MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY JAIPUR

Lab Sheet: ***DATA DEFINITION LANGUAGE***

Objectives:

- *What is DDL?*
  - *Brief overview of DDL*
- *Getting familiar with DDL*
  - *Create table*
  - *How to add constraints into the table:*
- Exercise to create TABLES
- Data Manipulation Language (DML)- INSERT

- **Data Definition Language (DDL)-** Commands that defines a database, including creating, altering, and dropping tables and also establishing constraints.
  - **Brief overview of DDL:**
    It is used to define the database
    - CREATE - to create table in the database
    - ALTER - alters the structure of the database
    - DROP - delete objects (a table) from the database
    - TRUNCATE – removes all records (rows) from a table
    - **DELETE-** used to remove all rows or specified selected row(s) from a table (it is a ***DML statement***)
    - RENAME - rename an object tables

- **Getting started with DDL:**
  - **How to create table**: Syntax for creating table in database is mentioned below:

```
CREATE TABLE TableName
(       ColumnName1     DataType(size),
        ColumnName2     DataType(size),
        .
        .
      ColumnNameN       DataType(size)
);
```

Note: we can also add data integrity constraints in the table during the creation which is described in this lab sheet below.

What is Data Integrity Constraints

Integrity constraints are used in RDBMS to enforce the business rules associated with your database and prevent the entry of invalid information into tables. Data constraints can be passed to DBA at cell creation time. It is important that column data adhere to a predefined set of rules, as determined by the database administrator or application developer. If data being loaded fails any of the data constraints, will not load into database cell, reject the entered row (record) and will flash an appropriate error messages. For example, some columns in a database table can have specific rules that constrain the data contained within them. These constraints can affect how data columns in one table relate to those in another table.

Business rules specify conditions and relationships that must always be true or must always be false. For example, each company defines its own policies about salaries, employee numbers, and inventory tracking, and so on. It is important that data maintain **data integrity**, which is adherence to these rules, as determined by the database administrator or application developer.

All these constraints are given a constraint name, which is stored with its name

and instructions internally along the cell itself by DBA.

When designing a database application, developers have various options for guaranteeing the integrity of data stored in the database. These options include:

- Enforcing business rules with triggers, stored in database (explained in subsequent labs).
- Using stored procedures to completely control access to data (explained in subsequent labs).
- Enforcing business rules in the code of a database application.

These constraints include

1. **Not Null**
2. **Default Values**
3. **Unique**
4. **Primary Key**
5. **Foreign Key**

These data constraints can be defined either during the table creation using CREATE TABLE statement or latter on after the table has been created using ALTER TABLE statement.

Now we start to go through these one by one:

1. Not Null Constraint

During the creation of a table, if a row lacks a data value for a particular column that value is said to be ***null***. Columns may contain null values unless it is defined as *not null*. But a column in a table can be specified as "**not null**" when table is created, then it's not possible to insert a null value in such columns. Hence such column(s) becomes a mandatory column(s).

Syntax:

- **Adding not null constraint while creating the table**

```
CREATE TABLE tableName
(       Column_name 1       DataType        NOT NULL,
        Column_name 2       DataType,
        Column_name 2       DataType        NOT NULL
);
```

- ***Adding not null constraint after creating the table***

```
ALTER TABLE tableName MODIFY columnName NOT NULL;
```

Note:(MySQL): `ALTER TABLE tableName MODIFY columnName DataType NOT NULL;`

(Also explore CHANGE in MY SQL to do same)

2. Default Constraint:

A particular column in a table can be assigned a default value at the time of creation of the cell. When user not specifies values of this column (cell) while inserting a record with values, such cell will be automatically loaded with the default vale specified. The data type of specified default value must match the data type of the column.

Syntax:

- ***Adding default constraint while creating the table***

```
CREATE TABLE tableName
(       Column name 1       DataType ,
        Column name 2       Data Type DEFAULT default_value,
        Column name 3       DataType ,
);
```

➢ ***Adding default constraint after creating the table***

ALTER TABLE *tablename* MODIFY *columnName* DEFAULT default_value;

(MySQL) ALTER TABLE table_name ALTER column_name SET DEFAULT default_value;

Task 1: a) Create table **supplier (id, name, phone)** with a not null constraint on column id and a default constraint to assign a default value '99999999' to the phone column.

```
CREATE TABLE supplier ( id number(4) NOT NULL,
                        name varchar2(20) ,
                        phone number(12) DEFAULT 99999999 );
```

Insert the following records and observe the effect:

```
INSERT INTO supplier VALUES ( 555,'Sangam',676767676);
INSERT INTO supplier (id,phone) VALUES (734, 303033030);
INSERT INTO supplier (id,name) VALUES ( 320, 'Sundraam');
SELECT * from supplier;                    //observe the records of supplier table.
```

What about the phone no of the last inserted row in supplier table corresponding to id=220?

b) Add a not null constraint on the name column of supplier table.

Now run the command and observe the effect:

```
INSERT INTO supplier (id,phone) VALUES (340, 333303030);
INSERT INTO supplier (name,phone) VALUES ('Hoffer',90072345667);
SELECT * from supplier;        //observe the records of supplier table
```

To see all of the fields and their formats for the created supplier table execute:

DESC supplier;

3. Unique Constraint

It ensures that information in the column for each row is unique. A table may have many unique keys. Unique constraint can contain **null values** as long as the combination of values is unique. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns. This constraint is used to checks whether a column value will be unique among all rows in a table.

Syntax:

➢ ***Faces of adding UNIQUE constraint while creating the table***

a)
```
CREATE TABLE tableName
(     Column name1      Data Type UNIQUE,
      Column name2      Data Type
);
```

b)
```
CREATE TABLE tableName
(     Column name1      CONSTRAINT constraint_name   UNIQUE,
      Column name2      Data Type
);
```

c)
```
CREATE TABLE tableName
(     Column name1            Data Type,
      Column name2            Data Type,
      CONSTRAINT constraint_name UNIQUE (Column name's)
);
```

➢ ***Adding UNIQUE constraint after creating the table***

d) ALTER TABLE *tableName* ADD UNIQUE (Column name);

e) ALTER TABLE *tableName* ADD CONSTRAINT constraint_name UNIQUE (Column name's);

Task 2: a) Add **unique** constraint on **id and name** columns to the **supplier** table.

Now, Run the following command and observe the effect:

```
INSERT INTO supplier values(234,'Anna',909090909090);
INSERT INTO supplier values (234,'Kapler',7070707070);
INSERT INTO supplier (567,'Anna',80800808080);
INSERT INTO supplier (567,'Kapler',90900909090);

Select * from supplier;        //observe the records of supplier table
```

4.Primary Key Constraint

Primary key is one or more columns in a table used to identify uniquely each row in the table. Technically a primary key combines a unique and a not null constraint. Additionally, a table can have at most one primary key. Primary key can be one attribute or combination of more than one attribute known as composite primary key.

Syntax:

➢ ***Adding PRIMARY KEY constraint while creating the table***

There are many ways to add primary key constraint while creating the table:

a)
```
CREATE TABLE tableName
( Column name1          Data Type      PRIMARY KEY,
  Column name2          Data Type
);
```

b)
```
CREATE TABLE tableName
(  Column name1          Data Type,
   Column name2          Data Type,
      PRIMARY KEY (Column name1)
);
```

c)
```
CREATE TABLE tableName
(  Column name1          Data Type,
   Column name2          Data Type,
   Column name3          Data Type,
      CONSTRAINT constraint_name PRIMARY KEY (Column name1)
);
```

d) ***Used to specify Primary key as composite key***

```
CREATE TABLE tableName
( Column name1          Data Type,
  Column name2          Data Type,
  Column name3          Data Type,
      CONSTRAINT constraint_name PRIMARY KEY (Col name1,Col name2 )
);
```

- ***Adding PRIMARY KEY constraint after creating the table using ALTER***

  a) ALTER TABLE tableName
  ADD CONSTRAINT *constraint_name* PRIMARY KEY (Column names);

  b) ALTER TABLE tableName ADD PRIMARY KEY (Column name);

Task 3: Assign **id** as Primary key constraint in the existing **supplier** table.
Run the following command:

DESC supplier; //observer whether primary key created or not

INSERT INTO supplier VALUES (576,'Jasmine',07653555544);

INSERT INTO supplier VALUES (764,'Boston',45007653544);

Select * from supplier; //observe the inserted records of table

5. Foreign Key Constraint:

A foreign key means that values in one of the table must also appear in another table. It represents relationship between tables. The referenced table is called the parent table while the table with the foreign key is called the child table. The foreign key in the child table will generally reference a primary key in the parent table. A foreign key in one table points to a primary key in another table i.e. a foreign key constraint requires values in one table to match values in another table.

- ***Adding FOREIGN KEY constraint while creating the table***

Let's consider the following **table1** and **table2** tables to understand the concept of foreign key while creating the table.

```
CREATE TABLE table1
        (   Column name1        Data Type,
            Column name2        Data Type,
            Column name3        Data Type,
             CONSTRAINT constraint_name PRIMARY KEY (Column name1)
        );
```

Now create a new table *table2* with foreign keys constraints:

There are many ***ways to add foreign key constraint*** while creating the table.

a)
```
CREATE TABLE  table2
 ( Column name4         Data Type,
   Column name5         Data Type,
   Column name6         Data Type,
   CONSTRAINT constraint_name PRIMARY KEY (Column name4),
  CONSTRAINT constraint_name FOREIGN KEY (Column name4)
                        REFERENCES table1 (Column name1)
 );
```

b) CREATE TABLE **table2**

(Column name4 Data Type PRIMARY KEY,
Column name5 Data Type FOREIGN KEY REFERENCES
table1 (**Column name1**),
Column name6 Data Type
);

c) CREATE TABLE **table2**

(Column name4 Data Type,
Column name5 Data Type,
Column name6 Data Type,
CONSTRAINT *constraint_name* PRIMARY KEY (Column name4),
CONSTRAINT FOREIGN KEY (**Column name4**) REFERENCES
table1 (**Column name1**)
);

- ***Adding FOREIGN KEY constraint after creating the table***

Let's consider the following **table1** and **table2** tables to understand the concept of foreign key after table created:

CREATE TABLE **table1**
(**Column name1** Data Type,
Column name2 Data Type,
Column name3 Data Type,
CONSTRAINT *constraint_name* PRIMARY KEY (**Column name1**)
);

CREATE TABLE **table2**
(**Column name4** Data Type,
Column name5 Data Type,
Column name6 Data Type,
CONSTRAINT *constraint_name* PRIMARY KEY (**Column name4**)
);

- Now, let's add foreign key constraint after *table2* has been created using ALTER statement

a) ALTER TABLE **table2** ADD FOREIGN KEY(**Column name4**)
REFERENCES **table1**(**Column name1**);

b) ALTER TABLE **table2**
ADD CONSTRAINT *constraint_name* FOREIGN KEY (**Column name4**)
REFERENCES **table1** (**Column name1**);

Task 4: a) Consider the existing **supplier** table created previously having **id** as Primary key. Create a table **product (pid,sid,Pname)** such that this table assigns foreign key as **sid** that references the column ***id*** of the **supplier** table.

For this perform the following:

CREATE TABLE product
(pid number(4) NOT NULL PRIMARY KEY,
sid number(4),
Pname varchar2(20),
CONSTRAINT fk_sply FOREIGN KEY (sid) REFERENCES supplier (id)
);

Perform the command: **DESC product;** //observer whether foreign key created or not

b) Now, insert the following records into product table and observer the effect:

```
INSERT INTO product (pid,sid,Pname) VALUES (2349, 220, 'Laptop');

INSERT INTO product (pid,sid,Pname) VALUES (3449,5555, 'Mobile');

INSERT INTO product (pid,sid,Pname) VALUES (4490, 434, 'Pen Drive');

INSERT INTO product (pid,sid,Pname) VALUES (9452, 7070, 'Pen drive');
Select * from product;      //observe the records of product table
```

Now, insert a record (7070,'Mukherjee',50000600068) in to supplier table.

After than again run,

```
INSERT INTO product (pid,sid,Pname) VALUES (9452, 7070, 'Pen drive');
```

which you executed above, observe the effect and reason with previous run as it inserted a record in database.

```
Select * from product;  //observe the records of product table
```

Data Manipulation Language (DML) : Those command that are used to maintain and query a database. DML is used for inserting, updating, deleting and querying (retrieving) the data in the database. They may be issued interactively, so that a result is returned immediately following the execution of the statement or they may be also included within programs written in a procedural programming language such as C, Java etc.

DML Commands which are used to manipulate or retrieve the database are:

a) **INSERT**: used to populate data into a table. It allows us to insert single or multiple records into database.

b) **SELECT** - retrieve data from the database.

c) **UPDATE** - updates existing data within a table.

d) **DELETE** - deletes all records from a table, the space for the records remain

INSERT statement: used to insert data into a table.

Syntax: There are two variations:

a) To insert single row of data

i) INSERT INTO *TableName* (*ColumnName1,ColumnName2 ,….., ColumnNameN)*

VALUES (*values1, values2,……., valuesN*)**;**

ii) INSERT INTO *TableName* VALUES (*values1,values2,…..,valuesN*);

Caution: Whenever we insert data values this way, we must ensure to give the field values in proper order, i.e. order in which they have created in the table.

b) **To insert *multiples records* quickly from another table having same structure**

i) INSERT INTO *TableNname* **(***ColumnName1, ColumnName2, ColumnName3,…)*
SELECT *ColumnName1, ColumnName2, ColumnName3,……*
FROM *TableName2* WHERE *Conditional-Expression ;*

ii) INSERT INTO *TableName*
SELECT *ColumnName1, ColumnName2, ColumnName3,……*
FROM *TableName2* WHERE *Conditional-Expression ;*

Task 5: Suppose we are creating table students:

```
CREATE TABLE students (
    studentID  number(5),
    student_name varchar2(20),
    student_address varchar 2(20),
    student_dob  date,
     constraint stud_pk primary key (student_ID)
    );
```

a) **INSERT INTO** students **VALUES** (200,'ashish','4123-MALVIYA','20-DEC-1988');

b) **INSERT INTO** students (studentID,student_name, student_address, student_dob)
VALUES (200,'ashish','4123-MALVIYA','20-DEC-1988');

//observe the records of students table

To Drop Constraints : DROP CONSTRAINT command delete various constraints such as a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint.

Syntax: `ALTER TABLE TableName DROP CONSTRAINT constraints_name;`

Task : perform DROP command to drop various constraints applied previously during the creation of tables and observer the effects.

Exercise TASK:

a) Create the following tables in database:

NOTE: Maintain same names as mentioned for databases.

***Use appropriate* MySQL notation *for data type* as mentioned for below tables**

a) *Create table* ***SAILORS***

Attribute Name	Data type
Sid	NUMBER(3)
Sname	VARCHAR2 (20)
Rating	NUMBER (2)
Age	NUMBER(3,1)

b) *Create table* ***BOATS***

Attribute Name	Data Type
Bid	NUMBER (3)
Bname	VARCHAR2 (12)
Color	VARCHAR2 (8)

c) *Create table* ***RESERVES***

Attribute Name	Data Type
Sid	NUMBER (3)
Bid	NUMBER (3)
Day	DATE

Note: Use corresponding MySQL data types such as float () ,char() , varchar() and int()/integer() etc.

b) Add the following constraints to the above created tables as shown below

Table Name	Column Name	Constraints As
SAILORS	sid rating sname	Primary Key NOT NULL NOT NULL
BOATS	bid bname color	Primary Key NOT NULL
RESERVES	(sid,bid) (sid,bid)	PRIMARY KEY

c) Now Insert the below given data into the created tables as described below:

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Reserve

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Boats

SQL/ORACLE Solution:

```
CREATE  TABLE   SAILORS (
    sid  NUMBER(3),
   sname VARCHAR2(20)  NOT  NULL,
   rating NUMBER(2) NOT NULL,
   age NUMBER(3,1),
  CONSTRAINT  pk_sailors  PRIMARY KEY(sid)
);
```

```
CREATE  TABLE  BOATS (
  bid   NUMBER(3),
   bname VARCHAR2(12) NOT NULL,
   color  VARCHAR2(8)  NOT NULL,
  CONSTRAINT  pk_boats  PRIMARY KEY(bid)
);
```

```
CREATE TABLE   RESERVES
(           sid   NUMBER(3),
            bid   NUMBER(3),
            day DATE,
CONSTRAINT pk_reserves PRIMARY KEY(sid,bid) ,
CONSTRAINT  fk_sailors  FOREIGN KEY(sid)  REFERENCES SAILORS(sid),
CONSTRAINT  fk_boats FOREIGN KEY(bid)   REFERENCES BOATS(bid)
);
```

Insert the following data in to corresponding tables:

SAILORS TABLES:

```
insert into SAILORS VALUES (22,'Dustin',7,45.0);
insert into SAILORS VALUES (29,'Brutus',1,33.0);
insert into SAILORS VALUES (31,'Lubber',8,55.5);
insert into SAILORS VALUES (32,'Andy',8,25.5);
insert into SAILORS VALUES (58,'Rusty',10,35.0);
insert into SAILORS VALUES (64,'Horatio',7,35.0);
insert into SAILORS VALUES (71,'Zorba',10,16.0);
insert into SAILORS VALUES (74,'Horatio',9,35.0);
insert into SAILORS VALUES (85,'Art',3,25.5);
insert into SAILORS VALUES (95,'Bob',3,63.5);
```

RESERVES TABLES:

```
insert into RESERVES VALUES (22,101,'1998-10-10');
insert into RESERVES VALUES (22,102, '1998-10-10');
insert into RESERVES VALUES (22,103, '1998-08-10');
insert into RESERVES VALUES (22,104, '1998-07-10');
insert into RESERVES VALUES (31,102, '1998-10-11');
insert into RESERVES VALUES (31,103, '1998-06-11');
insert into RESERVES VALUES (31,104, '1998-12-11');
insert into RESERVES VALUES (64,101, '1998-05-09');
insert into RESERVES VALUES (64,102, '1998-08-09');
insert into RESERVES  VALUES (74,103, '1998-08-09');
```

BOATS TABLES:

```
insert into BOATS VALUES (101,'Interlake','blue');
insert into BOATS VALUES (102,'Interlake','red');
insert into BOATS VALUES (103,'Clipper','green');
insert into BOATS  VALUES (104,'Marine','red');
```