Objectives:

Upon completing this exercise, you will know how to write more complex database queries whose results are refined by the WHERE clause, sorted in the way you specify with the ORDER BY clause, and grouped into categories with the GROUP BY and HAVING clauses.

PART I – Introductions and Definitions

During your work with the exercises of Lab 2, you saw how messy column headings can sometimes be. Wouldn't it be nice to be able to display column headings that mean something to you (and your boss!), not just to the DBA who created the tables? There is! With column aliases you can display column headings any way you like.

Your work with the exercises of Lab 2 also illustrated the power of the WHERE clause to refine and limit the content of your queries. There are three more clauses you need to know about:

- ORDER BY – Will sort the information, based on one or more columns, before displaying it.
- GROUP BY – Will produce one summary row for all selected rows that contain identical values in one or more of the specified columns.
- HAVING – Is used to determine which groups the GROUP BY will include.

PART II – Syntax and Usage

Column Aliases It is easy to display a column with the heading of your choice using a column alias. Just remember that aliases are just that: the real column names defined for table do not change. So, let's look at how easy it is to use column aliases with the following exercises.

1. Establish an SQL session to your ict4300 database as student
2. Enable spooling: tee <path to your folder>/filename
3. Select the customer's last name, customer ID and organization ID from the customers table. A lot of space is wasted with long, clumsy column headings, right?
4. Now specify an alias for each of those column names as follows:

```
SELECT last_name lname, customer_id cust, organization_id org
FROM customers;
```

You see the alias in bold type, separated from the real column name by a space. Don't make the mistake of separating the alias from the real column name with a comma, or MySQL will interpret your alias for a real column name, and the statement will fail. That's all there is to using column aliases!

WHERE Clause

You saw the WHERE clause in Lab 2, so let's review it briefly. WHERE allows you to refine the output of your queries to those records (rows) containing only the information specified in the WHERE clause. For example, to display only the customers (from the column alias example) whose last name is 'Smith', we should instruct the WHERE clause to limit output:

```
SELECT last_name lname, customer_id cust, organization_id org
FROM customers
WHERE last_name = 'Smith';
```

You also saw that the AND and OR modifiers can be used to expand the meaning of WHERE, such as:

```
SELECT last_name lname, customer_id cust, organization_id org
FROM customers
WHERE last_name = 'Smith'
AND first_name = 'TOM';
```

ORDER BY Clause

The ORDER BY clause allows your output to be sorted on the columns you specify. You may have noticed that MySQL sometimes displays the contents of its columns (both text and numbers) in an apparently random fashion. This is because MySQL doesn't require data to be sorted prior to being inserted into tables. This is beneficial, of course. Imagine order entry personnel entering new orders for your company's product into some new_orders table. You couldn't expect that information to arrive, throughout the day, to those people in a sorted form. But you can use ORDER BY to prepare a nicer, more usable database report.

ORDER BY works either in an ascending (increasing) or descending (decreasing) fashion, per your specification. It is important to remember that ORDER BY works on the basis of an 'ASCII sort'. For the unfamiliar, this can produce some unexpected results. ASCII sorts read one character at a time, proceeding from left to right, and different levels of precedence are assigned to upper and lower case characters and numbers.

For example, an ORDER BY (or other ASCII sort) of:
[abc, ghi, def] would produce: [abc, def, ghi]

But, an ORDER BY of:
[abc, DEF, ghi] would produce [DEF, abc, ghi]

because the upper case characters have a higher precedence than the lower case characters do. Further, numbers have a higher precedence than upper case characters.

The syntax is simple enough: ORDER BY <column name>. If you want to sort by more than one column, the column names must be separated by a comma. If neither an ascending nor descending sort is specified, MySQL will default to ascending order. To specify the sort direction, include the ASC or DESC keyword after the column name, as in: ORDER BY <column name> ASC or ORDER BY <column name> DESC.

Since you are still connected as student, try the following exercise:

1. Select the first name and last name of each customer from the customers table.

2. Now, select the first name and last name of each customer, but ORDER BY last name.

<u>GROUP BY Clause</u>

Sometimes it makes sense to group the output of a query into logical categories. The GROUP BY clause works well in many situations when you might wish to report calculations on data according to some group or category. For example, consider the shipment_items table of student, and imagine that your manager has come to you for information regarding the minimum, maximum and average weight of items shipped, and needs that information paired against actual shipments recorded in the database. Here's a simple way to meet that need:

1. Describe the shipment_items table.

2. Now see what you have to work with by typing:

   ```
   SELECT shipment_id, weight
   FROM shipment_items;
   ```

3. You notice that the shipment_id is occasionally repeated, indicating that there are sometimes multiple packages, each with a weight recorded, which are associated with each shipment_id. Using GROUP BY in this scenario will provide exactly what you need. See for yourself by executing the following:

   ```
   SELECT shipment_id, min(weight), max(weight), avg(weight)
   FROM shipment_items
   GROUP BY shipment_id;
   ```

Upon executing the code in the above statements, you will find that MySQL has shouldered the heavy work of calculations for you, and has presented you with an (almost) attractive report for your manager. You will likely also realize that, with small tables like the ones you work with in class, you may be able to meet your manager's needs quickly enough by performing the calculations by hand. The strength of GROUP BY and other clauses, and of MySQL's line functions (you saw some others during Lab

2) is many times best revealed when you work with full enterprise-scale tables with millions of rows!

HAVING Clause

The HAVING clause is sometimes confusing to new SQL programmers. However, it follows the same logic considered for the WHERE clause. In other words, it refines or filters the output generated from a GROUP BY clause operation. In fact, it might be helpful for you to think of HAVING as being the WHERE clause for the GROUP BY operation.

Even when the refining capabilities of the WHERE and GROUP BY clauses is used, the output resulting from your queries can still be too general for your requirements. For example, let's go back to the SELECT statement used in item (3) of the GROUP BY section. We will focus the example a bit to suit our needs here by only considering the average weight. The statement then becomes:

```
SELECT shipment_id, avg(weight)
FROM shipment_items
GROUP BY shipment_id;
```

Imagine that your manager has requested this information, but is only interested in seeing the records when the average weight is 15 pounds or more. Use the HAVING clause to filter the information obtained from the GROUP BY clause by executing the following statement:

```
SELECT shipment_id, avg(weight)
FROM shipment_items
GROUP BY shipment_id
HAVING avg(weight) >= 15;
```

Notice that the HAVING clause must reference a column being acted upon by a group function and which is SELECTed in the query. This makes sense when you think about it: why try to use HAVING to alter the output of a regularly SELECTed column? That is what the WHERE clause is for.

PART III – Using the Clauses Together

If you've stopped to think about it, you will have realized that the use of clauses to modify your SELECT statements is optional. In other words, a SELECT statement that lists existing columns coming FROM an existing table will produce results. We use WHERE, GROUP BY, and HAVING to further refine and restrict our output. When the optional clauses are used, MySQL has some rules about their placement in the statement. Essentially, WHERE (if used) must follow the FROM keyword; WHERE may also be modified by AND and OR. GROUP BY (if used) must follow WHERE (if used) or FROM. HAVING (if used) must follow GROUP BY. The following structure illustrates a complex SQL statement that makes use of all clauses and modifiers:

```
SELECT ....... <Required>
FROM ......... <Required>
WHERE ........ <Optional>
AND  ......... <Optional>
OR ........... <Optional>
GROUP BY ..... <Optional>
HAVING ....... <Optional>
ORDER BY ..... <Optional>
```

PART IV – Exercises

Let’s try a few queries with STUDENT’s shipment table using the new clauses you’ve been introduced to.

1. Write a statement to display the shipment ID, city and state of all the destination, and the shipping costs of all shipments destined for Salt Lake City, Utah.

2. Repeat Exercise (1), but include appropriate (your choice) column aliases.

3. Write a statement to display the city of destination and the average cost of shipping to each city of destination.

4. Repeat Exercise (3), but sort the displayed cities in descending order.

5. Write a statement to display the city of destination, the shipping company and the estimated shipping date for all UPS shipments.

6. Repeat Exercise (5), but further restrict your output to estimated shipping dates occurring after January 20, 2001. (Hint: your date restriction will require a format identical to the one displayed from Exercise (5), and it must be in single quotes – for example: ’05-JAN-01’).

7. Write a statement to display the shipment ID, average weight and average shipping cost of all shipment items. Group your output by the shipment ID.

8. Repeat Exercise (6), but restrict your output to records having average shipping costs in excess of $10.00.

PART V – Ending Your Session

End your SQL session by typing: `exit;`