CSC 3300 Homework 4 – SQL Scripting

Description

Your assignment is to create SQL scripts. These scripts will work on the Firebird employee database.

You will be creating 4 script files, one for each one of the problems below. When grading your script files, I will submit them to my Firebird database server via the following command:

```
isql-fb -user SYSDBA -password coursework
"localhost:/home/coursework/employee.fdb" < problemX.sql
```

Note in "problemX.sql", the X will be replaced with a 1, 2, 3, or 4 for each of the problems below. In other words, you will create files named problem1.sql, problem2.sql, problem3.sql, and problem4.sql. Turn them in zipped up in one file called hw4.zip

The Firebird syntax is different from the syntax given on the slide in class (the SQL standards). Therefore, you will need to look in the Firebird manual to learn the syntax differences. Here are some links that will help you on this assignment:

- The PDF Interbase 6 user manual in a zip file
- Other Firebird references, including docs describing the changes in Firebird for 2.0 and 2.5.
- A web site by Janus Software that is a good HTML resource (Firebird 2.0)

Preparation

In this assignment, you will be creating a table and then using stored procedures and triggers to manipulate it. To make it easy for you to test your scripts, execute the following code (either using isql-fb or Flamerobin) once you have connected to the employee database.

```
set term # ;
create procedure drop_if_exists(thing varchar(50),
                                name varchar(50))
as
-- don't need to declare anything here
begin
        execute statement 'drop ' || thing || ' ' || name;
        when any do
        begin
        end
end #
commit work #
```

Executing the above code will create a stored procedure. The stored procedure will drop database objects without throwing an exception if the object does not exist. You can make calls to it at the top of your scripts to drop tables, procedures, and triggers before you (re)create them. Therefore, you can run your scripts multiple times without Firebird complaining that your tables, triggers, and procedures are already defined. You call it like the following in Firebird:

```
execute procedure drop_if_exists('procedure', 'f_class');
execute procedure drop_if_exists('trigger',
                                 'tr_salary_class_update');
execute procedure drop_if_exists('trigger',
                                 'tr_salary_class_insert');
execute procedure drop_if_exists('trigger',
                                 'tr_salary_class_delete');
execute procedure drop_if_exists('table', 'salary_class');
```

Note that the arguments to the stored procedure are strings (varchar). The first parameter is the type of object to drop (must be valid SQL syntax for the corresponding drop statement, i.e. 'trigger' for a 'drop trigger', 'procedure' for 'drop procedure', and 'table' for 'drop table').

Also, before you begin problem 1, create a table called department_tester. You will make modifications to the department_tester table so that we do not destroy the original employee table. So, execute the following SQL command (either in isql-fb or Flamerobin):

```
CREATE TABLE DEPARTMENT_TESTER
(
    DEPT_NO DEPTNO NOT NULL PRIMARY KEY,
    DEPARTMENT VARCHAR(25) NOT NULL UNIQUE,
    HEAD_DEPT DEPTNO,
    MNGR_NO EMPNO,
    BUDGET BUDGET DEFAULT 50000,
    LOCATION VARCHAR(15),
    PHONE_NO PHONENUMBER DEFAULT '555-1234',
    FOREIGN KEY (HEAD_DEPT) REFERENCES DEPARTMENT (DEPT_NO),
    FOREIGN KEY (MNGR_NO) REFERENCES EMPLOYEE (EMP_NO)
);

CREATE DESCENDING INDEX BUDGETTESTERX ON DEPARTMENT_TESTER (BUDGET);
```

Problems

1. Create a SQL script called problem1.sql that creates a table called department_rating with two columns. The first column is named rating and is of type varchar(5), and the second column is called occurrences and is of type int. Next, insert values for ULTRA, HIGH, MID, and LOW, all with the occurrences column set to 0. Next, define a helper function that you will use in the next problem. The function is called f_rating. It takes has one parameter called budget. The function should return the string 'LOW' for a budget less than or equal to 500000, 'MID' for a budget less than or equal to 850000, 'HIGH' for a budget less than or equal to 1200000, and 'ULTRA' for a budget above 1200000. Show that you created the department_rating table correctly by including a query in your script that shows all the rows in the table. Also demonstrate your function by including a query that calls the function for each of the following salaries: 200000, 850000, 980000, and 1200001.

Notes:

- Due to how Firebird implements transactions, once you create the salary_class table, you will have to commit the transaction before you execute the insert statements.

- You may need to change the statement terminator (see the definition of drop_if_exists above for an example, or the definition of f_ten_percent_raise below), so that you can use the semi-colon to separate your statements within your trigger definition.
- Unfortunately, Firebird does not implement functions according to the standard (surprise, surprise). Instead, Firebird stored procedures can return table values. Here is an example:

```
set term # ;

create procedure f_ten_percent_raise(salary SALARY)
returns (amt SALARY) as
begin
  amt = salary * 1.10;
  suspend;
end #

set term ; #
```

The above store procedure returns a table with only one row and one column called amt. The suspend statement acts like a return in C/C++. Invoke the procedure as follows:

```
select amt from f_ten_percent_raise(50000);
```

- The syntax for the IF statement differs from the standard. Following is an example:

```
if (expression) then
  statement1;
else
  statement2;
```

If you have more than one statement in the if or else, then use begin...end.

2. Create a script called problem2.sql, and in the script define a trigger called tr_department_rating_insert that, when a new employee is inserted into the department_tester table, updates the corresponding occurrences column in the department_rating table. For example, if a new department that has a 200000 dollar salary is entered into the department_tester table, the trigger should increment the occurrences column for the class 'LOW' budgets in the department_rating table by 1.
   To show that your script works, include, in your script, SQL commands to copy all the data from the department table into the department_tester table. Finally, include, in your script, a query that shows the department_rating table after all the rows from the budget table are copied.

Notes:

- Firebird trigger syntax differs slightly from the standard. An example follows:

```
set term # ;

create trigger trigger_name for table_name after insert
  -- replace trigger_name with the trigger name and
  -- replace table name with the table's name
as
  -- variable declarations go here
  -- syntax is: DECLARE variable_name type DEFAULT value;
begin
  -- below is an example for calling the stored procedure and
  -- putting the return value in a variable.
  select class from f_class(NEW.salary) into :new_class;
  -- Notice that the variable name is preceded by a colon when
  -- used in an SQL statement.
  -- Also notice that the INTO goes at the end of the
  -- select, unlike the standard.
  -- Finally, NEW holds value of the new row that was inserted
  -- into the table.
  -- delete triggers have an OLD variable that holds the value of
  -- the deleted row, and update triggers have both NEW and OLD.
  -- rest of trigger definition goes here
end #

set term ; #
```

3. Create an SQL script called problem3.sql that defines a trigger tr_department_rating_delete that, when a department is deleted from the department_tester table, updates the corresponding occurrences column in the department_rating table. For example, if a department that has a 200000 dollar salary is deleted from the department_tester table, the trigger should decrement the occurrences column for the class 'LOW' salaries in the department_rating table by 1.
   To show that your trigger works, delete the employees with department numbers 115 and 672, and execute a query that shows all the rows in the department_rating table.
4. Create an SQL script called problem4.sql that updates the department_rating table if a budget is changed (an update statement is executed on the department_tester table). The update trigger should be fairly easy because it does what both the delete and insert triggers do, and has OLD and NEW row variables that have the column values for the row's values before the update and after the update, respectively.
   Lastly, include, in your script, an SQL command that adds 160000 dollars to the budget for the department with the department number of 120. Finally, include an SQL query that shows the department_rating table.