Unit – III
Database Integrity Constraints & Objects

constraints:

- A Constraint is a rule that restricts the values that may be present in the database
- As oracle is based on a Relational data model, Constraints provided by Oracle follows the general Constraints provided by relational model.
- These Constraints can be broadly classified into two categories.

1. Entity Integrity Constraints
2. Referential Integrity Constraints

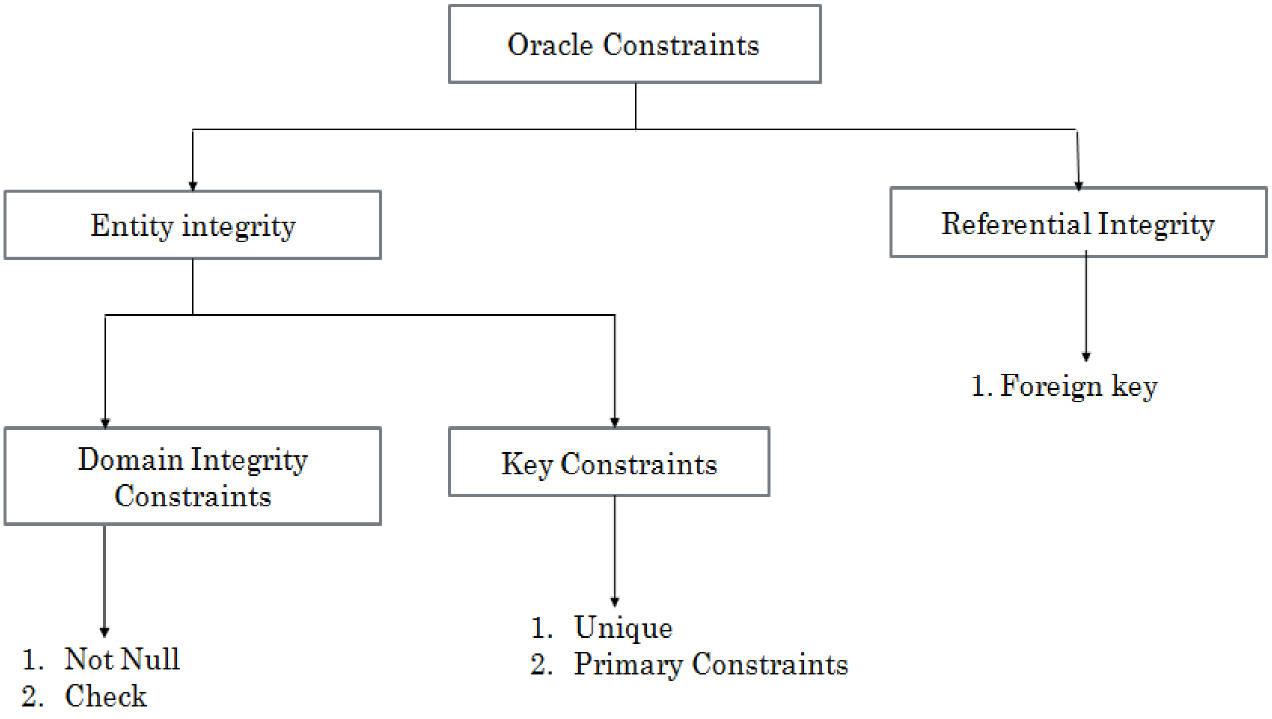
1. **Entity Integrity Constraints :**

- Entity integrity constraints are constraints that restrict the values in row of an individual table.
- This constraints also divided into two sub categories:

(I) Domain Integrity Constraints
(II) Key Constraints

2. Referential Integrity Constraints :

- Referential integrity Constraints specifies that a row of one table that refers to another row must refer to an existing row in that table.



3.1 Domain Integrity constraints: Not null, Check

❖ NOT NULL Constraints

- There may be Records in table that do not contain any value for some fields. In Oracle, Null values are stored in such fields. In other words, a NULL value represents an empty field.

- **CHARACTERSTICS**:
  - A NULL value indicates ‘not applicable’, ‘missing’, or ‘not known’.
  - A NULL value distinct from zero or other numeric value for numerical data.
  - A NULL value is also distinct from blank space for character data.
  - A NULL value will evaluate to null in any expression.
  - The result of any condition including null value is unknown, and treated as a FALSE.

ENFORCED RESTRICTION:

- A column, defined as a NOT NULL, cannot have a NULL value. In other words, such column becomes a mandatory column and cannot be left empty for any record.
- A **NOT NULL** CONSTRAINT DEFINED AT COLUMN LEVEL only

Syntax :

ColumnName datatype (size) NOT NULL

Example:

```
CREATE TABLE EMPLOYEES
(
EMPID INTEGER NOT NULL,
EName VARCHAR2(10) NOT NULL,
DOJ  DATE
);
```

❖ CHECK constraints

- “The CHECK constraint is used to implement business rules. This constraint is also referred as business rule constraints. Business rules may vary from system to system.”

ENFORCED RESTRICTION :

- The CHECK constraint is bound to a particular column.
- Once a CHECK constraint is implemented, any insert or update operation on that table must follow this constraints.
- If any operation violates condition, it will be rejected.

- ❖ A CHECK CONSTRAINTS DEFINED AT **COLUMN LEVEL** :

Syntax:

```
columnName datatype (size) CHECK ( CONDITION )
```

- ❖ A CHECK CONSTRAINTS DEFINED AT **TABLE LEVEL** :

Syntax:

```
CHECK ( CONDITION )
```

Example:

```
CREATE TABLE Orders
 (
  order_id number(10),
  amount number(10) CHECK (amount > 0)
);
```

3.2 Entity Integrity constraints: Unique, Primary key.

❖ Unique Constraints

- A column must have unique values. This is required to identify all records stored in table uniqly,

ENFORCED RESTRICTION:

- A column, defined as a UNIQUE, cannot have duplicate values across all records. In other words, such column must contain unique values.

- ❖ A **UNIQUE CONSTRAINTS** DEFINED AT **COLUMN LEVEL** :

Syntax:

```
columnName datatype (size) UNIQUE
```

- ❖ A **UNIQUE CONSTRAINTS** DEFINED AT **TABLE LEVEL** :

Syntax:

```
UNIQUE ( columnName [, columnName …] )
```

Example:

```
CREATE TABLE Colleges (
  college_id number(5) UNIQUE,
  college_code VARCHAR(20) UNIQUE,
  college_name VARCHAR(50)
);
```

❖ Primary Key Constraints

- "A primary key is a set of one or more columns used to identify each record uniquely in a column". A single column primary key is called as simple key, while a multi-column primary key is called a composite key

ENFORCED RESTRICTION:

- A column defined as a primary key, cannot have duplicate values across all records and cannot have NULL values.

❖ A PRIMARY KEY CONSTRAINTS DEFINED AT **COLUMN LEVEL** :

Syntax:

```
columnName datatype (size) PRIMARY KEY
```

❖ A PRIMARY KEY CONSTRAINTS DEFINED AT **TABLE LEVEL** :

Syntax:

```
PRIMARY KEY ( columnName [, columnName …] )
```

3.3 Referential Integrity constraints: Foreign key, referenced key, on delete cascade

❖ Foreign Key Constraints

- "A Foreign key is a set of one or more columns whose values are derived from the primary key or unique key of other table."
- The table, in which a foreign key is defined, is called a **foreign table, detail table or child table.**
- The table in which primary key or unique key is referred, is called a **primary table, master table or parent table.**

ENFORCED RESTRICTION:

- The foreign key constraints enforce different restriction on detail table and master table.
- If bname is defined as a foreign key in Account table referring to bname in branch table, then, there will be following restriction on both of these tables.

1. **Restriction on detail table:**
   - Detail table contains a Foreign key. And, it is related to master table.
   - Insert or update operation involving value of Foreign key are not allowed, if corresponding value does not exist in the master table.

2. **Restriction on master table:**
   - Master table contains a primary key or unique key, which is referred by Foreign key in detail.

- Delete or update operation on records in master table are not allowed, if corresponding records are present in detail table

❖ A FOREIGN KRY CONSTRAINTS DEFINED AT **COLUMN LEVEL:**

syntax :

```
columnName  datatype    ( size )
        REFRENCES tablename ( columnName )
        [ON DELETE CASCADE]
```

❖ A FOREIGN KRY CONSTRAINTS DEFINED AT **TABLE LEVEL**:

syntax :

```
FOREIGN KEY ( columnName [, columnName …] )
REFERENCES tablename (columnName [, columnName …] )
```

Example:

```
CREATE TABLE Orders
 (
   OrderID number(10),
   OrderNumber number(10),
   PersonID number(10),
   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

3.4 Views – Create, Alter, Drop views

- A **view** is a virtual or logical table that allows viewing or manipulating the parts of the tables.
- A view is derived from one or more tables known as **base tables**.
- A view looks like and works similarly to normal tables. But, unlike tables, a view does not have **storage space** to store data.
- A view is created by a query, i.e. a **SELECT** statement which uses base tables.
- Data for views are extracted from these base tables based on specified query.
- A view is **dynamic** and always **reflects the current data of the base tables**.
- Only definition of view is stored in the database.
- When a view is referenced in SQL statement following steps will be followed:
  - o Its definition is retrieved from database.
  - o The base tables are opened.
  - o A query, specified in definition is executed. o A view is created on top of the base tables.
- When any operation is performed on view, it is actually performed on the base table.

- For example, any **SELECT** operation on view displays data from the base table. In a similar way, **INSERT, UPDATE, DELETE** operations modify the contents of the base table.

❖ Types of Views

- View can be classified into two categories based on which type of operations they allow:

1) Read-only View:

- Allows only SELECT operation, this means user can only view data.
- No INSERT, UPDATE or DELETE operations are allowed. This means contents of base table cannot be modified.

2) Updateable View:

- Allows **SELECT** as well as **INSERT, UPDATE** and **DELETE** operations. This means contents of the base tables can be displayed as well as modified.

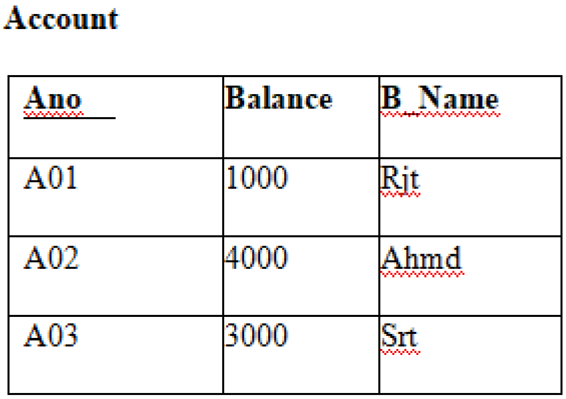
❖ Creating a View

- A view can be created using syntax as given below:

Syntax:

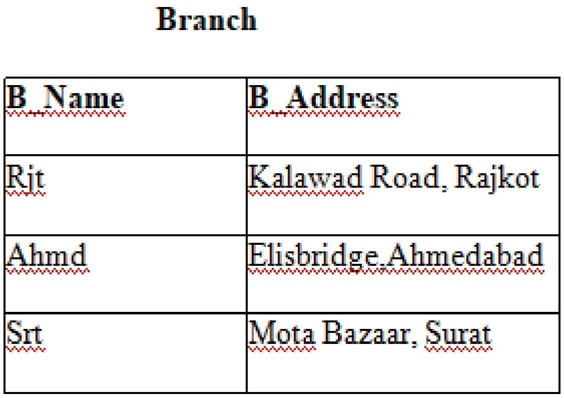
CREATE [OR REPLACE] VIEW viewName
As SELECT … ….
[WITH READ ONLY];

- This statement creates a view based on query specified in **SELECT** statement.
- **OR REPLACE** option re-creates the view if it is already existing maintaining the privileges granted to view that is given by view Name.
- **WITH READ ONLY** option creates **read-only views**. If this option is not provided then **by default updatable views** are created.
- The **SELECT** statement can include **WHERE, ORDER BY, GROUP BY** clauses if required.
- A view can be created using single base table as well as multiple base tables using joins.
- The following examples explain how to create views and how to use them in SQL statements. Consider tables – Account and Branch as given in below figure:



Account

Ano	Balance	B_Name
A01	1000	Rjt
A02	4000	Ahmd
A03	3000	Srt



Branch

B_Name	B_Address
Rjt	Kalawad Road, Rajkot
Ahmd	Elisbridge,Ahmedabad
Srt	Mota Bazaar, Surat

Example:

```
CREATE VIEW Acc_Rjt
AS SELECT * FROM Account
WHERE B_Name = ‘Rjt’;
```

Output:

```
View created.
```

❖ Advantages of View

- View the data without storing the data into the object.
- Restricts the view of a table. i.e. can hide some of columns in the tables.
- Join two or more tables and show it as one object to user.
- Restricts the access of a table so that nobody can insert the rows into the table.
- There are two major advantages of views:
    - Flexible enforcement of security
    - Simplification of complex query

❖ Disadvantages of Views

- Cannot use DML operations on view.
- When table is dropped view becomes inactive.
- View is an object, so it occupies space.

❖ Destroying a View

- The DROP VIEW command drops the specified view.
- The base table will not be affected if a view is destroyed.
- If a base table is dropped or column included in view are altered then view will not be valid further.
- Oracle issues an error message while using such in-valid views.

Syntax:

```
DROP VIEW  viewName;
```

Example:

```
DROP VIEW Acc_Branch;
```

Output: **View Dropped.**

3.5 Synonym: Create, Drop synonym

- A synonym is an alternative name for database object such as tables, indexes, sequences.

- A synonym can be used to hide the actual identity of the object being referenced.

- For example, if there is a need to hide name of some particular table, then a synonym can be created to refer that table, hiding the original name.

- Another use of the synonym is to abbreviate (Shorten) the table names, particularly tables from other users.

- For example, user1 can create synonym for Customer table owned by user2.Appropriate privileges must be granted to a user before the user can use the synonym.

❖ Creating a Synonym:

Syntax:

```
CREATE SYNONYM          synonymName
FOR    objectName;
```

Example:

```
CREATE  SYNONYM         Cust
FOR    user1.Customer;
```

Output:

```
Synonym Created.
```

❖ Destroying a Synonym:

Syntax:

```
DROP  SYNONYM   synonymName;
```

3.6 Sequences: Create, alter, Drop sequences

- To distinguish different records of a table from each other, it is required that each record must have distinct values.
- The **primary key** constrain ensures this by not allowing duplicate or NULL values in columns defined as a primary key.
- Such column generally contain sequential numbers such as 1, 2, 3,… or combination of sequential values with some strings, such as ‘A01’, ‘A02’,….
- While entering data manually in insert or update operations, it is difficult to track such type of sequence.
- An Oracle object, a **Sequence** helps to ease the process of creating unique identifiers for a record in a database.

- **A Sequence is simply an automatic counter, which generates sequential numbers whenever required**.
- A value generated can have maximum **38 digits**.
- A sequence can be defined for following purpose:
- To generate numbers in ascending or descending order. o To provide intervals between numbers.
- To caching sequence numbers in memory to speed up their availability.

❖ Creating a Sequence

Syntax:

```
CREATE SEQUENCE sequence_name
START WITH initial_value
INCREMENT BY increment_value
MINVALUE minimum value
MAXVALUE maximum value
CYCLE|NOCYCLE ;
```

sequence_name: Name of the sequence.

initial_value: starting value from where the sequence starts.
Initial_value should be greater than or equal to minimum value and less than equal to maximum value.

increment_value: Value by which sequence will increment itself.
Increment_value can be positive or negative.

minimum_value: Minimum value of the sequence.
maximum_value: Maximum value of the sequence.

cycle: When sequence reaches its set_limit it starts from beginning.

nocycle: An exception will be thrown if sequence exceeds its max_value.

- A default sequence created **without any options**, always start with 1, is in **ascending order** and values are **incremented by 1**.

❖ NEXTVAL and CURRVAL

- Oracle provides two pseudo column – **NEXTVAL** and **CURRVAL**.
- Once a sequence is created, you can access its values in **SQL** statements with the CURRVAL pseudo column, **which returns the current value of the sequence.**
- The **NEXTVAL** pseudocolumn, **which increments the sequence and returns the new value**.
- These pseudo columns are used with a Sequence name as described below:

Syntax:

squenceName**.CURRVAL**

- Returns the current value of the sequence.

Syntax:

squenceName**.NEXTVAL**

- Increases the value of the sequence and returns the next value.
- Generally the values generated by the Sequence are numerical values.

Example

```
CREATE SEQUENCE sequence_1
start with 1
increment by 1
minvalue 0
maxvalue 100
cycle;
```

Output:

Sequence Created.

❖ Destroying a Sequence

- A sequence can be destroyed as described below.

Syntax:

DROP Sequence sequence_name

3.7 Index: Unique and composite – Create, Drop

- Search is always efficient when data to be searched is sorted in some specific order such as in ascending order.
- If records are not sorted then any query fired on a table to search sequentially testing values of all records one by one.
- An Index is an ordered list of contents of the column (or a group of columns) of a table.
- An index is similar to a table. It contains at-least two columns:
  1) A column having sorted data on which an index is created.
  2) A column representing RowID for each row in a table.
- A RowIDis a unique identifier for each record inserted in a table.

❖ Advantage:

- As content of the name column is sorted in index, **searching process will be faster**.
- Also index contains only two columns. So, **updating index on each insert, update, delete operation on table will not consume much time**.

❖ Disadvantages:

- Indexes slow down DML (i.e. inserts, updates and deletes).
- Indexes may make your queries slower instead of faster.

❖ RowID – A Unique Identifier of a Record

- A **RowID** is a unique identifier for each record inserted in a table.
- A RowID is a hexadecimal string and contains logical address of the location in a database where a particular record is stored.
- Oracle assigns a unique id for each and every record inserted in a table and that is used to create indexes.
- Oracle provides a **pseudo column**, named ROWID, to retrieve RowID associated with records in a table.
- ROWID column can be used like any other column in SELECT statement.
- The format for RowID can be any of the following:

1) **Extended:**
   This format is an **18 digit string** of the form **OOOOOOFFFBBBBBBRRR**. o This format is used by **Oracle8i and higher** versions.

2) **Restricted:**
   This format is a **15 digit string** separated with **dots** of the form **BBBBBBBB.RRRR.FFF.** o This format is used by **Oracle7 and earlier** releases.

❖ Types of Indexes

- There are four types of indexes:

1) **Duplicate Indexes**
2) **Unique Indexes**
3) **Simple Indexes**
4) **Composite Indexes**

❖ Simple Indexes:

- An index created on a single column of a table is called a Simple Index.

❖ Composite Indexes:

- An index created on more than one column is called a Composite Index.

❖ Creating simple Index

Syntax:

```
CREATE [UNIQUE] INDEX indexName
ON  tableName  (columnName);
```

- By default indexes are created as Duplicate Indexes.
- If **UNIQUE** option is provided while creating an index, it will be considered as a unique Index.

Example:

CREATE INDEX indCustName **ON** Customer **(**Name**)**;

Output:

Index Created.

❖ Creating Composite Index

Syntax:

CREATE [UNIQUE] INDEX indexName
ON tableName **(**columnName1, columnName2 **)**;

- If **more than one column** is provided while creating an index, it will be considered as as **Composite Index**. Otherwise, indexes are created as Simple Indexes.
- If index is created on more than two columns, other column will be considered only when the previous all columns contain duplicate data.
- In above syntax, second column will be considered only when the first column contains duplicate data. In such case, **sorting is performed based on data of the second column**.

❖ Destroying an Index

Syntax:

DROP INDEX indexName;

- This command drops an index given by indexName.
- Once an index is dropped, it can be recreated whenever required.

Example

DROP INDEX indCustName;

Output:

Index Dropped.