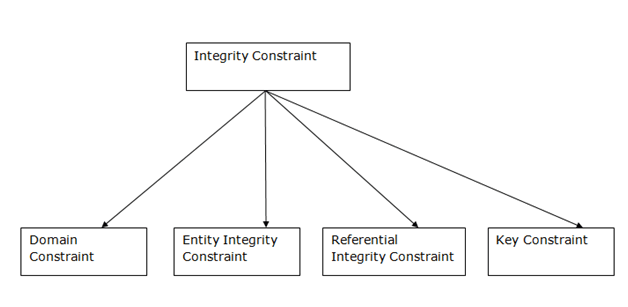
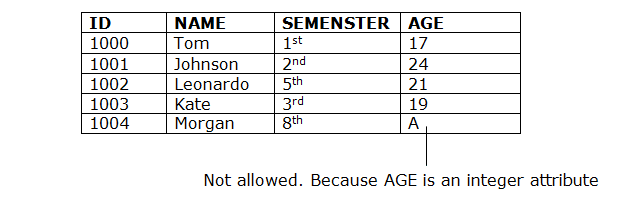
**Integrity Constraints**

* Integrity constraints are a set of rules used to maintain the accuracy and consistency of data in the database.
* Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
* Thus, integrity constraint is used to guard against accidental damage to the database.

**Types of Integrity Constraints **

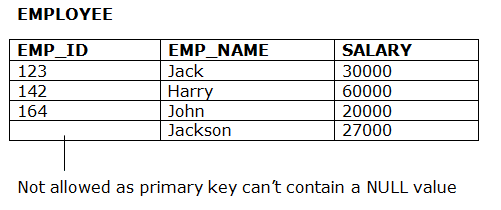
**1. Domain constraints**

A domain integrity constraint is a set of rules on the values that can be stored in a column.

**Example:   
2. Entity integrity constraints**

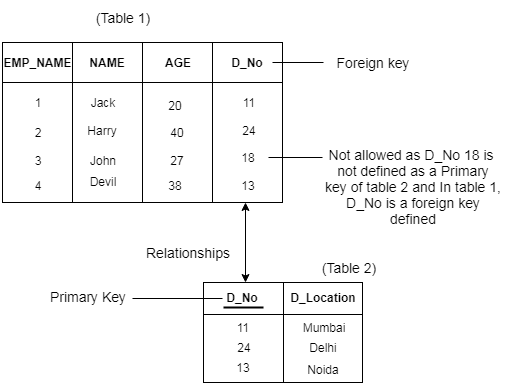
* The entity integrity constraint states that primary key value can't be null.
* This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.

**Example:**

****

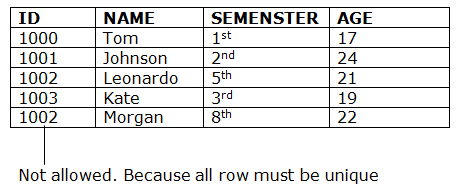
**3. Referential Integrity Constraints**

* A referential integrity constraint is specified between two tables.
* In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

**Example: **

**4. Key constraints**

* Keys are the entity set that is used to identify an entity within its entity set uniquely.
* An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and not null value in the relational table.

**Example:  
 **

Constraints can be defined in two ways.

* The constraints can be specified immediately after the column definition. This is called column-level definition.
* The constraints can be specified after all the columns are defined. This is called table-level definition.

**1. Primary Key Constraints**

* A primary key constraint (also known as a "primary key") is a type of key constraint that requires every value in a given column to be unique.
* The null value is not allowed in the primary key column(s).

**Syntax to define a Primary key at column level:**

Column name datatype [CONSTRAINT constraint\_name]  **PRIMARY KEY**

Here,[CONSTRAINT constraint\_name] is optional.

**Example**

CREATE TABLE employee ( id number(5) PRIMARY KEY,name char(20),dept char(10), age number(2), salary number(10),location char(10) );

**Syntax to define a Primary key at table level:** [CONSTRAINT constraint\_name] PRIMARY KEY (column\_name1,column\_name2,..)

column\_name1, column\_name2 are the names of the columns which define the primary Key.

**Example**

CREATE TABLE employee ( id number(5), name char(20), deptno number(4),deptname char(10),age number(2), salary number(10),location char(10),PRIMARY KEY (id,deptno));

**2. Unique Key Constraints**

* A unique key constraint is a column or set of columns that ensures that the values stored in the column are unique.
* A table can have more than one unique key and it can contain NULL values.

**Syntax to define a Unique key at column level:**

column name[CONSTRAINT constraint\_name] UNIQUE

**Example**

CREATE TABLE employee( id number(5) PRIMARY KEY,name char(20),dept char(10), age number(2),salary number(10),location char(10) UNIQUE );

**Syntax to define a Unique key at table level:**

[CONSTRAINT constraint\_name] UNIQUE(column\_name)

**Example**

CREATE TABLE employee ( id number(5) PRIMARY KEY,name char(20),dept char(10),age number(2),salary number(10),location char(10),UNIQUE(location));

**3. Foreign Key Constraints**

* A foreign key constraint defines a relationship between two tables. A foreign key in one table references a primary key in another table.
* Foreign keys prevent invalid data from being inserted into the foreign key column.

**Syntax to define a Foreign key at column level:**

[CONSTRAINT constraint\_name] REFERENCES Referenced\_Table\_name(column\_name)

Example

Lets use the "product" table and "order\_items"

CREATE TABLE product ( product\_id number(5) PRIMARY KEY, product\_name char(20),supplier\_name char(20),unit\_price number(10));

CREATE TABLE order\_items ( order\_id number(5) PRIMARY KEY, product\_id number(5) REFERENCES product(product\_id),product\_name char(20), supplier\_name char(20),unit\_price number(10));

**Syntax to define a Foreign key at table level:** [CONSTRAINT constraint\_name] FOREIGN KEY(column\_name) REFERENCES referenced\_table\_name(column\_name);

**Example**

CREATE TABLE product ( product\_id number(5), product\_name char(20),supplier\_name char(20),unit\_price number(10) ) PRIMARY KEY(product\_id));

CREATE TABLE order\_items( order\_id number(5) ,product\_id number(5), product\_name char(20),supplier\_name char(20),unit\_price number(10)PRIMARY KEY(order\_id), FOREIGN KEY(product\_id) REFERENCES product(product\_id));

**4. NOT NULL Constraints**

A NOT NULL constraint is used to ensure that no row can be inserted into the table without a value being specified for the column(s) with this type of constraint. Which means a null value is not allowed.

**Syntax to define a Not Null constraint at column level**:

Column name[CONSTRAINT constraint name] NOT NULL

Example:

CREATE TABLE employee( id number(5),name char(20)NOT NULL, dept char(10),age number(2),salary number(10),location char(10) );

**Syntax to define a Not Null constraint at table level:**

[CONSTRAINT constraint name] NOT NULL(Column name);

Example:

CREATE TABLE employee( id number(5),name char(20), dept char(10),age number(2),salary number(10),location char(10) NOT NULL(name));

**5. Check Constraints**

This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

**Syntax to define a Check constraint at column level:**

Column name[CONSTRAINT constraint\_name] CHECK (condition);

**Example**

CREATE TABLE employee ( id number(5) PRIMARY KEY, name char(20), dept char(10), age number(2), gender char(1) CHECK (gender in ('M','F')), salary number(10), location char(10));

**Check Constraint at table level:**

[CONSTRAINT constraint\_name] CHECK column name(condition);

**Example**

CREATE TABLE employee ( id number(5), name char(20), dept char(10), age number(2), gender char(1), salary number(10), location char(10),PRIMARY KEY(id),CHECK (gender in ('M','F')));

**SEQUENCE**

* A Sequence is a user created Database Object that can automatically generate **unique sequential integers.**
* It is mainly used to create primary key values.
* It can be used in columns of any table.
* It can be created even if there is no table.
* Ex:- generating A/C no, application no, transaction no etc..

**Syntax:**

**Create sequence <sequence name>**

**[start with <n>]**

**[increment by <m>]**

**[minvalue n]**

**[maxvalue n]**

**[Cycle/nocycle]**

**[Cache/nocache];**

**SQL> create sequence s1;**

Sequence created.

SQL> select **s1.nextval** from dual;

NEXTVAL

----------

1

SQL> select **s1.currval** from dual;

CURRVAL

----------

1

SQL> select s1.nextval from dual;

NEXTVAL

----------

2

SQL> select\* from emp1;

ENO ENAME ADDRESS

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

ram delhi

varun chennai

ravi banglore

amir delhi

**SQL> update emp1 set eno = s1.nextval;**

4 rows updated.

SQL> select\* from emp1;

ENO ENAME ADDRESS

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

1 ram delhi

2 varun chennai

3 ravi banglore

4 amir delhi

SQL>insert into emp1 values(s1.nextval , ’babu’ , ‘mumbai ’);

**Create sequence s2**

**start with 10**

**increment by 10**

**minvalue 10**

**maxvalue 30**

**cycle**

**nocache;**

Sequence created.

**SQL> update emp1 set eno=s2.nextval;**

4 rows updated.

SQL> select\* from emp1;

ENO ENAME ADDRESS

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

10 ram delhi

20 varun chennai

30 ravi banglore

10 amir delhi

**SQL> drop sequence s1;**

Sequence dropped.

**PSEUDO COLUMNS**

Pseudo columns are Virtual columns, that can be called only from SQL statement.

It behaves like a table column, but is not actually stored in the table.

We can select from Pseudo columns, but we cannot insert, update, or delete their values.

**1. CURRVAL -** Gives the most recently generated sequence value.

SQL> select **s1.currval** from dual;

**2. NEXTVAL -** Gives the next value in the sequence to generate.

SQL> select **s1.nextval** from dual;

**3. ROWID –** It contains physical address of each record which is stored in the table. It is a hexadecimal number(18 character string)

SQL> select rowid from emp1;

ROWID

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

AAAFNzAABAAALIxAAA

AAAFNzAABAAALIxAAB

AAAFNzAABAAALIxAAC

AAAFNzAABAAALIxAAD

**4. ROWNUM-**

It will return the row number for the result executed by the query.It is primarily used in WHERE CLAUSE and SET CLAUSE.

The first row has ROWNUM1,second has ROWNUM2 and so on.

SQL> select \* from emp1 where rownum<=3;

ENO ENAME ADDRESS

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

10 ram delhi

20 varun chennai

30 ravi banglore

SQL> select rownum from emp1;

ROWNUM

----------

1

2

3

4

**SYNONYM**

* Synonym is a Database Object.
* It creates permanent alias name(alternate name) for the Table ,View - for security & convenience.
* Before creating synonym need to get privilege from DBA.
* Two types of synonym 1) Private Synonym 2) Public Synonym. By default the synonym is Private –accessed only by the creator of the synonym (DBA).
* Public Synonym can be accessed by the authenticated users.

**SQL> create synonym inv for inventory;** ---- Private

**SQL> create public synonym eb for ebbill\_details;** --- public

ERROR at line 1:

ORA-01031: insufficient privileges

**SQL>grant create synonym to shakkina;**

Grant succeeded.

**SQL> create synonym eb for ebbill\_details;**

Synonym created.

**SQL>select\* from eb;**

CID NAME PREV\_READ CURR\_READ UNIT\_CONSUMED TOT\_AMOUNT BILL\_STATU

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

c01 hami 300 400 100 200 paid

c02 sree 100 300 200 600 not paid

c03 rohit 400 600 200 600 paid

co3 rani 200 500 300 1200 not paid

c04 raja 300 700 400 2000 paid

**Drop Synonym synonymname;**

**INDEX**

* Index is a Database Object. It is created on column(s) of a table.
* Index has two parts 1) Rowid 2)Data . By default Data is organized in ascending order. Index uses Rowid to fetch the data from table.
* Implicit Index is created when a column is declared as Primary key. Explicit Index is created using “Create Index…” syntax.
* Advantage of Index is Decreasing search time for the required data – Improve the performance-so that getting the result very fast.

**Syntax:**

**Create Index indexname on tablename(col1,col2…..);**

To Alter Index

**Alter Index indexname Rebuild;**

To drop an Index

**Drop Index indexname;**

**Example: create index idx\_sal on emp1(salary);**

**Index created.**

select salary from emp1 where salary<7000;

**Salary idx \_sal**

1. 9000 7 1000
2. 7000 4 1500
3. 10000 5 3000
4. 1500 2 7000
5. 3000 8 7000
6. 9000 1 9000
7. 1000 6 9000
8. 7000 3 10000
9. 10000 9 10000

**CLUSTER**

A Cluster is a Database Object that stores data related to two or more tables in a single disk space(single memory location called blocks).

As Indexes, Cluster also improves performance for faster retrieval of data from the table.

Only new tables can be created in a cluster. Existing tables cannot be moved into cluster.

To Create Cluster the following three steps need to be followed.

1. **Create Cluster**
2. **Create Index on Cluster**
3. **Create Tables**

1. Creating Cluster:-

**Create Cluster c1(d number(2));**

Cluster created.

2. Creating Index:-

**Create Index ic1 on cluster c1;**i

Index Created.

3. Creating Tables:-

**Create table dept10(dno number(2),dname varchar2(20)) cluster c1(dno);**

Table created.

This table is stored in cluster c1 organized based on deptno wise.

**Create table emp10(dno number(2), empno number(2),empname varchar2(20)) cluster c1(dno);**

Table created.

This table is also stored in cluster c1 organized based on deptno wise. Here deptno is a cluster key based on which join operation can be performed.

SQL> select rowid, dno, empno, empname from emp10;

ROWID DNO EMPNO EMPNAME

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

AAAFPQAABAAALJBAAA 10 34 dhinesh

AAAFPQAABAAALJCAAA 20 35 suresh

SQL> select rowid, dno, dname from dept10;

ROWID DNO DNAME

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

AAAFPQAABAAALJBAAA 10 hr

AAAFPQAABAAALJCAAA 20 sales

**ROLES**

* **Roles** are collection of privileges or access rights.
* Privileges - access rights provided to a user on a database object.
* When there are many users in a database it becomes difficult to grant or revoke privileges to users. Therefore if we define Roles, it is easier to grant or revoke privileges through a Role rather assigning directly to every user.
* We can create Roles or use the System Roles pre-defined by Oracle.

**System Role:**

1. CONNECT- Create Table, View, Synonym, Sequence, Session, Cluster, and Database Link.
2. RESOURCE- Create Table, Index, Procedure, Sequence,

Cluster, Trigger.

1. DBA- All System Privileges.

GRANT Resource, Connect to Username;

**CREATING ROLES:**

**Syntax**

Create ROLE Role\_name [identified by password];

**Example**

Grant CREATE TABLE, CREATE VIEW Privilege to a User by creating a Role Developer.

Step 1: Creating Role

CREATE ROLE Developer;

Step 2: Adding Privileges to Role

GRANT CREATE TABLE, CREATE VIEW TO Developer;

Step 3: Grant the Role to User

GRANT Developer to Dhinesh, James;

**Revoke the Privilege from the Role**

REVOKE CREATE TABLE, CREATE VIEW From Developer;

**Syntax to Drop Role**

Drop Role Role\_name;

Ex:

Drop Role Developer;

**String Functions:**

1. SQL> select upper ('bsc') from dual;

**Output**

UPP

------

BSC

1. SQL> select lower ('BSC') from dual;

**Output**

LOW

---

bsc

1. SQL> select initcap ('bsc') from dual;

**Output**

INI

---

Bsc

1. SQL> select concat('joseph','college') from dual;

**Output**

CONCAT('JOSEP

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

josephcollege

1. SQL> select length('josephs') from dual;

**Output**

LENGTH('JOSEPHS')

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

7

1. SQL> select substr('st.josephs college',1,10) from dual;

**Output**

SUBSTR('ST

----------

st.josephs

1. SQL> select Instr('st.josephs college','o',1) from dual;

**Output**

INSTR('ST.JOSEPHSCOLLEGE','O',1)

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

5

1. SQL> select Trim('h' from 'helloworld') from dual;

**Output**

TRIM('H')

---------

elloworld

1. SQL> select ltrim('helloworld','he') from dual;

**Output**

LTRIM('He’)

---------

lloworld

1. SQL> select rtrim('helloworld','d') from dual;

**Output**

RTRIM('d’)

---------

helloworl

1. SQL> select lpad('50000',10,'\*') from dual;

**Output**

LPAD('5000

----------

\*\*\*\*\*50000

1. SQL> select rpad('50000',10,'\*') from dual;

**Output**

RPAD('5000

----------

50000\*\*\*\*\*

**Number Functions**

1. SQL> select round(45.926,2) from dual;

**Output**

ROUND(45.926,2)

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

45.93

2. SQL> select trunc(45.926,2) from dual;

**Output**

TRUNC(45.926,2)

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



3. SQL> select mod(1000,300) from dual;

**Output**

MOD(1000,300)

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

100

1. SQL> select power(3,2) from dual;

**Output**

POWER(3,2)

----------

9

1. SQL> select sqrt(64) from dual;

**Output**

SQRT(64)

----------

2. SQL> select abs(-43) from dual;

**Output**

ABS(-43)

----------

43

1. SQL> select ceil(-2.3) from dual;

**Output**

CEIL(-2.3)

----------

-2

8. SQL> select floor(-2.3) from dual;

**Output**

FLOOR(-2.3)

-----------

-3

9. SQL> select cos(30) from dual;

**Output**

**COS(30)**

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

0.15425144988758405

10. SQL> select sin(0) from dual;

**Output**

**SIN(0)**

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

0

**Date Functions**

1. SQL> select months\_between('12-sep-12','05-aug-10')from dual;

**Output**

MONTHS\_BETWEEN('12-SEP-12','05-AUG-10')

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

25.2258065

1. SQL> select add\_months('10-jan-12',2) from dual;

**Output**

ADD\_MONTH

---------

10-MAR-12

1. SQL> select next\_day('15-sep-12','monday') from dual;

**Output**

NEXT\_DAY(

---------

17-SEP-12

1. SQL> select last\_day('15-sep-12') from dual;

**Output**

LAST\_DAY(

---------

30-SEP-12

1. SQL> select round(sysdate,'month') from dual;

**Output**

ROUND(SYS

---------

01-OCT-12

1. SQL> select trunc(sysdate,'month') from dual;

**Output**

TRUNC(SYS

---------

01-SEP-12

1. SQL> select to\_char(sysdate,'yyyy') from dual;

**Output**

TO\_C

----

2012

1. SQL> select to\_date('18/09/2012','dd/mm/yyyy')from dual;

**Output**

TO\_DATE('

---------

18-SEP-12

1. SQL> select to\_number('2012') from dual;

**Output**

TO\_NUMBER('2012')

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

2012