This in-class assignment is to practice with MySQL queries. We will be using the alketo database manager to write our queries, rather than building tables and inserting data via PHP.

Database manager: http://www.alketo.info/database/

Please check your school email inbox for your group's password to access. You will need to make sure to not have your work collide with your teammates', by prefixing your tables with your name.

Creating your first table, with a Primary Key

When creating a table, you will need to specify a **column name**, a **data type**, and any properties it may have. Some properties are “NOT NULL”, if a field should always be filled, or “UNIQUE”, to make sure any information filled in a record in this column is not repeated in any other records.

You will specify a primary key separately from your column list, but still within the CREATE statement. You will specify which column is a primary key with the “PRIMARY KEY” command.

Enter the following command into the database manager:

```
CREATE TABLE lastname_calendar (
      calendar_id int AUTO_INCREMENT,
      title varchar(64) NOT NULL,
      PRIMARY KEY( calendar_id )
)
```

After submitting the query, the table should show up in the right-side table list. If it does not, check your statement for syntax errors.

Creating a second table, with a Foreign Key

We can specify a foreign key similarly to the primary key. Enter this command into the database manager:

```
CREATE TABLE lastname_task (
      task_id int AUTO_INCREMENT,
      calendar_id int,
      title varchar(64) NOT NULL,
      start datetime NOT NULL,
      end datetime NOT NULL,
      importance int,
      urgency int,
      PRIMARY KEY( task_id ),
      FOREIGN KEY( calendar_id ) REFERENCES lastname_calendar( calendar_id )
)
```

Altering an existing table

Let's alter the Calendar table to also store a “color”, as RGB values. You can modify an existing table with the ALTER command:

```
ALTER TABLE lastname_calendar
ADD color_red int,
ADD color_green int,
ADD color_blue int
```

You can also DROP columns of your table this way.

Creating and dropping a table

Let's quickly create and drop a table, just to show how the functionality works.
Create this table:

```
CREATE TABLE lastname_mock (
     my_column int,
     PRIMARY KEY ( my_column )
)
```

Then drop it:

```
DROP TABLE lastname_mock
```

Note that, if you're trying to drop a table with data in it, and that table is being pointed to by another table with a foreign key, you will get an error message. You will need to delete any linked data before dropping the table.

Inserting information into your tables

Let's insert some information into the Calendar table. Since the Task table is linked to the Calendar via a foreign key, it would be best to set up the Calendar's data first.

```
INSERT INTO lastname_calendar
( title, color_red, color_green, color_blue )
VALUES
( "School", "255", "0", "0" )
```

Make sure to click the “Insert” radio button before hitting Submit Query.

Insert a couple more records into this table:

```
INSERT INTO lastname_calendar
( title, color_red, color_green, color_blue )
VALUES
( "Work", "0", "255", "0" )
```

```
INSERT INTO lastname_calendar
( title, color_red, color_green, color_blue )
VALUES
( "Errands", "0", "0", "255" )
```

You can also insert information into a table without specifying column names. If you do this, you need to make sure to specify VALUES in order!

Now, make sure to click on the [lastname_calendar] link in the database manager. You should see all of your rows listed under the **Rows** section.

Rows

Name: calendar_id Type: int(11) Nullable?: NO, Default: , Key: PRI	Name: title Type: varchar(64) Nullable?: NO, Default: , Key:	Name: color_red Type: int(11) Nullable?: YES, Default: , Key:	Name: color_green Type: int(11) Nullable?: YES, Default: , Key:	Name: color_blue Type: int(11) Nullable?: YES, Default: , Key:
1	School	255	0	0
2	Errands	0	0	255
3	Work	0	255	0

Now, let's insert some information into the **lastname_task** table. We will be linking to a calendar here!

```
INSERT INTO lastname_task
( calendar_id, title, start, end, importance, urgency )
VALUES
( 2,
'Turn in report',
'2013-10-23 10:00:00',
'2013-10-23 10:30:00',
2, 5 )
```

But, notice that we've had to hard-code the value for calendar #2! How can we select a calendar's ID based on its title? We can do this with a SELECT statement, in our INSERT statement.

```
INSERT INTO lastname_task
( calendar_id, title, start, end, importance, urgency )
VALUES
( (SELECT calendar_id FROM lastname_calendar WHERE title='School'),
'Midterm Exam',
'2013-10-23 10:00:00',
'2013-10-23 10:30:00',
5, 4 )
```

So, we are selecting a specific **calendar_id** from the calendar table, so long as the criteria (title is "School") is met.

Updating existing information

Now that we have a couple of tasks, let's update a record, such as the urgency.

```
UPDATE lastname_task
SET urgency = 3
WHERE task_id = 1
```

So, you specify what column to update with the **SET** command, and you specify how to choose which records to update with the **WHERE** command.
This could be a specific ID, or any other conditional statement:

```
UPDATE lastname_task
SET urgency = 5
WHERE start <= '2013-10-25'
```

Deleting records from your table

You can easily also delete records from the table with the **DELETE** command.

```
DELETE FROM lastname_task
WHERE importance < 2
```

Selecting all information from one table

You can select all records from a given table with the asterisk * wildcard:

```
SELECT * FROM lastname_task
```

Selecting specific columns from one table

You can also specify certain columns in your select statement:

```
SELECT title, start, importance FROM lastname_task
```

Selecting information from multiple tables

And, we can use the **JOIN** command to select information from multiple tables:

```
SELECT t.title, t.start, c.title
FROM lastname_task AS t
INNER JOIN lastname_calendar AS c
ON c.calendar_id = t.calendar_id
```

Notice the highlighted sections. Using **AS** allows us to give each table an alias. Without this, for any columns that have the same name, we would have to type out the full table name as a prefix:

```
SELECT lastname_task.title, lastname_calendar.title, …
```

But instead, we can give each table an alias and reference each with a shorter name.

For this select statement, we're retrieving the **title** and **start** values from the task table, and the **title** value from the calendar.

We have to use a **JOIN** (an inner join, here) to specify what two columns to link task to calendar with.