|  |  |  |
| --- | --- | --- |
| **CIS 422 DBMS** | **LAB 01** | **Points: 20** |
| **SQL Training and Exercises** | | |

**Elijah Chong Tay**

**Introduction:**

SQL is a programming language designed to manage data stored in a relational database management system (RDBMS). SQL stands for the structured query language. SQL consists of a data definition language, data manipulation language, and a data control language.

* The data definition language deals with the schema creation and modification e.g., CREATE TABLE statement allows you to create a new table in the database and the ALTER TABLE statement changes the structure of an existing table.
* The data manipulation language provides the constructs to query data such as the SELECT statement and to update the data such as INSERT, UPDATE, and DELETE statements.
* The data control language consists of the statements that deal with the user authorization and security such as GRANT and REVOKE statements.

It is worth noting that the community constantly requests new features and capabilities that do not exist in the SQL standard yet, therefore, even with the SQL standard in place, there are many SQL dialects in various database products.

**SQL Syntax:**

SQL is a declarative language; therefore, its syntax reads like a natural language. An SQL statement begins with a verb that describes the action, for example, SELECT, INSERT, UPDATE or DELETE. Following the verb are the subject and predicate.

* SQL commands

SQL is made up of many commands. Each SQL command is typically terminated with a semicolon (;). For example, the following are two different SQL commands separated by a semicolon (;):

**SELECT**

**first\_name, last\_name**

**FROM**

**employees;**

**DELETE FROM employees**

**WHERE**

**hire\_date < '1990-01-01';**

SQL uses the semicolon (;) to mark the end of a command. Each command is composed of tokens that can be literals, keywords, identifiers, or expressions. Tokens are separated by space, tabs, or newlines.

* Literals

Literals are explicit values which are also known as constants. SQL provides three kinds of literals: string, numeric, and binary.

* String literal consists of one or more alphanumeric characters surrounded by single quotes, for example:

**'John'**

**'1990-01-01'**

**'50'**

Typically, SQL is case sensitive with respect to string literals, so the value **'John'** is not the same as **'JOHN'**.

* Numeric literals are the integer, decimal, or scientific notation, for example:

**200**

**-5**

**6.0221415E23**

* SQL represents binary value using the notation x'0000', where each digit is hexadecimal value, for example:

**x'01'**

**x'0f0ff'**

* Keywords

SQL has many keywords that have special meanings such as SELECT, INSERT, UPDATE, DELETE, and DROP. These keywords are the reserved words, therefore, you cannot use them as the name of tables, columns, indexes, views, stored procedures, triggers, or other database objects.

* Identifiers

Identifiers refer to specific objects in the database such as tables, columns, indexes, etc. SQL is case-insensitive with respect to keywords and identifiers. The following statements are equivalent.

**Select \* From employees;**

**SELECT \* FROM EMPLOYEES;**

**select \* from employees;**

**SELECT \* FROM employees;**

To make the SQL commands more readable and clear, it’s highly recommended to use the SQL keywords in uppercase and identifiers in lower case.

* Comments

In SQL a comment is denoted by two consecutive hyphens (--) that allow you to comment the remaining line. See the following example.

**SELECT**

**employee\_id, salary**

**FROM**

**employees**

**WHERE**

**salary < 3000; -- employees with low salary**

To document the code that can span multiple lines, you use the multiline C-style notation ( /\*\*/) as the shown in the following statement:

**/\* increase 5% for employees whose salary is less than 3,000 \*/**

**UPDATE employees**

**SET**

**salary = salary \* 1.05**

**WHERE**

**salary < 3000;**

**Working with table structures**

* CREATE TABLE:

A table is a collection of data stored in a database. A table consists of columns and rows. To create a new table, you use the CREATE TABLE statement with the following syntax:

**CREATE TABLE table\_name(**

**column\_name\_1 data\_type default value column\_constraint,**

**column\_name\_2 data\_type default value column\_constraint,**

**...,**

**table\_constraint**

**);**

In the CREATE TABLE statement, you specify a comma-separated list of column definitions. Each column definition is composed of a column name, column’s data type, a default value, and one or more column constraints.

* The data type of a column specifies the type of data that column can store. The data

type of the column can be the numeric, characters, date, etc.

* The column constraint controls what kind of value that can be stored in the column. For example, the NOT NULL constraint ensures that the column does not contain any NULL value.
* A column may have multiple column constraints. For example, the username column of the users table can have both NOT NULL and UNIQUE constraints.
* In case a constraint contains multiple columns, you use the table constraint. For example, if a table has the primary key that consists of two columns, in this case, you have to use the PRIMARY KEY table constraint.
* ALTER TABLE
* Add column:

To modify the structure of a table, you use the ALTER TABLE statement. The ALTER TABLE statement allows you to perform the following operations on an existing table:

* Add a new column using the ADD clause.
* Modify attribute of a column such as constraint, default value, etc. using the MODIFY clause.
* Remove columns using the DROP clause.

The following statement illustrates the ALTER TABLE with the ADD clause that allows you to add one or more columns to a table:

**ALTER TABLE table\_name**

**ADD new\_colum data\_type column\_constraint [AFTER existing\_column];**

Note: If you omit the AFTER clause, all the new columns will be added after the last column of the table.

* Modify column:

The MODIFY clause allows you to change some attributes of the existing column e.g., NOT NULL ,UNIQUE, and data type. The following statement shows you the syntax of the ALTER TABLE statement with the DROP clause.

**ALTER TABLE table\_name**

**MODIFY column\_definition;**

* DROP columns

When a column of a table is obsolete and not used by any other database objects such as triggers, views, stored and stored procedures, you need to remove it from the table. To remove one or more columns, you use the following syntax:

**ALTER TABLE table\_name**

**DROP column\_name,**

**DROP colum\_name,**

**...**

**Constraints**

* **NOT NULL**

The NOT NULL constraint is a column constraint that ensures values stored in a column are not NULL. The syntax of defining a NOT NULL constraint is as follows:

**column\_name data\_type NOT NULL;**

Logically, a NOT NULL constraint is equivalent to a CHECK constraint, therefore, the above statement is equivalent to the following statement.

**column\_name data\_type,**

**CHECK (column\_name IS NOT NULL)**

Typically, you add NOT NULL constraints to columns when you create the table. Sometimes, you want to add a NOT NULL constraint to a NULL-able column of an existing table. The syntax of adding a NOT NULL constraint is as follows:

**ALTER TABLE table\_name**

**MODIFY column\_name data\_type NOT NULL;**

* **UNIQUE**

SQL UNIQUE constraint defines a rule that prevents duplicate values stored in specific columns that do not participate a primary key.

Sometimes, you want to make sure that the values in a column or a set of columns are not duplicate. For example, duplicate emails in the employees table are not acceptable. Since the email column is not the part of the primary key, the only way to prevent duplicate values in the email column is to use a UNIQUE constraint. The syntax of defining a UNIQE (key) constraint is as follows:

**column\_name data\_type UNIQE;**

**or**

**CONSTRAINT constraint\_name UNIQUE (column\_name1, …)**

Example:

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) NOT NULL,

password VARCHAR(255) NOT NULL,

CONSTRAINT uc\_username UNIQUE (username)

);

Suppose the users table was created without the UNIQUE constraint defined for the username column. To add the UNIQUE constraint to the username column, you use the ALTER TABLE statement as follows:

ALTER TABLE table\_name

ADD CONSTRAINT constraint\_name UNIQUE(column\_name1);

* **Primary key**

Typically, a table has a column or set of columns whose values uniquely identify each row in the table. This column or the set of columns is called the primary key. The primary key that consists of two or more columns is also known as the composite primary key. Each table has one and only one primary key. The primary key does not accept NULL or duplicate values.

Generally, you define the primary key when creating the table. If the primary key consists of one column, you can use the PRIMARY KEY constraint as a column or table constraint as follows:

**Attribute\_name datatype PRIMARY KEY,**

If the primary key consists of two or more columns, you must use the PRIMARY KEY constraint as the table constraint as follows:

**CONSTRAINT name\_of\_constraint PRIMARY KEY (attribute\_name1, attribute\_name2, …)**

* **Foreign Key Constraint**

A foreign key is a column or a group of columns that enforces a link between the data in two tables. In a foreign key reference, the primary key column (or columns) of the first table is referenced by the column (or columns) of the second table. The column (or columns) of the second table becomes the foreign key. You use the FOREIGN KEY constraint to create a foreign key when you create or alter table.

To add a FOREIGN KEY constraint to existing table, you use the ALTER ABLE statement.

**ALTER TABLE table\_1**

**ADD CONSTRAINT fk\_name FOREIGN KEY (fk\_key\_column)**

**REFERENCES table\_2(pk\_key\_column)**

* **CHECK constraint**

A CHECK constraint is an integrity constraint in SQL that allows you to specify that a value in a column or set of columns must satisfy a Boolean expression. You can define a CHECK constraint on a single column or the whole table. If you define the CHECK constraint on a single column, the CHECK constraint checks value for this column only. However, if you define a CHECK constraint on a table, it limits value in a column based on values in other columns of the same row.

The CHECK constraint consists of the keyword CHECK followed by a Boolean expression in parentheses:

**CHECK(Boolean\_expression)**

To assign a CHECK constraint a name, you use the following syntax:

**CONSTRAINT constraint\_name CHECK(Boolean\_expression)**

**Modifying data**

* INSERT:

SQL provides the INSERT statement that allows you to insert one or more rows into a table. The INSERT statement allows you to:

* Insert a single row into a table

To insert one row into a table, you use the following syntax of the INSERT statement:

**INSERT INTO table1 (column1, column2,...)**

**VALUES**

**(value1, value2,...);**

* Insert multiple rows into a table

To insert multiple rows using a single INSERT statement, you use the following construct:

**INSERT INTO table1**

**VALUES**

**(value1, value2,...),**

**(value1, value2,...),**

**(value1, value2,...),**

**...;**

* Copy rows from a table to another table.

You can use the INSERT statement to query data from one or more tables and insert it into another table as follows:

**INSERT INTO table1 (column1, column2)**

**SELECT**

**column1,**

**column2**

**FROM**

**table2**

**WHERE**

**condition1;**

* UPDATE

To change existing data in a table, you use the UPDATE statement. The following shows the syntax of the UPDATE statement:

**UPDATE table\_name**

**SET column1 = value1,**

**column2 = value2**

**WHERE**

**condition;**

The UPDATE statement affects one or more rows in a table based on the condition in the WHERE clause. Any row that causes the condition in the WHERE to evaluate to true will be modified. Because the WHERE clause is optional, therefore, if you omit it, the all the rows in the table will be affected.

**Worm up: Create a Database and tables.**

In this section, you will create Database called HR that manages the HR data of the small businesses. The HR sample database has seven tables as follows:

* The **employees** table stores the data of employees.
* The **jobs** table stores the job data including job title and salary range.
* The **departments** table stores department data.
* The **dependents** table stores the employee’s dependents.
* The **locations** table stores the location of the departments of the company.
* The **countries** table stores the data of countries where the company is doing business.
* The **regions** table stores the data of regions such as Asia, Europe, America, and the Middle East and Africa. The countries are grouped into regions.

Create a database called HR.

***Note: if you want to use the campus database server, don’t create a database since it’s not allowed, just use the database created for you on the server)***

Create the tables as follows:

Regions:

region\_id: integer 11 digits, auto increment, primary key. region\_name: 25 char, default null.

Countries

country\_id: 2 char, primary key.

country\_name: 40 char, default null.

region\_id: integer 11 digits, not null.

FOREIGN KEY to regions table: ON DELETE CASCADE ON UPDATE CASCADE.

locations

location\_id: integer 11 digits, auto increment, primary key. street\_address: 40 chars, default null.

postal\_code: 12 char default null.

city: 30 char, NOT NULL.

state\_province: 25 char, default null.

country\_id: 2 char, not null.

FOREIGN KEY to countries table: ON DELETE CASCADE ON UPDATE CASCADE.

jobs

job\_id: 11 integer, auto increment, primary key.

job\_title:35 char, not null,

min\_salary: decimal (8, 2), default null.

max\_salary: decimal (8, 2), default null.

departments

department\_id: integer 11 digits, auto increment, primary key.

department\_name: 30 char, not null.

location\_id: integer 11 digits, default null.

Forgen key to locations: ON DELETE CASCADE ON UPDATE CASCADE.

employees

employee\_id: integer 11 digints, auto increment.

first\_name: 20 char, default null.

last\_name: 25 char, not null.

email: 100 char, not null.

phone\_number: 20 char, default null.

hire\_date: date, not null.

job\_id: integer 11 digits, not null.

salary: decimal (8, 2) not null.

manager\_id: integer 11 digints, default null.

department\_id: integer 11 digits, default null.

Forgen key to jobs: ON DELETE CASCADE ON UPDATE CASCADE.

Forgen key to departments: ON DELETE CASCADE ON UPDATE CASCADE.

Forgen key to employees:

dependents

dependent\_id: integer 11 digits, auto increment, primary key.

first\_name: char (50) not null.

last\_name: char (50) not null.

relationship: char (25) not null.

employee\_id: integer (11) not null.

Forgen key to employees: ON DELETE CASCADE ON UPDATE CASCADE.

Insert the following data into the tables:

**Data for the table regions**

1,'Europe'

2,'Americas'

3,'Asia'

4,'Middle East and Africa'

**Data for the table countries**

'AR','Argentina',2

'AU','Australia',3

'BE','Belgium',1

'BR','Brazil',2

'CA','Canada',2

'CH','Switzerland',1

'CN','China',3

'DE','Germany',1

'DK','Denmark',1

'EG','Egypt',4

'FR','France',1

'HK','HongKong',3

'IL','Israel',4

'IN','India',3

'IT','Italy',1

'JP','Japan',3

'KW','Kuwait',4

'MX','Mexico',2

'NG','Nigeria',4

'NL','Netherlands',1

'SG','Singapore',3

'UK','United Kingdom',1

'US','United States of America',2

'ZM','Zambia',4

'ZW','Zimbabwe',4

**Data for the table locations**

1400,'2014 Jabberwocky Rd','26192','Southlake','Texas','US'

1500,'2011 Interiors Blvd','99236','South San Francisco','California','US'

1700,'2004 Charade Rd','98199','Seattle','Washington','US'

1800,'147 Spadina Ave','M5V 2L7','Toronto','Ontario','CA'

2400,'8204 Arthur St',NULL,'London',NULL,'UK'

2500,'Magdalen Centre, The Oxford Science Park','OX9 9ZB','Oxford','Oxford','UK'

2700,'Schwanthalerstr. 7031','80925','Munich','Bavaria','DE'

**Data for the table jobs**

1,'Public Accountant',4200.00,9000.00

2,'Accounting Manager',8200.00,16000.00

3,'Administration Assistant',3000.00,6000.00

4,'President',20000.00,40000.00

5,'Administration Vice President',15000.00,30000.00

6,'Accountant',4200.00,9000.00

7,'Finance Manager',8200.00,16000.00

8,'Human Resources Representative',4000.00,9000.00

9,'Programmer',4000.00,10000.00

10,'Marketing Manager',9000.00,15000.00

11,'Marketing Representative',4000.00,9000.00

12,'Public Relations Representative',4500.00,10500.00

13,'Purchasing Clerk',2500.00,5500.00

14,'Purchasing Manager',8000.00,15000.00

15,'Sales Manager',10000.00,20000.00

16,'Sales Representative',6000.00,12000.00

17,'Shipping Clerk',2500.00,5500.00

18,'Stock Clerk',2000.00,5000.00

19,'Stock Manager',5500.00,8500.00

**Data for the table departments**

1,'Administration',1700

2,'Marketing',1800

3,'Purchasing',1700

4,'Human Resources',2400

5,'Shipping',1500

6,'IT',1400

7,'Public Relations',2700

8,'Sales',2500

9,'Executive',1700

10,'Finance',1700

11,'Accounting',1700

**Data for the table employees**

100,'Steven','King','steven.king@sqltutorial.org','515.123.4567','1987-06-17',4,24000.00,NULL,9

101,'Neena','Kochhar','neena.kochhar@sqltutorial.org','515.123.4568','1989-09-21',5,17000.00,100,9

102,'Lex','De Haan','lex.de haan@sqltutorial.org','515.123.4569','1993-01-13',5,17000.00,100,9

103,'Alexander','Hunold','alexander.hunold@sqltutorial.org','590.423.4567','1990-01-03',9,9000.00,102,6

104,'Bruce','Ernst','bruce.ernst@sqltutorial.org','590.423.4568','1991-05-21',9,6000.00,103,6

105,'David','Austin','david.austin@sqltutorial.org','590.423.4569','1997-06-25',9,4800.00,103,6

106,'Valli','Pataballa','valli.pataballa@sqltutorial.org','590.423.4560','1998-02-05',9,4800.00,103,6

107,'Diana','Lorentz','diana.lorentz@sqltutorial.org','590.423.5567','1999-02-07',9,4200.00,103,6

108,'Nancy','Greenberg','nancy.greenberg@sqltutorial.org','515.124.4569','1994-08-17',7,12000.00,101,10

109,'Daniel','Faviet','daniel.faviet@sqltutorial.org','515.124.4169','1994-08-16',6,9000.00,108,10

110,'John','Chen','john.chen@sqltutorial.org','515.124.4269','1997-09-28',6,8200.00,108,10

111,'Ismael','Sciarra','ismael.sciarra@sqltutorial.org','515.124.4369','1997-09-30',6,7700.00,108,10

112,'Jose Manuel','Urman','jose manuel.urman@sqltutorial.org','515.124.4469','1998-03-07',6,7800.00,108,10

113,'Luis','Popp','luis.popp@sqltutorial.org','515.124.4567','1999-12-07',6,6900.00,108,10

114,'Den','Raphaely','den.raphaely@sqltutorial.org','515.127.4561','1994-12-07',14,11000.00,100,3

115,'Alexander','Khoo','alexander.khoo@sqltutorial.org','515.127.4562','1995-05-18',13,3100.00,114,3

116,'Shelli','Baida','shelli.baida@sqltutorial.org','515.127.4563','1997-12-24',13,2900.00,114,3

117,'Sigal','Tobias','sigal.tobias@sqltutorial.org','515.127.4564','1997-07-24',13,2800.00,114,3

118,'Guy','Himuro','guy.himuro@sqltutorial.org','515.127.4565','1998-11-15',13,2600.00,114,3

119,'Karen','Colmenares','karen.colmenares@sqltutorial.org','515.127.4566','1999-08-10',13,2500.00,114,3

120,'Matthew','Weiss','matthew.weiss@sqltutorial.org','650.123.1234','1996-07-18',19,8000.00,100,5

121,'Adam','Fripp','adam.fripp@sqltutorial.org','650.123.2234','1997-04-10',19,8200.00,100,5

122,'Payam','Kaufling','payam.kaufling@sqltutorial.org','650.123.3234','1995-05-01',19,7900.00,100,5

123,'Shanta','Vollman','shanta.vollman@sqltutorial.org','650.123.4234','1997-10-10',19,6500.00,100,5

126,'Irene','Mikkilineni','irene.mikkilineni@sqltutorial.org','650.124.1224','1998-09-28',18,2700.00,120,5

145,'John','Russell','john.russell@sqltutorial.org',NULL,'1996-10-01',15,14000.00,100,8

146,'Karen','Partners','karen.partners@sqltutorial.org',NULL,'1997-01-05',15,13500.00,100,8

176,'Jonathon','Taylor','jonathon.taylor@sqltutorial.org',NULL,'1998-03-24',16,8600.00,100,8

177,'Jack','Livingston','jack.livingston@sqltutorial.org',NULL,'1998-04-23',16,8400.00,100,8

178,'Kimberely','Grant','kimberely.grant@sqltutorial.org',NULL,'1999-05-24',16,7000.00,100,8

179,'Charles','Johnson','charles.johnson@sqltutorial.org',NULL,'2000-01-04',16,6200.00,100,8

192,'Sarah','Bell','sarah.bell@sqltutorial.org','650.501.1876','1996-02-04',17,4000.00,123,5

193,'Britney','Everett','britney.everett@sqltutorial.org','650.501.2876','1997-03-03',17,3900.00,123,5

200,'Jennifer','Whalen','jennifer.whalen@sqltutorial.org','515.123.4444','1987-09-17',3,4400.00,101,1

201,'Michael','Hartstein','michael.hartstein@sqltutorial.org','515.123.5555','1996-02-17',10,13000.00,100,2

202,'Pat','Fay','pat.fay@sqltutorial.org','603.123.6666','1997-08-17',11,6000.00,201,2

203,'Susan','Mavris','susan.mavris@sqltutorial.org','515.123.7777','1994-06-07',8,6500.00,101,4

204,'Hermann','Baer','hermann.baer@sqltutorial.org','515.123.8888','1994-06-07',12,10000.00,101,7

205,'Shelley','Higgins','shelley.higgins@sqltutorial.org','515.123.8080','1994-06-07',2,12000.00,101,11

206,'William','Gietz','william.gietz@sqltutorial.org','515.123.8181','1994-06-07',1,8300.00,205,11

**Data for the table dependents**

1,'Penelope','Gietz','Child',206

2,'Nick','Higgins','Child',205

3,'Ed','Whalen','Child',200

4,'Jennifer','King','Child',100

5,'Johnny','Kochhar','Child',101

6,'Bette','De Haan','Child',102

7,'Grace','Faviet','Child',109

8,'Matthew','Chen','Child',110

9,'Joe','Sciarra','Child',111

10,'Christian','Urman','Child',112

11,'Zero','Popp','Child',113

12,'Karl','Greenberg','Child',108

13,'Uma','Mavris','Child',203

14,'Vivien','Hunold','Child',103

15,'Cuba','Ernst','Child',104

16,'Fred','Austin','Child',105

17,'Helen','Pataballa','Child',106

18,'Dan','Lorentz','Child',107

19,'Bob','Hartstein','Child',201

20,'Lucille','Fay','Child',202

21,'Kirsten','Baer','Child',204

22,'Elvis','Khoo','Child',115

23,'Sandra','Baida','Child',116

24,'Cameron','Tobias','Child',117

25,'Kevin','Himuro','Child',118

26,'Rip','Colmenares','Child',119

27,'Julia','Raphaely','Child',114

28,'Woody','Russell','Child',145

29,'Alec','Partners','Child',146

30,'Sandra','Taylor','Child',176

**Lab Exercises**

***Please for each of the following question, provide the SQL statement and the screenshot for the result from the phpMyAdmin.***

* Create the following two tables as follows:

Courses:

    course\_id: integer, auto increment, primary key.

    course\_name: 50 chars, not null.

Trainings:

    employee\_id integer, primary key.

    course\_id integer, primary key.

    taken\_date date.

CREATE TABLE courses(

course\_id INT(8) PRIMARY KEY,

course\_name INT(50) NOT NULL

);

CREATE TABLE trainings (

employee\_id INT(8),

course\_id INT(8),

taken\_date DATE,

PRIMARY KEY(employee\_id,course\_id)

);

* Add a new column named **credit\_hours** to the **courses** table as follows:

credit\_hours: integer, not null.

ALTER TABLE courses

ADD credit\_hours INT(2) NOT NULL

* Add the **fee** and **max\_limit** columns to the **courses** table and places these columns after the **course\_name** column in one statement.

Fee: NUMERIC (10, 2).

max\_limit: integer.

ALTER TABLE courses

ADD fee NUMERIC(10,2),

ADD max\_limit INT(6);

* Change the attribute of the **fee** column in the **courses** table to NOT NULL?

ALTER TABLE courses

ALTER COLUMN fee NUMERIC(10,2) NOT NULL;

* Remove the **fee, max\_limit** and **credit\_hours** of the **courses** table?

ALTER TABLE courses

DROP COLUMN fee,

DROP COLUMN max\_limit,

DROP COLUMN credit\_hours;

* Create the **projects** and **project\_assignments** to manage the projects and project assignments of the company in the database?

Projects:

project\_id: integer, primary key

project\_name 255 char.

start\_date: date, not null.

end\_date: date, not null.

project\_assignments

project\_id: integer, primary key

employee\_id: integer, primary key

join\_date: date, not null.

CREATE TABLE Projects(

project\_id INTEGER(6) PRIMARY KEY,

project\_name VARCHAR(255),

start\_date DATE NOT NULL,

end\_date DATE NOT NULL

);

CREATE TABLE Project\_assignments(

project\_id INTEGER(6),

employee\_id INTEGER(6),

join\_date DATE NOT NULL,

PRIMARY KEY(project\_id, employee\_id)

);

* Create new table named **project\_ milestones**. Each project may have zero or more milestones while one milestone must belong to one and only one project. The application that uses these tables must ensure that for each row in the **project\_milestones** table there exists the corresponding row in the **projects** table. In other words, a milestone cannot exist without a project.

**project\_milestones**:

milestone\_id: integer, auto increment, primary key.

project\_id: integer.

milestone\_name: 100 char.

FOREIGN KEY to projects table.

CREATE TABLE project\_milestones (

milestone\_id INTEGER (6) PRIMARY KEY,

project\_id INTEGER(6),

milestone\_name VARCHAR(100),

FOREIGN KEY (project\_id) REFERENCES projects(project\_id)

);

* Create a new table named **products** as follows:

Products:

product\_id: integer, primary key.

product\_name: 255 char, not null.

selling\_price NUMERIC(10,2), must be positive.

cost: NUMERIC (10, 2), must be positive. CHECK (cost > 0).

selling\_price must be greater than cost.

CREATE TABLE products (

product\_id INTEGER(6) PRIMARY KEY,

product\_name VARCHAR(255),

selling\_price NUMERIC(10,2)

);

ALTER TABLE products

ADD CHECK (selling\_price>0);

* Insert the following data into the **dependents** table:

**'Cameron', 'Bell', 'Child', 192**

**'Michelle', 'Bell', 'Child', 192**

**INSERT INTO dependents (**

**first\_name,**

**last\_name,**

**relationship,**

**employee\_id**

**)**

**VALUES**

**(**

**'Cameron',**

**'Bell',**

**'Child',**

**192**

**),**

**(**

**'Michelle',**

**'Bell',**

**'Child',**

**192**

**);**

* Create a new table called **dependents\_archive** that has the same structure as the **dependents** table. Then copy all rows from the dependents table to the **dependents\_archive** table.

CREATE TABLE dependents\_archive SELECT \* FROM dependents;

* Update Sarah’s last name from Bell to Lopez in **employees** table.

UPDATE employees

SET

last\_name = 'Lopez'

WHERE

employee\_id = 192;