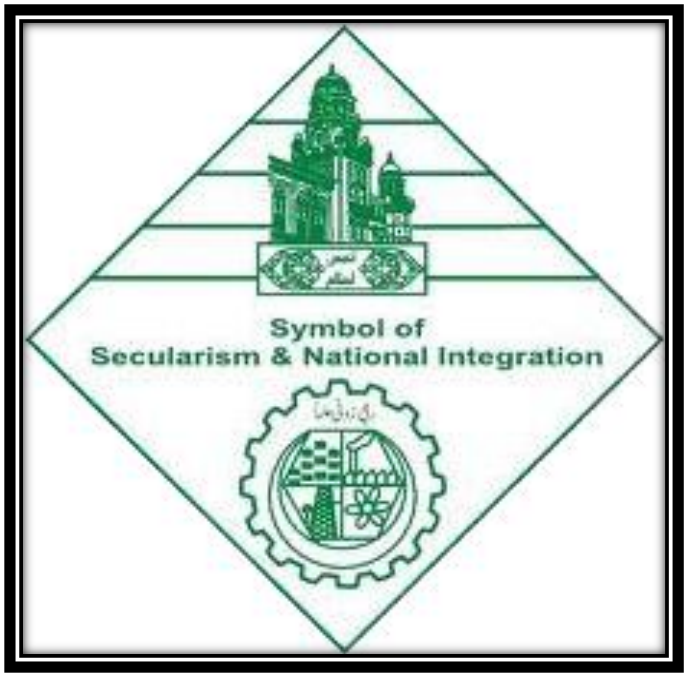
# M. H. SABOO SIDDIK POLYTECHNIC.




Database management system micro – project.

-UNDER THE GUAIDANCE OF
MOHAMMAD ALI SIR.

TOPIC :

Database of COVID-19 related Information.

# PRESENTED BY :

| SR.NO | ROLL NO : | NAME : |
|---|---|---|
| 1. | 19416 | Loladia Ayesha. |
| 2. | 19401 | Ansari Anam. |
| 3. | 19420 | Rakhangi Arisha. |
| 4. | 19428 | Shaikh Nusra. |

# MAHARASHTRA STATE
# BOARD OF TECHNICAL EDUCATION
## CERTIFICATE

This is to certify that

Ms: _____

Roll No: _____ Enrollment No: _____

of Third Semester of Diploma in **Computer Engineering** of Institute

**M. H. Saboo Siddik Polytechnic** (Code **: 0002**) has completed their term work satisfactorily in course **Database Management System(22319)** for the academic year 2020 to 2021 as prescribed in the curriculum.

Place : Mumbai Date : .....

Exam. Seat No : .....

Subject Teacher. Head of the Department. Principal

**Seal of Institute**

# MAHARASHTRA STATE
# BOARD OF TECHNICAL EDUCATION
## CERTIFICATE

This is to certify that

Ms: _____

Roll No: _____ Enrollment No: _____

of Third Semester of Diploma in **Computer Engineering** of Institute

**M. H. Saboo Siddik Polytechnic** (Code **: 0002**) has completed their term work satisfactorily in course **Database Management System(22319)** for the academic year 2020 to 2021 as prescribed in the curriculum.

Place : Mumbai Date : .....

Exam. Seat No : .....

Subject Teacher. Head of the Department. Principal

**Seal of Institute**

# MAHARASHTRA STATE
# BOARD OF TECHNICAL EDUCATION
## CERTIFICATE

This is to certify that

Ms: _____

Roll No: _____ Enrollment No: _____

of Third Semester of Diploma in **Computer Engineering** of Institute

**M. H. Saboo Siddik Polytechnic** (Code **: 0002**) has completed their term work satisfactorily in course **Database Management System(22319)** for the academic year 2020 to 2021 as prescribed in the curriculum.

Place : Mumbai Date : .....

Exam. Seat No : .....

Subject Teacher. Head of the Department. Principal

**Seal of Institute**

# MAHARASHTRA STATE
# BOARD OF TECHNICAL EDUCATION
## CERTIFICATE

This is to certify that

Ms: _____

Roll No: _____ Enrollment No: _____

of Third Semester of Diploma in **Computer Engineering** of Institute

**M. H. Saboo Siddik Polytechnic** (Code **: 0002**) has completed their term work satisfactorily in course **Database Management System(22319)** for the academic year 2020 to 2021 as prescribed in the curriculum.

Place : Mumbai Date : .....

Exam. Seat No : .....

Subject Teacher. Head of the Department. Principal

**Seal of Institute**

**(NAME AND SIGNATURE OF FACULTY)**

## ANNEXURE

**Evaluation sheet for the micro-project.**

**Academic year: 2020-21.** **Name of faculty: Mohammad Ali Sir.**

**Course: DMS.** **Course code: 22319.** **Semester: III.**

**Title of the project:**

**CO's addressed by the micro-project:**

**A.**_____

**B.**_____

**Major learning outcomes achieved by the students by doing the project :**

**(a) Practical Outcomes:**

_____

_____

_____

**(b) Unit outcomes in cognitive domain:**

_____

_____

_____

**(c) Outcomes in effective domain:**

_____

_____

_____

**Comments/ suggestions about teamwork/ leadership/ inter-personal communication (if any):**

| Roll No: | Student Name | Marks out of 6 for performance in group activity | Marks out of 4 for performance in oral or presentation | Total out of 10 |
|---|---|---|---|---|
| 19401 | Ansari Anam | | | |
| 19416 | Loladia Ayesha | | | |
| 19420 | Rakhangi Arisha | | | |
| 19428 | Shaikh Nusra | | | |

# NORMALIZATION IN DBMS

**Normalization** is the process of organizing the data in the database. Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization divides the larger table into the smaller table and links them using relationship. The words normalization and normal form refer to the structure of a database.

The normal form is used to reduce redundancy from the database table. Normalization also simplifies the database design so that it achieves the optimal structure composed of atomic elements (i.e. elements that cannot be broken down into smaller parts). By normalizing a database, you arrange the data into tables and columns. You ensure that each table contains only related data. If data is not directly related, you create a new table for that data.

The concept of normalization was first proposed by Edgar F. Codd in 1970, when he proposed the first normal form (1NF) in his paper *A Relational Model of Data for Large Shared Data Banks* (this is the paper in which he introduced the whole idea of relational databases).

Codd continued his work on normalization and defined the second normal form (2NF) and third normal form (3NF) in 1971.Codd then teamed up with Raymond F. Boyce to define the Boyce-Codd normal form (BCNF) in 1974.Ronald Fagin introduced the fourth normal form (4NF) in 1977.Fagin then introduced the fifth normal form (5NF) in 1979.

## Benefits of Normalization

- Normalization increases clarity in organizing data in Databases.
- Minimizes data redundancy (duplicate data).
- Minimizes null values.
- Results in a more compact database (due to less data redundancy/null values).
- Minimizes/avoids data modification issues.
- The database structure is cleaner and easier to understand.
- Searching, sorting, and creating indexes can be faster, since tables are narrower, and more rows fit on a data page.

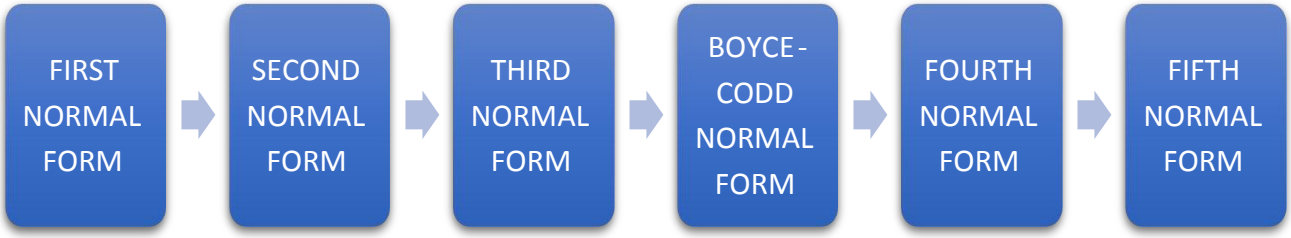
# NEED OF NORMALIZATION

- To maintain data integrity
- To eliminate redundant data
- To eliminate data duplicacy
- To increase storage efficiency
- To produce a clear and understandable/readable database structure.
- To avoid anomalies(Update/delete/insert)

If database is not normalized, then insertion, updation and deletion anomalies occur. Anomalies are error-prone situation arising when we process the tables. There are three types of anomalies :

1) Insert Anomaly: An insert anomaly occurs when it is not possible to insert certain attributes into the database without the availability of other attributes. i.e We tried to insert data in a record that does not exist at all.
2) Update anomaly : If data items are scattered and are not linked to each other properly, then it could lead to ambiguos situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state. This is called Update anomaly.
3) Delete Anomaly : This anomaly occurs when data of some attributes get lost because of deletion of data of other attributes in the same record.

# TYPES OF NORMAL FORM



# FUNCTIONAL DEPENDENCY :

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, A→B, where A functionally determines B. The left-hand side attributes determine the values of attributes on the right-hand side.

The left-hand side of the FD is called the determinant, and the righthand side is the dependent. If A is the determinant and B is the dependent then we say that B is dependent on A and A functionally determines B.

A functional dependency A->B in a relation holds if two tuples having same value of attribute A also have same value for attribute B.

EG.1) In country_id table country_name is functionally dependent on c_id (i.e country id)

```
SQL> select * from country_id;

      C_ID COUNTRY_NAME
---------- ---------------
        11 USA
        25 India
        50 France
        13 Brazil
        45 Netherlands
        84 Japan
        35 South Korea

7 rows selected.
```

EG.2)In vaccine_creation_info table all the attributes are functionally dependent on primary key i.e vac_id

```
SQL> set linesize 150
SQL> select * from vaccine_creation_info;

   VACC_ID VACC_NAME                    LABORATORY_NAME              MANUFACTURER                TESTING_D STATUS     TESTING_SUBJECT
---------- ---------------------------- ---------------------------- --------------------------- --------- ---------- --------------------
         1 Pneumococcal vaccine         clinical management of SARI  gen target inc              18-DEC-19 unsuccessful   hamster
         2 hemoephilus influenzae       THSTI                        pharma lonza                11-JAN-20 unsuccessful   pig
         3 pneumovax 23                 CNRS research laboratory     eurofins                    25-FEB-20 successful     rabbit
         4 pertusis                     LDV USP                      astrazeneca                 13-APR-20 successful     rat
         5 anthrax                      applikon biotechnology       virtuvax                    20-MAY-20 unsuccessful   lizard
         6 japanese anciphalitis        kakogawa laboratories        barr labs inc               16-JUL-20 unsuccessful   dog
         7 SK flu shot                  gwangju institute of medicine dynavax technologies       28-SEP-20 unsuccessful   human

7 rows selected.
```

EG.3) In sid_india table state_name is functionally dependent on primary key i.e s_id (state id)

```
SQL> select * from sid_india;

      S_ID STATE_NAME
---------- ---------------
        11 maharashtra
        23 madhya pradesh
        41 tamil nadu
        39 gujarat
        26 rajasthan
        31 punjab
        91 kerala

7 rows selected.
```

# PARTIAL DEPENDENCY :

Partial Dependency occurs when a non-prime attribute is functionally dependent on part of a candidate key. i.e B is functionally dependent on A, and A is the part of multipart candidate key.

Eg. If there is a table with following attributes :

| C_id | country_name | Vac_id | Vacc_name |
|---|---|---|---|

In this case, country_name depends on c_id and vacc_name depends on vac_id. That means there exist a partial dependency in this table.

To remove this partial dependency we split the table as :

| c_id | Country_name | Vac_id |
|---|---|---|

| vacc_id | vacc_name |
|---|---|

# TRANSITIVE DEPENDENCY :

Consider attributes A, B, C.

If A functionally determines B (A->B) and B functionally Determines C (B->C) then there exist a transitive dependency i.e C is transitively dependent on A through B (A->C).

# FIRST NORMAL FORM (1NF) :

According to first normal form, all the attributes in a relation must have atomic domains.

A domain is the set of all unique values which is permitted for an attribute. Atomic means that cannot be divided further.

## RULES FOR FIRST NF:

- A relation will be in 1NF if it contain atomic values.
- an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

EG.1) In countries table all the attributes are singlevalued and have atomic values

```
SQL> select * from countries;

      C_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------ ---------- ----------- ---------- ----------
        11   10205818      3517817        19480    6441744   156553957     245257  331690410
        25    8530754       532624         9944    7868968   116542304     128162 1384789754
        50    1771705      1600598         5421     127938    17651546      42169   65324901
        13    5676561       446931         9318    5064344    21900000     164286  213094038
        45     427401        22000         1630     300205     3535613       9690   17148361
        84     128941        27644         1194      96461     2846368       3809  126337911
        35      50427        21981         1058      24968     2702882       2478   51284907

7 rows selected.

SQL>
```

EG.2)In sid_brazil table all the attributes are singlevalued and have atomic values

```
L> select * from sid_brazil;

    S_ID STATE_NAME
-------- ---------------
      13 santa catarina
      25 bahia
      43 rio de janeiro
      41 mato grosso
      28 parana
      33 tocantis
      93 sao paulo

rows selected.

L>
```

EG.3)In vaccine_creation_info table all the attributes are single-valued and have atomic values

```
SQL> set linesize 150
SQL> select * from vaccine_creation_info;

   VACC_ID VACC_NAME                    LABORATORY_NAME               MANUFACTURER                TESTING_D STATUS     TESTING_SUBJECT
---------- ---------------------------- ----------------------------- --------------------------- --------- ---------- --------------------
         1 Pneumococcal vaccine         clinical management of SARI   gen target inc              18-DEC-19 unsuccessful   hamster
         2 hemoephilus influenzae       THSTI                         pharma lonza                11-JAN-20 unsuccessful   pig
         3 pneumovax 23                 CNRS research laboratory      eurofins                    25-FEB-20 successful     rabbit
         4 pertusis                     LDV USP                       astrazeneca                 13-APR-20 successful     rat
         5 anthrax                      applikon biotechnology        virtuvax                    20-MAY-20 unsuccessful   lizard
         6 japanese anciphalitis        kakogawa laboratories         barr labs inc               16-JUL-20 unsuccessful   dog
         7 SK flu shot                  gwangju institute of medicine dynavax technologies        28-SEP-20 unsuccessful   human

7 rows selected.
```

Similarly, all the tables in our database are single-valued and have atomic values.

# SECOND NORMAL FORM (2NF) :

For a relation to be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency, i.e all the non-prime attributes must be dependent on prime attributes (primary key).

- **Prime attribute** – An attribute, which is a part of the prime-key, is known as a prime attribute.
- **Non-prime attribute** – An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

## RULES FOR SECOND NF:

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key ➢ Partial dependency should not exist.

EG.1) According to second NF, all the non-prime attributes must be dependent on prime-key attributes.

In the following table non-prime attributes like cases, active_cases, serious_cases, recovered, total_tests, deaths , population are dependent prime-key attribute i.e s_id.

```
SQL> select * from brazil_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        13     593702        38586         28975     490676     5875659      50796    6091000
        25    1376870       429654        355820     296757     1965927      84650   14561000
        43     389548       168900         39690     298567     4968580      59600    5857904
        41    1965800        84650         49850    1295030     1987540      85470    2449024
        28     195963        49690         29675     128000      200630      40780     247863
        33    1005092        59386         39689     296785      986870       5000    1496880
        93     982648        73960         40600     887576     1968755       9032   11967825

7 rows selected.
```

✓ But in case of weak entity non-prime attributes are dependent on foreign key

EG.2) All the non-prime attributes like vacc_name,laboratory_name, manufacturer, testing_date, status, testing_subject are dependent on primary key i.e vacc_id

```
SQL> select * from vaccine_creation_info;

  VACC_ID VACC_NAME                    LABORATORY_NAME              MANUFACTURER              TESTING_D STATUS     TESTING_SUBJECT
---------- ---------------------------- ---------------------------- ---------------------------- --------- -------------- --------------------
         1 Pneumococcal vaccine         clinical management of SARI  gen target inc               18-DEC-19 unsuccessful   hamster
         2 hemoephilus influenzae       THSTI                        pharma lonza                 11-JAN-20 unsuccessful   pig
         3 pneumovax 23                 CNRS research laboratory     eurofins                     25-FEB-20 successful     rabbit
         4 pertusis                     LDV USP                      astrazeneca                  13-APR-20 successful     rat
         5 anthrax                      applikon biotechnology       virtuvax                     20-MAY-20 unsuccessful   lizard
         6 japanese anciphalitis        kakogawa laboratories        barr labs inc                16-JUL-20 unsuccessful   dog
         7 SK flu shot                  gwangju institute of medicine dynavax technologies        28-SEP-20 unsuccessful   human

7 rows selected.
```

Similarly, all the tables in the database are in second NF without any partial dependency.

# THIRD NORMAL FORM (3NF) :

A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes as well as it is in second normal form. 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.

## RULES FOR THIRD NF:

- A relation will be in 3NF if it is in 2NF.
- It should not contain any transitive dependency.

EG.1) The following table is in second NF as well as there is no transitive dependency. i.e the table is in 3NF.

```
SQL> select * from countries;

      C_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        11   10205818      3517817         19480    6441744   156553957     245257  331690410
        25    8530754       532624          9944    7868968   116542304     128162 1384789754
        50    1771705      1600598          5421     127938    17651546      42169   65324901
        13    5676561       446931          9318    5064344    21900000     164286  213094038
        45     427401        22000          1630     300205     3535613       9690   17148361
        84     128941        27644          1194      96461     2846368       3809  126337911
        35      50427        21981          1058      24968     2702882       2478   51284907

7 rows selected.
```

```
SQL> select * from brazil_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        13     593702        38586         28975     490676     5875659      50796    6091000
        25    1376870       429654        355820     296757     1965927      84650   14561000
        43     389548       168900         39690     298567     4968580      59600    5857904
        41    1965800        84650         49850    1295030     1987540      85470    2449024
        28     195963        49690         29675     128000      200630      40780     247863
        33    1005092        59386         39689     296785      986870       5000    1496880
        93     982648        73960         40600     887576     1968755       9032   11967825

7 rows selected.

SQL> select * from JAPAN_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        15     491087        29461         19460     390572      101048       4528    1957914
        27      15372         9845          3061       9994       18043        583      17297
        45    3981057       291402        100921     308193    30689150       4890   43300000
        43     936801       492685         29750    6092605      200920       5920   23010276
        30     209875        49028         28964     190589      300719       2459     335767
        35     983170        39086         10948     689305    20087201       6985   22757897
        95     983070        47808         10938    6098302    69379287       3090    7563428

7 rows selected.
```

```
SQL> select * from vaccine_creation_info;

   VACC_ID VACC_NAME                 LABORATORY_NAME             MANUFACTURER              TESTING_D STATUS     TESTING_SUBJECT
---------- ------------------------- --------------------------- ------------------------- --------- ---------- -------------------
         1 Pneumococcal vaccine      clinical management of SARI gen target inc            18-DEC-19 unsuccessful hamster
         2 hemoephilus influenzae    THSTI                       pharma lonza              11-JAN-20 unsuccessful pig
         3 pneumovax 23              CNRS research laboratory    eurofins                  25-FEB-20 successful   rabbit
         4 pertusis                  LDV USP                     astrazeneca               13-APR-20 successful   rat
         5 anthrax                   applikon biotechnology      virtuvax                  20-MAY-20 unsuccessful lizard
         6 japanese anciphalitis     kakogawa laboratories       barr labs inc             16-JUL-20 unsuccessful dog
         7 SK flu shot               gwangju institute of medicine dynavax technologies    28-SEP-20 unsuccessful human

7 rows selected.
```

# MANY-TO-MANY RELATIONSHIP :

This is a very common situation , it occurs when the table has only foreign-keys. If we add further columns in future there will be no problem due to the existing columns all being foreign keys.

EG.) The following table consist of only foreign-keys. Therefore, there exist a many-to-many relationship.

```
SQL> select * from link_vaccid_cid;

   VACC_ID       C_ID
---------- ----------
         1         11
         2         25
         3         50
         4         13
         5         45
         6         84
         7         35

7 rows selected.
```

# ER Diagram in DBMS

## What is an Entity Relationship Diagram (ERD)?

**ER Diagram** stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases.

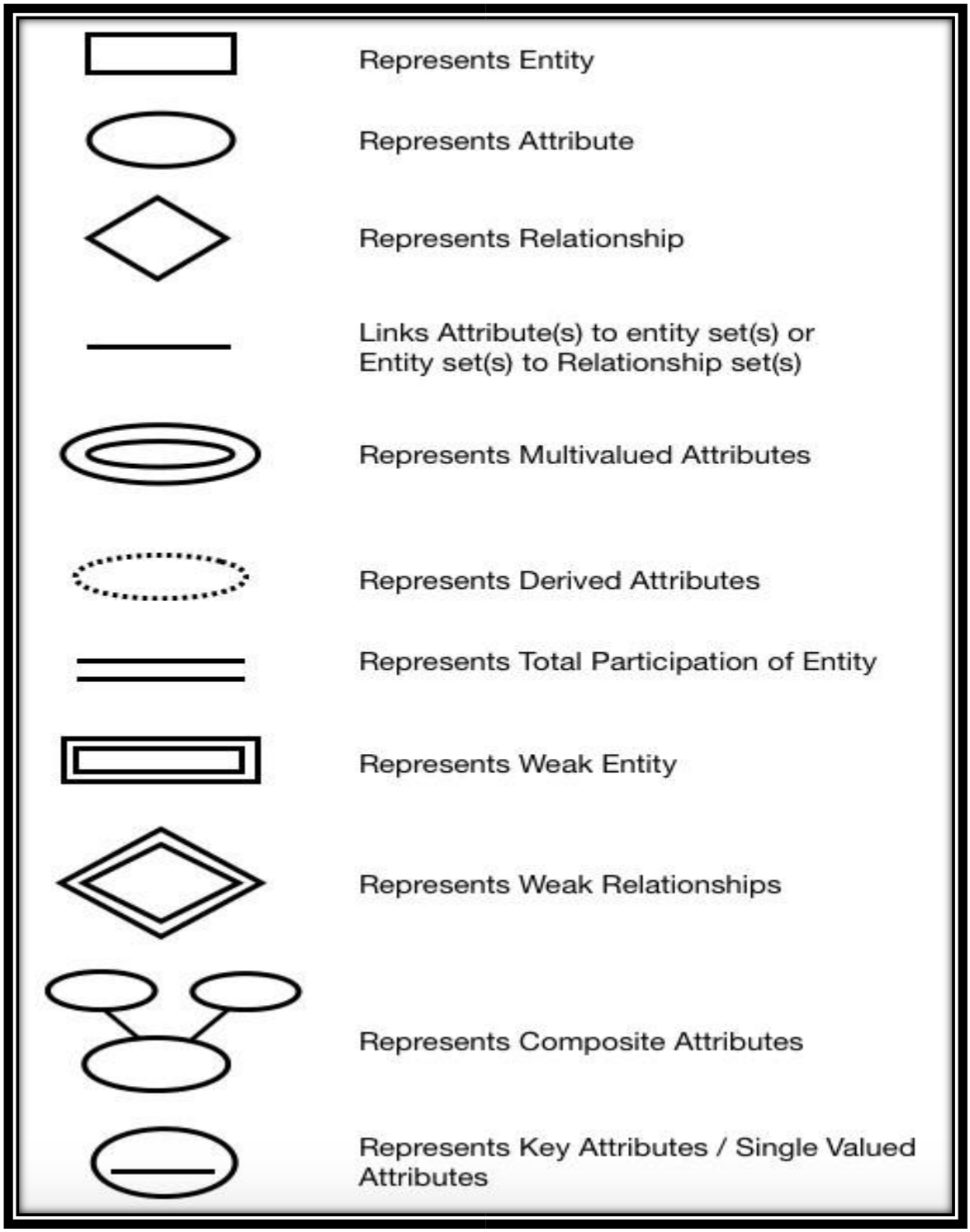
**ER diagrams are created based on three basic concepts: entities, attributes and relationships.**

ER diagrams are used to sketch out the design of a database.

## Why use ER Diagrams?

- Helps you to define terms related to entity relationship modeling
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- Helps to describe entities, attributes, relationships
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of ERP diagram
- ERD Diagram allows you to communicate with the logical structure of the database to users.

Following are the main components and its symbols in ER Diagrams:



# Types of DBMS Entities

The following are the types of entities in DBMS −

## Strong Entity

The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity.Strong Entity is represented by a single rectangle.

## Weak Entity

The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities. Weak Entity is represented by double rectangle.

# Mapping Constraints

A mapping constraint is a data constraint that expresses the number of entities to which another entity can be related via a relationship set.

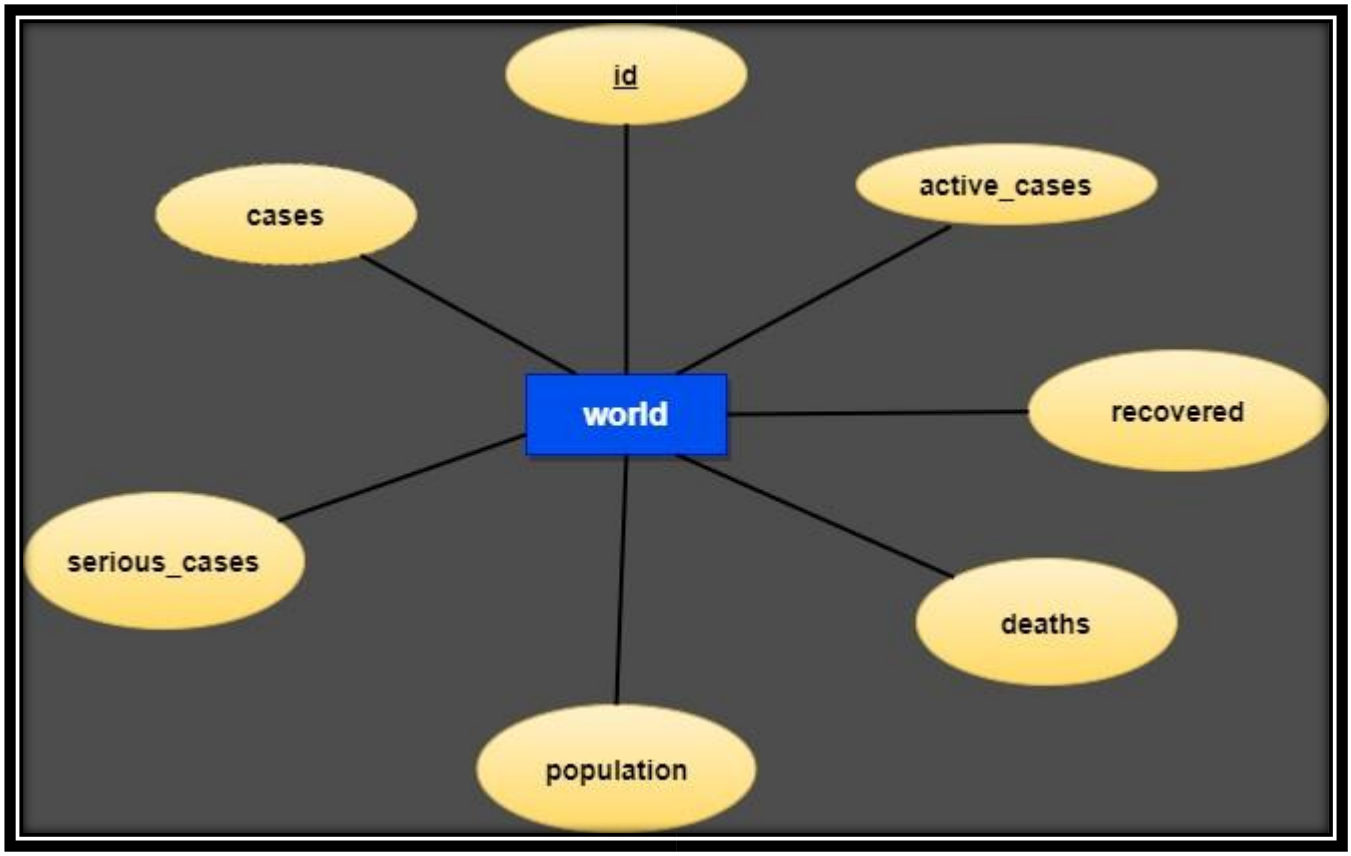
It is most useful in describing the relationship sets that involve more than two entity sets.

For binary relationship, there are four possible mapping cardinalities. These are as follows:

1. One to one (1:1)
2. One to many (1:M)
3. Many to one (M:1)
4. Many to many (M:M)

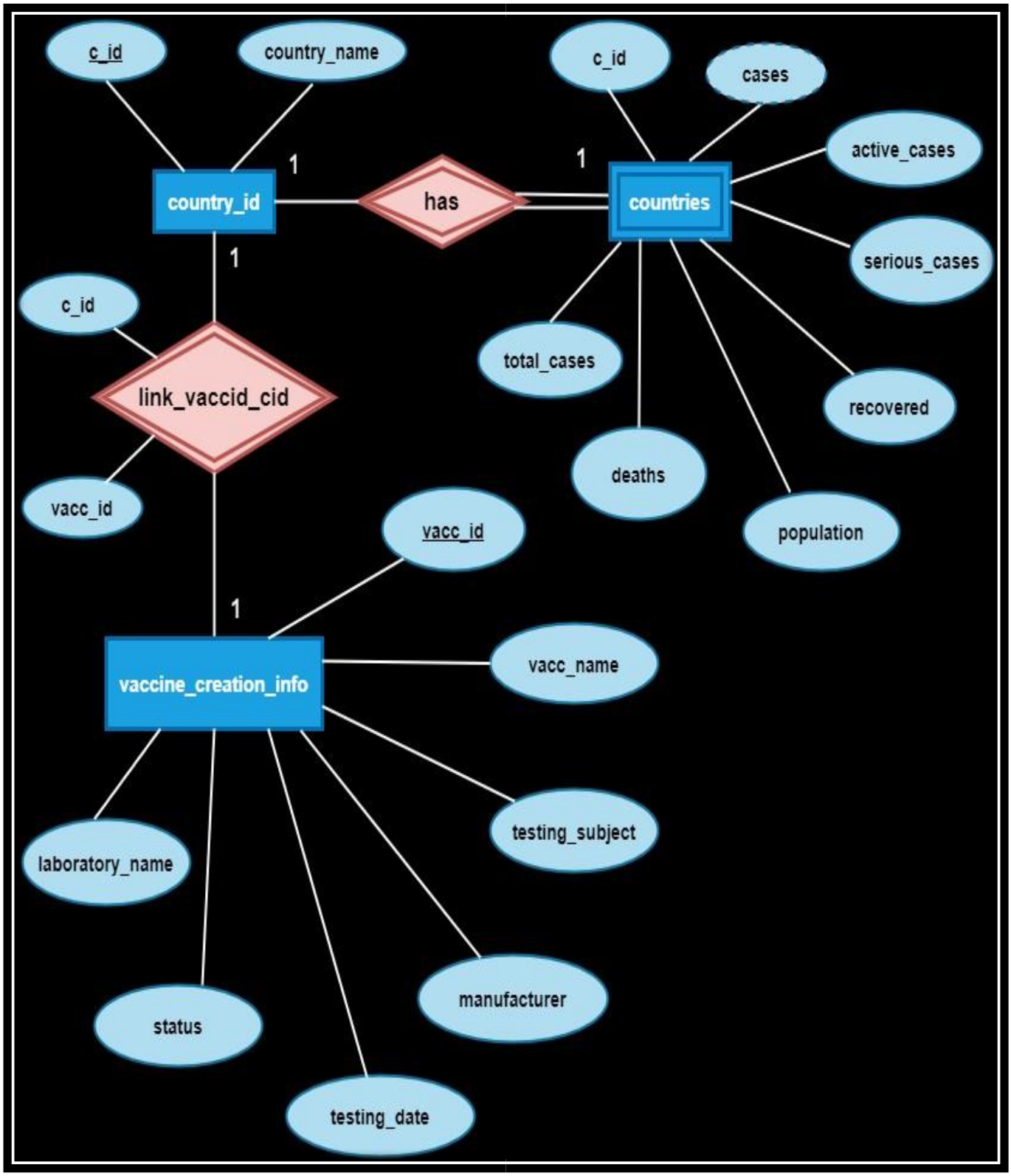
# ERD OF CORONAVIRUS DBMS

## ERD OF WORLD TABLE :




This  is a strong entity table because it has a primary key which is the id attribute. There are 7 attributes in this world table(entity). This erd has no relationship in it. It has only one row in it , which shows the record of the whole world in a single row.

# ERD OF COUNTRY_ID, COUNTRIES & VACCINE_CREATION_INFO TABLES :



## There are 3 entities in this erd .

### 1) country_id :

This is a strong entity with 2 attributes , c_id is the primary key over here.

### 2) countries :

This is a weak entity with 8 attributes, c_id is the foreign key here and cases is the derived attribute which is calculated using remaining attributes except population and c_id.

### 3) vaccine_creation_info :

This is a strong entity with 7 attributes , vacc_id is the primary key over here.

### This erd has 2 relatiosnhips :

1) A relationship (has) is a relationship between the country_id entity and countries entity with a common attribute as c_id .

Here one entity is a strong entity which is country_id , because it has its own primary key but the another entity which is countries is a weak entity because it does not has any primary key but it has a foreign to link with the strong entity i.e country_id. So the here the relationship ( has ) is a identifying relationship.

It is a one to one relationship because one record of country_id table is associated with only 1 record of countries table and vice versa

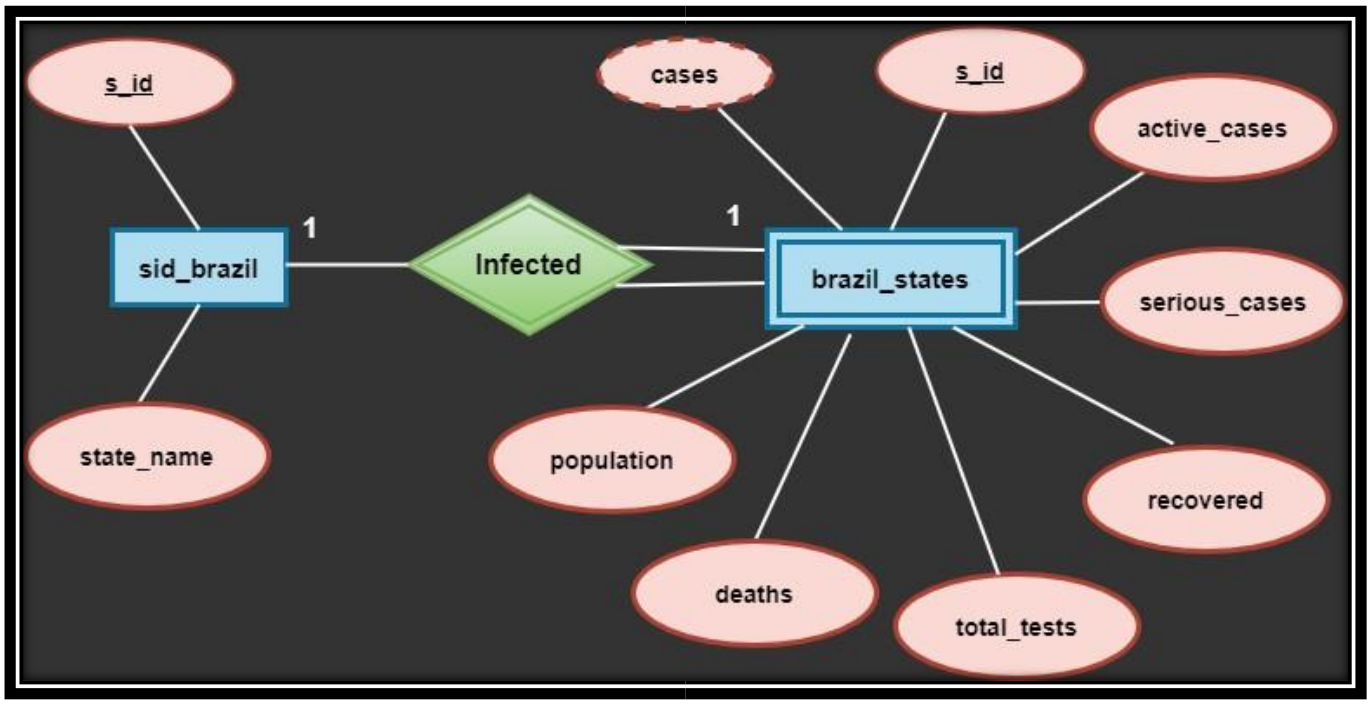
2) A relationship (link_vaccid_cid) is a relationship between the country_id entity and vaccine_creation_info entity .

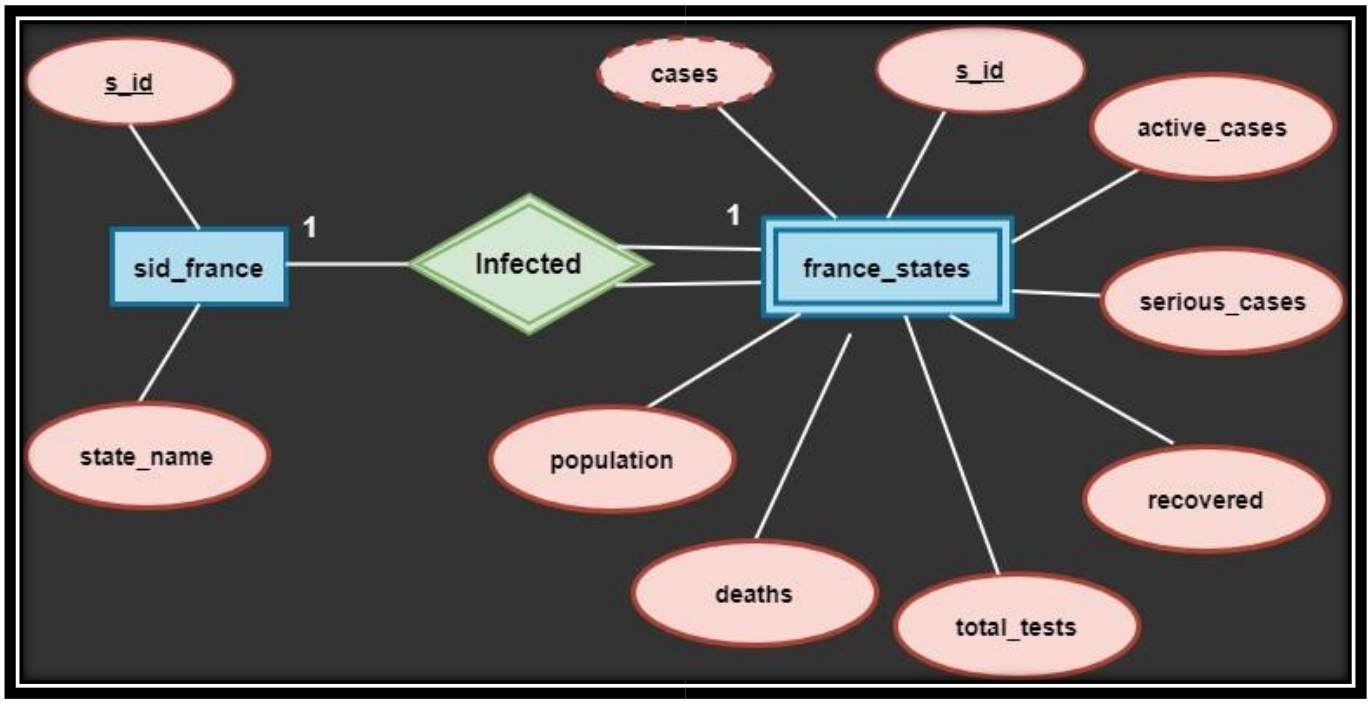
This link_vaccid_cid relationship has two attributes , 1 each from both the entities.The attributes of these relationship are foreign key in this relationship and primary key of their respective tables. Because it does not have any primary key, the relationship becomes a weak relationship , but with a 2 strong entities.

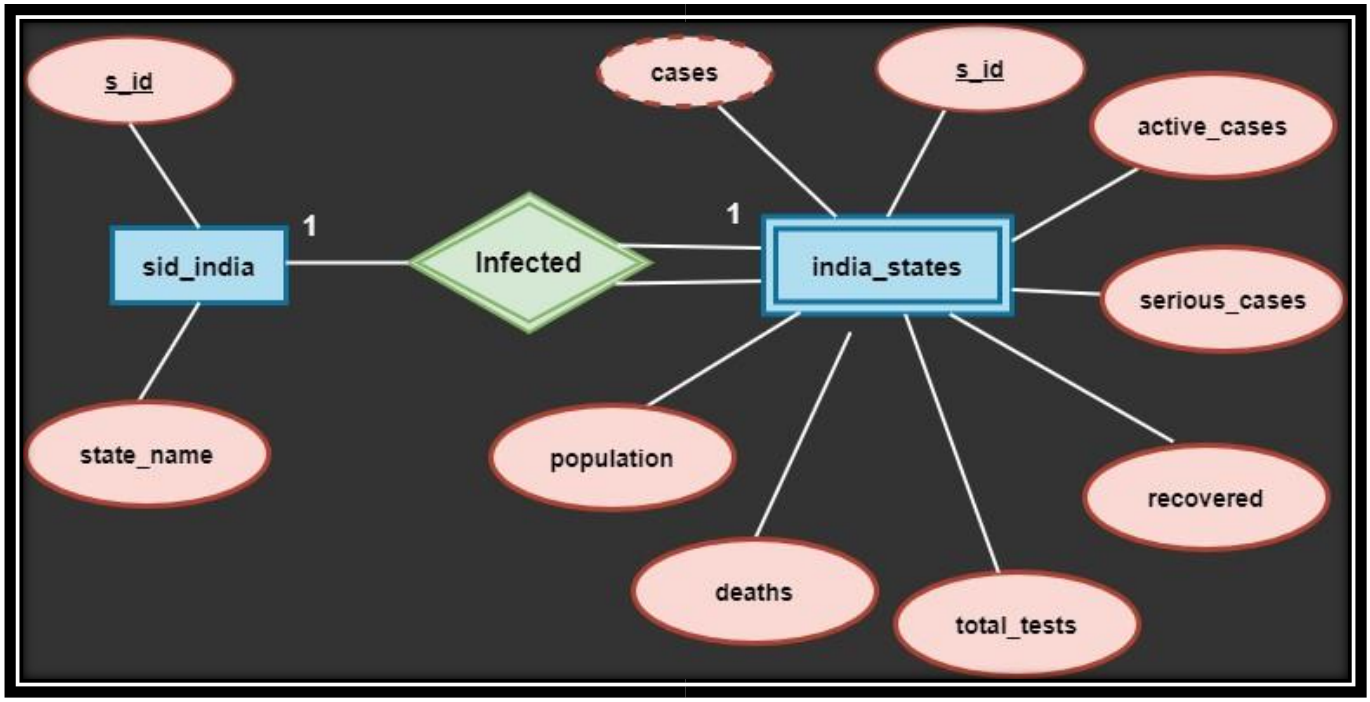
It is a one to one relationship because one record of country_id table is associated with only 1 record of vaccine_creation_info table and vice versa.

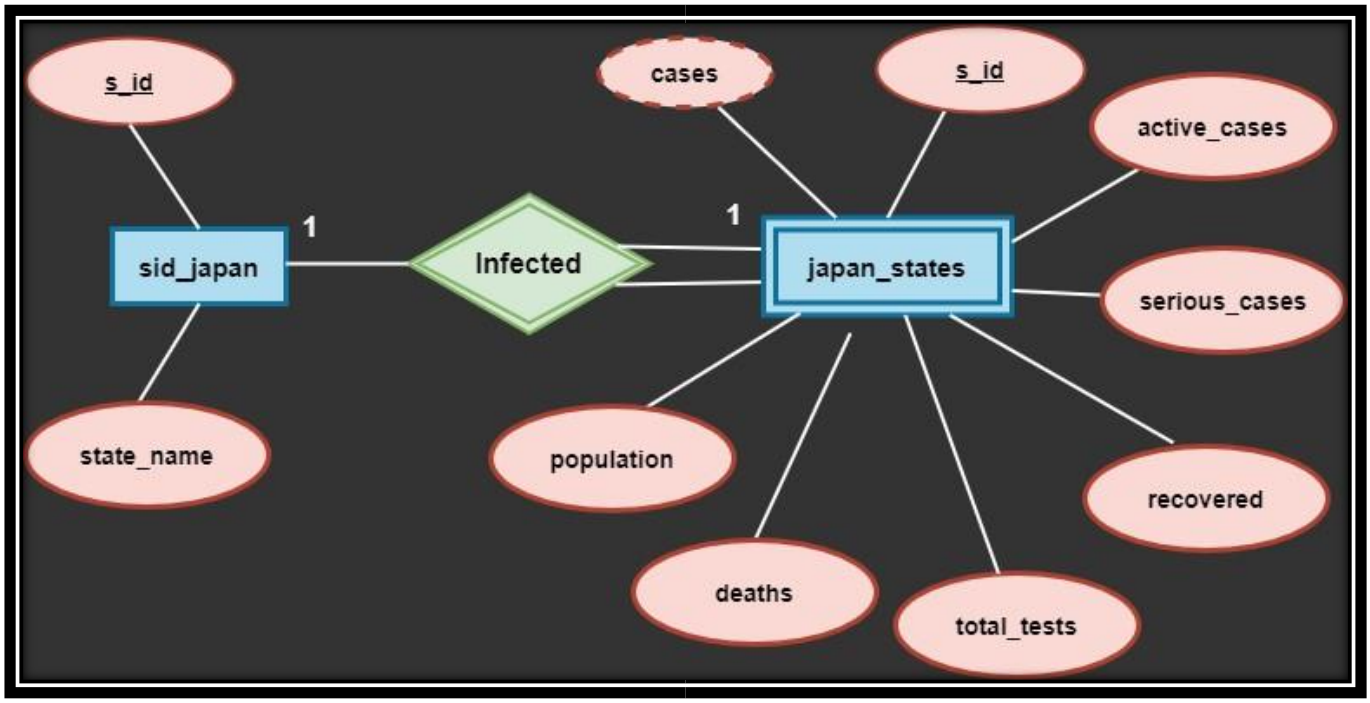
But this can be a many to many relationship if we add more records in it , because
1 vaccine can be supplied to many countries and 1 country can even take more than 1 vaccine.

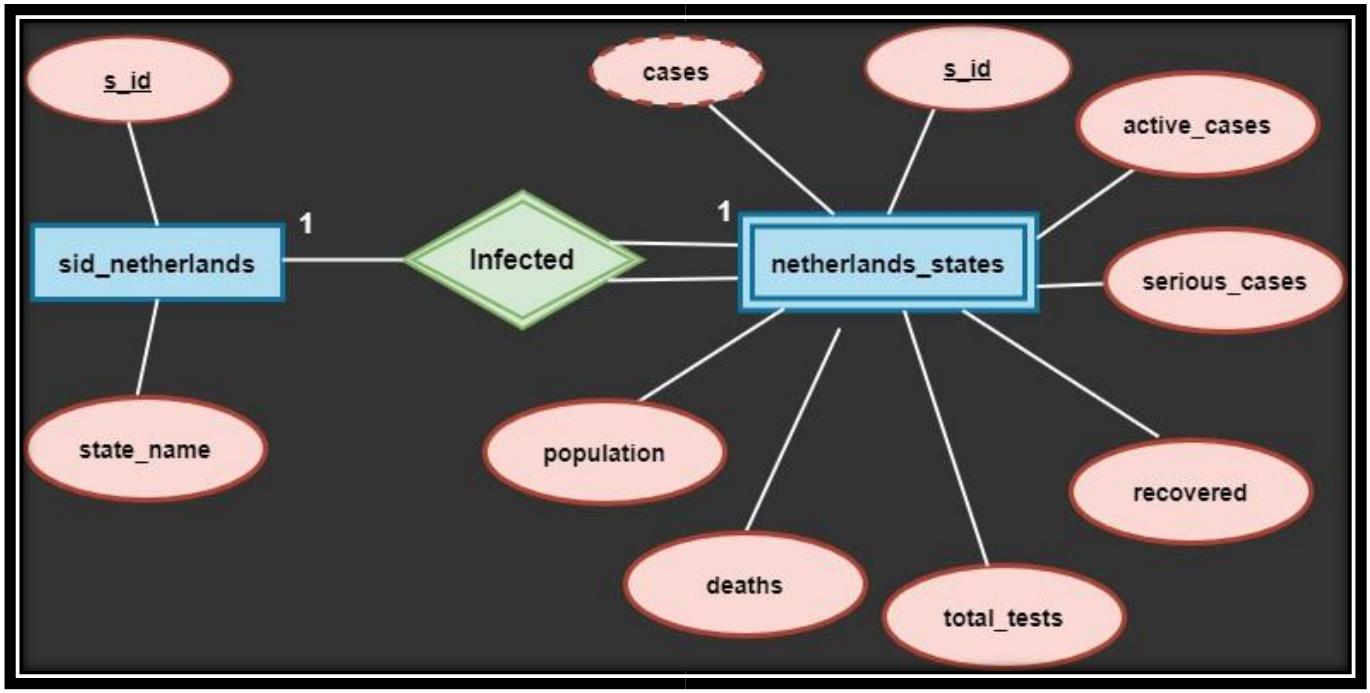
# ERD OF PARTICULAR COUNTRIES WITH THEIR RESPECTIVE STATE RECORDS.

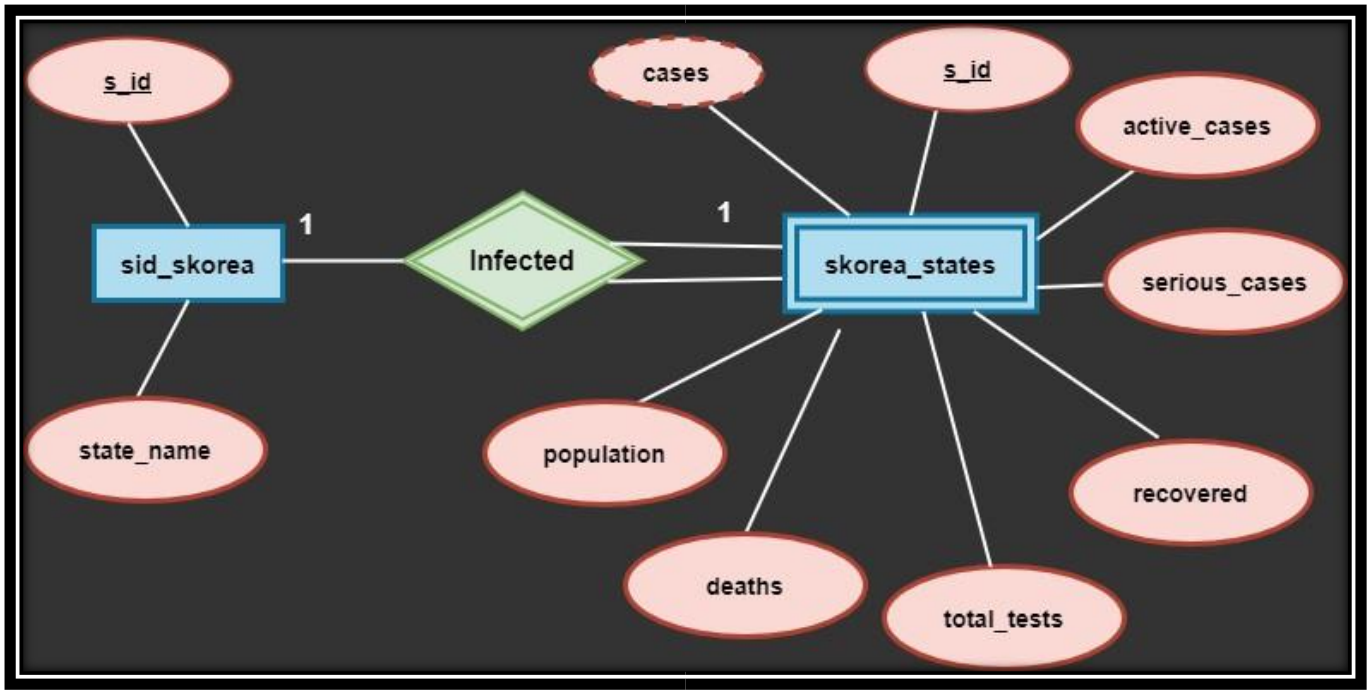


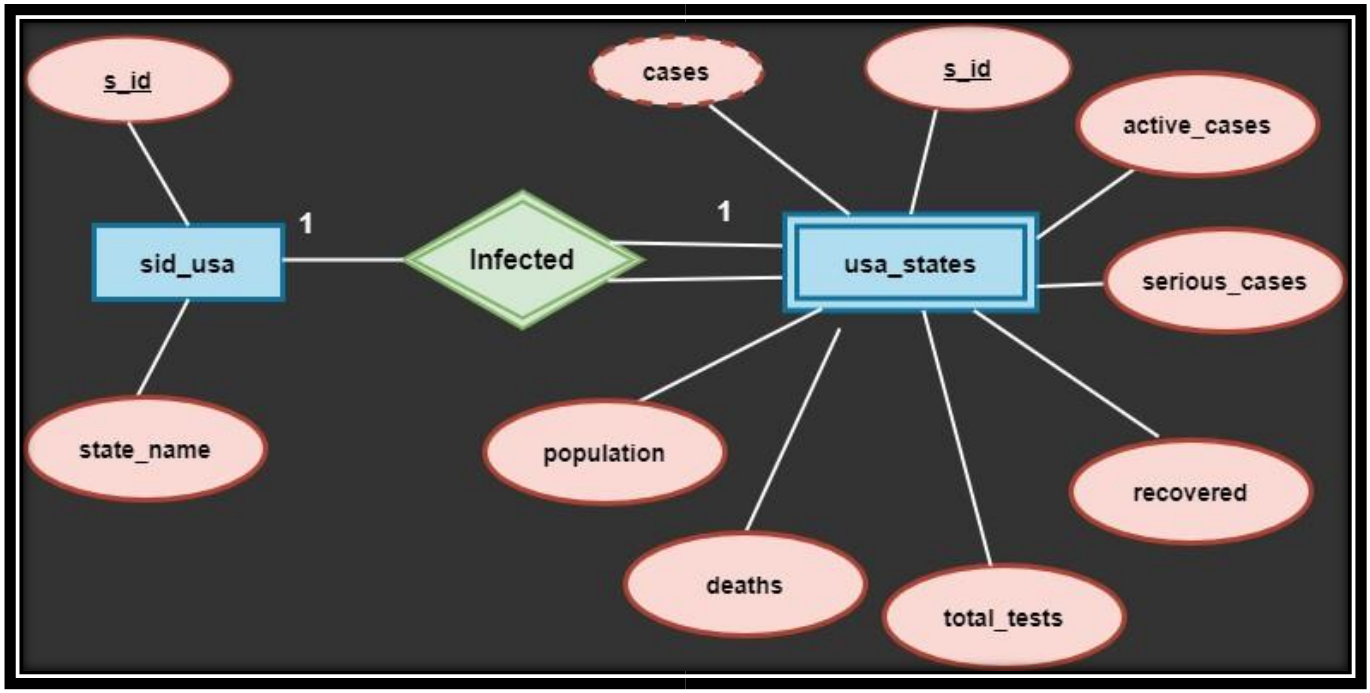




s_id
cases
s_id
active_cases
1
1
sid_india
Infected
india_states
serious_cases
state_name
population
recovered
deaths
total_tests


s_id
cases
s_id
active_cases
1
1
sid_japan
Infected
japan_states
serious_cases
state_name
population
recovered
deaths
total_tests


s_id
cases
s_id
active_cases
1
1
sid_netherlands
Infected
netherlands_states
serious_cases
state_name
population
recovered
deaths
total_tests


s_id
cases
s_id
active_cases
1
1
sid_skorea
Infected
skorea_states
serious_cases
state_name
population
recovered
deaths
total_tests




These ERD shows the records of a particular country along with their infected states.

All these ERD has 2 entites , one is strong entity which contains 2 attributes with 1 primary in them and the other entity is a weak entity , because it does not have any own primary but has a foreign key which is dependent on the strong entity  The relationship (infected) between these entities is a identifying relationship.

It is a one to one relationship because, one record of a table  is associated with only 1 record of another table and vice versa.

# CREATING A NEW USER

- A new user named as dbms_mp is created in sql command window using the command 'create user' .
- This new user is identified by password as 'dobby'. since security of database is essential.
- The whole database containing the information will be created inside this new user.
- We have granted all the privileges to this new user.

```
SQL> connect system
Enter password:
Connected.
SQL> create user dbms_mp identified by dobby;

User created.

SQL> grant all privileges to dbms_mp;

Grant succeeded.
```

- This is how we can connect to the user 'dbms_mp'.

```
SQL> connect dbms_mp
Enter password:
Connected.
SQL> _
```

We have the following tables in our database.

```
SQL> select table_name from user_tables;

TABLE_NAME
--------------------------------
COUNTRY_ID
COUNTRIES
SID_USA
USA_STATES
SID_INDIA
INDIA_STATES
SID_FRANCE
FRANCE_STATES
SID_BRAZIL
BRAZIL_STATES
SID_NETHERLANDS
NETHERLANDS_STATES
SID_JAPAN
JAPAN_STATES
SID_SKOREA
SKOREA_STATES
VACCINE_CREATION_INFO
LINK_VACCID_CID
WORLD

19 rows selected.
```

# DATABASE SCHEMA

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated.

**Our database consists of the following:**

- Describing Structure Of World Table.

```
SQL> desc world;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER(1)
 CASES                                              NUMBER(15)
 ACTIVE_CASES                                       NUMBER(15)
 SERIOUS_CASES                                      NUMBER(15)
 DEATHS                                             NUMBER(15)
 RECOVERED                                          NUMBER(15)
 POPULATION                                         NUMBER(20)
```

```
SQL> select * from world;

        ID      CASES ACTIVE_CASES SERIOUS_CASES     DEATHS  RECOVERED POPULATION
---------- ---------- ------------ ------------- ---------- ---------- ----------
         1   63641185     18137719        106010    1474981   44028485 7829114600
```

- Describing structure of country_id.

```
SQL> desc country_id;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 C_ID                                      NOT NULL NUMBER(2)
 COUNTRY_NAME                                       VARCHAR2(15)
```

```
SQL> select * from country_id;

      C_ID COUNTRY_NAME
---------- ---------------
        11 USA
        25 India
        50 France
        13 Brazil
        45 Netherlands
        84 Japan
        35 South Korea

7 rows selected.
```

- Describing structure of countries.

```
SQL> desc countries;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 C_ID                                               NUMBER(2)
 CASES                                              NUMBER(10)
 ACTIVE_CASES                                       NUMBER(10)
 SERIOUS_CASES                                      NUMBER(10)
 RECOVERED                                          NUMBER(10)
 TOTAL_TESTS                                        NUMBER(15)
 DEATHS                                             NUMBER(10)
 POPULATION                                         NUMBER(15)
```

```
SQL> select * from countries;

      C_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        11   10182818      3497817         18480    6441744   156553957     243257  331690410
        25    8507754       512624          8944    7868968   116542304     126162 1384789754
        50    1748705      1580598          4421     127938    17651546      40169   65324901
        13    5653561       426931          8318    5064344    21900000     162286  213094038
        45     404401         2000           630     300205     3535613       7690   17148361
        84     105941         7644           194      96461     2846368       1809  126337911
        35      27427         1981            58      24968     2702882        478   51284907

7 rows selected.
```

- Describing structure of vaccine_creation_info.

```
SQL> desc vaccine_creation_info;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 VACC_ID                                   NOT NULL NUMBER(1)
 VACC_NAME                                          VARCHAR2(30)
 LABORATORY_NAME                                    VARCHAR2(30)
 MANUFACTURER                                       VARCHAR2(30)
 TESTING_DATE                                       DATE
 STATUS                                             VARCHAR2(15)
 TESTING_SUBJECT                                    VARCHAR2(20)
```

```
SQL> select * from vaccine_creation_info;

  VACC_ID VACC_NAME                 LABORATORY_NAME              MANUFACTURER              TESTING_D STATUS       TESTING_SUBJECT
--------- ------------------------- ---------------------------- ------------------------- --------- ------------ ---------------
        1 Pneumococcal vaccine      clinical management of SARI  gen target inc            18-DEC-19 unsuccessful hamster
        2 hemoephilus influenzae    THSTI                        pharma lonza              11-JAN-20 unsuccessful pig
        3 pneumovax 23              CNRS research laboratory     eurofins                  25-FEB-20 successful   rabbit
        4 pertusis                  LDV USP                      astrazeneca               13-APR-20 successful   rat
        5 anthrax                   applikon biotechnology       virtuvax                  20-MAY-20 unsuccessful lizard
        6 japanese anciphalitis     kakogawa laboratories        barr labs inc             16-JUL-20 unsuccessful dog
        7 SK flu shot               gwangju institute of medicine dynavax technologies     28-SEP-20 unsuccessful human

7 rows selected.
```

- Describing structure of link_vaccid_cid.

```
SQL> desc link_vaccid_cid;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 VACC_ID                                            NUMBER(1)
 C_ID                                               NUMBER(2)
```

```
SQL> select * from link_vaccid_cid;

   VACC_ID       C_ID
---------- ----------
         1         11
         2         25
         3         50
         4         13
         5         45
         6         84
         7         35

7 rows selected.
```

- Describing structure of sid_usa.

```
SQL> desc sid_USA;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 S_ID                                      NOT NULL NUMBER(2)
 STATE_NAME                                         VARCHAR2(15)
```

```
SQL> select * from sid_USA;

      S_ID STATE_NAME
---------- ---------------
        10 Texas
        22 California
        40 Florida
        38 New York
        25 Arizona
        30 Pennsylvania
        90 Virginia

7 rows selected.
```

- Describing structure of usa_states.

```
SQL> desc USA_states;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 S_ID                                               NUMBER(2)
 CASES                                              NUMBER(10)
 ACTIVE_CASES                                       NUMBER(10)
 SERIOUS_CASES                                      NUMBER(10)
 RECOVERED                                          NUMBER(10)
 TOTAL_TESTS                                        NUMBER(15)
 DEATHS                                             NUMBER(10)
 POPULATION                                         NUMBER(15)
```

```
SQL> select * from USA_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        10    1012428       163723         82000     829458     9547430      19247   28995881
        22     970392       458159         24559     494270    19565151      17963   39512223
        40     837077       226127         38974     593853    10453650      17100   21477737
        38     562365       104023         28908     424541    15519172      33801   19453561
        25     257384       208287         35361      42950     2183128       6147    7278717
        30     232582        54778         18168     168708     2897339       9096   12801989
        90     190873       165728         22362      21441     2986354       2704    8535519

7 rows selected.
```

- Describing structure of sid_india.

```
SQL> desc sid_india;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 S_ID                                      NOT NULL NUMBER(2)
 STATE_NAME                                         VARCHAR2(15)
```

```
SQL> select * from sid_india;

      S_ID STATE_NAME
---------- ---------------
        11 maharashtra
        23 madhya pradesh
        41 tamil nadu
        39 gujarat
        26 rajasthan
        31 punjab
        91 kerala

7 rows selected.
```

- Describing structure of india_states.

```
SQL> desc india_states;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ----------------
 S_ID                                                           NUMBER(2)
 CASES                                                          NUMBER(10)
 ACTIVE_CASES                                                   NUMBER(10)
 SERIOUS_CASES                                                  NUMBER(10)
 RECOVERED                                                      NUMBER(10)
 TOTAL_TESTS                                                    NUMBER(15)
 DEATHS                                                         NUMBER(10)
 POPULATION                                                     NUMBER(15)
```

```
SQL> select * from india_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        11     211987        85725         42300      14209   110240153       4938  112372972
        23     178298        31050         20845     167681    45352090       3034   85358965
        41     746079        29080          4095     715892    50923530      11362   72147039
        39     181508        93021         24082     165552      530450       3765     627000
        26     213169        32904         10932     194629      209345       1998     689302
        31     137999        23401         10425     128727       23000       4338      28000
        91     489703        51094         24580     408460      235285       1715     348230

7 rows selected.
```

- Describing structure of skorea.

```
SQL> desc sid_skorea;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 S_ID                                      NOT NULL NUMBER(2)
 STATE_NAME                                         VARCHAR2(15)
```

```
SQL> select * from sid_skorea;

      S_ID STATE_NAME
---------- ---------------
        16 Seoul
        28 Busan
        46 Daegu
        44 Incheon
        31 Ulsan
        36 Gwangju
        96 Gyeryong

7 rows selected.
```

- Describing structure skorea_states

```
SQL> desc skorea_states;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 S_ID                                               NUMBER(2)
 CASES                                              NUMBER(10)
 ACTIVE_CASES                                       NUMBER(10)
 SERIOUS_CASES                                      NUMBER(10)
 RECOVERED                                          NUMBER(15)
 TOTAL_TESTS                                        NUMBER(15)
 DEATHS                                             NUMBER(10)
 POPULATION                                         NUMBER(15)
```

```
SQL> select * from skorea_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        16      98302         1988           305      79305    10005320        306   10349312
        28     287490        38960         18793     228630     2987027       1050    3678555
        46     590175        40928         19084     390781     2004891        980    2566540
        44     197709        98301         10937      98990     2004850       2800    2628000
        31      97302       308701         10982      87028     7932064       1507     962865
        36      29078         8930          2091      10949       76029        309      81780
        96       9820         2098          1008       8899       40910        256      43269

7 rows selected.
```

- Describing structure of sid_japan.

```
SQL> desc sid_japan;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 S_ID                                      NOT NULL NUMBER(2)
 STATE_NAME                                         VARCHAR2(15)
```

```
SQL> select * from sid_japan;

      S_ID STATE_NAME
---------- ---------------
        15 Hokkaido
        27 Tohoku
        45 Kanto
        43 Chubu
        30 Shikoku
        35 kansai
        95 Chugoku

7 rows selected.
```

- Describing structure of japan_states.

```
SQL> desc japan_states;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 S_ID                                               NUMBER(2)
 CASES                                              NUMBER(10)
 ACTIVE_CASES                                       NUMBER(10)
 SERIOUS_CASES                                      NUMBER(10)
 RECOVERED                                          NUMBER(15)
 TOTAL_TESTS                                        NUMBER(15)
 DEATHS                                             NUMBER(10)
 POPULATION                                         NUMBER(15)
```

```
SQL> select * from japan_states;

     S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
--------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
       15     491087        29461         19460     390572      101048       4528    1957914
       27      15372         9845          3061       9994       18043        583      17297
       45    3981057       291402        100921     308193    30689150       4890   43300000
       43     936801       492685         29750    6092605      200920       5920   23010276
       30     209875        49028         28964     190589      300719       2459     335767
       35     983170        39086         10948     689305    20087201       6985   22757897
       95     983070        47808         10938    6098302    69379287       3090    7563428

7 rows selected.
```

- Describing structure of sid_netherlands.

```
SQL> desc sid_netherlands;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 S_ID                                      NOT NULL NUMBER(2)
 STATE_NAME                                         VARCHAR2(15)
```

```
SQL> select * from sid_netherlands;

      S_ID STATE_NAME
---------- ---------------
        14 Zeeland
        26 Flevoland
        44 Limburg
        42 Friesland
        29 Drenthe
        34 Gelderland
        94 Utrecht

7 rows selected.
```

- Describing structure of Netherlands_states.

```
SQL> desc netherlands_states;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 S_ID                                               NUMBER(2)
 CASES                                              NUMBER(10)
 ACTIVE_CASES                                       NUMBER(10)
 SERIOUS_CASES                                      NUMBER(10)
 RECOVERED                                          NUMBER(10)
 TOTAL_TESTS                                        NUMBER(15)
 DEATHS                                             NUMBER(10)
 POPULATION                                         NUMBER(15)
```

```
SQL> select * from netherlands_states;

     S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
--------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
       14       3098         1079           876       2097        5062        200       5564
       26      39685        29896         19869      29049      306986       4092     423021
       44       9935         5097          3098       7900       73058        950      86823
       42      83670        59831          6893      73987       98752       1380     123107
       29      38917        29786         19679      32986      398680       3281     488600
       34     996785       797640        385670     832850     2042970       4210    2084478
       94      89689        58530         27960      70957      218567       5280     296305

7 rows selected.
```

- Describing structure of sid_france.

```
SQL> desc sid_france;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 S_ID                                      NOT NULL NUMBER(2)
 STATE_NAME                                         VARCHAR2(15)
```

```
SQL> select * from sid_france;

      S_ID STATE_NAME
---------- ---------------
        12 paris
        24 brittany
        42 burgandy
        40 corsica
        27 lomousine
        32 picardy
        92 lower normandy

7 rows selected.
```

- Describing structure of france_states.

```
SQL> desc france_states;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ---------------------
 S_ID                                                           NUMBER(2)
 CASES                                                          NUMBER(10)
 ACTIVE_CASES                                                   NUMBER(10)
 SERIOUS_CASES                                                  NUMBER(10)
 RECOVERED                                                      NUMBER(10)
 TOTAL_TESTS                                                    NUMBER(15)
 DEATHS                                                         NUMBER(10)
 POPULATION                                                     NUMBER(15)
```

```
SQL> select * from france_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        12      60653        23647         12687      47658      546536       2223     214827
        24     975207        20945          3698      94387      728954      20457    4475295
        42     176943        83653         48968     146996     1567350       4987    1610067
        40     176943        83653         48968     146996     1567350       4987    1610067
        27     680742        30878         16875     596750      698752       4920     742770
        32     968750        69768         36870     854852     1598653       8839    1857481
        92     365289        89682         48736     202653      139669       9582    1422193

7 rows selected.
```

- Describing structure of sid_brazil.

```
SQL> desc sid_brazil;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ----------------
 S_ID                                                  NOT NULL NUMBER(2)
 STATE_NAME                                                     VARCHAR2(15)
```

```
SQL> select * from sid_brazil;

      S_ID STATE_NAME
---------- ---------------
        13 santa catarina
        25 bahia
        43 rio de janeiro
        41 mato grosso
        28 parana
        33 tocantis
        93 sao paulo

7 rows selected.
```

- Describing structure of brazil_states.

```
SQL> desc brazil_states;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 S_ID                                               NUMBER(2)
 CASES                                              NUMBER(10)
 ACTIVE_CASES                                       NUMBER(10)
 SERIOUS_CASES                                      NUMBER(10)
 RECOVERED                                          NUMBER(15)
 TOTAL_TESTS                                        NUMBER(15)
 DEATHS                                             NUMBER(10)
 POPULATION                                         NUMBER(15)
```

```
SQL> select * from brazil_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        13     593702        38586         28975     490676     5875659      50796    6091000
        25    1376870       429654        355820     296757     1965927      84650   14561000
        43     389548       168900         39690     298567     4968580      59600    5857904
        41    1965800        84650         49850    1295030     1987540      85470    2449024
        28     195963        49690         29675     128000      200630      40780     247863
        33    1005092        59386         39689     296785      986870       4800    1496880
        93     982648        73960         40600     887576     1968755       9032   11967825

7 rows selected.
```

# CONSTRAINTS.

- **PRIMARY KEY CONSTRAINT:** Each table must normally contain a column or set of columns that uniquely identifies rows of data that are stored in the table. This column or set of columns is referred to as the primary key.

***We have the following primary keys in our database.***

- *Primary Key in sid_USA table is (s_id number).*

```
SQL> create table sid_USA
  2  (s_id number(2) primary key,state_name varchar2(15));

Table created.
```

- *Primary Key in sid_korea table is (s_id number).*

```
SQL> create table sid_skorea
  2  (s_id number(2) primary key, state_name varchar2(15));

Table created.
```

- *Primary Key in sid_netherlands table is (s_id number).*

```
SQL> create table sid_netherlands
  2  (s_id number(2) primary key,state_name varchar2(15));

Table created.
```

- *Primary Key in sid_japan table is (s_id number).*

```
SQL> create table sid_japan
  2  (s_id number(2) primary key, state_name varchar2(15));

Table created.
```

- *Primary Key in sid_india table is (s_id number).*

```
SQL> create table sid_india
  2  (s_id number(2) primary key,state_name varchar2(15));

Table created.
```

- *Primary Key in sid_france table is (s_id number).*

```
SQL> create table sid_france
  2  (s_id number(2) primary key,state_name varchar2(15));

Table created.
```

- *Primary Key in sid_brazil table is (s_id number).*

```
SQL> create table sid_brazil
  2  (s_id number(2) primary key, state_name varchar2(15));

Table created.
```

- *Primary Key in country_id table is (c_id number).*

```
SQL> create table country_id
  2  (c_id number(2) primary key,country_name varchar2(15));

Table created.
```

- *Primary Key in vaccine_creation_info table is (vacc_id number).*

```
SQL> create table vaccine_creation_info
  2  (vacc_id number(1) primary key,vacc_name varchar2(30),laboratory_name varchar2(30),
  3  manufacturer varchar2(30),testing_date date,status varchar2(15) check(status in('successful','unsuccessful')),
  4  testing_subject varchar2(20));

Table created.
```

- **CHECK CONSTRAINT:** Sometimes the data values stored in a specific column must fall within some acceptable range of values. A CHECK constraint is used to enforce this data limit.

***We have the following check constraints in our database.***

```
SQL> create table vaccine_creation_info
  2  (vacc_id number(1) primary key,vacc_name varchar2(30),laboratory_name varchar2(30),
  3  manufacturer varchar2(30),testing_date date,status varchar2(15) check(status in('successful','unsuccessful')),
  4  testing_subject varchar2(20));

Table created.
```

- **FOREIGN KEY CONSTRAINTS:** A foreign key is a key used to link two tables together. A FOREIGN KEY is a field in one table that refers to the PRIMARY KEY in another table. The table containing the foreign key is called the child table and the table containing the candidate key is called the referenced or parent table.

***We have the following foreign keys in our database.***

- *Foreign Key in USA_states table is (s_id number).*

```
SQL> create table USA_states
  2  (s_id number(2),cases number(10),active_cases number(10),serious_cases number(10),
  3  recovered number(10),total_tests number(15),deaths number(10),population number(15),
  4  foreign key (s_id) references sid_USA(s_id));

Table created.
```

- *Foreign Key in skorea_states table is (s_id number).*

```
SQL> create table skorea_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(15),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_skorea(s_id));

Table created.
```

- *Foreign Key in netherland_states table is (s_id number).*

```
SQL> create table netherlands_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(10),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_netherlands(s_id));

Table created.
```

- *Foreign Key in japan_states table is (s_id number).*

```
SQL> create table japan_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(15),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_japan(s_id));

Table created.
```

- *Foreign Key in india_states table is (s_id number).*

```
SQL> create table india_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(10),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_india(s_id));

Table created.
```

- *Foreign Key in france_states table is (s_id number).*

```
SQL> create table france_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(10),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_france(s_id));

Table created.
```

- *Foreign Key in brazil_states table is (s_id number).*

```
SQL> create table brazil_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(15),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_brazil(s_id));

Table created.
```

- *Foreign Key in countries table is (c_id number).*

```
SQL> create table countries
  2  (c_id number(2),cases number(10),active_cases number(10),serious_cases number(10),
  3  recovered number(10),total_tests number(15),deaths number(10),population number(15),
  4  foreign key (c_id) references country_id(c_id));

Table created.
```

- *Foreign Key in link_vaccid_id table is (c_id number).*

```
SQL> create table link_vaccid_cid
  2  (vacc_id number(1),c_id number(2),
  3  foreign key (vacc_id) references vaccine_creation_info(vacc_id),
  4  foreign key (c_id) references country_id(c_id));

Table created.
```

- ## UNIQUE CONSTRAINTS:

A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.

You can say that it is little like primary key but it can accept only one null value and it cannot have duplicate values.

We have the following unique keys in our database

```
SQL> alter table country_id add unique(country_name);

Table altered.
```

```
SQL> alter table sid_usa add unique(state_name);

Table altered.
```

```
SQL> alter table sid_india add unique(state_name);
Table altered.
```

```
SQL> alter table sid_france add unique(state_name);
Table altered.
```

```
SQL> alter table sid_brazil add unique(state_name);
Table altered.
```

```
SQL> alter table sid_netherlands add unique(state_name);
Table altered.
```

```
SQL> alter table sid_japan add unique(state_name);
Table altered.
```

```
SQL> alter table sid_skorea add unique(state_name);
Table altered.
```

# DDL COMMANDS.

**DDL(Data Definition Language) :** DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

***DDL commands are:***

1)CREATE COMMAND: One Of The First Steps In Creating A Database Is To Create The Tables That Will Store Company / Organization's Data.

- In Order To Create A Table , Four Pieces Of Information Must Be Determined:
  1. The Table Name
  2. The Column (Field) Names
  3. Column Data Types And
  4. Column Sizes.

**Examples of create commands are :**

- Creating table world.

```
SQL> create table world
  2  (id number(1) primary key,cases number(15),active_cases number(15),
  3  serious_cases number(15),deaths number(15),recovered number(15), population number(20));

Table created.
```

- Creating table country_id.

```
SQL> create table country_id
  2  (c_id number(2) primary key,country_name varchar2(15));
```

- Creating table countries.

```
SQL> create table countries
  2  (c_id number(2),cases number(10),active_cases number(10),serious_cases number(10),
  3  recovered number(10),total_tests number(15),deaths number(10),population number(15),
  4  foreign key (c_id) references country_id(c_id));

Table created.
```

- Creating table vaccine_creation_info

```
SQL> create table vaccine_creation_info
  2  (vacc_id number(1) primary key,vacc_name varchar2(30),laboratory_name varchar2(30),
  3  manufacturer varchar2(30),testing_date date,status varchar2(15) check(status in('successful','unsuccessful')),
  4  testing_subject varchar2(20));

Table created.
```

- Create table link_vaccid_cid

```
SQL> create table link_vaccid_cid
  2  (vacc_id number(1),c_id number(2),
  3  foreign key (vacc_id) references vaccine_creation_info(vacc_id),
  4  foreign key (c_id) references country_id(c_id));

Table created.
```

- Creating table sid_brazil.

```
SQL> create table sid_brazil
  2  (s_id number(2) primary key, state_name varchar2(15));

Table created.
```

- Creating table brazil_states.

```
SQL> create table brazil_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(15),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_brazil(s_id));

Table created.
```

- Create table sid_france.

```
SQL> create table sid_france
  2  (s_id number(2) primary key,state_name varchar2(15));

Table created.
```

- Create table france_states.

```
SQL> create table france_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(10),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_france(s_id));

Table created.
```

- Creating sid_india table.

```
SQL> create table sid_india
  2  (s_id number(2) primary key,state_name varchar2(15));

Table created.
```

- Creating table india_states.

```
SQL> create table india_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(10),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_india(s_id));

Table created.
```

- Creating table sid_japan

```
SQL> create table sid_japan
  2  (s_id number(2) primary key, state_name varchar2(15));

Table created.
```

- Creating table japan_states.

```
SQL> create table japan_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(15),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_japan(s_id));

Table created.
```

- Creating table sid_netherlands.

```
SQL> create table sid_netherlands
  2  (s_id number(2) primary key,state_name varchar2(15));

Table created.
```

- Creating table netherlands_states.

```
SQL> create table netherlands_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(10),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_netherlands(s_id));

Table created.
```

- Creating table sid_skorea.

```
SQL> create table sid_skorea
  2  (s_id number(2) primary key, state_name varchar2(15));

Table created.
```

- Creating table skorea_states.

```
SQL> create table skorea_states
  2  (s_id number(2),cases number(10),active_cases number(10),
  3  serious_cases number(10),recovered number(15),total_tests number(15),
  4  deaths number(10),population number(15),foreign key (s_id) references sid_skorea(s_id));

Table created.
```

- Creating state id table for usa.

```
SQL> create table sid_USA
  2  (s_id number(2) primary key,state_name varchar2(15));

Table created.
```

- Creating usa_states table.

```
SQL> create table USA_states
  2  (s_id number(2),cases number(10),active_cases number(10),serious_cases number(10),
  3  recovered number(10),total_tests number(15),deaths number(10),population number(15),
  4  foreign key (s_id) references sid_USA(s_id));

Table created.
```

2)**ALTER COMMAND:** This command is used to alter the structure of the database.

***Examples of alter comman are:***

- Alter Command Can Be Used To **Add** New Columns. We use the ADD clause to add columns.

```
SQL> alter table world add(population number(15));

Table altered.
```

- Alter Command Can Be Used To **Modify** an Existing Columns. We can change a column's data type, size, and default value.

```
SQL> alter table world modify(population number(30));

Table altered.
```

- Alter Command Can Be Used To **Rename** an Existing Columns.This command is used to rename an object existing in the database.

```
SQL> alter table planet rename column cases to covid_cases;

Table altered.
```

- Alter Command Can Be Used To **DROP** an Existing Columns. This command is used to delete objects from the database.

```
SQL> alter table world drop(population);

Table altered.
```

- Alter Command Can Be Used To **add constraint.**

```
SQL> alter table world add check(population=7827103450);

Table altered.
```

3)**DROP COMMAND** : The drop table command removes the data as well as definition(structure) of an oracle table. When we drop a table, the database loses structure with all the data in the table, related constraints and all the indexes associated with it.

```
SQL> drop table planet;

Table dropped.
```

- It can also be used to drop any specific column.

```
SQL> alter table world drop(population);

Table altered.
```

4)**TRUNCATE COMMAND** : The truncate table command is used to remove all rows from a table and to release the storage space used by the table but does not delete the table definition(structure) from the database.

```
SQL> select * from planet;

COVID_CASES  RECOVERED ACTIVE_CASES SERIOUS_CASES     DEATHS
----------- ---------- ------------ ------------- ----------
   49718121   35291386     13177095         90913    1249640

SQL> truncate table planet;

Table truncated.

SQL> select * from planet;

no rows selected
```

5)**DESC COMMAND :** The describe (desc) command can display the column names and data types for any table.
(we have used this command above to describe the structures of tables.)

6)**RENAME COMMAND :** A table can be renamed with the rename Command. This command does not affect table structure or Data; it simply gives the current table a new Name.

***Examples are :***

1)Renaming the name of the table.

```
SQL> rename world to planet;

Table renamed.
```

2)Renaming any specific column inside the table.

```
SQL> alter table planet rename column cases to covid_cases;

Table altered.
```

# DML COMMANDS.

**DML(Data Manipulation Language) :** The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

***DML Commands are :***

1)INSERT COMMAND: This command is used to insert data into a table.

*Examples of Insert command are :*

- Inserting rows into world table.

```
SQL> insert into world
  2  values(1,63641185,18137719,106010,1474981,44028485,7829114600);

1 row created.
```

- Inserting rows into country_id table.

```
SQL> insert all
  2  into country_id values(11,'USA')
  3  into country_id values(25,'India')
  4  into country_id values(50,'France')
  5  into country_id values(13,'Brazil')
  6  into country_id values(45,'Netherlands')
  7  into country_id values(84,'Japan')
  8  into country_id values(35,'South Korea')
  9  select * from dual;

7 rows created.
```

- Inserting rows into countries table.

```
SQL> insert all
  2  into countries values(11,10182818,3497817,18480,6441744,156553957,243257,331690410)
  3  into countries values(25,8507754,512624,8944,7868968,116542304,126162,1384789754)
  4  into countries values(50,1748705,1580598,4421,127938,17651546,40169,65324901)
  5  into countries values(13,5653561,426931,8318,5064344,21900000,162286,213094038)
  6  into countries values(45,404401,2000,630,300205,3535613,7690,17148361)
  7  into countries values(84,105941,7644,194,96461,2846368,1809,126337911)
  8  into countries values(35,27427,1981,58,24968,2702882,478,51284907)
  9  select * from dual;

7 rows created.
```

- Inserting rows into vaccine_creation_info

```
SQL> insert all
  2  into vaccine_creation_info values(1,'Pneumococcal vaccine','clinical management of SARI','gen target inc','18-dec-2019','un
successful','hamster')
  3  into vaccine_creation_info values(2,'hemoephilus influenzae','THSTI','pharma lonza','11-jan-2020','unsuccessful','pig')
  4  into vaccine_creation_info values(3,'pneumovax 23','CNRS research laboratory','eurofins','25-feb-2020','successful','rabbit
')
  5  into vaccine_creation_info values(4,'pertusis','LDV USP','astrazeneca','13-apr-2020','successful','rat')
  6  into vaccine_creation_info values(5,'anthrax','applikon biotechnology','virtuvax','20-may-2020','unsuccessful','lizard')
  7  into vaccine_creation_info values(6,'japanese anciphalitis','kakogawa laboratories','barr labs inc','16-jul-2020','unsucces
sful','dog')
  8  into vaccine_creation_info values(7,'SK flu shot','gwangju institute of medicine','dynavax technologies','28-sep-2020','uns
uccessful','human')
  9  select * from dual;

7 rows created.
```

- Inserting rows into link_vaccid_cid

```
SQL> insert all
  2  into link_vaccid_cid values(1,11)
  3  into link_vaccid_cid values(2,25)
  4  into link_vaccid_cid values(3,50)
  5  into link_vaccid_cid values(4,13)
  6  into link_vaccid_cid values(5,45)
  7  into link_vaccid_cid values(6,84)
  8  into link_vaccid_cid values(7,35)
  9  select * from dual;

7 rows created.
```

- Inserting rows into sid_usa.

```
SQL> insert all
  2  into sid_USA values(10,'Texas')
  3  into sid_USA values(22,'California')
  4  into sid_USA values(40,'Florida')
  5  into sid_USA values(38,'New York')
  6  into sid_USA values(25,'Arizona')
  7  into sid_USA values(30,'Pennsylvania')
  8  into sid_USA values(90,'Virginia')
  9  select * from dual;

7 rows created.
```

- Inserting rows into usa_states table.

```
SQL> insert all
  2  into USA_states values(10,1012428,163723,82000,829458,9547430,19247,28995881)
  3  into USA_states values(22,970392,458159,24559,494270,19565151,17963,39512223)
  4  into USA_states values(40,837077,226127,38974,593853,10453650,17100,21477737)
  5  into USA_states values(38,562365,104023,28908,424541,15519172,33801,19453561)
  6  into USA_states values(25,257384,208287,35361,42950,2183128,6147,7278717)
  7  into USA_states values(30,232582,54778,18168,168708,2897339,9096,12801989)
  8  into USA_states values(90,190873,165728,22362,21441,2986354,2704,8535519)
  9  select * from dual;

7 rows created.
```

- Inserting rows into sid_india.

```
SQL> insert all
  2  into sid_india values(11,'maharashtra')
  3  into sid_india values(23,'madhya pradesh')
  4  into sid_india values(41,'tamil nadu')
  5  into sid_india values(39,'gujarat')
  6  into sid_india values(26,'rajasthan')
  7  into sid_india values(31,'punjab')
  8  into sid_india values(91,'kerala')
  9  select * from dual;

7 rows created.
```

- Inserting rows into india_states.

```
SQL> insert all
  2  into india_states values(11,211987,85725,42300,14209,110240153,4938,112372972)
  3  into india_states values(23,178298,31050,20845,167681,45352090,3034,85358965)
  4  into india_states values(41,746079,29080,4095,715892,50923530,11362,72147039)
  5  into india_states values(39,181508,93021,24082,165552,530450,3765,627000)
  6  into india_states values(26,213169,32904,10932,194629,209345,1998,689302)
  7  into india_states values(31,137999,23401,10425,128727,23000,4338,28000)
  8  into india_states values(91,489703,51094,24580,408460,235285,1715,348230)
  9  select * from dual;

7 rows created.
```

- Inserting rows into sid_skorea.

```
SQL> insert all
  2  into sid_skorea values(16,'Seoul')
  3  into sid_skorea values(28,'Busan')
  4  into sid_skorea values(46,'Daegu')
  5  into sid_skorea values(44,'Incheon')
  6  into sid_skorea values(31,'Ulsan')
  7  into sid_skorea values(36,'Gwangju')
  8  into sid_skorea values(96,'Gyeryong')
  9  select * from dual;

7 rows created.
```

- Inserting rows into skorea_states.

```
SQL> insert all
  2  into skorea_states values(16,98302,1988,305,79305,10005320,306,10349312)
  3  into skorea_states values(28,287490,38960,18793,228630,2987027,1050,3678555)
  4  into skorea_states values(46,590175,40928,19084,390781,2004891,980,2566540)
  5  into skorea_states values(44,197709,98301,10937,98990,2004850,2800,2628000)
  6  into skorea_states values(31,97302,308701,10982,87028,7932064,1507,962865)
  7  into skorea_states values(36,29078,8930,2091,10949,76029,309,81780)
  8  into skorea_states values(96,9820,2098,1008,8899,40910,256,43269)
  9  select * from dual;

7 rows created.
```

- Inserting rows into sid_japan

```
SQL> insert all
  2  into sid_japan values(15,'Hokkaido')
  3  into sid_japan values(27,'Tohoku')
  4  into sid_japan values(45,'Kanto')
  5  into sid_japan values(43,'Chubu')
  6  into sid_japan values(30,'Shikoku')
  7  into sid_japan values(35,'kansai')
  8  into sid_japan values(95,'Chugoku')
  9  select * from dual;

7 rows created.
```

- Inserting rows into japan_states.

```
SQL> insert all
  2  into japan_states values(15,491087,29461,19460,390572,101048,4528,1957914)
  3  into japan_states values(27,15372,9845,3061,9994,18043,583,17297)
  4  into japan_states values(45,3981057,291402,100921,308193,30689150,4890,43300000)
  5  into japan_states values(43,936801,492685,29750,6092605,200920,5920,23010276)
  6  into japan_states values(30,209875,49028,28964,190589,300719,2459,335767)
  7  into japan_states values(35,983170,39086,10948,689305,20087201,6985,22757897)
  8  into japan_states values(95,983070,47808,10938,6098302,69379287,3090,7563428)
  9  select * from dual;

7 rows created.
```

- Inserting rows into sid_netherlands.

```
SQL> insert all
  2  into sid_netherlands values(14,'Zeeland')
  3  into sid_netherlands values(26,'Flevoland')
  4  into sid_netherlands values(44,'Limburg')
  5  into sid_netherlands values(42,'Friesland')
  6  into sid_netherlands values(29,'Drenthe')
  7  into sid_netherlands values(34,'Gelderland')
  8  into sid_netherlands values(94,'Utrecht')
  9  select * from dual;

7 rows created.
```

- Inserting rows into Netherlands_states.

```
SQL> insert all
  2  into netherlands_states values(14,3098,1079,876,2097,5062,200,5564)
  3  into netherlands_states values(26,39685,29896,19869,29049,306986,4092,423021)
  4  into netherlands_states values(44,9935,5097,3098,7900,73058,950,86823)
  5  into netherlands_states values(42,83670,59831,6893,73987,98752,1380,123107)
  6  into netherlands_states values(29,38917,29786,19679,32986,398680,3281,488600)
  7  into netherlands_states values(34,996785,797640,385670,832850,2042970,4210,2084478)
  8  into netherlands_states values(94,89689,58530,27960,70957,218567,5280,296305)
  9  select * from dual;

7 rows created.
```

- Inserting rows into sid_france.

```
SQL> insert all
  2  into sid_france values(12,'paris')
  3  into sid_france values(24,'brittany')
  4  into sid_france values(42,'burgandy')
  5  into sid_france values(40,'corsica')
  6  into sid_france values(27,'lomousine')
  7  into sid_france values(32,'picardy')
  8  into sid_france values(92,'lower normandy')
  9  select * from dual;

7 rows created.
```

- Inserting rows into france_states.

```
SQL> insert all
  2  into france_states values(12,60653,23647,12687,47658,546536,2223,214827)
  3  into france_states values(24,975207,20945,3698,94387,728954,20457,4475295)
  4  into france_states values(42,176943,83653,48968,146996,1567350,4987,1610067)
  5  into france_states values(40,176943,83653,48968,146996,1567350,4987,1610067)
  6  into france_states values(27,680742,30878,16875,596750,698752,4920,742770)
  7  into france_states values(32,968750,69768,36870,854852,1598653,8839,1857481)
  8  into france_states values(92,365289,89682,48736,202653,139669,9582,1422193)
  9  select * from dual;

7 rows created.
```

- Inserting rows into sid_brazil.

```
SQL> insert all
  2  into sid_brazil values(13,'santa catarina')
  3  into sid_brazil values(25,'bahia')
  4  into sid_brazil values(43,'rio de janeiro')
  5  into sid_brazil values(41,'mato grosso')
  6  into sid_brazil values(28,'parana')
  7  into sid_brazil values(33,'tocantis')
  8  into sid_brazil values(93,'sao paulo')
  9  select * from dual;

7 rows created.
```

- Inserting rows into brazil_states.

```
SQL> insert all
  2  into brazil_states values(13,593702,38586,28975,490676,5875659,50796,6091000)
  3  into brazil_states values(25,1376870,429654,355820,296757,1965927,84650,14561000)
  4  into brazil_states values(43,389548,168900,39690,298567,4968580,59600,5857904)
  5  into brazil_states values(41,1965800,84650,49850,1295030,1987540,85470,2449024)
  6  into brazil_states values(28,195963,49690,29675,128000,200630,40780,247863)
  7  into brazil_states values(33,1005092,59386,39689,296785,986870,4800,1496880)
  8  into brazil_states values(93,982648,73960,40600,887576,1968755,9032,11967825)
  9  select * from dual;

7 rows created.
```

2)DELETE COMMAND : This command is used to delete records from a database table.

## ***Examples Of Delete Command Are :***

1)Deleting a particle row satisfying the condition

```
SQL> select * from usa_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        10    1012428       163723         82000     829458     9547430      19247   28995881
        22     970392       458159         24559     494270    19565151      17963   39512223
        40     837077       226127         38974     593853    10453650      17100   21477737
        38     562365       104023         28908     424541    15519172      33801   19453561
        25     257384       208287         35361      42950     2183128       6147    7278717
        30     232582        54778         18168     168708     2897339       9096   12801989
        90     190873       165728         22362      21441     2986354       2704    8535519

7 rows selected.

SQL> delete from usa_states where s_id=25;

1 row deleted.

SQL> select * from usa_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        10    1012428       163723         82000     829458     9547430      19247   28995881
        22     970392       458159         24559     494270    19565151      17963   39512223
        40     837077       226127         38974     593853    10453650      17100   21477737
        38     562365       104023         28908     424541    15519172      33801   19453561
        30     232582        54778         18168     168708     2897339       9096   12801989
        90     190873       165728         22362      21441     2986354       2704    8535519

6 rows selected.
```

2) Deleting the whole table

```
SQL> select * from usa_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        10    1012428       163723         82000     829458     9547430      19247   28995881
        22     970392       458159         24559     494270    19565151      17963   39512223
        40     837077       226127         38974     593853    10453650      17100   21477737
        38     562365       104023         28908     424541    15519172      33801   19453561
        30     232582        54778         18168     168708     2897339       9096   12801989
        90     190873       165728         22362      21441     2986354       2704    8535519

6 rows selected.

SQL> delete from usa_states;

6 rows deleted.

SQL> select * from usa_states;

no rows selected
```

3)UPDATE COMMAND :This Command Is Used To Update Existing Data Within A Table.

***Examples of update command are:***

```
SQL> select * from vaccine_creation_info;

  VACC_ID VACC_NAME                 LABORATORY_NAME             MANUFACTURER                TESTING_D STATUS       TESTING_SUBJECT
--------- ------------------------- --------------------------- --------------------------- --------- ------------ ------------------
        1 Pneumococcal vaccine      clinical management of SARI gen target inc              18-DEC-19 unsuccessful hamster
        2 hemoephilus influenzae    THSTI                       pharma lonza                11-JAN-20 unsuccessful pig
        3 pneumovax 23              CNRS research laboratory    eurofins                    25-FEB-20 successful   rabbit
        4 pertusis                  LDV USP                     astrazeneca                 13-APR-20 successful   rat
        5 anthrax                   applikon biotechnology      virtuvax                    20-MAY-20 unsuccessful lizard
        6 japanese anciphalitis     kakogawa laboratories       barr labs inc               16-JUL-20 unsuccessful dog
        7 SK flu shot               gwangju institute of medicine dynavax technologies      28-SEP-20 unsuccessful human

7 rows selected.

SQL> update vaccine_creation_info set testing_subject='human' where vacc_id=6;

1 row updated.
```

```
SQL> select * from vaccine_creation_info;

  VACC_ID VACC_NAME                 LABORATORY_NAME             MANUFACTURER                TESTING_D STATUS       TESTING_SUBJECT
--------- ------------------------- --------------------------- --------------------------- --------- ------------ ------------------
        1 Pneumococcal vaccine      clinical management of SARI gen target inc              18-DEC-19 unsuccessful hamster
        2 hemoephilus influenzae    THSTI                       pharma lonza                11-JAN-20 unsuccessful pig
        3 pneumovax 23              CNRS research laboratory    eurofins                    25-FEB-20 successful   rabbit
        4 pertusis                  LDV USP                     astrazeneca                 13-APR-20 successful   rat
        5 anthrax                   applikon biotechnology      virtuvax                    20-MAY-20 unsuccessful lizard
        6 japanese anciphalitis     kakogawa laboratories       barr labs inc               16-JUL-20 unsuccessful human
        7 SK flu shot               gwangju institute of medicine dynavax technologies      28-SEP-20 unsuccessful human

7 rows selected.
```

## ➢ Using Update Command In Multiple Fields

```
SQL> select * from vaccine_creation_info;

   VACC_ID VACC_NAME                 LABORATORY_NAME              MANUFACTURER                TESTING_D STATUS       TESTING_SUBJECT
---------- ------------------------- ---------------------------- --------------------------- --------- ------------ -------------------
         1 Pneumococcal vaccine      clinical management of SARI  gen target inc              18-DEC-19 unsuccessful hamster
         2 hemoephilus influenzae    THSTI                        pharma lonza                11-JAN-20 unsuccessful pig
         3 pneumovax 23              CNRS research laboratory     eurofins                    25-FEB-20 successful   rabbit
         4 pertusis                  LDV USP                      astrazeneca                 13-APR-20 successful   rat
         5 anthrax                   applikon biotechnology       virtuvax                    20-MAY-20 unsuccessful lizard
         6 japanese anciphalitis     kakogawa laboratories        barr labs inc               16-JUL-20 unsuccessful dog
         7 SK flu shot               gwangju institute of medicine dynavax technologies       28-SEP-20 unsuccessful human

7 rows selected.
```

```
SQL> update vaccine_creation_info set status='successful',testing_subject='dog' where vacc_id=1;

1 row updated.

SQL> select * from vaccine_creation_info;

   VACC_ID VACC_NAME                 LABORATORY_NAME              MANUFACTURER                TESTING_D STATUS       TESTING_SUBJECT
---------- ------------------------- ---------------------------- --------------------------- --------- ------------ -------------------
         1 Pneumococcal vaccine      clinical management of SARI  gen target inc              18-DEC-19 successful   dog
         2 hemoephilus influenzae    THSTI                        pharma lonza                11-JAN-20 unsuccessful pig
         3 pneumovax 23              CNRS research laboratory     eurofins                    25-FEB-20 successful   rabbit
         4 pertusis                  LDV USP                      astrazeneca                 13-APR-20 successful   rat
         5 anthrax                   applikon biotechnology       virtuvax                    20-MAY-20 unsuccessful lizard
         6 japanese anciphalitis     kakogawa laboratories        barr labs inc               16-JUL-20 unsuccessful dog
         7 SK flu shot               gwangju institute of medicine dynavax technologies       28-SEP-20 unsuccessful human

7 rows selected.
```

# DQL COMMANDS.

**SELECT CLAUSE**: This clause is used for Retrieving Certain Records From One Or More Tables.
(select clause is also used with various operators and functions as discussed further.)

```
SQL> select * from country_id;

      C_ID COUNTRY_NAME
---------- ---------------
        11 USA
        25 India
        50 France
        13 Brazil
        45 Netherlands
        84 Japan
        35 South Korea

7 rows selected.
```

```
SQL> select c_id,cases,deaths,population from countries;

      C_ID      CASES     DEATHS POPULATION
---------- ---------- ---------- ----------
        11   10182818     243257  331690410
        25    8507754     126162 1384789754
        50    1748705      40169   65324901
        13    5653561     162286  213094038
        45     404401       7690   17148361
        84     105941       1809  126337911
        35      27427        478   51284907

7 rows selected.
```

# TCL COMMANDS.

**TCL(transaction Control Language) :** TCL commands deals with the transaction within the database.

ROLLBACK COMMAND : Restore Database To Original Since The Last Commit.

***Example1:***

We deleted a particular row from the usa_states table.

```
SQL> delete from usa_states where s_id=25;

1 row deleted.

SQL> select * from usa_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        10    1012428       163723         82000     829458     9547430      19247   28995881
        22     970392       458159         24559     494270    19565151      17963   39512223
        40     837077       226127         38974     593853    10453650      17100   21477737
        38     562365       104023         28908     424541    15519172      33801   19453561
        30     232582        54778         18168     168708     2897339       9096   12801989
        90     190873       165728         22362      21441     2986354       2704    8535519

6 rows selected.
```

Then we deleted the whole usa_states table

```
SQL> select * from usa_states;

no rows selected
```

Then we performed the rollback operation

```
SQL> rollback;

Rollback complete.

SQL> select * from usa_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        10    1012428       163723         82000     829458     9547430      19247   28995881
        22     970392       458159         24559     494270    19565151      17963   39512223
        40     837077       226127         38974     593853    10453650      17100   21477737
        38     562365       104023         28908     424541    15519172      33801   19453561
        25     257384       208287         35361      42950     2183128       6147    7278717
        30     232582        54778         18168     168708     2897339       9096   12801989
        90     190873       165728         22362      21441     2986354       2704    8535519

7 rows selected.
```

And all the operations performed were rolbacked since it was dml commands .Rollback can't be done on ddl commands.

***Example2:***

We have a table as vaccine_creation_info

```
SQL> select * from vaccine_creation_info;

  VACC_ID VACC_NAME                 LABORATORY_NAME             MANUFACTURER                TESTING_D STATUS       TESTING_SUBJECT
--------- ------------------------- --------------------------- --------------------------- --------- ------------ -------------------
        1 Pneumococcal vaccine      clinical management of SARI gen target inc              18-DEC-19 unsuccessful hamster
        2 hemoephilus influenzae    THSTI                       pharma lonza                11-JAN-20 unsuccessful pig
        3 pneumovax 23              CNRS research laboratory    eurofins                    25-FEB-20 successful   rabbit
        4 pertusis                  LDV USP                     astrazeneca                 13-APR-20 successful   rat
        5 anthrax                   applikon biotechnology      virtuvax                    20-MAY-20 unsuccessful lizard
        6 japanese anciphalitis     kakogawa laboratories       barr labs inc               16-JUL-20 unsuccessful dog
        7 SK flu shot               gwangju institute of medicine dynavax technologies      28-SEP-20 unsuccessful human

7 rows selected.
```

This table is updated using the update command

```
SQL> update vaccine_creation_info set testing_subject='human' where vacc_id=6;

1 row updated.

SQL> select * from vaccine_creation_info;

  VACC_ID VACC_NAME                 LABORATORY_NAME             MANUFACTURER                TESTING_D STATUS       TESTING_SUBJECT
--------- ------------------------- --------------------------- --------------------------- --------- ------------ -------------------
        1 Pneumococcal vaccine      clinical management of SARI gen target inc              18-DEC-19 unsuccessful hamster
        2 hemoephilus influenzae    THSTI                       pharma lonza                11-JAN-20 unsuccessful pig
        3 pneumovax 23              CNRS research laboratory    eurofins                    25-FEB-20 successful   rabbit
        4 pertusis                  LDV USP                     astrazeneca                 13-APR-20 successful   rat
        5 anthrax                   applikon biotechnology      virtuvax                    20-MAY-20 unsuccessful lizard
        6 japanese anciphalitis     kakogawa laboratories       barr labs inc               16-JUL-20 unsuccessful human
        7 SK flu shot               gwangju institute of medicine dynavax technologies      28-SEP-20 unsuccessful human

7 rows selected.
```

The updated command can be rollbackd since it's a dml command

So after performing rollback

```
SQL> rollback;

Rollback complete.

SQL> select * from vaccine_creation_info;

   VACC_ID VACC_NAME                 LABORATORY_NAME             MANUFACTURER                TESTING_D STATUS       TESTING_SUBJECT
---------- ------------------------- --------------------------- --------------------------- --------- ------------ -------------------
         1 Pneumococcal vaccine      clinical management of SARI gen target inc              18-DEC-19 unsuccessful hamster
         2 hemoephilus influenzae    THSTI                       pharma lonza                11-JAN-20 unsuccessful pig
         3 pneumovax 23              CNRS research laboratory    eurofins                    25-FEB-20 successful   rabbit
         4 pertusis                  LDV USP                     astrazeneca                 13-APR-20 successful   rat
         5 anthrax                   applikon biotechnology      virtuvax                    20-MAY-20 unsuccessful lizard
         6 japanese anciphalitis     kakogawa laboratories       barr labs inc               16-JUL-20 unsuccessful dog
         7 SK flu shot               gwangju institute of medicine dynavax technologies      28-SEP-20 unsuccessful human

7 rows selected.
```

The updated row is rollbacked to it's original value.

**COMMIT COMMAND:** Used to commit the work done so far (saves work done)

```
SQL> commit;

Commit complete.
```

**SAVEPOINT COMMAND**: Identify A Point In A Transaction To Which You Can Later Roll Back.

# DCL COMMANDS.

**DCL(Data Control Language) :** DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system. DCL commands can only be performed by super users.

```
SQL> connect system
Enter password:
Connected.
SQL> _
```

**GRANT COMMAND:** This command gives user's access privileges to database.

Example:
We can grant all the privileges to user at once :

```
SQL> grant all privileges to dbms_mp;

Grant succeeded.
```

Grant any specific privileges.

```
SQL> grant create any table to dbms_mp;
Grant succeeded.
SQL> grant drop any table to dbms_mp;
Grant succeeded.
```

**REVOKE COMAMND:** This command withdraw user's access privileges given by using the GRANT command.

Example:
We can revoke all the privileges from user at once:

```
SQL> revoke all privileges from dbms_mp;

Revoke succeeded.
```

Revoke any specific privelges.

```
SQL> revoke create table from dbms_mp;
Revoke succeeded.
SQL> revoke create view from dbms_mp;
Revoke succeeded.
```

# SQL OPERATORS.

**ARITHMETIC OPERATORS :** An arithmetic operator is an operator that denotes a specific mathematical operation.These operators can perform arithmetical operations on numeric operands involved. Arithmetic operators are addition(+), subtraction(-), multiplication(*) and division(/).

1)ADDITION(+): This Operator Is Used To Add Two Or More Expressions Or Numbers.

***Example:***

The below query displays the total cases by adding active_cases, serious_cases, and recovered covid patients.

```
SQL> select c_id,active_cases+serious_cases+recovered as total_cases,
  2  deaths,population from countries;

      C_ID TOTAL_CASES     DEATHS POPULATION
---------- ----------- ---------- ----------
        11     9958041     243257  331690410
        25     8390536     126162 1384789754
        50     1712957      40169   65324901
        13     5499593     162286  213094038
        45      302835       7690   17148361
        84      104299       1809  126337911
        35       27007        478   51284907
```

2)SUBTRACTION(-): This Operator Is Used To Subtract One Expression Or Number From Another Expression Or Number.

Example:

This query displays the cured covid patients.

```
SQL> select c_id, cases-deaths as population_immune_from_covid
  2  from countries;

      C_ID POPULATION_IMMUNE_FROM_COVID
---------- ----------------------------
        11                      9939561
        25                      8381592
        50                      1708536
        13                      5491275
        45                       396711
        84                       104132
        35                        26949

7 rows selected.
```

3)MULTIPLICATION (*): This Operator Is Used To Multiply Two Or More Expressions Or Numbers.

Example:

4)DIVISION (/): This operator is used to divide one expressions or numbers by another.

Example:

**COMPARISON OPERATORS:-** A comparison (or relational) operator is a mathematical symbol which is used to compare two values. Comparison operators are used in conditions that compares one expression with another. The result of a comparison can be true, false, or unknown (an operator that has one or two null expressions returns unknown).

**1.] EQUAL (=):** The equal to operator is used for equality test within two numbers or expressions.

Example:

This query displays the row from country_id table which has c_id=35.

```
SQL> select * from country_id where c_id=35;

      C_ID COUNTRY_NAME
---------- ---------------
        35 South Korea
```

**2.]GREATER THAN (>):**The greater than operator is used to test whether an expression (or number) is greater than another one.

Example:

This query displays the records of states in usa where the covid cases are greater than 900000.

```
SQL> select * from usa_states where cases>900000;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        10    1012428       163723         82000     829458     9547430      19247   28995881
        22     970392       458159         24559     494270    19565151      17963   39512223
```

**3.] LESS THAN (<):** The less than operator is used to test whether an expression (or number) is less than another one.

Example:

This query displays the records of states in south korea having population less than 2000000.

```
SQL> select * from skorea_states where population<2000000;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        31      97302       308701         10982      87028     7932064       1507     962865
        36      29078         8930          2091      10949       76029        309      81780
        96       9820         2098          1008       8899       40910        256      43269
```

**4.] GREATER THAN EQUAL TO (>=):**The greater than equal to operator is used to test whether an expression (or number) is either greater than or equal to another one.

Example:

This query displays the records of states from japan having no. of deaths greater than or equal to 4890.

```
SQL> select * from japan_states where deaths>=4890;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        45    3981057       291402        100921     308193    30689150       4890   43300000
        43     936801       492685         29750    6092605      200920       5920   23010276
        35     983170        39086         10948     689305    20087201       6985   22757897
```

**5.] LESS THAN EQUAL TO (<=):**The less than equal to operator is used to test whether an expression (or number) is either less than or equal to another one.

Example:

This query displays the records of states from brazil having total tests less than or equal to 986870.

```
SQL> select * from brazil_states where total_tests<=986870;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        28     195963        49690         29675     128000      200630      40780     247863
        33    1005092        59386         39689     296785      986870       4800    1496880
```

**6.] NOT EQUAL TO (<>,!=):** The not equal to operator is used for inequality test between two numbers or expression.
Example:
This query displays the row from sid_skorea table whose s_id is not equal to 96.

```
SQL> select * from sid_skorea where s_id<>96;

      S_ID STATE_NAME
---------- ---------------
        16 Seoul
        28 Busan
        46 Daegu
        44 Incheon
        31 Ulsan
        36 Gwangju

6 rows selected.
```

**LOGICAL OPERATORS :-** The Logical operators are those that are true or false. They return a true or false values to combine one or more true or false values.

**AND:** Logical AND compares between two Booleans as expression and returns true when both expressions are true.

***Example:***
This query displays the records from vaccine_creation_info table where the status of vaccine is ‘successful’ and testing subject is ’rabbit’.

```
SQL> select * from vaccine_creation_info where status='successful' and testing_subject='rabbit';

  VACC_ID VACC_NAME                 LABORATORY_NAME             MANUFACTURER                TESTING_D STATUS       TESTING_SUBJECT
---------- ------------------------- --------------------------- --------------------------- --------- ------------ -------------------
         3 pneumovax 23              CNRS research laboratory    eurofins                    25-FEB-20 successful   rabbit
```

**OR:** Logical OR compares between two Booleans as expression and returns true when one of the expression is true.

***Example:***

This query displays the records from vaccine_creation_info table where the status of vaccine is ‘successful’ or manufactuer is ’barr labs inc’.

```
SQL> select * from vaccine_creation_info where status='successful' or manufacturer='barr labs inc';

  VACC_ID VACC_NAME                 LABORATORY_NAME             MANUFACTURER                TESTING_D STATUS        TESTING_SUBJECT
--------- ------------------------- --------------------------- --------------------------- --------- ------------- -------------------
        3 pneumovax 23              CNRS research laboratory    eurofins                    25-FEB-20 successful    rabbit
        4 pertusis                  LDV USP                     astrazeneca                 13-APR-20 successful    rat
        6 japanese anciphalitis     kakogawa laboratories       barr labs inc               16-JUL-20 unsuccessful  dog
```

**NOT:** Not takes a single Boolean as an argument and changes its value from false to true or from true to false.

***Example:***

This query displays the records from vaccine_creation_info table where the status of vaccine is ‘unsuccessful’ (i.e not ‘successful’).

```
SQL> select * from vaccine_creation_info where not status='successful';

  VACC_ID VACC_NAME                 LABORATORY_NAME             MANUFACTURER                TESTING_D STATUS        TESTING_SUBJECT
--------- ------------------------- --------------------------- --------------------------- --------- ------------- -------------------
        1 Pneumococcal vaccine      clinical management of SARI gen target inc              18-DEC-19 unsuccessful  hamster
        2 hemoephilus influenzae    THSTI                       pharma lonza                11-JAN-20 unsuccessful  pig
        5 anthrax                   applikon biotechnology      virtuvax                    20-MAY-20 unsuccessful  lizard
        6 japanese anciphalitis     kakogawa laboratories       barr labs inc               16-JUL-20 unsuccessful  dog
        7 SK flu shot               gwangju institute of medicine dynavax technologies      28-SEP-20 unsuccessful  human
```

**SET OPERATORS :-** Set operators are used to join the results of two (or more) SELECT statements.

**UNION :** When multiple SELECT queries are joined using UNION operator, Oracle displays the combined result from all the compounded SELECT queries, after removing all duplicates and in sorted order (ascending by default), without ignoring the NULL values.

***Example:***

This query combine two tables i.e india_states and skorea_states and display the records of both the tables.

```
SQL> select * from india_states
  2  union
  3  select * from skorea_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        11     211987        85725         42300      14209   110240153       4938  112372972
        16      98302         1988           305      79305    10005320        306   10349312
        23     178298        31050         20845     167681    45352090       3034   85358965
        26     213169        32904         10932     194629      209345       1998     689302
        28     287490        38960         18793     228630     2987027       1050    3678555
        31      97302       308701         10982      87028     7932064       1507     962865
        31     137999        23401         10425     128727       23000       4338      28000
        36      29078         8930          2091      10949       76029        309      81780
        39     181508        93021         24082     165552      530450       3765     627000
        41     746079        29080          4095     715892    50923530      11362   72147039
        44     197709        98301         10937      98990     2004850       2800    2628000
        46     590175        40928         19084     390781     2004891        980    2566540
        91     489703        51094         24580     408460      235285       1715     348230
        96       9820         2098          1008       8899       40910        256      43269

14 rows selected.
```

**UNION ALL :** UNION and UNION ALL are similar in their functioning with a slight difference. But UNION ALL gives the result set without removing duplication and sorting the data. For example,in above query UNION is replaced by UNION ALL to see the effect.

***Example:***

This query combine two tables i.e sid_india and sid_skorea and display the records of both the tables.

```
SQL> select * from sid_india
  2  union
  3  select * from sid_skorea;

      S_ID STATE_NAME
---------- ---------------
        11 maharashtra
        16 Seoul
        23 madhya pradesh
        26 rajasthan
        28 Busan
        31 Ulsan
        31 punjab
        36 Gwangju
        39 gujarat
        41 tamil nadu
        44 Incheon
        46 Daegu
        91 kerala
        96 Gyeryong

14 rows selected.
```

## RANGE SEARCHING OPERATORS (BETWEEN, NOT BETWEEN):-

In order to select data that is within a range of values, the BETWEEN operator is used. In order to select the data outside the range of values, the NOT BETWEEN operator is used.

***Example:***

This query displays the records from countries table where the no.of deaths are between 1800 and 4000.

```
SQL> select * from countries where deaths between 1800 and 40000;

     C_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
--------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
       45     404401         2000           630     300205     3535613       7690   17148361
       84     105941         7644           194      96461     2846368       1809  126337911
```

## PATTERN MATCHING OPERATORS (LIKE):-

This operator is used to check whether a character string matches a specified pattern or not.

The following example finds the state whose name starts with the letter “G “:

```
SQL> select * from sid_skorea where state_name like 'G%';

      S_ID STATE_NAME
---------- ---------------
        36 Gwangju
        96 Gyeryong
```

The following example finds the state whose name ends with the string “ce”:

```
QL> select * from country_id where country_name like '%ce';

     C_ID COUNTRY_NAME
--------- --------------
       50 France
```

The following example finds the state whose name contains 7 letters only.

```
SQL> select * from sid_usa where state_name like '_______';

      S_ID STATE_NAME
---------- ---------------
        40 Florida
        25 Arizona
```

# SQL BUILT IN FUNCTIONS

**COLUMN ALIAS NAME:** Column Aliases Are Used To Make Column Headings In Our Result Set Easier To Read.

***Example:***

```
SQL> select c_id, cases-deaths as population_immune_from_covid
  2  from countries;

      C_ID POPULATION_IMMUNE_FROM_COVID
---------- ------------------------------
        11                        9939561
        25                        8381592
        50                        1708536
        13                        5491275
        45                         396711
        84                         104132
        35                          26949

7 rows selected.
```

```
SQL> select c_id,active_cases+serious_cases+recovered as total_cases,
  2  deaths,population from countries;

      C_ID TOTAL_CASES     DEATHS POPULATION
---------- ----------- ---------- ----------
        11     9958041     243257  331690410
        25     8390536     126162 1384789754
        50     1712957      40169   65324901
        13     5499593     162286  213094038
        45      302835       7690   17148361
        84      104299       1809  126337911
        35       27007        478   51284907
```

**TABLE ALIAS NAME:** Table Aliases Are Used To Shorten Sql Query To Make It Easier To Read Or When We Are Performing A Join (i.e: Listing The Same Table More Than Once In The From Clause).

***Example:***

This query will display country id , country name and population from country id table and countries table.

```
SQL> select c.c_id,c.country_name,co.population from country_id c,countries co where c.c_id=co.c_id;

      C_ID COUNTRY_NAME    POPULATION
---------- -------------- ----------
        11 USA            331690410
        13 Brazil         213094038
        25 India         1384789754
        35 South Korea     51284907
        45 Netherlands     17148361
        50 France          65324901
        84 Japan          126337911

7 rows selected.
```

# STRING FUNCTIONS :-

## *Case Manipulation Function :*

1. **CONCAT:** Concatenates [joins] the first character value to the second character value; equivalent to concatenation operator (||).

Example:

```
SQL> select concat(laboratory_name, manufacturer) as origin_of_vaccine from vaccine_creation_info;

ORIGIN_OF_VACCINE
------------------------------------------------------------
clinical management of SARIgen target inc
THSTIpharma lonza
CNRS research laboratoryeurofins
LDV USPastrazeneca
applikon biotechnologyvirtuvax
kakogawa laboratoriesbarr labs inc
gwangju institute of medicinedynavax technologies

7 rows selected.
```

2. **SUBSTR:** Returns specific characters from character value starting at a specific character position and going specified character positions long.

Example:

```
SQL> select substr('covid',1,3) from dual;

SUB
---
cov
```

3. **INSTR:** Returns the numeric position of a named string.

Example:

```
SQL> select instr('coronavirus','a') from dual;
INSTR('CORONAVIRUS','A')
-------------------------
                        6
```

4. **LENGTH:** Returns the number of characters in the expression.

Example:

```
SQL> select length(vacc_name) from vaccine_creation_info;

LENGTH(VACC_NAME)
-----------------
               20
               22
               12
                8
                7
               21
               11

7 rows selected.
```

**Character Manipulation Function**: The case-conversion functions perform letter case conversion on alphabetic characters. In the default locale, only the ASCII characters in the ranges A - Z and a - z can be modified by these functions, which enable you to perform case-insensitive searches in your queries and to specify the format of the output. The case-conversion functions are UPPER, LOWER, and INITCAP.

1)**UPPER :** The UPPER function accepts an *expression* argument and returns a character string in which every lowercase alphabetical character in the *expression* is replaced by a corresponding uppercase alphabetic character.
Example:
This query will display the names of states in capital letters (upper case) from sid_india table.

```
SQL> select upper(state_name) from sid_india;

UPPER(STATE_NAM
---------------
MAHARASHTRA
MADHYA PRADESH
TAMIL NADU
GUJARAT
RAJASTHAN
PUNJAB
KERALA

7 rows selected.
```

2)**LOWER :** The LOWER function accepts an *expression* argument and returns a character string in which every uppercase alphabetic character in the *expression* is replaced by a corresponding lowercase alphabetic character.

***Example:***

This query will display the names of states in lower case from sid_india table.

```
SQL> select lower(state_name) from sid_india;

LOWER(STATE_NAM
---------------
maharashtra
madhya pradesh
tamil nadu
gujarat
rajasthan
punjab
kerala

7 rows selected.
```

3)**INITCAP :** The INITCAP function returns a copy of the *expression* in which every word in the *expression* begins with an uppercase letter. With this function, a *word* begins after any character other than a letter. Thus, in addition to a blank space, symbols such as commas, periods, colons, and so on, introduce a new word.

***Example:***

```
SQL> select initcap(state_name) from sid_india;

INITCAP(STATE_N
---------------
Maharashtra
Madhya Pradesh
Tamil Nadu
Gujarat
Rajasthan
Punjab
Kerala

7 rows selected.
```

**ARITHEMATIC FUNCTIONS:** A mathematical function executes a mathematical operation usually based on input values that are provided as arguments, and return a numeric value as the result of the operation.

## COUNT:

Example:

This query will display the total no. of countries in country_id table.

```
SQL> select count(country_name) from country_id;

COUNT(COUNTRY_NAME)
-------------------
                  7
```

## MAX:

Example:

This query will display the maximum population from countries table.

```
SQL> select max(population) as largest_population from countries;

LARGEST_POPULATION
------------------
        1384789754
```

## MIN:

Example: This query will display the minimum population from countries table.

```
SQL> select min(population) as smallest_population from countries;

SMALLEST_POPULATION
-------------------
           17148361
```

## DATE & TIME FUNCTIONS:

Date and time functions operate on a date and time input value and return a string, numeric, or date and time value.

Example:

1)Addmonths

This query will add 100 months in testing_date row from vaccine_creation_info table and display the records.

```
SQL> select vacc_id,vacc_name,testing_date,add_months(testing_date,100) from vaccine_creation_info;

   VACC_ID VACC_NAME                    TESTING_D ADD_MONTH
---------- ---------------------------- --------- ---------
         1 Pneumococcal vaccine         18-DEC-19 18-APR-28
         2 hemoephilus influenzae       11-JAN-20 11-MAY-28
         3 pneumovax 23                 25-FEB-20 25-JUN-28
         4 pertusis                     13-APR-20 13-AUG-28
         5 anthrax                      20-MAY-20 20-SEP-28
         6 japanese anciphalitis        16-JUL-20 16-NOV-28
         7 SK flu shot                  28-SEP-20 28-JAN-29

7 rows selected.
```

2)To_char

This query will display the years in words from vaccine_creation_info table.

```
SQL> select vacc_id,vacc_name,testing_date,to_char(testing_date,'Ddspth month year')
  2  " year of vaccine testing " from vaccine_creation_info;

   VACC_ID VACC_NAME                    TESTING_D  year of vaccine testing
---------- ---------------------------- --------- ----------------------------------------------------
         1 Pneumococcal vaccine         18-DEC-19 Eighteenth december  twenty nineteen
         2 hemoephilus influenzae       11-JAN-20 Eleventh january   twenty twenty
         3 pneumovax 23                 25-FEB-20 Twenty-Fifth february  twenty twenty
         4 pertusis                     13-APR-20 Thirteenth april     twenty twenty
         5 anthrax                      20-MAY-20 Twentieth may       twenty twenty
         6 japanese anciphalitis        16-JUL-20 Sixteenth july      twenty twenty
         7 SK flu shot                  28-SEP-20 Twenty-Eighth september twenty twenty

7 rows selected.
```

**AGGREGATE FUNCTIONS:** SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.

Example:

1)Sum

This query will add the population from countries table and display the records.

```
SQL> select sum(population) "total population" from countries;

total population
----------------
     2189670282
```

2)Avg

This query displays the average population from countries table.

```
SQL> select avg(population) "average population" from countries;

average population
------------------
        312810040
```

# CLAUSE.

**ORDER BY CLAUSE:** The WHERE clause is used to filter records.The WHERE clause is used to extract only those records that fulfill a specified condition.

Example:

```
SQL> select * from country_id order by country_name asc;

      C_ID COUNTRY_NAME
---------- ---------------
        13 Brazil
        50 France
        25 India
        84 Japan
        45 Netherlands
        35 South Korea
        11 USA

7 rows selected.
```

```
SQL> select * from country_id order by country_name desc;

      C_ID COUNTRY_NAME
---------- ---------------
        11 USA
        35 South Korea
        45 Netherlands
        84 Japan
        25 India
        50 France
        13 Brazil

7 rows selected.
```

```
  2
SQL> select * from sid_skorea order by state_name;

      S_ID STATE_NAME
---------- ---------------
        28 Busan
        46 Daegu
        36 Gwangju
        96 Gyeryong
        44 Incheon
        16 Seoul
        31 Ulsan

7 rows selected.
```

**GROUP BY CLAUSE:** The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement. The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

Example:

```
SQL> select c_id,avg(cases),max(cases),min(cases)
  2  from countries group by c_id;

      C_ID AVG(CASES) MAX(CASES) MIN(CASES)
---------- ---------- ---------- ----------
        25    8507754    8507754    8507754
        11   10182818   10182818   10182818
        13    5653561    5653561    5653561
        84     105941     105941     105941
        35      27427      27427      27427
        50    1748705    1748705    1748705
        45     404401     404401     404401

7 rows selected.
```

# VIEWS.

In SQL, a view is a virtual table based on the result-set of an SQL statement. view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

## CREATING A VIEW:

**SIMPLE VIEW:** Simple views can only contain a single base table.

***Example:***

```
SQL> create or replace view country_covid_cases
  2  as
  3  select c_id,cases,deaths,population
  4  from countries;

View created.
```

**COMPLEX VIEW:** Complex views can be constructed on more than one base table.

***Example:***

```
SQL> create or replace view countrywise_population
  2  as
  3  select c.c_id,c.country_name,co.population
  4  from country_id c,countries co
  5  where c.c_id=co.c_id;

View created.
```

## CREATING READ ONLY VIEW:

```
SQL> create or replace view india_population
  2  as
  3  select s.s_id,s.state_name,i.population
  4  from sid_india s,india_states i
  5  where s.s_id=i.s_id with read only;

View created.
```

# DISPLAYING A VIEW:

***Example:***

```
SQL> select * from country_covid_cases;

      C_ID      CASES     DEATHS POPULATION
---------- ---------- ---------- ----------
        11   10182818     243257  331690410
        25    8507754     126162 1384789754
        50    1748705      40169   65324901
        13    5653561     162286  213094038
        45     404401       7690   17148361
        84     105941       1809  126337911
        35      27427        478   51284907

7 rows selected.
```

```
SQL> select * from countrywise_population;

      C_ID COUNTRY_NAME    POPULATION
---------- --------------- ----------
        11 USA              331690410
        13 Brazil           213094038
        25 India           1384789754
        35 South Korea       51284907
        45 Netherlands       17148361
        50 France            65324901
        84 Japan            126337911

7 rows selected.
```

# DROPPING A VIEW:

***Example:***

```
SQL> drop view countrywise_population;

View dropped.
```

# SEQUENCES.

A sequence is a set of integers 1, 2, 3, ... that are generated in order on demand. Sequences are frequently used in databases because many applications require each row in a table to contain a unique value and sequences provide an easy way to generate them.

We have created a dummy table seq_ex to implement sequence.

```
SQL> desc seq_ex;
 Name                      Null?    Type
 ------------------------- -------- ----------------
 VACC_ID                            NUMBER(1)
 VACC_NAME                          VARCHAR2(30)
```

## CREATING SEQUENCES

Creating a sequence which will start by 1, increment by 1, with maximum value 10.

```
SQL> create sequence vacc_seq
  2  start with 1
  3  increment by 1
  4  maxvalue 10
  5  nocycle;

Sequence created.
```

USING SEQUENCES

```
SQL> insert into seq_ex
  2  values(vacc_seq.nextval,'pneumonal vaccine');

1 row created.

SQL> insert into seq_ex
  2  values(vacc_seq.nextval,'anthrax');

1 row created.

SQL> select * from seq_ex;

   VACC_ID VACC_NAME
---------- ------------------------------
         1 pneumonal vaccine
         2 anthrax
```

## ALTERING SEQUENCES

```
SQL> alter sequence vacc_seq
  2  increment by 2
  3  maxvalue 10
  4  nocycle;

Sequence altered.
```

```
SQL> insert into seq_ex
  2  values(vacc_seq.nextval,'antivirus');

1 row created.
```

```
SQL> select * from seq_ex;

   VACC_ID VACC_NAME
---------- ------------------------------
         1 pneumonal vaccine
         2 anthrax
         4 antivirus
```

## DROPPING SEQUENCE

```
SQL> drop sequence vacc_seq;

Sequence dropped.
```

# INDEXES.

Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

**CREATING A INDEX :** The create index command is used to create indexes in tables (allows duplicate values).

**TYPES :-**

**1)SIMPLE COLUMN INDEX:** A single-column index should be created when retrieval will be executed using one column only as the key.

***Example:***

```
SQL> create index c_name on country_id(country_name);

Index created.
```

**2)UNIQUE COLUMN INDEX:** Unique indexes are indexes that help maintain data integrity by ensuring that no two rows of data in a table have identical key values. When you create a unique index for an existing table with data, values in the columns or expressions that comprise the index key are checked for uniqueness.

***Example:***

```
SQL> create unique index sid on sid_usa(state_name);

Index created.
```

**3)COMPOSITE COLUMN INDEX:**A Composite Index Is An Index Which Is Created On Multiple Columns In A Table.

***Example:***

```
SQL> create index india on india_states(cases,deaths,population);

Index created.
```

**DROPING INDEX:** The Drop index command is used to delete an index in a table.

```
SQL> drop index india;

Index dropped.

SQL> drop index sid;

Index dropped.

SQL> drop index c_name;

Index dropped.
```

# SYNONYMS.

A synonym is an alternative name for objects such as tables, views, sequences, stored procedures, and other database objects. A synonym will make easy to access database objects owned by other users. Synonyms are used when you are granting access to an object from another schema.

## CREATING SYNONYMS

***Example:***

```
SQL> create synonym india for india_states;

Synonym created.
```

```
SQL> select * from india;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        11     211987        85725         42300      14209   110240153       4938  112372972
        23     178298        31050         20845     167681    45352090       3034   85358965
        41     746079        29080          4095     715892    50923530      11362   72147039
        39     181508        93021         24082     165552      530450       3765     627000
        26     213169        32904         10932     194629      209345       1998     689302
        31     137999        23401         10425     128727       23000       4338      28000
        91     489703        51094         24580     408460      235285       1715     348230

7 rows selected.

SQL> select * from india_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        11     211987        85725         42300      14209   110240153       4938  112372972
        23     178298        31050         20845     167681    45352090       3034   85358965
        41     746079        29080          4095     715892    50923530      11362   72147039
        39     181508        93021         24082     165552      530450       3765     627000
        26     213169        32904         10932     194629      209345       1998     689302
        31     137999        23401         10425     128727       23000       4338      28000
        91     489703        51094         24580     408460      235285       1715     348230

7 rows selected.
```

```
SQL> create synonym vaccines for vaccine_creation_info;

Synonym created.
```

```
SQL> select * from vaccines;

   VACC_ID VACC_NAME                    LABORATORY_NAME              MANUFACTURER                 TESTING_D STATUS       TESTING_SUBJECT
---------- ---------------------------- ---------------------------- ---------------------------- --------- ------------ -------------------
         1 Pneumococcal vaccine         clinical management of SARI  gen target inc               18-DEC-19 unsuccessful hamster
         2 hemoephilus influenzae       THSTI                        pharma lonza                 11-JAN-20 unsuccessful pig
         3 pneumovax 23                 CNRS research laboratory     eurofins                     25-FEB-20 successful   rabbit
         4 pertusis                     LDV USP                      astrazeneca                  13-APR-20 successful   rat
         5 anthrax                      applikon biotechnology       virtuvax                     20-MAY-20 unsuccessful lizard
         6 japanese anciphalitis        kakogawa laboratories        barr labs inc                16-JUL-20 unsuccessful dog
         7 SK flu shot                  gwangju institute of medicine dynavax technologies        28-SEP-20 unsuccessful human

7 rows selected.

SQL> select * from vaccine_creation_info;

   VACC_ID VACC_NAME                    LABORATORY_NAME              MANUFACTURER                 TESTING_D STATUS       TESTING_SUBJECT
---------- ---------------------------- ---------------------------- ---------------------------- --------- ------------ -------------------
         1 Pneumococcal vaccine         clinical management of SARI  gen target inc               18-DEC-19 unsuccessful hamster
         2 hemoephilus influenzae       THSTI                        pharma lonza                 11-JAN-20 unsuccessful pig
         3 pneumovax 23                 CNRS research laboratory     eurofins                     25-FEB-20 successful   rabbit
         4 pertusis                     LDV USP                      astrazeneca                  13-APR-20 successful   rat
         5 anthrax                      applikon biotechnology       virtuvax                     20-MAY-20 unsuccessful lizard
         6 japanese anciphalitis        kakogawa laboratories        barr labs inc                16-JUL-20 unsuccessful dog
         7 SK flu shot                  gwangju institute of medicine dynavax technologies        28-SEP-20 unsuccessful human

7 rows selected.
```

# DROPPING SYNONYMS

***Example:***

```
SQL> drop synonym india;
Synonym dropped.
SQL> drop synonym vaccines;
Synonym dropped.
```

# PL/SQL PROGRAMMING

**EXEPTION HANDLING:** An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error condition.

- There are two types of exceptions –

  - ***Pre-defined exceptions:*** PL/SQL provides many pre-defined exceptions, which are executed when any database rule is violated by a program. For example, the predefined exception *NO_DATA_FOUND* is raised when a *SELECT INTO* statement returns no rows.

- Example for predefined exception no_data_found.

exeption - Notepad

File Edit Format View Help

```
declare
id country_id.c_id%type:=&id;
c_name country_id.country_name%type;
begin
select country_name into c_name
from country_id where c_id=id;
dbms_output.put_line('country name: '||c_name);
exception
when no_data_found then
dbms_output.put_line('No such country!');
when others then
dbms_output.put_line('Error!');
end;
/
```

```
SQL> @c:/sql/exeption.sql;
Enter value for id: 35
old   2: id country_id.c_id%type:=&id;
new   2: id country_id.c_id%type:=35;
country name: South Korea

PL/SQL procedure successfully completed.
```

```
SQL> @c:/sql/exeption.sql;
Enter value for id: 12
old   2: id country_id.c_id%type:=&id;
new   2: id country_id.c_id%type:=12;
No such country!

PL/SQL procedure successfully completed.
```

- Example for predefined exception too_many_rows.

too_many_rows - Notepad

File Edit Format View Help

```
declare
id vaccines.vacc_id%type:=&id;
v_name vaccines.vacc_name%type;
m_name vaccines.manufacturer%type;
begin
select vacc_name,manufacturer into v_name,m_name
from vaccines where id<>vacc_id;
dbms_output.put_line('vaccine name: '||v_name);
dbms_output.put_line('manufacturer name: '||m_name);
exception
when too_many_rows then
dbms_output.put_line('too many rows found!');
when others then
dbms_output.put_line('Error!');
end;
/
```

```
SQL> @c:/sql/too_many_rows.sql;
Enter value for id: 1
old   2: id vaccines.vacc_id%type:=&id;
new   2: id vaccines.vacc_id%type:=1;
too many rows found!

PL/SQL procedure successfully completed.
```

- ***User-defined exceptions:*** PL/SQL allows you to define your own exceptions according to the need of your program. A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure *DBMS_STANDARD.RAISE_APPLICATION_ERROR*.

***Exapmle:***

invalid_id - Notepad

File Edit Format View Help

```
declare
id countries.c_id%type:=&id;
co_cases countries.cases%type;
recov countries.recovered%type;
dth countries.deaths%type;
pop countries.population%type;
--user defined exception
invalid_id exception;
begin
if id<=0 then
raise invalid_id;
else
select cases,recovered,deaths,population
into co_cases,recov,dth,pop
from countries
where id=c_id;
dbms_output.put_line('cases: '||co_cases);
dbms_output.put_line('recovered: '||recov);
dbms_output.put_line('deaths: '||dth);
dbms_output.put_line('population: '||pop);
end if;
exception
when invalid_id then
dbms_output.put_line('Id must be greater than zero!');
when no_data_found then
dbms_output.put_line('no such record!');
when others then
dbms_output.put_line('Error!');
end;
/
```

```
SQL> @c:/sql/invalid_id.sql;
Enter value for id: -5
old   2: id countries.c_id%type:=&id;
new   2: id countries.c_id%type:=-5;
Id must be greater than zero!

PL/SQL procedure successfully completed.
```

```
SQL> @c:/sql/invalid_id.sql;
Enter value for id: 5
old   2: id countries.c_id%type:=&id;
new   2: id countries.c_id%type:=5;
no such record!

PL/SQL procedure successfully completed.
```

```
SQL> @c:/sql/invalid_id.sql;
Enter value for id: 25
old   2: id countries.c_id%type:=&id;
new   2: id countries.c_id%type:=25;
cases: 8519254
recovered: 7868968
deaths: 127162
population: 1384789754

PL/SQL procedure successfully completed.
```

# ➢ CURSORS

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data.

## TYPES:

- **EXPLICIT CURSOR:** Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block.

*Example:*

1) Cursor to print all countries' record.

countriescur - Notepad

File Edit Format View Help

```
set serveroutput on
DECLARE

cid country_id.c_id%type;
cname country_id.country_name%type;
c_cases countries.cases%type;

CURSOR c_countries is
select c.c_id, c.country_name, c1.cases from country_id c , countries c1 where c.c_id=c1.c_id;

BEGIN

        dbms_output.put_line('COUNTRY_Id'||chr(9)||'COUNTRY_NAME'||chr(9)||'TOTAL_CASES');
        dbms_output.put_line('_____');
        OPEN c_countries;
        LOOP
        FETCH c_countries into cid , cname , c_cases;
                EXIT WHEN c_countries %notfound;
        dbms_output.put_line(cid||chr(9)||chr(9)||cname||chr(9)||chr(9)||chr(9)||c_cases);

END LOOP;

CLOSE c_countries;

End;
/
```

OUTPUT:

```
SQL> ed c:/sql/countriescur.sql;

SQL> @c:/sql/countriescur.sql;
COUNTRY_Id      COUNTRY_NAME    TOTAL_CASES
-----------------------------------------------
11              USA                     10182818
13              Brazil                  5653561
25              India                   8507754
35              South Korea                     27427
45              Netherlands                     404401
50              France                  1748705
84              Japan                   105941

PL/SQL procedure successfully completed.
```

2) Cursor to update cases in countries and world table.

cursor_cw - Notepad

File Edit Format View Help

```
DECLARE

CURSOR c1 is
select c.cases,c.active_cases,c.serious_cases,c.deaths,w.cases from countries c, world w where c.population<w.population;

BEGIN

        OPEN c1;

        update countries set active_cases=active_cases+10000;
        update countries set serious_cases=serious_cases+500;
        update countries set deaths=deaths+1000;
        update countries set cases=cases+11500;
        update world set active_cases=active_cases+10000;
        update world set serious_cases=serious_cases+500;
        update world set deaths=deaths+1000;
        update world set cases=cases+11500;

        dbms_output.put_line('Cases UPDATED!!');
CLOSE c1;
END;
/
```

OUTPUT:

```
SQL> @c:/sql/cursor_cw.sql;
Cases UPDATED!!

PL/SQL procedure successfully completed.
```

3.) Cursor to print data of vaccine_creation_info table in a formatted manner.

vaccinecur - Notepad

File Edit Format View Help

```
declare
v_id vaccines.vacc_id%type;
v_name vaccines.vacc_name%type;
l_name vaccines.laboratory_name%type;
m_name vaccines.manufacturer%type;
t_date vaccines.testing_date%type;
t_sub vaccines.testing_subject%type;
total_rows number(2);
c_rownum number(2);
cursor vaccine_info is
select rownum,vacc_id,vacc_name,laboratory_name,manufacturer,testing_date,testing_subject from vaccines;
begin
dbms_output.put_line(chr(9)||chr(9)||'vaccine name'||chr(9)||chr(9)||'laboratory name'||chr(9)||chr(9)||'manufacturer'||chr(9)||chr(9)||'testing date'||chr(9)||
dbms_output.put_line('_____');
open vaccine_info;
loop
fetch vaccine_info into c_rownum,v_id,v_name,l_name,m_name,t_date,t_sub;
exit when vaccine_info%notfound;
dbms_output.put_line('vaccine no: '||c_rownum||chr(9)||v_id||chr(9)||v_name||chr(9)||l_name||chr(9)||m_name||chr(9)||t_date||chr(9)||t_sub);
dbms_output.put_line('=====================================================================================================================');
end loop;
total_rows:=vaccine_info%rowcount;
dbms_output.put_line(chr(10));
dbms_output.put_line(total_rows||' Vaccine records selected.');
close vaccine_info;
end;
/
```

OUTPUT:

```
SQL> @c:/sql/vaccinecur;
              vaccine name          laboratory name        manufacturer          testing date          testing subject
_____
vaccine no: 1   1       Pneumococcal vaccine    clinical management of SARI     gen target inc  18-DEC-19       hamster
=====================================================================================================================
vaccine no: 2   2       hemoephilus influenzae  THSTI   pharma lonza    11-JAN-20       pig
=====================================================================================================================
vaccine no: 3   3       pneumovax 23    CNRS research laboratory        eurofins        25-FEB-20       rabbit
=====================================================================================================================
vaccine no: 4   4       pertusis        LDV USP astrazeneca     13-APR-20       rat
=====================================================================================================================
vaccine no: 5   5       anthrax applikon biotechnology  virtuvax        20-MAY-20       lizard
=====================================================================================================================
vaccine no: 6   6       japanese anciphalitis   kakogawa laboratories   barr labs inc   16-JUL-20       dog
=====================================================================================================================
vaccine no: 7   7       SK flu shot     gwangju institute of medicine   dynavax technologies    28-SEP-20       human
=====================================================================================================================


7 Vaccine records selected.

PL/SQL procedure successfully completed.
```

4.) Cursor to print vaccine status.

c_vaccine - Notepad

File Edit Format View Help

```
Declare
c_status vaccine_creation_info.status%type;
s_rows number(2);
us_rows number(2);
Cursor  c_vaccine is
select status from vaccine_creation_info;

Begin
        dbms_output.put_line('========== Vaccine Status ==========');
        Open c_vaccine;
        Loop
        fetch c_vaccine into c_status;

        IF c_status in('successful') then
        s_rows:=c_vaccine%rowcount;

        ELSE
        us_rows:=c_vaccine%rowcount;

        EXIT WHEN c_vaccine%notfound;
END IF;
End Loop;

        dbms_output.put_line(chr(10));
        dbms_output.put_line(s_rows||' Vaccine successfully under process');

        dbms_output.put_line(chr(10));
        dbms_output.put_line(us_rows||' Vaccine are unsuccessful');



Close c_vaccine;
End;
/
```

Output:

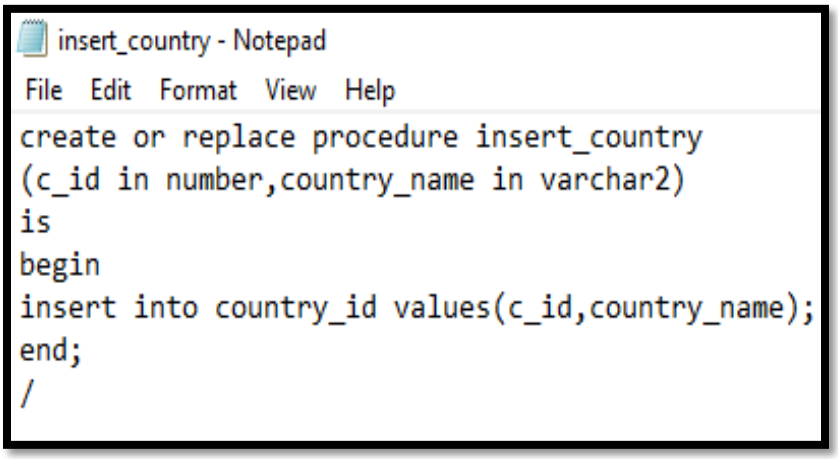
```
SQL> @c:/sql/c_vaccine.sql;
========== Vaccine Status ==========


4 Vaccine successfully under process


7 Vaccine are unsuccessful

PL/SQL procedure successfully completed.
```

# ➢ PROCEDURES

A Stored Procedure is a type of code in SQL that can be stored for later use and can be used many times. Stored Procedures are created to perform one or more DML operations on Database.

## *Creating a Procedure:*

- Procedure to insert record in country_id table.




```
SQL> @c:\sql\insert_country.sql;

Procedure created.
```

- Pl/sql block to use insert_country procedure

```
SQL> begin
  2  insert_country(10,'Russia');
  3  dbms_output.put_line('record inserted successfully');
  4  end;
  5  /

PL/SQL procedure successfully completed.
```

➢ Procedure to display covid instructons

instructions - Notepad

File Edit Format View Help

```
CREATE OR REPLACE PROCEDURE INSTRUCTIONS AS

BEGIN

DBMS_OUTPUT.PUT_LINE(chr(10)||'COVID cases are rising massively day by day !!!');
DBMS_OUTPUT.PUT_LINE('In that case it is necessary to take proper safety precautions.....');
DBMS_OUTPUT.PUT_LINE(chr(10)||chr(9)||'1.Wear mask');
DBMS_OUTPUT.PUT_LINE(chr(10)||chr(9)||'2.Maintain social distancing');
DBMS_OUTPUT.PUT_LINE(chr(10)||chr(9)||'3.wash your hands frequently');
DBMS_OUTPUT.PUT_LINE(chr(10)||chr(9)||'4.Do not touch you face with dirty hands');
DBMS_OUTPUT.PUT_LINE(chr(10)||chr(9)||'5.Cover your cough');
DBMS_OUTPUT.PUT_LINE(chr(10)||chr(9)||'6.Prevent unnecessary outing');
DBMS_OUTPUT.PUT_LINE(chr(10)||'STAY HOME !! STAY SAFE !!');

END;
/
```

```
SQL> @c:/sql/instructions.sql;

Procedure created.
```

➢ Output of the procedure

```
SQL> execute instructions;

COVID cases are rising massively day by day !!!
In that case it is necessary to take proper safety precautions.....

        1.Wear mask

        2.Maintain social distancing

        3.wash your hands frequently

        4.Do not touch you face with dirty hands

        5.Cover your cough

        6.Prevent unnecessary outing

STAY HOME !! STAY SAFE !!

PL/SQL procedure successfully completed.
```

- Dropping the procedure:

```
SQL> drop procedure insert_country;

Procedure dropped.
```

```
SQL> drop procedure instructions;

Procedure dropped.
```

# ➢ FUNCTIONS

A function is a group or set of SQL and PL/SQL statements that perform a specific task. A function and procedure is a named PL/SQL Block which is similar. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

## *Creating And Executing a function:*

Example 1:

- Function to count total number of countries.

f_countries - Notepad

File Edit Format View Help

```
create or replace function total_countries
return number is
total number(2):=0;
begin
        select count(*) into total
        from countries;
        return (total);
end;
/
```

```
SQL> @c:/sql/f_countries;

Function created.
```

- Calling function

```
SQL> declare
  2  c number;
  3  begin
  4  c:=total_countries();
  5  dbms_output.put_line('Total no. of countries: '||c);
  6  end;
  7  /
Total no. of countries: 7

PL/SQL procedure successfully completed.
```

- Drop function

```
SQL> drop function total_countries;

Function dropped.
```

Example 2:

4.) Function to display avg, population.

population - Notepad

File Edit Format View Help

```
create or replace function population
return number is
avg_pop number(15):=0;
begin
select avg(population) into avg_pop from countries;
return avg_pop;
end;
/
```

```
SQL> @c:/sql/population.sql;

Function created.
```

5.) Calling function

```
SQL> declare
  2  a number(15);
  3  begin
  4  a:=population();
  5  dbms_output.put_line('average population: '||a);
  6  end;
  7  /
average population: 312810040

PL/SQL procedure successfully completed.
```

## ➢ TRIGGERS

A Database Trigger Is A Stored Pl/Sql Program Unit associated With A Specific Database Table. Oracle Executes. A Database Trigger Automatically When A Given SQL Operation (Like Insert, Update Or Delete) Affects The Table.

## ▪ TYPES OF TRIGGERS:

1 )ROW LEVEL TRIGGER: If the statement FOR EACH ROW is present in the CREATE TRIGGER clause of trigger, the trigger is a row trigger. A row trigger is fired for each row affected by an triggering statement.

Example:

1. Create A Row-Level Trigger On Table Brazil_States Which Will Store The Deleted Rows In Another Table D_Tab And Updated Tables In Another Table U_Tab.

First the tables d_tab and u_tab were created in the database.

Creating a trigger

trig_brazil - Notepad

File Edit Format View Help

```
CREATE OR REPLACE TRIGGER trig_brazil
AFTER DELETE OR UPDATE ON brazil_states
FOR EACH ROW
BEGIN
IF DELETING THEN
INSERT INTO d_tab(s_id, cases, active_cases, serious_cases, recovered, total_tests, deaths , population , del_date)
values (:old.s_id, :old.cases, :old.active_cases, :old.serious_cases, :old.recovered, :old.total_tests, :old.deaths , :old.population , sysdate);

ELSIF UPDATING THEN
INSERT INTO u_tab(s_id, cases, active_cases, serious_cases, recovered, total_tests, deaths , population , up_date)
values (:old.s_id, :old.cases, :old.active_cases, :old.serious_cases, :old.recovered, :old.total_tests, :old.deaths , :old.population , sysdate);

END IF;
END;
/
```

```
SQL> @c:/sql/trig_brazil.sql;

Trigger created.

SQL>
```

Triggering a trigger

```
SQL> select * from brazil_states;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION
---------- ---------- ------------ ------------- ---------- ----------- ---------- ----------
        13     593702        38586         28975     490676     5875659      50796    6091000
        25    1376870       429654        355820     296757     1965927      84650   14561000
        43     389548       168900         39690     298567     4968580      59600    5857904
        41    1965800        84650         49850    1295030     1987540      85470    2449024
        28     195963        49690         29675     128000      200630      40780     247863
        33    1005092        59386         39689     296785      986870       4800    1496880
        93     982648        73960         40600     887576     1968755       9032   11967825

7 rows selected.

SQL> delete from brazil_states where s_id=93;

1 row deleted.

SQL> select * from d_tab;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION DEL_DATE
---------- ---------- ------------ ------------- ---------- ----------- ---------- ---------- ---------
        93     982648        73960         40600     887576     1968755       9032   11967825 04-DEC-20
```

```
SQL> update brazil_states set deaths=5000 where s_id=33;

1 row updated.

SQL> select * from u_tab;

      S_ID      CASES ACTIVE_CASES SERIOUS_CASES  RECOVERED TOTAL_TESTS     DEATHS POPULATION UP_DATE
---------- ---------- ------------ ------------- ---------- ----------- ---------- ---------- ---------
        33    1005092        59386         39689     296785      986870       4800    1496880 04-DEC-20
```

Enabling and disabling trigger.

```
SQL> alter trigger trig_brazil disable;

Trigger altered.

SQL> alter trigger trig_brazil enable;

Trigger altered.
```

Dropping trigger.

```
SQL> drop trigger trig_brazil;

Trigger dropped.
```

2. Row level trigger for the skorea_states table that would fire for INSERT or UPDATE or DELETE operations performed on the skorea_states table. This trigger will display the difference between the old values and new values of covid cases:

Creating a trigger

skorea_trig - Notepad

File Edit Format View Help

```
create or replace trigger case_changes
before delete or insert or update on skorea_states
for each row
when (new.s_id>0)
declare
case_diff number(5);
begin
case_diff:=:new.cases-:old.cases;
dbms_output.put_line('old no. of cases: '||:old.cases);
dbms_output.put_line('new no. of cases: '||:new.cases);
dbms_output.put_line('difference in no. of cases: '||case_diff);
end;
/
```

```
SQL> @c:/sql/skorea_trig.sql;

Trigger created.
```

Triggering trigger

```
SQL> update skorea_states
  2  set cases=100000 where s_id=31;
old no. of cases: 97302
new no. of cases: 100000
difference in no. of cases: 2698

1 row updated.
```

```
SQL> insert into skorea_states values(70,9820,2098,1008,8899,40910,256,43269);
old no. of cases:
new no. of cases: 9820
difference in no. of cases:

1 row created.
```

**2)STATEMENT LEVEL TRIGGER:** A statement trigger, however, is fired only once for the triggering statement, regardless of the number of rows affected by the triggering statement.

Example:

1) Creating a statement-level trigger on table sid_france' which will allow the DML operations between a specific time only.

Creating trigger

france_trig - Notepad

File Edit Format View Help

```
CREATE OR REPLACE TRIGGER france_trig

BEFORE DELETE OR INSERT OR UPDATE ON sid_france

BEGIN

IF (TO_CHAR(SYSDATE,'hh:mi') BETWEEN '00:00' AND '18:00') THEN

RAISE_APPLICATION_ERROR(-20500, 'Confidential data. Table is secured.'||chr(10) ||chr(9)||
'You are not allowed to perform DML operations between
12.00 am and  6.00
pm.' );

END IF;

END;

/
```

```
SQL> @c:/sql/france_trig.sql;

Trigger created.
```

Triggering a trigger

Triggering by deleting

```
SQL> delete from sid_france where s_id=12;
delete from sid_france where s_id=12
            *
ERROR at line 1:
ORA-20500: Confidential data. Table is secured.
You are not allowed to perform DML operations between
12.00 am and  6.00
pm.
ORA-06512: at "DBMS_MP.FRANCE_TRIG", line 5
ORA-04088: error during execution of trigger 'DBMS_MP.FRANCE_TRIG'
```

Triggering by inserting

```
SQL> insert into sid_france
  2  values(45,'Brittany');
insert into sid_france
            *
ERROR at line 1:
ORA-20500: Confidential data. Table is secured.
You are not allowed to perform DML operations between
12.00 am and  6.00
pm.
ORA-06512: at "DBMS_MP.FRANCE_TRIG", line 5
ORA-04088: error during execution of trigger 'DBMS_MP.FRANCE_TRIG'
```

Triggering by updating

```
SQL> update sid_france
  2  set state_name='Paris'
  3  where s_id=40;
update sid_france
       *
ERROR at line 1:
ORA-20500: Confidential data. Table is secured.
You are not allowed to perform DML operations between
12.00 am and  6.00
pm.
ORA-06512: at "DBMS_MP.FRANCE_TRIG", line 5
ORA-04088: error during execution of trigger 'DBMS_MP.FRANCE_TRIG'
```