# CSE 4308
# DBMS LAB
# Documentations

Prepared by,
Sabbir Ahmed
Lecturer,
Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)
Board Bazar, Gazipur-1704
Bangladesh.
Email: iamsabbirahmed12345@gmail.com , sabbirahmed@iut-dhaka.edu
Contact: +8801754221481, +8801630210656

# Table of Contents

# 1 LAB1

## 1.1 SQL BASICS

- How to open sql?

  Just write sqlplus in the command prompt!
  Or,
  Search 'run sql command line' in the program search bar.

- How to connect to a database?

  Open sql, write

  CONNECT USERNAME/PASSWORD;
  Or
  Type

  CONNECT

  Then there will be prompt for username. Input the username. Then it will ask for password. But now if u type the password, it will not show the characters as oracle keeps it hidden. After inputting the password hit enter and see the system will be connected.

- How to see existing users?

  First u have to connect to the database.

  SELECT USERNAME FROM DBA_USERS;

  To see the existing usernames sorted:

  SELECT USERNAME FROM DBA_USERS ORDER BY USERNAME;

  To see in which date the user was created:

  SELECT USERNAME, CREATED FROM DBA_USERS ORDER BY CREATED;

  The default order is ascending. Do make it appear in descending order you have to use the following query:

  SELECT USERNAME, CREATED FROM DBA_USERS ORDER BY CREATED DESC;
  (Here desc stands for Descending).

When the new user is created, if you try to login with it, Login will be denied as the super user (system) has not given him the privilege to login. There are some privileges in oracle like:

Create session,
Connect,
Resource, (to be able to create table and add data)
DBA, (database admin privilege)
Etc....

- How to grant session privilege?

  GRANT CREATE SESSION TO USERNAME;
  Or
  GRANT CREATE SESSION TO USERNAME WITH ADMIN OPTION.

  To give him both create session and resource privilege:

  GRANT CREATE SESSION, RESOURCE TO USERNAME;

  To give all possible privilege at a single go, you can write:

  GRANT ALL PRIVILEGES TO USERNAME;

  (This is strictly NOT RECOMMENDED as the new user might get some unwanted privileges!)

- How to alter the system password in case of u forgotten it?

  CONN / AS SYSDBA
  ALTER USER USERNAME IDENTIFIED BY NEWPASSWORD;

  Or

  CONN SYSTEM / PASSWORD AS SYSDBA
  ALTER USER USERNAME IDENTIFIED BY NEWPASSWORD;

- How to create a user in oracle?

  CREATE USER username IDENTIFIED BY password

- How to change password?

  ALTER USER username IDENTIFIED BY password;

  But for that you have to login as superuser! (System/sys/new user who has such privilege!)

- To drop a user:

  DROP USER USERNAME CASCADE;

Here cascade means all the objects (for example: tables) should also be deleted with the removal of the user.

- We can format time in different ways using sql.Some examples are showed below:

```
SQL> select to_char(created, 'DD-MON-YYYY') from dba_users;

TO_CHAR(CRE
-----------
23-JAN-2018
23-JAN-2018
07-FEB-2006
07-FEB-2006
07-FEB-2006
07-FEB-2006
07-FEB-2006
07-FEB-2006
07-FEB-2006
07-FEB-2006

TO_CHAR(CRE
-----------
07-FEB-2006
07-FEB-2006
07-FEB-2006
07-FEB-2006

15 rows selected.

SQL> select to_char(created, 'DD-MONTH-YYYY') from dba_users;

TO_CHAR(CREATED,'
-----------------
23-JANUARY  -2018
23-JANUARY  -2018
07-FEBRUARY -2006
07-FEBRUARY -2006
07-FEBRUARY -2006
07-FEBRUARY -2006
07-FEBRUARY -2006
07-FEBRUARY -2006
07-FEBRUARY -2006
07-FEBRUARY -2006
07-FEBRUARY -2006

TO_CHAR(CREATED,'
-----------------
07-FEBRUARY -2006
07-FEBRUARY -2006
07-FEBRUARY -2006
07-FEBRUARY -2006

15 rows selected.

SQL>
```

```
SQL> select to_char(created, 'DD-MON-YYYY HH24:MI:SS') from dba_users;

TO_CHAR(CREATED,'DD-
--------------------
23-JAN-2018 00:52:15
23-JAN-2018 00:42:39
07-FEB-2006 22:10:13
07-FEB-2006 22:10:13
07-FEB-2006 22:40:15
07-FEB-2006 22:44:47
07-FEB-2006 22:10:24
07-FEB-2006 22:17:03
07-FEB-2006 22:27:15
07-FEB-2006 22:52:43
07-FEB-2006 22:38:38

TO_CHAR(CREATED,'DD-
--------------------
07-FEB-2006 22:35:21
07-FEB-2006 22:52:43
07-FEB-2006 22:40:14
07-FEB-2006 22:51:21

15 rows selected.
```

# 2 LAB2:

## 2.1 Creating table:

If u want to connect sql from cmd, just type sqlplus!

- How to create a table?

  ```
  Create table dept (
  DeptID number,
  DeptName varchar2 (20),
  CONSTRAINTS PK_DEPT PRIMARY KEY (DEPTID)
  );
  ```

  Remember that sql is not case sensitive! But the values are

- To see the column names and their types in sql:

  ```
  DESC TABLE_NAME;
  ```

- How to show all the tables that are under a particular user?

  DESC DBA_TABLES; [this dba_tables is a table containing all the tables under each user.]

  ```
  SELECT TABLE_NAME
  FROM DBA_TABLES
  WHERE OWNER='USERNAME';
  ```

- Suppose you want to create a new table 'dept' but you are not sure whether the table is already created or not. If it is already existing, oracle will not let u create a new table with the same name.

  How to be sure that whether a table called 'dept' is existing or not?

  One solution might be deleting the existing table.

  ```
  DROP TABLE TABLE_NAME;
  ```

  But this is bad practice as the information might be necessary. So we can search

  ```
  SELECT TABLE_NAME
  FROM DBA_TABLES;
  ```

  But it will return a long list of the existing tables. It is difficult to search my desired table_name from the list.
  Another way is:

```
SELECT TABLE_NAME
FROM DBA_TABLES
WHERE TABLE_NAME='DEPT';
```

IF the dept table is existing, some rows will be selected from the table.

- How to disconnect a user?
  Just write DISC
- Syntax of primary key:

```
CONSTRAINT constraint_name PRIMARY KEY (column1, column2, ... column_n)
```

- How to add primary key to a table which is already created, but suppose u have forgotten to add the primary key?

  Creating a table without primary key:

```
CREATE TABLE STD (
SID NUMBER,
SNAME VARCHAR (20),
);
```

```
ALTER TABLE TABLE_NAME ADD CONSTRAINT CONSTRAINT_NAME PRIMARY KEY (COL_NAME);
```

- How to assign a composite primary key?

```
CONSTRAINT CONSTARINT_NAME PRIMARY KEY (COL1, COL2...);
```

- How to insert values to a table?

```
INSERT INTO table_name VALUES (value1, value2, value3, ...);
```

- You can also specify the column names:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Remember that sql is not case sensitive. But it is case sensitive in cases like saved values like passwords.

For inserting string/characters you have to put single quotes 'value'.

- How to update values of a table?

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

- HOW TO DELETE VALUES FROM A TABLE?

```
DELETE FROM table_name
WHERE condition;
```

```
DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste';
```

## 2.2 How to add foreign key to a table?

Suppose we have a table called 'country' which has the columns as c_name, c_id and c_id is the primary key.

Now If I want to create a table called 'citizen' with the following columns like citizenID, citizenName and countryID we can define citizenId as the primary key of the citizen table. But here countryID should be referencing the 'country' table. Thus here countryID in the 'citizen' table is acting as the foreign key.

SQL:

```
CREATE TABLE COUNTRY (
        C_NAME VARCHAR2 (30),
        C_id NUMBER,
        CONSTRAINTS PK_COUNTRY PRIMARY KEY (C_ID)
);

CREATE TABLE CITIZEN (
        CITIZENID NUMBER,
        CITIZENNAME VARCHAR2 (20),
        COUNTRYID NUMBER,
        CONSTRAINTS PK_CITIZEN PRIMARY KEY (COUNTRYID),
        CONSTRAINTS FK_CITIZEN_COUNTRY FOREIGN KEY (COUNTRYID)
        REFERNECES COUNTRY (C_ID)
);
```

REMEMBER THAT THE PARENT TABLE SHOULD ALWAYS BE CREATED FIRST!

And both of the columns (referencing and referenced) should be of the same types.

Now insert some values to both of the table. Here you'll see that if u want to add a data to citizen table with the c_id which is not present in the country table, it will show error. This means the foreign key is working properly.

- What will be the constraint if a table contains two foreign keys?

  There should be two foreign key statements referencing each to the tables!

  NOTE:
- Why do we give constraint names?

  Here pk_dept is the rule/constraint name. It has many uses. You can write anything in the place of pk_dept. but this is the widely practiced naming convention. For example if table named 'student' has the deptID as the foreign key, the name of that foreign key constraint might be given as 'fk_student_dept'.

  One of the use of this constraint name is, suppose you have inserted a value to the student table with student_id=1 where student_id is the primary key. Now if you add another student with the student_id=1, the command prompt will show that, 'pk_dept' rule is violated. In another case, suppose we want to add a row to student

table where the dept name is 'abc' which is not a valid department it is not entered in the dept table. So in this case command prompt will show that the 'fk_student_dept' constraint is violated.

## 2.3 Alter table

- How to add column to an existing table?

  ALTER TABLE *table_name*
  ADD *column_name datatype*;

- To delete column from an existing table:

  ALTER TABLE *table_name*
  DROP COLUMN *column_name*;

- How to change the datatype of an existing column?

  ALTER TABLE *table_name*
  MODIFY *column_name datatype*;

# 3 LAB4 & LAB5

## 3.1 Select query

- How to show all the values?

  Select * from table_name;

- How to select specific columns from a table?

  SELECT *column1*, *column2, ...*
  FROM *table_name*;

- To select distinct values from a table:

  SELECT DISTINCT C_HOME FROM CITIZEN;

  See the difference with

  SELECT C_HOME FROM CITIZEN;

- Renaming Output Column:

  In sql, when you write the above query, output will be a column with the name 'c_home' containing all the districts. We can rename the output column name for better visualization!
  [Renaming a column in the select query doesn't have any effect on the main table definition. It is only used for visualization]

  SYNTAX:

  SELECT COL_NAME AS NEW_COL_NAME FROM TABLE_NAME;

  EXAMPLE:

  SELECT C_HOME AS HOME_LIST
  FROM CITIZEN;

  SELECT DISTINCT (C_HOME) AS UNIQUE_DISTRICTS
  FROM CITIZEN;

- How can I count the number of unique districts?

  SELECT COUNT (DISTINCT (C_HOME) ) FROM CITIZEN;

  [In this case the output column name will be 'COUNT (DISTINCT (C_HOME) )' which looks odd! So we can rename the output as

  SELECT COUNT (DISTINCT (C_HOME) ) AS TOTAL_UNIQUE_DIST FROM CITIZEN;

Note:
Here count () is an aggregate function which counts the total number of columns presented by the query.

Note:

```
SELECT COUNT (DISTINCT (C_HOME) ) FROM CITIZEN;
```

This query can also be written as

```
SELECT COUNT (*) AS TOTAL_UNIQUE_HOME
FROM (SELECT DISTINCT (C_HOME) FROM CITIZEN);
```

[EXPLAINATION: this sort of query is called NESTED query. Here first the inner query in the FROM section is being calculated. Inner query first returns the names of the distinct district names which are 'Dhaka', 'Ctg', 'Comilla', 'Khulna', 'Gazipur'. Then the outer query count the number of rows supplied by the inner query. So output will be 5]

## 3.2 Adding condition to query

If u want to select some values based on a specific condition, u have to use 'where' clause as well!

```
SELECT COL1, COL2...
FROM TABLE_NAME
WHERE CONTIDION;
```

In the where clause you can use operators like, =, >, <, <>, >=, <=, BETWEEN, LIKE, IN etc.

i. Find the citizen names and id who are more than 50 years old

```
SELECT C_ID, C_NAME
FROM CITIZEN
WHERE AGE>50;
```

ii. Find the citizens living in Dhaka

```
SELECT C_NAME AS NAME
FROM CITIZEN
WHERE C_HOME='Dhaka';
```

iii. Find the persons not living in Dhaka

```
SELECT C_NAME, C_HOME FROM CITIZEN WHERE C_HOME<>'Dhaka';
```

< > means! =
[NOTE: here Dhaka is CASE SENSETIVE! ]

iv. Find the citizens having salary more than 20,000 and less than 50000

```
SELECT C_NAME, SALARY
FROM CITIZEN
WHERE SALARY BETWEEN 20000 AND 50000;
```

Suppose we want to see the values which are greater than 50000 and less than 20000, then we can use the 'NOT BETWEEN' clause.

```
SELECT C_NAME, SALARY
FROM CITIZEN
WHERE SALARY NOT BETWEEN 20000 AND 50000;
```

If you want to sort the output based on salary:

```
SELECT C_NAME, SALARY
FROM CITIZEN
WHERE SALARY NOT BETWEEN 20000 AND 50000 ORDER BY SALARY;
```

Default order of sorting is ascending. For Descending:

```
SELECT C_NAME, SALARY
FROM CITIZEN
```

```
WHERE SALARY NOT BETWEEN 20000 AND 50000
ORDER BY SALARY DESC;
```

[Note: this query should not necessarily be written in 3 lines, writing like 'SELECT C_ID, C_NAME FROM CITIZEN WHERE AGE>50;' will work also!]

The select clause may also contain arithmetic expressions involving operations +,-,*, /.

For example:

- Give the teachers 5% increment and show their new salary?

```
SELECT C_ID, C_NAME, SALARY*1.05
FROM CITIZEN
WHERE OCCUPATION='Teacher';
```

- How much money should I add to my budget if I want to give the teachers 5% bonus for EID?

```
SELECT SUM (SALARY*0.05) AS TOTAL_ADDITIONAL_BUDGET
FROM CITIZEN
WHERE OCCUPATION='Teacher';
```

## 3.3 SQL USING TWO/MORE TABLES [Natural join]

For better understanding, let's create the following two tables.

```
CREATE TABLE STD (
  STD_ID NUMBER (3),
  STD_NAME VARCHAR (2),
  STD_DEPT NUMBER (3),
  STD_CG NUMBER (3, 2),
  CONSTRAINTS PK_STD PRIMARY KEY (STD_ID),
  CONSTRAINTS FK_STD_DEPT FOREIGN KEY (STD_DEPT) REFERENCES DEPT (DEPT_ID)
);

CREATE TABLE DEPT (
  DEPT_ID NUMBER (3),
  DEPT_NAME VARCHAR (3),
  DEPT_BUILDING VARCHAR (3),
  DEPT_ESTD NUMBER (4),
  CONSTRAINTS PK_DEPT PRIMARY KEY (DEPT_ID)
);
```

Now suppose you need a list of students along with their dept_name.
Sql:

```
SELECT STD.STD_ID, DEPT.DEPT_NAME
FROM STD, DEPT
WHERE STD.STD_DEPT= DEPT.DEPT_ID;
```

In sql you can rename the tables as well!
The above query could be written also as:

```
SELECT S.STD_ID, D.DEPT_NAME
FROM STD S, DEPT D
WHERE S.STD_DEPT= D.DEPT_ID;
```

It makes writing queries easier!

## 3.4 USING AND, OR, NOT IS SQL

### 3.4.1 AND:

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Example:
Show the list who are from 'DHAKA' AND salary is a Teacher:

```
SELECT *
FROM CITIZEN
WHERE C_HOME='Dhaka' AND OCCUPATION='Teacher';
```

### 3.4.2 OR

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Example:

Show the people who are either a teacher or a doctor

```
SELECT *
FROM CITIZEN
WHERE OCCUPATION='Teacher' OR OCCUPATION='Doctor';
```

### 3.4.3 NOT

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

EXAMPLE:

Show the person not from Dhaka:

```
SELECT *
FROM CITIZEN
WHERE NOT C_HOME='Dhaka';
```

LET'S MIX UP THE CLAUSES!

Find the teachers from Dhaka or Ctg:

```
SELECT *
FROM CITIZEN
WHERE OCCUPATION='Teacher' AND (C_HOME='Dhaka' OR C_HOME='Ctg');
```

Show the teachers who are not from Dhaka:

```
SELECT *
FROM CITIZEN
WHERE OCCUPATION='Teacher' AND NOT C_HOME='Dhaka';
```

## 3.5 ORDER BY

Used to sort the output into ascending (default) or descending order.

SYNTAX:

```
SELECT COL1, COL2 . . .
FROM  TABLE
WHERE CONDITION
ORDER BY COL1, COL2... ASC|DESC;
```

Here the where clause is not necessary. In that case:

```
SELECT COL1, COL2. . .
FROM TABLE
ORDER BY COL1,COL2... ASC|DESC
```

# 3.6 Aggregate Functions

## 3.6.1 MIN ():

SYNTAX:

```
SELECT MIN (COL_NAME)
FROM TABLE_NAME;
```

EXAMPLE:

To find the person with the minimum salary:

```
SELECT MIN (SALARY)
FROM CITIZEN;
```

## 3.6.2 MAX ():

SYNTAX:

```
SELECT MAX (COL_NAME)
FROM TABLE_NAME;
```

EXAMPLE:

To find the person with maximum salary:

```
SELECT MAX (SALARY)
FROM CITIZEN;
```

We can always rename the column from MAX (SALARY) to 'desired name'

EXAMPLE:

```
SELECT MAX (SALARY) AS MAXIMUM_SALARY
FROM CITIZEN;
```

For selecting the maximum salary along with the salary holders name is quite a bit tricky!

```
SELECT C_NAME, MAX (SALARY)
FROM CITIZEN;
```

Will not work!

Because the c_name is returning all the c_name of citizen table and the max (salary) is returning only a single value. Thus oracle can't relate the two things!

What is the solution?

If we could find the max (salary) first and then match that salary with the table then return the name, the problem will be solved!

For this we need nested query!

```
SELECT C_NAME, SALARY
FROM CITIZEN
WHERE SALARY= (SELECT MAX (SALARY) FROM CITIZEN);
```

Here the (SELECT MAX (SALARY) FROM CITIZEN); executes first. It returns the maximum salary. Then the outer query matches this salary with all the salaries. Then the name is found! Tricky, Right?!

### 3.6.3 COUNT ():

COUNT () functions returns the number of rows matching a specific criteria.

EXAMPLE:

Find the number of teachers:

```
SELECT COUNT (c_id)
FROM CITIZEN
WHERE OCCUPATION='Teacher';
```

To find the number of rows present in a table:

```
SELECT COUNT (*)
FROM CITIZEN;
```

### 3.6.4 AVG ():

Find the average salary of the citizens:

```
SELECT AVG (SALARY)
FROM CITIZEN;
```

### 3.6.5 SUM ():

```
SELECT SUM (SALARY)
FROM CITIZEN;
```

Will return the addition of all the salaries of all citizens.

## 3.7 Aggregating and Grouping

There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples; we specify this wish in SQL using the **group by** clause. The attribute or attributes given in the **group by** clause are used to form groups. Tuples with the same value on all attributes in the **group by** clause are placed in one group.

Suppose you want the average salary based on occupations!

```
SELECT OCCUPATION, AVG (SALARY)
FROM CITIZEN
GROUP BY OCCUPATION;
```

## 3.8 Having Clause

Sometimes we might work with conditions which apply on a group of tuples but not individual ones. In that case we use the **Having** clause.
It only works for aggregate functions.

Group the citizens based on living place and show the number of citizens living in each district where at least 2 people belong to that district.

```
SELECT COUNT (C_HOME) AS NUMBER_OF_CITIZEN, C_HOME
FROM CITIZEN
GROUP BY C_HOME
HAVING COUNT (C_HOME)>2;
```

Here HAVING clause works similar to where clause. But the difference is 'where' clause works on each tuples, but 'having' clause works on a group of tuples.

## 3.9 USING ROWNUM

Suppose you want the top 5 salary holders. But using aggregate functions you can only get the maximum or minimum.
Here we want to sort the table and cut the top 5 rows. Right?

In these cases we use rownum.

```
SELECT *
FROM (SELECT * FROM CITIZEN ORDER BY SALARY)
WHERE ROWNUM<=5;
```

Here the inner sql in the form section returns a sorted table. The outer sql just takes out the first 5 rows of that table.

# 4 Referenced Tables

## 4.1 Citizen

```
CREATE TABLE CITIZEN (
        C_ID NUMBER (3),
        C_NAME VARCHAR2 (10),
        C_HOME VARCHAR2 (10),
        AGE NUMBER (2),
        OCCUPATION VARCHAR2 (15),
        GENDER VARCHAR2 (6),
        SALARY NUMBER,
        CONSTRAINTS PK_CITIZEN PRIMARY KEY (C_ID)
);

INSERT INTO CITIZEN VALUES (1, 'A', 'Dhaka', 25, 'Teacher', 'Male', 50000);
INSERT INTO CITIZEN VALUES (2, 'B', 'Dhaka', 56, 'Service', 'Male', 60000);
INSERT INTO CITIZEN VALUES (3, 'C', 'Ctg', 71, 'Retired', 'Male', 10000);
INSERT INTO CITIZEN VALUES (4, 'D', 'Ctg', 13, 'Student', 'Female', 500);
INSERT INTO CITIZEN VALUES (5, 'E', 'Dhaka', 45, 'Service', 'Male', 40000);
INSERT INTO CITIZEN VALUES (6, 'F', 'Gazipur', 54, 'Doctor', 'Female', 55000);
INSERT INTO CITIZEN VALUES (7, 'G', 'Gazipur', 65, 'Musician', 'Female', 5000);
INSERT INTO CITIZEN VALUES (8, 'H', 'Dhaka', 56, 'Engineer', 'Male', 60000);
INSERT INTO CITIZEN VALUES (9, 'I', 'Ctg', 23, 'Student', 'Male', 1000);
INSERT INTO CITIZEN VALUES (10, 'J', 'Comilla', 32, 'Teacher', 'Male', 45000);
INSERT INTO CITIZEN VALUES (11, 'K', 'Comilla', 51, 'Farmer', 'Male', 20000);
INSERT INTO CITIZEN VALUES (12, 'L', 'Khulna', 15, 'Student', 'Female', 1500);
INSERT INTO CITIZEN VALUES (13, 'M', 'Ctg', 25, 'Business', 'Male', 100000);
INSERT INTO CITIZEN VALUES (14, 'N', 'Comilla', 52, 'Doctor', 'Male', 70000);
INSERT INTO CITIZEN VALUES (15, 'O', 'Gazipur', 53, 'Teacher', 'Male', 50000);
INSERT INTO CITIZEN VALUES (16, 'P', 'Dhaka', 35, 'Musician', 'Female', 50000);
INSERT INTO CITIZEN VALUES (17, 'Q', 'Khulna', 43, 'Service', 'Male', 50000);
INSERT INTO CITIZEN VALUES (18, 'R', 'Khulna', 34, 'Service', 'Female', 45000);
INSERT INTO CITIZEN VALUES (19, 'S', 'Ctg', 16, 'Student', 'Male', 500);
```

## 4.2 STD

```
CREATE TABLE STD (
        STD_ID NUMBER (3),
        STD_NAME VARCHAR (2),
        STD_DEPT NUMBER (3),
        STD_CG NUMBER (3, 2),
        CONSTRAINTS PK_STD PRIMARY KEY (STD_ID),
        CONSTRAINTS FK_STD_DEPT FOREIGN KEY (STD_DEPT) REFERENCES DEPT (DEPT_ID)
);

INSERT INTO STD VALUES (41,'A', 101, 3.5);
INSERT INTO STD VALUES (42,'B', 102, 3.6);
INSERT INTO STD VALUES (43,'C', 103, 3.7);
INSERT INTO STD VALUES (44,'D', 101, 3.8);
```

# 4.3 Student

```
CREATE TABLE STUDENT (
        S_ID NUMBER (2),
        S_NAME VARCHAR (3),
        S_HOME VARCHAR (10),
        S_DEPT NUMBER (3),
        S_SUP_ID NUMBER (4),
        S_CG NUMBER (3, 2),
        CONSTRAINTS PK_STUDENT PRIMARY KEY (S_ID),
        CONSTRAINTS FK_STUDENT_DEPT FOREIGN KEY (S_DEPT) REFERENCES DEPT
        (DEPT_ID),
        CONSTRAINTS FK_STUDENT_SUPERVISOR FOREIGN KEY (S_SUP_ID) REFERENCES
        SUPERVISOR (SUP_ID)
);


INSERT INTO STUDENT VALUES (1,'SA','DHAKA', 101, 1001, 3.5);
INSERT INTO STUDENT VALUES (2,'SB','CTG', 102, 1002, 3.6);
INSERT INTO STUDENT VALUES (3,'SC','DHAKA', 103, 1003, 3.7);
INSERT INTO STUDENT VALUES (4,'SD','COMILLA', 104, 1004, 3.8);
INSERT INTO STUDENT VALUES (5,'SE','SYLHET', 105, 1005, 3.5);
INSERT INTO STUDENT VALUES (6,'SF','DHAKA', 101, 1006, 3.9);
INSERT INTO STUDENT VALUES (7,'SG','RAJSHAHI', 101, 1001, 3.6);
```

# 4.4 DEPT

```
CREATE TABLE DEPT (
        DEPT_ID NUMBER (3),
        DEPT_NAME VARCHAR (3),
        DEPT_BUILDING VARCHAR (3),
        DEPT_ESTD NUMBER (4),
        CONSTRAINTS PK_DEPT PRIMARY KEY (DEPT_ID)
);


INSERT INTO DEPT VALUES (101,'CSE','NEW', 1998);
INSERT INTO DEPT VALUES (102,'SWE','NEW', 2018);
INSERT INTO DEPT VALUES (103,'MCE','OLD', 1998);
INSERT INTO DEPT VALUES (104,'EEE','OLD', 1998);
INSERT INTO DEPT VALUES (105,'CEE','OLD', 2009);
INSERT INTO DEPT VALUES (106,'BTM','OLD', 2018);
```

## 4.5 Supervisor

```
CREATE TABLE SUPERVISOR (
        SUP_ID NUMBER (4),
        SUP_STAFF_ID NUMBER (5),
        SUP_DEPT NUMBER (3),
        SUP_DPET_ID NUMBER (3),
        CONSTRAINTS PK_SUP PRIMARY KEY (SUP_ID),
        CONSTRAINTS FK_SUP_DEPT FOREIGN KEY (SUP_DEPT) REFERENCES DEPT (DEPT_ID),
        CONSTRAINTS FK_SUP_STAFF FOREIGN KEY (SUP_STAFF_ID) REFERENCE STAFF
        (STAFF_ID)
);
```

# 4.6 Manufacturers

```
CREATE TABLE Manufacturers (
        Code INTEGER,
        Name VARCHAR (255) NOT NULL,
        PRIMARY KEY (Code)
);


INSERT INTO Manufacturers (Codename) VALUES (1,'Sony');
INSERT INTO Manufacturers (Codename) VALUES (2,'Creative Labs');
INSERT INTO Manufacturers (Codename) VALUES (3,'Hewlett-Packard');
INSERT INTO Manufacturers (Codename) VALUES (4,'Iomega');
INSERT INTO Manufacturers (Codename) VALUES (5,'Fujitsu');
INSERT INTO Manufacturers (Codename) VALUES (6,'Winchester');
```

# 4.7 Products

```
CREATE TABLE Products (
        Code INTEGER,
        Name VARCHAR (255) NOT NULL,
        Price DECIMAL NOT NULL,
        Manufacturer INTEGER NOT NULL,
        PRIMARY KEY (Code),
        FOREIGN KEY (Manufacturer) REFERENCES Manufacturers (Code)
);


INSERT INTO Products (Code, Name, Price, and Manufacturer) VALUES (1,'Hard drive', 240,
5);
INSERT INTO Products (Code, Name, Price, and Manufacturer) VALUES (2,'Memory', 120, 6);
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(3,'ZIP drive',150,4);
INSERT INTO Products (Code, Name, Price, and Manufacturer) VALUES (4,'Floppy disk', 5, 6);
INSERT INTO Products (Code, Name, Price, and Manufacturer) VALUES (5,'Monitor', 240, 1);
INSERT INTO Products (Code, Name, Price, Manufacturer) VALUES (6,'DVD drive', 180, 2);
INSERT INTO Products (Code, Name, Price, Manufacturer) VALUES (7,'CD drive', 90, 2);
INSERT INTO Products (Code, Name, Price, and Manufacturer) VALUES (8,'Printer', 270, 3);
INSERT INTO Products (Code, Name, Price, and Manufacturer) VALUES (9,'Toner cartridge',
66, 3);
INSERT INTO Products (Code, Name, Price, and Manufacturer) VALUES (10,'DVD burner', 180,
2);
INSERT INTO Products (Code, Name, Price, and Manufacturer) VALUES (11,'Card Reader',
180, 2);
```