oracle.html>

**JOINS IN ORACLE-different joins in oracle with examples**

1. The purpose of a join is to combine the data across tables.

2. A join is actually performed by the where clause which combines the specified rows of tables.

3. If a join involves in more than two tables then oracle joins first two tables based on the joins condition and then compares the result with the next table and so on.

**TYPES**

1     Equi join

2     Non-equi join

3     Self join

4     Natural join

5     Cross join

6     Outer join

* Left outer
* Right outer
* Full outer

7     Inner join

8     Using clause  
9     On clause

Assume that we have the following tables.

SQL> select \* from dept;

|  |  |  |
| --- | --- | --- |
| **DEPTNO** | **DNAME** | **LOC** |
| 10 | INVENTORY | HYBD |
| 20 | FINANCE | BGLR |
| 30 | HR | MUMBAI |

SQL> select \* from emp;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **MGR** | **DEPTNO** |
| 111 | saketh | analyst | 444 | 10 |
| 222 | sudha | clerk | 333 | 20 |
| 333 | jagan | manager | 111 | 10 |
| 444 | madhu | engineer | 222 | 40 |

**1.**      **EQUI JOIN**

A join which contains an equal to ‘=’ operator in the joins condition.

*Ex:*

*SQL>*select empno,ename,job,dname,loc from emp e,dept d where e.deptno=d.deptno;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |

Using clause

SQL> select empno,ename,job ,dname,loc from emp e join dept d using(deptno);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |

On clause

SQL> select empno,ename,job,dname,loc from emp e join dept d on(e.deptno=d.deptno);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |

**2.**      **NON-EQUI JOIN**

 A join which contains an operator other than equal to ‘=’ in the joins condition.

 Ex:

SQL> select empno,ename,job,dname,loc from emp e,dept d where e.deptno > d.deptno;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 222 | sudha | clerk | INVENTORY | HYBD |
| 444 | madhu | engineer | INVENTORY | HYBD |
| 444 | madhu | engineer | FINANCE | BGLR |
| 444 | madhu | engineer | HR | MUMBAI |

**3.**      **SELF JOIN**

Joining the table itself is called self join.

Ex:

SQL> select e1.empno,e2.ename,e1.job,e2.deptno from emp e1,emp e2 where e1.empno=e2.mgr;

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DEPTNO** |
| 111 | jagan | analyst | 10 |
| 222 | madhu | clerk | 40 |
| 333 | sudha | manager | 20 |
| 444 | saketh | engineer | 10 |

**4.**      **NATURAL JOIN**

Natural join compares all the common columns.

Ex:

SQL> select empno,ename,job,dname,loc from emp natural join dept;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |

**5.**      **CROSS JOIN**

This will gives the cross product.

Ex:

SQL> select empno,ename,job,dname,loc from emp cross join dept;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 222 | sudha | clerk | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 444 | madhu | engineer | INVENTORY | HYBD |
| 111 | saketh | analyst | FINANCE | BGLR |
| 222 | sudha | clerk | FINANCE | BGLR |
| 333 | jagan | manager | FINANCE | BGLR |
| 444 | madhu | engineer | FINANCE | BGLR |
| 111 | saketh | analyst | HR | MUMBAI |
| 222 | sudha | clerk | HR | MUMBAI |
| 333 | jagan | manager | HR | MUMBAI |
| 444 | madhu | engineer | HR | MUMBAI |

**6.**      **OUTER JOIN**

Outer join gives the non-matching records along with matching records.

**LEFT OUTER JOIN**

This will display the all matching records and the records which are in left hand side table those that are not in right hand side table.

Ex:

SQL> select empno,ename,job,dname,loc from emp e left outer join dept d

on(e.deptno=d.deptno);

Or

SQL> select empno,ename,job,dname,loc from emp e,dept d where

e.deptno=d.deptno(+);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |
| 444 | madhu | engineer |  |  |

**RIGHT OUTER JOIN**

This will display the all matching records and the records which are in right hand side table those that are not in left hand side table.

Ex:

SQL> select empno,ename,job,dname,loc from emp e right outer join dept d

on(e.deptno=d.deptno);

Or

SQL> select empno,ename,job,dname,loc from emp e,dept d where e.deptno(+) =

d.deptno;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |
|  |  |  | HR | MUMBAI |

**FULL OUTER JOIN**

This will display the all matching records and the non-matching records from both tables.

Ex:

SQL> select empno,ename,job,dname,loc from emp e full outer join dept d

on(e.deptno=d.deptno);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 333 | jagan | manager | INVENTORY | HYBD |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |
| 444 | madhu | engineer |  |  |
|  |  |  | HR | MUMBAI |

**7.**      **INNER JOIN**

This will display all the records that have matched.

Ex:

SQL> select empno,ename,job,dname,loc from emp inner join dept using(deptno);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerkx` | FINANCE | BGLR |

**What is index and its use?How to create unique index?And Types of Indexes**

**INDEXES:**

* Just like a book index helps us in finding the chapter and page number, an oracle index speeds up access time to the **rows.**
* Indexes **are optional structures** associated with tables.
* An index is also a schema object.
* We can drop an index without dropping the table it indexes.
* An index can be created explicitly or automatically.
* When we drop a table the corresponding indexes of the table are dropped.
* **A unique index gets created when we create a unique key or primary key in a table definition.**
* The name of the index is the name of the constraint.
* Indexes can be unique or non-unique.  **Unique indexes guarantee that no two rows of a table have duplicate values in the columns that define the index.  Non-unique indexes do not impose this restriction on the column values.**
* The presences of more number of indexes on a table decreases the performance of DML statements, because Oracle must make changes to the indexes associated with the table

**Syntax:**

**create index <index\_name> on <table\_name> (column\_names\_separated\_by\_comma);**

An index on a table can be created for one column or more than one column [composite index]

Examples:

**create index my\_index on emp(empno); [default NONUNIQUE]**

Index created.

SQL> desc user\_indexes               [**use this dd VIEW**]

**How to create unique index:**

**create unique index my\_indx on emp(ename);**

Index created.

**select index\_name,index\_type from user\_indexes where table\_name='EMP';**

**INDEX\_NAME                     INDEX\_TYPE**

**------------------------------ -----------**

**MY\_INDEX                       NORMAL**

**MY\_INDX                        NORMAL**

**drop index my\_index;**

Index dropped.

**drop index my\_indx;**

Index dropped.

**create index my\_indx on emp(empno,ename);**

Index created.

***Index created with above statement will be used to retrieve data, when the where clause has either***

         ***Both empno,ename or***

         ***empno alone.***

***But, not when ename alone is used.***

So it creates the index based on the first column.

**Types of Indexes:**

1.**Reverse key indexes**

**2.** **Bitmap Indexes**

**3.** **Index-Organized Tables**

**Bitmap index**: When we have low cardinality columns [columns in which number of bitmap values is small compared to the number of rows].  Ex: yes and no.  Here we can use the bit map index.  For example:

If the values in a column are repeated more than a hundred times is a candidate for bitmap index.  On a table with one million rows, a column with 10,000 distinct values is a candidate for a bitmap index.

**create bitmap index <name\_of\_index> on <table\_name>(column\_name);**

**Reverse Key Indexes**:  Creating a reverse key index, when compared to a standard index, reverses each byte of the column being indexed while keeping the column order.  ***This can be used in some situations where there can be performance degradation.***  Using the reverse key index can avoid such performance degradations.

**Index-Organized Tables**:  An index-organized table differs from a regular table in that the data for the table is held in its associated index.  Changes to the table data, such as adding new rows, updating rows, or deleting rows, result only in updating the index.  The index-organized table is like a regular table with an index on one or more of its columns**.**  But instead of maintaining two separate storages for the table and the index, the database system only maintains a single index, which contains both the encoded key value and the associated column values for the corresponding row.  Instead of having the row ROWID as the second element of the index entry, the actual data of the row is stored in the index.

We can create a index organized table using create table command with ORGANIZATION INDEX clause.

**create table myemp\_index(empno number(5) primary key,ename varchar2(12)) ORGANIZATION INDEX;**

Table created.

The primary key is a must for creating index-organized table.

**INSERT INTO MYEMP\_INDEX(SELECT EMPNO,ENAME FROM EMP);**

14 rows created.

**SELECT \* FROM MYEMP\_INDEX;**

    EMPNO ENAME

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

     7369 SMITH

     7499 ALLEN

     7521 WARD

     7566 JONES

     7654 MARTIN

     7698 BLAKE

     7782 CLARK

     7788 SCOTT

     7839 KING

     7844 TURNER

     7876 ADAMS

     7900 JAMES

     7902 FORD

     7934 MILLER

14 rows selected.

**Correlated Sub Queries**

Correlated sub queries are used **for row-by-row processing.**

Here each sub query is executed once for every row of the outer query.The oracle server performs a correlated subquery when the subquery references a column from a table referred to in the parent statement.

**A correlated sub query is evaluated once for each row processed by the parent statement (query)**

The parent statement can be a SELECT, UPDATE or DELETE statement.

The general for of a correlated sub query is:

**select column1, column2, . . . . . . .**

**from table1 outer**

**where [column1] *operator* (select column1, column2,**

**from table2 where expr1= outer.expr2);**

The sub query references a column from a table in the parent query, using the Exists operator:  a Boolean operator.

The *exists*operator tests for existence of rows in the results of the sub query.

* **If a sub query value is found:**

1. The search does not continue in the inner query
2. The condition is flagged true.

* **If a sub query row value is not found:**

1. The condition is flagged false.
2. The search continues in the inner query.

While nesting select statement, all logical operators are valid.  In addition, we can use the *exists* operator.  This operator is frequently used with correlated sub queries to test whether a value retrieved by the outer query exists in the results set of the values retrieved by the inner query.

Example:

To find all the employees who have at least one person reporting to him, we go as follows

**select empno, ename,job,deptno from emp outer**

**where exists(select 'x' from emp where mgr=outer.empno);**

    EMPNO ENAME      JOB          DEPTNO

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

     7566 JONES      MANAGER          20

     7698 BLAKE      MANAGER          30

     7782 CLARK      MANAGER          10

     7788 SCOTT      ANALYST          20

     7839 KING       PRESIDENT        10

     7902 FORD       ANALYST          20

6 rows selected.

**Using the Not Exists operator:**

To find all the departments those do not any employees.

**select deptno, dname,loc from dept d**

**where not exists(select 'x' from emp where deptno=d.deptno);**

   DEPTNO DNAME          LOC

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

       40 Operations     BOSTON

The correlated sub query can be an update or a delete also.

**What is an Inline View?**

**INLINE VIEW**

* It is not a schema object like a normal view.
* It is sub query with a name (alias) placed in the from clause of another select statement (main query) for which it (the sub query) acts as a data source.
* The outer query will have a reference of the inline view.
* The inline view can have a GROUP BY clause, order by clause or even inline view itself can be join.
* Inline views are useful for performing the Top-N (Top 3 sales reps or top 10 students etc) analysis.

See AN EXAMPLE OF INLINE VIEW, WHICH HAS THE GROUP BY CLAUSE. The query finds the employees in the emp table whose salary is less than the maximum salary of their department.

**SQL> SELECT ENAME,SAL,E1.DEPTNO,E2.MAXSAL FROM EMP E1,**

**(SELECT DEPTNO,MAX(SAL) MAXSAL FROM EMP GROUP BY DEPTNO)E2**

**WHERE E1.DEPTNO=E2.DEPTNO AND E1.SAL<E2.MAXSAL;**

ENAME             SAL     DEPTNO     MAXSAL

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

CLARK            2450         10       5000

MILLER          1300         10       5000

SMITH            800           20       3000

ADAMS           1100        20       3000

JONES            2975         20       3000

ALLEN            1600        30       2850

MARTIN         1250        30       2850

JAMES             950          30       2850

TURNER         1500         30       2850

WARD             1250         30       2850

10 rows selected.

**SELECT ENAME,SAL,E1.DEPTNO,E2.MAXSAL FROM EMP E1,**

**(SELECT DEPTNO,MAX(SAL) MAXSAL FROM EMP GROUP BY DEPTNO order by deptno)E2 WHERE E1.DEPTNO=E2.DEPTNO AND E1.SAL<E2.MAXSAL;**

ENAME             SAL     DEPTNO     MAXSAL

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

CLARK            2450         10       5000

MILLER          1300         10       5000

SMITH            800           20       3000

ADAMS          1100         20       3000

JONES            2975         20       3000

ALLEN           1600         30       2850

MARTIN        1250         30       2850

JAMES           950            30       2850

TURNER       1500         30       2850

WARD           1250         30       2850

10 rows selected.

**Nested Sub query VS Correlated Sub query**

With a normal nested sub query, the inner SELECT query runs **first** and executes **once**, returning values to be used by the main query.  A correlated sub query however, **executes once for each candidate row considered by the outer query**.  **In other words the inner query is driven by the outer query.**

**Steps of execution:**

 Nested sub query execution:

* The inner query is executed first and finds a value
* The outer query executes once, using the value from the inner query.

Correlated Sub query execution:

* Get the candidate row (fetched by the outer query).
* Execute the inner query using the value of the candidate row.
* Use the values resulting from the inner query to test “qualify or disqualify” the candidate.
* Repeat until no candidate row remains tested.

The general format of a correlated sub query is:

select column1, column2, . . . . . . .

from table1 outer

where [column1] *operator* (select column1, column2,

from table2 where expr1= outer.expr2);