Objective:

Using basic database queries (by SELECT) to learn about tables and the data they contain.

PART I – A Brief Review of SELECT Statement Syntax

In Lab #2A you learned some of the most basic uses of SELECT statements.

After logging onto the database as one of those users, you can display a list of their tables by typing:

```
SHOW tables;
```

And, you can describe the contents of those tables by typing:

```
desc <table name>;
```

where <table name> is the table's name, as indicated by the above SELECT statement.

You also know the basic syntax of an SQL SELECT statement to include:

```
SELECT <column name>
FROM <table name>;
```

where <column name> is a single column or a list of column names, each separated by a comma, and <table name> is the table's name. However, what if you want to display the information in ALL the columns of a table?

PART II – Displaying All Information in a Table

1. Log onto the ict4300 database as STUDENT.

2. Enable spooling to save your output. (use "tee")

3. Display a list of STUDENT's tables.

4. Describe the STUDENT's **invoices** table.

5. Write a SELECT statement to display each column in the **invoices** table.

This is a relatively simple example, but some tables have many more columns than this.

Fortunately, there is a shortcut for obtaining the same result. Execute the following statement:

```
SELECT *
FROM invoices;
```

6. Inspect the results of the SELECT * query to those you obtained by typing each column name. Are they identical? The '*' is a *wildcard* variable that indicates to Oracle you want data from every column in the table. Using a '*' to identify all columns in a table is sometimes a handy way to display all the information in a table, and is sometimes referred to as a *full-table scan*.

7. Perform full-table scans of each of STUDENT's six tables.

PART III – Refining Information for Display with the 'WHERE' Clause

In Part II, you asked for and got all the information in each table. As you will likely agree, this can sometimes be an instructive approach for small tables. However, and more often, the amount of information displayed with a full-table scan is simply overwhelming and of little real value.

There are several ways to limit output from a query to match only what you need, and to exclude that which you do not care to see. The first of these techniques will involve use of the 'WHERE' clause. The 'WHERE' clause is added to the end of a simple query written against STUDENT's organization table, such as:

```
SELECT *
FROM organizations
WHERE organization_id = 100;
```

The above query would display all rows WHERE the value for organization_id (a column name) is equal to 100. The logical modifiers 'AND' and 'OR' can also be used to make WHERE queries deliver inclusive or exclusive information, as in:

```
SELECT *
FROM organizations
WHERE organization_id = 100
AND name = 'MOBILE MEDIA';

SELECT *
FROM organizations
WHERE organization_id = 100
OR name = 'E*BOOKS';
```

Because you are still logged onto the database as student, execute both of the above queries. Briefly describe the results. Specifically, describe the relationships between WHERE and its AND or OR modifiers. Are the results what you expected?

PART IV – Conditional Modifiers of WHERE

MySQL honors a number of conditional modifiers of WHERE, including:

= Equals (as already seen)!= Not Equal to (<> and ^= are also allowed operators for Not Equal) < Less Than> Greater Than<= Less Than or Equal To>= Greater Than or Equal To

So, the statement

```
SELECT organization_id
FROM organization
WHERE organization_id >= 102;
```

would return all values for organization_id greater than or equal to 102.

PART V – Line Functions and Dates

MySQL provides many built-in capabilities for handling commonly need functionality. We will describe and work with several here – there are many others, and you should consult a suitable reference. In general, the syntax of their usage is simple:

SELECT *function* (column_1, column_2, ..., column_n)

Several of the most common line functions are:

nvl – Displays a value or string in place of a null (that is, a missing value).

sum – Displays an arithmetic sum of the values represented in two or more columns with numeric data types (also works with actual numbers selected against the 'dual' table).

count – Displays the number of occurrences.

cower – Returns text fields in lower case letters.

upper – Returns text fields in upper case letters.

min, max – Returns the lowest and highest numbers from the column passed,

respectively.

round – Rounds a numeric field (or list of actual numbers from 'dual') to the number of decimal places indicated right of the comma (but will not increase precision). For example,

`SELECT round(2.2+3.33, 1) FROM dual;` will result in 5.5

`SELECT round(2.2+3.3, 2) FROM dual;` will result in 5.

`SELECT round(2.2+3.3, 0) FROM dual;` will result in 6.

truncate – Removes a number of digits specified right of the comma. For example,

`SELECT truncate (2.2+3.3, 1) FROM dual;` will result in 5.5

`SELECT truncate (2.2+3.3, 2) FROM dual;` will result in 5.50

`SELECT truncate (2.2+3.3, 0) FROM dual;` will result in 5. **Note: This is NOT rounding.**

length – Displays the number of characters in a column for character data types.

distinct – Displays each unique entry only once.

Several line functions specific to DATE data types are also provided, including:

add_months – Returns a date equal to the input date, plus the number of months right of the comma.

months_between – Returns the number of months between two input dates. May be a decimal value.

sysdate – Returns the current date from your local machine.

PART VI – Problems

NOTE: You will need to spend some time reviewing the table and getting to know the data. The questions are written from a user point of view and the end users may never know where things are store. This is an important exercise in learning to communicate with end users and then translate that discussion into technical code...

Refer to the list of tables owned by STUDENT. See Lab 2A for help to solve the following problems.

If you are not already logged on as student, do so.

1. Write an SQL statement to display the tables owned by STUDENT.
2. Write an SQL statement to display the number of tables owned by STUDENT.
3. Write a statement to display the column names, null characteristics and data types of STUDENT's cost_table table.
4. Write a statement to display the column names, null characteristics and data type of STUDENT's shipment table.
5. Perform a full-table scan on STUDENT's invoices table.
6. Write an SQL statement to display each invoice date for customer 103 from STUDENT's invoices table. Display both the customer ID number and invoice date.