Northeastern University
Khoury College of
Computer Sciences

NEU Database Exercise Lab Book

For CS 5200 Database Management Systems

Last modified: July 14, 2022

By

Professor Jeongkyu Lee, PhD

Northeastern University
Silicon Valley

Lab 1: Install MySQL on Local Machine

A. MySQL?

- In this class, you are going to learn and practice the database using MySQL database management system (DBMS, but called "db" in short)
- MySQL is an open-source relational database management system (RDBMS)
- MySQL is free and open-source software under the terms of the GNU General Public License. MySQL was owned and sponsored by the Swedish company MySQL AB, which was bought by Sun Microsystems (now Oracle Corporation).
- Website: https://www.mysql.com/
- Repository: https://github.com/mysql/mysql-server

B. Use of MySQL

1. MySQL on my computer → Today
2. MySQL on Cloud Platform (GCP or AWS) → Exercise Lab3
3. MySQL on Cloud Instance (db4free.net)
4. Alternative to MySQL (MariaDB)
5. Not using MySQL. Instead, SQLite, Oracle, Postgres and many others

In this class and labs, you are going to use Method 1 and 2. MySQL on GCP will be covered in Exercise Lab3.

C. Install MySQL on my computer

Use the instructions on the Oracle MySQL Website
https://dev.mysql.com/doc/refman/8.0/en/installing.html

Version? Current version is 8.0, but you can still use 5.7 (or 5.6) in this class

C1. For Windows

- MySQL provides the MySQL installer for all applications and drivers associated with MySQL. Once installed, the MySQL installer can help you keep your software up to date as well as identify missing software components. We recommend using it. You can download the installer from the web page: Or https://dev.mysql.com/downloads/installer/
- You can find more info at:
https://drive.google.com/file/d/1Imm8_FBOlPUe_dPMDEUUp9_xAqPAMfyS

C2. For MacOS

- If you are using homebrew

```
$ brew install mysql
$ brew services start mysql
$ mysql.server start
```

- If you do not use homebrew, follow the instruction at https://drive.google.com/file/d/1OHAOMScaiV9Jj9IBXDMPyBxtT9NI5cz7/

D. Install MySQL Workbench (Client)

- You can install MySQL Workbench to help administer any local or remote database https://www.mysql.com/products/workbench/
- MySQL Workbench is a unified visual tool for database architects, developers, and DBAs.
  - Design: you need this in Module 2
  - Develop
  - Admin
  - Migration
- For more info, https://youtu.be/X_umYKqKaF0

E. Command Line MySQL client

CS 5200 Exercise Lab will use command line (5.x) or "mysqlsh" (8.0) tool

- For Windows,
  - Start cmd.exe OR Power shell
- For MacOS,
  - Start Terminal
- Connect to local mysql

```
$ mysql -uroot -p
```

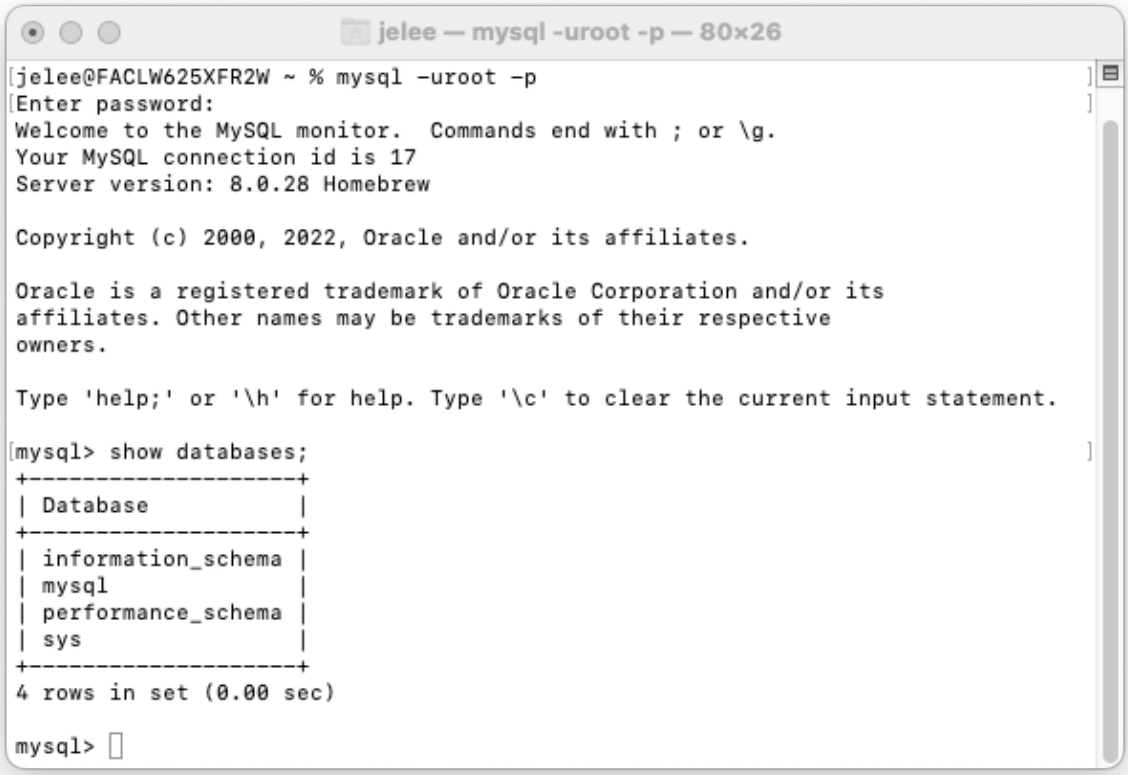
OR if you are using MySQL 8.0 or above

```
$ mysqlsh -uroot -p
MySQL localhost:33060+ ssl JS> \sql
MySQL localhost:33060+ ssl SQL>
```

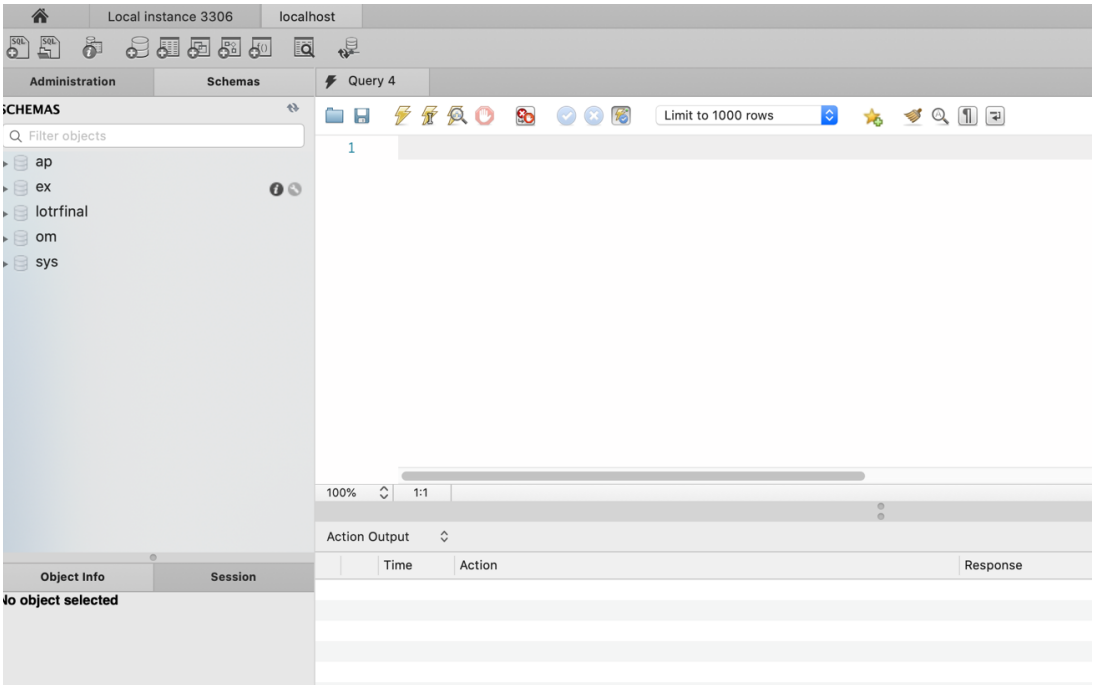
- Then, type the following command

```
mysql> show databases;
```

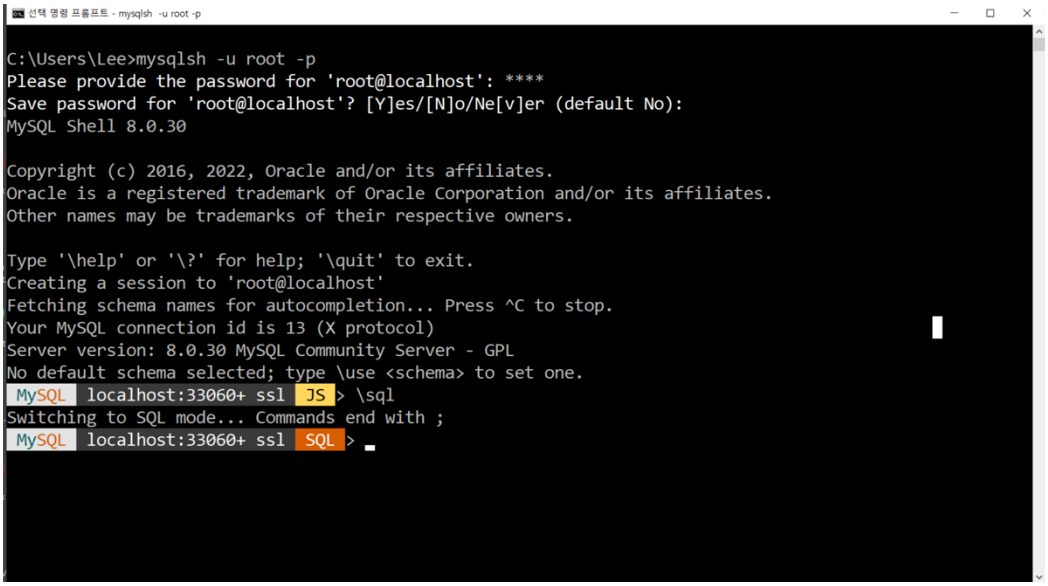
- mysql client (5.x)



- MySQL Workbench



- MySQL Shell client (8.0): mysqlsh



Exercise

Submit a couple of screenshots that show your connection

- One is from command line “mysql” or “mysqlsh”
- Another is from MySQL Workbench

Lab 2: Conceptual Modeling using erwin Data Modeler

A. ER Diagrams using erwin

erwin is one of the most popular software applications for data modeling worldwide. erwin, manufactured by Quest Software, supports many database related functions including database design, logical data modeling, physical data modeling, and reengineering for a variety of DBMS such as Oracle, MySQL, DB2, SQL Server and others. erwin is not free software, but you can also request academic license for students:

- http://go.erwin.com/erwin-academic-edition-free-trial

For Mac users, erwin data modeler is not available for MacOS, so you can use MySQL Workbench instead

- https://www.mysql.com/products/workbench/

First, look at the description of SALES database and try to understand it. This includes entities, their attributes and data types, key attribute (underlined), and some constraints. Look at the following description of SALES database. Notice that the key attributes CANNOT be null.

SALES database:

```
Customers
(
  cust_id       varchar (10)   NOT NULL ,
  cust_name     varchar (50)   NOT NULL ,
  cust_address  varchar (50),
  cust_city     varchar (50),
  cust_state    varchar (5) ,
  cust_zip      varchar (10),
  cust_country  varchar (50),
  cust_contact  varchar (50),
  cust_email    varchar (255)
)

Vendors
(
  vend_id       varchar (10) NOT NULL ,
  vend_name     varchar (50) NOT NULL ,
  vend_address  varchar (50),
  vend_city     varchar (50),
  vend_state    varchar (5) ,
  vend_zip      varchar (10),
  vend_country  varchar (50),
)

Products
(
  prod_id       varchar (10)       NOT NULL ,
  vend_id       varchar (10)       NOT NULL ,
  prod_name     varchar (255)      NOT NULL ,
  prod_price    number(8,2)        NOT NULL ,
  prod_desc     varchar (255)
)

Orders
(
  order_num     number         NOT NULL ,
  order_date    date           NOT NULL ,
  cust_id       varchar(10)    NOT NULL ,
)

OrderItems
(
  order_item    number         NOT NULL ,
  order_num     number         NOT NULL ,
  prod_id       varchar (10)   NOT NULL ,
  quantity      number         NOT NULL ,
  item_price    number(8,2)    NOT NULL
)
```

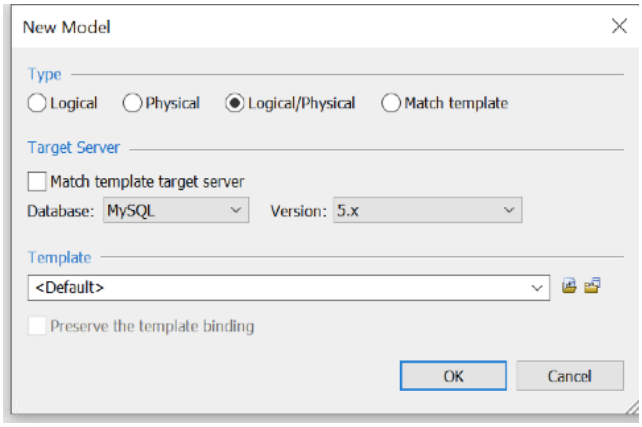
You are going to create ER diagram using erwin from the SALES database. **There is an example ER diagram at the end of this exercise, please only look at it if you are stuck! If you just copy it in you will learn nothing!**

To start erwin: search “erwin data modeler” at search menu

Start erwin Data Modeler.

A1. Setup design environment

1. Create a new model with “File → New”

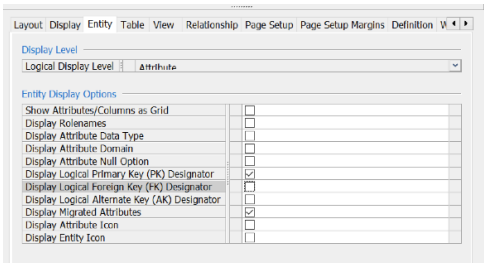


- Select “Logical/Physical” for New Model Type
- Select “MySQL” and “5.x” for Database

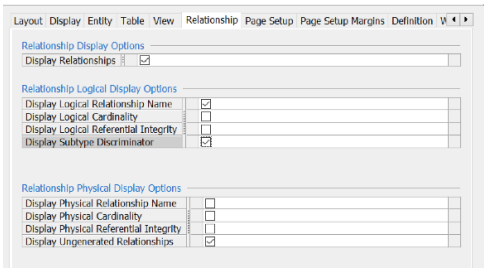
2. Select the level of detail in the display as follows:

Right-click on model template (desktop) to open diagram editor

- Choose Entity Display and uncheck “Foreign Key Designator (FK)”
- Choose Entity Display again and check “Primary Key Designator”
- Choose Entity Display again and check “Show Migrated Attributes”
- Choose Relationship Display and check “Verb Phrase”



Entity Tab

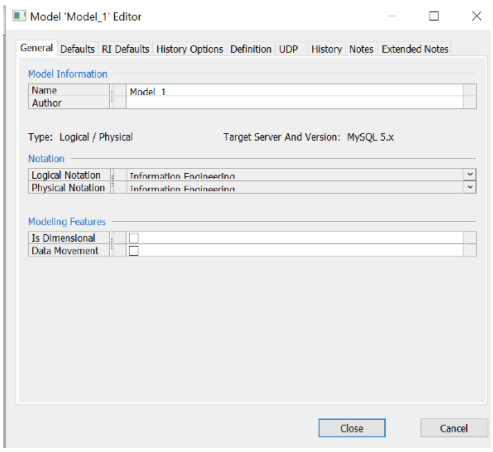


Relationship Tab

3. Choose the ER diagram notation

From the Model Explorer

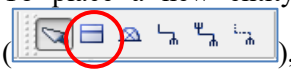
- Right-click and select “Properties”
- When the Model Properties window opens, click on the “General” tab
- At notation menu, Select “Information Engineering” under “Logical Notation” and “Physical Notation”

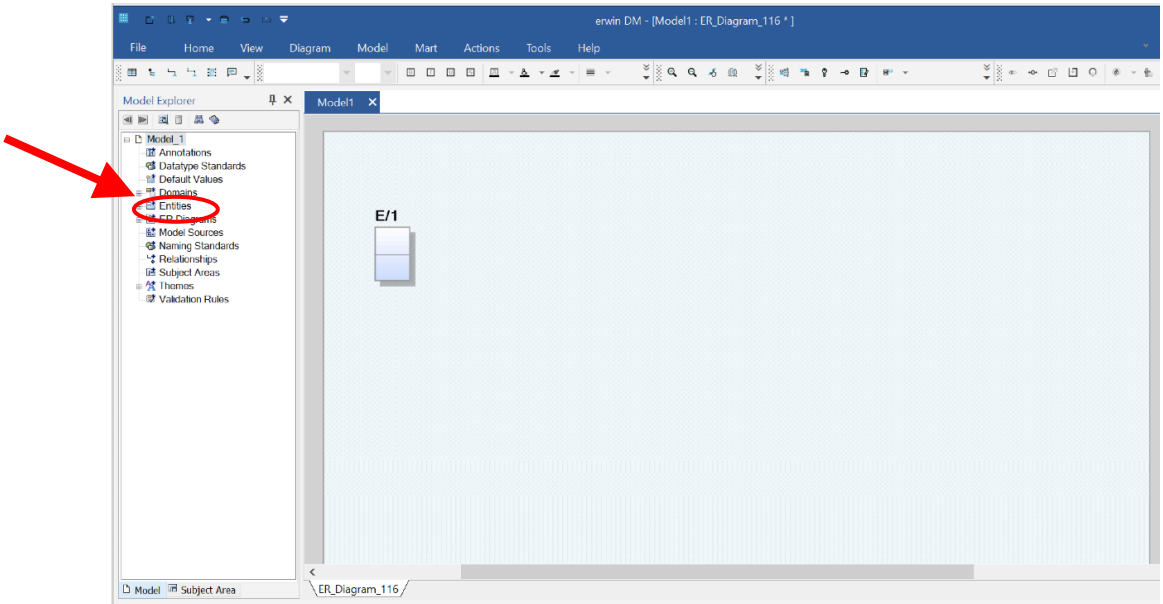


Model Properties window

A2. Create a new Entity

1. Place a new entity representing CUSTOMERS on the screen

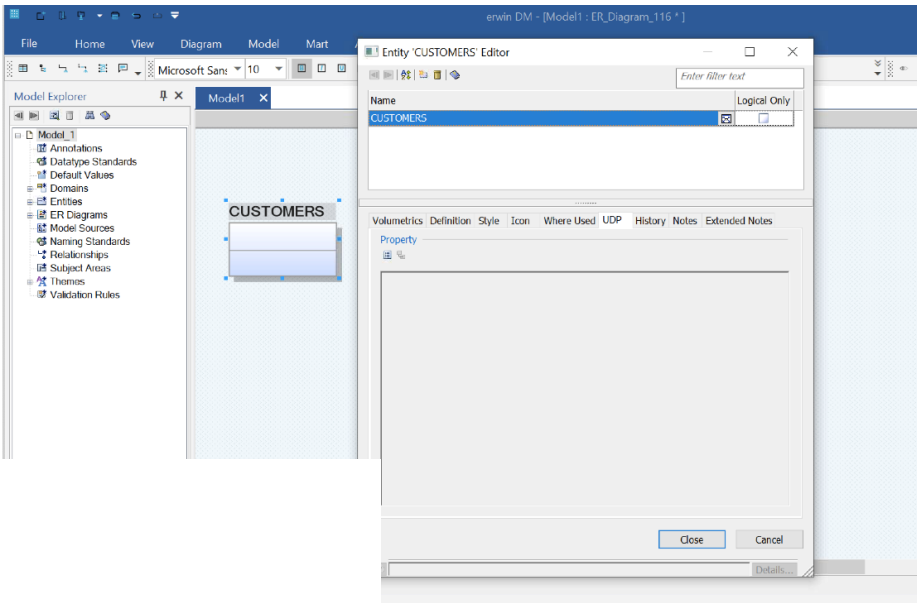
- To place a new entity onto the model, click the entity icon () on the tool bar (), and then click a position in the display window where you would like the entity to appear. NOTE: An alternate method is to right-click on the word "Entities" in the Model Navigator.



- The default name will be [E/X] where 'X' is sequence number, e.g. '1' in this example. The entity consists of three parts, "Entity Name", "Primary Key Attribute(s)" and "Non-Primary Key Attribute(s)". You can move to each part by pressing 'TAB' key.

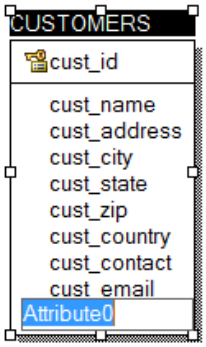
- Press tab key until the entity name is highlighted, and then enter the entity name, CUSTOMERS.

- To edit an existing entity, right-click on the entity and select 'Entity Properties....'. Then, double click on the entity. The Entity Editor will be shown.



2. Add Attributes to the CUSTOMERS entity

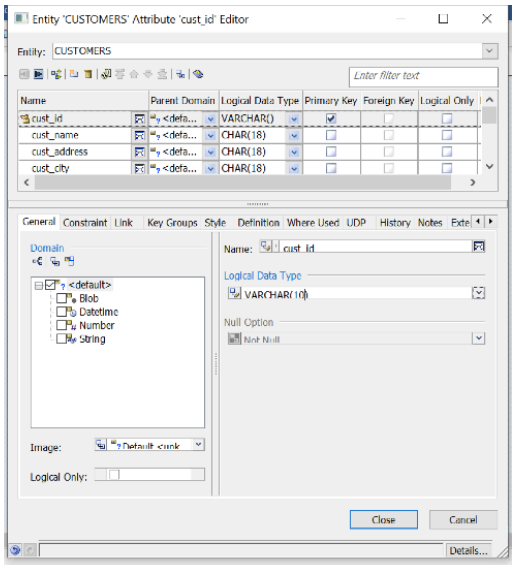
- Press the tab key to highlight the primary key box, enter the name of the primary key attribute (i.e., cust_id).
- To add non-primary keys, press the tab key to highlight attribute box, where you can add non-primary key attributes. Type the attribute name and enter, continuously, until all attributes for CUSTOMERS are entered (i.e., cust_name, cust_address, ... , cust_email). Then ESC to exit the entity.



Press ESC to quit adding attributes

3. Edit the Datatype for each attribute of the CUSTOMER entity

- Right-click on the entity and select "Attributes ..." .
- When the Attributes window opens, select the attribute to edit and choose the "Datatype" tab.



- Select the appropriate Datatype from the list. If the assigned datatype has parameters (e.g. VARCHAR(10)) enter the wanted value between ( ). Repeat this step for all attributes.
- Check 'Not Null' if the attribute should not be null.
- Once you have created the 'CUSTOMERS' entity, save the file in your home directory, and then define all the other entities. (You need to save your file frequently)

4. Repeat steps 1, 2, and 3 for all entities

A3. Make relationships

Once the entities have been designed, the relationships between them must be defined. We must link primary and foreign keys, and link the relations together using the correct cardinality. Here are the relationships (you can give them your own names):

Relationships for CUSTOMER database

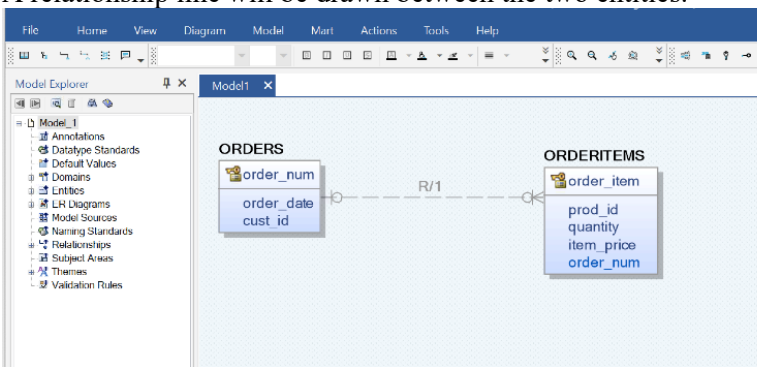
- ORDERS has a 1..N relationship with ORDERITEMS
  FK_OrderItems_Orders FOREIGN KEY (order_num) REFERENCES Orders (order_num)
- ORDERITEMS has a 1..N relationship with PRODUCTS
  FOREIGN KEY (OrderItems_order_item) in Products REFERENCES OrderItems (Order_order_item)
- CUSTOMERS has a 0..N relationship with ORDERS
  FOREIGN KEY (Customers_cust_id) in Orders REFERENCES Customers (cust_id)
- VENDORS has a 1..N relationship with PRODUCTS
  FK_Products_Vendors REFERENCES Vendors (vend_id)

1. Add a relationship between ORDERS and ORDERITEMS entities
   The Parent Entity is the entity you want to be on the "1" side of the relationship (i.e., parent), and the entity you want to be on the "N" side is the Child Entity.

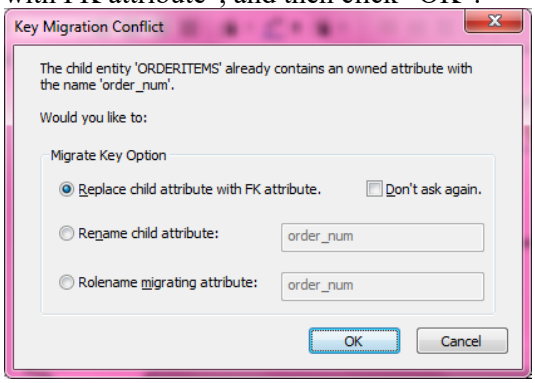
   - Click on the 1-to-N Non-identifying relationship icon () on the toolbar ().
   - Select the Parent Entity first (click on ORDERS)

- Then select the Child Entity (click on ORDERITEMS).

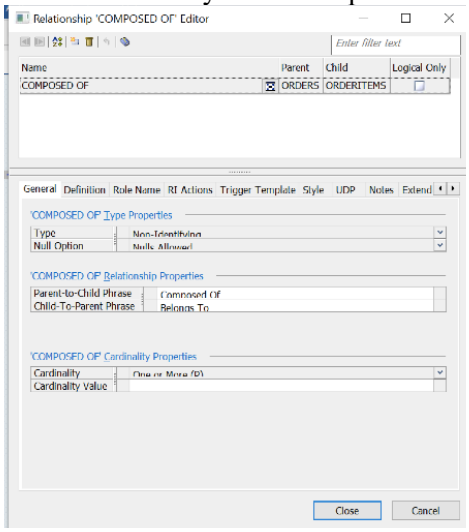
  A relationship line will be drawn between the two entities.



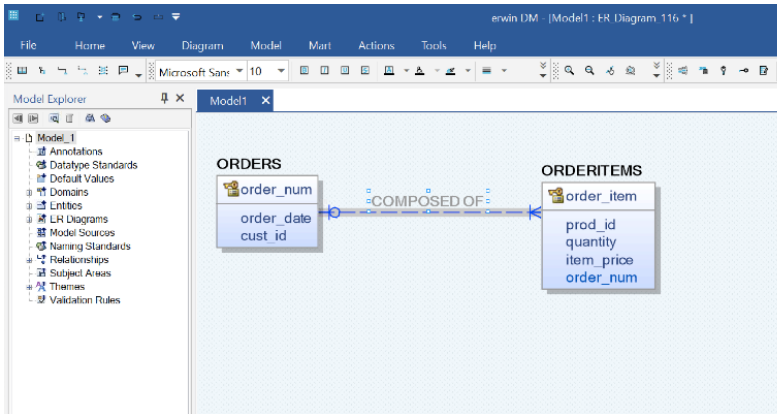
  NOTE: If the child entity has an attribute with the same name as an attribute of parent entity (i.e., order_num) you will see the following pop-up window. Select "Replace child attribute with FK attribute", and then click "OK".



2. Name the relationship and change the cardinality
   - Double-click the relationship. The "Relationship" window will open.
   - Enter a name in the "Name" textbox (i.e., Composed of).
   - Enter a Verb Phrase in the "Parent-to-Child" textbox (i.e., Composed of)
   - Enter a Verb Phrase in the "Child-to-Parent" textbox (i.e., Belongs to)
   - For the 1-to-many relationship select the "One or More (P)" option under "Cardinality".

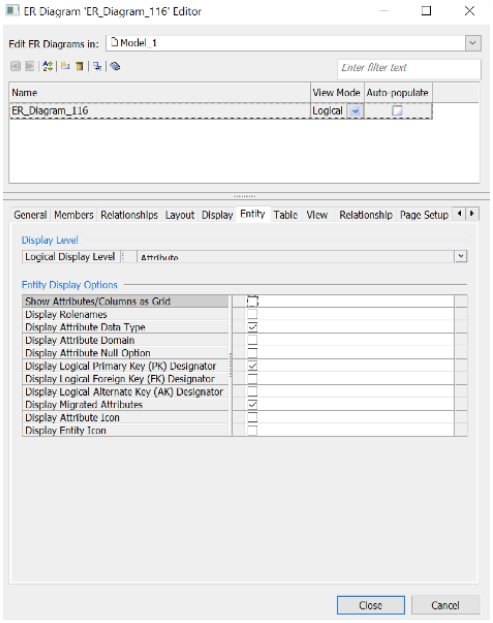


The following figure is for the 1-to-N relationship between ORDERS and ORDERITEMS.



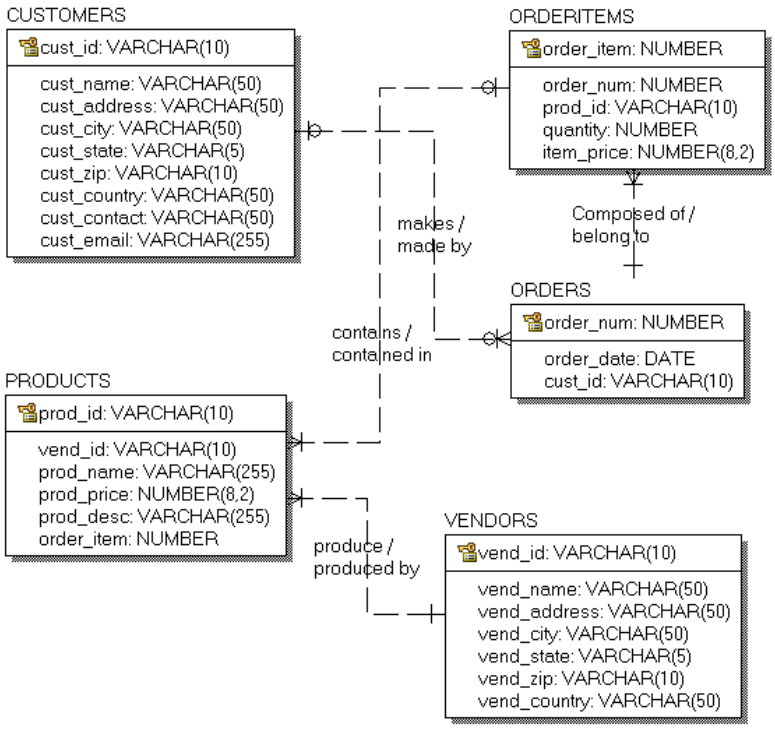
3. Repeat steps 1 and 2 for the remaining relationships

NOTE: To display Datatype for each attribute, select “Attribute Datatype” from the “ER Model Editor Properties → Entity Tab → Entity Display Option”.



A4. Save File

When have defined all the relationships save the file. The completed diagram should look something like this when you are finished:



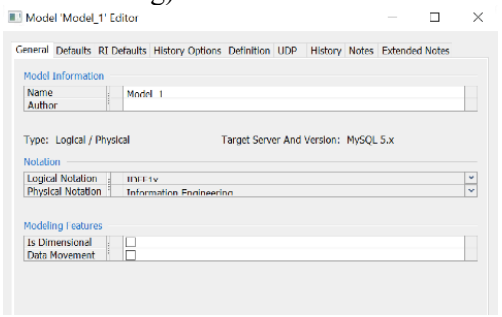
B. EER Diagrams using ERwin

The following steps will help you to complete the EER diagram for your homework.

B1. Setup design environment

1. **Create a new model with "File → New"**
   - Select "Logical/Physical" for New Model Type
   - Select "MySQL" and "5.x" for Database

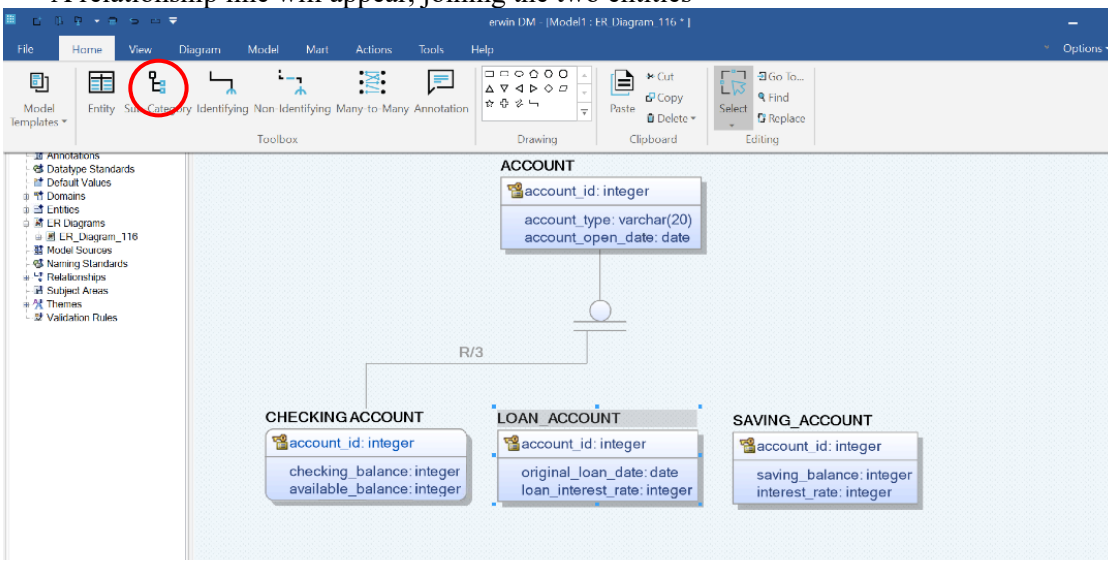
2. **Modify the Model Properties as follows:**
   From the Model Menu
   - Choose "Properties… "
   - When the Model Properties window opens click on the "Notation" tab
   - In the "Logical Notation" section select "IDEF1X (Integration DEFinition for Information Modeling)"



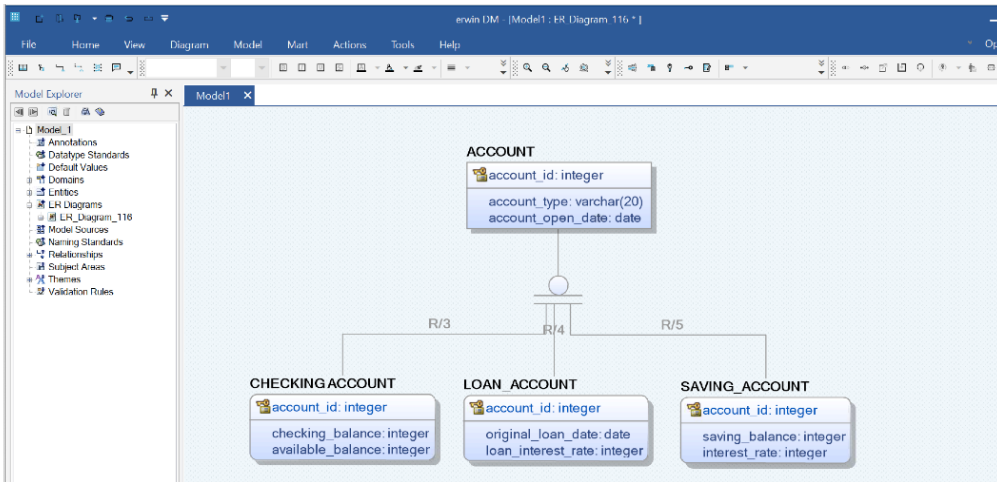
B2. Create entities (use the diagram below and follow procedure outlined in section A)

B3. Join the Super and Sub Classes

1. **Create the relationship between the Parent Entity and the first Child Entity**
   - Click on the "complete sub-category" icon on the toolbar. The mouse cursor will change to resemble the icon.
   - Select the Parent Entity (click on ACCOUNT)
   - Select one of the Child Entities (click on CHECKING_ACCOUNT)
     A relationship line will appear, joining the two entities

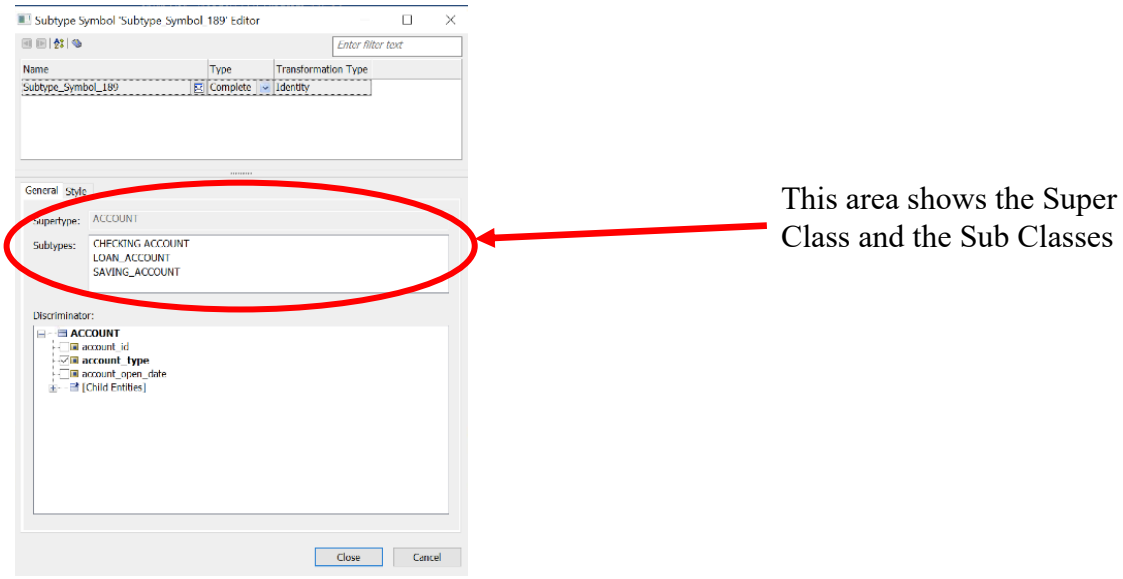


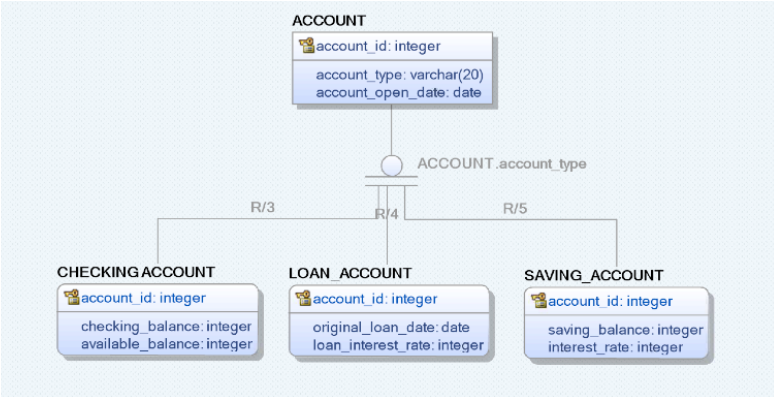
2. **Create the relationship between the Parent Entity and the remaining Child Entities**
   - Make sure the "complete sub-category" icon is selected on the toolbar.
   - Click on the Subtype symbol in the Display area.
   - Click on the Child Entity.
   - Repeat for all Child Entities.



3. Name the relationship

- Right-click on the Subtype symbol in the Display area.
- Select “Subtype properties...” from the menu.
- In the Subtype Properties window, select the properties on which the Specialization/Generalization has been done. In this case the Specialization/Generalization was done on Account_type, so put a check in the Account_type checkbox.





Exercise

Optional Exercise 1: ER Diagram for COMPANY database

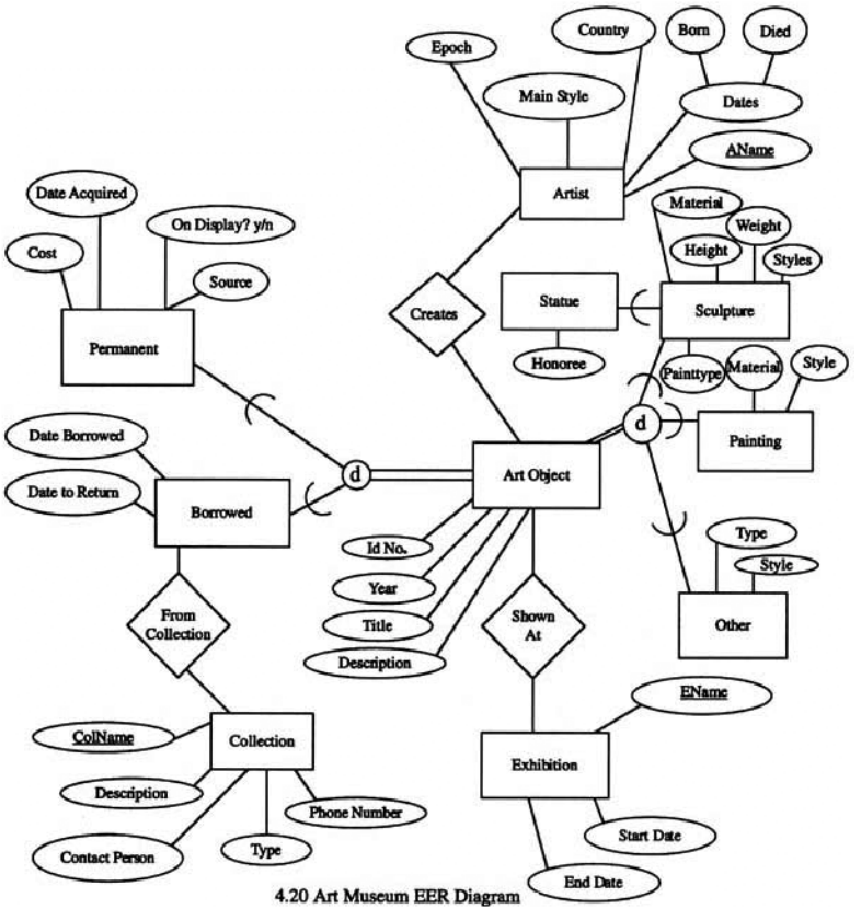
Use COMPANY database. (Figure 3.6, and Figure 3.7)
You have to create a Crow's Foot diagram using ERwin for the COMPANY database that you have used for lab exercises as follows:

- Create all entities.
- Change entity names.
- Create all attributes.
- Modify appropriate data types, primary key, foreign key, and not null constraints.
- Define the relationships in the COMPANY database.
- Create relations, and give the names.

Optional Exercise 2: EER Diagram for Art Museum database

This is the EER diagram for Art Museum database. From the diagram, create EER diagram using ERwin.

- You need to study by yourself how to generate EER diagram using ERwin.



4.20 Art Museum EER Diagram

[Lab Assignment] Exercise 3: ER modeling

Design an ER schema for keeping track of information about vote taken in the U.S. House of Representatives during the current two-year congressional session. The database needs to keep track of each U.S. **STATE**'s Name (e.g., 'Texas', 'New York', 'Connecticut') and include the **Region** of the state (whose domain is {'Northeast', 'Midwest', 'Southeast', 'Southwest', 'West'}). Each **CONGRESS_PERSON** in the House of Representatives is described by his or her **Name**, plus the **District** represented, the **Start_date** when the congressperson was first elected, and the political **Party** to which he or she belongs (whose domain is "'Republican', 'Democrat', 'Independent', 'Other')). The database keeps track of each **BILL** (i.e., proposed law), including the **Bill_name**, the **Date_of_vote** on the bill, whether the bill **Passesed_or_failed** (whose domain is {'Yes', 'No'}, and the **Sponsor** (the congressperson(s) who sponsored-that is, proposed-the bill). The database keeps track of how each congressperson voted on each bill (domain of Vote attribute is {'Yes', 'No', 'Abstain', 'Absent'}).

- Find Entities.
- Find Relationships.
- Draw an ER schema diagram using erwin with appropriate options.

State clearly any assumptions you make.

Lab 3: Setting Up MySQL Database on your GCP

There 3 different ways to install MySQL database on your GCP Compute Engine as below:

1. Google Cloud SQL
2. Google Cloud Launcher
3. MySQL on Compute Engine

In this exercise lab, we are going to use Option 1 to create MySQL instance. However, students can create MySQL instance using either Option 2 or 3.

Before you start Lab 3, you should download exercise lab files from Khoury Github:

```
$ git clone https://github.khoury.northeastern.edu/jelee0408/CS5200_Lab.git
```

Check if all lab files are downloaded correctly.

A. Creating MySQL instance

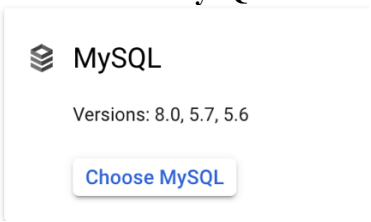
1. Go to the Cloud SQL Instances page in the Google Cloud Platform Console.

   ```
   GCP Console -> SQL
   ```

2. Then, **click** "**Create Instance**"

3. Select **MySQL** and click **Next**.

4. Click **Choose MySQL**.

5. Enter the following info:
   - Instance ID: **neu-test-db**
   - Root password:
   - Choose database version: MySQL 5.6 or **MySQL 5.7** (default)
   - Choose a configuration to start with: **Development**
   - Region: **us-west1 (Oregon)**
   - Zone availability: **Single zone**
     - Primary zone: Any

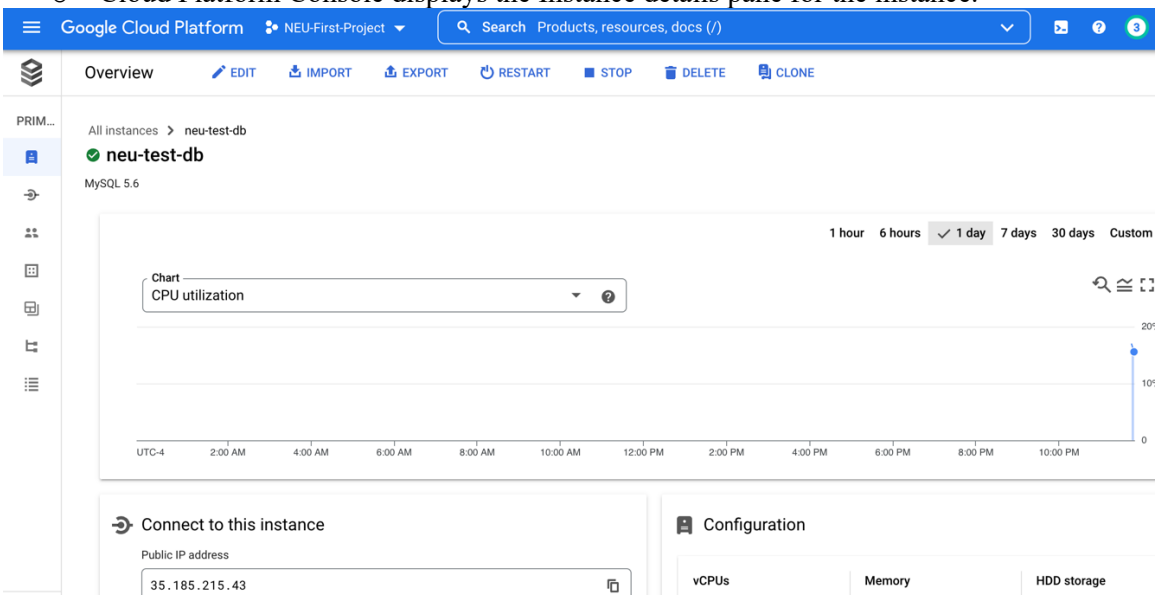
6. Show "Customize your instance" option to change any of the following options:
   - Machine Type: **Shared core (1 vCPU, 0.614 GB)**
   - Storage type: **HDD**
   - Storage capacity: **10 GB**
   - **Enable automatic storage increase**
   - Set connectivity: check **Public IP**

- New network:
    - Name: **All IP**
    - Network: **0.0.0.0/0**
    - Then, click “Done”
- Backups: **uncheck automatic backups** and **Instance deletion protection**

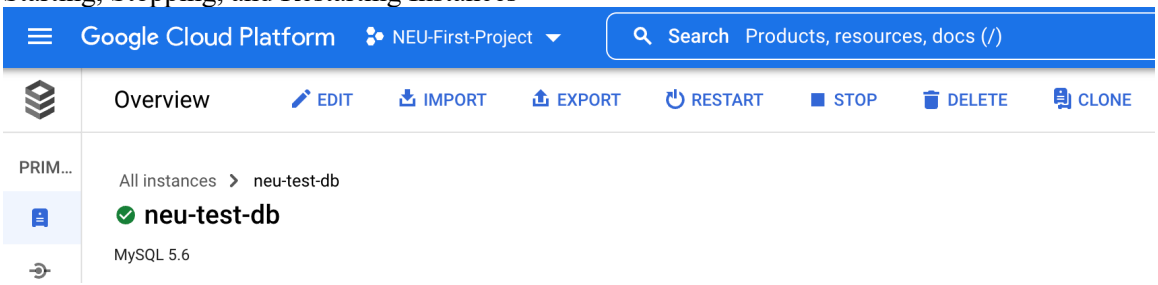
7. Click **Create (It will take approximately 10 ~ 15 minutes)**

B. Maintaining MySQL instance

1. Viewing information about MySQL instance
    - Select an instance to view by clicking the instance name.
    - Cloud Platform Console displays the Instance details pane for the instance.



2. Starting, Stopping, and Restarting Instances



- Starting an Instance
    - Select an instance to start by clicking the instance name.
    - Click Start, in the button bar at the top of the page.
    - In the Start database instance? dialog box, click Start.
    - The activation policy of the instance is set to Always and the instance is started.

- Stopping an Instance
    - Select an instance to restart by clicking the instance name.

- Click Stop, in the button bar at the top of the page.
- In the Stop database instance? dialog box, click Stop.
- The activation policy of the instance is set to Off (Never) and the instance is stopped.

- Restarting an Instance
  - Select an instance to restart by clicking the instance name.
  - Click Restart, in the button bar at the top of the page.
  - In the Restart database instance dialog box, click Restart.

- Deleting an Instance
  - Select an instance to delete by clicking the instance name.
  - Click Delete.
  - In the Delete database instance dialog box, click OK.

C. Connecting to MySQL instance

There are many different ways to connect MySQL instance running on GCP that you created before. Among them, you are going to test the following methods:

1. Connecting MySQL instance from your PC using IP Addresses
2. Connecting MySQL Client from Compute Engine
3. Connecting from App Engine
4. Connecting to Cloud SQL from External Applications

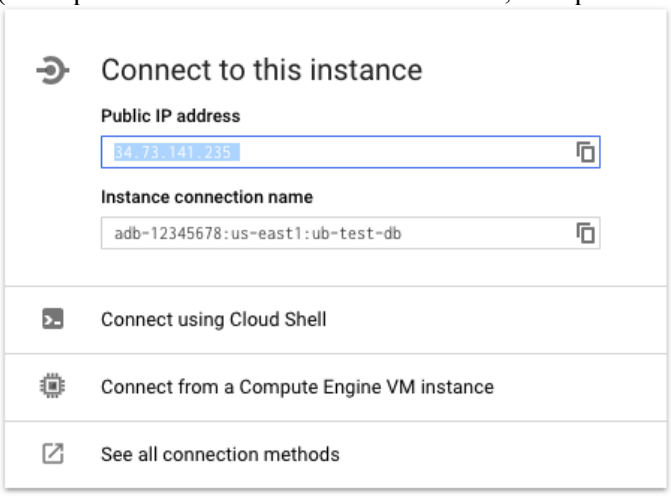
We will practice for the first method here, but you can use other connecting methods using the following instructions:

https://cloud.google.com/sql/docs/mysql/how-to

1. Connecting MySQL instance from your PC using IP Addresses
   - Install the mysql client on your PC (e.g., Windows or MacOS)
     1) Going to MySQL Community Server download page.
        https://dev.mysql.com/downloads/mysql/
        If you want to install MySQL client only
        https://dev.mysql.com/downloads/shell/
     2) Download the files

3) Install the Community Server (or extract zip file where you want), following the directions on the download page.

- Configure access to your Cloud SQL instance
  1) Click the instance to open its Overview page, and record its IPv4 address.
     - MySQL instance IP address: **11.11.11.11**
       (Your public IP address should be different, so replace 11.11.11.11 by yours)

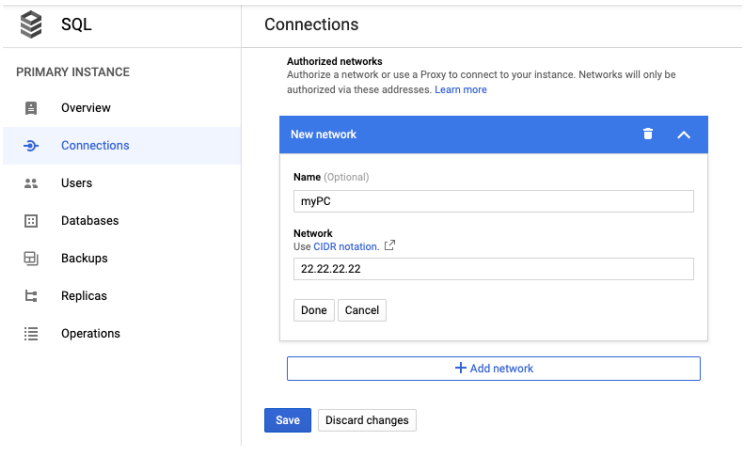


Skip 2) ~ 4) if you already created "All IP" network when you created an instance.

2) Log in to the client machine where your mysql client (i.e., your PC IP address) is installed to get external IP address
   If you are not sure how to get it, click http://ipv4.whatismyv6.com/
   - Client IP address: 22.22.22.22

Note: You need to update every time you connect to mysql, as it changes.

3) MySQL instance, and **Connection** menu. Select the **Authorized Network**.

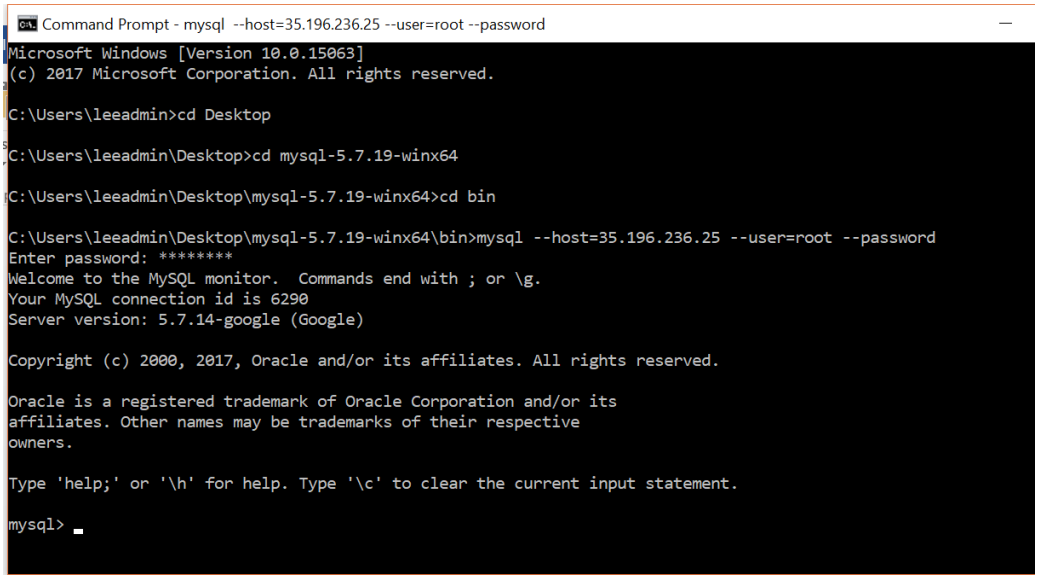


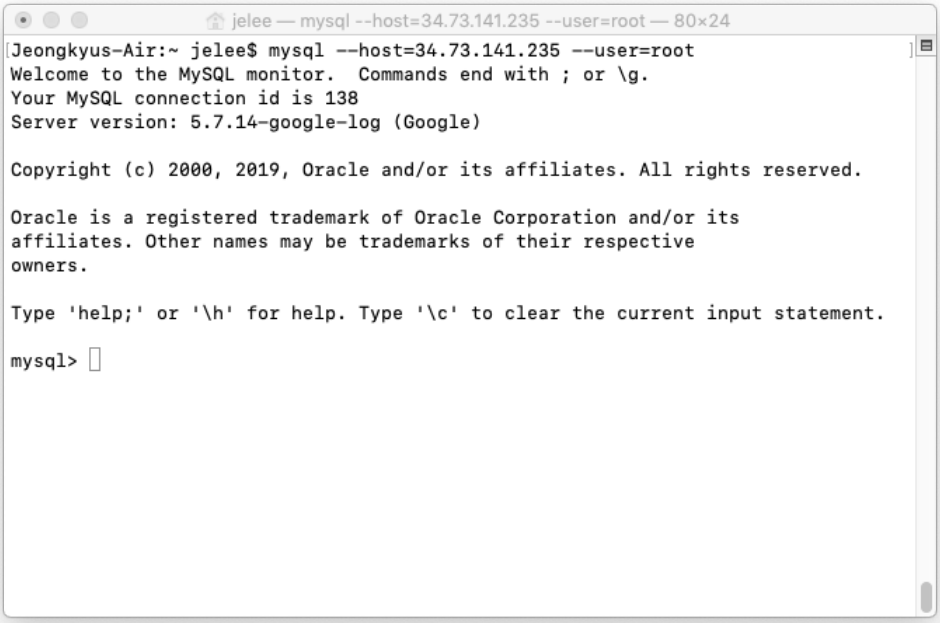
4) Click **Done**, then click **Save** at the bottom of your page to save your changes

- Connect to your Cloud SQL instance
  1) Windows:
     - Start cmd.exe on Windows
     - Go to where you extract mysql downloaded file on
     - **C:/Program Files/MySQL/MySQL Workbench 6.3 CE – This varies based on your path selection during installation**
  2) MacOS:
     - Start Terminal on MacOS
  3) Start the mysql client with IP address for MySQL instance

```
mysql --host=11.11.11.11 --user=root --password
```

  4) Enter your password
  5) You should see mysql prompt





For more information about GCP: https://goo.gl/NYDpBc

Exercise

After performing Section C, do the following and then submit the captured file to Canvas.

- Capture a screenshot of mysqls that shows success of creating a sample table.

- [**Optional**] Capture screenshot of mysql that can connect MySQL instance using the following methods:
    - Connecting MySQL Client from Compute Engine
    - Connecting from App Engine

- Submit the screen shot into Canvas. When you submit it, you should indicate your name and NEU ID

Lab 4: Connecting to MySQL Database

A. Connect to MySQL

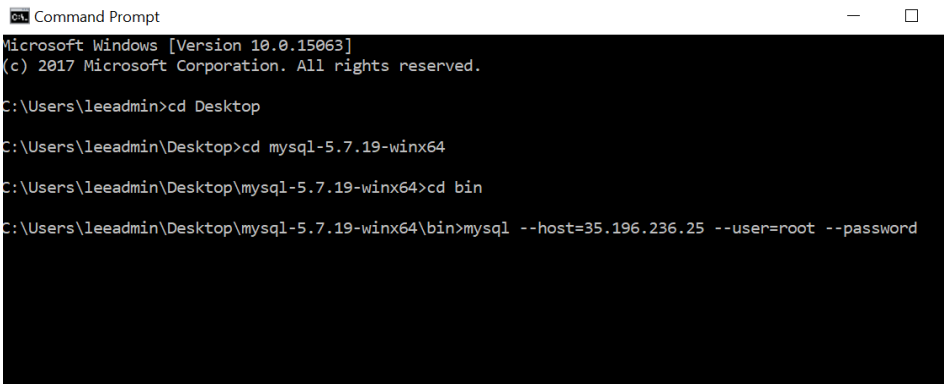
A1. Log on MySQL server using mysql client

- Start mysql client using cmd.exe (Windows) or Terminal (MacOS)
  Note that you can use any method to connect your MySQL instance

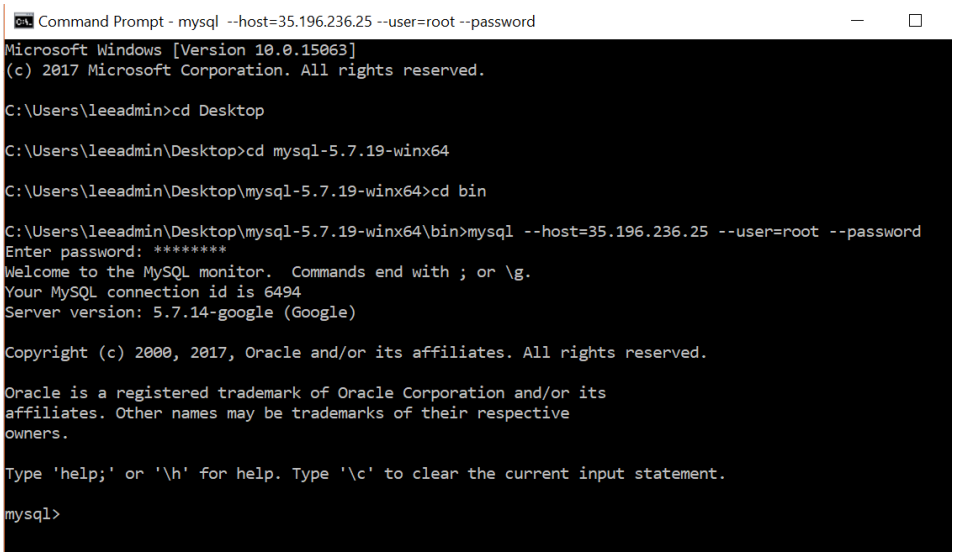
```
$ mysql --host=[IPv4 address] --user=root --password
```

For example,

```
$ mysql --host=11.11.11.11  --user=root -password
OR
$ mysql -h 11.11.11.11 -u root -p
```



NOTE: **User Name** is root, and **Password** is one you setup during creating MySQL instance. **Host** is IP address of your MySQL instance on GCP.



B. Taste mysql client

B1. help

```
mysql> help

For information about MySQL products and services, visit:
   http://www.mysql.com/
For developer information, including the MySQL Reference Manual, visit:
   http://dev.mysql.com/
To buy MySQL Enterprise support, training, or other products, visit:
   https://shop.mysql.com/

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear the current input statement.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
notee     (\t) Don't write into outfile.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an
argument.
status    (\s) Get status information from the server.
tee       (\T) Set outfile [to_outfile]. Append everything into given
outfile.
use       (\u) Use another database. Takes database name as argument.
charset   (\C) Switch to another charset. Might be needed for processing
binlog with multi-byte charsets.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.
resetconnection(\x) Clean session context.

For server side help, type 'help contents'

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.11 sec)

mysql> connect mysql;
```

```
Connection id:    6679
Current database: mysql

mysql> source Sec4/fill_help_tables.sql;
. . .

mysql> help contents;
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the
following
categories:
   Account Management
   Administration
   Compound Statements
   Data Definition
   Data Manipulation
   Data Types
   Functions
   Functions and Modifiers for Use with GROUP BY
   Geographic Features
   Help Metadata
   Language Structure
   Plugins
   Procedures
   Storage Engines
   Table Maintenance
   Transactions
   User-Defined Functions
   Utility
```

For example,

```
mysql> help logs;
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following
topics:
   PURGE BINARY LOGS
   SHOW
   SHOW BINARY LOGS

mysql> help show binary logs;
Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS

Lists the binary log files on the server. This statement is used as
part of the procedure described in [HELP PURGE BINARY LOGS], that shows
how to determine which logs can be purged.

mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
```

```
| binlog.000016 |    733481 |
+---------------+-----------+

URL: http://dev.mysql.com/doc/refman/5.7/en/show-binary-logs.html
```

B2. show

SHOW has many forms that provide information about databases, tables, columns, or status information about the server. This section describes

```
mysql> help show;

SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW CHARACTER SET [like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION func_name
SHOW CREATE PROCEDURE proc_name
SHOW CREATE TABLE tbl_name
SHOW CREATE TRIGGER trigger_name
SHOW CREATE VIEW view_name
SHOW DATABASES [like_or_where]
SHOW ENGINE engine_name {STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW EVENTS
SHOW FUNCTION CODE func_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW MASTER STATUS
SHOW OPEN TABLES [FROM db_name] [like_or_where]
SHOW PLUGINS
SHOW PROCEDURE CODE proc_name
SHOW PROCEDURE STATUS [like_or_where]
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW PROFILE [types] [FOR QUERY n] [OFFSET n] [LIMIT n]
SHOW PROFILES
SHOW RELAYLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW SLAVE HOSTS
SHOW SLAVE STATUS [FOR CHANNEL channel]
SHOW [GLOBAL | SESSION] STATUS [like_or_where]
SHOW TABLE STATUS [FROM db_name] [like_or_where]
SHOW [FULL] TABLES [FROM db_name] [like_or_where]
SHOW TRIGGERS [FROM db_name] [like_or_where]
SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
SHOW WARNINGS [LIMIT [offset,] row_count]

mysql> show databases;

mysql> show tables;
```

B3. Useful mysql commands

Switch to a database.
```
mysql> use [db name];
```

To see table's field formats.
```
mysql> describe [table name];
```

To return columns and column information.
```
mysql> show columns from [table name];
```

B4. Creating a new user

Creating a new user. Login as root.
mysql -u root -p
```
mysql> CREATE USER 'username' IDENTIFIED BY 'password';
mysql> grant all privileges on [db_name].* to 'username';
```

For example, create user 'jelee' with password '1234'
```
mysql> CREATE USER 'jelee' IDENTIFIED BY '1234';
```

Create a new database "test"
```
mysql> create database test;
Query OK, 1 row affected (0.03 sec)

mysql> use test;
```

Grant all permission on "test" database to jelee
```
mysql> grant all privileges on test.* to 'jelee';
```
* Note that since MySQL 8 it is not allowed to grant *.* to user (i.e., all permission to all databases). So, you need to create database first, then grant all permission on database to a user.

Creating a new user. Login as a new user.
mysql -u jelee -p
```
mysql> use test
```

Change a users password from MySQL prompt. Login as root. Set the password. Update privileges.
mysql -u root -p
```
mysql> SET PASSWORD FOR 'user'@'hostname' = PASSWORD('password');
mysql> flush privileges;
```

C. Catalog Database in MySQL (i.e., INFORMATION_SCHEMA)

% start mysql using information_schema
```
mysql> connect information_schema;

mysql> use information_schema
Database changed
mysql> show tables;
```

```
+---------------------------------------+
| Tables_in_information_schema          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
. . . . . .
| INNODB_BUFFER_POOL_STATS              |
| INNODB_SYS_COLUMNS                    |
| INNODB_SYS_FOREIGN                    |
| INNODB_SYS_TABLESTATS                 |
+---------------------------------------+
```

% Related to TABLE
TABLES
```
mysql> desc TABLES;
mysql> SELECT table_name FROM tables;
```

% Related to INDEX
INNODB_SYS_INDEXES
```
mysql> desc INNODB_SYS_INDEXES;
SQL> SELECT name, table_id, type FROM INNODB_SYS_INDEXES;
```

% Related to COLUMN
COLUMNS
```
mysql> desc COLUMNS;
mysql> SELECT ORDINAL_POSITION, COLUMN_NAME, DATA_TYPE, NUMERIC_PRECISION
    -> FROM COLUMNS
    -> WHERE TABLE_NAME = 'EMPLOYEE';
```

% Related to VIEWS
VIEWS
```
mysql> desc VIEWS;
mysql> SELECT table_name, VIEW_DEFINITION FROM VIEWS;
```

% Related to CONSTRAINTS
TABLE_CONSTRAINTS
```
mysql> desc TABLE_CONSTRAINTS;
mysql> SELECT constraint_name, table_name, constraint_type
    ->  FROM TABLE_CONSTRAINTS ;
```

Exercise

Please, complete the following and then submit the captured file to Canvas.

- Capture a screenshot of mysqls that shows success of the following commands
    - Connect to MySQL on GCP
    - Create a user
    - Create a database
    - Show databases
- Submit the screen shots into Canvas.

Lab 5: Basic SQL

Use lab5 folder

- Start mysql> at **lab5** directory
- Save the file and extract it under Lab folder
- Remember to start by creating a silicon database to use.

As root user

```
mysql> create database silicon;
mysql> grant all privileges on silicon.* to 'jelee';
```

As development user (e.g., 'jelee' here)

```
mysql> use silicon;
```

A. Data Definition Language (DDL)

A1. Create table

% Create EMPLOYEE table (To save time, COPY and PASTE the script from cr_company.sql)

```
mysql> CREATE TABLE employee
  -> (fname            VARCHAR(8),
  ->  minit            VARCHAR(2),
  ->  lname            VARCHAR(8),
  ->  ssn              VARCHAR(9) NOT NULL,
  ->  bdate            DATE,
  ->  address          VARCHAR(27),
  ->  sex              VARCHAR(1),
  ->  salary           INT(7) NOT NULL,
  ->  superssn         VARCHAR(9),
  ->  dno              INT(1) NOT NULL) ;

Query OK, 0 rows affected (0.18 sec)
```

% Describe Table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| fname     | varchar(8)  | YES  |     | NULL    |       |
| minit     | varchar(2)  | YES  |     | NULL    |       |
| lname     | varchar(8)  | YES  |     | NULL    |       |
| ssn       | varchar(9)  | NO   |     | NULL    |       |
| bdate     | date        | YES  |     | NULL    |       |
| address   | varchar(27) | YES  |     | NULL    |       |
| sex       | varchar(1)  | YES  |     | NULL    |       |
| salary    | int(7)      | NO   |     | NULL    |       |
| super_ssn | varchar(9)  | YES  |     | NULL    |       |
| dno       | int(1)      | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

% run script on mysql client

Use arrow to see the last command
```
mysql>
```

Execute SQL from a file (e.g., in another cmd.exe (Win) to create `cr_emp.sql` file using notepad.exe or any text editor. And copy and paste `CREATE TABLE employee . . .` Then, save it. For Mac users, vim on terminal or textedit can be used to create `cr_emp.sql` file.

```
mysql> source cr_emp.sql;
ERROR 1050 (42S01): Table 'employee' already exists
```

You will get an error, since name is already used by an existing table

```
mysql> drop table employee;
Query OK, 0 rows affected (0.05 sec)

mysql> commit;
Query OK, 0 rows affected (0.03 sec)

mysql> source cr_emp.sql;
Query OK, 0 rows affected (0.05 sec)

mysql> describe employee;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| fname    | varchar(8)  | YES  |     | NULL    |       |
| minit    | varchar(2)  | YES  |     | NULL    |       |
| lname    | varchar(8)  | YES  |     | NULL    |       |
| ssn      | varchar(9)  | NO   |     | NULL    |       |
| bdate    | date        | YES  |     | NULL    |       |
| address  | varchar(27) | YES  |     | NULL    |       |
| sex      | varchar(1)  | YES  |     | NULL    |       |
| salary   | int(7)      | NO   |     | NULL    |       |
| superssn | varchar(9)  | YES  |     | NULL    |       |
| dno      | int(1)      | NO   |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
10 rows in set (0.03 sec)
```

% DROP Table
```
mysql> DROP TABLE employee;
```

% Create Table EMPLOYEE with Primary Key

```
mysql> CREATE TABLE employee
  -> (fname            VARCHAR(8),
  ->  minit            VARCHAR(2),
  ->  lname            VARCHAR(8),
  ->  ssn              VARCHAR(9) NOT NULL,
  ->  bdate            DATE,
  ->  address          VARCHAR(27),
  ->  sex              VARCHAR(1),
  ->  salary           INT(7) NOT NULL,
  ->  superssn         VARCHAR(9),
```

```
  ->  dno              INT(1) NOT NULL,
  ->  PRIMARY KEY (ssn) );
```

% Otherwise, you can modify cr_emp.sql using editor, then source cr_emp.sql;

% Create Table DEPARTMENT with PRIMARY Key (To save time, COPY and PASTE the script from cr_company.sql and add PRIMARY KEY)

```
mysql> DROP TABLE department;
mysql> CREATE TABLE department(dnumber      INT(1),
    -> dname          VARCHAR(15),
    -> mgr_ssn        VARCHAR(9),
    -> mgr_start_date DATE,
    -> PRIMARY KEY (dnumber) );
```

% Create Table EMPLOYEE with Foreign Key (To save time, COPY and PASTE the script from cr_company.sql and add PRIMARY KEY and FOREIGN KEY)

```
mysql> DROP TABLE employee;

mysql> CREATE TABLE employee  (fname      VARCHAR(8),
    -> minit          VARCHAR(2),
    -> lname          VARCHAR(8),
    -> ssn            VARCHAR(9) NOT NULL,
    -> bdate          DATE,
    -> address        VARCHAR(27),
    -> sex            VARCHAR(1),
    -> salary         INT(7) NOT NULL,
    -> super_ssn      VARCHAR(9),
    -> dno            INT(1) NOT NULL,
    -> PRIMARY KEY (ssn),
    -> FOREIGN KEY (dno) REFERENCES department (dnumber)
    -> ON DELETE CASCADE );

mysql> insert into employee values
    -> ('Joe', 'M', 'Smith', '123456789', NOW(),
    -> '123 Smith St.', 'm', 45000, '123456789', 1);

ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails (`silicon`.`employee`, CONSTRAINT `fk_dno` FOREIGN KEY
(`dno`) REFERENCES `department` (`dnumber`) ON DELETE CASCADE)
```

% Why? And how to resolve the issue?

```
mysql> insert into department values (1,'HEADQUARTERS','888665555',NOW());

mysql> insert into employee values
    -> ('Joe', 'M', 'Smith', '123456789', NOW(),
    -> '123 Smith St.', 'm', 45000, '123456789', 1);

Query OK, 1 row affected, 1 warning (0.05 sec)

mysql> select * from employee;
+-------+-------+-------+-----------+------------+---------------+------+--
------+-----------+-----+
```

```
| fname | minit | lname | ssn       | bdate      | address       | sex  |
salary | super_ssn | dno |
+-------+-------+-------+-----------+------------+---------------+------+--
------+-----------+-----+
| Joe   | M     | Smith | 123456789 | 2017-09-27 | 123 Smith St. | m    |
45000 | 123456789 |   1 |
+-------+-------+-------+-----------+------------+---------------+------+--
------+-----------+-----+
1 row in set (0.08 sec)
```

A2. Alter Table

```
mysql> describe department;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| dnumber        | int(1)      | NO   | PRI | NULL    |       |
| dname          | varchar(15) | YES  |     | NULL    |       |
| mgr_ssn        | varchar(9)  | YES  |     | NULL    |       |
| mgr_start_date | date        | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
4 rows in set (0.03 sec)
```

% Modify Table

```
mysql> ALTER TABLE department
    ->  ADD (manager VARCHAR(8)) ;
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC department;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| dnumber        | int(1)      | NO   | PRI | NULL    |       |
| dname          | varchar(15) | YES  |     | NULL    |       |
| mgr_ssn        | varchar(9)  | YES  |     | NULL    |       |
| mgr_start_date | date        | YES  |     | NULL    |       |
| manager        | varchar(8)  | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
5 rows in set (0.03 sec)
```

% What is difference?

```
mysql> ALTER TABLE department
    -> MODIFY COLUMN manager VARCHAR(15);

mysql> DESC department;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| dnumber        | int(1)      | NO   | PRI | NULL    |       |
| dname          | varchar(15) | YES  |     | NULL    |       |
| mgr_ssn        | varchar(9)  | YES  |     | NULL    |       |
| mgr_start_date | date        | YES  |     | NULL    |       |
| manager        | varchar(15) | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
```

```
5 rows in set (0.03 sec)
```
% What is difference?

A3. Create and Drop Index

```
mysql> CREATE INDEX employee_ssn_idx
    -> ON employee (ssn);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0


mysql> DROP INDEX employee_superssn_idx
    -> ON employee;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

A4. Create and Drop View

% Create View emp_dno_1 limited in columns (fname, lname, dno) and set of data (WHERE dno=1) from the employee table.

```
mysql> CREATE VIEW emp_dno_1
    -> AS SELECT fname, lname, dno
    -> FROM employee
    -> WHERE dno = 1;
Query OK, 0 rows affected (0.08 sec)

mysql> desc emp_dno_1;

mysql> DROP VIEW emp_dno_1;
```

% Create View dept_manager using join

```
mysql> CREATE VIEW dept_managers AS
    -> SELECT dnumber, dname, mgr_ssn, lname, fname
    -> FROM   employee, department
    -> WHERE  employee.ssn = department.mgr_ssn ;
Query OK, 0 rows affected (0.05 sec)

mysql> desc dept_managers;
```

% Create View dept_average_salary using aggregate function.

```
mysql> CREATE VIEW dept_average_salary AS
    -> SELECT dnumber, dname, AVG(salary) AS average_salary
    -> FROM   department, employee
    -> WHERE  employee.dno = department.dnumber
    -> GROUP BY dnumber, dname ;
Query OK, 0 rows affected (0.05 sec)

mysql> desc dept_average_salary;
```

B. Data Manipulation Language (DML)

B0. Create COMPANY Database

```
mysql> source cr_company.sql
```

% If you want to create COMPANY database with Primary Key and Foreign Keys, use the following script:

```
mysql> source cr_company_full.sql

+---------------------+
| Tables_in_lab       |
+---------------------+
| department          |
| dependent           |
| dept_average_salary |
| dept_locations      |
| dept_managers       |
| emp_department_1    |
| employee            |
| project             |
| works_on            |
+---------------------+
9 rows in set (0.05 sec)
```

B1. SELECT

% Every query in B.1, B.2 and B3 is scripted on the file named '**Q#.sql**' in which # indicates the query number. For example, to run '**Query 0**', type

```
mysql> source Q0.sql
```

```
% SELECT – FROM – WHERE
```

Query 0: Retrieve the birth date and address of the employee(s) whose name is 'Joe M. SMITH'.

```
mysql> SELECT bdate, address
    -> FROM employee
    -> WHERE fname = 'JOHN'
    ->   AND minit = 'B'
    ->   AND lname = 'SMITH';

    Or

mysql> source Q0.sql
+------------+--------------------------+
| bdate      | address                  |
+------------+--------------------------+
| 2022-05-08 | 731 FONDREN, HOUSTON, TX |
+------------+--------------------------+
1 row in set (0.03 sec)
```

Query 1: Retrieve the name and address of all employees who work for the 'RESEARCH' department.

```
mysql> SELECT fname, lname, address
    ->   FROM employee, department
    ->  WHERE dname = 'RESEARCH'
    ->    AND dnumber = dno;
+----------+---------+-------------------------+
| fname    | lname   | address                 |
+----------+---------+-------------------------+
| JOHN     | SMITH   | 731 FONDREN, HOUSTON, TX |
| FRANKLIN | WONG    | 638 VOSS,HOUSTON TX      |
| RAMESH   | NARAYAN | 975 FIRE OAK, HUMBLE, TX |
| JOYCE    | ENGLISH | 5631 RICE, HOUSTON, TX   |
+----------+---------+-------------------------+
4 rows in set (0.05 sec)
```

Query 2: For every project located in 'STAFFORD', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
mysql> SELECT pnumber, dnum, lname, address, bdate
    -> FROM project, department, employee
    -> WHERE dnum=dnumber
    -> AND mgr_ssn = ssn
    -> AND plocation = 'STAFFORD';
+---------+------+---------+-------------------------+------------+
| pnumber | dnum | lname   | address                 | bdate      |
+---------+------+---------+-------------------------+------------+
|      10 |    4 | WALLACE | 291 BERRY, BELLAIRE, TX | 2017-08-21 |
|      30 |    4 | WALLACE | 291 BERRY, BELLAIRE, TX | 2017-08-21 |
+---------+------+---------+-------------------------+------------+
2 rows in set (0.06 sec)
```

% Aliasing

Query 1A:

```
mysql> SELECT fname, employee.lname, address
    ->   FROM employee, department
    ->  WHERE department.dname = 'RESEARCH'
    ->    AND  department.dnumber = employee.dno;
+----------+---------+-------------------------+
| fname    | lname   | address                 |
+----------+---------+-------------------------+
| JOHN     | SMITH   | 731 FONDREN, HOUSTON, TX |
| FRANKLIN | WONG    | 638 VOSS,HOUSTON TX      |
| RAMESH   | NARAYAN | 975 FIRE OAK, HUMBLE, TX |
| JOYCE    | ENGLISH | 5631 RICE, HOUSTON, TX   |
+----------+---------+-------------------------+
4 rows in set (0.05 sec)
```

Query 8: For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
mysql> SELECT E.fname, E.lname, S.fname, S.lname
    ->   FROM employee E, employee S
```

```
    ->  WHERE E.super_ssn = S.ssn;
+----------+---------+----------+---------+
| fname    | lname   | fname    | lname   |
+----------+---------+----------+---------+
| JOHN     | SMITH   | FRANKLIN | WONG    |
| RAMESH   | NARAYAN | FRANKLIN | WONG    |
| JOYCE    | ENGLISH | FRANKLIN | WONG    |
| ALICIA   | ZELAYA  | JENNIFER | WALLACE |
| AHMAD    | JABBAR  | JENNIFER | WALLACE |
| FRANKLIN | WONG    | JAMES    | BORG    |
| JENNIFER | WALLACE | JAMES    | BORG    |
+----------+---------+----------+---------+
7 rows in set (0.06 sec)
```

% missing WHERE

Queries 9 and 10: Slect all EMPLOYEE ssns (Q9) and all combinations of EMPLOYEE ssn and DEPARTMENT dname (Q10) in the database.

```
mysql> SELECT ssn
    ->   FROM employee;

mysql> SELECT ssn, dname
    ->   FROM employee, department;
   (How many rows are returned?)
```

% Asterisk

Q1c:

```
mysql> SELECT *
    -> FROM employee
    -> WHERE DNO = 5;
```

% Set operations in SQL (DISTICT, UNION)

Query 11: Retrieve the salary of every employee and all distinct salary values.

```
mysql> SELECT  ALL salary
    -> FROM employee;

mysql> SELECT  DISTINCT salary
  -> FROM employee;
```

% What is the difference?

Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'SMITH', either as a worker or as a manager of the department that controls the project.

```
mysql> SELECT DISTINCT pnumber
    -> FROM project, department, employee
    -> WHERE dnum =  dnumber
    ->   AND  mgr_ssn = ssn
    ->   AND lname = 'SMITH'
    -> UNION
    -> SELECT DISTINCT pnumber
    -> FROM  project, works_on, employee
```

```
    -> WHERE pnumber =  pno
    ->   AND essn = ssn
    ->   AND  lname = 'SMITH';
```

% Pattern matching (LIKE)
Query 12: Retrieve all employees whose address is in Houston, Texas.

```
mysql> SELECT  fname, lname
    -> FROM  employee
    -> WHERE  address LIKE '%HOUSTON, TX%';
```

Query 12A: Find all employees who were born during the 2020s.

```
mysql> SELECT  fname, lname
    -> FROM    employee
    -> WHERE   bdate LIKE '__2_______';
     (2 times of '_' + '1' + 7 times of "_")
```

Query 13: Show the resulting salaries if every employee working on the 'PRODUCTX' project is given a 10 percent raise.

```
mysql> SELECT  fname, lname, 1.1*salary AS INCREASED_SAL
    -> FROM employee, works_on, project
    -> WHERE ssn=essn AND pno=pnumber AND pname='PRODUCTX';
```

Query 14: Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

```
mysql> SELECT  *
    -> FROM employee
    -> WHERE (salary BETWEEN 30000 AND 40000) AND dno = 5;
```

Query 15: Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, first name.

```
mysql> SELECT      dname, lname, fname, pname
    -> FROM        department, employee, works_on, project
    -> WHERE       dnumber=dno AND ssn=essn AND pno=pnumber
    -> ORDER BY    dname, lname, fname;
```

B2. Complex SELECT

% NULL
Query 18: Retrieve the names of all employees who do not have supervisors.

```
mysql> SELECT fname, lname
    -> FROM employee
    -> WHERE super_ssn IS null;
```

#replace with NOT NULL

```
mysql> SELECT fname, lname
    ->     FROM employee
    -> WHERE super_ssn IS NOT null;
```

```
+----------+---------+
| fname    | lname   |
+----------+---------+
| JOHN     | SMITH   |
| FRANKLIN | WONG    |
| ALICIA   | ZELAYA  |
| JENNIFER | WALLACE |
| RAMESH   | NARAYAN |
| JOYCE    | ENGLISH |
| AHMAD    | JABBAR  |
+----------+---------+
7 rows in set (0.05 sec)
```

% Nested Queries
Q4A:

```
mysql> SELECT DISTINCT pnumber
    -> FROM project
    -> WHERE pnumber IN (SELECT pnumber
    ->        FROM project, department, employee
    ->        WHERE dnum=dnumber
    ->         AND mgr_ssn=ssn
    ->         AND lname = 'SMITH')
    ->     OR pnumber IN (SELECT pno
    ->        FROM works_on, employee
    ->       WHERE essn=ssn
    ->         AND lname = 'SMITH');
```

Query 16: Retrieve the name of each employee who has a dependent with the same first name and same sex as the employee.

```
mysql> SELECT  E.fname, E.lname
    -> FROM  employee E
    -> WHERE E.ssn IN (SELECT essn
    ->      FROM dependent
    ->     WHERE E.fname = dependent_name
    ->       AND E.sex=sex);
```

% Correlated Nested queries
Q16A:

```
mysql> SELECT E.fname, E.lname
    -> FROM  employee E, dependent D
    -> WHERE E.ssn=D.essn
    ->   AND  E.sex=D.sex
    ->   AND  E.fname=D.dependent_name;
```

Query 3: Retrieve the name of each employee who works on all the projects controlled by department number 4.

```
mysql> SELECT fname, lname
    -> FROM employee
    -> WHERE NOT EXISTS (SELECT pnumber FROM project
    ->                   WHERE dnum=4 AND pnumber NOT IN
    ->      (SELECT pno FROM works_on WHERE employee.ssn=works_on.essn));
```

Q3B:

```
mysql> SELECT      fname, lname
    ->   FROM        employee
    ->  WHERE       NOT EXISTS (SELECT *
    ->                          FROM works_on B
    ->                      WHERE (B.pno IN (SELECT pnumber
    ->                                       FROM   project
    ->                                       WHERE  dnum=4)
    ->                              AND
    ->                                    NOT EXISTS (SELECT *
    ->                                        FROM   works_on C
    ->                                         WHERE  C.essn=ssn
    ->                                         AND    C.pno=B.pno)));
```

% Explicit Sets
Query 17: Retrieve the social security numbers of all employees who work on project number 1,2, or 3.

```
mysql> SELECT  DISTINCT essn
    -> FROM  works_on
    -> WHERE  pno IN (1,2,3);
```

% Outer Joins
Q8C:

```
mysql> SELECT E.lname, S.lname
    -> FROM employee E left outer join employee S
    -> ON E.super_ssn  = S.ssn;
```

% Aggregate Functions
Query 19: Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
mysql> SELECT  SUM(salary), MAX(salary), MIN(salary), AVG(salary)
    -> FROM employee;
```

Query 20: Find the sum of the salaries of employees in the 'RESEARCH' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
mysql> SELECT  SUM(salary), MAX(salary), MIN(salary), AVG(salary)
    -> FROM employee, department
    -> WHERE  dno=dnumber AND dname='RESEARCH';
```

Queries 21 and 22: Retrieve the total number of employees in the company (Q21) and the number of employees in the 'RESEARCH' department (Q22).

```
mysql> SELECT     COUNT(*)
    ->   FROM     employee;

mysql> SELECT  COUNT(*)
    -> FROM  employee, department
    -> WHERE  dno=dnumber AND dname='RESEARCH';
```

Query 23: Count the number of distinct salary values in the database.

```
mysql> SELECT COUNT(DISTINCT salary) FROM employee;
```

% Grouping (GROUP BY, HAVING)
Query 24: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
mysql> SELECT dno, COUNT(*), AVG(salary)
    -> FROM   employee
    -> GROUP BY dno;
```

Query 25: For each project, retrieve the project number, the project name, and the number of employees who work on that project.

```
mysql> SELECT      pnumber, pname, COUNT(*)
    -> FROM        project, works_on
    -> WHERE       pnumber=pno
    -> GROUP BY pnumber, pname;
```

Query 26: For each project, on which more than two employees work, retrieve the project name, and the number of employess who work on the project.

```
mysql> SELECT      pnumber, pname, COUNT(*)
    -> FROM        project, works_on
    -> WHERE       pnumber=pno
    -> GROUP BY pnumber, pname
    -> HAVING      COUNT(*) > 2;
```

Query 27: For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.

```
mysql> SELECT     pnumber, pname, COUNT(*)
    -> FROM       project, works_on,employee
    -> WHERE      pnumber=pno AND ssn=essn AND dno=5
    -> GROUP BY pnumber, pname;
```

Query 28: For each department that has more than three employees, retrieve the department number and the number of its employees who are making more than $25,000.

```
mysql> SELECT     dnumber, COUNT(*)
    -> FROM       department, employee
    -> WHERE      dnumber=dno AND salary>25000
    -> AND       dno in (SELECT  dno
    ->                   FROM    employee
    ->                   GROUP BY dno
    ->                   HAVING  COUNT(*)>3)
    -> GROUP BY dnumber;
```

B3. Insert, Delete, and Update

```
% INSERT
U1:
mysql> INSERT INTO employee
    ->  VALUES ('RICHARD','K','MARINI','653298653','1969-04-30',
```

```
    ->          '98 OAK FOREST, KATY, TX','M',37000,'653298653',4)

mysql> INSERT INTO employee (fname, lname, dno, ssn, salary)
    ->  VALUES ('RICHARD','MARINI',4,'653298653', 40000);
```

U2:

```
mysql> INSERT INTO employee (fname, lname, ssn, dno,salary)
    -> VALUES ('ROBERT','HATCHER','980735245',2,40000);

mysql> INSERT INTO employee (fname, lname, ssn, dno)
    -> VALUES ('ROBERT','HATCHER','980735245',2);
(Why error?)
```

U3:

```
mysql> CREATE TABLE depts_info
    -> (dept_name varchar(15),
    -> no_of_emps int(2),
    -> total_sal int(6));

mysql> INSERT INTO depts_info (dept_name, no_of_emps, total_sal)
    -> SELECT dname, count(*), sum(salary)
    -> FROM department, employee
    -> WHERE dnumber = dno
    -> GROUP BY dname;

mysql> select * from depts_info;
```

% DELETE

U4:

```
mysql> DELETE FROM employee WHERE lname = 'BROWN';
mysql> DELETE FROM employee WHERE ssn='123456789';
mysql> DELETE FROM employee
    -> WHERE dno in (SELECT dnumber
    ->                 FROM department
    ->                WHERE dname='RESEARCH');
```

% UPDATE

U5:

```
mysql> UPDATE project
    -> SET plocation='BELLAIRE', dnum=5
    -> WHERE pnumber = 10;
```

U6:

```
mysql> UPDATE employee
    -> SET salary=salary*1.1
    -> WHERE dno IN (SELECT dnumber
    ->                 FROM department
    ->                 WHERE dname = 'RESEARCH');
```

Exercise

Exercise 1: Creating and Altering Table

As an exercise, create a table called **students** with the following columns and data types:

```
Column Name      Data Type
StudentID        INT(5)       NOT NULL
Name             VARCHAR(25)
Major            VARCHAR(15)
GPA              DECIMAL(6,3)
```

Create another table called **courses** with the following columns and data types:

```
Column Name      Data Type
StudentID        INT(5)           NOT NULL
CourseNumber     VARCHAR(15)      NOT NULL
CourseName       VARCHAR(25)
Semester         VARCHAR(10)
Year             INT(4)
Grade            VARCHAR(2)
```

Use the DESCRIBE command to display the data types of the columns after each table is created. Next, use the ALTER TABLE statement to add the following column to the **students** table:

```
Column Name      Data Type
TutorID          INT(5)
```

Use the ALTER TABLE statement to define the StudentID as the PRIMARY KEY for the **students** table.
Use the ALTER TABLE statement to define the StudentID and CourseNumber as the PRIMARY KEY for the **courses** table. To do this, list both of the column names separated by a comma.

Use the ALTER TABLE statement to define StudentID in the **courses** table as a FOREIGN KEY that references the StudentID in the **students** table.

Finally, add some data to the **students** and **courses** tables (simply copy and paste these statements into your script file to add the data):

```
INSERT INTO students VALUES (101, 'Bill', 'CIS', 3.45,  102);
INSERT INTO students VALUES (102, 'Mary', 'CIS', 3.10,  NULL);
INSERT INTO students VALUES (103, 'Sue',  'Marketing', 2.95, 102);
INSERT INTO students VALUES (104, 'Tom',  'Finance', 3.5, 106);
INSERT INTO students VALUES (105, 'Alex', 'CIS', 2.75, 106);
INSERT INTO students VALUES (106, 'Sam',  'Marketing', 3.25, 103);
INSERT INTO students VALUES (107, 'Jane', 'Finance', 2.90, 102);
INSERT INTO courses VALUES (101, 'CIS3400', 'DBMS I', 'FALL', 1997, 'B+');
INSERT INTO courses VALUES (101, 'CIS3100', 'OOP I', 'SPRING', 1999, 'A-');
INSERT INTO courses VALUES (101, 'MKT3000', 'Marketing', 'FALL', 1997,'A');
INSERT INTO courses VALUES (102, 'CIS3400', 'DBMS I', 'SPRING', 1997,'A-');
INSERT INTO courses VALUES (102, 'CIS3500', 'Network I','SUMMER',1997,'B');
INSERT INTO courses VALUES (102, 'CIS4500', 'Network II','FALL',1997,'B+');
INSERT INTO courses VALUES (103, 'MKT3100', 'Advertise','SPRING',1998,'A');
INSERT INTO courses VALUES (103, 'MKT3000', 'Marketing', 'FALL', 1997,'A');
```

```
INSERT INTO courses VALUES (103, 'MKT4100', 'Marketing II', 'SUMMER', 1998,
'A-');
```

Exercise 2: More Queries with UNIVERSITY database

Using UNIVERSITY database that you created in Exercise 1, create SQL below:

1. Average GPA of all students
2. Average GPA of Finance and CIS students
3. Give the name of the student with the highest GPA
4. Show the students with the GPA grades in each major
5. Show the students with the GPA grades in each major ordered by GPA in descending order
6. Provide a listing of each student and the name of their tutor (Recursive query). Note that recursive query is only available on MySQL 8.X. So, if you are using 5.X, you can ignore this query.
7. How many students does each tutor work with?
8. Add .05 to all of the Marketing major's GPA's
9. Change Sam's tutor from Sue to Jane
10. For any student who is currently majoring in CIS and who has a GPA of less than 3.0, change their major to Marketing.

Lab 6: Advanced SQL

Use lab6 folder

- Start mysql> at **lab6** directory
- Save the file and extract it under Lab folder
- Remember to start by creating a silicon database to use.

A. Advanced DDL

A1. CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database.

1. **CREATE TABLE Syntax**

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

Query 1: CREATE TABLE Examples:

```
mysql> create table student(
    -> sno varchar(10) primary key,
    -> sname varchar(20),
    -> sage int(2),
    -> ssex varchar(5) );
```

Common Datatypes:
VARCHAR(size)*: Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis.*

INT(size)*: Holds a variable length number(only integers).*

2. **DESCRIBE TABLE Syntax**

```
DESCRIBE table_name     or
DESC table_name
```

DESCRIBE TABLE Examples:

```
mysql> describe student;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| sno   | varchar(10) | NO   | PRI | NULL    |       |
| sname | varchar(20) | YES  |     | NULL    |       |
```

```
| sage  | int(2)      | YES  |     | NULL    |       |
| ssex  | varchar(5)  | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Query 2: Continue to create another 3 tables

```
mysql> create table teacher(
    -> tno varchar(10),
    -> tname varchar(20) );

mysql> create table course(
    -> cno varchar(10),
    -> cname varchar(20),
    -> tno varchar(20) );

mysql> create table sc(
    -> sno varchar(10),
    -> cno varchar(10),
    -> score decimal(4,2) );
```

A2. Constraints for Table Statement

Constraints are used to limit the type of data that can go into a table. Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement). We will focus on the following constraints:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

1. NOT NULL Constraint

The NOT NULL constraint enforces a column to NOT accept NULL values. The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record or update a record without adding a value to this field.

Query 3: The following SQL enforces the "sno" column to not accept NULL values:

```
mysql> drop table student;
Query OK, 0 rows affected (0.06 sec)

mysql> create table student(
    ->  sno varchar(10) not null,
    ->  sname varchar(20),
    ->  sage int(2),
    ->  ssex varchar(5)
    -> );
Query OK, 0 rows affected (0.17 sec)
```

```
mysql> desc student;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| sno   | varchar(10) | NO   |     | NULL    |       |
| sname | varchar(20) | YES  |     | NULL    |       |
| sage  | int(2)      | YES  |     | NULL    |       |
| ssex  | varchar(5)  | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

2. UNIQUE Constraint

The UNIQUE constraint uniquely identifies each record in a table. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns. A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it. Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Query 4: The following SQL create a UNIQUE constraint on the "sno" column when the "student" table is created:

```
mysql> drop table student;
Query OK, 0 rows affected (0.16 sec)

mysql> create table student(
    ->  sno varchar(10) not null unique,
    ->  sname varchar(20),
    ->  sage int(2),
    ->  ssex varchar(5)
    -> );
```

3. PRIMARY KEY Constraint:

Query 5: The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain unique values. A primary key column cannot contain NULL values. Each table should have a primary key, and each table can have only ONE primary key.

```
mysql> drop table student;
Query OK, 0 rows affected (0.09 sec)

mysql> create table student(
    ->  sno varchar(10) primary key,
    ->  sname varchar(20),
    ->  sage int(2),
    ->  ssex varchar(5)
    -> );
```

Or **Query 5A**:

```
mysql> drop table student;
Query OK, 0 rows affected (0.11 sec)

mysql> create table student(
    ->  sno varchar(10),
```

```
    ->  sname varchar(20),
    ->  sage int(2),
    ->  ssex varchar(5),
    ->  constraint pk_student primary key (sno)
    -> );
```

PRIMARY KEY Constraint on ALTER TABLE

Query 6: To create a PRIMARY KEY constraint on the “sno" column when the table is already created, use the following SQL:

```
% ALTER TABLE student
% ADD PRIMARY KEY (sno);

mysql> drop table student;
Query OK, 0 rows affected (0.06 sec)

mysql> create table student(
    ->   sno varchar(10),
    ->   sname varchar(20),
    ->   sage int(2),
    ->   ssex varchar(5));
Query OK, 0 rows affected (0.12 sec)

mysql> ALTER TABLE student
    ->  ADD PRIMARY KEY (sno);
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

To DROP a PRIMARY KEY Constraint

Query 7: To drop a PRIMARY KEY constraint, use the following SQL:

```
%ALTER TABLE student
% DROP PRIMARY KEY;

mysql> drop table student;
Query OK, 0 rows affected (0.06 sec)

mysql> create table student(
    ->  sno varchar(10),
    ->  sname varchar(20),
    ->  sage int(2),
    ->  ssex varchar(5),
    ->  constraint pk_student primary key (sno)
    -> );

mysql> ALTER TABLE student
    ->  DROP PRIMARY KEY;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

4. FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table. Let's illustrate the foreign key with an example. Look at the following two tables:

The “Persons” table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The “Orders” table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Note that the “P_Id” column in the “Orders” table points to the “P_Id” column in the “Persons” table. The “P_Id” column in the “Persons” table is the PRIMARY KEY in the “Persons” table. The “P_Id” column in the “Orders” table is a FOREIGN KEY in the “Orders” table.

FOREIGN KEY Constraint on CREATE TABLE

`Query 8`: The following SQL creates a FOREIGN KEY on the "P_Id" column when the "Orders" table is created:

```
mysql> source Q6.sql;

mysql> drop table sc;
Query OK, 0 rows affected (0.06 sec)

mysql> create table sc(
    ->   sno varchar(10),
    ->   cno varchar(10),
    ->   score decimal(4,2),
    ->   constraint sc_sno_fk foreign key (sno) references student(sno),
    ->   constraint sc_sno_cno_fk primary key (sno,cno)
    -> );
```

FOREIGN KEY Constraint on ALTER TABLE

`Query 9:` To drop a FOREIGN KEY constraint, use the following SQL:

```
mysql> ALTER TABLE sc
    -> DROP FOREIGN KEY sc_sno_fk;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

5. CHECK Constraint

CHECK Constraint is available ONLY at MySQL database +8.0 version.

6. DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified.

DEFAULT Constraint on CREATE TABLE

Query 12: The following SQL creates a DEFAULT constraint on the "City" column when the "Persons" table is created:

```
mysql> drop table persons;

mysql> CREATE TABLE Persons
    ->   (
    ->   P_Id int NOT NULL,
    ->   LastName varchar(255) NOT NULL,
    ->   FirstName varchar(255),
    ->   Address varchar(255),
    ->   City varchar(255) DEFAULT 'Sandnes'
    -> );
Query OK, 0 rows affected (0.16 sec)
```

DEFAULT Constraint on ALTER TABLE

Query 13: To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

```
mysql> ALTER TABLE Persons
    -> MODIFY COLUMN City VARCHAR(255) DEFAULT 'SANDNES';
```

A3. ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

1. ALTER TABLE Syntax

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype
```

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name
```

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
MODIFY column_name datatype
```

```
mysql> desc Persons;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| P_Id      | int(11)     | NO   |     | NULL    |       |
| LastName  | varchar(255)| NO   |     | NULL    |       |
| FirstName | varchar(255)| YES  |     | NULL    |       |
| Address   | varchar(255)| YES  |     | NULL    |       |
| City      | varchar(255)| YES  |     | SANDNES |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.05 sec)
```

2. ADD Column
We are going to add a new column "DateOfBirth" with the type of date.

```
mysql> alter table Persons
    -> add dateofbirth date;
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc Persons;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| P_Id        | int(11)     | NO   |     | NULL    |       |
| LastName    | varchar(255) | NO   |     | NULL    |       |
| FirstName   | varchar(255) | YES  |     | NULL    |       |
| Address     | varchar(255) | YES  |     | NULL    |       |
| City        | varchar(255) | YES  |     | SANDNES |       |
| dateofbirth | date        | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
6 rows in set (0.05 sec)
```

3. **Change Data Type**
Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.

```
mysql> alter table Persons
    -> modify dateofbirth int;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc Persons;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| P_Id        | int(11)     | NO   |     | NULL    |       |
| LastName    | varchar(255) | NO   |     | NULL    |       |
| FirstName   | varchar(255) | YES  |     | NULL    |       |
| Address     | varchar(255) | YES  |     | NULL    |       |
| City        | varchar(255) | YES  |     | SANDNES |       |
| dateofbirth | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
6 rows in set (0.05 sec)
```

4. **DROP COLUMN**
Next, we want to delete the column named "City" in the "Persons" table.

```
mysql> alter table Persons
    -> drop column city;
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc Persons;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| P_Id        | int(11)     | NO   |     | NULL    |       |
```

```
| LastName    | varchar(255) | NO   |     | NULL    |       |
| FirstName   | varchar(255) | YES  |     | NULL    |       |
| Address     | varchar(255) | YES  |     | NULL    |       |
| dateofbirth | int(11)      | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

A4. VIEW Statement

1. CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW Syntax:

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

2. SQL CREATE VIEW Examples

Query 14: View all the teacher and their courses (Note: Run Query 14 after TEACHER and COURSE tables are created in the next section)

```
mysql> create view te_cs
    ->    as select te.tno, te.tname, cs.cno
    ->    from  teacher te, course cs
    ->    where te.tno = cs.tno;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from te_cs;

TNO        TNAME                CNO
---------- -------------------- ----------
t002       MARSHALL             c001
t002       MARSHALL             c002
t001       JAMES                c003
t001       JAMES                c004
t003       BABER                c005
t003       BABER                c006
t002       MARSHALL             c007
t001       JAMES                c008
t003       BABER                c009
t002       MARSHALL             c010
```

B. INSERT, UPDATE and DELETE

B1. INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

1. INSERT INTO Syntax:

It is possible to write the INSERT INTO statement in two forms. The first form doesn't specify the column names where the data will be inserted, only their values:

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

Whenever you changing the data, you have to end with "**commit**". If something unexpected happens, however, the program would issue a rollback command, which instructs the server to undo all changes made since the transaction began. Then, you will loss the changes.

The second form specifies both the column names and the values to be inserted:

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

SQL INSERT INTO Example:

`Query 15:` Using the following command to create 4 tables:

```
mysql> drop table student;
Query OK, 0 rows affected (0.17 sec)

mysql> drop table teacher;
Query OK, 0 rows affected (0.10 sec)

mysql> drop table course;
Query OK, 0 rows affected (0.10 sec)

mysql> drop table sc;
Query OK, 0 rows affected (0.10 sec)

mysql> create table student(
    ->   sno varchar(10) primary key,
    ->   sname varchar(20),
    ->   sage int(2),
    ->   ssex varchar(5)
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> create table teacher(
    ->   tno varchar(10) primary key,
    ->   tname varchar(20)
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> create table course(
    ->   cno varchar(10),
    ->   cname varchar(20),
    ->   tno varchar(20),
    ->   constraint pk_course primary key (cno,tno)
```

```
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> create table sc(
    ->   sno varchar(10),
    ->   cno varchar(10),
    ->   score decimal(4,2),
    ->   constraint pk_sc primary key (sno,cno)
    -> );
Query OK, 0 rows affected (0.06 sec)
```

Query 16: Now we have to insert a new row into the "student" table. We use the following statement:

```
mysql> insert into student values ('s001','JACK',23,'M');
Query OK, 1 row affected (0.05 sec)

mysql> commit;
Query OK, 0 rows affected (0.04 sec)
```

The student table will now look like this:

```
mysql> select * from student;
+------+-------+------+------+
| sno  | sname | sage | ssex |
+------+-------+------+------+
| s001 | JACK  |   23 | M    |
+------+-------+------+------+
1 row in set (0.13 sec)
```

2. **Insert Data Only in Specified Columns**

It is also possible to only add data in specific columns.
The following SQL statement will add a new row, but only add data in the "sno", "sname" and the "sage" columns:

```
mysql> insert into student (sno, sname, sage)
    ->  values ('s002', 'DANA', 23);
Query OK, 1 row affected (0.16 sec)

mysql> commit;
Query OK, 0 rows affected (0.05 sec)

mysql> select * from student;
+------+-------+------+------+
| sno  | sname | sage | ssex |
+------+-------+------+------+
| s001 | JACK  |   23 | M    |
| s002 | DANA  |   23 | NULL |
+------+-------+------+------+
2 rows in set (0.06 sec)
```

B2. The UPDATE Statement

The UPDATE statement is used to update existing records in a table.

1. **UPDATE Syntax:**

```
UPDATE table_name
```

```
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

2. **UPDATE Example:**

Now we want to update the student "s002, BILL, 25, M", in the "student" table.
We use the following SQL statement:

```
mysql> update student
    ->   set sname = 'BILL', sage = 25, ssex = 'M'
    ->   where sno = 's002';
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.04 sec)
```

The "student" table will now look like this:

```
mysql> select * from student;
+------+-------+------+------+
| sno  | sname | sage | ssex |
+------+-------+------+------+
| s001 | JACK  |   23 | M    |
| s002 | BILL  |   25 | M    |
+------+-------+------+------+
```

3. UPDATE Warning

Be careful when updating records. If we had omitted the WHERE clause in the example above, like this:

```
mysql> update student
    ->   set sname = 'BILL', sage = 25, ssex = 'M';
Query OK, 1 row affected (0.04 sec)
Rows matched: 2  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.05 sec)
```

The "student" table would have looked like this:

```
mysql> select * from student;
+------+-------+------+------+
| sno  | sname | sage | ssex |
+------+-------+------+------+
| s001 | BILL  |   25 | M    |
| s002 | BILL  |   25 | M    |
+------+-------+------+------+
```

B3. DELETE Statement

The DELETE statement is used to delete records in a table.

1. SQL DELETE Syntax

```
DELETE FROM table_name
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

2. DELETE Example:
We want to delete the student "s001, JACK, 23, M" in the "student" table.
We use the following SQL statement:

```
mysql> delete from student
    ->  where sno = 's001' and sname = 'BILL'
    ->  and sage = 25 and ssex = 'M';
Query OK, 1 row affected (0.05 sec)

mysql> commit;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from student;
+------+-------+------+------+
| sno  | sname | sage | ssex |
+------+-------+------+------+
| s002 | BILL  |   25 | M    |
+------+-------+------+------+
```

or (Because sno is the primary key)

```
mysql> DELETE FROM student WHERE sno = 's001';
Query OK, 0 rows affected (0.03 sec)
```

3. DELETE ALL ROWS
It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name

or

DELETE * FROM table_name
```

C. Advanced DML

C0. INSERT test data into Tables

`Query 17:` Before we practice **SELECT**, please insert data into the 4 tables (student, teacher, course, sc) by the following command. You can copy and paste from given insert statements below, which are between cut and paste lines.

```
mysql> source Q17.sql;

   (Note: Q17.sql drop all tables and create again)

OR
```

Initialization of the **student** table:

```
================ Cut & Paste ======================
insert into student values ('s003','BILL',25,'M');
insert into student values ('s004','STEVE',20,'F');
```

```
insert into student values ('s005','BAKR',20,'F');
insert into student values ('s006','TOM',21,'M');
insert into student values ('s007','JERRY',21,'M');
insert into student values ('s008','MACY',21,'F');
insert into student values ('s009','MICK',23,'F');
insert into student values ('s010','COOKER',22,'F');
commit;
================ Cut & Paste ========================
```

Initialization of the Teacher table:

```
================ Cut & Paste ========================
insert into teacher values ('t001', 'JAMES');
insert into teacher values ('t002', 'MARSHALL');
insert into teacher values ('t003', 'BABER');
commit;
================ Cut & Paste ========================
```

Initialization of the Course table:

```
================ Cut & Paste ========================
insert into course values ('c001','J2SE','t002');
insert into course values ('c002','Java Web','t002');
insert into course values ('c003','SSH','t001');
insert into course values ('c004','Oracle','t001');
insert into course values ('c005','SQL SERVER 2005','t003');
insert into course values ('c006','C#','t003');
insert into course values ('c007','JavaScript','t002');
insert into course values ('c008','DIV+CSS','t001');
insert into course values ('c009','PHP','t003');
insert into course values ('c010','EJB3.0','t002');
commit;
================ Cut & Paste ========================
```

Initialization of the SC Table:

```
================ Cut & Paste ========================
insert into sc values ('s001','c001',78.9);
insert into sc values ('s002','c001',80.9);
insert into sc values ('s003','c001',81.9);
insert into sc values ('s004','c001',60.9);
insert into sc values ('s001','c002',82.9);
insert into sc values ('s002','c002',72.9);
insert into sc values ('s003','c002',81.9);
insert into sc values ('s001','c003','59');
commit;
================ Cut & Paste ========================
```

C1. SELECT-FROM-WHERE

The SELECT statement allows you to retrieve records from one or more tables in your database. The basic syntax for the SELECT statement is:

SELECT <attribute list>
FROM <table list>
WHERE <condition>;

where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

In SQL, the basic logical comparison operators for comparing attribute values with one another and with literal constants are =, <, <=, >, >=, and <>. These correspond to the relational algebra operators =, <, ≤, >, ≥, and ≠, respectively, and to the C/C++ programming language operators =, <, <=, >, >=, and !=. The main syntactic difference is the *not equal* operator. SQL has additional comparison operators.

Query 18. Retrieve all the students' name.

```
mysql> select sname from student;
+--------+
| sname  |
+--------+
| BILL   |
| STEVE  |
| BAKR   |
| TOM    |
| JERRY  |
| MACY   |
| MICK   |
| COOKER |
+--------+
8 rows in set (0.06 sec)
```

Query 19. Retrieve all the students

```
mysql> select *
    ->  from student;
+------+--------+------+------+
| sno  | sname  | sage | ssex |
+------+--------+------+------+
| s003 | BILL   |   25 | M    |
| s004 | STEVE  |   20 | F    |
| s005 | BAKR   |   20 | F    |
| s006 | TOM    |   21 | M    |
| s007 | JERRY  |   21 | M    |
| s008 | MACY   |   21 | F    |
| s009 | MICK   |   23 | F    |
| s010 | COOKER |   22 | F    |
+------+--------+------+------+
8 rows in set (0.05 sec)
```

Query 20.Retrieve the student whose name is 'STEVE'.

```
mysql> select *
    ->   from student
    ->   where sname = 'STEVE';
+------+-------+------+------+
| sno  | sname | sage | ssex |
+------+-------+------+------+
| s004 | STEVE |   20 | F    |
```

```
+------+-------+------+------+
1 row in set (0.06 sec)
```

WHERE Clause:
The WHERE clause is optional. When specified, it always follows the FROM clause. The WHERE clause filters rows from the FROM clause tables. Omitting the WHERE clause specifies that all rows are used.

Query 21. Retrieve any student's name, whose score is greater than 70;

```
mysql> SELECT st.sname, c.cname, sc.score FROM student st,sc,course c
    ->   WHERE sc.sno=st.sno AND sc.cno=c.cno AND sc.score>70;
+-------+----------+-------+
| sname | cname    | score |
+-------+----------+-------+
| BILL  | J2SE     | 81.90 |
| BILL  | Java Web | 81.90 |
+-------+----------+-------+
2 rows in set (0.04 sec)
```

C2. NATURAL JOIN

The most common use of JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is = , is called an **EQUIJOIN**. Notice that in the result of an EQUIJOIN we always have one or more pairs of attributes that have identical values in every tuple. Because one each pair of attributes with identical values is superfluous, a new operation called **NATURAL JOIN** – denoted by * - was created to get rid of the second attribute in an EQUIJOIN condition.
The standard definition of NATURAL JOIN requires that the two join attributes have the same name in both relations. If this is not the case, a renaming operation is applied first.
Suppose we want to combine each STUDENT tuple with the SC tuple that has the sno. In the following example:

Query 22. Retrieve students' id number and name and the course number they selected and the corresponding score;

```
mysql> select st.sno, st.sname, sc.cno, sc.score
    ->   from sc, student st
    ->   where sc.sno = st.sno;
+------+-------+------+-------+
| sno  | sname | cno  | score |
+------+-------+------+-------+
| s003 | BILL  | c001 | 81.90 |
| s003 | BILL  | c002 | 81.90 |
| s004 | STEVE | c001 | 60.90 |
+------+-------+------+-------+
3 rows in set (0.05 sec)
```

Query 23. Retrieve the student's id number that get same grade in different courses, and display their id number, course id number, and the grade that are the same;

```
mysql> SELECT a.* FROM sc a ,sc b WHERE a.score=b.score
    -> AND a.cno<>b.cno;
+------+------+-------+
| sno  | cno  | score |
+------+------+-------+
| s003 | c002 | 81.90 |
| s003 | c001 | 81.90 |
+------+------+-------+
2 rows in set (0.06 sec)
```

C3. Tables as SET in SQL

SQL usually treats a table not as a set but rather as a **multiset**; *duplicate tuples can appear more than once* in a table, and in the result of a query.
An SQL table with a key is restricted to being a set, since the key value must be distinct in each tuple. If we want to eliminate duplicate tuples from the result of an SQL query, we use the keyword **DISTINCT** in the SELECT clause, meaning that only distinct tuples should remain in the result. In general, a query with SELECT DISTINCT eliminates duplicates, whereas a query with SELECT ALL does not.

Query 24. Retrieve the total number of students who SELECT course;

```
mysql> SELECT count(DISTINCT sno)
    -> from sc;
+----------------------+
| count(DISTINCT sno)  |
+----------------------+
|                    4 |
+----------------------+
1 row in set (0.05 sec)
```

SET OPERATIONS:
SQL has directly incorporated some set operations. There is a union operation (UNION), and in some versions of SQL there are set difference (MINUS) and intersection (INTERSECT) operations. The resulting relations of these set operations are sets of tuples; duplicate tuples are eliminated from the result. The set operations apply only to union compatible relations; the two relations must have the same attributes and the attributes must appear in the same order.

C4. Nested Queries, Tuples, and Set/Multiset Comparisons

A completes SELECT query, called a *nested query*, can be specified within the WHERE-clause of another query, called the *outer query*.

Query 26. Retrieve the students' name who have never select any course from a teacher "MARSHALL";

```
mysql> SELECT st.sname FROM student st
    ->     WHERE st.sno NOT IN
    ->     (
    ->     SELECT DISTINCT sc.sno
    ->       FROM sc,course c,teacher t
```

```
    ->       WHERE sc.cno=c.cno AND c.tno=t.tno AND t.tname='MARSHALL'
    ->     );
+--------+
| sname  |
+--------+
| BAKR   |
| TOM    |
| JERRY  |
| MACY   |
| MICK   |
| COOKER |
+--------+
6 rows in set (0.06 sec)
```

Query 27. Retrieve the students' id number, Name who haven't studied Teacher "MARSHALL"'s course.

```
mysql> SELECT a.sno,a.sname FROM student a
    ->     WHERE a.sno
    ->     NOT IN
    ->     ( SELECT DISTINCT  s.sno
    ->       FROM sc s,
    ->       (  SELECT c.*
    ->         FROM course c ,
    ->         (  SELECT tno
    ->           FROM teacher t
    ->           WHERE tname='MARSHALL' ) t
    ->        WHERE c.tno=t.tno ) b
    ->       WHERE s.cno = b.cno ) ;
+------+--------+
| sno  | sname  |
+------+--------+
| s005 | BAKR   |
| s006 | TOM    |
| s007 | JERRY  |
| s008 | MACY   |
| s009 | MICK   |
| s010 | COOKER |
+------+--------+
6 rows in set (0.06 sec)

mysql> SELECT *
    ->     FROM student st
    ->     WHERE st.sno
    ->     NOT IN
    ->     ( SELECT DISTINCT sno
    ->     FROM sc s
    ->     JOIN course c ON s.cno=c.cno
    ->     JOIN teacher t ON c.tno=t.tno
    ->     WHERE tname='MARSHALL' );
+------+--------+------+------+
| sno  | sname  | sage | ssex |
+------+--------+------+------+
| s005 | BAKR   |   20 | F    |
| s006 | TOM    |   21 | M    |
| s007 | JERRY  |   21 | M    |
```

```
| s008 | MACY   |   21 | F    |
| s009 | MICK   |   23 | F    |
| s010 | COOKER |   22 | F    |
+------+--------+------+------+
```

C5. EXISTS and UNIQUE

The EXISTS function in SQL is used to check whether the result of a correlated nested query is *empty* (contains no tuples) or not. The result of EXISTS is a Boolean value **TRUE** if the nested query result contains at least one tuple, or **FALSE** if the nested query result contains no tuples.

Query 28. Retrieve all the students' id number whose 'c001' score is higher than their 'c002';

```
mysql> SELECT a.* FROM
    ->     (SELECT * FROM sc a WHERE a.cno='c001') a,
    ->     (SELECT * FROM sc b WHERE b.cno='c002') b
    ->     WHERE a.sno=b.sno AND a.score > b.score;
+------+------+-------+
| sno  | cno  | score |
+------+------+-------+
| s002 | c001 | 80.90 |
+------+------+-------+
1 row in set (0.05 sec)

mysql> SELECT * FROM sc a
    ->     WHERE a.cno='c001'
    ->     AND EXISTS(SELECT * FROM sc b WHERE b.cno='c002' AND
a.score>b.score
    ->     AND a.sno = b.sno);
+------+------+-------+
| sno  | cno  | score |
+------+------+-------+
| s002 | c001 | 80.90 |
+------+------+-------+
1 row in set (0.05 sec)
```

Query 29. Insert some records into table sc: the students' id number and average score who have studied 'c002'.

```
mysql> INSERT INTO sc (sno,cno,score)
    ->     SELECT DISTINCT st.sno,sc.cno,(SELECT AVG (score)FROM sc
WHERE cno='c002')
    ->     FROM student st,sc
    ->     WHERE NOT EXISTS
    ->     (SELECT * FROM sc WHERE cno='c002' AND sc.sno=st.sno)
    ->     AND sc.cno='c002';
Query OK, 7 rows affected, 7 warnings (0.06 sec)
Records: 7  Duplicates: 0  Warnings: 7
```

AVG(score) is a function to obtain the average number of the list of score.

C6. Aggregate Functions

Aggregate functions are used to summarize information from multiple tuples into a single-tuple summary. Grouping is used to create subgroups of tuples before summarization. Grouping and aggregation are required in many database applications, and we will introduce their use in SQL through examples. A number of built-in aggregate functions exist: **COUNT, SUM, MAX, MIN,** and **AVG**. The COUNT function returns the number of tuples or values as specified in a query. The functions SUM, MAX, MIN, and AVG can be applied to a set or multiset of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values. These functions can be used in the SELECT clause or in a **HAVING** clause (which we introduce later). The functions MAX and MIN can also be used with attributes that have nonnumeric domains if the domain values have a *total ordering* among one another.

Grouping: The GROUP BY and HAVING Clauses

In many cases we want to apply the aggregate functions *to subgroups of tuples in a relation,* where the subgroups are based on some attribute values. For example, we may want to find the average salary of employees *in each department* or the number of employees who work *on each project.* In these cases we need to **partition** the relation into non-overlapping subsets(or **groups**) of tuples. Each group (partition) will consist of the tuples that have the same value of some attribute(s), called the **grouping attribute(s)**. We can then apply the function to each such group independently to produce summary information about each group. SQL has a **GROUP BY** clause for this purpose. The GROUP BY clause specifies the grouping attributes, which should *also appear in the SELECT clause*, so that the value resulting from applying each aggregate function to a group of tuples appears along with the value of the grouping attribute(s).

Query 30. Retrieve all the students' id number and average score whose average score is greater than 60;

```
mysql> SELECT sno,avg(score)
    ->     FROM sc
    ->     GROUP BY sno
    ->     HAVING avg(score)>60;
+------+------------+
| sno  | avg(score) |
+------+------------+
| s001 |  73.600000 |
| s002 |  76.900000 |
| s003 |  81.900000 |
| s004 |  70.065000 |
| s005 |  79.230000 |
| s006 |  79.230000 |
| s007 |  79.230000 |
| s008 |  79.230000 |
| s009 |  79.230000 |
| s010 |  79.230000 |
+------+------------+
10 rows in set (0.04 sec)
```

Query 31. Retrieve all the students' id number, name, the number of courses selected, and total grade;

```
mysql> SELECT a.*,s.sname
    ->     FROM ( SELECT sno, SUM(score), COUNT(cno)
    ->     FROM sc
    ->     GROUP BY sno) a ,student s
```

```
    ->     WHERE a.sno=s.sno ;
+------+-----------+-----------+--------+
| sno  | SUM(score) | COUNT(cno) | sname  |
+------+-----------+-----------+--------+
| s003 |     163.80 |          2 | BILL   |
| s004 |     140.13 |          2 | STEVE  |
| s005 |      79.23 |          1 | BAKR   |
| s006 |      79.23 |          1 | TOM    |
| s007 |      79.23 |          1 | JERRY  |
| s008 |      79.23 |          1 | MACY   |
| s009 |      79.23 |          1 | MICK   |
| s010 |      79.23 |          1 | COOKER |
+------+-----------+-----------+--------+
8 rows in set (0.13 sec)
```

C7. SUBSTRING Comparison

The **LIKE** comparison operator is used to compare partial strings. Two reserved characters are used: '%' replaces an arbitrary number of characters, and '_' replaces a single arbitrary character.

Query 32. Retrieve the students whose name begin with ‘J’;

```
mysql> SELECT * FROM student WHERE sname LIKE 'J%';
+------+-------+------+------+
| sno  | sname | sage | ssex |
+------+-------+------+------+
| s007 | JERRY |   21 | M    |
+------+-------+------+------+
1 row in set (0.06 sec)
```

Query 33. Retrieve the number of teachers whose Family name is begin with ‘B’;

```
mysql> SELECT count(*) FROM teacher
    ->     WHERE tname LIKE 'B%';
+----------+
| count(*) |
+----------+
|        1 |
+----------+
```

C8. Arithmetic Operations

The standard arithmetic operators, such as '+', '-'. '*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result.

Query 34. List the students who were born in 1990;

```
mysql> SELECT sno,sname,sage,ssex
    ->     FROM student t
    ->     WHERE DATE_FORMAT(SYSDATE(), '%Y')-sage >= 1992;
```

```
+------+--------+------+------+
| sno  | sname  | sage | ssex |
+------+--------+------+------+
| s003 | BILL   |   25 | M    |
| s004 | STEVE  |   20 | F    |
| s005 | BAKR   |   20 | F    |
| s006 | TOM    |   21 | M    |
| s007 | JERRY  |   21 | M    |
| s008 | MACY   |   21 | F    |
| s009 | MICK   |   23 | F    |
| s010 | COOKER |   22 | F    |
+------+--------+------+------+
8 rows in set (0.05 sec)
```

C9. ORDER BY

The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s). The default order is in ascending order of values.
We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default.

Query 35. Sorting table sc by ascending order of average score and the descending order of passed-rate;

```
mysql> SELECT cno,AVG(score),SUM(case when score>=60 then 1 else 0
end)/count(*)
    ->     as passedRate
    ->     FROM sc GROUP BY cno
    ->     ORDER BY avg(score) , passedRate desc;
+------+------------+------------+
| cno  | AVG(score) | passedRate |
+------+------------+------------+
| c003 |  59.000000 |     0.0000 |
| c001 |  75.650000 |     1.0000 |
| c002 |  79.231000 |     1.0000 |
+------+------------+------------+
3 rows in set (0.05 sec)
```

Query 36. Retrieve the average score of different courses from different teachers by descending order;

```
mysql> SELECT
max(t.tno),max(t.tname),max(c.cno),max(c.cname),c.cno,avg(score)
    -> FROM sc , course c,teacher t
    ->     WHERE sc.cno=c.cno AND c.tno=t.tno
    ->     GROUP BY c.cno
    ->     ORDER BY avg(score) desc;
+------------+-------------+------------+-------------+------+------------+
| max(t.tno) | max(t.tname) | max(c.cno) | max(c.cname) | cno  | avg(score) |
+------------+-------------+------------+-------------+------+------------+
| t002       | MARSHALL    | c002       | Java Web    | c002 |  79.231000 |
| t002       | MARSHALL    | c001       | J2SE        | c001 |  75.650000 |
| t001       | JAMES       | c003       | SSH         | c003 |  59.000000 |
```

```
+-----------+-------------+-----------+-------------+------+------------+
3 rows in set (0.04 sec)
```

C10. More SQL practices

Query 37. Update the grade of the course taught by "MARSHALL" IN SC table and set it as the average grade in that course.

```
mysql> UPDATE sc c SET score=
    -> (
    -> SELECT AVG(c.score)
    -> FROM course a,teacher b
    -> WHERE a.tno=b.tno
    -> AND b.tname='MARSHALL'
    -> AND a.cno=c.cno
    -> GROUP BY c.cno
    -> )
    -> WHERE cno IN
    -> (
    ->  SELECT cno FROM course a,teacher b
    ->  WHERE a.tno=b.tno
    ->  AND b.tname='MARSHALL'
    -> );
Query OK, 0 rows affected (0.05 sec)
Rows matched: 14  Changed: 0  Warnings: 0
```

(**Note**: For your practice, any changed data will be rolled back. Otherwise, it should be committed.)

```
mysql> rollback;
```

Query 38. Delete all the records from sc table where the teacher is "MARSHALL";

```
mysql> DELETE FROM sc
    -> WHERE sc.cno IN
    -> (
    ->  SELECT cno FROM course c
    ->  left JOIN teacher t ON c.tno=t.tno
    ->  WHERE t.tname='MARSHALL'
    -> );
Query OK, 14 rows affected (0.05 sec)
```

Query 39. Insert some records into table sc: the students' id number and average score who have studied 'c002'.

```
mysql> INSERT INTO sc (sno,cno,score)
    -> SELECT DISTINCT st.sno,sc.cno,
    -> (SELECT AVG(score)FROM sc WHERE cno='c002')
    -> FROM student st,sc
    -> WHERE NOT EXISTS
    -> (SELECT * FROM sc WHERE cno='c002' AND sc.sno=st.sno)
    -> AND sc.cno='c002';
```

Query 40. Retrieve the max and min score and display them with the format: Course Id, Max Score, Min Score;

```
mysql> SELECT cno ,MAX(score),MIN(score) FROM sc GROUP BY cno;
+------+------------+------------+
| cno  | MAX(score) | MIN(score) |
+------+------------+------------+
| c003 |      59.00 |      59.00 |
+------+------------+------------+
1 row in set (0.09 sec)
```

Query 41. Retrieve the students' id number AND name at least one of whose course are the same as a certain student's course whose id is 's001';

```
mysql> SELECT st.* FROM student st,
    ->     ( SELECT DISTINCT a.sno FROM
    ->       ( SELECT * FROM sc ) a,
    ->       ( SELECT * FROM sc WHERE sc.sno='s001' ) b
    ->       WHERE a.cno=b.cno ) h
    ->     WHERE st.sno=h.sno AND st.sno<>'s001';
```

Query 42. List each course's id number, name and the number of students distributed in the scales: [100-85], [85-70], [70-60], [<60];

```
mysql> SELECT sc.cno,c.cname,
    ->   sum(case when score between 85 AND 100 then 1 else 0 end) AS "[100-85]",
    ->   sum(case when score between 70 AND 85 then 1 else 0 end) AS "[85-70]",
    ->   sum(case when score between 60 AND 70 then 1 else 0 end) AS "[70-60]",
    ->   sum(case when score <60 then 1 else 0 end) AS "[<60]"
    ->   FROM sc, course c
    ->   WHERE sc.cno=c.cno
    ->   GROUP BY sc.cno ,c.cname;
+------+-------+----------+---------+---------+-------+
| cno  | cname | [100-85] | [85-70] | [70-60] | [<60] |
+------+-------+----------+---------+---------+-------+
| c003 | SSH   |        0 |       0 |       0 |     1 |
+------+-------+----------+---------+---------+-------+
1 row in set (0.05 sec)
```

Query 43. Retrieve the number of students for each course;

```
mysql> SELECT cno,count(sno) FROM sc GROUP BY cno;
+------+------------+
| cno  | count(sno) |
+------+------------+
| c003 |          1 |
+------+------------+
```

Query 44. Retrieve the number of the Male AND Female students.

```
mysql> SELECT ssex,count(*) FROM student GROUP BY ssex;
+------+----------+
| ssex | count(*) |
+------+----------+
| F    |        5 |
| M    |        3 |
```

```
+------+----------+
2 rows in set (0.03 sec)
```

Query 45. Count the number of duplicated name;

```
mysql> SELECT sname,count(*)FROM student
    ->   GROUP BY sname
    ->   HAVING count(*)>1;
```

Query 46. Retrieve all the courses that have students' grade less than 60 by descending order of the course number;

```
mysql> SELECT sc.sno,c.cname,sc.score FROM sc,course c
    -> WHERE sc.cno=c.cno AND sc.score<60 ORDER BY sc.cno desc;
+------+-------+-------+
| sno  | cname | score |
+------+-------+-------+
| s001 | SSH   | 59.00 |
+------+-------+-------+
```

Query 47. Retrieve the student's name who has the highest score in teacher "MARSHALL"'s course and display the highest score;

```
mysql> SELECT st.sname,score FROM student st,sc ,course c,teacher t
    ->   WHERE st.sno=sc.sno AND sc.cno=c.cno AND c.tno=t.tno
    ->   AND t.tname='MARSHALL' AND sc.score=
    ->   (SELECT max(score)FROM sc WHERE sc.cno=c.cno);
```

Query 48. Retrieve all the courses and the number of students who select that course;

```
mysql> SELECT cno,count(sno) FROM sc GROUP BY cno;
+------+------------+
| cno  | count(sno) |
+------+------------+
| c003 |          1 |
+------+------------+
```

Query 49. Retrieve the students' id number who selected at least two courses;

```
mysql> SELECT sno FROM sc GROUP BY sno HAVING count(cno)>1;
Empty set (0.05 sec)
```

OR

```
mysql> SELECT sno FROM sc GROUP BY sno HAVING count(sno)>1;
Empty set (0.05 sec)
```

Query 50. Retrieve the courses' id and name that all the students have selected;

```
mysql> SELECT DISTINCT(c.cno),c.cname FROM course c ,sc
    ->   WHERE sc.cno=c.cno;
+------+-------+
| cno  | cname |
+------+-------+
| c003 | SSH   |
+------+-------+
1 row in set (0.05 sec)
```

OR

```
mysql> SELECT cno,cname FROM course c
    ->  WHERE c.cno IN
    ->  (SELECT cno FROM sc GROUP BY cno);
+------+-------+
| cno  | cname |
+------+-------+
| c003 | SSH   |
+------+-------+
1 row in set (0.05 sec)
```

Query 51. Retrieve the students' name who have never select any course from teacher "MARSHALL";

```
mysql> SELECT st.sname FROM student st
    ->   WHERE st.sno not IN
    ->   (SELECT DISTINCT sc.sno FROM sc,course c,teacher t
    ->   WHERE sc.cno=c.cno AND c.tno=t.tno AND t.tname='MARSHALL');
+--------+
| sname  |
+--------+
| BILL   |
| STEVE  |
| BAKR   |
| TOM    |
| JERRY  |
| MACY   |
| MICK   |
| COOKER |
+--------+
8 rows in set (0.05 sec)
```

Query 52. Retrieve the students' id and average score who have failed at least two courses;

```
mysql> SELECT sno,avg(score)FROM sc
    ->   WHERE sno IN
    ->   (SELECT sno FROM sc WHERE sc.score < 60
    ->   GROUP BY sno HAVING count(sno)>1 )
    ->   GROUP BY sno;
```

Query 53. Delete the grade in the students' 'c001'course who select both 'c001' and 'c002';

```
mysql> DELETE FROM sc WHERE sno='s002' AND cno='c001';
```

Query 54. Retrieve the average grade of each course by its ascending order and the course number's descending order.

```
mysql> SELECT cno,AVG(score)
    -> FROM sc
    -> GROUP BY cno
    -> ORDER BY AVG(score) asc, cno desc;
+------+------------+
| cno  | AVG(score) |
+------+------------+
| c003 |  59.000000 |
+------+------------+
1 row in set (0.07 sec)
```

Query 55. Count the number of students of each course (only the number more than 10). Display the course id, number of student by descending order of number of students and ascending ORDER BY course id.

```
mysql> SELECT cno,count(sno)
    ->   FROM sc GROUP BY cno
    ->   HAVING count(sno)>10
    ->   ORDER BY count(sno) desc,cno asc;
```

Query 56. Retrieve the students' id number whose grade of course 'c004' is less than 60, by descending order;

```
mysql> SELECT sno FROM sc
    -> WHERE cno='c004' AND score<90
    -> ORDER BY score desc;
```

Query 57. Retrieve the students' id number and name who have studied the courses 'c001' and 'c002';

```
mysql> SELECT st.* FROM sc a
    ->   JOIN sc b ON a.sno=b.sno
    ->   JOIN student st
    ->   ON st.sno=a.sno
    ->   WHERE a.cno='c001' AND b.cno='c002' AND st.sno=a.sno;
```

Query 58. Retrieve all the students' id number and name who have studied all the courses from Query Teacher "MARSHALL".

```
mysql> SELECT st.* FROM student st JOIN sc s ON st.sno=s.sno
    ->   JOIN course c ON s.cno=c.cno
    ->   JOIN teacher t ON c.tno=t.tno
```

```
    ->   WHERE t.tname='MARSHALL';
```

Query 59. Retrieve all the students' id number and name whose grade of course 'c002' is less than their grade of course 'c001';

```
mysql> SELECT * FROM student st
    ->   JOIN sc a ON st.sno=a.sno
    ->   JOIN sc b ON st.sno=b.sno
    ->   WHERE a.cno='c002' AND b.cno='c001' AND a.score < b.score;
```

Query 60. Retrieve the students' id number and name whose some grade is less than '60';

```
mysql> SELECT st.*,s.score FROM student st
    ->   JOIN sc s ON st.sno=s.sno
    ->   JOIN course c ON s.cno=c.cno
    ->   WHERE s.score <60;
```

Query 61. Retrieve the other (not s001) students' id number and name who have learned all the courses that student 's001' has learned before;

```
mysql> SELECT * FROM sc
    ->   LEFT JOIN student st
    ->   ON st.sno=sc.sno
    ->   WHERE sc.sno<>'s001'
    ->   AND sc.cno IN
    ->   (SELECT cno FROM sc
    ->   WHERE sno='s001');
```

Query 62. Retrieve the students' id number and name who select only one course;

```
mysql> SELECT sc.sno,st.sname,count(cno) FROM student st
    ->   LEFT JOIN sc
    ->   ON sc.sno=st.sno
    ->   GROUP BY st.sname,sc.sno HAVING count(cno)=1;
```

Query 63. Retrieve the students' id number, name, and average grade whose average is greater than 85;

```
mysql> SELECT st.sno,st.sname,avg(score) FROM student st
    ->   LEFT JOIN sc
    ->   ON sc.sno=st.sno
    ->   GROUP BY st.sno,st.sname HAVING AVG(score) > 85;
```

Exercise

Exercise 1: Creating View

For this exercise, create a view called V_CIS_MAJORS based upon the following SQL SELECT statement:

```
SELECT * FROM students WHERE major = 'CIS';
```

Query the view and show the output.

Create another view called V_COURSES_TAKEN based upon the following SQL SELECT statement:

```
SELECT name, major, coursenumber, coursename,
       semester, year, grade
FROM   students, courses
WHERE  students.studentid = courses.studentid;
```

Query the V_COURSES_TAKEN view and show the output.

Exercise 2: More JOIN Queries with COMPANY database

% Use COMPANY database. You can re-create COMPANY database using cr_company.sql

1. List all of the employees working in Houston
2. List each employee name and the location they work in. List them in order of location and name
3. What is the highest paid salary in Houston?
4. In which states do our employees live?
5. List the Department name and the total salaries for each department

Lab 7: Stored Procedure (PL/SQL)

Use lab7 folder

- Start mysql> at **lab7** directory
- Save the file and extract it under Lab folder
- Remember to start by creating a silicon database to use.

A. Stored Procedure

A0. Basic of Stored Procedure

Why? Stored procedures are fast. MySQL server takes some advantage of caching, just as prepared statements do. The main speed gain comes from reduction of network traffic. If you have a repetitive task that requires checking, looping, multiple statements, and no user interaction, do it with a single call to a procedure that's stored on the server.

Before creating a procedure:

- **Check MySQL** version:

  ```
  mysql>SELECT VERSION();
  ```

- Check the **privileges** of the current user :

  ```
  mysql> SHOW PRIVILEGES;
  ```

  Need CREATE ROUTINE privilege, and CREATE FUNCTION might require the SUPER privilege
- Create 'hr' database:

  ```
  mysql> source hr-schema-mysql.sql;
  ```

- Pick a **Delimiter**: The delimiter is the character or string of characters which is used to complete an SQL statement. By default we use semicolon (;) as a delimiter. But this causes problem in stored procedure because a procedure can have many statements, and everyone must end with a semicolon. So for your delimiter, pick a string which is rarely occur within statement or within procedure. Here we have used double dollar sign i.e. $$.

  ```
  mysql> DELIMITER $$ ;
  ```

- DELIMITER is set to it's default.

  ```
  mysql> DELIMITER ; $$
  ```

A1. Creating simple procedure

Query 1: Simple procedure

```
mysql> DELIMITER $$ ;
mysql> CREATE PROCEDURE job_data()
    -> SELECT * FROM jobs; $$
Query OK, 0 rows affected (0.05 sec)
mysql> DELIMITER ; $$
```

A2. Executing procedure

Query 2.

```
mysql> call job_data;
+------------+----------------------------------+------------+------------+
| job_id     | job_title                        | min_salary | max_salary |
+------------+----------------------------------+------------+------------+
| AC_ACCOUNT | Public Accountant                |       4200 |       9000 |
| AC_MGR     | Accounting Manager               |       8200 |      16000 |
| AD_ASST    | Administration Assistant         |       3000 |       6000 |
| AD_PRES    | President                        |      20000 |      40000 |
| AD_VP      | Administration Vice President    |      15000 |      30000 |
| FI_ACCOUNT | Accountant                       |       4200 |       9000 |
| FI_MGR     | Finance Manager                  |       8200 |      16000 |
| HR_REP     | Human Resources Representative   |       4000 |       9000 |
| IT_PROG    | Programmer                       |       4000 |      10000 |
| MK_MAN     | Marketing Manager                |       9000 |      15000 |
| MK_REP     | Marketing Representative         |       4000 |       9000 |
| PR_REP     | Public Relations Representative  |       4500 |      10500 |
| PU_CLERK   | Purchasing Clerk                 |       2500 |       5500 |
| PU_MAN     | Purchasing Manager               |       8000 |      15000 |
| SA_MAN     | Sales Manager                    |      10000 |      20000 |
| SA_REP     | Sales Representative             |       6000 |      12000 |
| SH_CLERK   | Shipping Clerk                   |       2500 |       5500 |
| ST_CLERK   | Stock Clerk                      |       2000 |       5000 |
| ST_MAN     | Stock Manager                    |       5500 |       8500 |
+------------+----------------------------------+------------+------------+
```

A3. Show create procedure

This statement is a MySQL extension. It returns the exact string that can be used to re-create the named stored procedure. Both statement require that you be the owner of the routine.

Query 3.

```
   mysql> SHOW CREATE PROCEDURE job_data;
+-----------+-------------------------------------------------------------------
  ---------------------------------------------------------------------------
  +-------------------------------------------------------------------------+----
  ------------------+----------------------+--------------------+
| Procedure | sql_mode
  | Create Procedure                                                          |
  character_set_client | collation_connection | Database Collation |
+-----------+-------------------------------------------------------------------
  ---------------------------------------------------------------------------
  +-------------------------------------------------------------------------+----
  ------------------+----------------------+--------------------+
| job_data  |
  ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_F
```

```
  OR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION | CREATE
  DEFINER=`root`@`%` PROCEDURE `job_data`()
SELECT * FROM jobs | cp850                | cp850_general_ci     |
  utf8_general_ci    |
+----------+-------------------------------------------------------------------
  ---------------------------------------------------------------------------
  +-------------------------------------------------------------------------+----
  -----------------+---------------------+--------------------+
1 row in set (0.10 sec)
```

A4. MySQL : Compound-Statement

A compound statement is a block that can contain other blocks; declarations for variables, condition handlers, and cursors; and flow control constructs such as loops and conditional tests. As of version 5.6 MySQL have following compound statements :

- BEGIN ... END Compound-Statement
- Statement Label
- DECLARE
- Variables in Stored Programs
- Flow Control Statements
- Cursors
- Condition Handling

Query 4.

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE my_procedure_Local_Variables()
    -> BEGIN   /* declare local variables */
    -> DECLARE a INT DEFAULT 10;
    -> DECLARE b, c INT;    /* using the local variables */
    -> SET a = a + 100;
    -> SET b = 2;
    -> SET c = a + b;
    -> BEGIN      /* local variable in nested block */
    -> DECLARE c INT;
    -> SET c = 5;
    -> /* local variable c takes precedence over the one of the
   /*> same name declared in the enclosing block. */
    -> SELECT a, b, c;
    -> END;
    -> SELECT a, b, c;
    -> END$$
Query OK, 0 rows affected (0.05 sec)

mysql> DELIMITER ;
mysql> call my_procedure_Local_Variables;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|  110 |    2 |    5 |
+------+------+------+
1 row in set (0.04 sec)

+------+------+------+
| a    | b    | c    |
```

```
+------+------+------+
|  110 |    2 |  112 |
+------+------+------+
1 row in set (0.04 sec)

Query OK, 0 rows affected (0.09 sec)
```

A5. User variables

User variables are referenced with an ampersand (@) prefixed to the user variable name (for example, @x and @y). The following example shows the use of user variables within the stored procedure :

Query 5.

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE my_procedure_User_Variables()
    -> BEGIN
    -> SET @x = 15;
    -> SET @y = 10;
    -> SELECT @x, @y, @x-@y;
    -> END$$
Query OK, 0 rows affected (0.08 sec)

mysql> DELIMITER ;
mysql> CALL my_procedure_User_Variables();
+------+------+-------+
| @x   | @y   | @x-@y |
+------+------+-------+
|   15 |   10 |     5 |
+------+------+-------+
1 row in set (0.06 sec)

Query OK, 0 rows affected (0.06 sec)
```

A6. Procedure Parameters

We can divide the CREATE PROCEDURE statement in the following ways :

```
0. CREATE PROCEDURE sp_name () ...
A. CREATE PROCEDURE sp_name ([IN] param_name type)...
B. CREATE PROCEDURE sp_name ([OUT] param_name type)...
C. CREATE PROCEDURE sp_name ([INOUT] param_name type)...
```

Query 6A. Parameter IN example

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE my_proc_IN (IN var1 INT)
    -> BEGIN
    -> SELECT * FROM jobs LIMIT var1;
    -> END$$
Query OK, 0 rows affected (0.05 sec)
mysql> DELIMITER ;
```

% To execute the first 2 rows from the 'jobs' table

```
mysql> CALL my_proc_in(2);
```

```
+------------+--------------------+------------+------------+
| job_id     | job_title          | min_salary | max_salary |
+------------+--------------------+------------+------------+
| AC_ACCOUNT | Public Accountant  |       4200 |       9000 |
| AC_MGR     | Accounting Manager |       8200 |      16000 |
+------------+--------------------+------------+------------+
2 rows in set (0.05 sec)

Query OK, 0 rows affected (0.06 sec)
```

% To execute the first 5 rows from the 'jobs' table

```
mysql> CALL my_proc_in(5);
+------------+---------------------------------+------------+------------+
| job_id     | job_title                       | min_salary | max_salary |
+------------+---------------------------------+------------+------------+
| AC_ACCOUNT | Public Accountant               |       4200 |       9000 |
| AC_MGR     | Accounting Manager              |       8200 |      16000 |
| AD_ASST    | Administration Assistant        |       3000 |       6000 |
| AD_PRES    | President                       |      20000 |      40000 |
| AD_VP      | Administration Vice President   |      15000 |      30000 |
+------------+---------------------------------+------------+------------+
5 rows in set (0.04 sec)

Query OK, 0 rows affected (0.05 sec)
```

Query 6B. Parameter OUT example

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE my_proc_OUT (OUT highest_salary INT)
    -> BEGIN
    -> SELECT MAX(MAX_SALARY) INTO highest_salary FROM jobs;
    -> END$$
Query OK, 0 rows affected (0.05 sec)
sql> DELIMITER ;

mysql> CALL my_proc_OUT(@M);
Query OK, 1 row affected (0.05 sec)

mysql> SELECT @M$$
+-------+
| @M    |
+-------+
| 40000 |
+-------+
1 row in set (0.05 sec)
```

Query 6C. Parameter INOUT example

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE my_proc_INOUT (INOUT empcnt INT, IN dept_id INT)
    -> BEGIN
    -> SELECT COUNT(employee_id) INTO empcnt FROM employees WHERE
   department_id = dept_id;
    -> END$$
Query OK, 0 rows affected (0.08 sec)
mysql> DELIMITER ;
```

```
mysql> CALL my_proc_INOUT(@C,10);
Query OK, 1 row affected (0.05 sec)

mysql> SELECT @C;
+------+
| @C   |
+------+
|    1 |
+------+
1 row in set (0.05 sec)

mysql> CALL my_proc_INOUT(@C,100);
Query OK, 1 row affected (0.05 sec)

mysql> SELECT @C;;
+------+
| @C   |
+------+
|    6 |
+------+
1 row in set (0.05 sec)
```

A7. Flow Control Statements

Query 7. If Statement

```
IF condition THEN statement(s)
[ELSEIF condition THEN statement(s)] ...
[ELSE statement(s)]
END IF
```

% In the following example, we pass user_id through IN parameter to get the user name. Within the procedure, we have used IF ELSEIF and ELSE statement to get user name against multiple user id. The user name will be stored into INOUT parameter user_name.

```
mysql> DELIMITER $$

mysql> CREATE PROCEDURE GetUserName(INOUT user_name varchar(16),
    -> IN user_id varchar(16))
    -> BEGIN
    -> DECLARE uname varchar(16);
    -> SELECT name INTO uname
    -> FROM user
    -> WHERE userid = user_id;
    -> IF user_id = "scott123"
    -> THEN
    -> SET user_name = "Scott";
    -> ELSEIF user_id = "ferp6734"
    -> THEN
    -> SET user_name = "Palash";
    -> ELSEIF user_id = "diana094"
    -> THEN
    -> SET user_name = "Diana";
    -> END IF;
    -> END$$
```

```
Query OK, 0 rows affected (0.05 sec)
mysql> DELIMITER ;

mysql> CALL GetUserName(@A,'scott123');
Query OK, 1 row affected (0.03 sec)

mysql> SELECT @A;
+-------+
| @A    |
+-------+
| Scott |
+-------+
```

Query 8. Case Statement

```
CASE case_value
WHEN when_value THEN statement_list
[WHEN when_value THEN statement_list] ...
[ELSE statement_list] END CASE
or

CASE
WHEN search_condition THEN statement_list
[WHEN search_condition THEN statement_list] ...
[ELSE statement_list] END CASE
```

% We want to count the number of employees with following conditions :
- MIN_SALARY > 10000
- MIN_SALARY < 10000
- MIN_SALARY = 10000

```
mysql> DELIMITER $$

mysql> CREATE PROCEDURE my_proc_CASE
    -> (INOUT no_employees INT, IN salary INT)
    -> BEGIN
    -> CASE
    -> WHEN (salary>10000)
    -> THEN (SELECT COUNT(job_id) INTO no_employees
    -> FROM jobs
    -> WHERE min_salary>10000);
    -> WHEN (salary<10000)
    -> THEN (SELECT COUNT(job_id) INTO no_employees
    -> FROM jobs
    -> WHERE min_salary<10000);
    -> ELSE (SELECT COUNT(job_id) INTO no_employees
    -> FROM jobs WHERE min_salary=10000);
    -> END CASE;
    -> END$$
Query OK, 0 rows affected (0.19 sec)
mysql> DELIMITER ;
```

% Number of employees whose salary greater than 10000 :

```
mysql> CALL my_proc_CASE(@C,10001);
Query OK, 1 row affected (1.39 sec)

mysql> SELECT @C;
```

```
+------+
| @C   |
+------+
|    2 |
+------+
1 row in set (0.23 sec)
```

% Number of employees whose salary less than 10000 :

```
mysql> CALL my_proc_CASE(@C,9999);
Query OK, 1 row affected (0.22 sec)

mysql> SELECT @C;
+------+
| @C   |
+------+
|   16 |
+------+
1 row in set (1.00 sec)
```

% Number of employees whose salary equal to 10000 :

```
mysql> CALL my_proc_CASE(@C,10000);
Query OK, 1 row affected (1.57 sec)

mysql> SELECT @C;
+------+
| @C   |
+------+
|    1 |
+------+
1 row in set (0.05 sec)
```

Query 9. Loop Statement

```
[begin_label:]
LOOP
statement_list
END LOOP
[end_label]
```

% In the following procedure rows will be inserted in 'number' table until x is less than num (number supplied by the user through IN parameter). A random number will be stored every time.

```
mysql> DROP TABLE number;
Query OK, 0 rows affected (1.15 sec)

mysql> CREATE TABLE number (num decimal);
Query OK, 0 rows affected (0.06 sec)

mysql> DELIMITER $$
mysql> CREATE PROCEDURE my_proc_LOOP (IN num INT)
    -> BEGIN
    -> DECLARE x INT;
    -> SET x = 0;
    -> loop_label: LOOP
    -> INSERT INTO number VALUES (rand());
    -> SET x = x + 1;
    -> IF x >= num
```

```
    -> THEN
    -> LEAVE loop_label;
    -> END IF;
    -> END LOOP;
    -> END$$
ERROR 1304 (42000): PROCEDURE my_proc_LOOP already exists
mysql> DELIMITER ;

mysql> CALL my_proc_LOOP(3);
Query OK, 1 row affected (0.09 sec)

mysql> select * from number;
+------+
| num  |
+------+
|    0 |
|    1 |
|    0 |
+------+
```

Return Statement

The RETURN statement terminates execution of a stored function and returns the value expr to the function caller. There must be at least one RETURN statement in a stored function. There may be more than one if the function has multiple exit points. Here is the syntax :

```
RETURN expr
```

Query 10. While Statement

The WHILE statement executes the statement(s) as long as the condition is true. The condition is checked every time at the beginning of the loop. Each statement is terminated by a semicolon (;). Here is the syntax :

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

```
mysql> DELIMITER $$

mysql> CREATE PROCEDURE my_proc_WHILE(IN n INT)
    -> BEGIN
    -> SET @sum = 0;
    -> SET @x = 1;
    -> WHILE @x<n
    -> DO
    ->    IF mod(@x, 2) <> 0 THEN
    -> SET @sum = @sum + @x;
    -> END IF;
    -> SET @x = @x + 1;
    -> END WHILE;
    -> END$$
Query OK, 0 rows affected (0.37 sec)
mysql> DELIMITER $$

mysql> CALL my_proc_WHILE(5);
Query OK, 0 rows affected (0.05 sec)

mysql> SELECT @sum;
+------+
```

```
| @sum |
+------+
|    4 |
+------+
1 row in set (0.05 sec)

mysql> CALL my_proc_WHILE(10);
Query OK, 0 rows affected (0.05 sec)

mysql> SELECT @sum;
+------+
| @sum |
+------+
|   25 |
+------+
1 row in set (0.05 sec)
```

A8. Alter and Drop Statements

Query 11. ALTER Procedure

```
ALTER PROCEDURE proc_name [characteristic ...]characteristic:
COMMENT 'string'
| LANGUAGE SQL
| { CONTAINS SQL
| NO SQL | READS SQL DATA
| MODIFIES SQL DATA }
| SQL SECURITY { DEFINER
| INVOKER }
```

```
mysql> ALTER PROCEDURE my_proc_WHILE
    -> COMMENT 'Modify Comment';
```

Query 12. DROP Procedure

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

```
mysql> DROP PROCEDURE my_proc_WHILE;
Query OK, 0 rows affected (0.05 sec)
```

A9. CURSOR

A database cursor is a control structure that enables traversal over the records in a database. Cursors are used by database programmers to process individual rows returned by database system queries. Cursors enable manipulation of whole result sets at once. In this scenario, a cursor enables the rows in a result set to be processed sequentially. In SQL procedures, a cursor makes it possible to define a result set (a set of data rows) and perform complex logic on a row by row basis. By using the same mechanics, an SQL procedure can also define a result set and return it directly to the caller of the SQL procedure or to a client application.

MySQL supports cursors inside stored programs. The syntax is as in embedded SQL. Cursors have these properties :
- Asensitive: The server may or may not make a copy of its result table
- Read only: Not updatable
- Nonscrollable: Can be traversed only in one direction and cannot skip rows

To use cursors in MySQL procedures, you need to do the following :
- Declare a cursor.
- Open a cursor.
- Fetch the data into variables.
- Close the cursor when done.

Declare a cursor:
The following statement declares a cursor and associates it with a SELECT statement that retrieves the rows to be traversed by the cursor.

```
DECLARE cursor_name
CURSOR FOR select_statement
```

Open a cursor :
The following statement opens a previously declared cursor.

```
OPEN cursor_name
```

Fetch the data into variables :
This statement fetches the next row for the SELECT statement associated with the specified cursor (which must be open) and advances the cursor pointer. If a row exists, the fetched columns are stored in the named variables. The number of columns retrieved by the SELECT statement must match the number of output variables specified in the FETCH statement.

```
FETCH [[NEXT] FROM] cursor_name
INTO var_name [, var_name] ...
```

Close the cursor when done :
This statement closes a previously opened cursor. An error occurs if the cursor is not open.

```
CLOSE cursor_name
```

Query 13. Cursor Example

The procedure starts with three variable declarations. Incidentally, the order is important. First, declare variables. Then declare conditions. Then declare cursors. Then, declare handlers. If you put them in the wrong order, you will get an error message.

```
mysql> DELIMITER $$

mysql> CREATE PROCEDURE my_procedure_cursors(INOUT return_val INT)
    -> BEGIN
    -> DECLARE a,b INT;
    -> DECLARE cur_1 CURSOR FOR
    -> SELECT max_salary FROM jobs;
    -> DECLARE CONTINUE HANDLER FOR NOT FOUND SET b = 1;
    -> OPEN cur_1;
    -> REPEAT FETCH cur_1 INTO a;
    -> UNTIL b = 1
    -> END REPEAT;
    -> CLOSE cur_1;
    -> SET return_val = a;
    -> END;
    -> $$
Query OK, 0 rows affected (0.05 sec)

mysql> DELIMITER ;
```

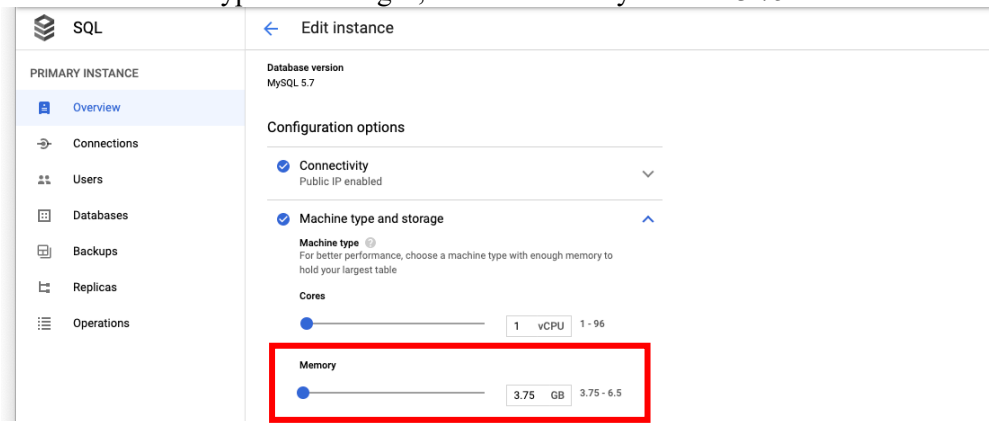
```
mysql> CALL my_procedure_cursors(@R);
Query OK, 0 rows affected (0.05 sec)

mysql> SELECT @R$$
+------+
| @R   |
+------+
| 8500 |
+------+
1 row in set (0.05 sec)
```

B. Trigger in MySQL

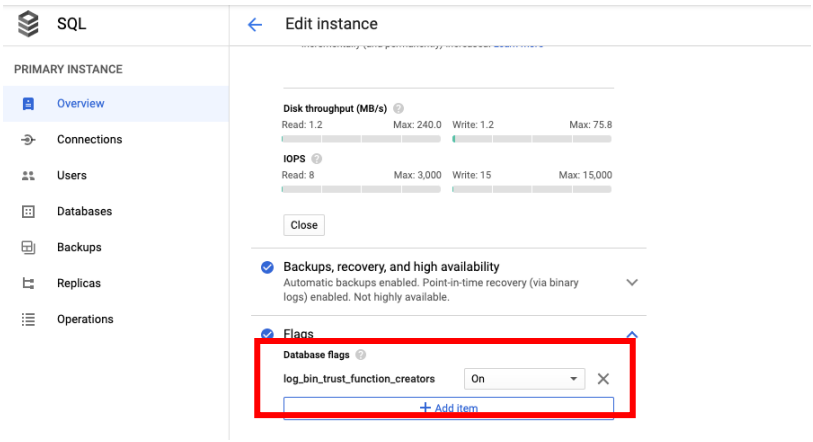
A trigger is a set of actions that are run automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a specified table. Triggers are useful for tasks such as enforcing business rules, validating input data, and keeping an audit trail.

Before you move on the next, you should change the configuration of MySQL running on GCP.

- Open GCP console → SQL service → Select MySQL instance → Select Overview
- Fine Configuration tile, then select "→ Edit Configuration"
- Select "Machine Type and storage", then set Memory size as "3.75 GB"



- Select "Flags"
- Find "log_bin_trust_function_creators" and set "ON"



- Save and restart the instance
- After restart, connect mysql

B1. Syntax

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
trigger_body
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
```

Query 14 Simple Example

```
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
(error due to missing account table)
```

B2. Example AFTER INSERT

In the following example, we have two tables : emp_details and log_emp_details. To insert some information into log_ emp_details table (which have three fields employee id and salary and edttime) every time, when an INSERT happen into emp_details table we have used the following trigger :

Query 15.

```
mysql> use hr;
mysql> CREATE TABLE log_emp_details (
    -> emp_details int(11),
    -> salary decimal(8,2),
    -> edittime datetime);
Query OK, 0 rows affected (0.06 sec)

mysql> CREATE TRIGGER emp_details_AINS
    -> AFTER INSERT ON employees
    -> FOR EACH ROW
    -> -- Edit trigger body code below this line. Do not edit lines above
  this one
    -> BEGIN
```

```
        -> INSERT INTO log_emp_details
        -> VALUES(NEW.employee_id, NEW.salary, NOW());
        -> END$$
    Query OK, 0 rows affected (0.17 sec)

    mysql> select * from log_emp_details$$
    Empty set (0.05 sec)

    mysql> INSERT INTO employees VALUES (236,'RABI', 'CHANDRA',
          -> 'RABI','590.423.45700',
          ->  STR_TO_DATE('12-JAN-2013', '%d-%M-%Y'),'AD_VP', 15000,
          ->  NULL, NULL,90)$$
    Query OK, 1 row affected (0.15 sec)

    mysql> SELECT * FROM employees WHERE employee_id = 236$$
+-------------+------------+-----------+-------+---------------+------------
  +--------+----------+----------------+------------+---------------+
| employee_id | first_name | last_name | email | phone_number  | hire_date  |
  job_id | salary   | commission_pct | manager_id | department_id |
+-------------+------------+-----------+-------+---------------+------------
  +--------+----------+----------------+------------+---------------+
|         236 | RABI       | CHANDRA   | RABI  | 590.423.45700 | 2013-01-12 |
  AD_VP  | 15000.00 |           NULL |       NULL |            90 |
+-------------+------------+-----------+-------+---------------+------------
  +--------+----------+----------------+------------+---------------+
1 row in set (0.05 sec)

    mysql> SELECT * FROM log_emp_details$$
    +-------------+----------+---------------------+
    | emp_details | salary   | edittime            |
    +-------------+----------+---------------------+
    |         236 | 15000.00 | 2017-08-22 02:04:34 |
    +-------------+----------+---------------------+
    1 row in set (0.04 sec)
```

B3. Example BEFORE INSERT

In the following example, before insert a new record in employees table, a trigger check the column value of FIRST_NAME, LAST_NAME, JOB_ID and
- If there are any space(s) before or after the FIRST_NAME, LAST_NAME, TRIM() function will remove those.
- The value of the JOB_ID will be converted to upper cases by UPPER() function.

Query 16.

```
mysql> use hr;

mysql> DELIMITER $$

mysql> CREATE TRIGGER emp_details_BINS
    -> BEFORE INSERT
    -> ON employees FOR EACH ROW
    -> -- Edit trigger body code below this line. Do not edit lines above
   this one
    -> BEGIN
    -> SET NEW.FIRST_NAME = TRIM(NEW.FIRST_NAME);
```

```
    -> SET NEW.LAST_NAME = TRIM(NEW.LAST_NAME);
    -> SET NEW.JOB_ID = UPPER(NEW.JOB_ID);
    -> END;$$
Query OK, 0 rows affected (0.23 sec)
mysql> DELIMITER ;

mysql> INSERT INTO employees VALUES (334, ' Ana ', ' King', 'ANA',
      -> '690.432.45701', STR_TO_DATE('05-FEB-2013', '%d-%M-%Y'),
      -> 'it_prog', 17000, NULL, NULL,90);
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM employees WHERE employee_id = 334;
+-------------+------------+-----------+-------+--------------+-----------
  -+---------+----------+----------------+------------+---------------+
| employee_id | first_name | last_name | email | phone_number  | hire_date
  | job_id  | salary   | commission_pct | manager_id | department_id |
+-------------+------------+-----------+-------+--------------+-----------
  -+---------+----------+----------------+------------+---------------+
|         334 | Ana        | King      | ANA   | 690.432.45701 | 2013-02-05
  | IT_PROG | 17000.00 |           NULL |       NULL |            90 |
+-------------+------------+-----------+-------+--------------+-----------
  -+---------+----------+----------------+------------+---------------+
1 row in set (0.05 sec)
```

Check this:

```
FIRST_NAME - > ' Ana ' has changed to 'Ana'
LAST_NAME - > ' King' has changed to 'King'
JOB_ID - > ' it_prog' has changed to 'IT_PROG'
```

Exercise

Exercise 1: Creating Two Tables

For this exercise, first create a student_mast table including three columns STUDENT_ID, NAME, ST_CLASS.

```
mysql> SELECT * FROM STUDENT_MAST;
+------------+------------------+----------+
| STUDENT_ID | NAME             | ST_CLASS |
+------------+------------------+----------+
|          1 | Steven King      |        7 |
|          2 | Neena  Kochhar   |        8 |
|          3 | Lex  De Haan     |        8 |
|          4 | Alexander Hunold |       10 |
+------------+------------------+----------+
```

stu_log table has two columns student_id and description.

Create student_mast and stu_log tables.

Exercise 2: Creating Trigger for AFTER UPDATE

Let us promote all the students in next class i.e., 7 will be 8, 8 will be 9 and so on. After updating a single row in student_mast table a new row will be inserted in stu_log table where we will store the current student id and a small description regarding the current update.

- Create a trigger
- Update student_mast table using the following:
- mysql> UPDATE STUDENT_MAST SET ST_CLASS = ST_CLASS + 1;
- Select both student_mast and stu_log tables

Submit a source of trigger and a screenshot to Canvas.

Lab 8: Database Programming with Python

Use lab8 folder

- `Change directory into` **`lab8`** `directory`
- `Save the file and extract it under Lab folder`
- `Remember to start by creating a silicon database to use.`

A. Python Application

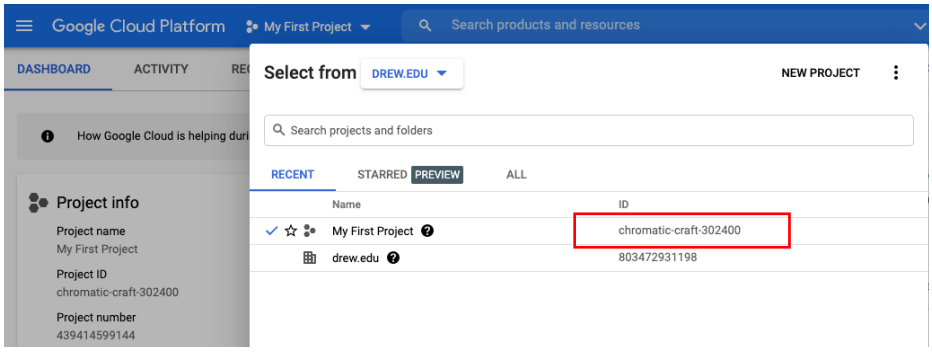
EXERCISE DESCRIPTION: In this exercise, we will create simple python applications that connect MySQL instance directly and displays the entries in the database table in Google cloud platform.

- Required software:
  - Python Version = 3.7 +
  - pip
  - virtualenv (`$ pip install virtualenv`)

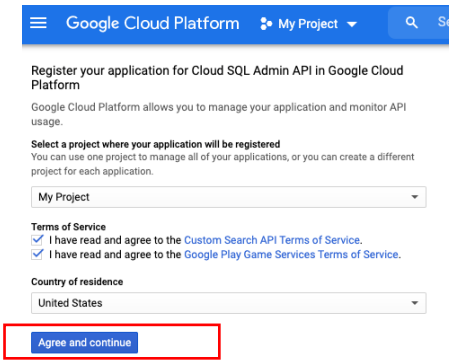
A1. Requirements

- Python 3.7 or higher is required
- Google Cloud Project
  - In the Google Cloud Console, on the project selector page, select or create a Google Cloud project.
  - You can use a default project, i.e., "My First Project"
  - Check and copy the project ID (not Project Name). The project ID will be used during Google Cloud SDK installation and setting.

    Project ID [ ]



- Enable the Cloud SQL Admin API.
  - From "APIs & Services" → Dashboard
  - Select "+ ENABLE APIS AND SERVICES"
  - Search "SQL Admin" → Select "Cloud SQL Admin API"
  - Select a project, e.g., "My First Project" and Checkbox



  - "Agree and continue"

- Install and initialize the Cloud SDK on your computer with the following link https://cloud.google.com/sdk/docs/install
  - Note that most of computers are 64 bit
  - Optional are optional (you can install them if you want, but not required)
- Log in to gcloud
  - Acquire new credentials to use the Cloud SQL Admin API:
  - Open your terminal (MacOS) or PowerShell (Windows)

```
$ gcloud auth application-default login
```

  - Chose your GCP account

A2. Configuring MySQL database

- Create a database

```
$ mysql --host=[MySQL Public IP] --user=[DB_USER] --password
mysql> CREATE DATABASE company;
mysql> use company;
mysql> source cr_emp.sql
mysql> source cr_dept.sql
mysql> desc employee;
```

```
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| fname     | varchar(8)  | YES  |     | NULL    |       |
| minit     | varchar(2)  | YES  |     | NULL    |       |
| lname     | varchar(8)  | YES  |     | NULL    |       |
| ssn       | varchar(9)  | NO   | PRI | NULL    |       |
| bdate     | date        | YES  |     | NULL    |       |
| address   | varchar(27) | YES  |     | NULL    |       |
| sex       | varchar(1)  | YES  |     | NULL    |       |
| salary    | int(7)      | NO   |     | NULL    |       |
| super_ssn | varchar(9)  | YES  |     | NULL    |       |
| dno       | int(1)      | NO   |     | NULL    |       |
```

```
+-----------+-------------+------+-----+---------+-------+
10 rows in set (0.05 sec)

mysql> desc department;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| dnumber        | int(1)      | YES  |     | NULL    |       |
| dname          | varchar(15) | YES  |     | NULL    |       |
| mgr_ssn        | varchar(9)  | YES  |     | NULL    |       |
| mgr_start_date | date        | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
```

A3. Install Python Package

Create an isolated Python environment if needed.

```
$ virtualenv dbenv
For specific Python version $ virtualenv -p python3.8 dbenv
For Mac $ source dbenv/bin/activate
[For Win $ dbenv\scripts\activate]
(dbenv) $ pip install -r requirements.txt
(dbenv) $ pip install mysqlclient
(dbenv) $ pip install mysql-connector-python
(dbenv) $ pip install PyMySQL
```

Note that all exercise lab in this chapter will be performed under `(dbenv)` virtual environment even though it is not specified.

A4. MySQL adapters for Python

There are three MySQL adapters for Python that are currently maintained:

- `mysqlclient` : By far the fastest MySQL connector for CPython. Requires the mysql-connector-c C library to work.

- `PyMySQL` : Pure Python MySQL client. According to the maintainer of both mysqlclient and PyMySQL, you should use PyMySQL if:
  - You can't use libmysqlclient for some reason.
  - You want to use monkeypatched socket of gevent or eventlet.
  - You wan't to hack mysql protocol.

- `mysql-connector-python` : MySQL connector developed by the MySQL group at Oracle, also written entirely in Python. It's performance appears to be the worst out of the three. Also, due to some licensing issues, you can't download it from PyPI. For more info about MySQL connector:

- https://dev.mysql.com/doc/connector-python/en/connector-python-introduction.html

In this lab, we are going to use `mysql-connector-python` adaptor for Python.

A6. Python Application to SELECT from MySQL using MySQL Connector

- File name: A5.py
- Basic database programming with Python and MySQL connector
  - SQL: retrieve all employees and display
- Replace the following in A6.py
  - SERVER_URL : IP address for MySQL instance on GCP
  - DB : Database name of MySQL on GCP
  - USER_NAME : database user name of MySQL on GCP
  - PASSWORD : database password of MySQL on GCP

```
(dbenv) $ python A6.py
```

```
import mysql.connector

def main():
    SERVER_URL = "11.11.11.11"
    DB = "[DB_NAME]"
    USER_NAME = "[USER_NAME]"
    PASSWORD = "[PASSWORD]"

    SQL_CONNECTION = mysql.connector.connect(host=SERVER_URL,
                        user=USER_NAME,
                        password=PASSWORD,
                        database=DB)

    SQL = """SELECT * FROM employee"""

    with SQL_CONNECTION.cursor() as cursor:
        try:
            cursor.execute(SQL)
            results = cursor.fetchall()
            if results:
                display_results(results)
            else:
                print("No Record")
        except (mysql.connector.Error, mysql.connector.Warning)
as e:
            print(f'error! {e}')

        finally:
            SQL_CONNECTION.close()

def display_results(results):
```

```
        for row in results:
            print(row)

if __name__ == '__main__':
    main()
```

A7. Python Application to Dynamic SQL-1 using MySQL Connector

- File name: A7.py
- Dynamic SQL database programming with Python and MySQL connector
  - SQL: retrieve an employee for a specific SSN
  - User inputs SSN and assemble SQL using string concatenation
  - Output: dictionary format
- Replace the following in A7.py
  - SERVER_URL : IP address for MySQL instance on GCP
  - DB : Database name of MySQL on GCP
  - USER_NAME : database user name of MySQL on GCP
  - PASSWORD : database password of MySQL on GCP

```
(dbenv) $ python A7.py

import mysql.connector

def main():
    SERVER_URL = "11.11.11.11"
    DB = "[DB_NAME]"
    USER_NAME = "[USER_NAME]"
    PASSWORD = "[PASSWORD]"

    SQL_CONNECTION = mysql.connector.connect(host=SERVER_URL,
                        user=USER_NAME,
                        password=PASSWORD,
                        database=DB)

    while True:
        SQL = """SELECT * FROM employee WHERE """
        ssn = input("Enter SSN to search. 0 for Exit: ")

        if ssn != "0":
            SQL = SQL + "ssn = " + ssn
            SQL_CONNECTION.reconnect()
            with SQL_CONNECTION.cursor(dictionary=True) as
cursor:
                try:
                    cursor.execute(SQL)
                    results = cursor.fetchall()
                    if results:
```

```
                        display_results(results)
                    else:
                        print("No Record")
                except (mysql.connector.Error,
mysql.connector.Warning) as e:
                    print(f'error! {e}')

                finally:
                    SQL_CONNECTION.close()
            else:
                break


def display_results(results):
    for row in results:
        for col, val in row.items():
            print(col, ":", val)
        print(" ")


if __name__ == '__main__':
    main()
```

A8. Python Application to Dynamic SQL-2 using MySQL Connector

- File name: A8.py
- Dynamic SQL database programming with Python and MySQL connector
  - SQL: retrieve an employee for a specific SSN
  - User inputs SSN and assemble SQL using cursor.execute
  - Output: dictionary format
- Replace the following in A8.py
  - SERVER_URL : IP address for MySQL instance on GCP
  - DB : Database name of MySQL on GCP
  - USER_NAME : database user name of MySQL on GCP
  - PASSWORD : database password of MySQL on GCP

```
(dbenv) $ python A8.py

import mysql.connector

def main():
    SERVER_URL = "11.11.11.11"
    DB = "[DB_NAME]"
    USER_NAME = "[USER_NAME]"
    PASSWORD = "[PASSWORD]"

    SQL_CONNECTION = mysql.connector.connect(host=SERVER_URL,
                        user=USER_NAME,
```

```
                            password=PASSWORD,
                            database=DB)

    while True:
        SQL = """SELECT * FROM employee WHERE ssn = %(ssn)s"""
        ssn = input("Enter SSN to search. 0 for Exit: ")
        val = {'ssn': ssn}

        if ssn != "0":
            SQL_CONNECTION.reconnect()
            with SQL_CONNECTION.cursor(dictionary=True) as
cursor:
                try:
                    cursor.execute(SQL, val)
                    results = cursor.fetchall()
                    if results:
                        display_results(results)
                    else:
                        print("No Record")
                except (mysql.connector.Error,
mysql.connector.Warning) as e:
                    print(f'error! {e}')
                finally:
                    SQL_CONNECTION.close()
        else:
            break


def display_results(results):
    for row in results:
        for col, val in row.items():
            print(col, ":", val)
        print(" ")

if __name__ == '__main__':
    main()
```

A9 Python Application to SELECT from MySQL using PyMySQL

- File name: A9.py
- Basic database programming with Python and PyMySQL connector
    - SQL: retrieve all employees
    - Output: dictionary format
- Replace the following in A9.py
    - SERVER_URL : IP address for MySQL instance on GCP
    - DB : Database name of MySQL on GCP
    - USER_NAME : database user name of MySQL on GCP
    - PASSWORD : database password of MySQL on GCP

```
(dbenv) $ python A9.py
```

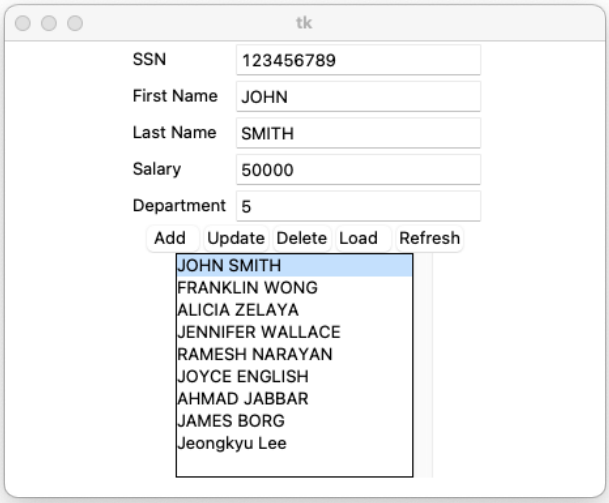
```
import pymysql.cursors

def main():
    SERVER_URL = "11.11.11.11"
    DB = "[DB_NAME]"
    USER_NAME = "[USER_NAME]"
    PASSWORD = "[PASSWORD]"

    SQL_CONNECTION = pymysql.connect(host=SERVER_URL,
                        user=USER_NAME,
                        passwd=PASSWORD,
                        db=DB,
                        cursorclass=pymysql.cursors.DictCursor,
                        autocommit=True)

    SQL = """SELECT * FROM employee"""

    with SQL_CONNECTION.cursor() as cursor:
        try:
            sql_exec = cursor.execute(SQL)
            if sql_exec:
                #print(sql_exec)
                results = cursor.fetchall()
                for row in results:
                    print(row)
            else:
                print(sql_exec)
                print("No Record")
        except (pymysql.Error, pymysql.Warning) as e:
            print(f'error! {e}')

        finally:
            SQL_CONNECTION.close()

if __name__ == '__main__':
    main()
```

A10. Python GUI Application using tkinter

- File name: A10.py
- GUI database application for CRUD
  - GUI: tkinter
  - SQL: dynamic SQL for select, update, delete and insert
- Replace the following in A9.py
  - SERVER_URL : IP address for MySQL instance on GCP

- DB : Database name of MySQL on GCP
- USER_NAME : database user name of MySQL on GCP
- PASSWORD : database password of MySQL on GCP

```
(dbenv) $ python A10.py
```




```
import mysql.connector
from tkinter import *

SERVER_URL = "11.11.11.11"
DB = "[DB_NAME]"
USER_NAME = "[USER_NAME]"
PASSWORD = "[PASSWORD]"

SQL_CONNECTION = mysql.connector.connect(host=SERVER_URL,
                    user=USER_NAME,
                    password=PASSWORD,
                    database=DB)

def main():
    win = makeWindow()
    setSelect()
    win.mainloop()

def whichSelected():
    return int(select.curselection()[0])

def loadEntry():
    ssn, fname, lname, salary, dno = empList[whichSelected()]
    ssnVar.set(ssn)
    fnameVar.set(fname)
    lnameVar.set(lname)
    salaryVar.set(salary)
```

```
    dnoVar.set(dno)

def addEntry():
    SQL = """insert into employee (ssn, fname, lname, salary,
dno) values ('%s','%s','%s',%s,%s)""" % \
          (ssnVar.get(),fnameVar.get(), fnameVar.get(),
salaryVar.get(), dnoVar.get())

    SQL_CONNECTION.reconnect()
    with SQL_CONNECTION.cursor() as cursor:
        try:
            cursor.execute(SQL)
        except (mysql.connector.Error, mysql.connector.Warning)
as e:
            print(f'error! {e}')
        finally:
            SQL_CONNECTION.commit()
            SQL_CONNECTION.close()


def deleteEntry():
    ssn = empList[whichSelected()][0]
    SQL = """delete from employee where ssn='%s'""" % ssn

    SQL_CONNECTION.reconnect()
    with SQL_CONNECTION.cursor() as cursor:
        try:
            cursor.execute(SQL)
        except (mysql.connector.Error, mysql.connector.Warning)
as e:
            print(f'error! {e}')
        finally:
            SQL_CONNECTION.commit()
            SQL_CONNECTION.close()

def updateEntry():
    ssn = empList[whichSelected()][0]
    SQL = """update employee set fname='%s', lname='%s',
salary=%s, dno=%s where ssn='%s'""" % \
          (fnameVar.get(), lnameVar.get(), salaryVar.get(),
dnoVar.get(), ssn)
    #print(SQL)
    SQL_CONNECTION.reconnect()
    with SQL_CONNECTION.cursor() as cursor:
        try:
            cursor.execute(SQL)
        except (mysql.connector.Error, mysql.connector.Warning)
as e:
            print(f'error! {e}')
        finally:
            SQL_CONNECTION.commit()
            SQL_CONNECTION.close()
```

```
def makeWindow () :
    global ssnVar, fnameVar, lnameVar, salaryVar, dnoVar, select
    win = Tk()

    frame1 = Frame(win)
    frame1.pack()

    Label(frame1, text="SSN").grid(row=0, column=0, sticky=W)
    ssnVar = StringVar()
    ssn = Entry(frame1, textvariable=ssnVar)
    ssn.grid(row=0, column=1, sticky=W)

    Label(frame1, text="First Name").grid(row=1, column=0,
sticky=W)
    fnameVar = StringVar()
    fname = Entry(frame1, textvariable=fnameVar)
    fname.grid(row=1, column=1, sticky=W)

    Label(frame1, text="Last Name").grid(row=2, column=0,
sticky=W)
    lnameVar = StringVar()
    lname = Entry(frame1, textvariable=lnameVar)
    lname.grid(row=2, column=1, sticky=W)

    Label(frame1, text="Salary").grid(row=3, column=0, sticky=W)
    salaryVar= StringVar()
    salary = Entry(frame1, textvariable=salaryVar)
    salary.grid(row=3, column=1, sticky=W)

    Label(frame1, text="Department").grid(row=4, column=0,
sticky=W)
    dnoVar= StringVar()
    dno = Entry(frame1, textvariable=dnoVar)
    dno.grid(row=4, column=1, sticky=W)

    frame2 = Frame(win)        # Row of buttons
    frame2.pack()
    b1 = Button(frame2,text=" Add  ",command=addEntry)
    b2 = Button(frame2,text="Update",command=updateEntry)
    b3 = Button(frame2,text="Delete",command=deleteEntry)
    b4 = Button(frame2, text="Load  ", command=loadEntry)
    b5 = Button(frame2, text="Refresh", command=setSelect)
    b1.pack(side=LEFT)
    b2.pack(side=LEFT)
    b3.pack(side=LEFT)
    b4.pack(side=LEFT)
    b5.pack(side=LEFT)

    frame3 = Frame(win)        # select of names
    frame3.pack()
    scroll = Scrollbar(frame3, orient=VERTICAL)
```

```
    select = Listbox(frame3, yscrollcommand=scroll.set,
height=10)
    scroll.config (command=select.yview)
    scroll.pack(side=RIGHT, fill=Y)
    select.pack(side=LEFT,  fill=BOTH, expand=1)
    return win

def setSelect () :
    global empList
    SQL = """select ssn, fname, lname, salary, dno from
employee"""

    SQL_CONNECTION.reconnect()
    with SQL_CONNECTION.cursor() as cursor:
        try:
            cursor.execute(SQL)
            empList = cursor.fetchall()
            if empList:
                display_results(empList)
            else:
                print("No Record")
        except (mysql.connector.Error, mysql.connector.Warning)
as e:
            print(f'error! {e}')

        finally:
            SQL_CONNECTION.close()

    select.delete(0, END)
    for ssn, fname, lname, salary, dno in empList:
        select.insert(END, fname + " " + lname)


def display_results(results):
    for row in results:
        print(row)

if __name__ == '__main__':
    main()
```

Exercise: Create Python Application

First, create MAIL_ORDER database of Exercise 3.32 in the text book as follows:

```
Connect to MySQL instance
Connect to lab database

mysql> source cr_mailorder.sql
```

Write and test a python application program that display an invoice for a given Order number.

- File name: invoice.py
    - As a result, in your terminal, you need to display all this information which is related to the given Order Number [Customer Name, Customer No, ZIP Code, Taken By, Received on, Shipped on, Part No, Part Name, Quantity, Price and Sum].
    - Display Total Amount at the last line.
    - Your application asks a user to input “Order number”, then you need to use “Dynamic SQL” technique to create SQL

Lab 9: Database Programing with Django

Use lab9 folder

- `Change directory into` **`lab9`** `directory`
- `Save the file and extract it under Lab folder`
- `Remember to start by creating a silicon database to use.`

A. Python 3 with Django

EXERCISE DESCRIPTION: In this exercise, we will create a simple python Django project that connect MySQL instance and displays the entries in the database table in Google cloud platform.

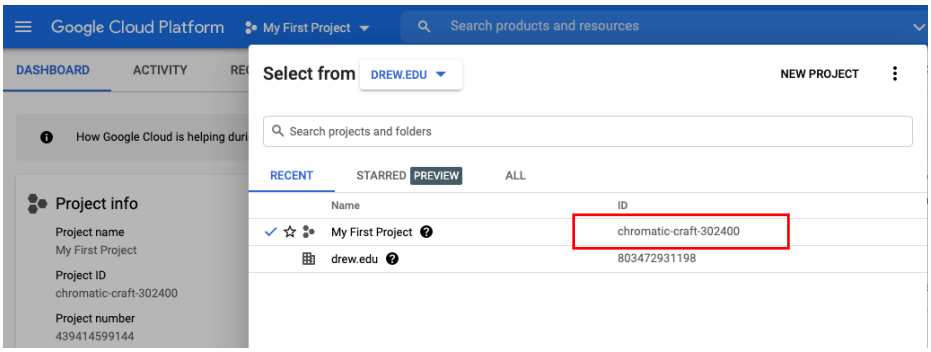
- Required software:
  - Python Version = 3.7 +
  - pip
  - virtualenv (`$ pip install virtualenv`)

* Note: If you already complete the previous Python Application, i.e., **Lab8**, you can skip **A1** ~ **A4** and start from **A5**.

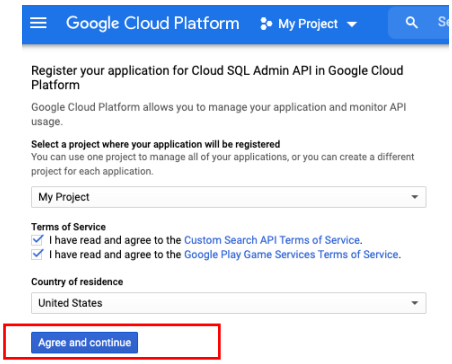
A1. Requirements

- Python 3.7 or higher is required
- Google Cloud Project
  - In the Google Cloud Console, on the project selector page, select or create a Google Cloud project.
  - You can use a default project, i.e., "My First Project"
  - Check and copy the project ID (not Project Name)

    Project ID [ ]



- Enable the Cloud SQL Admin API.
  - From "APIs & Services" → Dashboard
  - Select "+ ENABLE APIS AND SERVICES"
  - Search "SQL Admin" → Select "Cloud SQL Admin API"
  - Select a project, e.g., "My First Project" and Checkbox



  - "Agree and continue"
- Install and initialize the Cloud SDK on your computer with the following link https://cloud.google.com/sdk/docs/install
  - Note that most of computers are 64 bit
  - Optional are optional (you can install them if you want, but not required)
- Log in to gcloud
  - Acquire new credentials to use the Cloud SQL Admin API:
  - Open your terminal (MacOS) or PowerShell (Windows)

```
$ gcloud auth application-default login
```

  - Chose your GCP account

A2. Directory for Django App

- `dbtest` : Sample Django Project name. If you are using a different name, you need to replace dbetst by your project name
- `dbapp` : Sample Django App name. If you are using a different name, you need to replace dbapp by your app name

A3. Configuring MySQL database

- Create a database

```
$ mysql --host=[MySQL Public IP] --user=[DB_USER] --password
```

mysql --host=35.233.156.223 -u root -p

```
mysql> CREATE DATABASE company;
mysql> use company;
mysql> source cr_emp.sql
mysql> source cr_dept.sql
mysql> desc employee;

+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| fname     | varchar(8)  | YES  |     | NULL    |       |
```

```
| minit     | varchar(2)  | YES  |     | NULL    |       |
| lname     | varchar(8)  | YES  |     | NULL    |       |
| ssn       | varchar(9)  | NO   | PRI | NULL    |       |
| bdate     | date        | YES  |     | NULL    |       |
| address   | varchar(27) | YES  |     | NULL    |       |
| sex       | varchar(1)  | YES  |     | NULL    |       |
| salary    | int(7)      | NO   |     | NULL    |       |
| super_ssn | varchar(9)  | YES  |     | NULL    |       |
| dno       | int(1)      | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
10 rows in set (0.05 sec)

mysql> desc department;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| dnumber        | int(1)      | YES  |     | NULL    |       |
| dname          | varchar(15) | YES  |     | NULL    |       |
| mgr_ssn        | varchar(9)  | YES  |     | NULL    |       |
| mgr_start_date | date        | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
```

A4. Install Django

Create an isolated Python environment

```
$ virtualenv dbenv
For specific Python version $ virtualenv -p python3.8 dbenv
For Mac $ source dbenv/bin/activate
[For Win $ dbenv\scripts\activate]
(dbenv) $ pip install -r requirements.txt
(dbenv) $ pip install mysqlclient
```

Note that all exercise lab in this chapter will be performed under `(dbenv)` virtual environment even though it is not specified.

A5. Create Django Project

From the command line, cd into a directory where you'd like to store your code, then run the following command:

```
$ django-admin startproject [your-project-name]
```

e.g.,

```
$ django-admin startproject dbtest
```

The above command will create the necessary Django project files in the directory. Then, go inside the directory and open the folder with the same name as your project name and open the `settings.py` file and do the following modifications

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'Your database name',
        'USER': 'your mysql username',
        'PASSWORD': 'your mysql instance password',
        'HOST': 'Your IP for your mysql instance',
        'PORT': '3306'
    }
}
```

For example:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'company',
        'USER': 'jelee',
        'PASSWORD': 'password',
        'HOST': '11.11.11.11',
        'PORT': '3306'
    }
}
```

A6. Create a Simple Django App

From the command line, **cd** into your project directory and type the below command:

```
$ python manage.py startapp [your_app_name]
```

e.g.,

```
$ python manage.py startapp dbapp
```

The above command will create the necessary Django app files in the directory.

Whenever you create a new app in your Django project, you should include your app in the **INSTALLED_APPS** list in `settings.py` file in order for Django to recognize it.

```
INSTALLED_APPS = [
    'YOUR_APP',
]
```

For example:

```
INSTALLED_APPS = [
    'django.contrib.admin',
```

```
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.messages',
        'django.contrib.staticfiles',
        'dbapp',
    ]
```

A6.1 Auto-Generate the Models (models.py)

Django comes with a utility called **inspectdb** that can create models by introspecting an existing database. You can view the output by running this command under your app folder (i.e., dbapp) :

```
$ python manage.py inspectdb
```

Save this as a file by using standard Unix output redirection (or copy and paste):

```
$ python manage.py inspectdb > models.py
```

Then move models.py file under newly created app (e.g., dbapp) folder. By default, inspectdb creates unmanaged models. That is, `managed = False` in the model's Meta class tells Django not to manage each table's creation, modification, and deletion:

In order for the below code to work you have to create a database called **company** and database table called **employee** and **department** in it, or you can create your own database and tables and change the code accordingly.

```
from django.db import models

class Department(models.Model):
    dnumber = models.IntegerField(blank=True, null=True)
    dname = models.CharField(max_length=15, blank=True, null=True)
    mgr_ssn = models.CharField(max_length=9, blank=True, null=True)
    mgr_start_date = models.DateField(blank=True, null=True)

    class Meta:
        #managed = False
        db_table = 'department'


class Employee(models.Model):
    fname = models.CharField(max_length=8, blank=True, null=True)
    minit = models.CharField(max_length=2, blank=True, null=True)
    lname = models.CharField(max_length=8, blank=True, null=True)
    ssn = models.CharField(primary_key=True, max_length=9)
    bdate = models.DateField(blank=True, null=True)
    address = models.CharField(max_length=27, blank=True, null=True)
    sex = models.CharField(max_length=1, blank=True, null=True)
```

```
    salary = models.IntegerField()
    super_ssn = models.CharField(max_length=9, blank=True, null=True)
    dno = models.IntegerField()

    class Meta:
        #managed = False
        db_table = 'employee'
```

Next, run the **migrate** command to install any extra needed database records such as admin permissions and content types (at project home):

```
$ python manage.py migrate
```

A6.2 Creating Views (views.py)

Next go to the views.py file and type in the below code:

```
from django.shortcuts import render
from .models import Employee


def testmysql(request):
    employee = Employee.objects.all()
    context = {
        'user_ssn': employee[0].ssn,
        'user_name': employee[0].lname,
    }
    return render(request, 'home.html', context)
```

A6.3 Adding the URLs (urls.py)

Whenever your create a function based views (in this case), you should assign a url for django to get the result out of this function which is nothing but a html file and this can be done in `urls.py` file in your project folder.

```
from django.contrib import admin
from django.urls import path
from [your_App_name] import views

urlpatterns = [
    #path('admin/', admin.site.urls),
    path('', views.testmysql),
]
```

A6.4 Adding the Templates (templates/home.html)

Next create a folder called **templates** under your project folder which contains all the html files for your project and include a home.html file with the below code.

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Test MySQL</title>
</head>
<body>
 <h1>EMPLOYEE SSN: {{ user_ssn }}</h1>
 <h1>NAME: {{ user_name }} </h1>
</body>
</html>
```

In order for your Django project to recognize the html files in the template folder, you should include this in the **TEMPLATES** list entry in `settings.py` (under your project folder) as below:

```
import os
...

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Also, in the same file (settings.py), you need to add all (‘*’) to ALLOWED_HOSTS as below:

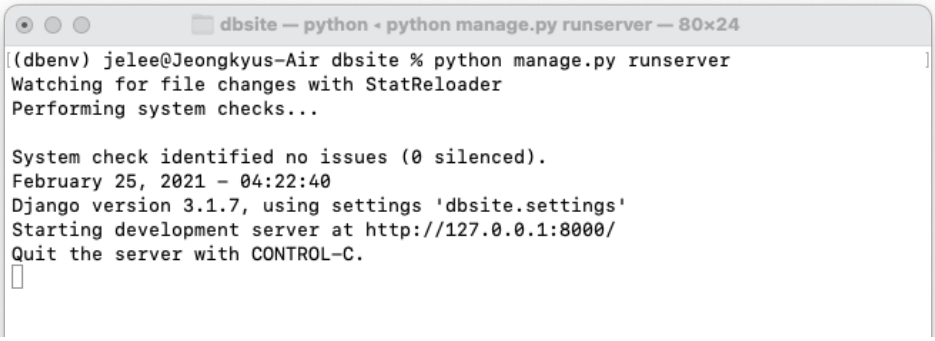
```
ALLOWED_HOSTS = ['*']
```

The highlighted code is a way to tell Django to look for contents stored in the templates folder in your project

A7 Run development server (on your local computer)

Next step is to run your Django project using the below command

```
$ python manage.py runserver
```



Copy and paste the development url in your browser and see the results.

http://127.0.0.1:8000/

A8 Deploying the app to GCP App Engine standard environment

- Gather all the static content into one folder by moving all of the app's static files into the folder specified by `STATIC_ROOT` at the end of `settings.py`:

```
...
STATIC_ROOT = 'static'
STATIC_URL = '/static/'
```

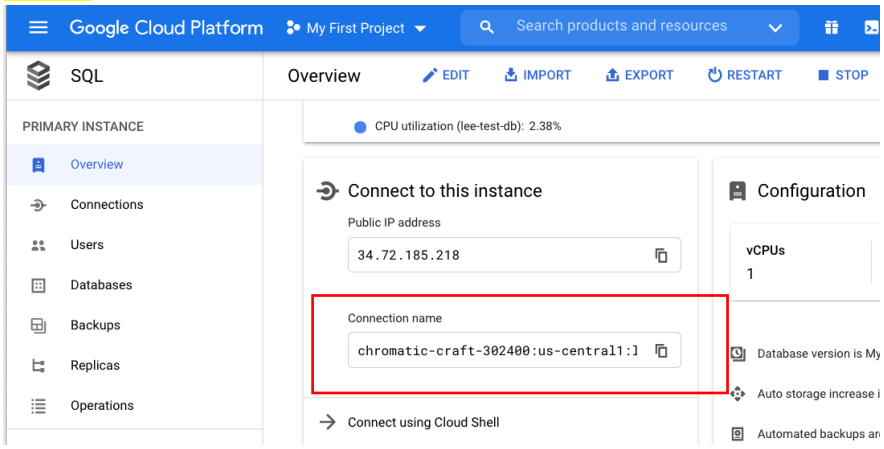
- Run the following command in your project folder

```
$ python manage.py collectstatic
```

- Copy the following files from the Sec7 lab folder to your project folder: `dbtest` should be replace by your project name.

```
app.yaml
main.py
   o  from [Your_Project_Name].wsgi import application

noxfile_config.py
   o  'DJANGO_SETTINGS_MODULE': '[Your_Project_Name].settings'

requirements.txt
```

- The `settings.py` file contains the configuration for your SQL database. The code in settings.py uses the `GAE_APPLICATION` environment variable to determine whether the app is running on App Engine or running on your local computer:
    - When the app runs on App Engine, it connects to the MySQL host by using the `/cloudsql` Unix socket.
    - Find mysql connection string from GCP console → SQL → Overview → Connection to This instance menu
    - Copy the Connection Name, and replace it for `/cloudsql/[MySQL Connection Name]`




```
# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases
import pymysql  # noqa: 402
pymysql.version_info = (1, 4, 6, 'final', 0)  # change mysqlclient version
pymysql.install_as_MySQLdb()
# [START db_setup]
if os.getenv('GAE_APPLICATION', None):
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'HOST': '/cloudsql/[MySQL Connection Name]',
            'USER': 'DBUSER',
            'PASSWORD': 'DBPASS',
            'NAME': 'company',
        }
    }
else:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME': 'company',
            'USER': 'DBUSER',
            'PASSWORD': 'DBPASS',
            'HOST': '11.11.11.11',
            'PORT': '3306'
        }
    }
# [END db_setup]
```

- Enter this command to deploy the sample:

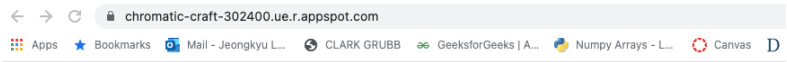
  ```
  $ gcloud app deploy
  ```

- In your browser, enter the following URL:

  `https://`**`PROJECT_ID`**`.`**`REGION_ID`**`.r.appspot.com/`
  - Replace the following:
  - **`PROJECT_ID:`** Your Google Cloud project ID
  - **`REGION_ID:`** A code that App Engine assigns to your app

- For company app,

  `https://`**`PROJECT_ID`**`.`**`REGION_ID`**`.r.appspot.com/`



MIAS USER ID: 123456789

MIAS USER NAME: SMITH

Exercise: Completing Install and Sample Django

First, complete Exercise Lab 9 including:

- Install Django
- Create Django project [Name]
- Create a simple Django App [Name]
- Setup the simple Django App
- Deploy your simple Django App [URL]

Submit the following information:

- Django project name:
- Django app name:
- Django app URL:

Your Django app should be running correctly to get full credit.

Lab 10: Database Programing with Django

Use lab10 folder

- There is no file in **lab10** directory.
- Instead, you are going to create your own project and app by following Lab 9.

A. Create a Django App for CRUD with Generic Class-Based Views

In this lab, you are going to continue to work for the same Django App for CRUD example application using generic class based views. Note that in order to continue to work for this lab, you must complete the previous lab (i.e., up to lab9) first.

A1. Updating the CRUD views

To perform CRUD operations against our database, let's create our own CRUD views. You'll be using the Django class based generic views to define your own views. Open `views.py` file and start by adding the following:

```
from django.shortcuts import render

# Create your views here.
from .models import Employee
from django.views.generic import ListView, DetailView
from django.views.generic.edit import CreateView, UpdateView, DeleteView

class EmployeeList(ListView):
    template_name='dbapp/employee_list.html'
    model = Employee

class EmployeeDetail(DetailView):
    model = Employee

class EmployeeCreate(CreateView):
    model = Employee
    fields = [
        'fname',
        'minit',
        'lname',
        'ssn',
        'bdate',
        'address',
        'sex',
        'salary',
        'super_ssn',
        'dno',
    ]
    #template_name_suffix = '_update_form'
    success_url = "/list"
```

```
class EmployeeUpdate(UpdateView):
    model = Employee
    fields = [
        'fname',
        'minit',
        'lname',
        'ssn',
        'bdate',
        'address',
        'sex',
        'salary',
        'super_ssn',
        'dno',
    ]
    #template_name_suffix = '_update_form'
    success_url = "/list"

class EmployeeDelete(DeleteView):
    model = Employee
    success_url = "/list"
```

A2. Adding the Templates

After defining the CRUD views, you next need to add the template for each of your view. Each view expects a template with a specific name or default in the templates folder of your application. We are going to use template under your app folder.

Inside your app folder, e.g., dbapp in this example, create a `templates/dbapp/` folder and start by adding the html files with the following content:

`employee_list.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Test MySQL</title>
</head>
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
  {% for employee in object_list %}
    <tr>
      <td>{{ employee.lname }}</td>
      <td>
        <a href="{% url "employee_detail" employee.ssn %}">details</a>
        <a href="{% url "employee_update" employee.ssn %}">edit</a>
```

```
        <a href="{% url "employee_delete" employee.ssn %}">delete</a>
      </td>
   </tr>
  {% endfor %}
  </tbody>
</table>
</html>
```

employee_detail.html

```
<h1>Employee Details</h1>
<p>First Name: {{object.fname}}</p>
<p>Middle Name: {{object.minit}}</p>
<p>Last Name: {{object.lname}}</p>
<p>SSN: {{object.ssn}}</p>
<p>Date of Birth: {{object.bdate}}</p>
<p>Address: {{object.address}}</p>
<p>Sex: {{object.sex}}</p>
<p>Salary: {{object.salary}}</p>
<p>Supervisor: {{object.super_ssn}}</p>
<p>Department: {{object.dno}}</p>
```

employee_form.html

```
<h1>Employee</h1>

<form  method="post">
{% csrf_token %}
{{ form.as_p }}
<input  type="submit"  value="Submit">
</form>
```

employee_confirm_delete.html

```
<h1>Employee Delete?</h1>

<form  method="post">
{% csrf_token %}
Are you sure you want to delete this employee?
<input  type="submit"  value="Submit"  />
</form>
```

A3. Adding the URLs

You need to add various URLs to the views you have defined. Open the `urls.py` file of your project and add the following URLs:

```
urlpatterns = [
    path('', views.testmysql),
    path('list/', views.EmployeeList.as_view(), name='employee_list'),
    path('list/<int:pk>', views.EmployeeDetail.as_view(),
name='employee_detail'),
    path('create', views.EmployeeCreate.as_view() ),
    path('update/<int:pk>', views.EmployeeUpdate.as_view(),
name='employee_update'),
    path('delete/<int:pk>', views.EmployeeDelete.as_view(),
name='employee_delete'),
]
```

A4. Run and Deploy the App

- Run your App on local machine

  ```
  $ python manage.py runserver
  ```

- Copy and paste the development url in your browser and see the results.

  http://127.0.0.1:8000/

- Deploy your app to GCP AppEngine

  ```
  $ gcloud app deploy
  ```

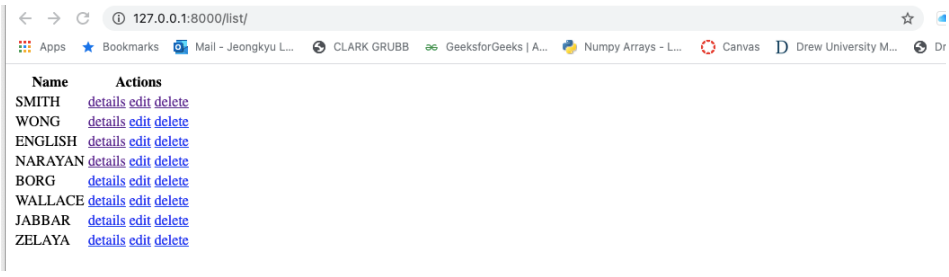
- In your browser, enter the following URL:

  `https://PROJECT_ID.REGION_ID.r.appspot.com/list/`

  - Replace the following:
  - **`PROJECT_ID:`** Your Google Cloud project ID
  - **`REGION_ID:`** A code that App Engine assigns to your app

- For company app,

  `https://PROJECT_ID.REGION_ID.r.appspot.com/list/`




dbtest — -zsh — 82×44
.
├── app.yaml
├── dbapp
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-38.pyc
│   │   ├── admin.cpython-38.pyc
│   │   ├── models.cpython-38.pyc
│   │   └── views.cpython-38.pyc
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   ├── __init__.py
│   │   └── __pycache__
│   │   └── __init__.cpython-38.pyc
│   ├── models.py
│   ├── templates
│   │   └── dbapp
│   │   ├── employee_confirm_delete.html
│   │   ├── employee_detail.html
│   │   ├── employee_form.html
│   │   └── employee_list.html
│   ├── tests.py
│   └── views.py
├── dbtest
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-38.pyc
│   │   ├── settings.cpython-38.pyc
│   │   ├── urls.cpython-38.pyc
│   │   └── wsgi.cpython-38.pyc
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── main.py
├── manage.py
├── noxfile_config.py
├── requirements.txt
└── templates
    └── home.html
9 directories, 31 files
(dbenv) jelee@Jeongkyus-Air dbtest %

B. Create a Django App for Raw SQL query

In this lab, you are going to create Django app to perform raw SQL queries. Django gives you two ways of performing raw SQL queries: you can use `Manager.raw()` to perform raw queries and **return model instances**, or you can **avoid the model layer** entirely and execute custom SQL directly.

First, you can practice raw SQL queries using Django shell.

B1. Performing raw queries using Django shell

- Start Django Shell

```
$ python manage.py shell
```

- Run the following command to access data using raw SQL

```
>>> from dbapp.models import Employee
>>> employee = Employee.objects.raw('SELECT ssn, lname FROM
employee')
>>> for p in employee:
...     print(p.ssn, p.lname)
...
123456789 SMITH
333445555 WONG
453453453 ENGLISH
666884444 NARAYAN
888665555 BORG
987654321 WALLACE
987987987 JABBAR
999887777 ZELAYA
>>>
```

B2. Updating Raw SQL views

To perform raw SQL queries against our database, let's create our own views. Open `views.py` file and start by editing `testmysql` function as following:

```
def testmysql(request):
     employee = Employee.objects.raw('SELECT ssn, lname FROM
employee')
     context = {
         'user_ssn': employee[0].ssn,
         'user_name': employee[0].lname,
     }
     return render(request, 'home.html', context)
```

Then, run and deploy the app like A4.

B3. Performing bypassing raw queries + fetchone using Django shell

Sometimes even `Manager.raw()` isn't quite enough: you might need to perform queries that don't map cleanly to models, or directly execute UPDATE, INSERT, or DELETE queries. In these cases, you can always access the database directly, routing around the model layer entirely.

The object `django.db.connection` represents the default database connection. To use the database connection, call `connection.cursor()` to get a cursor object. Then, call `cursor.execute(sql, [params])` to execute the SQL and `cursor.fetchone()` or `cursor.fetchall()` to return the resulting rows.

- Start Django Shell

```
$ python manage.py shell
```

- Run the following command to access data using raw SQL bypassing model

```
>>> from django.db import connection
>>> cursor = connection.cursor()
>>> cursor.execute('''select ssn, dname from employee, department
where dno = dnumber and ssn = "333445555"''')
  1
>>> employee = cursor.fetchone()
>>> print(employee[0], employee[1])
333445555 RESEARCH
>>>
>>> cursor.execute('''select ssn, dname from employee, department
where dno = dnumber''')
8
>>> employee = cursor.fetchall()
>>> for p in employee:
...     print(p)
...
('123456789', 'RESEARCH')
('333445555', 'RESEARCH')
('453453453', 'RESEARCH')
('666884444', 'RESEARCH')
('888665555', 'HEADQUARTERS')
('987654321', 'ADMINISTRATION')
('987987987', 'ADMINISTRATION')
('999887777', 'ADMINISTRATION')
>>>
```

B4. Updating Bypassing Raw SQL views

To perform bypassing raw SQL queries against our database, let's create our own views. Open `views.py` file and start by editing `testmysql` function as following:

```
from django.shortcuts import render

# Create your views here.
from .models import Employee
from django.views.generic import ListView, DetailView
from django.views.generic.edit import CreateView, UpdateView,
DeleteView
from django.db import connection

def testmysql(request):
    cursor = connection.cursor()
    cursor.execute('''select ssn, dname from employee, department
where dno = dnumber and ssn = "333445555"''')
    employee = cursor.fetchone()
    context = {
     'user_ssn': employee[0],
     'user_name': employee[1],
    }

    return render(request, 'home.html', context)
```

Then, run and deploy the app like A4.

B5. Updating views and template for Bypassing Raw SQL (Multiple rows)

- Updating `views.py` : add testmysql1 function in `views.py`

```
def testmysql1(request):
    ''' Using Raw SQL bypassing Model '''
    cursor = connection.cursor()
    cursor.execute('''select ssn, dname from employee, department
where dno = dnumber''')
    rows = cursor.fetchall()
    context = {
      "data" : rows
    }

    return render(request, 'home1.html', context)
```

- Updating template: create `home1.html` file under template folder

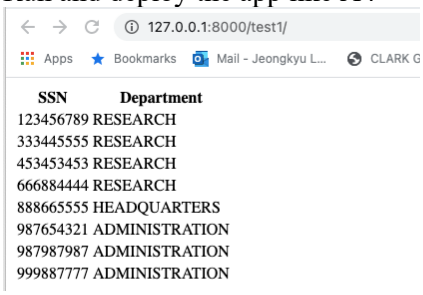
```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Test MySQL</title>
</head>
<table>
  <thead>
```

```
    <tr>
      <th>SSN</th>
      <th>Department</th>
    </tr>
  </thead>
  <tbody>
  {% for employee in data %}
    <tr>
        <td>{{ employee.0 }}</td>
        <td>{{ employee.1 }}</td>
    </tr>
   {% endfor %}
  </tbody>
</table>
</html>
```

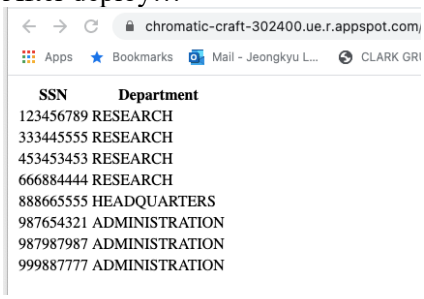
- Updating urls.py

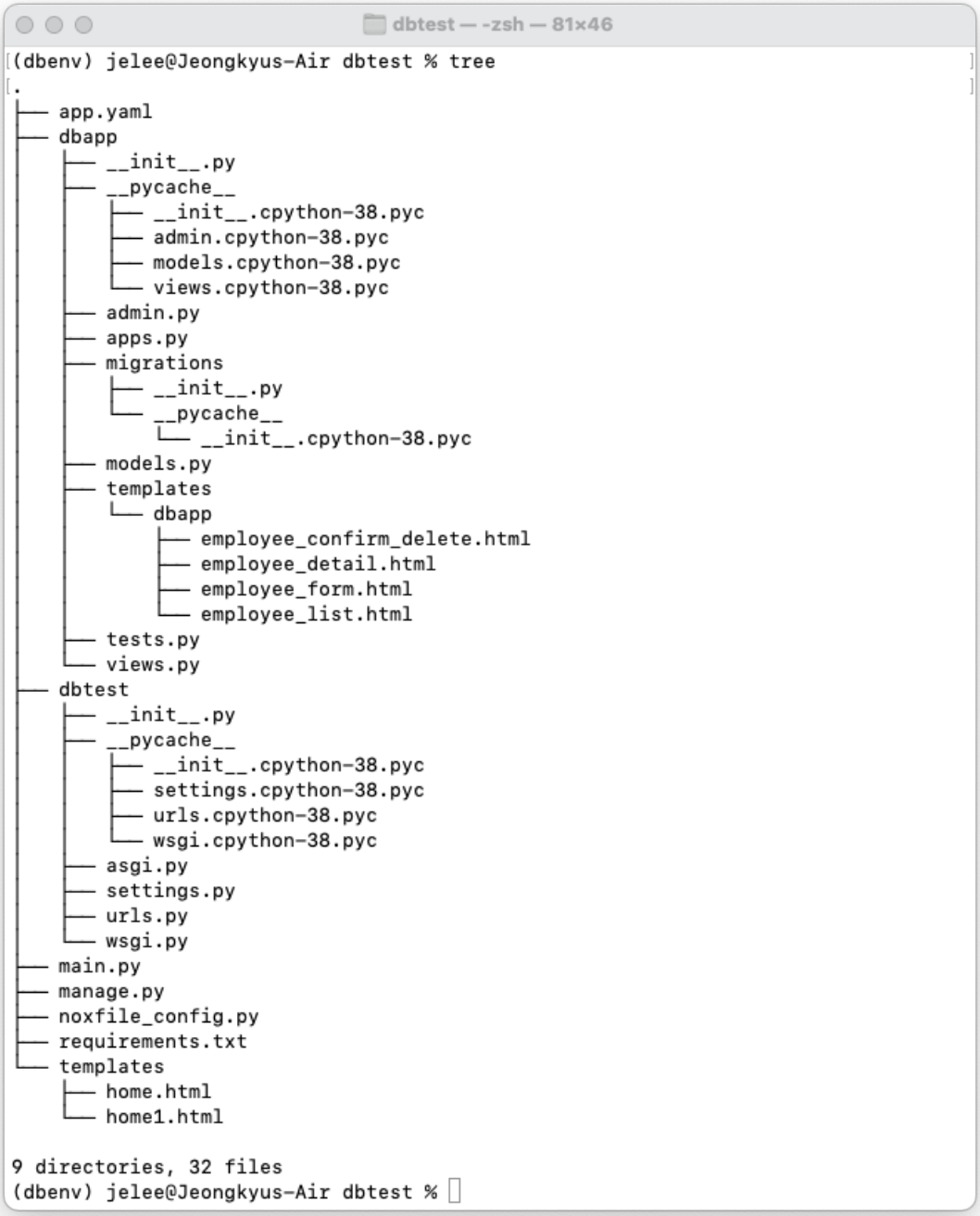
```
urlpatterns = [
    #path('admin/', admin.site.urls),
    path('', views.testmysql),
    path('test1/', views.testmysql1),
    path('list/', views.EmployeeList.as_view(),
name='employee_list'),
    path('list/<int:pk>', views.EmployeeDetail.as_view(),
name='employee_detail'),
    path('create', views.EmployeeCreate.as_view() ),
    path('update/<int:pk>', views.EmployeeUpdate.as_view(),
name='employee_update'),
    path('delete/<int:pk>', views.EmployeeDelete.as_view(),
name='employee_delete'),
]
```

- Run and deploy the app like A4



After deploy…




dbtest — -zsh — 81×46
(dbenv) jelee@Jeongkyus-Air dbtest % tree
.
├── app.yaml
├── dbapp
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-38.pyc
│   │   ├── admin.cpython-38.pyc
│   │   ├── models.cpython-38.pyc
│   │   └── views.cpython-38.pyc
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   ├── __init__.py
│   │   └── __pycache__
│   │   └── __init__.cpython-38.pyc
│   ├── models.py
│   ├── templates
│   │   └── dbapp
│   │   ├── employee_confirm_delete.html
│   │   ├── employee_detail.html
│   │   ├── employee_form.html
│   │   └── employee_list.html
│   ├── tests.py
│   └── views.py
├── dbtest
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-38.pyc
│   │   ├── settings.cpython-38.pyc
│   │   ├── urls.cpython-38.pyc
│   │   └── wsgi.cpython-38.pyc
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── main.py
├── manage.py
├── noxfile_config.py
├── requirements.txt
└── templates
    ├── home.html
    └── home1.html
9 directories, 32 files
(dbenv) jelee@Jeongkyus-Air dbtest %

Exercise: Create an Django + Python Application

First, create MAIL_ORDER database of Exercise 3.32 in the text book as follows:

```
Connect to MySQL instance
Connect to lab database

mysql> source cr_mailorder.sql
```

Write and test a Django application program that display an invoice for a given Order number.
(As a result in your browser, you need to display all this information which is related to the given Order Number [customer, Customer No, ZIP Code, Taken By, Received on, Shipped on, Part No, Part Name, Quantity, Price, Sum, and Total of the order]).

Lab 11: Query Processing and Optimizer

Use lab11 folder

- Change directory into **lab11** directory
- Save the file and extract it under Lab folder
- Remember to start by creating a sanjose database to use.

A. Query Processing and Optimizer in MySQL

A1. Create sanjose Database with mysql

For this exercise lab, create sanjose database using mysql. Then, create COMPANY database using `cr_company.sql` file. If needed, you can use `lab1.sql` file.

```
mysql> CREATE DATABASE sanjose;
mysql> USE sanjose;
mysql> source cr_company.sql
```

After, you create COMPANY database on sanjose, you can check the tables using desc command.

A2. Check Catalog Database

MySQL catalog database is "INFORMATION_SCHEMA", and there exist many tables that store meta data in the mysql. Let's check several important tables: STATISTICS, TABLES, TABLE_CONSTRAINTS . . . if needed, you can use `lab2.sql` file.

```
mysql> use information_schema;
mysql> show tables;
mysql> desc tables;
mysql> desc statistics;
mysql> select table_name, table_rows, avg_row_length, data_length
      -->    from information_schema.tables
      -->  where table_name = 'EMPLOYEE';
mysql> select * from information_schema.statistics;
```

A3. Collect Statistical Data using ANALYZE

It is very important to keep the up to date information about the database in order to get accurate decision of query optimizer. MySQL uses `ANAYZE` command to collect statistical data. You can use `lab3.sql` file if needed.

First, delete one row (i.e., ssn = '333445555'). Then, check the catalog database if the deletion is applied or not:

```
mysql> use sanjose;
mysql> delete from employee where ssn = “333445555”;
mysql> select table_name, table_rows, avg_row_length, data_length
      -->    from information_schema.tables
      -->   where table_name = 'EMPLOYEE’;
```

Any Problem?
Run analyze table on “EMPLOYEE” to update the information about EMPLOYEE table:

```
mysql> analyze table employee;
mysql> select table_name, table_rows, avg_row_length, data_length
      -->    from information_schema.tables
      -->   where table_name = 'EMPLOYEE’;
```

You can see the updated information about the table.

A4. Execution Plan

Execution plan is the output of query processor in MySQL, and it shows how MySQL accesses the database for a given query. In order to get the execution plan in MySQL, you can use EXPLAIN command as below: You can use lab4.sql file if needed.

- Simple Example: Check what is the plan of accessing employee table.
  ```
  mysql> explain select fname, lname from employee;
  mysql> explain select ssn from employee;
  ```

- Let’s create Primary Key of EMPLOYEE on SSN
  ```
  mysql> alter table employee add primary key (ssn);
  mysql> analyze table employee;
  ```

- Check the execution plan again for the same query. What is the difference?
  ```
  mysql> explain select fname, lname from employee;
  mysql> explain select ssn from employee;
  ```

A5. More Complex Query

You can check the execution plan for more complex query. For example,

```
select d.dname, sum(e.salary)
from employee e, department d
where e.dno = d.dnumber
and e.salary >= 3000
group by d.dname;
```

Find the execution plan for above query. If needed, you can use `lab5.sql` file.

```
mysql> explain select d.dname, sum(e.salary)
       -->  from employee e, department d
       -->  where e.dno = d.dnumber
       -->  and e.salary >= 3000
       -->  group by d.dname;
```

Now, you can add Primary Key of DEPARTMENT on DNUMBER. Then, check the execution plan again.

```
mysql> alter table department add primary key (dnumber);
```

What is the difference?

Then, Remove e.salary >=, then check the execution plan again. Any difference you find? If so, why?

A6. Hint

In MySQL, you can force the query processor to use specific plan using HINT. For example, you can ignore to use PRIMARY KEY as below:

```
mysql> explain select ssn from employee;
mysql> explain select ssn from employee ignore index (PRIMARY);
```

Also, you can use 'USE INDEX(TYPE)' to specify an index to be used. You can find more examples of hint from the lecture slide.

Exercise: Query Optimization using Explain and Hint

First, find (or create) 3 SQL queries on COMPANY database including:

- `Q1.sql: Simple query (accessing single table with where)`
- `Q2.sql: Subquery (accessing at least 2 tables)`
- `Q3.sql: Aggerate query including Group By`

Second, check execution plan for each query

Third, you can try different query execution plan using HINT, then compare with the results from Second.

Lab 12: Transaction Management

Use lab12 folder

- Change directory into **lab12** directory
- Save the file and extract it under Lab folder
- Remember to start by creating a sanjose database to use.

A. Transaction Management

A1. Create sanjose Database with mysql

For this exercise lab, create sanjose database using mysql (if needed). Then, create a table, named ACCT using `cr_db.sql` file.

```
mysql> CREATE DATABASE sanjose;
mysql> USE sanjose;
mysql> CREATE TABLE ACCT ( ID  VARCHAR(10), BAL INT);
mysql> INSERT INTO ACCT VALUES('A', 100);
mysql> INSERT INTO ACCT VALUES('B', 1000);
mysql> COMMIT;
```

A2. Transaction with Python programming

First open tx_lab.py file. Then, change the following information:

- localhost: if you are using MySQL on GCP, replace it by IP address. Otherwise, leave as it is.
- USER: database username
- PASS: database password

Run tx_lab.sql file

```
$ python3 tx_lab.sql
```

```
import mysql.connector

try:
    conn = mysql.connector.connect(host='localhost',
                                   database='sanjose',
                                   user='USER',
                                   password='PASS')

    conn.autocommit = False
    cursor = conn.cursor()
    # Deposit to account A
    sql_update_query = """Update acct set bal = bal + 100 where id =
'A'"""
    cursor.execute(sql_update_query)
```

```
        # Withdraw from account B
        sql_update_query = """Update acct set bal = bal - 100  where id =
'B'"""
        # Withdraw from account B with Error
        #sql_update_query = """Update acct set bal = bal - 100  where d =
'B'"""
        cursor.execute(sql_update_query)
        print ("Record Updated successfully ")

        #Commit your changes
        conn.commit()

    except mysql.connector.Error as error :
        print("Failed to update record to database rollback:
    {}".format(error))
        #reverting changes because of exception
        conn.rollback()
    finally:
        #closing database connection.
        if(conn.is_connected()):
             cursor.close()
             conn.close()
             print("connection is closed")
```

You can check ACCT table BAL if the python program run correctly.

Next, remove comment (#) of "withdraw from account B with error" (i.e., `sql_update_query = """Update acct set bal = bal - 100  where d = 'B'"""`) part, while adding comment (#) into "withdraw from account B".

Run tx_lab.py file again. Then, check what happened in ACCT table (BAL).

A3. Simulation of Multiple Transaction with two mysql sessions

In this lab, you are going to simulated the situation of multiple transactions accessing the same data. One transaction (tx1.sql) is updating ACCT.BAL, while the 2nd transaction (tx2.sql) is selecting ACCT.BAL. See what happens.
Open two mysql connections: one is for tx1 using `tx1.sql`, while 2nd is for tx2 using `tx2.sql`.

Sequence	Tx 1 (tx1.sql)	Tx 2 (tx2.sql)
0	`Connect mysql`	`Connect mysql`
1	`mysql> SET AUTOCOMMIT = 0`	`mysql> SET AUTOCOMMIT = 0`
2	`mysql> UPDATE ACCT SET BAL = 100 WHERE ID = 'A';`	
3		`mysql> SELECT * FROM ACCT;` (check BAL if you can see updated bal)
4	`mysql> COMMIT;`	
5		`mysql> SELECT * FROM ACCT;` (check BAL if you can see updated bal)
6		`mysql> COMMIT;`

7		mysql> SELECT * FROM ACCT; (check BAL if you can see updated bal)

Discuss why Tx 2 can't or can see the updated value.

B. Consistency with Isolation in MySQL

In this lab, you are going to demonstrate the following isolation levels:

- [B1] Read Uncommitted: iso1_tx1.sql and iso1_tx2.sql
- [B2] Read Committed: iso2_tx1.sql and iso2_tx2.sql
- [B3] Repeatable Read: iso3_tx1.sql and iso3_tx2.sql

B1. Read Uncommitted

In this lab, Tx 1 is updating the data, while Tx 2 which is "Read Uncommitted" isolation level is selecting the data.

Open two mysql connections: one is for tx1 using `iso1_tx1.sql`, while 2nd is for tx2 using `iso1_tx2.sql`.

Sequence	Tx 1 (iso1_tx1.sql)	Tx 2 (iso1_tx2.sql)
0	Connect mysql	Connect mysql
1		mysql> SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
2	mysql> SET AUTOCOMMIT = 0	
3	mysql> SELECT * FROM ACCT;	
4	mysql> UPDATE ACCT SET BAL = 200 WHERE ID = 'A';	
5	SELECT * FROM ACCT;	
6		mysql> SELECT * FROM ACCT;
7	mysql> ROLLBACK;	
8	mysql> SELECT * FROM ACCT;	
9		mysql> SELECT * FROM ACCT;

In Step 6, can Tx 2 see the updated BAL? Compare with a result in A3. Why?

B2. Read Committed

In this lab, Tx 1 is updating the data, while Tx 2 which is "Read Committed" isolation level is selecting the data.

Open two mysql connections: one is for tx1 using `iso2_tx1.sql`, while 2nd is for tx2 using `iso2_tx2.sql`.

Sequence	Tx 1 (iso2_tx1.sql)	Tx 2 (iso2_tx2.sql)
0	Connect mysql	Connect mysql
1		mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
2	mysql> SET AUTOCOMMIT = 0	
3	mysql> SELECT * FROM ACCT;	
4	mysql> UPDATE ACCT SET BAL = 200 WHERE ID = 'A';	
5	mysql> SELECT * FROM ACCT;	
6		mysql> SELECT * FROM ACCT;
7	mysql> COMMIT;	
8	mysql> SELECT * FROM ACCT;	
9		mysql> SELECT * FROM ACCT;

In Step 6, can Tx 2 see the updated BAL? Compare with a result in B1. Why? Also, can Tx 2 see the updated BAL after COMMIT? Why?

B3. Repeatable Read

In this lab, Tx 1 is updating the data, while Tx 2 which is "Repeatable Read" isolation level is reading the data if Tx1 changes affect it or not.

Open two mysql connections: one is for tx1 using `iso3_tx1.sql`, while 2nd is for tx2 using `iso3_tx2.sql`.

Sequence	Tx 1 (iso3_tx1.sql)	Tx 2 (iso3_tx2.sql)
0	Connect mysql	Connect mysql
1	mysql> SET AUTOCOMMIT = 0	mysql> SET AUTOCOMMIT = 0
2		mysql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
3		mysql> SELECT ID FROM ACCT WHERE BAL = 200;
4	mysql> SELECT * FROM ACCT;	
5	mysql> INSERT INTO ACCT VALUES ('C', 200);	
6	SELECT * FROM ACCT;	
7		mysql> SELECT ID FROM ACCT WHERE BAL = 200;
8	mysql> COMMIT;	
9	mysql> SELECT * FROM ACCT;	
10		mysql> SELECT ID FROM ACCT WHERE BAL = 200;

Check Step 7 and 10 of Tx 2. Before and after COMMIT in Tx 1, can Tx 2 read 'C' or not? Why?

Exercise: Transaction Management

After you complete Exercise Lab 12, answer the following questions:

1. In A2, after running tx_lab.py file again. what happened in ACCT table (BAL)?

2. In A3, why Tx 2 can't or can see the updated value?

4. In B1- Step 6, can Tx 2 see the updated BAL? Compare with a result in A3. Why?

5. In B2 - Step 6, can Tx 2 see the updated BAL? Compare with a result in B1. Why? Also, can Tx 2 see the updated BAL after COMMIT? Why?

6. In B3, check Step 7 and 10 of Tx 2. Before and after COMMIT in Tx 1, can Tx 2 read 'C' or not? Why?

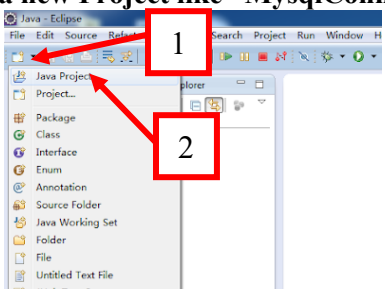
Then, submit your answers on Canvas.

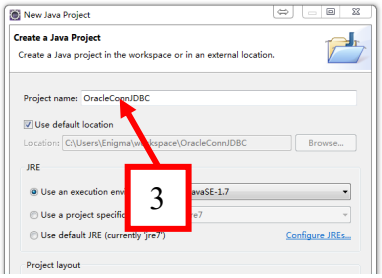
Appendix I: Database Programming

A. Java Database Connectivity (JDBC)

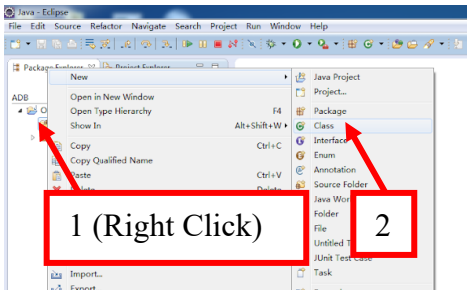
A1. Example 1 – Using Eclipse (IDE) to Get Data

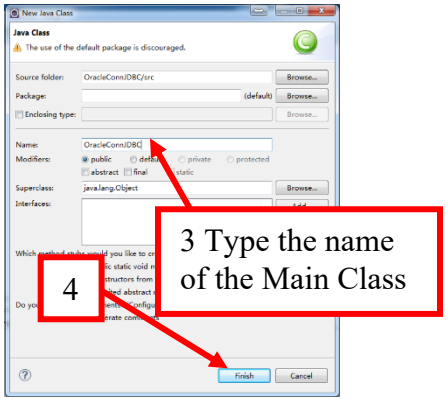
Step 1: Create a new Project like "MysqlConnJDBC".





Step 2: Create the Main Class

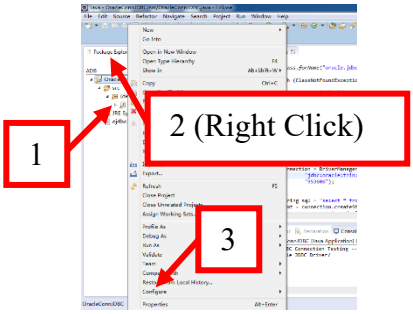


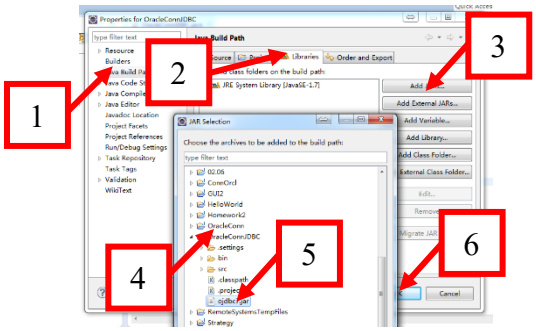


Step 3: Add the JDBC Library

a) Download the JDBC driver from Oracle Website.
https://dev.mysql.com/downloads/connector/j/5.1.html

b) After downloading, extract *.zip file. Then, copy the mysql-connector-java-5.1.43-bin.jar file into your project and Add to the Library.





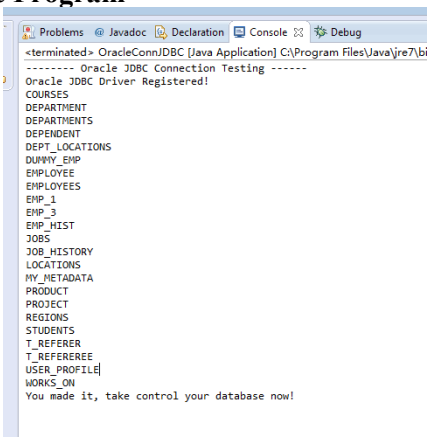
Step 4: Copy and Paste the following Code.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class MysqlConnJDBC {

    public static void main(String[] argv) {

        System.out.println("-------- MySQL JDBC Connection Testing ------");

        try {

            Class.forName("com.mysql.jdbc.Driver");

        } catch (ClassNotFoundException e) {

            System.out.println("Where is your MySQL JDBC Driver?");
            return;

        }

        System.out.println("MySQL JDBC Driver Registered!");

        Connection connection;
        Statement stmt;
        ResultSet rs;
        try {


            connection = DriverManager.getConnection(
                    "jdbc:mysql://11.11.11.11:3306/lab", "root",
                    "YOURPASSWORD");
            //Replace YOURPASSWORD" by your password for root


            String sql = "select * from employee";
            stmt = connection.createStatement();
            rs = stmt.executeQuery(sql);
            rs.next();
            while (rs.next()) {
                System.out.println(rs.getString(1));
            }


        } catch (Exception e) {

            System.out.println(e);
            return;
```

Replace 11.11.11.11 by IP address of your MySQL instance

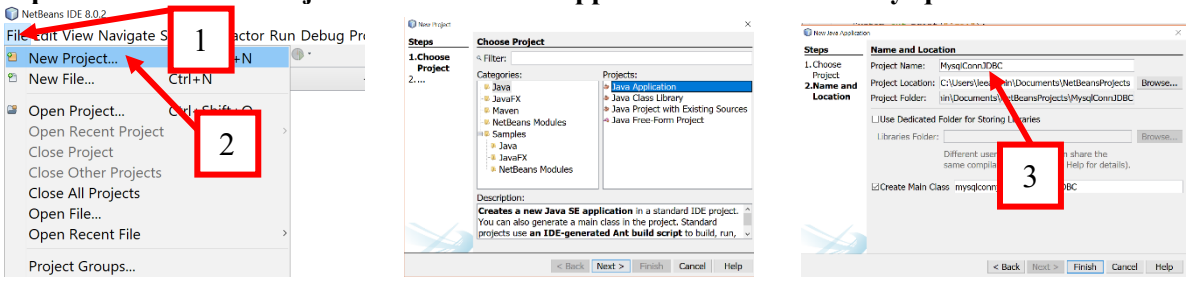
```
        }

        if (connection != null) {
            System.out.println("You made it, take control your database now!");
        } else {
            System.out.println("Failed to make connection!");
        }
    }
```

Step 5: Run the Program

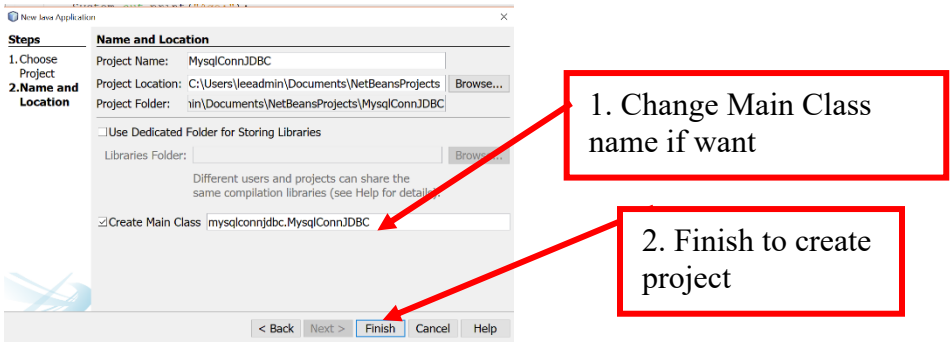


A2. Example 2 – Using Netbean (IDE) to Get Data

Step 1: Create a new Project with Java:Java Application and name: "MysqlConnJDBC".

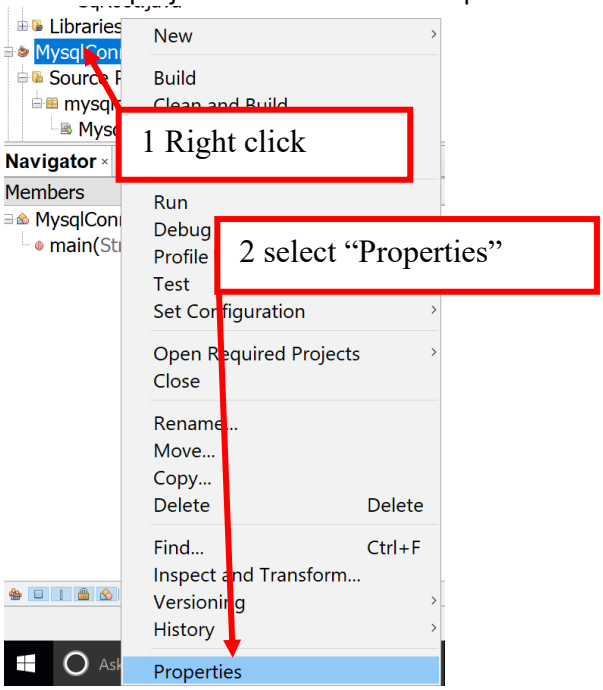


Step 2: Create the Main Class

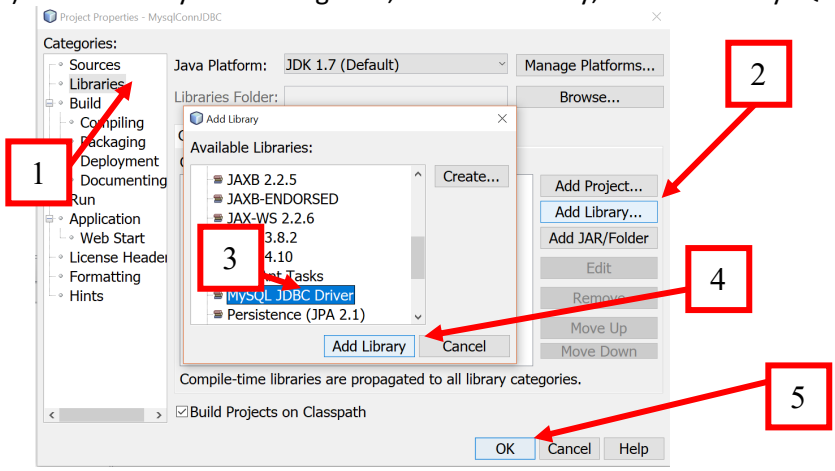


Step 3: Add the JDBC Driver

(a) Right click the project and then select “Properties”



(b) Select library from Categories, then add library, and select “MySQL JDBC Driver”.



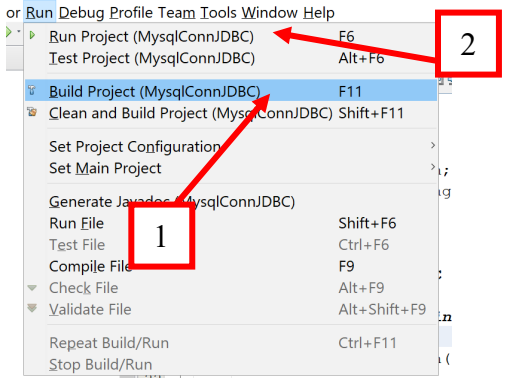
Step 4: Copy and Paste the following Code.

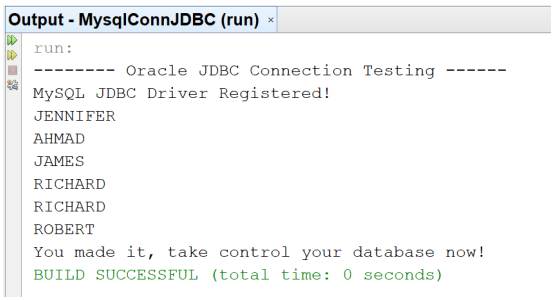
```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class MysqlConnJDBC {

    public static void main(String[] argv) {

        System.out.println("-------- MySQL JDBC Connection Testing ------");

        try {

            Class.forName("com.mysql.jdbc.Driver");
```

```
        } catch (ClassNotFoundException e) {

            System.out.println("Where is your MySQL JDBC Driver?");
            return;

        }

        System.out.println("MySQL JDBC Driver Registered!");

        Connection connection;
        Statement stmt;
        ResultSet rs;
        try {


            connection = DriverManager.getConnection(
                    "jdbc:mysql://11.11.11.11:3306/lab", "root",
                    "YOURPASSWORD");
            //Replace YOURPASSWORD" by your password for root

            String sql = "select * from employee";
            stmt = connection.createStatement();
            rs = stmt.executeQuery(sql);
            rs.next();
            while (rs.next()) {
                System.out.println(rs.getString(1));
            }


        } catch (Exception e) {

            System.out.println(e);
            return;

        }

        if (connection != null) {
            System.out.println("You made it, take control your database now!");
        } else {
            System.out.println("Failed to make connection!");
        }
    }
```

Replace 11.11.11.11 by IP address of your MySQL instance

}**Step 5: Build and Run the Program**



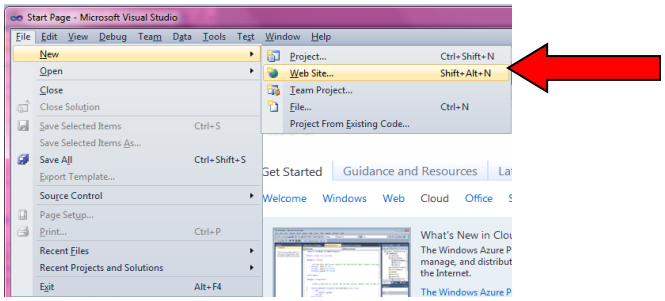

Output - MysqlConnJDBC (run)
run:
-------- Oracle JDBC Connection Testing ------
MySQL JDBC Driver Registered!
JENNIFER
AHMAD
JAMES
RICHARD
RICHARD
ROBERT
You made it, take control your database now!
BUILD SUCCESSFUL (total time: 0 seconds)

B. ASP.NET (C#) Microsoft Visual Studio

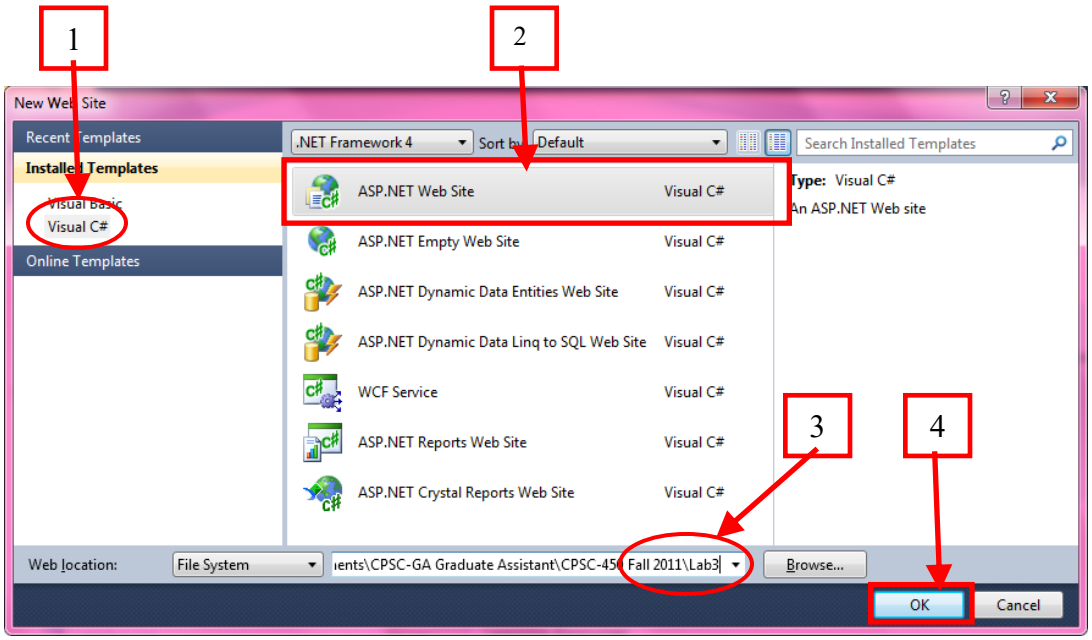
EXERCISE DESCRIPTION: This exercise we will create a simple ASP.NET program that displays, in a table, the first name and salary of an employee given his or her department number.

B1. PART 1: Open Visual Studio and Create a New Web page

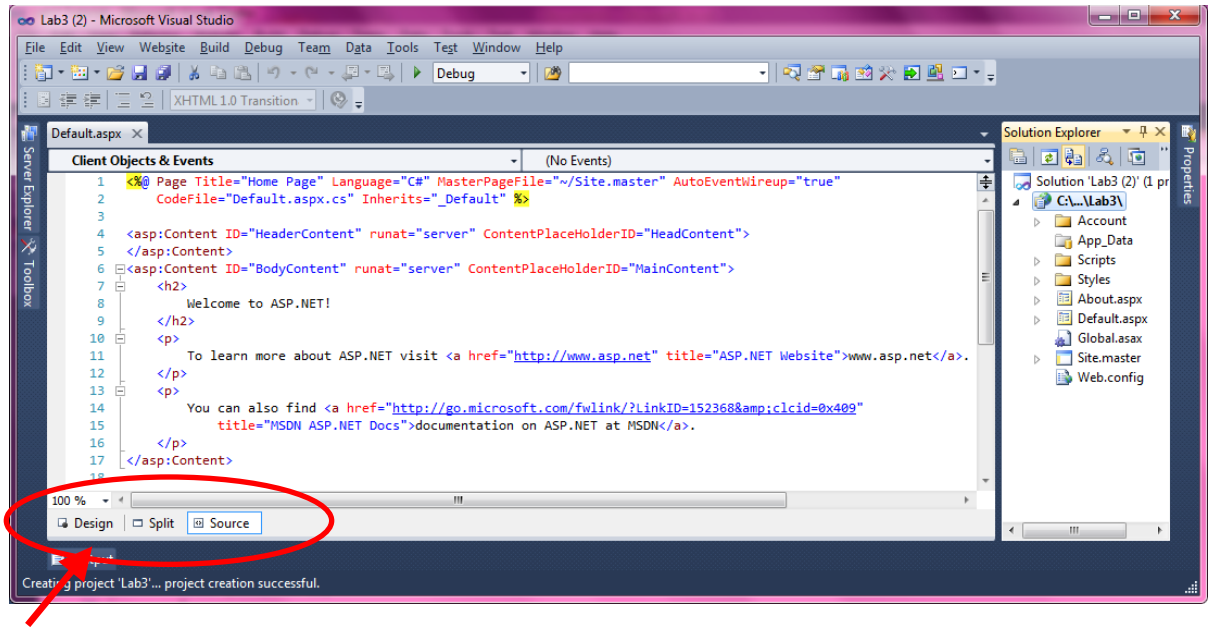
STEP 1: Open Microsoft Visual Studio.
STEP 2: From the Start Page, select File -> New -> Web Site…



STEP 3: In the New Web Site Window, Make sure Visual C# is selected and the choose ASP.NET Web Site. Select a location and name your project.

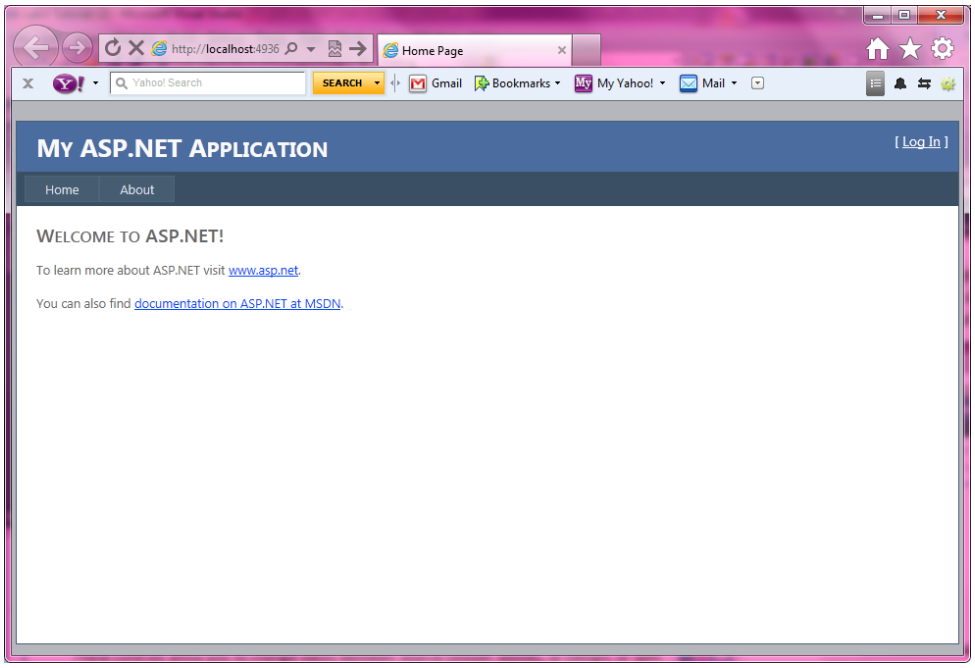


Visual Studio automatically creates the code for a default web page called Default.aspx . Visual Studio also creates several other folders and pages by default.



These controls allow you to change views between Source (shown above), or Design, or Split (combination of Source and Design)

If you run the program at this point, the web page looks like this:

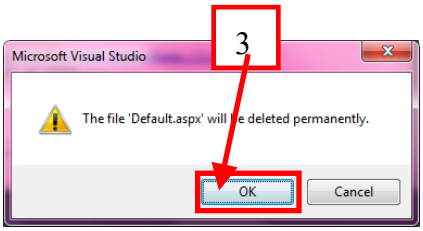


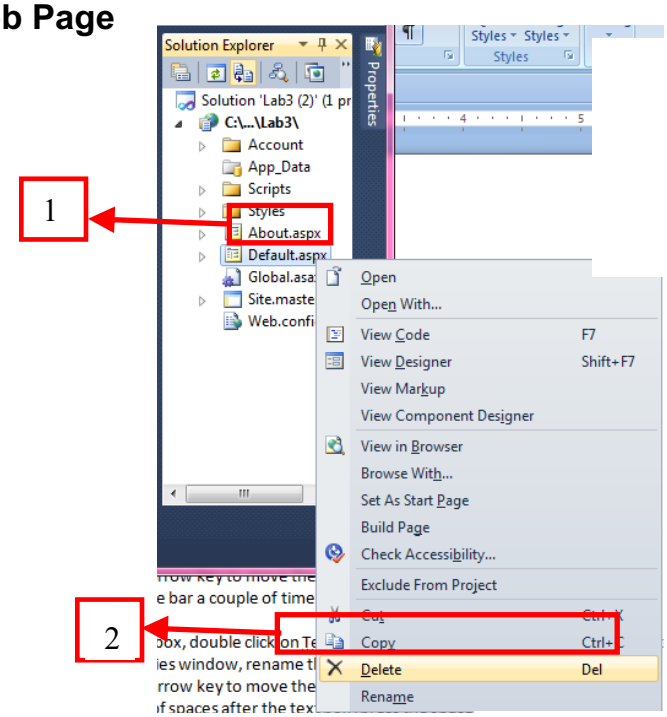
B2. PART 2: Create The Default Web Page

STEP 1: Delete the current Default.aspx

From Solution Explorer,
Right-click on Default.aspx
Select Delete

Click OK in the message box that appears

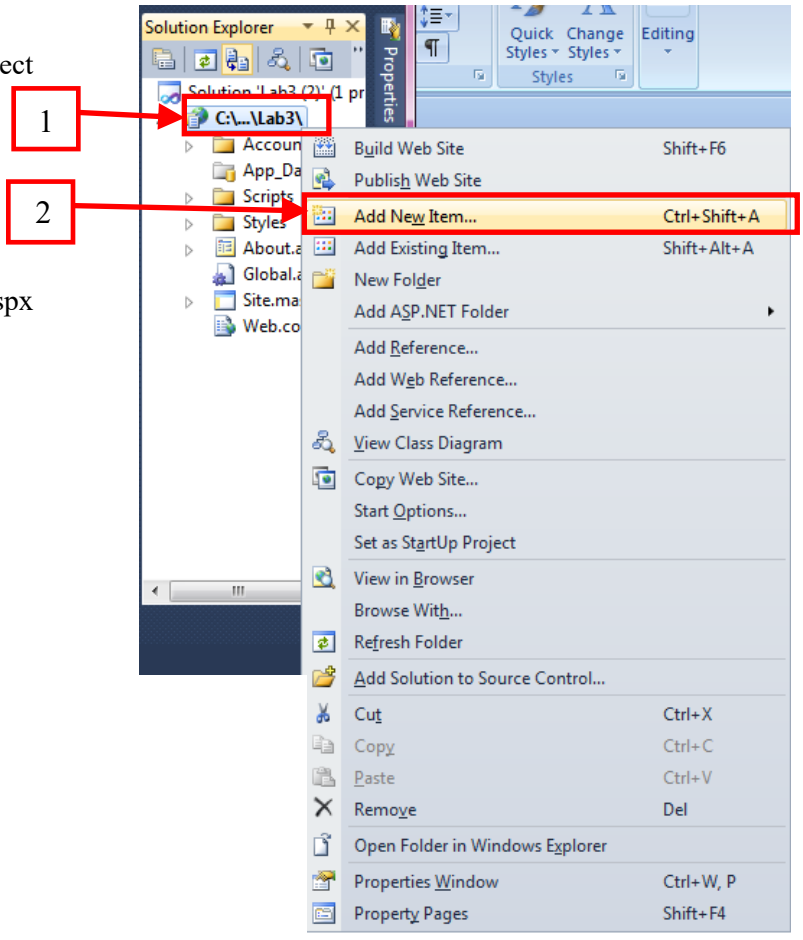


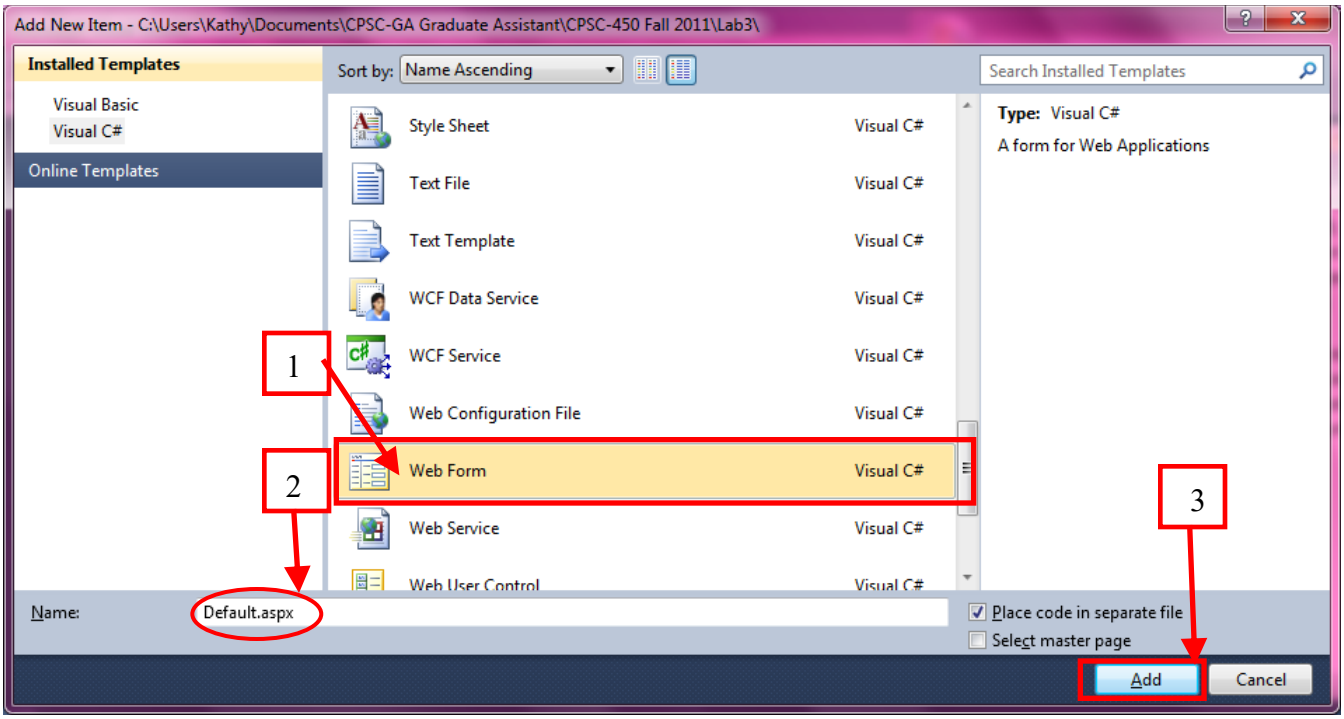


STEP 2: Create a new Web form

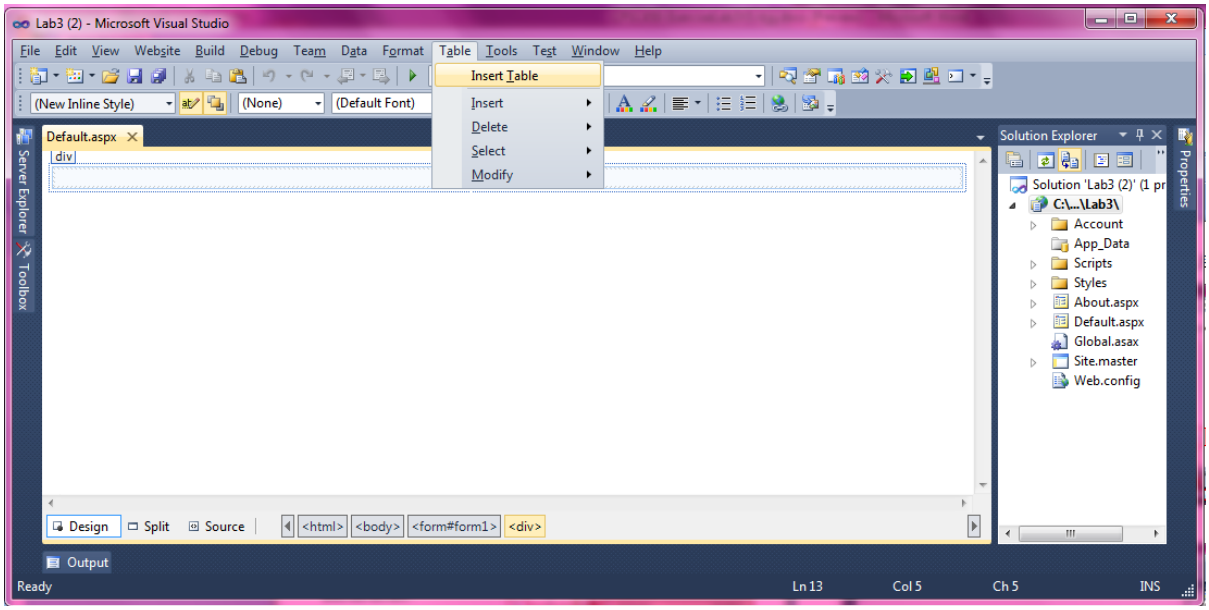
From Solution Explorer,
Right-click on the name of the project
Select Add New Item…

Choose Web Form
Name the new web form Default.aspx
Click Add
(see figure next page)



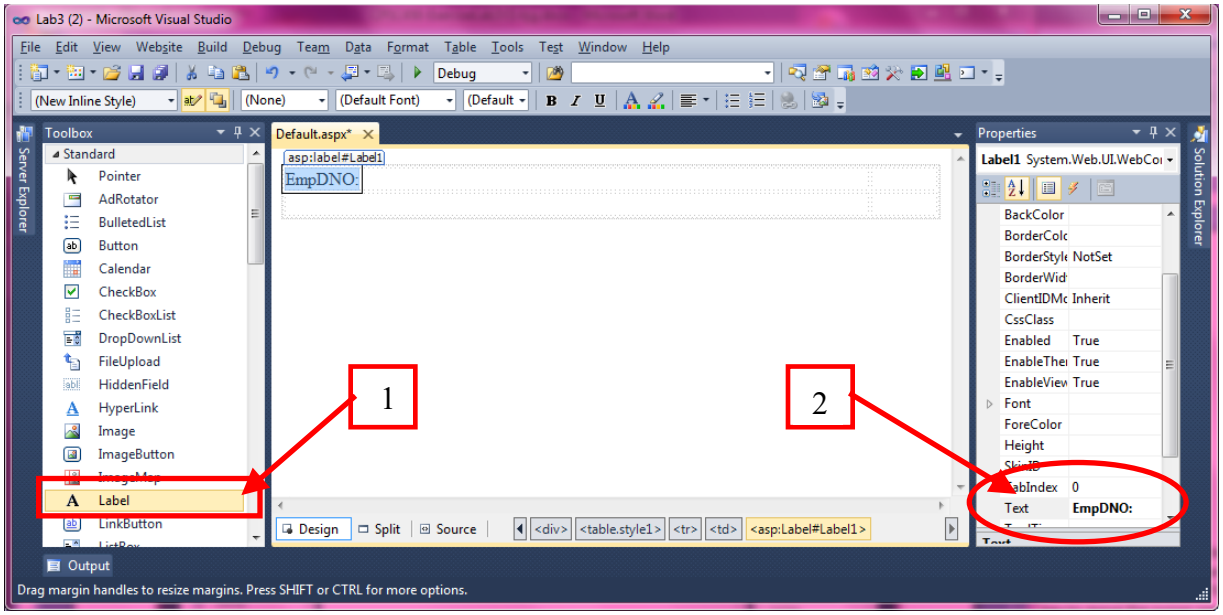


STEP 3: Switch to Design view (see figure on page 2).

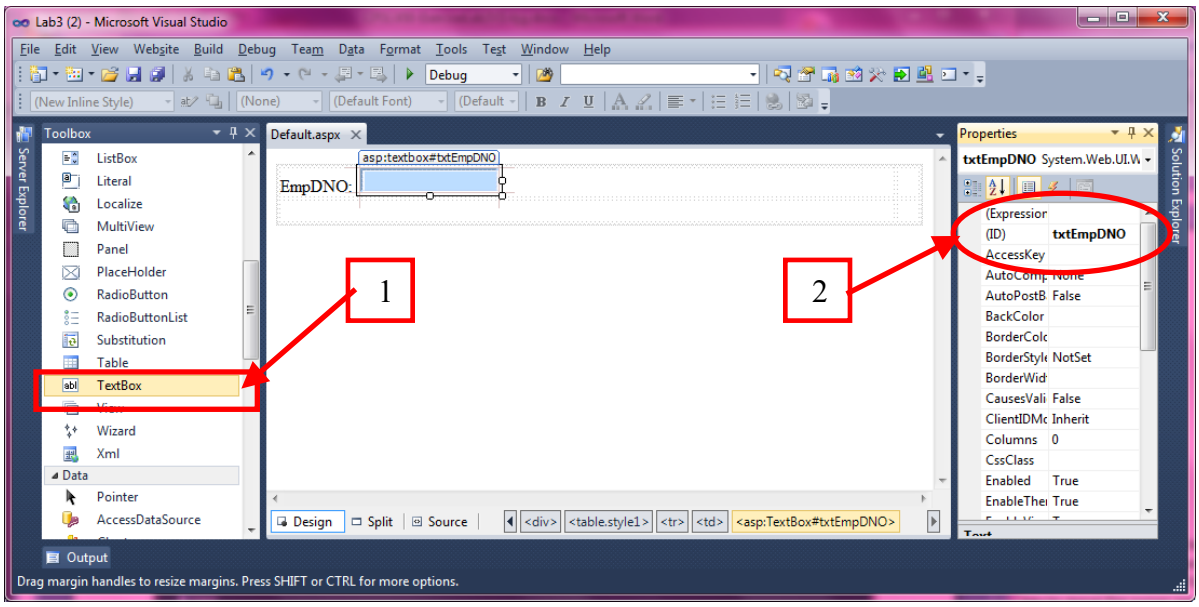


STEP 4: Add a table.
From the menu bar select Table -> Insert Table
Accept the default values and click OK

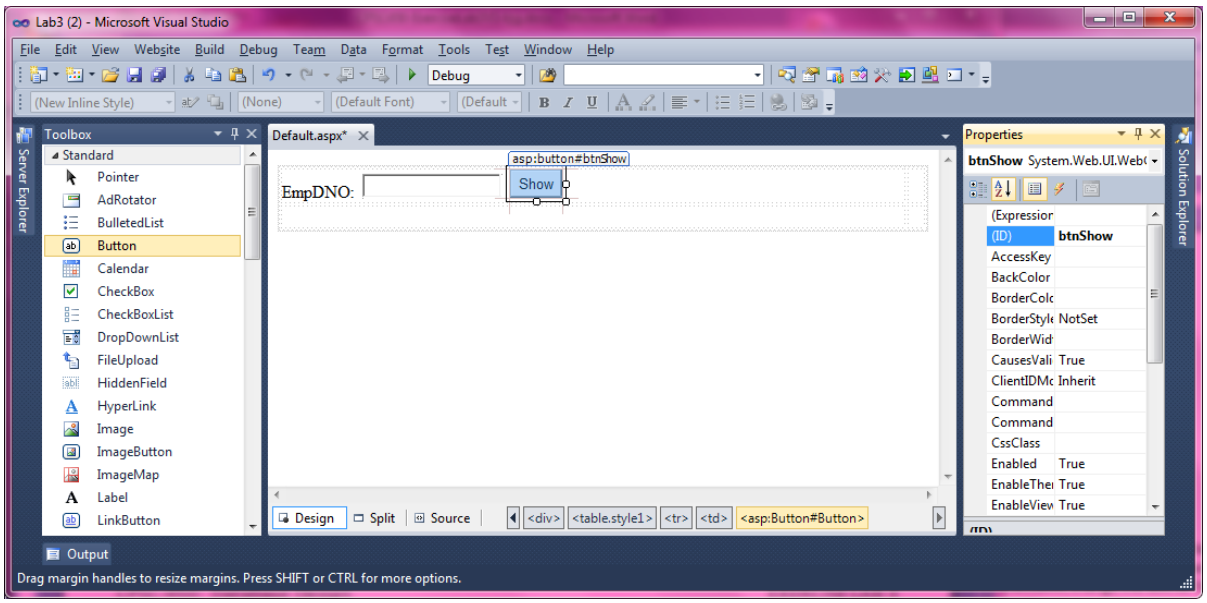
STEP 5: Add a label to the table
Make sure the cursor is in the first cell of the table.
From the Toolbox, double click on Label (a label will appear where the cursor was)
In the properties window, change the text to 'EmpDNO:'
Use the right arrow key to move the cursor past the text box
Press the space bar a couple of times to create some blank space after the label



STEP 6: Add a Text Box
From the Toolbox, double click on TextBox
In the properties window, change the ID of the textbox to 'txtEmpDNO'
Use the right arrow key to move the cursor past the text box
Add a couple of spaces after the text box (press the space bar a couple of times)



STEP 7: Add a button
From the Toolbox, double click on Button
In the properties window, change the ID of the button to 'btnShow'
Also in the properties window, change the text to 'Show'
Double click on the button
(this opens the Default.aspx.cs page and creates a method for the btnShow_Click event – we will write the code later)
Go back to Default.aspx

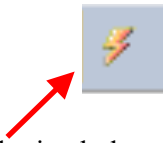


STEP 8: Add a GridView

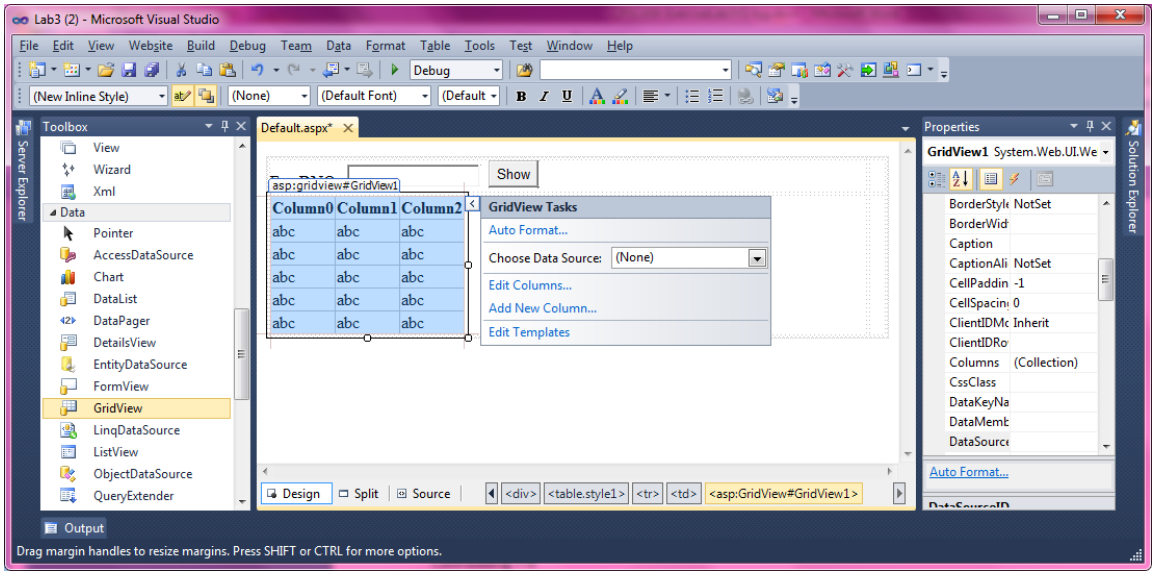
NOTE: The methods used to create the gridview (and get data from the database) are similar to methods used to create other lists such as DropDownList and DataList.

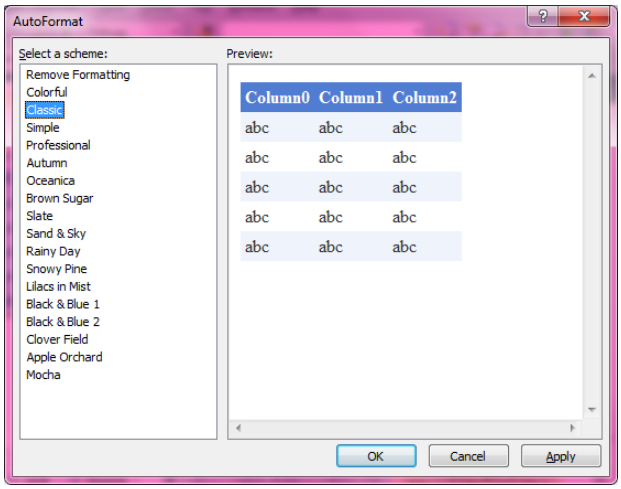
- Move the cursor to the next row
- From the Toolbox, double click on GridView
- Click on the arrow at the right of the gridview
- Select AutoFormat
- Choose Classic and click OK
- In the properties window, change the gridview ID to 'gv1'
- Many other properties can be set from the Properties window. Set the following:

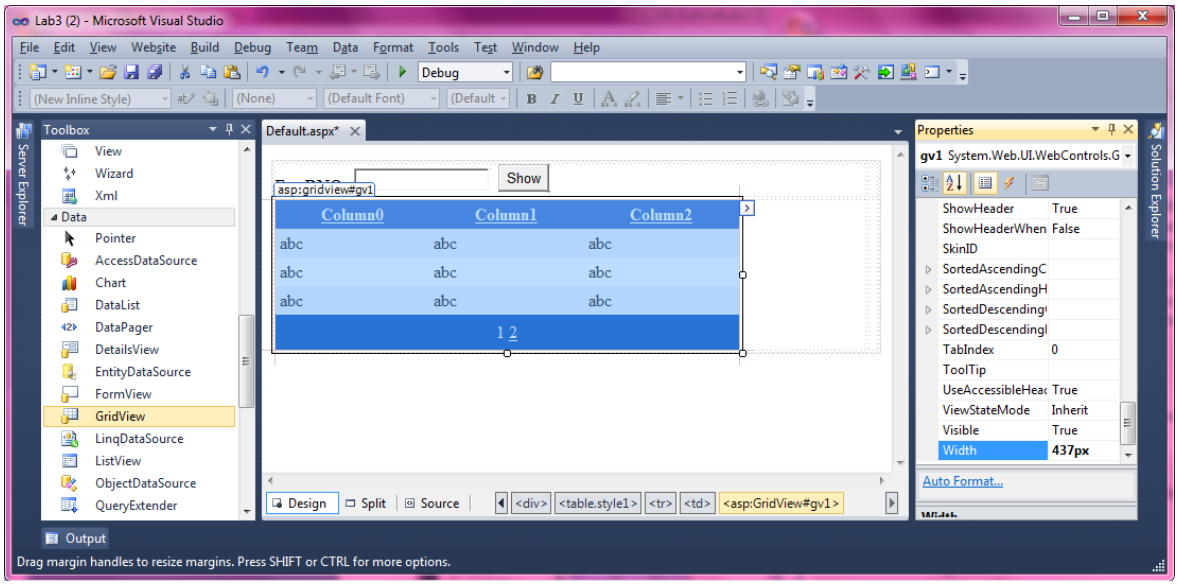
  AllowPaging = True
  AllowSorting = True
  CellPadding = 4
  PageSize = 3
  Width = 437px



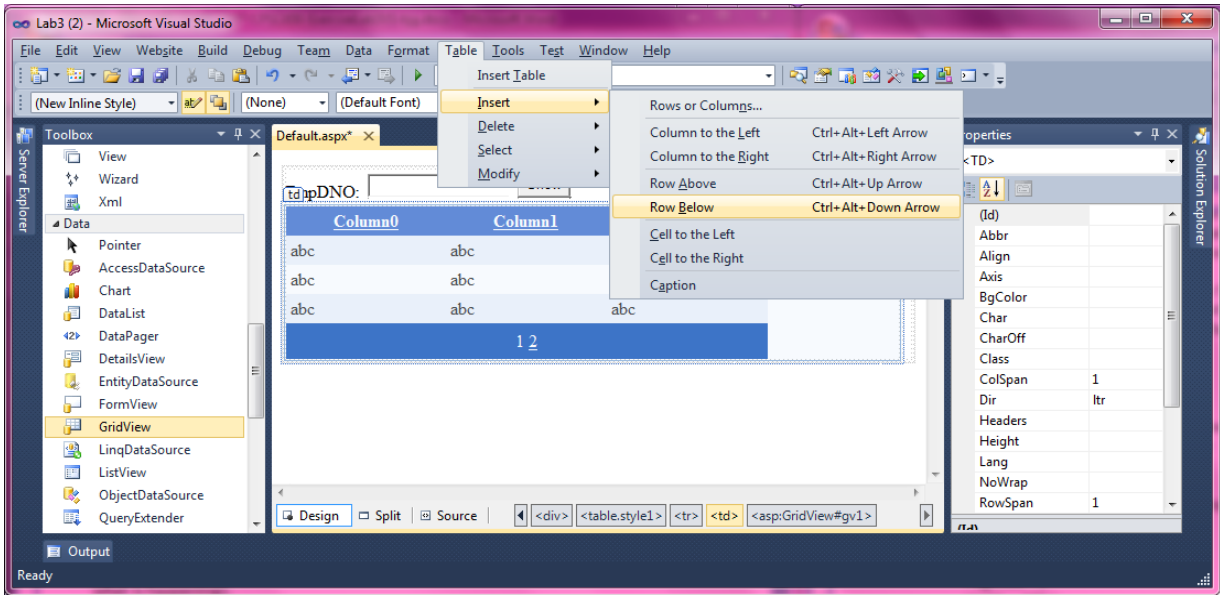
We can also set properties for various events. In the Properties window, click the lightning bolt symbol
Scroll down the list and double click on 'PageIndexChanging' (this opens the Default.aspx.cs page and creates a method for the gv1_PageIndexChanging – we will write the code later)
Go back to the events list, scroll down and double click on 'Sorting' to add a method to sort the gridview


Lab3 (2) - Microsoft Visual Studio
File Edit View Website Build Debug Team Data Format Table Tools Test Window Help
Debug
Toolbox
View
Wizard
Xml
Data
Pointer
AccessDataSource
Chart
DataList
DataPager
DetailsView
EntityDataSource
FormView
GridView
LinqDataSource
ListView
ObjectDataSource
QueryExtender
Default.aspx*
Show
asp:gridview#GridView1
Column0 Column1 Column2
abc abc abc
GridView Tasks
Auto Format...
Choose Data Source: (None)
Edit Columns...
Add New Column...
Edit Templates
Properties
GridView1 System.Web.UI.We
Auto Format...
Design Split Source
 <table.style1> <tr> <td> <asp:GridView#GridView1>
Output
Drag margin handles to resize margins. Press SHIFT or CTRL for more options.

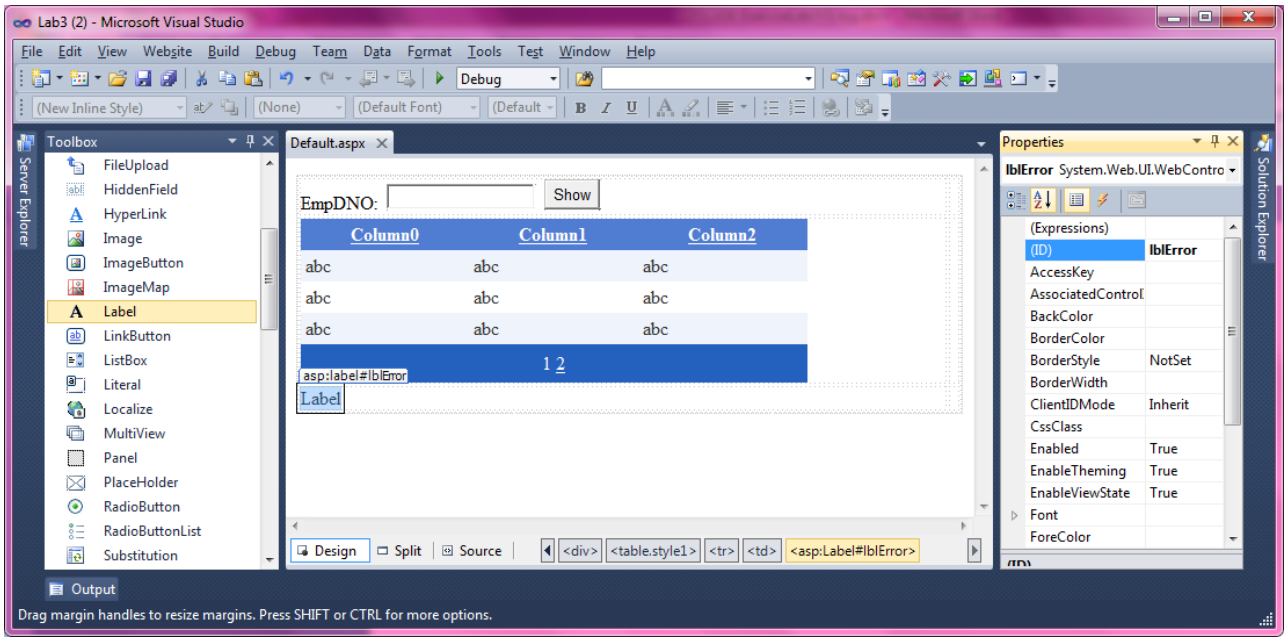

AutoFormat
Select a scheme:
Preview:
Remove Formatting
Colorful
Classic
Simple
Professional
Autumn
Oceanica
Brown Sugar
Slate
Sand & Sky
Rainy Day
Snowy Pine
Lilacs in Mist
Black & Blue 1
Black & Blue 2
Clover Field
Apple Orchard
Mocha
Column0 Column1 Column2
abc abc abc
OK
Cancel
Apply


Lab3 (2) - Microsoft Visual Studio
File Edit View Website Build Debug Team Data Format Tools Test Window Help
Debug
Toolbox
Default.aspx*
Show
asp:gridview#gv1
Column0 Column1 Column2
abc abc abc
1 2
Properties
gv1 System.Web.UI.WebControls.G
ShowHeader True
ShowHeaderWhen False
SkinID
TabIndex 0
ToolTip
ViewStateMode Inherit
Visible True
Width 437px
Auto Format...
Design Split Source
 <table.style1> <tr> <td> <asp:GridView#gv1>
Output
Drag margin handles to resize margins. Press SHIFT or CTRL for more options.

STEP 9: Add another row to the Design view. From the menu bar select Table -> Insert -> Row Below



STEP 10: Add a label that will display error messages (it is always good to include this to let the user know what is happening).
Make sure the cursor is in the last row (the one you just created)
From the Toolbox, double click on Label
In the properties window, change the label ID to 'lblError', and set the Visible property to 'False'



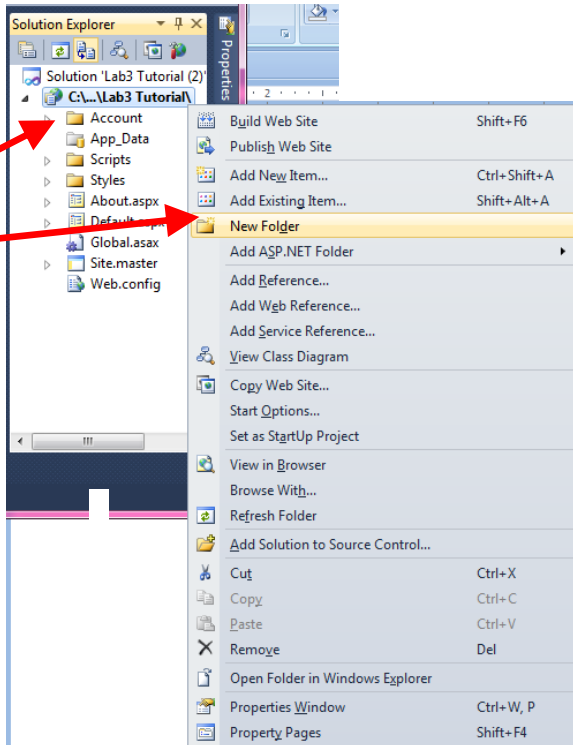
STEP 11: Change to Source View to see all the html code that Visual Studio automatically generates

B3. PART 3: Set Up The Classes and Interfaces

STEP 1: Classes and Interface s that you create should be placed in a separate folder called App_Code.

In the Solution Explorer Window, right-click on the name of the project, then choose New Folder…

Name the new folder App_Code

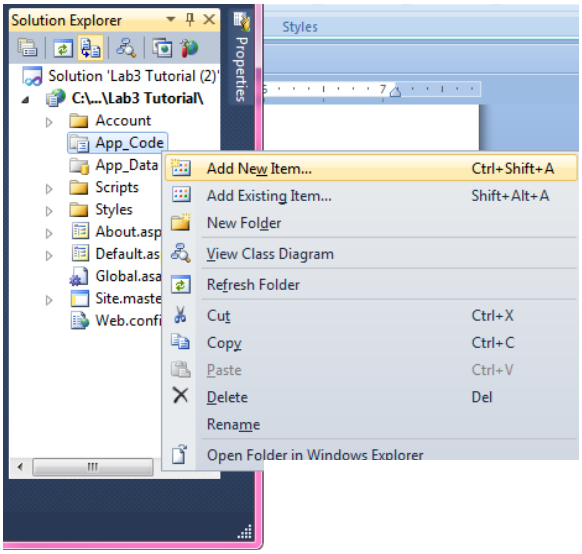


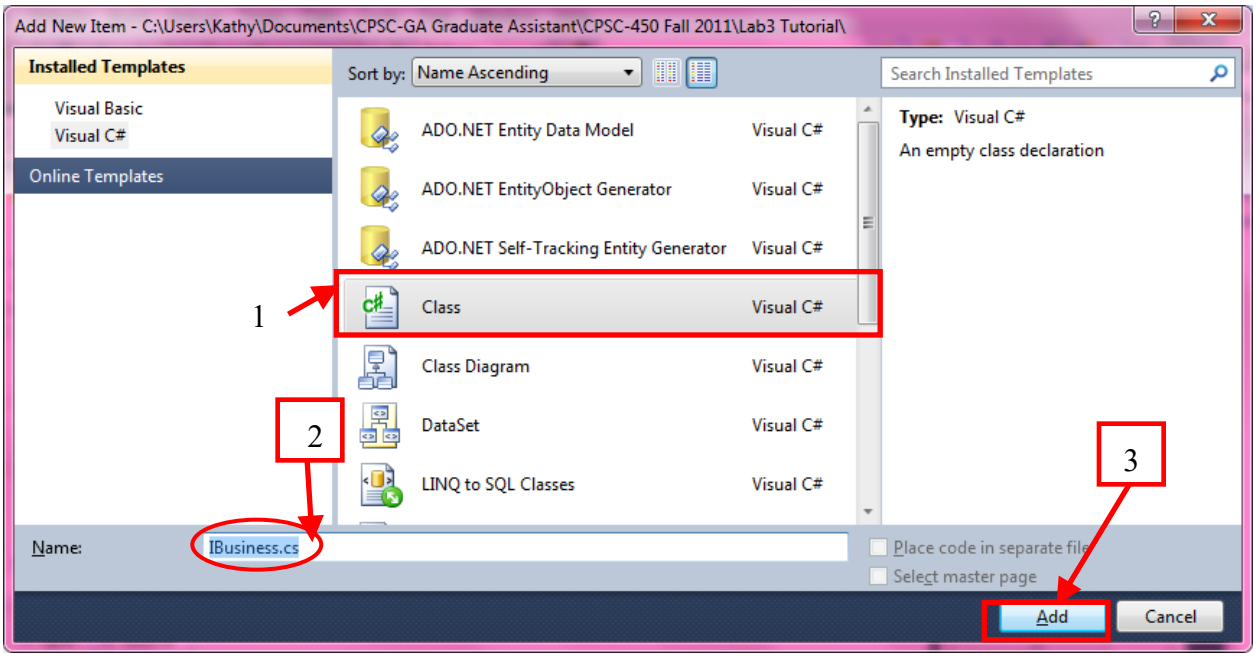
STEP 2: In the App_Code folder create the IBusiness, IData, and IDBAccess interfaces.

In the Solution Explorer window, Right-click on the App_Code folder And select Add New Item…

In the Add New Item window, select the Visual C# Class, name the new class IBusiness, then click Add (see figure next page)

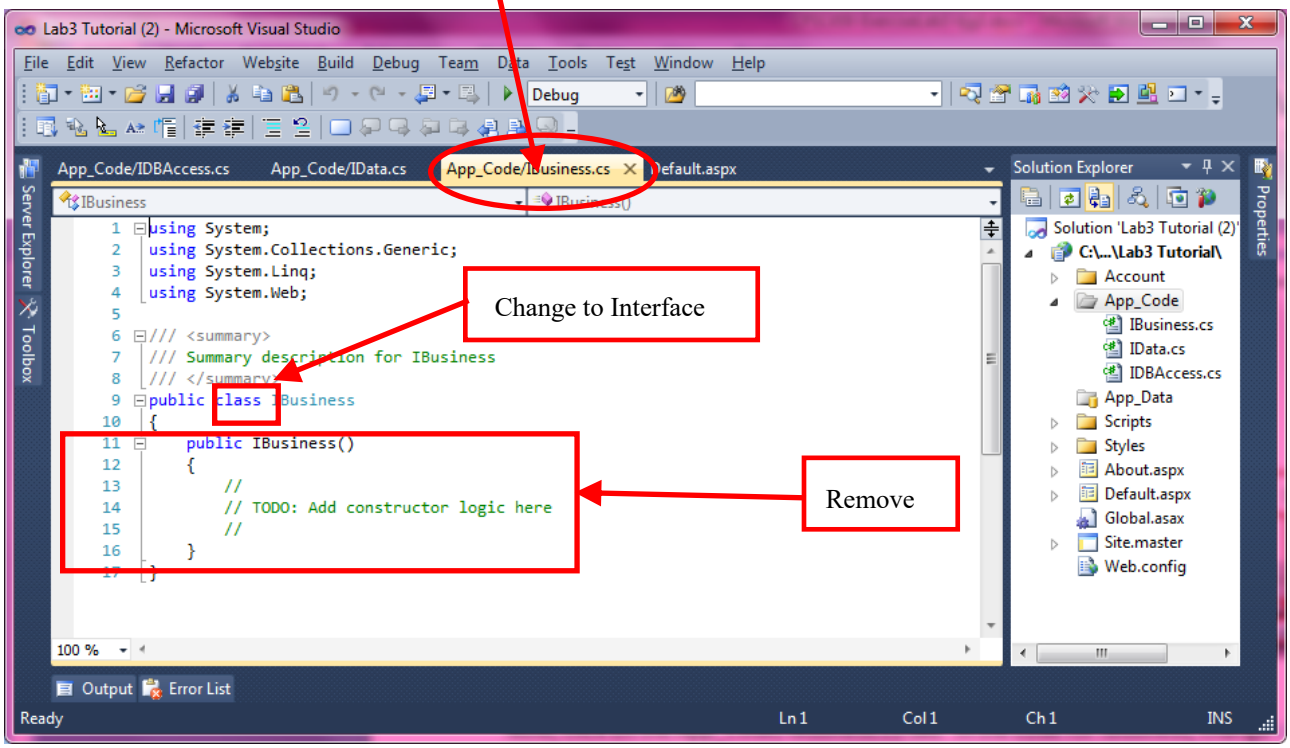
Repeat, for IData and IDBAccess.





STEP 3: Now, click on the App_Code/IBusiness.cs tab. In the code for IBusiness, change class to interface.

Remove the constructor.
Repeat, for IData and IDBAccess.



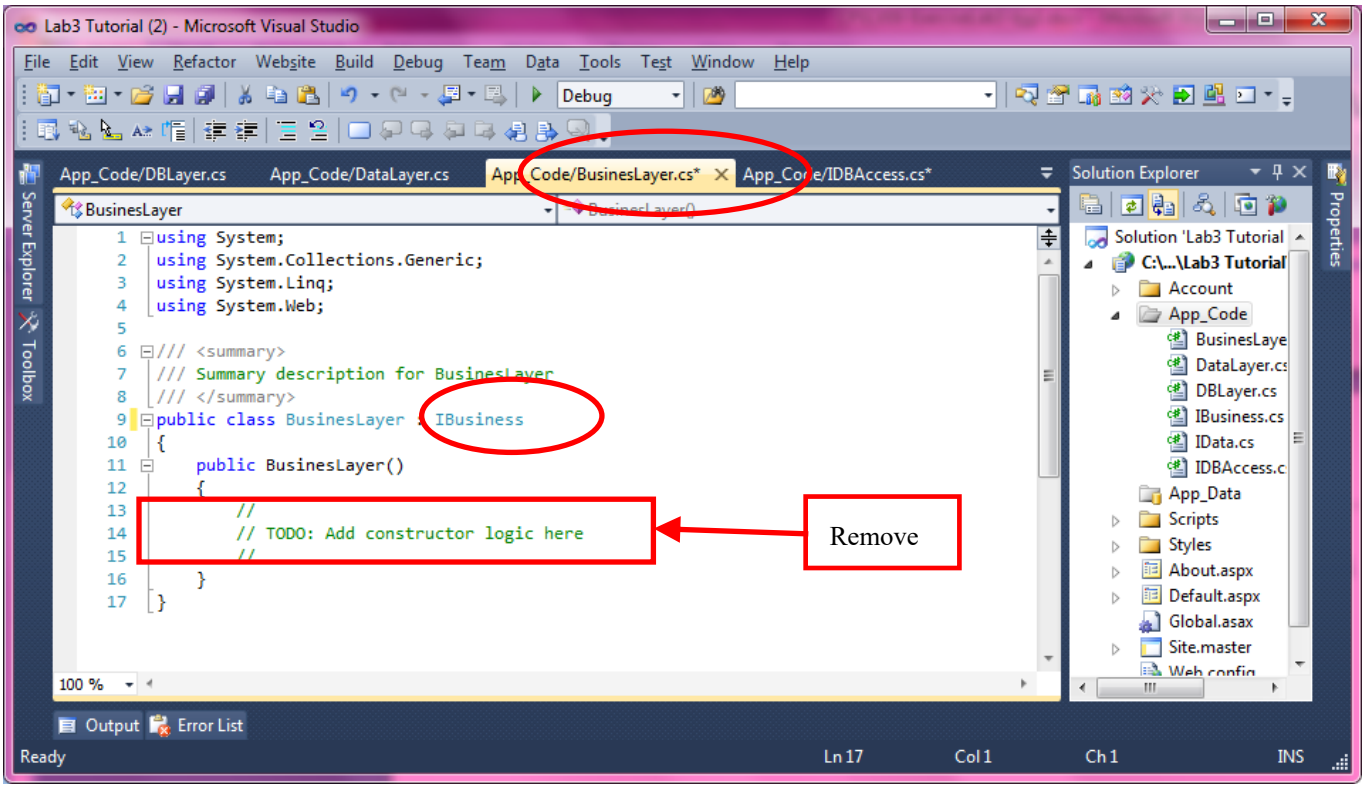
STEP 4: Add the classes that will implement the interfaces that you just created.
In the Solution Explorer window,
Right-click on the App_Code folder
Select Add New Item…
In the Add New Item window,
Select the Visual C# Class, name the new class BusinessLayer
Click Add

Repeat, for DataLayer and DBLayer.

STEP 5: The classes need to inherit their interfaces.

Click on the App_Code/BusinessLayer.cs tab.
Have this class inherit the interface IBusiness by typing " : IBusiness" as shown in the figure below.

Repeat for the DataLayer class and DBLayer class
For the DataLayer class type " : IData"
For the DBLayer class type " : IDBAccess"



B4. PART 4: Write Code for Methods added to Default.aspx.cs

STEP 1: Click on the Default.aspx.cs tab.

STEP 2: Copy and paste the following code inside the { } for the class: (you will replace whatever is already inside the { })

```
public static int Emp_DNO = 0;
IBusiness ib = new BusinessLayer();

protected void Page_Load(object sender, EventArgs e)
{
    txtEmpDNO.Focus();
}

protected void btnShow_Click(object sender, EventArgs e)
{
    try
    {
        Emp_DNO = int.Parse(txtEmpDNO.Text);
        DataSet ds = ib.GetDataSet(Emp_DNO);
        gv1.DataSource = ds.Tables[0];
        gv1.DataBind();
        Session["EMP_TABLE"] = ds.Tables[0];
    }
    catch (Exception ex)
    {
        lblError.Text = ex.Message;
    }
}

protected void gv1_PageIndexChanging(object sender, GridViewPageEventArgs
e)//Note: GridViewPageEventArgs
{
    gv1.PageIndex = e.NewPageIndex;
    DataSet ds = (DataSet)Session["EMP_TABLE"];
    gv1.DataSource = ds;
    gv1.DataBind();
    lblError.Text = "Current Page : " + (e.NewPageIndex + 1);
    lblError.Visible = true;
}

protected void gv1_Sorting(object sender, GridViewSortEventArgs e)
{
    DataTable dt = (DataTable)Session["EMP_TABLE"];

    // Create a DataView from the DataTable.
    DataView dv = new DataView(dt);

    // The DataView provides an easy way to sort. Simply set the
    // Sort property with the name of the field to sort by.
    int dir = 0;
    lblError.Text = "Sort by " + e.SortExpression;
    lblError.Visible = true;
```

```
    if (ViewState["SORTDIRECTION"] != null)
    {
        dir = (int)ViewState["SORTDIRECTION"];
        if (dir == 0) ViewState["SORTDIRECTION"] = 1;
        else ViewState["SORTDIRECTION"] = 0;
    }
    else
    {
        ViewState["SORTDIRECTION"] = 1;
        dir = 0;
    }
    if (dir == 1)
        dv.Sort = e.SortExpression + " ASC";
    else
        dv.Sort = e.SortExpression + " DESC";

    // Rebind the data source and specify that it should be sorted
    // by the field specified in the SortExpression property.
    gv1.DataSource = dv;
    gv1.DataBind();
}
```

STEP 3: There will be some words that Visual Studio underlines with a red wavy line.
The first underlined word is 'DataSet'. Move your cursor over DataSet – Visual Studio will display the following message

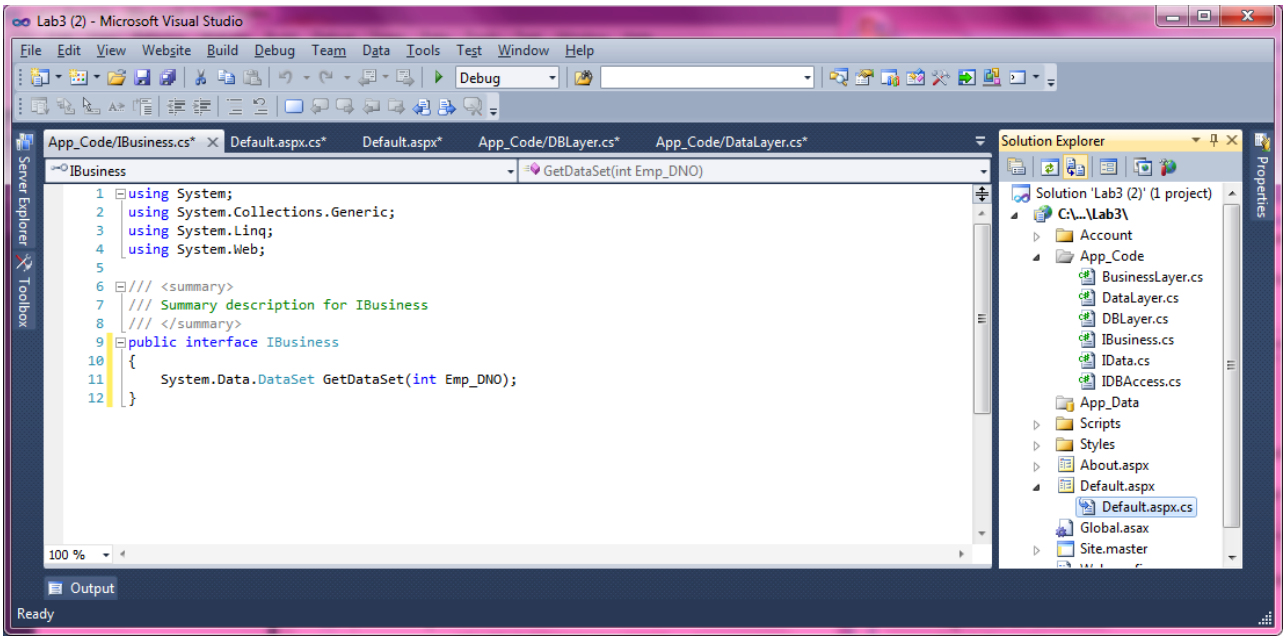
To correct this error:
right click on 'DataSet'
Select Resolve
Select using System.Data (the using statement is added at the top of the code)

The other underlined word is 'GetDataSet'. Move your cursor over the word to see the following message:

We get this error because GetDataSet is a method in the BusinessLayer class and we have not defined this method yet. To correct this issue:

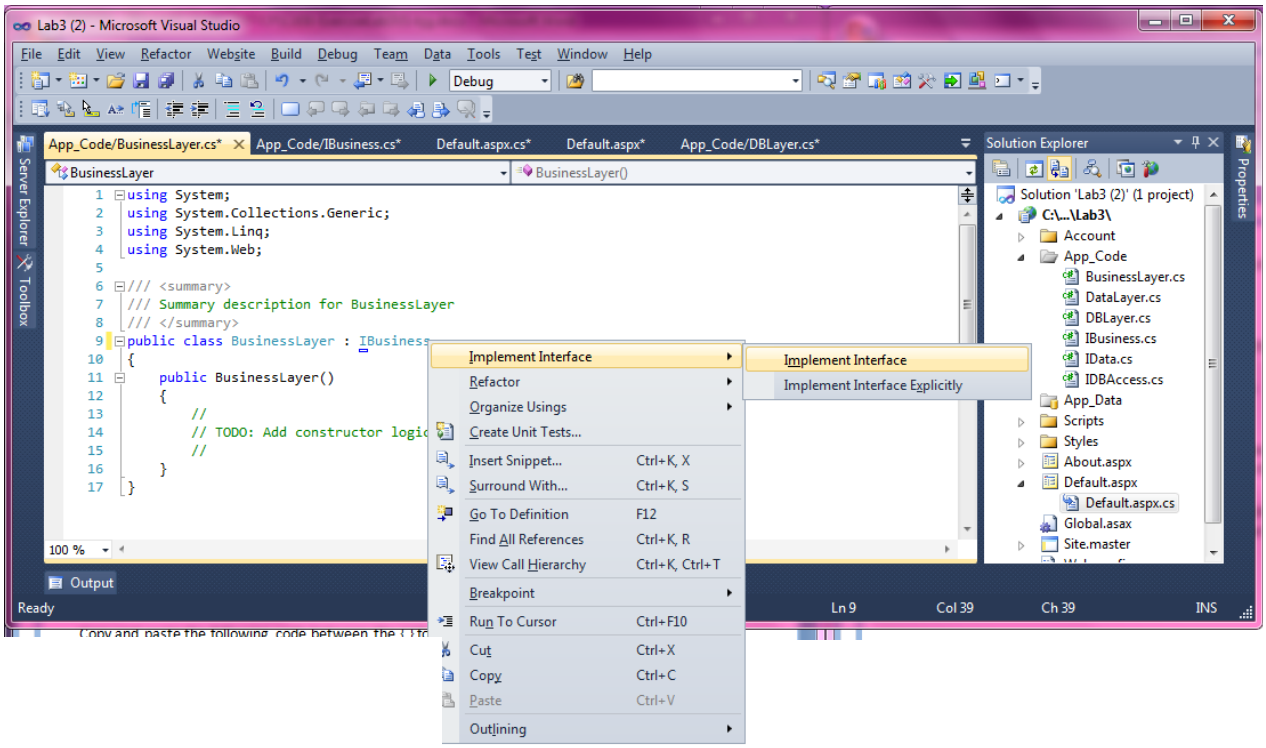
Right click on GetDataSet
Select Generate
Select Method Stub
(the method is added to the IBusiness interface – click on the App_Code/IBusiness.cs tab to see)

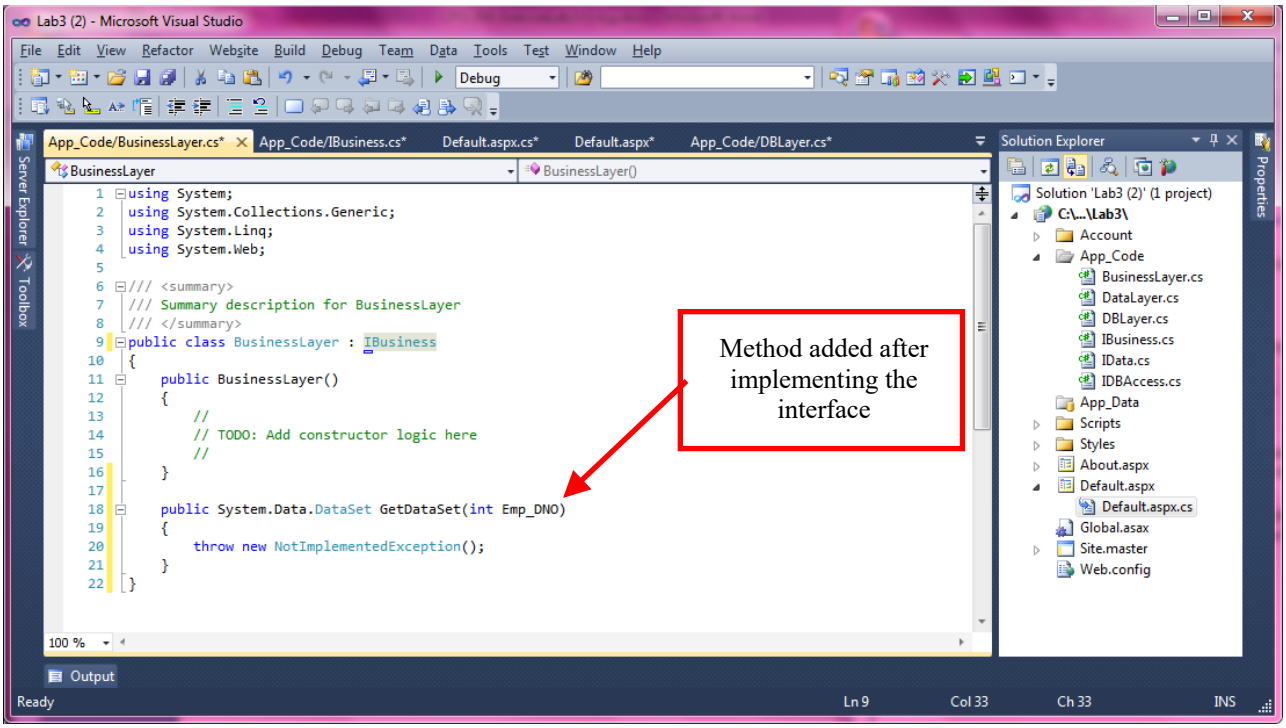
See figure next page



B5. PART 5: Write Code for The Classes and Interfaces

STEP 1: Code for the BusinessLayer class.
Click on the App_Code/BusinessLayer.cs tab
Right-click on IBusiness
Select Implement Interface
Select Implement Interface (all methods listed in the interface will be added to the class)





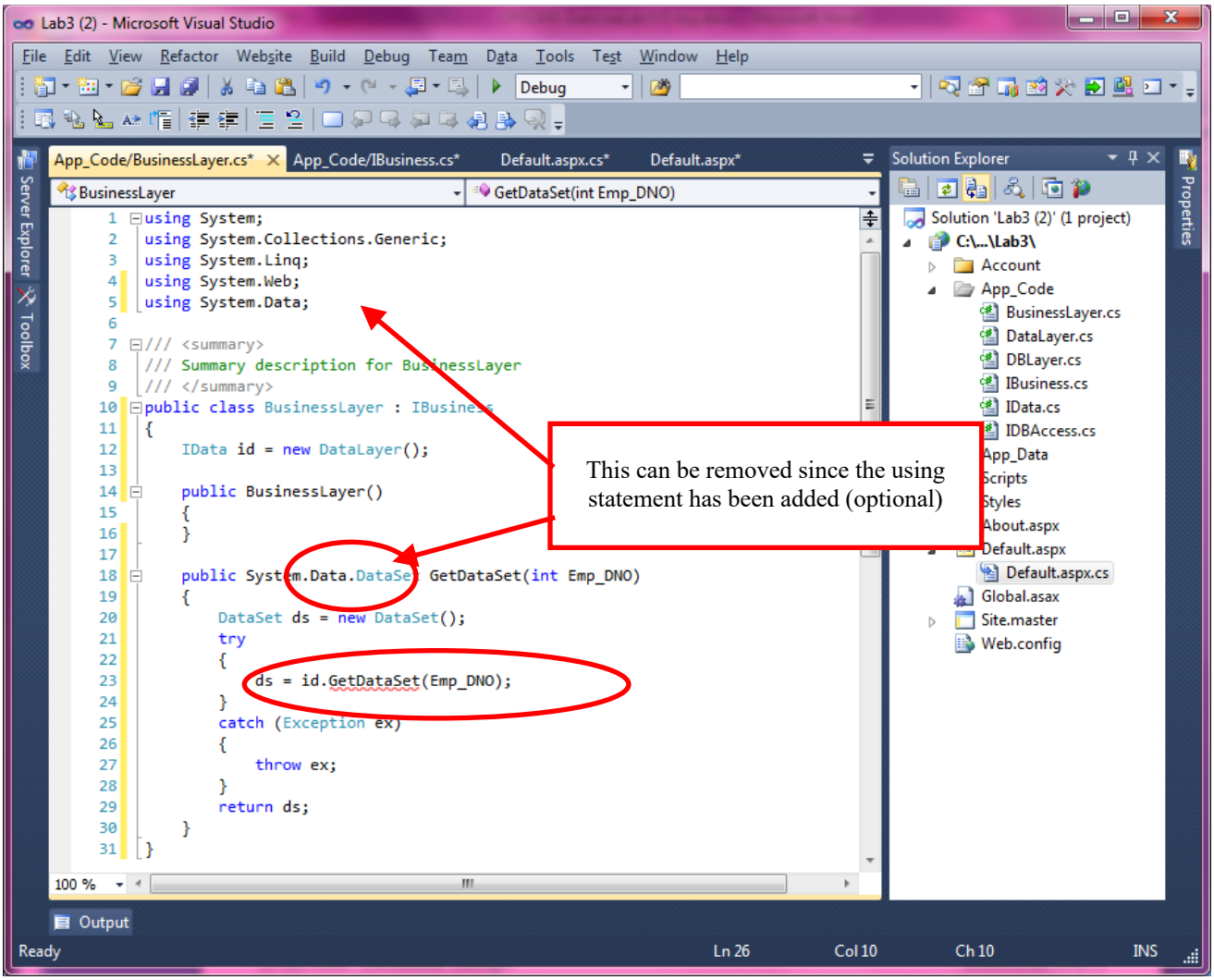
Copy and paste the following code between the { } for the GetDataSet method

```
DataSet ds = new DataSet();
try
{
    ds = id.GetDataSet(Emp_DNO);
}
catch (Exception ex)
{
    throw ex;
}
return ds;
```

Resolve the underlined word 'DataSet' (see previous)
There is another underlined word ('id') - type the following code above the constructor:

```
IData id = new DataLayer();
```

Note, once you add the above line of code you will see that another red wavy line appears (see the figure on the next page)



We get this error because GetDataSet is a method in the DataLayer class and we have not defined this method yet. To correct this issue:

Right click on GetDataSet
Select Generate
Select Method Stub
(the method is added to the IData interface – click on the App_Code/IData.cs tab to see)

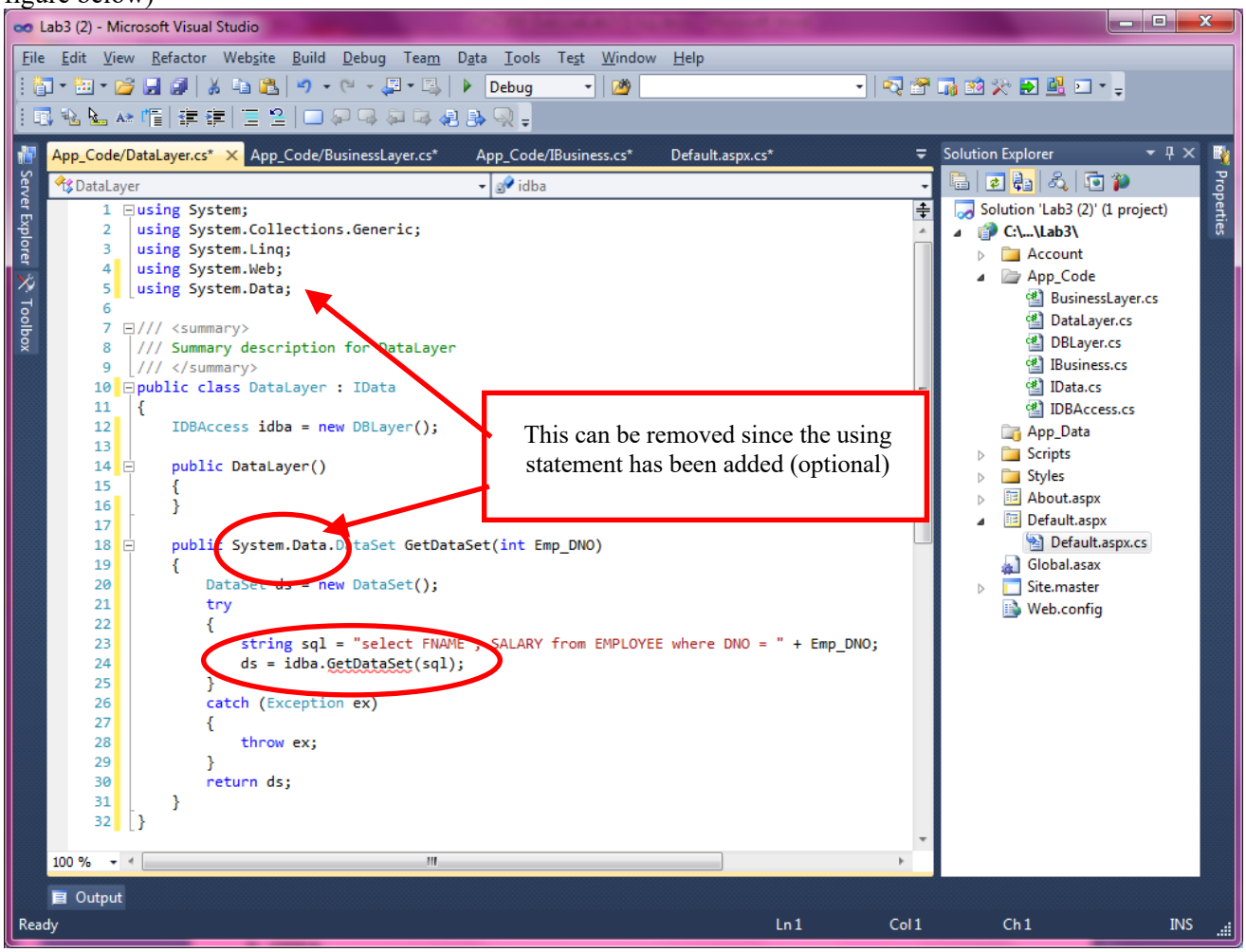
STEP 2: Code for the DataLayer class. This is the layer where we will construct the SQL statements.
Click on the App_Code/DataLayer.cs tab
Right-click on IData
Select Implement Interface
Select Implement Interface (all methods listed in the interface will be added to the class – refer to figures in step 1)
Copy and paste the following code between the { } for the GetDataSet method

```
        DataSet ds = new DataSet();
        try
        {
            string sql = "select FNAME , SALARY from EMPLOYEE where DNO = " +
Emp_DNO;
            ds = idba.GetDataSet(sql);
        }
        catch (Exception ex)
        {
            throw ex;
        }
        return ds;
```

Resolve the underlined word 'DataSet' (see previous)
There is another underlined word ('idba') - type the following code above the constructor:

```
IDBAccess idba = new DBLayer();
```

Note, once you add the above line of code you will see that another red wavy line appears (see the figure below)

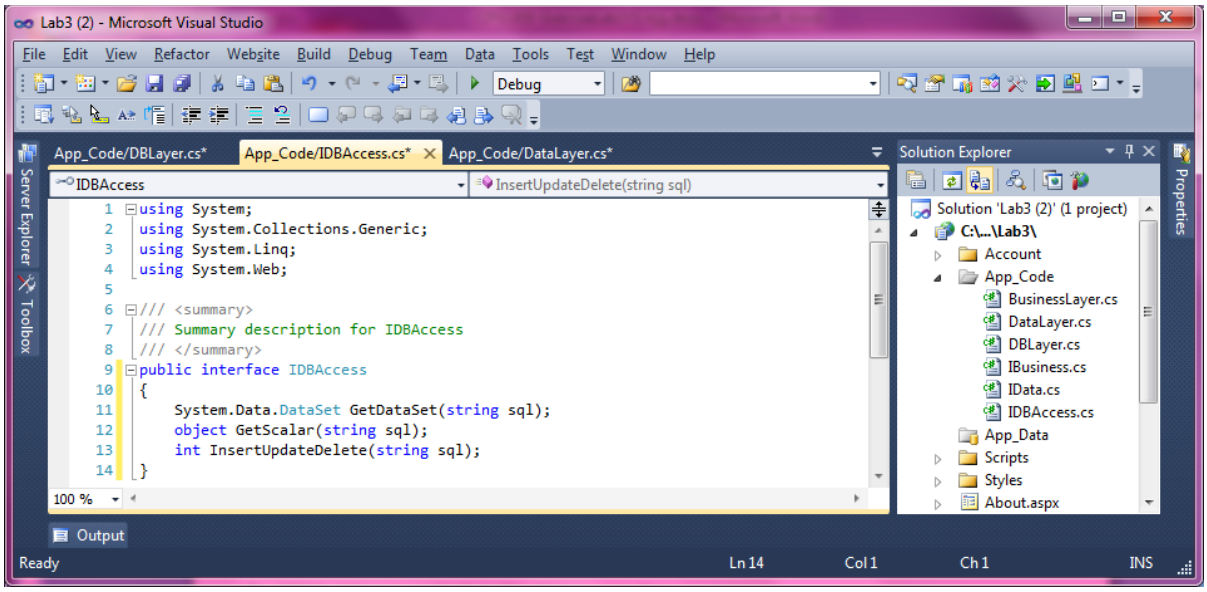


We get this error because GetDataSet is a method in the DataLayer class and we have not defined this method yet. To correct this issue:

Right click on GetDataSet
Select Generate
Select Method Stub
(the method is added to the IData interface – click on the App_Code/IDBAccess.cs tab to see)

STEP 3: Add more methods to the IDBAccess interface.
Click on the App_Code/IDBAccess.cs tab
Copy and paste the following code after the signature for the GetDataSet method

```
object GetScalar(string sql);
int InsertUpdateDelete(string sql);
```



STEP 4: Code for the DBLayer class. This is the layer where we open the connection to the database, and actually communicate with the database.
Click on the App_Code/DBLayer.cs tab
Right-click on IDBAccess
Copy and paste the following code between the { } for the class (NOTE: replace any existing code that is already there)

Important (Add MySQL dll for accessing MySQL database):
Right Click on your project -> Add -> Reference -> Assemblies -> Framework -> Add MySql.Data dll (by searching mysql in the search bar)

Server (i.e., 11.11.11.11) should be replaced by MySQL instance IP address, and YOURPASSWORD should be replace by your paasoword for root.
If you needed, you can change lab database by your database name.

```
    string connStr = "server=11.11.11.11; UID=Your MySQL user(Mostly root);
PASSWORD=YOURPASSWORD; database=lab";

    public DBLayer()
    {
    }

    public System.Data.DataSet GetDataSet(string sql)
    {
        string connstr =
ConfigurationManager.ConnectionStrings[connStr].ConnectionString;
        MySqlConnection conn = new MySqlConnection(
        ConfigurationManager.ConnectionStrings[connStr].ConnectionString);

        DataSet ds = new DataSet();
        try
        {
            conn.Open();
            MySqlDataAdapter da = new MySqlDataAdapter(sql, conn);

            da.Fill(ds);
        }
        catch (Exception ex)
        {
            throw ex;
        }
        finally
        {
            conn.Close();
        }
        return ds;
    }

    public object GetScalar(string sql)
    {
        MySqlConnection conn = new MySqlConnection(
        ConfigurationManager.ConnectionStrings[connStr].ConnectionString);

        object obj = null;
        try
        {
            conn.Open();
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            obj = cmd.ExecuteScalar();
        }
        catch (Exception ex)
        {
            throw ex;
        }
        finally
        {
            conn.Close();
        }
        return obj;
    }

    public int InsertUpdateDelete(string sql)
```

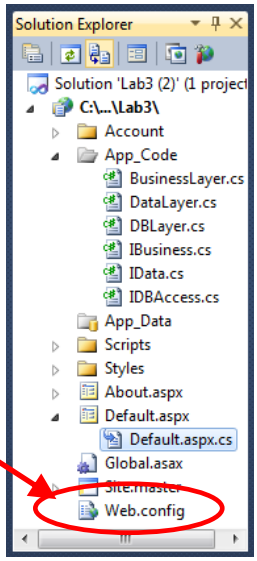
```
{
    MySqlConnection conn = new MySqlConnection(
    ConfigurationManager.ConnectionStrings[connStr].ConnectionString);
    int rows = 0;
    try
    {
        conn.Open();
        MySqlCommand cmd = new MySqlCommand(sql, conn);
        rows = cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        throw ex;
    }
    finally
    {
        conn.Close();
    }
    return rows;
}
```

Take care of all words underlined with the red wavy line,
Right—click on the word
Select Resolve
Select using System._____

B6. PART 6: Modify the Connection String To Connect to The MySQL Database

STEP 1: Now, you have to change the default connection string in order to connect to the Oracle database.

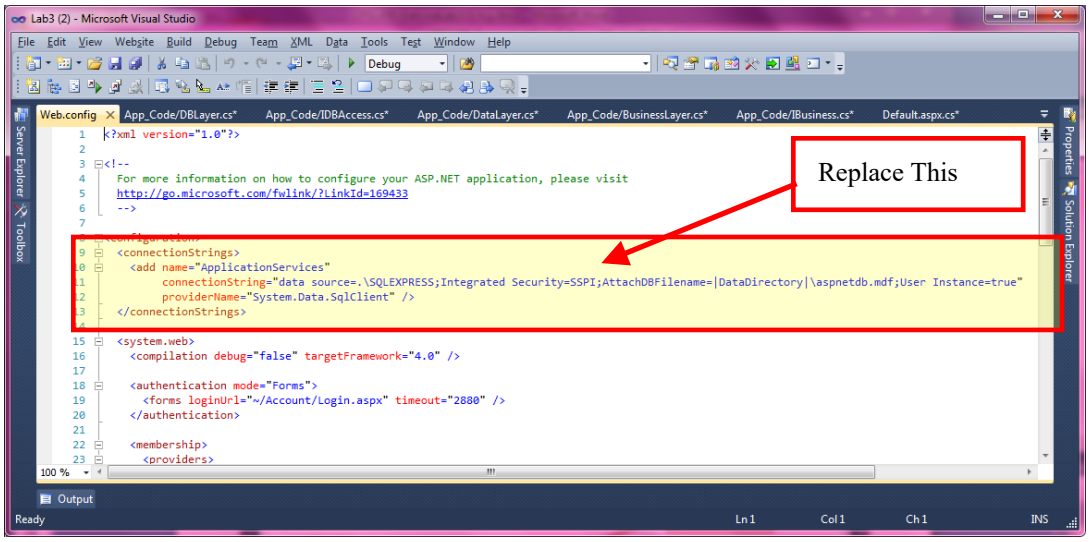
From Solution Explorer, double click on Web.config



Find the connection string and replace with the following:

```
<connectionStrings>
  <add name="Your Connection Name" connectionString="server=11.11.11.11; UID=your userid;
            PASSWORD=your mysql password; database=lab;"/>
</connectionStrings>
```

Your MySQL Username and Pwd. MySQL IP address.

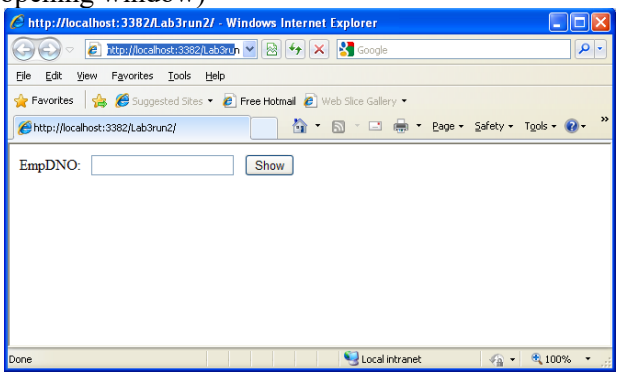


B7. PART 7: Run The Program

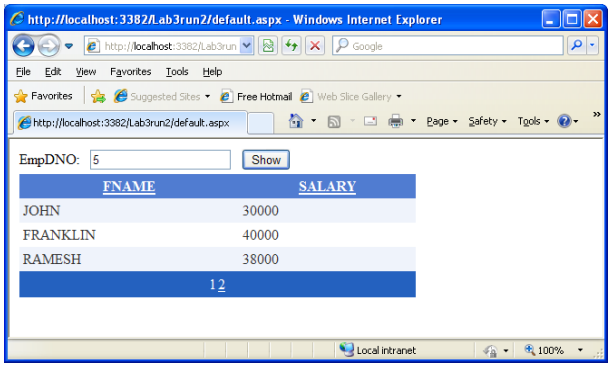
STEP 1: From the menu bar select Debug -> Start Without Debugging

STEP 2: When the web page opens, type '5' in the textbox and click on the Show button – you should see the following:

(opening window)

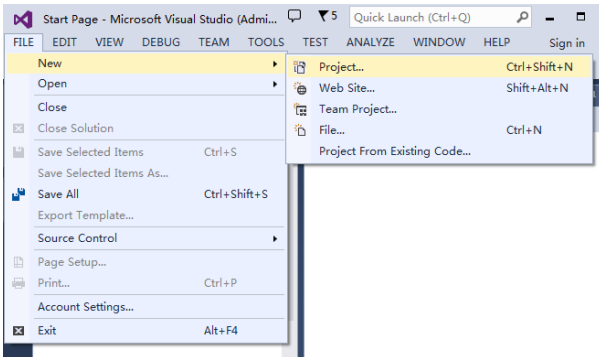


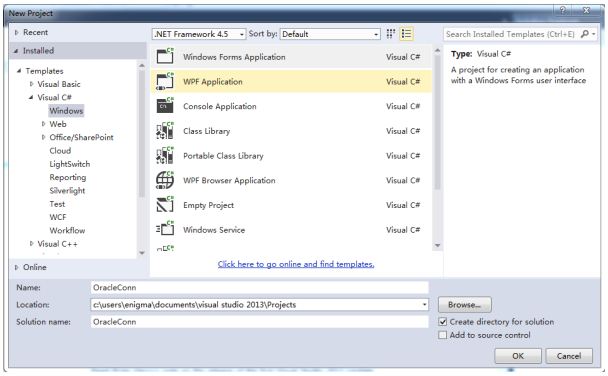
(resulting window)



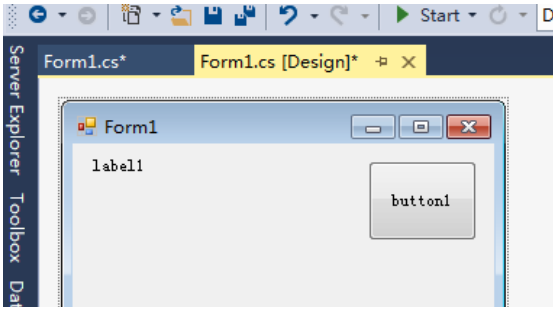
C. C# Visual Studio 2013 Professional

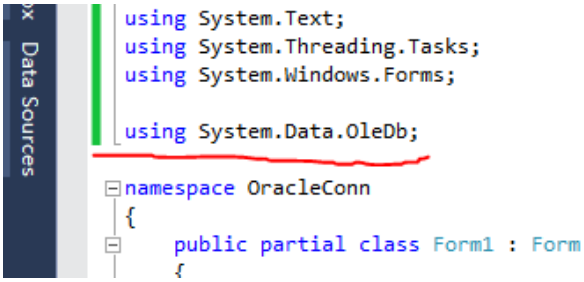
C1. Create a new Project





C2. Added a label and a button in the Form1, add library for Oracle



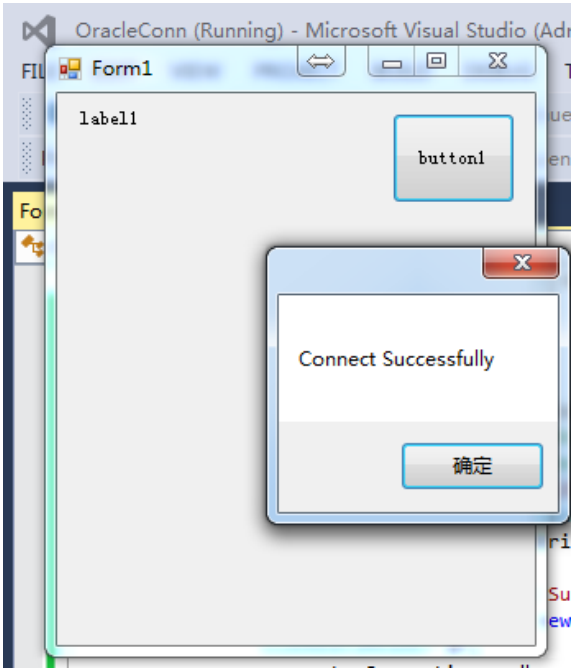


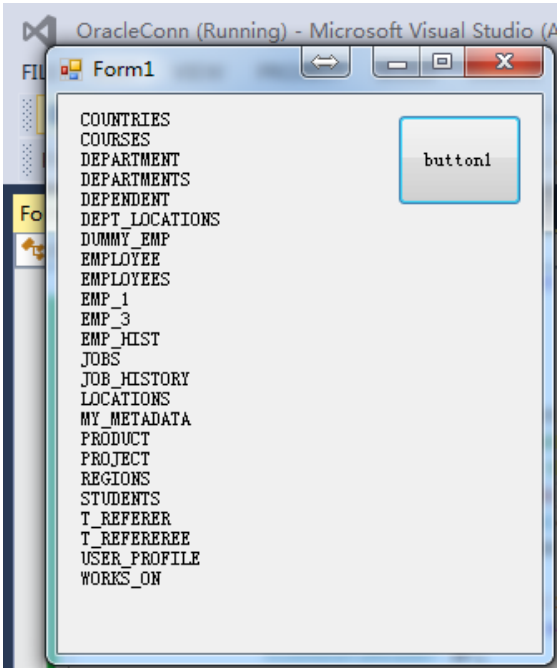
C3. Double click the button1 to copy the code for get data

```
        private void button1_Click(object sender, EventArgs e)
         {
             try
             {
                 string conn;
                  MySqlConnection dbconnection;
                 conn = @"server=11.11.11.11; UID=root; PASSWORD=Your MySQL password;
database=lab";
                 //server= Your MySQL IP address
                 //UID=user name (i.e., root) and PASSWORD=your root password
                 //database= database name (i.e., lab)

                 dbconnection = new MySqlConnection();
                 dbconnection.ConnectionString = conn;
                 dbconnection.Open();
                 MessageBox.Show("Connect Successfully");
                 MySqlCommand comm_str = new MySqlCommand ();
                 MySqlDataReader dr;
                 comm_str.Connection = dbconnection;
                 comm_str.CommandText = "select table_name from user_tables";
                 dr = comm_str.ExecuteReader();
                 label1.Text = "";
```

```
            while (dr.Read())
                label1.Text += (!dr.IsDBNull(0) ? dr.GetString(0) : "") + "\n";
            dr.Close();
            dbconnection.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
```

C4. Run the program

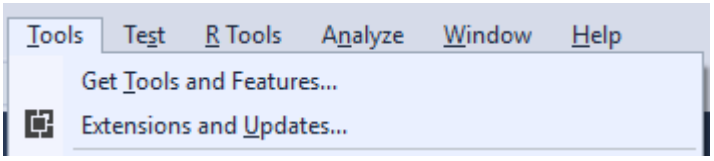




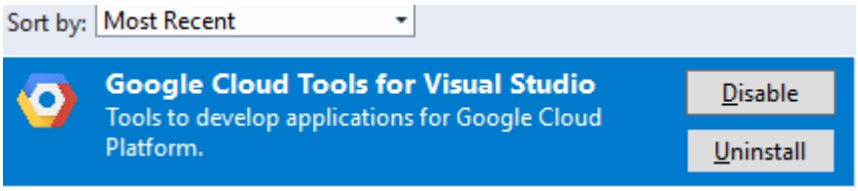
Appendix II: Google Cloud Platform with .Net

A. GCP tools for Visual Studio

1. Install Google Cloud Tools for Visual Studio (at least 2015 version)
   a. Open Tools=> Extensions and Updates

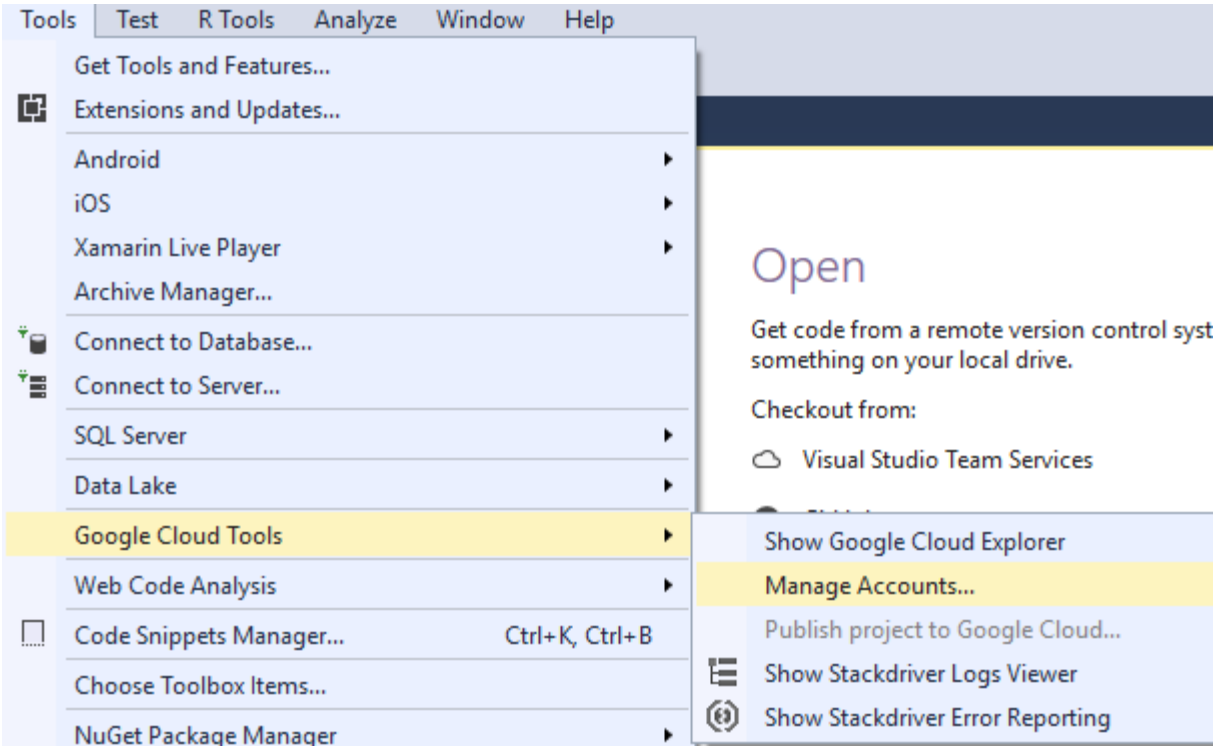


   b.

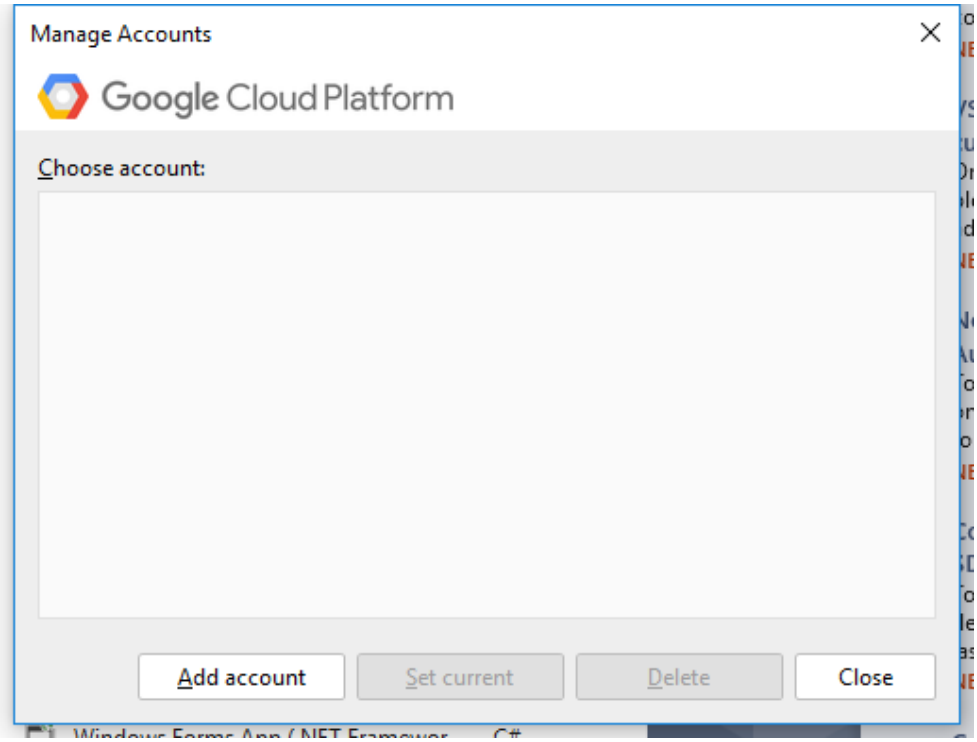


2. Follow the install instructions. You need to close Visual Studio and wait for the installation.

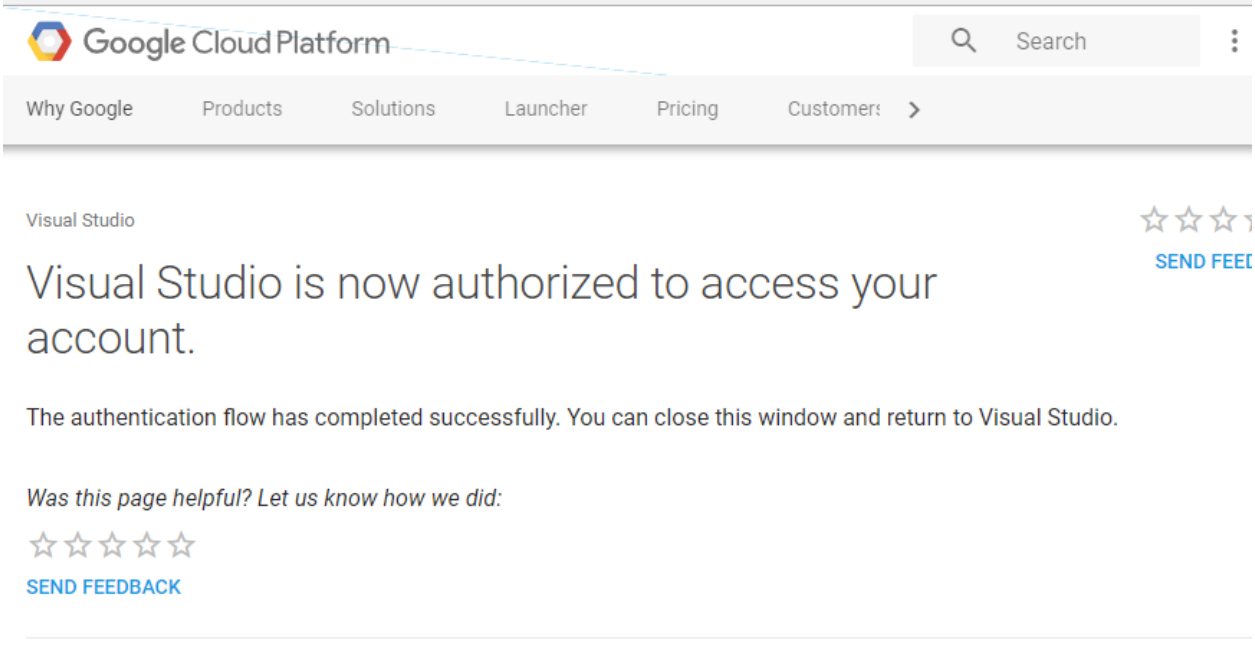
3. Link your google account to Visual Studio.



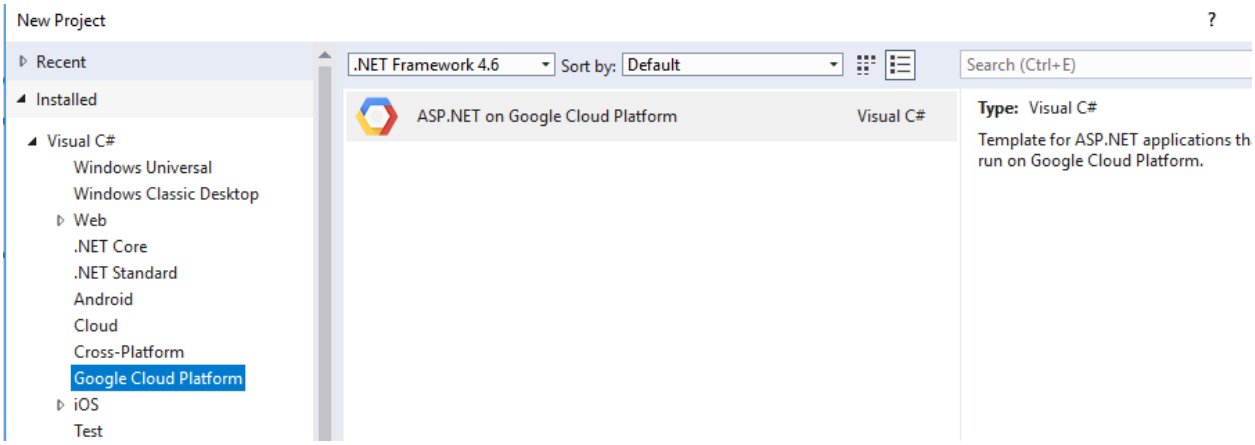
Choose add account and it will open the browser so you can link your google account.



Choose your personal Gmail account instead of UB email account.



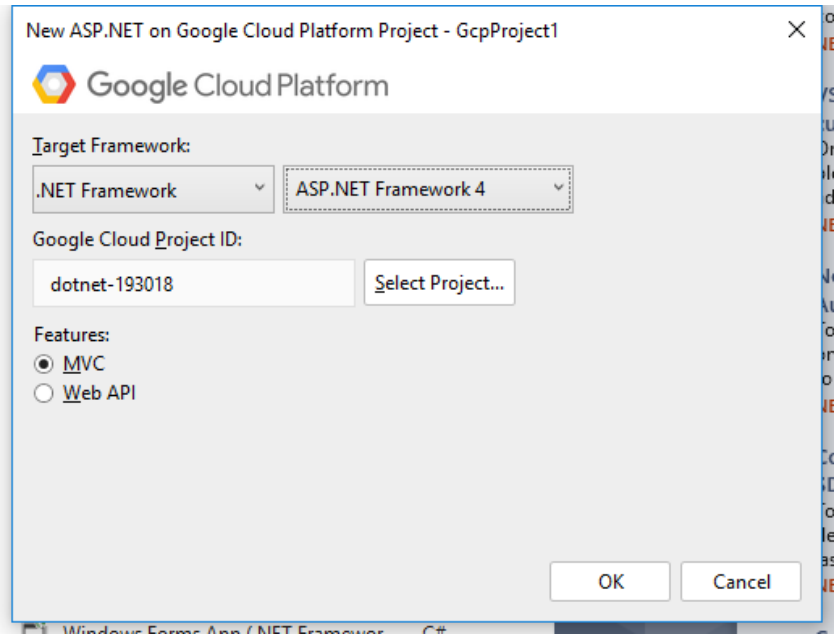
4. After installation, open Visual Studio and create a new project and now you can see a new option called Google cloud Platform.



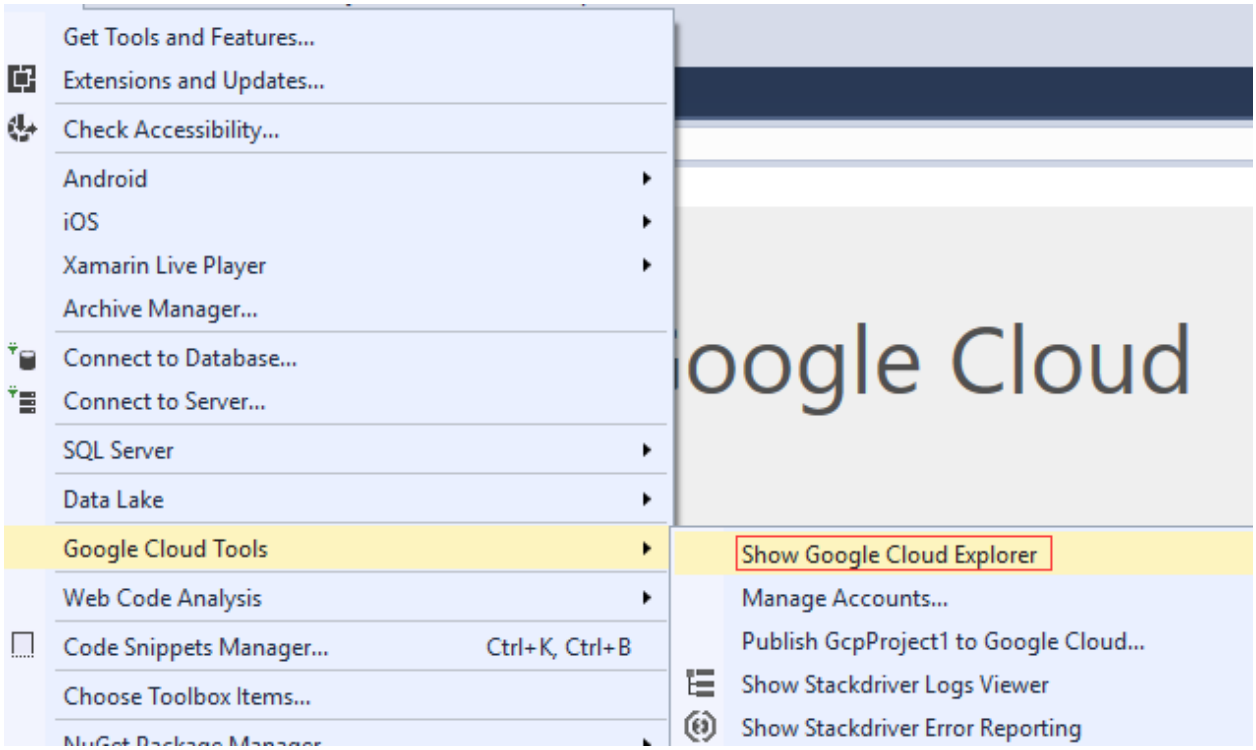
Remember you can choose to use .Net Framework or .Net Core as you wish. You can only use MVC or Web API.

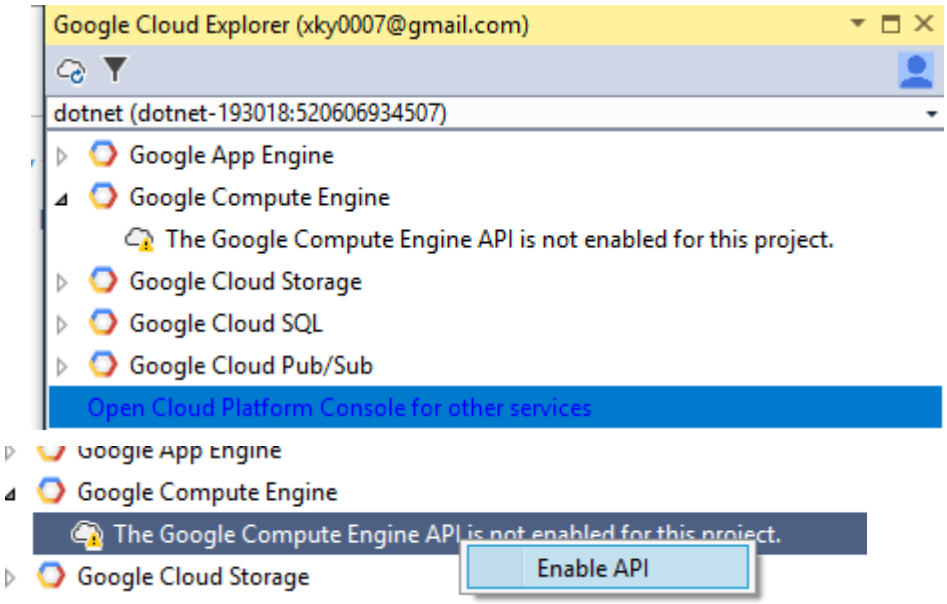
B. .Net Framework

Select your project and set target framework to .net framework.

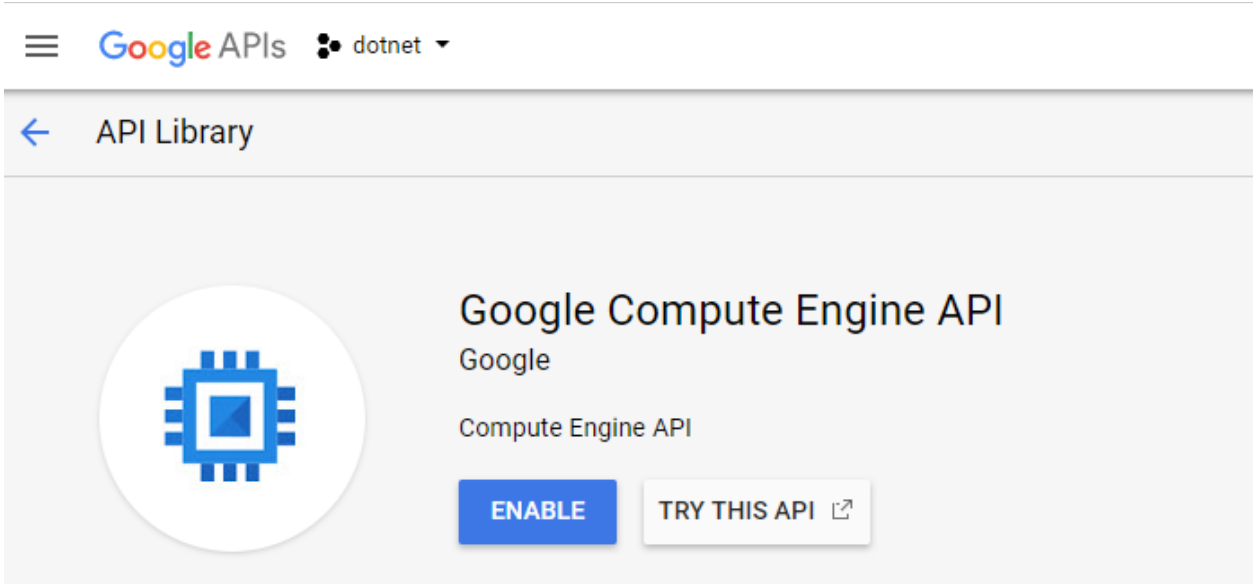


Open Google Cloud Explorer

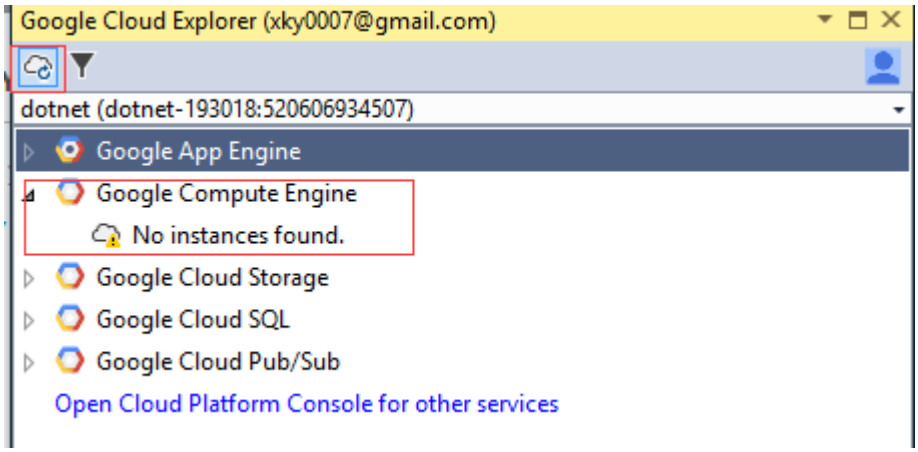


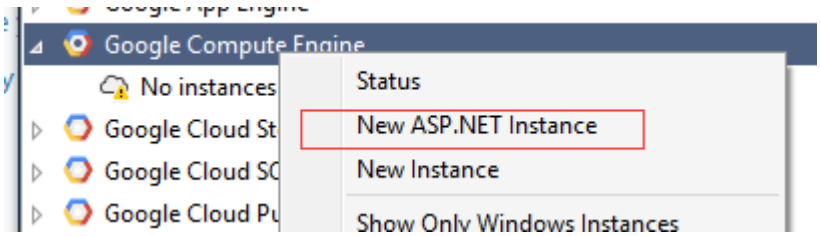


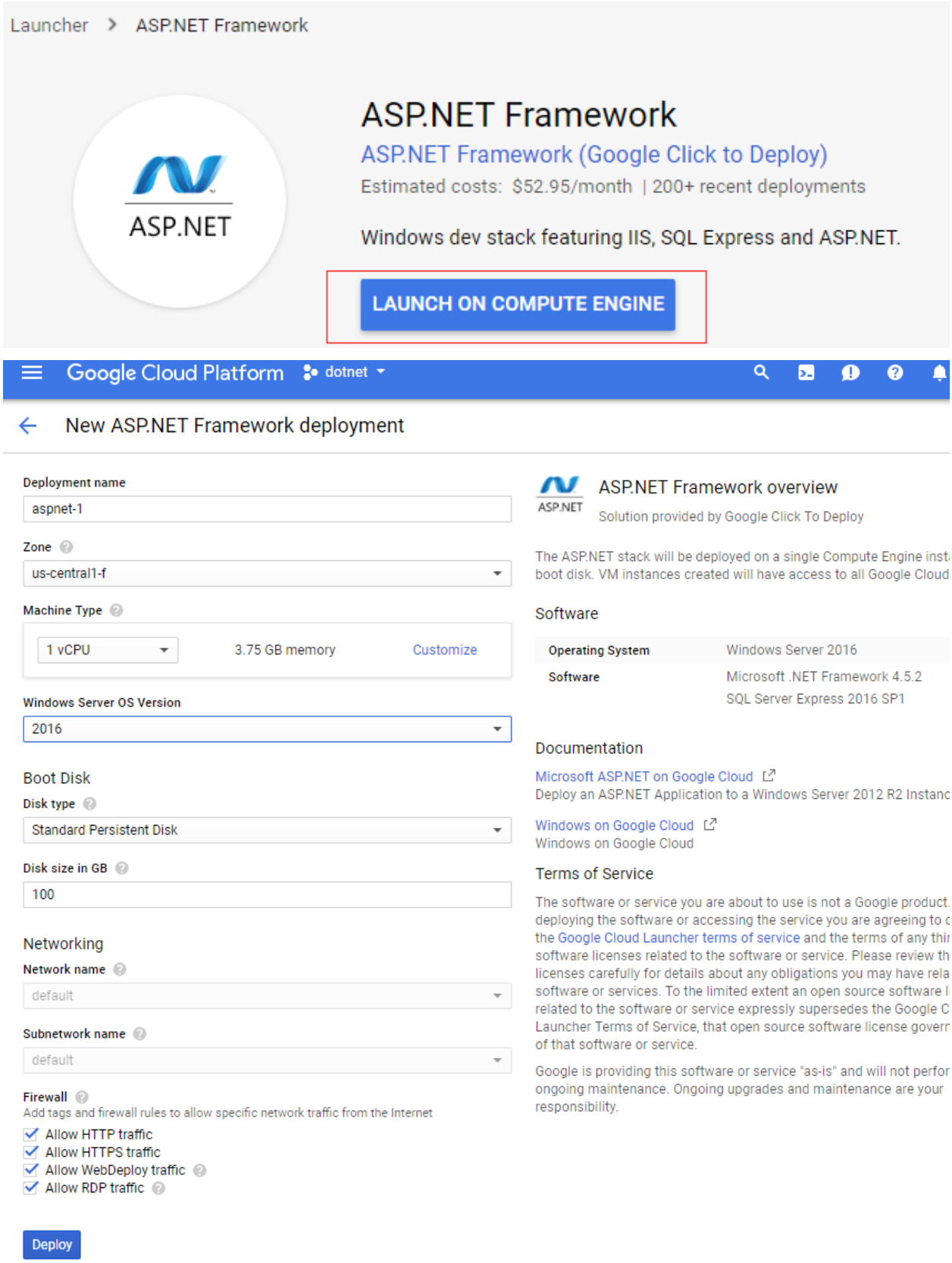
Click enable



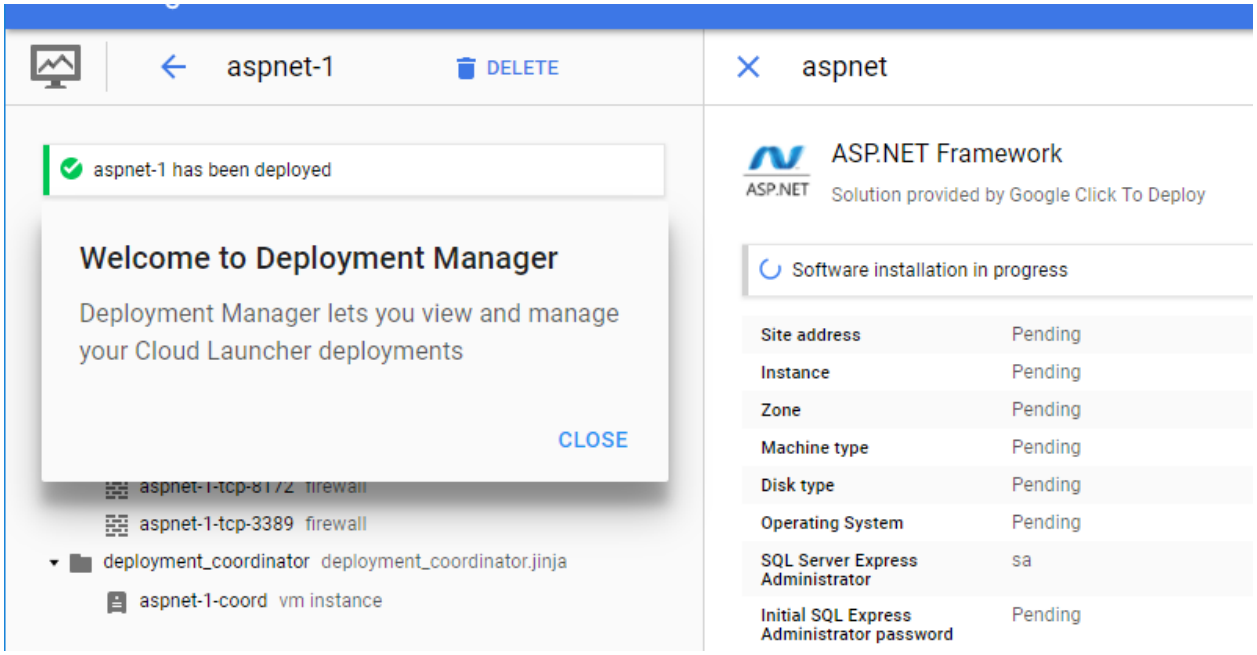
After enable API, click the refresh button in Google Cloud Explorer.



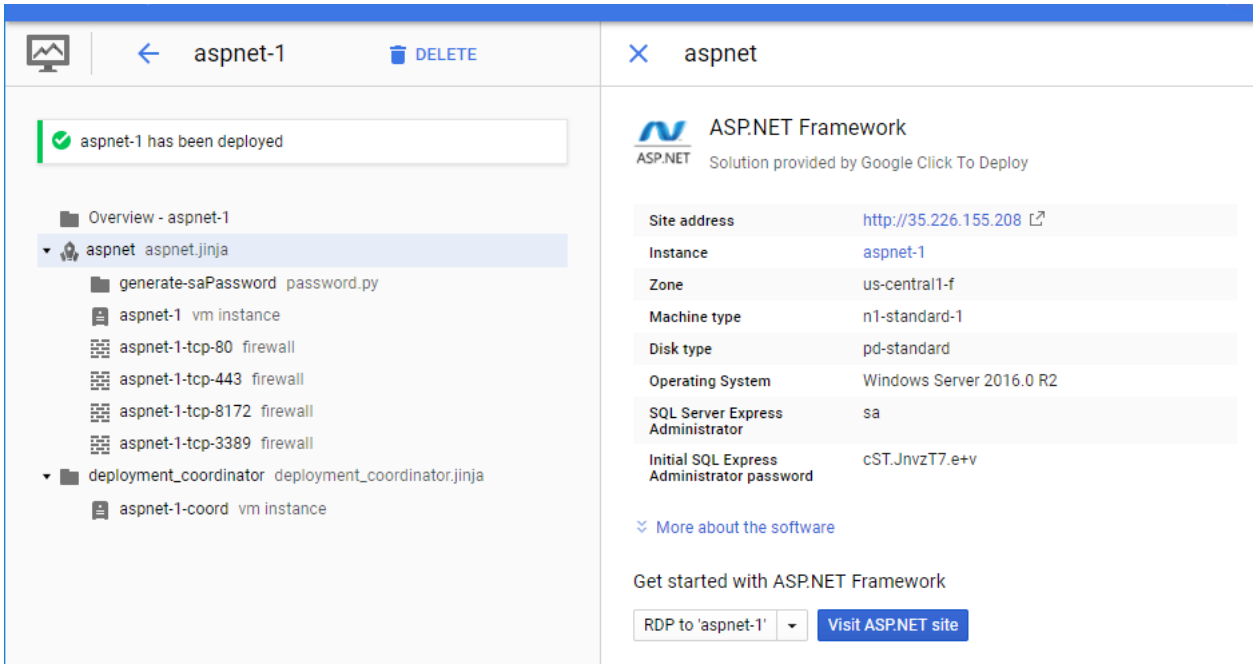



Launcher > ASP.NET Framework
ASP.NET Framework
ASP.NET Framework (Google Click to Deploy)
Estimated costs: $52.95/month | 200+ recent deployments
Windows dev stack featuring IIS, SQL Express and ASP.NET.
LAUNCH ON COMPUTE ENGINE
Google Cloud Platform
dotnet
New ASP.NET Framework deployment
Deployment name
aspnet-1
Zone
us-central1-f
Machine Type
1 vCPU
3.75 GB memory
Customize
Windows Server OS Version
2016
Boot Disk
Disk type
Standard Persistent Disk
Disk size in GB
100
Networking
Network name
default
Subnetwork name
default
Firewall
Add tags and firewall rules to allow specific network traffic from the Internet
Allow HTTP traffic
Allow HTTPS traffic
Allow WebDeploy traffic
Allow RDP traffic
Deploy
ASP.NET Framework overview
Solution provided by Google Click To Deploy
The ASP.NET stack will be deployed on a single Compute Engine inst
boot disk. VM instances created will have access to all Google Cloud
Software
Operating System
Windows Server 2016
Software
Microsoft .NET Framework 4.5.2
SQL Server Express 2016 SP1
Documentation
Microsoft ASP.NET on Google Cloud
Deploy an ASP.NET Application to a Windows Server 2012 R2 Instanc
Windows on Google Cloud
Windows on Google Cloud
Terms of Service
The software or service you are about to use is not a Google product.
deploying the software or accessing the service you are agreeing to
the Google Cloud Launcher terms of service and the terms of any thi
software licenses related to the software or service. Please review th
licenses carefully for details about any obligations you may have rela
software or services. To the limited extent an open source software l
related to the software or service expressly supersedes the Google C
Launcher Terms of Service, that open source software license goverr
of that software or service.
Google is providing this software or service "as-is" and will not perfor
ongoing maintenance. Ongoing upgrades and maintenance are your
responsibility.

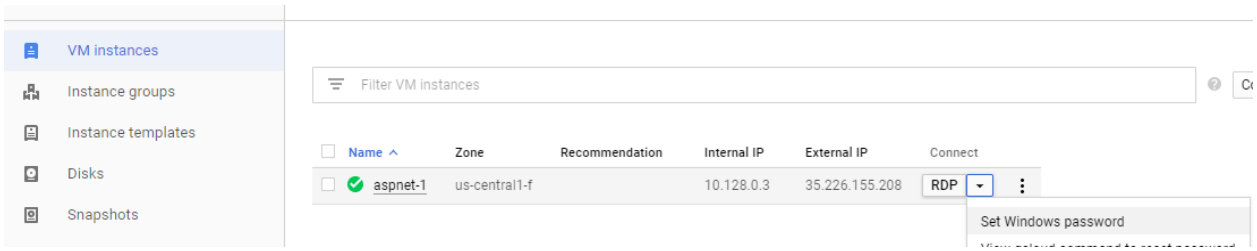
It may take several minutes until the server is deployed. Do not publish your project when you see this as everything is pending.

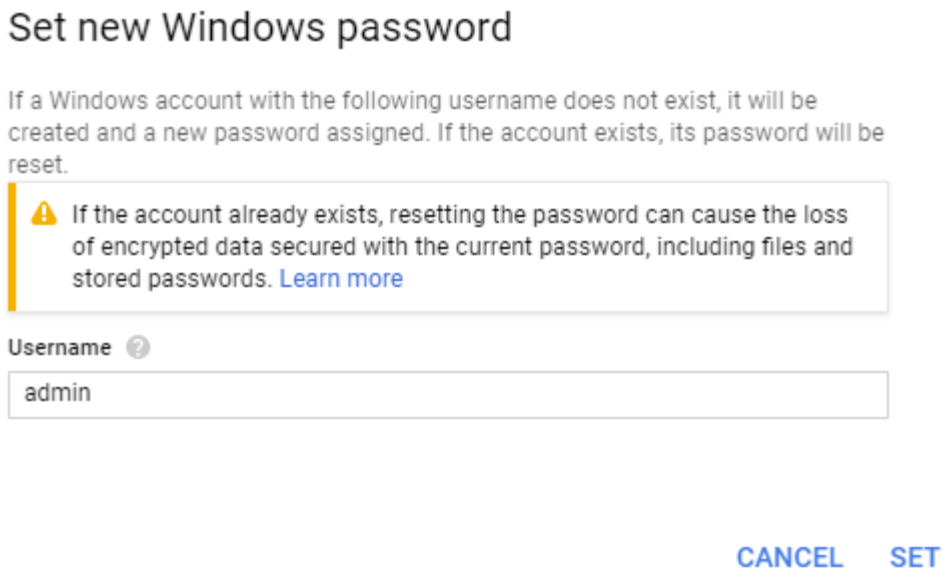


After everything is installed, you can see

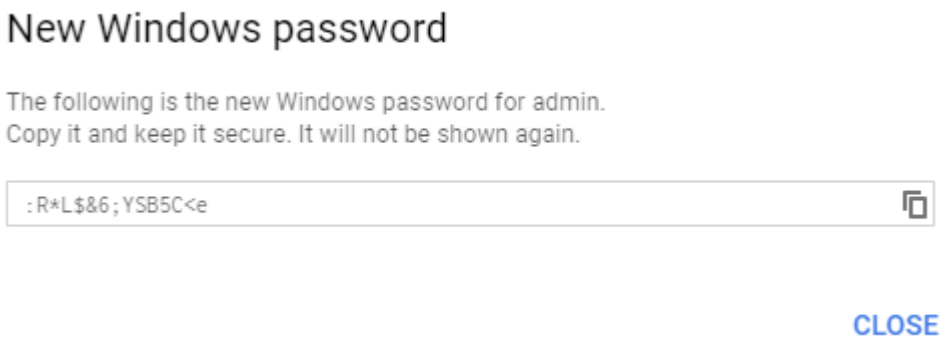


Open the CM instances in GCP, set windows password

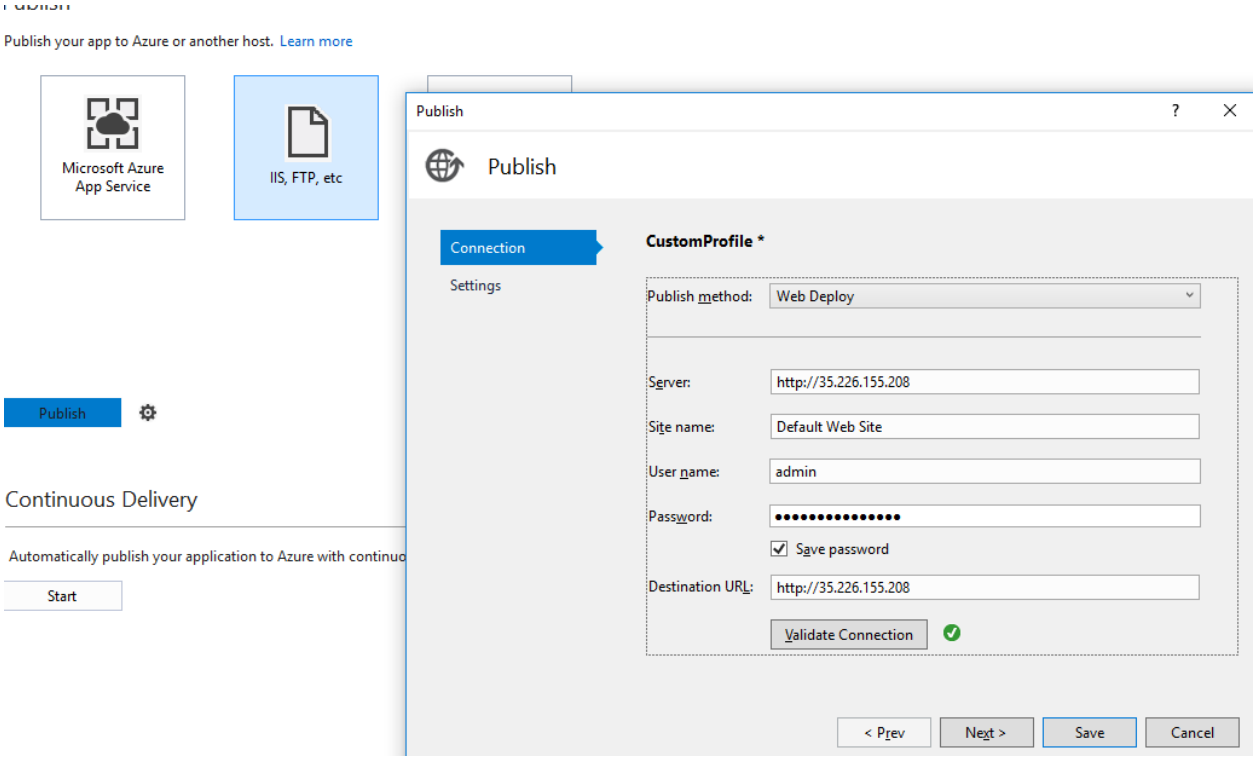




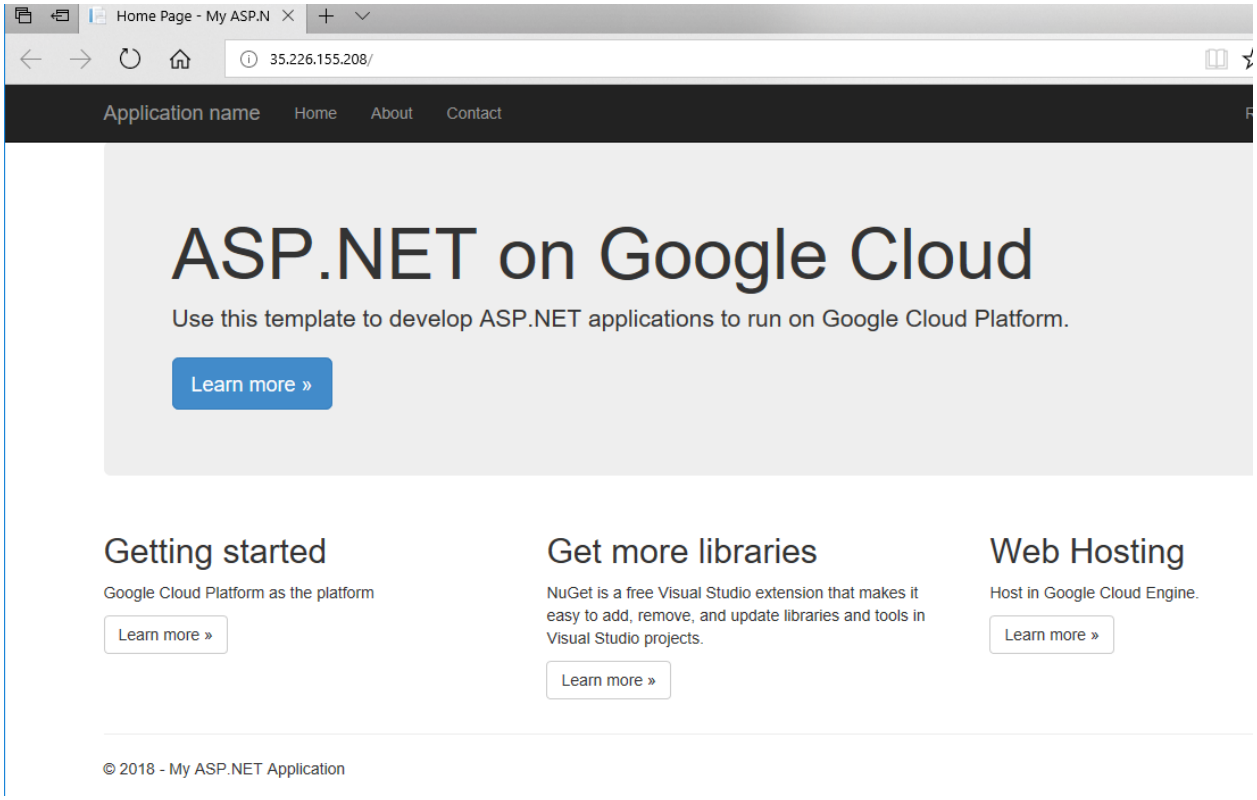
Copy the password



Go to visual studio. Right click the project and select publish. The address is the external IP for you VM instance.

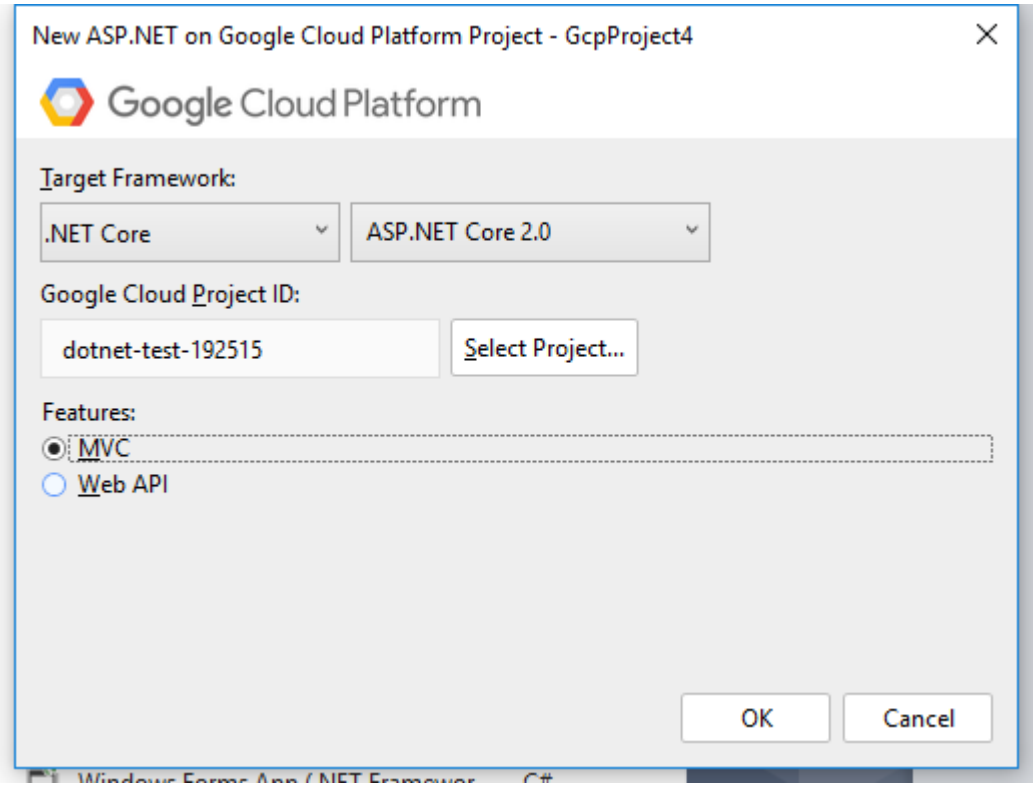


After save and click publish. Open the url for your app and you will see the sample MVC project now.

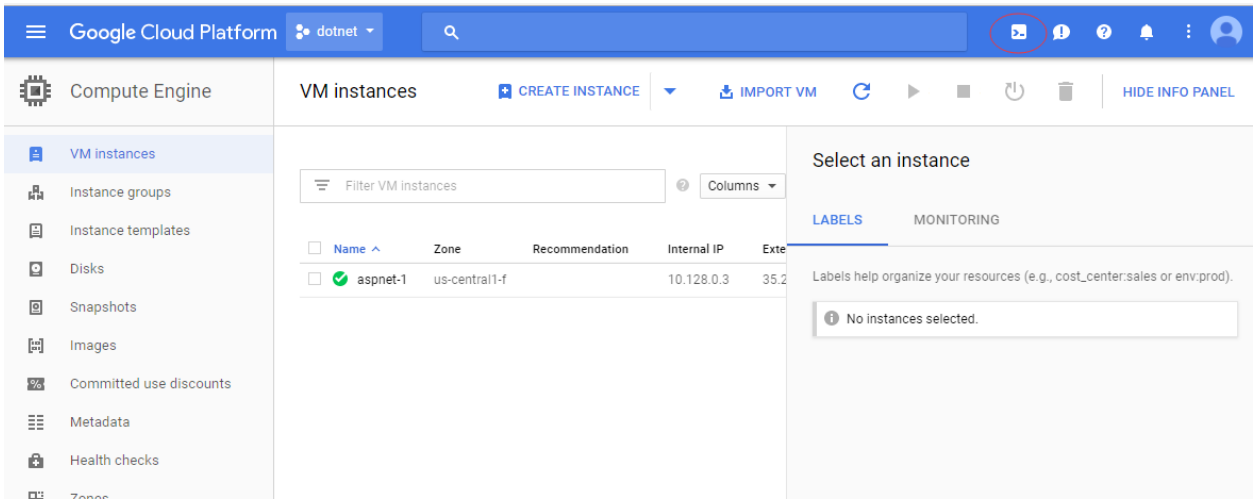


C. .Net Core

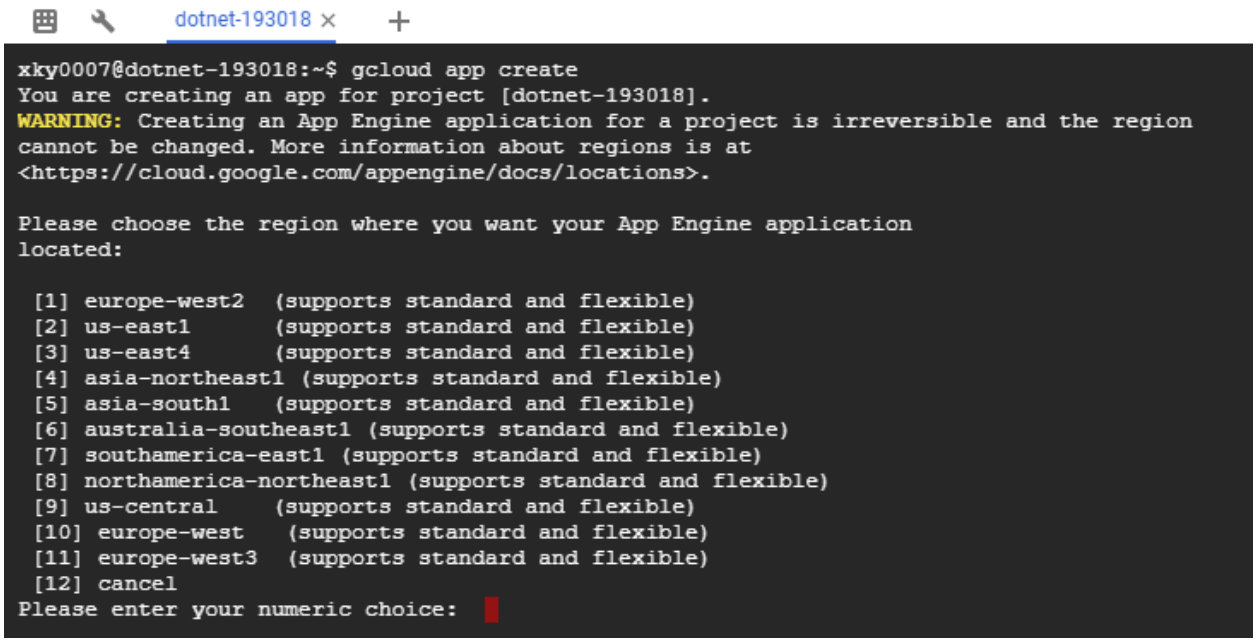
Now let's deploy an .Net Core project. Create a new Google Cloud Platform project.



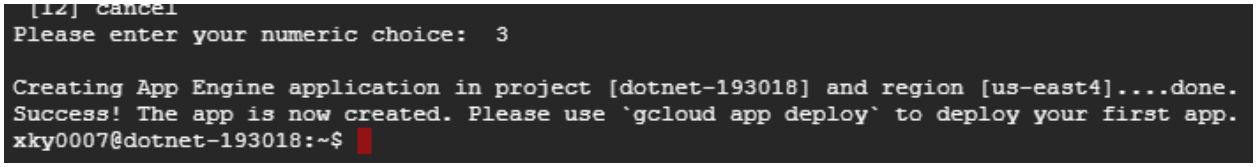
Open your GCP project and open the Google Cloud Shell



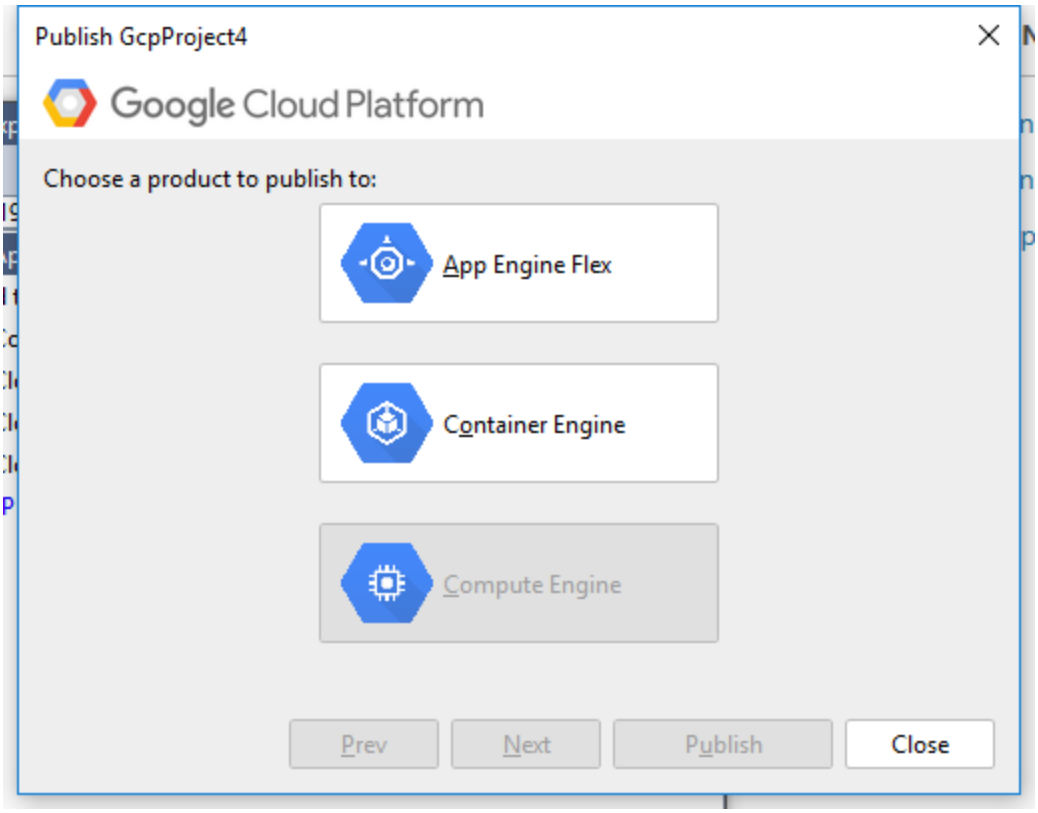
Type cloud app create and you will see some selections.



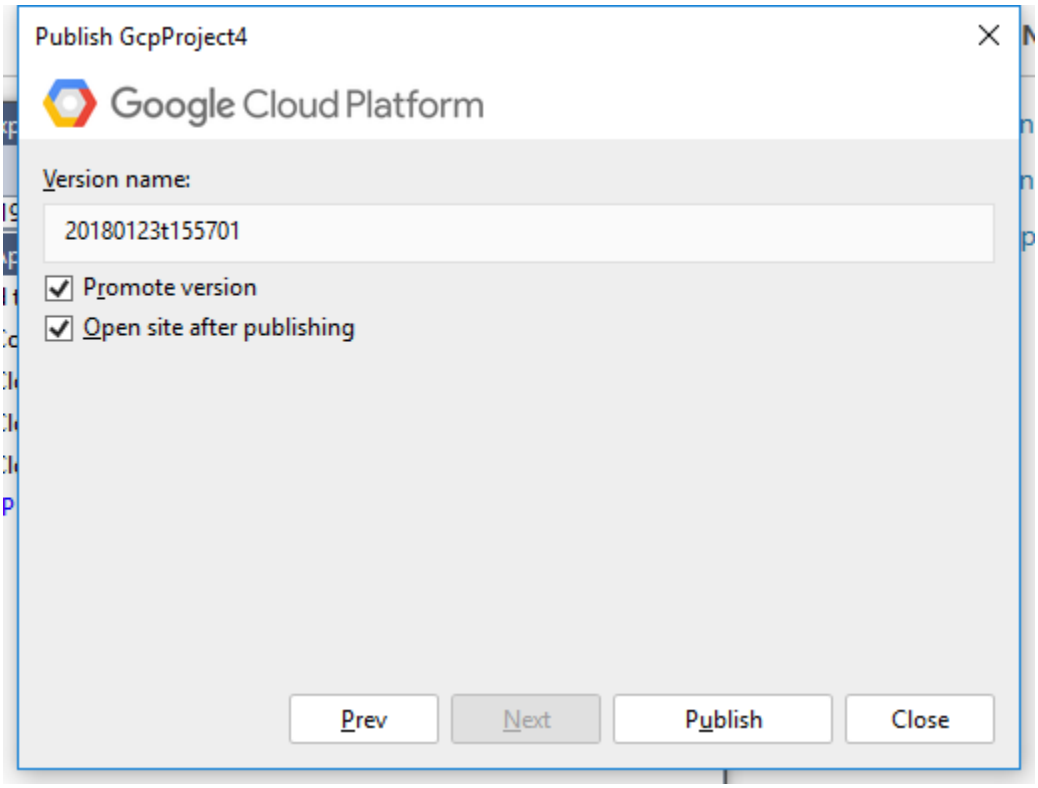
Choose your preferred app engine location.



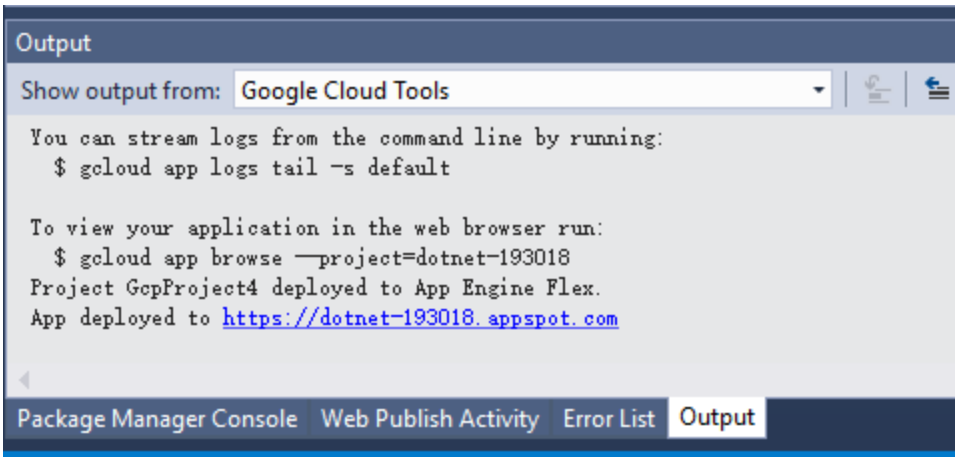
Right click the project in solution explorer. Select publish to Google Cloud and select app engine flex.



Set your version name and publish.



You will see this after deployment.



Open the link and you will see. Now your .Net Core MVC has been published to the app engine in Google Cloud Platform.