Unit – 3
Database Integrity Constraints and Objects

Y.A.HATHALIYA

Definitions

- Attribute = Name of Column in a Relation
- 2 attribute in same relation (table) can't have same name

Arity of a Table = No. of Columns = 3

STAFF_ID	STAFF_NAME	STAFF_MNO
1	CEANS	8200106799
2	CECNK	1234567890
3	CENJR	123456789

Total No. of Tuples = Cardinality of relation

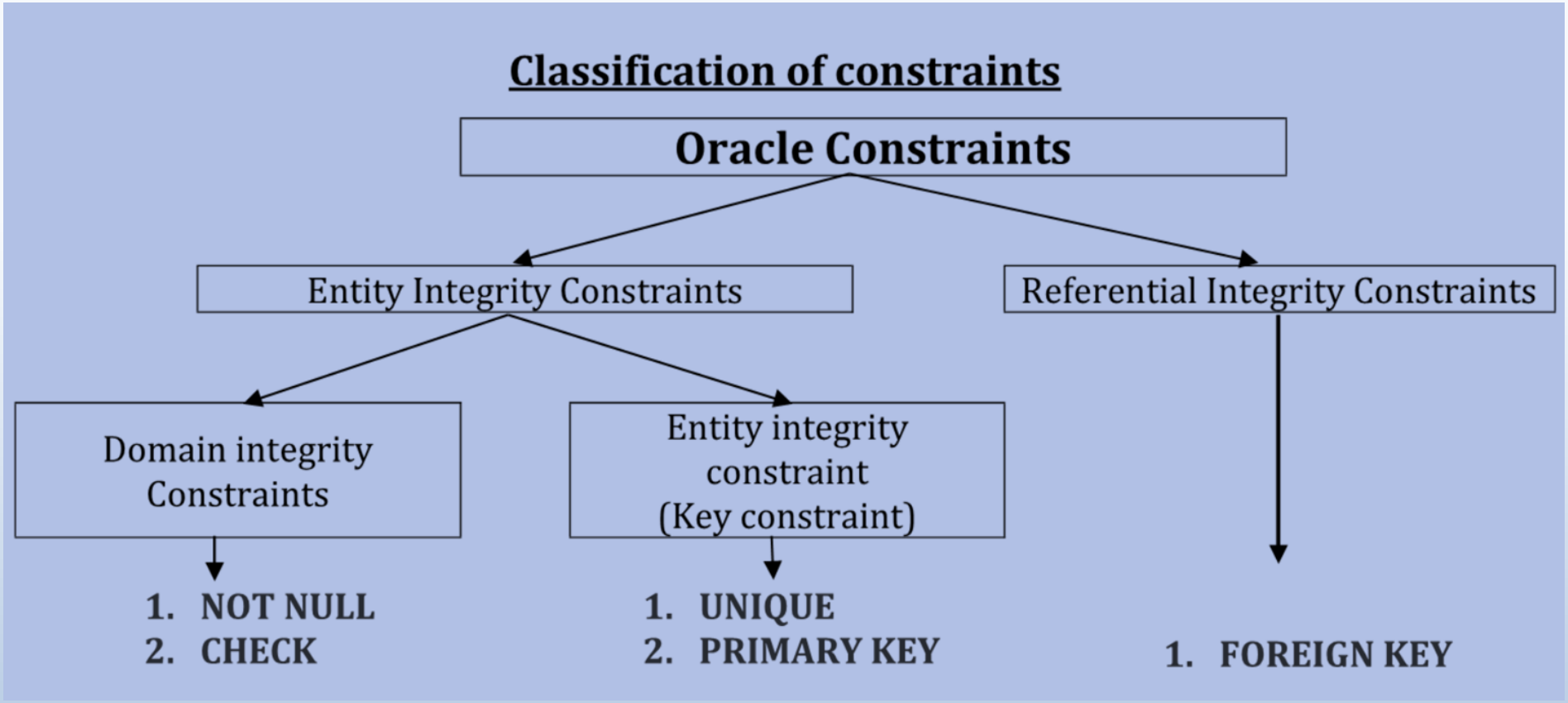
Row = Record = Tuple

Set of permitted values for that attribute = Domain

- Predefined row/column format for storing information.
- Relation = Table

Constraints

- Constraint means → restriction.
- In oracle, ***'a constraint is a rule that restricts the values that may be present in the database'.***
- Constraints are used to limit the type of data that can go into a table.
- It maintains the quality of information in data.
- Main purpose → ensure data integrity and reliability.
- If any SQL statement tries to violate a constraint, oracle cancels that operation and gives error message.
- Constraints can be classified into two broad categories:
  1. ***Entity integrity constraints***
  2. ***Referential integrity constraints***


Classification of constraints
Oracle Constraints
Entity Integrity Constraints
Referential Integrity Constraints
Domain integrity Constraints
Entity integrity constraint (Key constraint)
1. NOT NULL
2. CHECK
1. UNIQUE
2. PRIMARY KEY
1. FOREIGN KEY

1. Entity integrity constraints

- Restrict the values in a row of a table. These constraints are further divided into two sub categories.
  - ***I. Domain integrity constraints***
  - ***II. Key constraints (OR Entity integrity constraints)***

I. Domain integrity constraints

- *Domain* means permitted set of values for an attribute of an entity.
- These constraints specify that the value of each column must belong to the domain of that column.
- For example, balance for any account should not be negative, mobile number of any person never contains any characters etc.
- NOT NULL and CHECK constraints are domain integrity constraints.

II. Key constraints

- These constraints are used to identify each row (or record) uniquely in the table.
- PRIMARY KEY and UNIQUE constraints are key constraints.

2. Referential integrity constraints

- These constraints specified between two tables.
- It ensures relationship between two tables.
- FOREIGN KEY is referential integrity constraint.

Constraint Name	Description
NOT NULL	Ensures that a column cannot have NULL value.
CHECK	Ensures that all the values in a column satisfy certain conditions.
UNIQUE	Ensures that all the values in a column are different, not duplicate.
PRIMARY KEY	Ensures that all the values in a column are different (unique) and cannot have NULL values.
FOREIGN KEY	Provides relationship between two tables. Uniquely identifies records in another table.

- Constraints defined at the time of creating the table. They can be defined at column level or table level.

❑ **Column level constraints**

- Defined along with the column definition.

Syntax

```
Column_Name datatype (size) constraint_definition
```

❑ **Table level constraints**

- Defined after defining all the columns of the table.

Syntax

```
CREATE TABLE table_name
(Column1 data_type1(size),
Column2 data_type2(size),...,
ColumnN data_typeN (size),
Constraint_definition1,..., Constraint_definitionN);
```

NOT NULL Constraints

- It is an entity integrity constraint.
- It is a kind of domain integrity constraint.

In every constraint, you have to write its related statements initially (as above), like it is of which type or which kind of constraint.

- By default Oracle allows NULL values in database. But in some situation, it is required that a field cannot be empty. That means we must have some value at this place. NOT NULL provides solution in that case.
- For example, the balance column of banking table cannot be empty. Or in a name field of student data cannot be empty or NULL.
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- This constraint cannot be defined at table level. (*In fact, this is the only constraint which cannot be defined at table level*)

Example:

```
SQL> CREATE TABLE test
     (eno number(3) NOT NULL, name varchar2 (10));

Table created.

SQL> DESC test;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ENO                                       NOT NULL NUMBER(3)
 NAME                                               VARCHAR2(10)

SQL> INSERT INTO test VALUES (1, 'abc');

1 row created.

SQL> INSERT INTO test VALUES (NULL, 'xyz');
INSERT INTO test VALUES (NULL, 'xyz')
                         *
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."TEST"."ENO")
```

Define NOT NULL at 'eno' column.

Violation of constraint, command rejected

(In above example, we can see that after defining NOT NULL constraint, we can check that constraint by 'DESC' command and we cannot insert NULL value then after.)

CHECK Constraints

- It is an entity integrity constraint.
- It is a kind of domain integrity constraint.
- Generally used to implement business rules. Give the examples of some business rules.
- After implementation of CHECK constraint on any column, insert or update operation on that column must follow this constraint.
- If any operation violates condition, it will be rejected.
- It allows NULL values to be inserted.

Syntax (*CHECK constraint can be defined at column level and table level:***)**

Syntax (at column level definition):
...Column_Name datatype (size) CHECK (condition)
Syntax (at table level definition):
...CHECK (condition)

Example (Column level definition)

```
SQL> CREATE TABLE test
  2  (id number CHECK (id < 10), name varchar2(10));

Table created.

SQL> INSERT INTO test VALUES(1, 'AA');

1 row created.

SQL> INSERT INTO test VALUES(NULL, 'BB');

1 row created.

SQL> INSERT INTO test VALUES(15, 'CC');
INSERT INTO test VALUES(15, 'CC')
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C004596) violated
```

Column level definition: Define CHECK at 'id' column

NULL can be inserted

Violation of CHECK constraint

Example (Table level definition)

```
SQL> CREATE TABLE test
  2  (id number(3), name varchar2(10),
  3  CHECK (id < 10 AND name like 'a%'));

Table created.

SQL> INSERT INTO test VALUES (1, 'aarush');

1 row created.

SQL> INSERT INTO test VALUES (2, 'bhagya');
INSERT INTO test VALUES (2, 'bhagya')
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C004597) violated


SQL> INSERT INTO test VALUES (15, 'aarush');
INSERT INTO test VALUES (15, 'aarush')
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C004597) violated
```

Table level definition

Violation of CHECK constraint

(We can combine multiple columns with CHECK at table level definition.)

UNIQUE Constraints

- Also called key constraints.
- These constraints are used to identify each row (or record) uniquely in the table.
- UNIQUE and PRIMARY KEY constraints are key constraints.

❖ **UNIQUE constraint**

- It is a kind of entity integrity constraint or key constraint.
- State the situation where it is used.
- A column or combination of columns in which we apply UNIQUE constraint is called ***UNIQUE key***.
- Used to identify data uniquely from the database.
- Don't allow duplicate values on which this constraint applied. But allows NULL values.
- Can be in composite form (defined on more than one columns).

Syntax

Syntax (at column level definition):
...Column_Name datatype (size) UNIQUE
Syntax (at table level definition):
...UNIQUE (column1, column2,..., columnN)

Example (column level)

```
SQL> CREATE TABLE test
  2  (id number(3) UNIQUE,
  3  name varchar2 (10));

Table created.

SQL> INSERT INTO test VALUES (1, 'aa');

1 row created.

SQL> INSERT INTO test VALUES (NULL, 'bb');

1 row created.

SQL> INSERT INTO test VALUES (1, 'cc');
INSERT INTO test VALUES (1, 'cc')
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C004598) violated
```

Define UNIQUE for 'id' column at column level definition

UNIQUE allows NULL values, but cannot allow duplicate values

Example (table level)

```
SQL> CREATE TABLE test
  2  (id number(3), name varchar2(10), mob_no number(10),
  3  UNIQUE (name, mob_no));

Table created.

SQL> INSERT INTO test VALUES (1, 'aa', 9898989898);

1 row created.

SQL> INSERT INTO test VALUES (2, 'aa', 9898989898);
INSERT INTO test VALUES (2, 'aa', 9898989898)
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C004599) violated
```

PRIMARY KEY Constraints

- It is a kind of entity integrity constraint or key constraint.
- ***PRIMARY KEY is a column or combination of columns by which we can uniquely identify a record in a table.***
- It does not allow duplicate values and NULL values.
- Concept of simple and composite primary key.
- We cannot define more than one primary key in a table.
- **PRIMARY KEY = UNIQUE + NOT NULL**

Syntax

Syntax (at column level definition):
...Column_Name datatype (size) PRIMARY KEY
Syntax (at table level definition):
...PRIMARY KEY (column1, [column2,..., columnN)

Example

```
SQL> CREATE TABLE test
  2  (id number(3) PRIMARY KEY,
  3  name varchar2(10));

Table created.

SQL> INSERT INTO test VALUES (1, 'aa');

1 row created.

SQL> INSERT INTO test VALUES (NULL, 'bb');
INSERT INTO test VALUES (NULL, 'bb')
                         *
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."TEST"."ID")


SQL> INSERT INTO test VALUES (1, 'cc');
INSERT INTO test VALUES (1, 'cc')
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C004600) violated
```

Column level definition of PRIMARY KEY

PRIMARY KEY does not allow NULL or duplicate values

```
SQL> CREATE TABLE test
  2  (id number(3), name varchar2(10),
  3  PRIMARY KEY (id));

Table created.
```

Table level definition of PRIMARY KEY

FOREIGN KEY Constraints

- It links two tables, so it ensures relationship between two tables.
- FOREIGN KEY is referential integrity constraint.
- ***FOREIGN KEY is a column or combination of columns that link two tables in which a foreign key of a table refers to a primary key of another table.***
- It also ensures consistency among records of two tables.
- Generally, the table containing the foreign key is known as **child table** or **detail table** or **foreign table**, and the table containing primary key that is referenced by foreign key is known as **parent table** or **master table**.
- The FOREIGN KEY constraint enforces different restrictions on detail table and master table.

➥ ***Restrictions on detail table (child table or foreign table)***

➥ ***Restrictions on master table (or parent table)***

➥ **Restrictions on master table (or parent table):**

- Master table must contains a primary key which is referred by foreign key in detail table.
- Delete or update operation on records is not allowed in master table, if that records are present in detail (foreign) table.
- We cannot delete master (parent) table if it is referred by any other table using foreign.

Consider the following two tables for example:

- In above example, in student_marks table, Roll_no column is a foreign key which is reference to the student_master table and the same column is primary key in master table.
  We can define more than one key as foreign key in a table that are referencing to more than one tables.

Syntax (at column level definition):

...Column_Name datatype (size) REFERENCES table_name (column_name) [ON DELEET CASCADE]

Syntax (at table level definition):

...FOREIGN KEY (column1, [column2,..., columnN]) REFERENCES table_name (column_name) [ON DELETE CASCADE]

Example:

```
SQL> CREATE TABLE test11
     (id number(3) PRIMARY KEY, name varchar2(10));

Table created.

SQL> CREATE TABLE test22
     (id number(3) REFERENCES test11 (id), subject varchar2(10));

Table created.

SQL> drop table test11;
drop table test11
           *
ERROR at line 1:
ORA-02449: unique/primary keys in table referenced by foreign keys
```

Column level definition of FOREIGN KEY

We can't drop table if it is referred in another table

```
SQL> CREATE TABLE test11
      (id number(3) PRIMARY KEY,
     name varchar2(10));

Table created.

SQL> CREATE TABLE test22
     (id number(3), subject varchar2(10),
     FOREIGN KEY (id) REFERENCES test11 (id));

Table created.
```

Table level definition of FOREIGN KEY

Note:

- Master table must exist before creating a detailed table (child table).
- It is also necessary for records to be present in master table, before inserting corresponding records in detail table (child table).
- You cannot update a value in master table which is referred by foreign table.
- You cannot delete or drop the master table that is referred by another table.
- You cannot insert data in the detail table that is not available in master table.

Primary key	Foreign key
- It is a kind of entity integrity constraint.	- It is a kind of referential integrity constraint.
- Primary key uniquely identifies a record in a table.	- It is used to link two tables. It means the foreign key in one table refers to the primary key of another table.
- Primary key cannot accept NULL value.	- Foreign key can accept NULL values.
- Primary key cannot allow duplicate values.	- Foreign key allows duplicate values.
- There can be maximum of one primary key in a table.	- There can be more than one foreign key in a table.
- It cannot create a parent-child relationship in a table.	- It can make a parent-child relationship in a table.

On Delete Cascade

- This option overcomes the restriction enforced on master table for delete operation.
- With this option, when any record from master table is deleted, all the corresponding records from detail table will be deleted automatically.

Example (*define foreign key with ON DELETE CASCADE option***)**

```
SQL> CREATE TABLE test11
  2  (id number(2) PRIMARY KEY,
  3  name varchar2(10));

Table created.

SQL> CREATE TABLE test22
  2  (id number(2), subject varchar2(10),
  3  FOREIGN KEY (id) REFERENCES test11 (id)
  4  ON DELETE CASCADE);

Table created.
```

Define 'on delete cascade' option with foreign key

```
SQL> select * from test11;

        ID NAME
---------- ----------
         1 AA
         2 BB
         3 CC

SQL> select * from test22;

        ID SUBJECT
---------- ----------
         2 XX
         3 YY
         1 ZZ
```

Suppose we have data in two tables with defined primary key and foreign key

```
SQL> DELETE FROM test11 WHERE id=2;

1 row deleted.

SQL> SELECT * FROM test11;

        ID NAME
---------- ----------
         1 AA
         3 CC

SQL> SELECT * FROM test22;

        ID SUBJECT
---------- ----------
         3 YY
         1 ZZ
```

When we delete a record from master table, the corresponding detail table data is deleted automatically.

Database Objects

- A database object is any defined object in a database that is used to store or reference data.
- Anything which we make from CREATE command is known as Database Object.
- Some examples of database objects are: table, view, sequence, index, synonym etc.

Views

- It is one of database objects in Oracle.
- It is virtual or logical table.
- It does not have any storage space, only its definition is stored in the database.
- View is created using CREATE command.
- A View can be created using one or more tables (called ***base tables***)
- Fields of the views are from one or more real tables.
- We can perform operations on views same as base tables.
- The names of the columns will remain same as base table.
- Views are mainly used to solve complex queries.
- We cannot give a name to view that is already taken by any other database object.

❖ **Types of view**

➥ **Read only view**

  – This view can only be viewed by a user and cannot be modified.
  – It allows only SELECT operation.

➥ **Updatable view**

  – In it, we can insert, delete or update the records of the base table using a view.
  – It allows SELECT as well as INSERT, DELETE and UPDATE operations.

❖ Create a View from single table

- A view is created with the CRAETE VIEW statement.

Syntax

```
CREATE [OR REPLACE] VIEW View_Name
AS SELECT column1, column2,,,
FROM table_name
WHERE condition
[WITH READ ONLY];
```

In above syntax

- CREATE keyword is used to create a view.
- OR REPLACE keyword creates a new view if we try to create a view which name is already exist. It is optional part.
- AS SELECT statement with FROM and WHERE clause create a view from base table with specific condition. SELECT statement can also include ORDER BY or GROUP BY clause.
- [WITH READ ONLY] is optional, using that key word we can create read only views. If this option is not provided at the time of creating view, by default updateable view is created.

Example (consider a table ***test*** *as below***)**

```
SQL> select * from test;

        ID NAME       CITY
---------- ---------- -----
         1 a          aa
         2 b          bb
         3 c          cc
         4 d          dd
         5 e          ee
```

- Now, we are creating an updateable view 'test_*view*' that contains the records having id less than 4.

```
SQL> CREATE VIEW test_view as
  2  SELECT * FROM test
  3  WHERE id < 4;

View created.

SQL> SELECT * FROM test_view;

        ID NAME       CITY
---------- ---------- --------
         1 a          aa
         2 b          bb
         3 c          cc
```

Example (create read only view using 'WITH READ ONLY'**)**

```
SQL> CREATE VIEW test_view
  2  AS SELECT * FROM test
  3  WHERE id < 4
  4  WITH READ ONLY;

View created.

SQL> SELECT * FROM test_view;

        ID NAME       CITY
---------- ---------- -------
         1 a          aa
         2 b          bb
         3 c          cc
```

- Now we try to update the records of base table using this view.

```
SQL> UPDATE test_view
  2  SET id = 11 WHERE id = 1;
SET id = 11 WHERE id = 1
    *
ERROR at line 2:
ORA-01733: virtual column not allowed here
```

❖ Creating view from multiple tables

Here we can create a view from multiple base tables.

Suppose we have two tables → student and branch (as shown below)

```
SQL> SELECT * FROM student;

SI         BID SNAME           SPI CR
-- ---------- -------- ---------- --------
c1           7 aarush         9.5
m1          19 bhagya         7.9
c2           7 avani          8.3 aarush
m2          19 aarav          6.4 bhagya
a1           2 ved            6.8
i1          21 misha          7.1
```

student table

```
SQL> SELECT * FROM branch;

       BID BNAME      HOD
---------- ---------- --------
        19 mechanical dinesh
         7 computer   parimal
         9 electrical ekta
         2 automobile parth
        51 cddm
```

branch table

Example:

Create a view from above tables which display the consistent information of student name and concerned branch name.

```
SQL> CREATE VIEW stbr_view
  2  AS SELECT sname, bname
  3  FROM student, branch
  4  WHERE student.bid = branch.bid;

View created.

SQL> SELECT * FROM stbr_view;

SNAME    BNAME
-------- ----------
aarush   computer
bhagya   mechanical
avani    computer
aarav    mechanical
ved      automobile
```

Now we can execute different queries on above view like simple table.

❖ Drop a view (Remove a view)

- DROP command is used.
- The data of the base table will not be affected.
- Even when a base table is deleted, view is no longer valid. If a column which is referred in a view is deleted from base table, view is not valid then after.

Syntax

```
DROP VIEW View_Name;
```

Example

```
SQL> DROP VIEW person_view;

View dropped.
```

- **Advantages of view**
  - ✓ Simplifying data retrieval
  - ✓ Simplicity in structure
  - ✓ Implementing data security
  - ✓ Useful in re-designing the database
  - ✓ Maintains logical data independence
- **Limitations of view**
  - ✓ It is logical or virtual, do not have its own data.
  - ✓ Not possible to create a view on index.
  - ✓ We can reference maximum of 61 tables in a view.
  - ✓ Columns of a view cannot be renamed.

Synonym

- It is a kind of database object.
- It provides another name for database objects.
- It hides actual identity of the object.
- Another use → make the object name smaller and easy.
- We can manipulate the table data using synonym.

Syntax

```
CREATE SYNONYM Synonym_Name
FOR Object_Name;
```

Example

Create synonym for *test* table.

Input command →

```
SQL> CREATE SYNONYM tst
  2  FOR test;
```

Output →

```
Synonym created.

SQL> SELECT * FROM tst;

        NO NAME                 AGE
---------- ---------- ----------
         1 aarush                5
         2 bhagya               10
```

❖ Removing or Drop synonym

- To remove the synonym DROP command is used.

Syntax

```
DROP SYNONYM Synonym_Name
```

Example

Input command →

```
SQL> DROP SYNONYM tst;
```

Output →

```
Synonym dropped.
```

Sequence

- It is a kind of database object.
- ***A sequence is an automatic counter which generates sequential numbers whenever required.***
- Purpose of a sequence:
  - ✓ Generates numbers in order
  - ✓ Provides internal between numbers
  - ✓ Caching of sequence numbers in memory

Syntax

```
CREATE SEQUENCE Sequence_Name
[ START WITH Initial_Value ]
[ INCREMENT BY Increment_Value ]
[ MINVALUE Minimum_Value ]
[ MAXVALUE Maximum_Value ]
[ CYCLE  | NO CYCLE ]
[ CACHE  <size value>| NOCACHE ];
```

Description of different arguments in above syntax

Argument Name	Description
Sequence Name	It specifies the 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 and equal to maximum value.
Increment Value	Value by which the 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	Specifies to repeat cycle of generating values after reaching maximum value.
NOCYCLE	Specifies that no more numbers can be generated after reaching maximum values
CACHE	Specifies how many values to generate in advance and to keep in memory for faster access. Minimum value is 2 for this option.
NOCAHCE	Specifies that no value is generated in advance.

Example

Create a simple sequence in which initial value is 1, incremented by 1 and maximum value is 5.

Input command →

```
SQL> CREATE SEQUENCE firstseq
  2  START WITH 1
  3  INCREMENT BY 1
  4  MAXVALUE 5;

Sequence created.
```

Output →

Now we have a sequence 'firstseq'. But only creating a sequence is useless until we are not using the values of it.

To use the values generated in sequence can only be accessed by two pseudo columns → **NEXTVAL** and **CURRVAL**.

Syntax of NEXTVAL and CURRVAL

SequenceName.currval	Returns the current value of the sequence
SequenceName.nextval	Increments the value of sequence and returns the next value

Example

```
SQL> CREATE SEQUENCE firstseq
  2  START WITH 1
  3  INCREMENT BY 1;

Sequence created.

SQL> CREATE TABLE test(roll_no number(10), name varchar2(10));

Table created.

SQL> INSERT INTO test VALUES (firstseq.nextval, 'BHAGYA');

1 row created.

SQL> INSERT INTO test VALUES (firstseq.nextval, 'AARUSH');

1 row created.

SQL> SELECT * FROM test;

   ROLL_NO NAME
---------- ----------
         1 BHAGYA
         2 AARUSH
```

In above example, we have created a sequence 'firstseq' and then using the pseudocode 'nextval' we have used the values of sequence and these values are inserted in table *test*.

➥ Alter sequence

- Used to modify the properties of an existing sequence.
- ALTER SEQUENCE command is used for that.

Syntax

```
ALTER SEQUENCE Sequence_Name
[ START WITH Initial_Value ]
[ INCREMENT BY Increment_Value ]
[ MINVALUE Minimum_Value ]
[ MAXVALUE Maximum_Value ]
[ CYCLE  | NO CYCLE ]
[ CACHE  <size value>| NOCACHE ];
```

✍ *All the parameters are same as creating a sequence, only difference is that we can change these parameters as per our need using ALTER SEQUENCE command.*

Example (*How to alter the existing sequence*)

```
SQL> create sequence firstseq
  2  start with 1
  3  increment by 1
  4  maxvalue 3;

Sequence created.

SQL> ALTER SEQUENCE firstseq
  2  MAXVALUE 5;
```

Here, the sequence is altered

❖ Removing or Drop a sequence

To remove the synonym DROP command is used.

Syntax

```
DROP SEQUENCE Sequence_Name
```

Example

Input command →

```
SQL> DROP SEQUENCE firstseq;
```

Output →

```
Sequence dropped.
```

Index

- It is a kind of database object which make data searching or data retrieval faster. Works same way as an index of a book.
- Index created on a column of a table is called ***index key***.
- For data retrieval, oracle uses two methods → table scan and index scan. By default, Oracle use table scan when an index is not defined.
- In case of index scan, Oracle used RowId to get the data.

➥ **Creating an index**

Syntax

```
CREATE [UNIQUE] INDEX Index_Name
ON Table_Name (Column_Name);
```

Example

```
SQL> CREATE INDEX I1
  2  ON test(age);

Index created.
```

➥ Types of an index

- Different types of index are listed below:

✓ Duplicate index	Unique index
Simple index	Composite index

❑ Duplicate index

It allows duplicate values in a column where we define index.

❑ Unique index

We cannot have duplicate values in a column where we have declared unique index. We cannot create unique index on a column which has duplicate values.

It is automatically created in case of primary key or unique key constraints.

Syntax

```
CREATE UNIQUE INDEX Index_Name
ON Table_Name (Column_Name);
```

❑ Simple index

When an index is created on a single column of a table, it is called simple index.

Syntax

```
CREATE [UNIQUE] INDEX Index_Name
ON Table_Name (Column_Name);
```

❑ Composite index

When an index is created on a more than one column of a single table, it is called ~~simple~~ index.

Composite

Syntax

```
CREATE [UNIQUE] INDEX Index_Name
ON Table_Name (Column_Name1, Column_Name2,,,);
```

❑ Implicit index

Automatically created by database server.

Automatically created when we define primary key or unique key constraint.

➢ Disadvantage of index

- Index is faster with SELECT and WHERE clause but slower with INSERT and UPDATE clause So update a table with index on that column will be slower.

➢ When Index should not be used

- Index should not be used with column that contains NULL values.
- When the table needs to be updated frequently.
- When the table size is small.

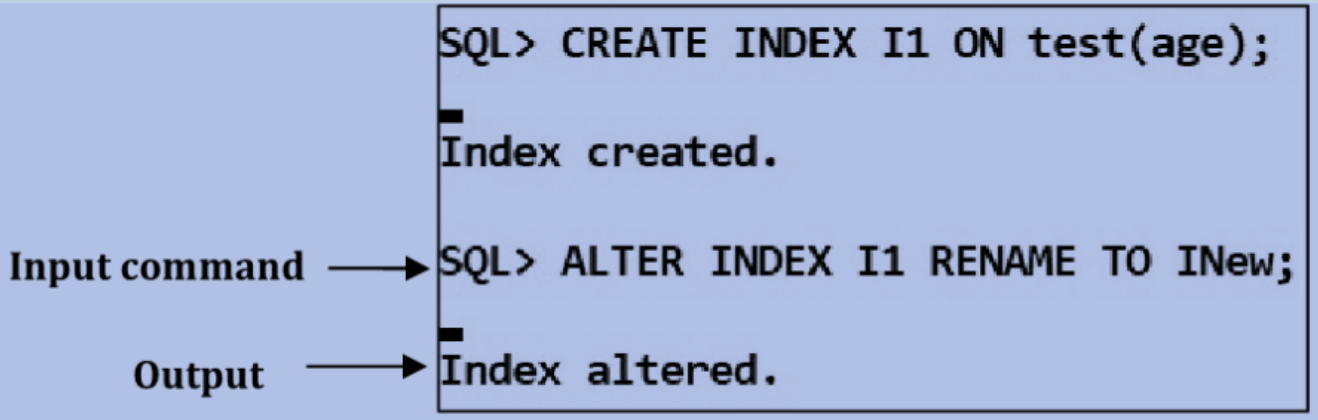
❖ Rename an index

- We can rename the existing index using **ALTER** command.

Syntax

```
ALTER INDEX Old_Index_Name
RENAME TO New_Index_Name;
```

Example



❖ Drop index (Remove an index)

- To remove or drop the index, DROP command is used.

Syntax

```
DROP INDEX Index_Name;
```

Example