Introduction to SQL

Exercises marked with a * are based on material from Learning SQL, Alan Beaulieu, O’Reilly, 2005

11.1 Walkthrough examples

This section is designed to walk you through the main queries you need to be able to perform on a MySQL database.

11.1.1 Walkthrough 1: Importing and exploring a database*

This exercise shows you how to import a database into MySQL then do some basic investigation of the data. We will be using data from the Learning SQL book which you can download from http://examples.oreilly.com/learningsql/. MySQL is freely available and you can download it for your home machine from http://www.mysql.com/.

1. To start log in to MySQL as the root user
   
   % mysql

   By default with a new MySQL server the password is empty by default, unless you have set one in the installation process. So just press <Enter> if you haven’t set a root password.

2. Create a new user that will who you run as for this session by typing the following, with your own username and a password. Don’t use any of your regular passwords because other people running on this machine may have access to your password depending on the MySQL server configuration.

   mysql> grant all privileges on *.* to 'username'@'localhost' identified by 'password';

   Note that all commands in MySQL must end in a semicolon.

3. Now exit MySQL (by typing quit; ) and log back in as the user you have just created

   % mysql

4. We will now create the bank database and import the example data

   mysql> CREATE database bank;
   mysql> USE bank;
   mysql> SOURCE LearningSQLExample.sql;

5. From now on, when you login to MySQL you can specify the bank database with the -p option

   % mysql -u username -p bank

6. We can now check the data has been correctly loading it by exploring some of the tables. Try the following commands and see if the results are what you expect

   • SHOW TABLES;
   • DESC department;
   • DESC branch;

7. If you press enter before putting a semicolon, MySQL will go onto the next line. You can then put a semicolon to end the previous statement. For example
8. We can now try our first SQL query to retrieve some information from the database

```sql
mysql> SELECT fname, lname
     -> FROM employee;
```

<table>
<thead>
<tr>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael</td>
<td>Smith</td>
</tr>
<tr>
<td>Susan</td>
<td>Barker</td>
</tr>
<tr>
<td>Robert</td>
<td>Tyler</td>
</tr>
<tr>
<td>Susan</td>
<td>Hawthorne</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

18 rows in set (0.26 sec)

9. If your query has no results, the server will return an empty set:

```sql
mysql> SELECT fname, lname
     -> FROM employee
     -> WHERE fname = 'frog';
```

Empty set (0.00 sec)

10. Remember that if you want to do a query using only built-in functions, that does not retrieve any data from the database, then you only need the `SELECT` clause. For example, to get some information about the database itself

```sql
mysql> SELECT VERSION(), USER(), DATABASE();
```

<table>
<thead>
<tr>
<th>VERSION()</th>
<th>USER()</th>
<th>DATABASE()</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0.18-max</td>
<td>tara@localhost</td>
<td>bank</td>
</tr>
</tbody>
</table>

1 row in set (0.00 sec)

You have now loaded a pre-existing database into MySQL and done some preliminary investigation of the data. We will use this sample database for the rest of this session.
11.1.2 Walkthrough 2: Filtering results and NULL values

In this walkthrough we look at some of the ways results can be filtered using the where clause.

1. The simplest filter we can do is to check for equality, for example to find all the employees whose last name is Smith

   ```sql
   mysql> SELECT emp_id, fname, lname, start_date
       -> FROM employee
       -> WHERE lname = 'Smith';
   +--------+---------+-------+------------+
   | emp_id | fname   | lname | start_date |
   +--------+---------+-------+------------+
   | 1      | Michael | Smith | 2001-06-22 |
   +--------+---------+-------+------------+
   1 row in set (0.00 sec)
   
   Note that by default MySQL is case-insensitive when checking strings, so WHERE lname = 'smith'; produces the same results. This behaviour can be changed.

2. You can also specify a range of values that must be satisfied, for example to select employees based on their starting date:

   ```sql
   mysql> SELECT emp_id, fname, lname, start_date
       -> FROM employee
       -> WHERE start_date >= '2002-06-01'
       -> AND start_date < '2003-01-01';
   +--------+-------+---------+------------+
   | emp_id | fname | lname   | start_date |
   +--------+-------+---------+------------+
   | 2      | Susan | Barker  | 2002-09-12 |
   | 8      | Sarah | Parker  | 2002-12-02 |
   | 10     | Paula | Roberts | 2002-07-27 |
   | 14     | Cindy | Mason   | 2002-08-09 |
   | 17     | Beth  | Fowler  | 2002-06-29 |
   | 18     | Rick  | Tulman  | 2002-12-12 |
   +--------+-------+---------+------------+
   6 rows in set (0.01 sec)
   
   3. An equivalent query, using the BETWEEN operator

   ```sql
   mysql> SELECT emp_id, fname, lname, start_date
       -> FROM employee
       -> WHERE start_date BETWEEN '2002-06-01' AND '2003-01-01';
   +--------+-------+---------+------------+
   | emp_id | fname | lname   | start_date |
   +--------+-------+---------+------------+
   | 2      | Susan | Barker  | 2002-09-12 |
   | 8      | Sarah | Parker  | 2002-12-02 |
   | 10     | Paula | Roberts | 2002-07-27 |
   | 14     | Cindy | Mason   | 2002-08-09 |
   | 17     | Beth  | Fowler  | 2002-06-29 |
   | 18     | Rick  | Tulman  | 2002-12-12 |
   +--------+-------+---------+------------+
   6 rows in set (0.01 sec)
   
   4. Remember in the lecture we discussed that values can be null but not equal null. We can now try this out. We want to find all employees who do not have a superior (or boss).

   ```sql
   mysql> SELECT emp_id, fname, lname, superior_emp_id
       -> FROM employee
       -> WHERE superior_emp_id IS NULL;
   
   The above query will tell you that the answer is 'Michael Smith'. The following query, using = NULL does not return any results.

   ```sql
   mysql> SELECT emp_id, fname, lname, superior_emp_id
       -> FROM employee
       -> WHERE superior_emp_id = NULL;
   
   This difference catches a lot of people out!
5. Our next task is to find all employees who are not managed by Helen Fleming who has employee ID is 6. At a first guess, your SQL query might be

```sql
mysql> SELECT emp_id, fname, lname, superior_emp_id
-> FROM employee
-> WHERE superior_emp_id != 6;
```

which should result in 14 employees who do not work for Helen. However one is missing — Michael Smith. We know his ID is NULL. To make sure NULL values are included in the results we have to modify the query to

```sql
mysql> SELECT emp_id, fname, lname, superior_emp_id
-> FROM employee
-> WHERE superior_emp_id != 6 OR superior_emp_id IS NULL;
```

Now there are 15 employees, including Michael Smith.

6. Finally lets look at a query involving wildcard matching. We want to find all employees whose name has 4 letters. SQL has the following wildcard characters One possible query is

<table>
<thead>
<tr>
<th>Character</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Exactly one character</td>
</tr>
<tr>
<td>%</td>
<td>Zero or more characters</td>
</tr>
</tbody>
</table>

```sql
mysql> SELECT emp_id, fname, lname
-> FROM employee
-> WHERE fname LIKE '____';
```

```
+--------+-------+----------+
<table>
<thead>
<tr>
<th>emp_id</th>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>John</td>
<td>Gooding</td>
</tr>
<tr>
<td>9</td>
<td>Jane</td>
<td>Grossman</td>
</tr>
<tr>
<td>13</td>
<td>John</td>
<td>Blake</td>
</tr>
<tr>
<td>17</td>
<td>Beth</td>
<td>Fowler</td>
</tr>
<tr>
<td>18</td>
<td>Rick</td>
<td>Tulman</td>
</tr>
</tbody>
</table>
+--------+-------+----------+
5 rows in set (0.00 sec)
```

7. Likewise we could find all the employees with a 4 letter name that starts with ‘J’ using

```sql
mysql> SELECT emp_id, fname, lname
-> FROM employee
-> WHERE fname LIKE 'J__';
```

```
+--------+-------+----------+
<table>
<thead>
<tr>
<th>emp_id</th>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>John</td>
<td>Gooding</td>
</tr>
<tr>
<td>9</td>
<td>Jane</td>
<td>Grossman</td>
</tr>
<tr>
<td>13</td>
<td>John</td>
<td>Blake</td>
</tr>
</tbody>
</table>
+--------+-------+----------+
3 rows in set (0.00 sec)
```

You should now know some of the ways of selecting which rows you want in your results, using the `WHERE` clause. You should also keep in mind that you always need to be aware of which columns in which tables are allowed to contain NULL values (use `DESC employee` etc.).
11.1.3  Walkthrough 3: Joining tables

In this walkthrough we look at joining tables which is necessary for all queries where the data is split over multiple tables.

1. Start by doing the Cartesian product (or cross join) to try and find the name of the department that each employee works in.

   ```
   mysql> SELECT e.fname, e.lname, d.name
   >   --> FROM employee e
   >   --> JOIN department d;
   ```

   If you look down the list of results you see that this has just printed each employee name next to each department, and hence there are 54 results \((18 \times 3)\) instead of 18 — one for each employee.

2. To get the results we expect we need to use an inner join which makes use of the foreign key to link up two tables.

   ```
   mysql> SELECT e.fname, e.lname, d.name
   >   --> FROM employee e
   >   --> JOIN department d ON e.dept_id = d.dept_id;
   ```

3. You should be aware of the pre-SQL92 syntax for doing an inner join as it is still widely used. Try the following query and check you get the same results

   ```
   mysql> SELECT e.fname, e.lname, d.name
   >   --> FROM employee e, department d
   >   --> WHERE e.dept_id = d.dept_id;
   ```

4. Also notice that in all these queries we are giving the tables aliases to make them easier to refer to \((e.fname)\) rather than \((employee.fname)\).

5. Now we will try a three-table join to retrieve the account ID, federal tax number and the name of the teller that opened the account, for all business accounts. Looking through the database schema we find that we need three tables — `account`, `customer` and `employee`.

   ```
   mysql> SELECT a.account_id, c.fed_id, e.fname, e.lname
   >   --> FROM account a
   >   --> JOIN customer c ON a.cust_id = c.cust_id
   >   --> JOIN employee e ON a.open_emp_id = e.emp_id
   >   --> WHERE c.cust_type_cd = 'B';
   ```

   +------------+------------+---------+---------+
   | account_id | fed_id     | fname   | lname   |
   +------------+------------+---------+---------+
   | 20         | 04-1111111 | Theresa | Markham |
   | 21         | 04-1111111 | Theresa | Markham |
   | 22         | 04-2222222 | Paula   | Roberts |
   | 23         | 04-3333333 | Theresa | Markham |
   | 24         | 04-4444444 | John    | Blake   |
   +------------+------------+---------+---------+
   5 rows in set (0.00 sec)
6. Finally let’s do an outer join example. Say we want to retrieve all of the account IDs and include the business name when the account is linked to a business customer. If we do an inner join we get the following

```
mysql> SELECT a.account_id, b.cust_id, b.name
    -> FROM account a
    -> JOIN business b ON a.cust_id = b.cust_id;
```

```
+------------+---------+------------------------+
| account_id | cust_id | name                   |
+------------+---------+------------------------+
| 20         | 10      | Chilton Engineering    |
| 21         | 10      | Chilton Engineering    |
| 22         | 11      | Northeast Cooling Inc. |
| 23         | 12      | Superior Auto Body     |
| 24         | 13      | AAA Insurance Inc.     |
+------------+---------+------------------------+
5 rows in set (0.24 sec)
```

Results are only returned for cases where the account is a business account. However, what we want is a list of all accounts, including the business name if the account is linked to a business.

7. The solution is to use an outer join as follows

```
mysql> SELECT a.account_id, b.cust_id, b.name
    -> FROM account a
    -> LEFT OUTER JOIN business b ON a.cust_id = b.cust_id;
```

```
+------------+---------+------------------------+
<table>
<thead>
<tr>
<th>account_id</th>
<th>cust_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>22</td>
<td>11</td>
<td>Northeast Cooling Inc.</td>
</tr>
<tr>
<td>23</td>
<td>12</td>
<td>Superior Auto Body</td>
</tr>
<tr>
<td>24</td>
<td>13</td>
<td>AAA Insurance Inc.</td>
</tr>
</tbody>
</table>
+------------+---------+------------------------+
24 rows in set (0.02 sec)
```

The resulting table contains the business name when the account is linked to a business, and NULL values otherwise.
11.1.4 Walkthrough 4: Creating a new database

In this walkthrough we look at how to create a database from scratch, rather than importing one that has already been created. Creating a database from scratch involves creating the individual tables and then populating them with data. For this exercise we have the following tables containing information about our friends

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Allowable Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>person_id</td>
<td>Smallint (unsigned)</td>
<td></td>
</tr>
<tr>
<td>first_name</td>
<td>VARCHAR(20)</td>
<td></td>
</tr>
<tr>
<td>last_name</td>
<td>VARCHAR(20)</td>
<td></td>
</tr>
<tr>
<td>gender</td>
<td>CHAR(1)</td>
<td>M, F</td>
</tr>
<tr>
<td>dob</td>
<td>Date</td>
<td></td>
</tr>
</tbody>
</table>

Table 11.1: Person table

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Allowable Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>person_id</td>
<td>Smallint (unsigned)</td>
<td></td>
</tr>
<tr>
<td>food</td>
<td>VARCHAR(20)</td>
<td></td>
</tr>
</tbody>
</table>

Table 11.2: Favourite_food table

1. Firstly we need to create a new database

   ```sql
   mysql> CREATE database friends;
   mysql> USE friends;
   ```

2. We can now add the tables using the `CREATE TABLE` command. The `person` table:

   ```sql
   mysql> CREATE TABLE person (  
   2    -> person_id SMALLINT UNSIGNED AUTO_INCREMENT,  
   3    -> fname VARCHAR(20),  
   4    -> lname VARCHAR(20),  
   5    -> gender ENUM('M', 'F'),  
   6    -> dob DATE,  
   7    -> PRIMARY KEY (person_id)  
   8    -> );
   ```

   And the `favourite_food` table:

   ```sql
   mysql> CREATE TABLE favourite_food (  
   2    -> person_id SMALLINT UNSIGNED,  
   3    -> food VARCHAR(20),  
   4    -> PRIMARY KEY (person_id, food),  
   5    -> FOREIGN KEY (person_id) REFERENCES person (person_id)  
   6    -> );
   ```

3. **Question**: Why does the primary key in the `favourite_food` table consist of `person_id` and `food`?

4. We can check the tables have been created successfully by using

   ```sql
   mysql> SHOW TABLES;
   +-------------------+  
   | Tables_in_friends |  
   +-------------------+  
   | favourite_food    |  
   | person            |  
   +-------------------+  
   2 rows in set (0.00 sec)
5. We can now start adding data to the tables. We will insert our first person, William Turner, into the person table:

```sql
mysql> INSERT INTO person
    -> (person_id, fname, lname, gender, dob)
    -> VALUES (null, 'William', 'Turner', 'M', '1972-05-27');
```

and take a look at the results

```sql
mysql> SELECT * FROM person;
```

<table>
<thead>
<tr>
<th>person_id</th>
<th>fname</th>
<th>lname</th>
<th>gender</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>William</td>
<td>Turner</td>
<td>M</td>
<td>1972-05-27</td>
</tr>
</tbody>
</table>

1 row in set (0.01 sec)

6. Note that we don’t have to pass in a value for `person_id` because that column was set to auto increment when we originally defined the table. This avoids conflicts when entering data, since the `person_id` is the primary key and hence has to be unique.

7. Also note that dates are of the form `YYYY-MM-DD`. They can be entered as strings and will be converted to dates by MySQL.

8. Next we will add some of William’s favourite foods:

```sql
mysql> INSERT INTO favourite_food (person_id, food)
    -> VALUES (1, 'nachos');
```

```sql
mysql> INSERT INTO favourite_food (person_id, food)
    -> VALUES (1, 'pizza');
```

```sql
mysql> INSERT INTO favourite_food (person_id, food)
    -> VALUES (1, 'chips');
```

```sql
mysql> SELECT * FROM favourite_food;
```

<table>
<thead>
<tr>
<th>person_id</th>
<th>food</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>chips</td>
</tr>
<tr>
<td>1</td>
<td>nachos</td>
</tr>
<tr>
<td>1</td>
<td>pizza</td>
</tr>
</tbody>
</table>

3 rows in set (0.01 sec)

9. Finally we’ll add another person and some of her favourite food.

```sql
mysql> INSERT INTO person (person_id, fname, lname, gender, dob)
    -> VALUES (null, 'Susan', 'Lo', 'F', '1975-11-02');
```

```sql
mysql> INSERT INTO favourite_food (person_id, food)
    -> VALUES (2, 'Chicken');
```

---

1In this exercise we will add a small amount of data manually. Later you will learn how to automate this process with Python.
10. Data in a table can be modified using the `UPDATE` statement. For example, to change William’s date of birth we could do

```sql
mysql> UPDATE person
    -> SET dob = '1977-12-03'
    -> WHERE person_id = 1;
```

Query OK, 1 row affected (0.24 sec)
Rows matched: 1 Changed: 1 Warnings: 0

and now look at the updated table:

```sql
mysql> SELECT *
    -> FROM person;
```

```
+-----------+---------+--------+--------+------------+
<table>
<thead>
<tr>
<th>person_id</th>
<th>fname</th>
<th>lname</th>
<th>gender</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>William</td>
<td>Turner</td>
<td>M</td>
<td>1977-12-03</td>
</tr>
<tr>
<td>2</td>
<td>Susan</td>
<td>Lo</td>
<td>F</td>
<td>1975-11-02</td>
</tr>
</tbody>
</table>
+-----------+---------+--------+--------+------------+
2 rows in set (0.00 sec)
```

11. You can also remove an entry entirely, using the `DELETE` statement:

```sql
mysql> DELETE FROM person
    -> WHERE person_id = 2;
```

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails ('friends/favourite_food', CONSTRAINT 'favourite_... FOREIGN KEY ('person_id') REFERENCES 'person' ('person_id'))
```

This is a demonstration of the power of foreign key constraints. By trying to remove the row for Susan Lo in `person`, we would have made the row (2, 'Chicken') in the `favourite_food` table refer to an entity that no longer exists.

12. We must first delete the offending row(s) in `favourite_food` and then we can delete Susan Lo from `friends`. We can do this as two separate `DELETE` statements:

```sql
mysql> DELETE FROM favourite_food WHERE person_id = 2;
mysql> DELETE FROM person WHERE person_id = 2;
```

13. Now the `person` table only contains William

```sql
mysql> SELECT *
    -> FROM person;
```

```
+-----------+---------+--------+--------+------------+
<table>
<thead>
<tr>
<th>person_id</th>
<th>fname</th>
<th>lname</th>
<th>gender</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>William</td>
<td>Turner</td>
<td>M</td>
<td>1977-12-03</td>
</tr>
</tbody>
</table>
+-----------+---------+--------+--------+------------+
1 row in set (0.00 sec)
```

and the `favourite_food` table only contains information about William.

In this walkthrough we have covered the 4 basic statements required for setting up tables in a database: `CREATE, UPDATE, INSERT, DELETE`. You will need to use these when creating your database for the major project.
11.2 Exercises

11.2.1 Question 1: Basic queries

Construct queries to retrieve the following from the bank database:

1. All accounts opened in 2002.
2. The customer ID of all customers with a savings account.
3. A list of the years in which a new employee started at the bank.
   Hint: the builtin function YEAR('2002-01-20') returns 2002.
4. All non-business customers whose last name ends in 'y'.
5. A list of employees who are no longer working at the bank.

11.2.2 Question 2: Joining tables

Construct queries to retrieve the following from the bank database:

1. A list of employees and the name of the branch that they work in.
2. The account ID for each non-business customer along with the customer’s federal ID and the name of the product on which the account is based.
3. A list of all employees whose supervisor is assigned to a different department.
4. A list of all product names along with the accounts based on that product. Include all products, even if no accounts have been opened for that product.

11.2.3 Question 3: Creating tables

Part I Add a pets table to the friends database that we created in Walkthrough 4. It should contain the following information:

- Owner ID
- Pet ID
- Pet name
- Pet type (e.g. dog, cat)
- Age of the pet

The first step is to choose an appropriate type for this information. Remember the table has to be linked to the person table using the Owner ID as a foreign key.

Once you have added the table, populate it with a couple of pets belonging to William or Susan.

Part II Next add a vetvisits table which should have a list of all of the visits that each pet has made to the vet. Each visit should have the following information:

- Pet ID
- Surgery name
- Date of visit

and should be linked to the pets table using the Pet ID as the foreign key.

Again, populate the table with a couple of vet visits for one of the pets you created in Part I.
11.3 Capability checklist

When you’ve finished this lab, check that you know how to:

1. Import a database into MySQL
2. Explore the data using commands like DESC, SHOW
3. Run basic SQL queries using select
4. Filter results using where
5. Join multiple tables in a query
6. Understand how to select and avoid Null values
7. Create new tables and populate them

11.4 Appendix: MySQL data types

These are the MySQL data types that you should choose from when designing and creating a new database table. It is not a comprehensive list, but includes examples of all of the main types.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR()</td>
<td>A fixed-length string from 0 to 255 characters</td>
</tr>
<tr>
<td>VARCHAR()</td>
<td>A variable-length string from 0 to 255 characters</td>
</tr>
<tr>
<td>TEXT</td>
<td>A string with a maximum length of 65 535 characters</td>
</tr>
<tr>
<td>MEDIUMTEXT</td>
<td>A string with a maximum length of 16 777 215 characters</td>
</tr>
<tr>
<td>TINYINT()</td>
<td>-128 to 127 normal; 0 to 255 UNSIGNED</td>
</tr>
<tr>
<td>SMALLINT()</td>
<td>-32768 to 32767 normal; 0 to 65535 UNSIGNED</td>
</tr>
<tr>
<td>INT()</td>
<td>-2147483648 to 2147483647 normal; 0 to 4294967295 UNSIGNED</td>
</tr>
<tr>
<td>FLOAT</td>
<td>A floating point (real) number</td>
</tr>
<tr>
<td>DOUBLE()</td>
<td>A double precision float</td>
</tr>
<tr>
<td>DATE</td>
<td>YYYY-MM-DD</td>
</tr>
<tr>
<td>DATETIME</td>
<td>YYYY-MM-DD HH:MM:SS</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>YYYYMMDDHHMMSS</td>
</tr>
<tr>
<td>TIME</td>
<td>HH:MM:SS</td>
</tr>
<tr>
<td>YEAR</td>
<td>YYYY</td>
</tr>
<tr>
<td>ENUM()</td>
<td>A list of possible values</td>
</tr>
</tbody>
</table>