

Birla Institute of Technology and Science, Pilani

CS F212 Database Systems

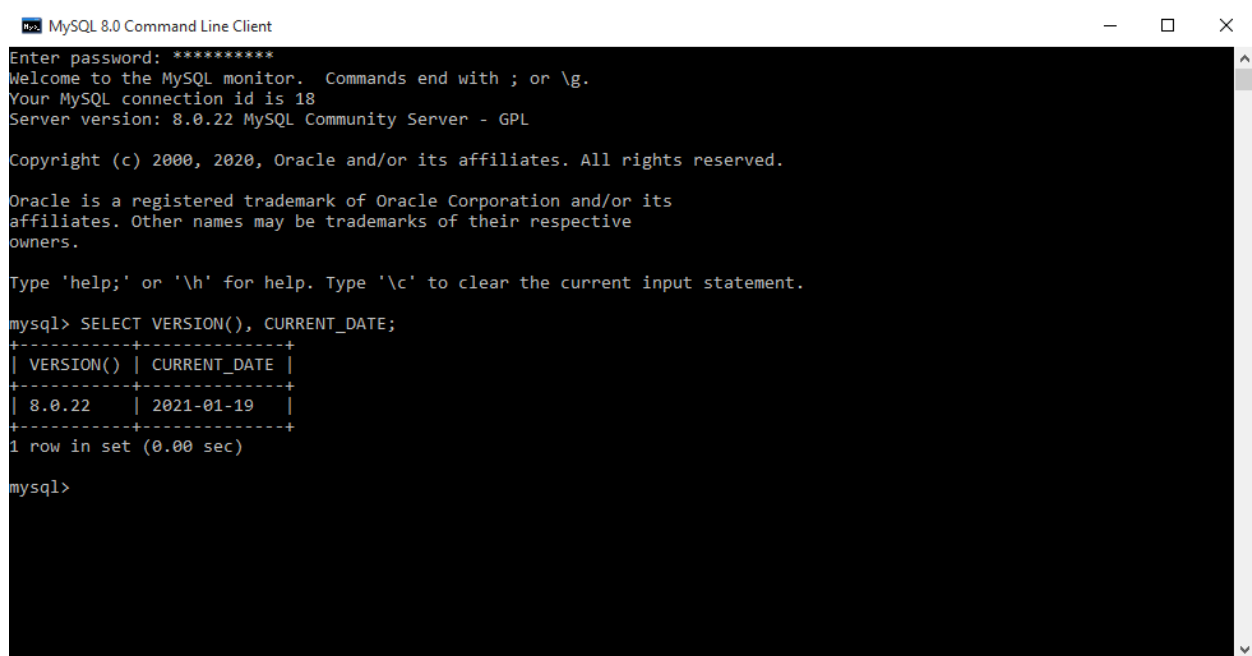
Lab No # 1

1. Introduction

A database is a separate application that stores a collection of data. Each database has one or more specific APIs for creating, accessing, managing, searching, and replicating the data it holds. Other kinds of data stores can also be used, such as files on the file system or large hash tables in memory, but data fetching and writing would not be so fast and easy with those type of systems. MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses.

1.1 Entering Queries

You can execute queries either in MySQL Workbench or in MySQL Command Line Client. Here is a simple query that asks the server to tell you its version number and the current date.

A screenshot of the MySQL 8.0 Command Line Client window. The window title is "MySQL 8.0 Command Line Client". The prompt "Enter password: *****" is shown. Below it, the text reads: "Welcome to the MySQL monitor. Commands end with ; or \g.", "Your MySQL connection id is 18", and "Server version: 8.0.22 MySQL Community Server - GPL". Further down, it says "Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved." and "Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners." The prompt "Type 'help;' or '\h' for help. Type '\c' to clear the current input statement." is shown. The user has entered the query "mysql> SELECT VERSION(), CURRENT_DATE;". The output is displayed in a table format: a header row with "VERSION()" and "CURRENT_DATE", and a data row with "8.0.22" and "2021-01-19". Below the table, it says "1 row in set (0.00 sec)". The prompt "mysql>" is shown again at the bottom.

```
MySQL 8.0 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SELECT VERSION(), CURRENT_DATE;
+-----+-----+
| VERSION() | CURRENT_DATE |
+-----+-----+
| 8.0.22    | 2021-01-19   |
+-----+-----+
1 row in set (0.00 sec)

mysql>
```

Figure: 1 MySQL Command Line Client

Keywords may be entered in any letter-case. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

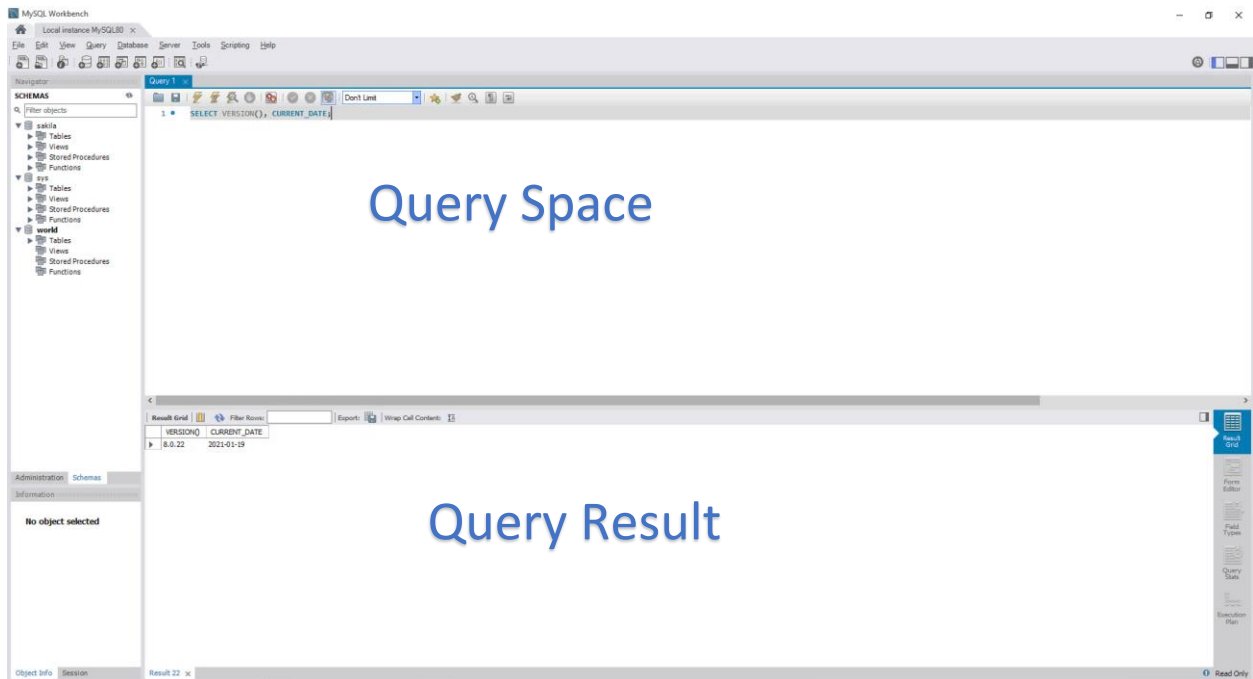


Figure: 2 MySQL Workbench

Here is another query. It demonstrates that you can use MySQL as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-----+-----+
| SIN(PI()/4) | (4+1)*5 |
+-----+-----+
| 0.70710678118655 | 25 |
+-----+-----+
1 row in set (0.02 sec)
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
+-----+
| VERSION() |
+-----+
| 8.0.13 |
+-----+
1 row in set (0.00 sec)

+-----+
| NOW() |
+-----+
| 2018-08-24 00:56:40 |
+-----+
1 row in set (0.00 sec)
```

A query need not be given all on a single line, so lengthy queries that require several lines are not a problem. MySQL determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. In other words, MySQL accepts free-format input: it collects input lines but does not execute them until it sees the semicolon. Here is a simple multiple-line statement:

```
mysql> SELECT
-> USER()
-> ,
-> CURRENT_DATE;
+-----+-----+
| USER() | CURRENT_DATE |
+-----+-----+
| jon@localhost | 2018-08-24 |
+-----+-----+
```

If you decide you do not want to execute a query that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
-> USER()
-> \c
mysql>
```

1.2 Creating and Using a Database

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sakila |
| sys |
| world |
+-----+
6 rows in set (0.01 sec)
```

If the “sakila” database exists, try to access it:

```
mysql> use sakila
Database changed
```

1.3 Getting Information About Databases and Tables

You have previously seen `SHOW DATABASES`, which lists the databases managed by the server. To find out which database is currently selected, use the `DATABASE()` function:

```
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| sakila |
+-----+
1 row in set (0.00 sec)
```

To find out what tables the default database contains (for example, when you are not sure about the name of a table), use this statement:

```
mysql> SHOW TABLES;
+-----+
| Tables_in_sakila |
+-----+
| actor              |
| actor_info         |
| address            |
| category           |
| city               |
| country            |
| customer           |
| customer_list      |
| film               |
| film_actor         |
| film_category      |
| film_list          |
| film_text          |
| inventory          |
| language           |
| nicer_but_slower_film_list |
| payment            |
| rental             |
| sales_by_film_category |
| sales_by_store     |
| staff              |
| staff_list         |
| store              |
+-----+
23 rows in set (0.01 sec)
```

If you want to find out about the structure of a table, the DESCRIBE statement is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE city;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default          | Extra          |
+-----+-----+-----+-----+-----+-----+
| city_id | smallint unsigned | NO   | PRI | NULL             | auto_increment |
| city    | varchar(50)      | NO   |     | NULL             |                |
| country_id | smallint unsigned | NO   | MUL | NULL             |                |
| last_update | timestamp        | NO   |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURRENT_TIMESTAMP |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Field indicates the column name, Type is the data type for the column, NULL indicates whether the column can contain NULL values, Key indicates whether the column is indexed, and Default specifies the column's default value. Extra displays special information about columns: If a column was created with the AUTO_INCREMENT option, the value is auto_increment rather than empty. DESC is a short form of DESCRIBE.

1.4 Basic Data Types in MySQL

Although there are lots of data types available but for our purpose, we shall be covering only the following:

Data Type	Format	Explanation
Number	INT	A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
	FLOAT(M,D)	A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.
	DECIMAL(M,D)	An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length

		(M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.
Character	CHAR(M)	A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
	VARCHAR(M)	A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.
Date and Time	DATE	A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.
	DATETIME	A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.
	TIMESTAMP	A timestamp between midnight, January 1st, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).

2. Creating Tables

CREATE TABLE creates a table with the given name. You must have the CREATE privilege for the table. By default, tables are created in the default database, using the InnoDB storage engine. An error occurs if the table exists, if there is no default database, or if the database does not exist. MySQL has no limit on the number of tables. The underlying file system may have a limit on the number of files that represent tables. Individual storage engines may impose engine-specific constraints. InnoDB permits up to 4 billion tables.

There are several aspects to the CREATE TABLE statement, described as following:

1. Table Name
 - tbl_name - The table name can be specified as db_name.tbl_name to create the table in a specific database. This works regardless of whether there is a default database, assuming that the database exists. If you use quoted identifiers, quote the database and table names separately. For example, write `mydb`.`mytbl`, not `mydb.mytbl`.
 - IF NOT EXISTS - Prevents an error from occurring if the table exists. However, there is no verification that the existing table has a structure identical to that indicated by the CREATE TABLE statement.
2. Temporary Tables - You can use the TEMPORARY keyword when creating a table. A TEMPORARY table is visible only within the current session and is dropped automatically when the session is closed.
3. Table Cloning and Copying
 - LIKE - Use CREATE TABLE ... LIKE to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

- [AS] query_expression - To create one table from another, add a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl AS SELECT * FROM orig_tbl;
```

- IGNORE | REPLACE - The IGNORE and REPLACE options indicate how to handle rows that duplicate unique key values when copying a table using a SELECT statement.

4. Column Data Types and Attributes

- data_type represents the data type in a column definition. For a full description of the syntax available for specifying column data types, as well as information about the properties of each type.
- CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
- If neither NULL nor NOT NULL is specified, the column is treated as though NULL had been specified.
- Specifies a default value for a column. For more information about default value handling, including the case that a column definition includes no explicit DEFAULT value.
- VISIBLE, INVISIBLE - Specify column visibility. The default is VISIBLE if neither keyword is present. A table must have at least one visible column. Attempting to make all columns invisible produces an error.
- AUTO_INCREMENT - An integer or floating-point column can have the additional attribute AUTO_INCREMENT. When you insert a value of NULL (recommended) or 0 into an indexed AUTO_INCREMENT column, the column is set to the next sequence value. Typically this is value+1, where value is the largest value for the column currently in the table. AUTO_INCREMENT sequences begin with 1.
- COMMENT - A comment for a column can be specified with the COMMENT option, up to 1024 characters long. The comment is displayed by the SHOW CREATE TABLE and SHOW FULL COLUMNS statements.
- GENERATED ALWAYS - Used to specify a generated column expression.

5. Indexes, Foreign Keys, and CHECK Constraints – Described in Section 5.

6. Table Options - Table options are used to optimize the behavior of the table. In most cases, you do not have to specify any of them.

7. Table Partitioning - partition_options can be used to control partitioning of the table created with CREATE TABLE.

Create Table syntax:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name  
  (create_definition,...)  
  [table_options]  
  [partition_options]
```

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name  
  [(create_definition,...)]  
  [table_options]  
  [partition_options]  
  [IGNORE | REPLACE]  
  [AS] query_expression
```

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name  
  { LIKE old_tbl_name | (LIKE old_tbl_name) }
```

Create Table Example:

```
mysql> CREATE TABLE IF NOT EXISTS products (
->     productID INT UNSIGNED NOT NULL AUTO_INCREMENT,
->     productCode CHAR(3) NOT NULL DEFAULT '',
->     name VARCHAR(30) NOT NULL DEFAULT '',
->     quantity INT UNSIGNED NOT NULL DEFAULT 0,
->     price DECIMAL(7,2) NOT NULL DEFAULT 99999.99,
->     PRIMARY KEY (productID)
-> );
Query OK, 0 rows affected (0.06 sec)

mysql> DESCRIBE products;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| productID  | int unsigned  | NO   | PRI | NULL    | auto_increment |
| productCode | char(3)       | NO   |     |          |                 |
| name       | varchar(30)   | NO   |     |          |                 |
| quantity   | int unsigned  | NO   |     | 0        |                 |
| price      | decimal(7,2)  | NO   |     | 99999.99 |                 |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.01 sec)
```

Show the complete CREATE TABLE statement used by MySQL to create this table use following statement:

```
mysql> SHOW CREATE TABLE products \G
***** 1. row *****
      Table: products
Create Table: CREATE TABLE `products` (
  `productID` int unsigned NOT NULL AUTO_INCREMENT,
  `productCode` char(3) NOT NULL DEFAULT '',
  `name` varchar(30) NOT NULL DEFAULT '',
  `quantity` int unsigned NOT NULL DEFAULT '0',
  `price` decimal(7,2) NOT NULL DEFAULT '99999.99',
  PRIMARY KEY (`productID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.01 sec)
```

3. Examples Common Queries

Here are examples of how to solve some common problems with MySQL. To create and populate the example table, use these statements:

```
CREATE TABLE shop (
  article INT UNSIGNED DEFAULT '0000' NOT NULL,
  dealer CHAR(20) DEFAULT '' NOT NULL,
  price DECIMAL(16,2) DEFAULT '0.00' NOT NULL,
  PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
  (1, 'A', 3.45), (1, 'B', 3.99), (2, 'A', 10.99), (3, 'B', 1.45),
  (3, 'C', 1.69), (3, 'D', 1.25), (4, 'D', 19.95);

mysql> CREATE TABLE shop (
->     article INT UNSIGNED DEFAULT '0000' NOT NULL,
->     dealer CHAR(20) DEFAULT '' NOT NULL,
->     price DECIMAL(16,2) DEFAULT '0.00' NOT NULL,
->     PRIMARY KEY(article, dealer));
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO shop VALUES
->     (1, 'A', 3.45), (1, 'B', 3.99), (2, 'A', 10.99), (3, 'B', 1.45),
->     (3, 'C', 1.69), (3, 'D', 1.25), (4, 'D', 19.95);
Query OK, 7 rows affected (0.01 sec)
Records: 7 Duplicates: 0 Warnings: 0
```

After issuing the statements, the table should have the following contents:

```
SELECT * FROM shop ORDER BY article;
```

article	dealer	price
1	A	3.45
1	B	3.99
2	A	10.99
3	B	1.45
3	C	1.69
3	D	1.25
4	D	19.95

3.1 The Maximum Value for a Column

“What is the highest item number?”

```
SELECT MAX(article) AS article FROM shop;
```

article
4

3.2 The Row Holding the Maximum of a Certain Column

Task: Find the number, dealer, and price of the most expensive article.

```
SELECT article, dealer, price
FROM shop
WHERE price=(SELECT MAX(price) FROM shop);
```

article	dealer	price
0004	D	19.95

3.3 Maximum of Column per Group

Task: Find the highest price per article.

```
SELECT article, MAX(price) AS price
FROM shop
GROUP BY article
ORDER BY article;
```

article	price
0001	3.99
0002	10.99
0003	1.69
0004	19.95

3.4 The Rows Holding the Group-wise Maximum of a Certain Column

Task: For each article, find the dealer or dealers with the most expensive price.

This problem can be solved with a subquery like this one:


```

SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM   shop s2
              WHERE  s1.article = s2.article)
ORDER BY article;

```

article	dealer	price
0001	B	3.99
0002	A	10.99
0003	C	1.69
0004	D	19.95

4. Inserting records in the table

In the previously created table, we define primary key on combination of column “article, dealer”. So if we try to give duplicate values to the combination of both columns. It gives error.

```

mysql> INSERT INTO shop VALUES
-> (3,'A',4.45),(2,'B',4.99),(3,'A',11.99),(4,'B',2.45),
-> (4,'C',2.69),(1,'D',2.25),(2,'D',21.95);
ERROR 1062 (23000): Duplicate entry '3-A' for key 'shop.PRIMARY'
|

```

After removing these duplicate values.

```

mysql> INSERT INTO shop VALUES
-> (4,'A',4.45),(2,'B',4.99),(3,'A',11.99),(4,'B',2.45),
-> (4,'C',2.69),(1,'D',2.25),(2,'D',21.95);
Query OK, 7 rows affected (0.00 sec)
Records: 7 Duplicates: 0 Warnings: 0

```

5. Constraints

- ✓ Your DBMS can do much more than just store and access data. It can also enforce rules (called constraints) on what data are allowed in the database. Such constraints are important because they help maintain data integrity. For example, you may want to ensure that each cgpa is not less than 2.0. When you specify the data type of a column, you constrain the possible values that column may hold. This is called a domain constraint. For example, a column of type INTEGER may only hold whole numbers within a certain range. Any attempt to insert an invalid value will be rejected by SQL. SQL allows the specification of many more constraint types. SQL enforces constraints by prohibiting any data in the database that violate any constraint. Any insert, update, or delete that would result in a constraint violation is rejected without changing the database.
- ✓ There are two forms of constraint specification:
 - o Column level Constraints:- Apply to individual columns only (are specified along with the column definition only)
 - o Table Level constraints:- Apply to one or more columns (are specified at the end)
- ✓ Constraints can be added to the table at the time of creation or after the creation of the table using ‘alter table’ command.

5.1 NOT NULL

- ✓ By default, most DBMSs allow NULL as a value for any column of any data type. You may not be so keen on allowing NULL values for some columns. You can require the database to prohibit NULL

values for particular columns by using the NOT NULL column constraint. Many DBMSs also include a NULL column constraint, which specifies that NULL values are allowed; however, because this is the default behavior, this constraint usually is unnecessary. Note that the NULL column constraint is not part of the SQL specification.

```
CREATE TABLE STAFF1 (
  SID NUMERIC (10) NOT
  NULL, NAME VARCHAR (20), DEPT
  VARCHAR (15)
);
```

- ✓ Now if you try inserting,

```
INSERT INTO STAFF1 (NAME, DEPT)
VALUES ('KRISHNA', 'PSD');
```

You will get error “Cannot insert the value NULL into column ...”

```
mysql> CREATE TABLE STAFF1 (
-> SID NUMERIC (10) NOT
-> NULL, NAME VARCHAR (20), DEPT
-> VARCHAR (15)
-> );
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> desc STAFF1;
```

Field	Type	Null	Key	Default	Extra
SID	decimal(10,0)	NO		NULL	
NAME	varchar(20)	YES		NULL	
DEPT	varchar(15)	YES		NULL	

```
3 rows in set (0.01 sec)
```

```
mysql> INSERT INTO STAFF1 (NAME, DEPT)
-> VALUES ('KRISHNA', 'PSD');
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL
server version for the right syntax to use near '?KRISHNA?, ?PSD?')' at line 2
mysql>
```

5.2 UNIQUE

- ✓ The UNIQUE constraint forces distinct column values. Suppose you want to avoid duplicate course numbers. Just specify the UNIQUE column constraint on the courseno column as follows:

```
CREATE TABLE COURSE (
  COMPCODE NUMERIC (4) UNIQUE,
  COURSENO VARCHAR (9) NOT NULL UNIQUE,
  COURSE_NAME VARCHAR (20),
  UNITS NUMERIC (2) NOT NULL
);
```

Note that UNIQUE only applies to non-NULL values. A UNIQUE column may have many rows containing a NULL value. Of course, we can exclude all NULL values for the column using the NOT NULL constraint with the UNIQUE constraint.

- ✓ UNIQUE also has a table constraint form that applies to the entire table instead of just a single column. Table constraints are specified as another item in the comma-delimited list of table elements.

Such table constraints apply to groups of one or more columns. Consider the following CREATE TABLE statement:

```
DROP TABLE COURSE; --TO DELETE THE TABLE

CREATE TABLE COURSE (
  COMPCODE NUMERIC (4),
  COURSENO VARCHAR (9) NOT NULL,
  COURSE_NAME VARCHAR (20),
  UNITS NUMERIC (2) NOT NULL,
  UNIQUE (COMPCODE, COURSENO)    -- TABLE LEVEL CONSTRAINT
);
```

- ✓ The combination of compcode, courseno is always unique. Note that table level unique constraint can also be for single column. Unique is nothing but the candidate key constraint but a unique column can take null values.

5.3 PRIMARY KEY

- ✓ It is similar to unique but with implicit NOT NULL constraint. The primary key of a table is a column or set of columns that uniquely identifies a row in the table. For example, idno is the primary key from the students table. We can declare a primary key using the PRIMARY KEY constraint. Here we show PRIMARY KEY used as a column constraint.

```
CREATE TABLE EMPLOYEE (
  EMPID NUMERIC (4) PRIMARY KEY,
  NAME VARCHAR (30) NOT NULL
);
```

- ✓ Creating primary key as a table constraint is shown below.

```
CREATE TABLE REGISTERED (
  COURSENO VARCHAR (9),
  IDNO CHAR (11),
  GRADE VARCHAR (10),
  PRIMARY KEY (COURSENO, IDNO)
);
```

- ✓ You can name your constraints by using an optional prefix.

```
CONSTRAINT <NAME> <CONSTRAINT
CREATE TABLE REGISTERED (
  COURSENO VARCHAR(9),
  IDNO CHAR(11),
  GRADE VARCHAR(10),
  CONSTRAINT PK_REGISTERED PRIMARY KEY(COURSENO, IDNO)    );
```

- ✓ This naming can be applied to unique constraints also. Naming constraint helps in altering or dropping that constraint at a later time.

5.4 FOREIGN KEY

- ✓ A foreign key restricts the values of a column (or a set of columns) to the values appearing in another column (or set of columns) or to NULL. In table registered (child table), courseno is a foreign key that refers to courseno in table course (parent table). We want all of the values of

courseno in the registered table either to reference a courseno from course or to be NULL. Any other courseno in the registered table would create problems because you couldn't look up information about the courseno which is not offered.

- ✓ In SQL, we specify a foreign key with the REFERENCES column constraint.

```
REFERENCES <REFERENCED TABLE> [ (<REFERENCED COLUMN> ) ]
```

- ✓ A column with a REFERENCES constraint may only have a value of either NULL or a value found in column <referenced column> of table <referenced table>. If the <referenced column> is omitted, the primary key of table <referenced table> is used.
- ✓ Before creating a foreign keys in registered table, let us re-create course, students tables with proper constraints.

```
DROP TABLE STUDENTS;  
CREATE TABLE STUDENTS (  
  IDNO CHAR(11),  
  NAME VARCHAR(30),  
  DOB DATE,  
  CGPA NUMERIC(4,2),  
  AGE NUMERIC(2),  
  CONSTRAINT PK_STUDENTS PRIMARY KEY(IDNO)  
);
```

```
DROP TABLE COURSE;  
CREATE TABLE COURSE (  
  COMPCODE NUMERIC(4),  
  COURSENO VARCHAR(9) NOT NULL,  
  COURSE_NAME VARCHAR(20),  
  UNITS NUMERIC(2) NOT NULL,  
  CONSTRAINT UN_COURSE UNIQUE (COMPCODE, COURSENO),    -- TABLE  
  LEVEL CONSTRAINT  
  CONSTRAINT PK_COURSE PRIMARY KEY (COURSENO)  
);
```

```
CREATE TABLE REGISTERED1 (  
  COURSENO VARCHAR (9) REFERENCES COURSE, --COLUMN LEVEL FOREIGN  
  KEY  
  IDNO CHAR(11) REFERENCES STUDENTS,  
  GRADE VARCHAR(10),  
  PRIMARY KEY(COURSENO, IDNO)  
);
```

- ✓ The same can be declared as table level constraint with proper naming.

```
CREATE TABLE REGISTERED2 (  
  COURSENO VARCHAR(9),  
  IDNO CHAR(11),  
  GRADE VARCHAR(10),  
  CONSTRAINT PK_REGISTERED2 PRIMARY KEY(COURSENO, IDNO),
```

```

CONSTRAINT FK_CNO FOREIGN KEY (COURSENO) REFERENCES COURSE,
CONSTRAINT FK_IDNO FOREIGN KEY (IDNO) REFERENCES STUDENTS
);

```

- ✓ To create a foreign key reference, SQL requires that the referenced table/column already exist.
- ✓ Maintaining foreign key constraints can be painful. To update or delete a referenced value in the parent table, we must make sure that we first handle all foreign keys referencing that value in the child table. For example, to update or delete 2007A7PS001 from the students table, we must first update or delete all registered. idno. SQL allows us to specify the default actions for maintaining foreign key constraints for UPDATE and DELETE on the parent table by adding a referential action clause to the end of a column or table foreign key constraint:

```
ON UPDATE <ACTION>
```

```
ON DELETE <ACTION>
```

- ✓ Any UPDATE or DELETE on the parent table triggers the specified <action> on the referencing rows in the child table.

Action	Definition
SET NULL	Sets any referencing foreign key values to NULL.
SET DEFAULT	Sets any referencing foreign key values to the default value (which may be NULL).
CASCADE	On delete, this deletes any rows with referencing foreign key values. On update, this updates any row with referencing foreign key values to the new value of the referenced column.
NO ACTION	Rejects any update or delete that violates the foreign key constraint. This is the default action.
RESTRICT	Same as NO ACTION with the additional restriction that the action cannot be deferred

- ✓ Try the following.

```

DROP TABLE REGISTERED2; CREATE
TABLE REGISTERED2( COURSENO VARCHAR(9) , IDNO CHAR(11),
GRADE VARCHAR(10) , CONSTRAINT
PK_REGISTERED2
PRIMARY KEY(COURSENO, IDNO),
CONSTRAINT FK_CNO FOREIGN KEY (COURSENO) REFERENCES COURSE ON
DELETE CASCADE,
CONSTRAINT FK_IDNO FOREIGN KEY (IDNO) REFERENCES STUDENTS ON
DELETE
CASCADE
);

```

- ✓ Modify the above query with other actions ON DELETE SET NULL, ON DELETE NO ACTION, ON UPDATE NO ACTION.

5.5 CHECK

- ✓ We can specify a much more general type of constraint using the CHECK constraint. A CHECK constraint specifies a boolean value expression to be evaluated for each row before allowing any data change. Any INSERT, UPDATE, or DELETE that would cause the condition for any row to evaluate to false is rejected by the DBMS.

```
CHECK (<condition>)
```

- ✓ A CHECK constraint may be specified as either a column or table constraint. In the following example, we specify CHECK constraints on the students table:

```
CREATE TABLE STUDENT1(  
  IDNO CHAR(11) PRIMARY KEY,  
  NAME VARCHAR(20) NOT NULL,  
  CGPA NUMERIC(4,2) CHECK(CGPA >= 2 AND CGPA <= 10), -- CGPA  
  CONSTRAINT  
  ROOMNO NUMERIC(3) CHECK(ROOMNO >99),  
  HOSTEL_CODE VARCHAR(2) CHECK(HOSTEL_CODE IN ("VK","RP","MB"))  
);
```

- ✓ Check constraints can also be named.
- ✓ Does a roomno with a NULL value violate the CHECK constraint? No. In this case, the CHECK condition evaluates to unknown. The CHECK constraint only rejects a change when the condition evaluates to false. In the SQL standard, a CHECK constraint condition may even include subqueries referencing other tables; however, many DBMSs do not implement this feature.

6. More on Creating, Deleting, and Altering Tables: (DDL)

In the previous section, creation tables with defaults values, NOT NULL constraint, UNIQUE constraint, Primary key, foreign key and check constraints have been discussed.

6.1 Creating a table from another table:

LIKE - Use CREATE TABLE ... LIKE to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
mysql> create table product2 like products;  
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> desc product2;
```

Field	Type	Null	Key	Default	Extra
productID	int unsigned	NO	PRI	NULL	auto_increment
productCode	char(3)	NO			
name	varchar(30)	NO			
quantity	int unsigned	NO		0	
price	decimal(7,2)	NO		99999.99	

```
5 rows in set (0.02 sec)
```

Now, can copy values from table "products" into "product2".

```
mysql> INSERT INTO product2
-> select * from products;
Query OK, 7 rows affected (0.01 sec)
Records: 7 Duplicates: 0 Warnings: 0
```

```
mysql> select * from product2;
```

productID	productCode	name	quantity	price
1001	PEN	Pen Red	5000	1.23
1002	PEN	Pen Blue	8000	1.25
1003	PEN	Pen Black	2000	1.25
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49
1006	PEC	Pencil 2B	10000	0.48
1007	PEC	Pencil 2H	8000	0.49

```
7 rows in set (0.00 sec)
```

[AS] query_expression - To create one table from another, add a SELECT statement at the end of the CREATE TABLE statement:

```
mysql> CREATE TABLE product3 AS SELECT * FROM products;
Query OK, 7 rows affected (0.12 sec)
Records: 7 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM product3;
```

productID	productCode	name	quantity	price
1001	PEN	Pen Red	5000	1.23
1002	PEN	Pen Blue	8000	1.25
1003	PEN	Pen Black	2000	1.25
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49
1006	PEC	Pencil 2B	10000	0.48
1007	PEC	Pencil 2H	8000	0.49

```
7 rows in set (0.00 sec)
```

6.2 Alter Table

Begin with a table t1 created as shown here:

```
CREATE TABLE t1 (a INTEGER, b CHAR(10));
```

To rename the table from t1 to t2:

```
ALTER TABLE t1 RENAME t2;
```

To change column a from INTEGER to TINYINT NOT NULL (leaving the name the same), and to change column b from CHAR(10) to CHAR(20) as well as renaming it from b to c:

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

6.3 ADD [COLUMN] <column definition>

To add a new TIMESTAMP column named d:

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

6.4 MODIFY <column name>

To add a new AUTO_INCREMENT integer column named c:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
ADD PRIMARY KEY (c);
```

6.5 DROP <column name>

To remove column c:

```
ALTER TABLE t2 DROP COLUMN c;
```

6.6 ADD <table constraint>

To add an index on column d and a UNIQUE index on column a:

```
ALTER TABLE t2 ADD INDEX (d), ADD UNIQUE (a);
```

7. Dropping a Table

DROP TABLE removes one or more tables. You must have the DROP privilege for each table. Be careful with this statement! For each table, it removes the table definition and all table data. If the table is partitioned, the statement removes the table definition, all its partitions, all data stored in those partitions, and all partition definitions associated with the dropped table.

Dropping a table also drops any triggers for the table. DROP TABLE causes an implicit commit, except when used with the TEMPORARY keyword.

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

The RESTRICT and CASCADE keywords do nothing. They are permitted to make porting easier from other database systems.

```
mysql> SELECT * FROM product3;
```

productID	productCode	name	quantity	price
1001	PEN	Pen Red	5000	1.23
1002	PEN	Pen Blue	8000	1.25
1003	PEN	Pen Black	2000	1.25
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49
1006	PEC	Pencil 2B	10000	0.48
1007	PEC	Pencil 2H	8000	0.49

```
7 rows in set (0.00 sec)
```



```
mysql> DROP TABLE product3;
```

Query OK, 0 rows affected (0.05 sec)


```
mysql> SELECT * FROM product3;
```

ERROR 1146 (42S02): Table 'sakila.product3' doesn't exist

7 Exercise

- Write the following queries in SQL, using the university schema.
 - Find the titles of courses in the Comp. Sci. department that have 3 credits.
 - Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.
 - Find the highest salary of any instructor.
 - Find all instructors earning the highest salary (there may be more than one with the same salary).
 - Find the enrollment of each section that was offered in Fall 2017.
 - Find the maximum enrollment, across all sections, in Fall 2017.

- g. Find the sections that had the maximum enrollment in Fall 2017.
- 2. Write the following inserts, deletes, or updates in SQL, using the university schema.
 - a. Increase the salary of each instructor in the Comp. Sci. department by 10%.
 - b. Delete all courses that have never been offered (i.e., do not occur in the *section* relation).
 - c. Insert every student whose *tot cred* attribute is greater than 100 as an instructor in the same department, with a salary of \$10,000.

*****END*****