Exercise 2. Create Objects

What This Exercise Is About

This exercise gives you an opportunity to implement a small data model.

What You Should Be Able to Do

At the end of the lab, you should be able to:

- Create tables with check constraints
- Create index
- Use referential integrity
- Create triggers
- Create views
- Understand the effect of CHECK OPTION in a view
- Appreciate some of the advantages of using views

Introduction

See the data model at the start of this Exercise Guide to get the column names and descriptions for each table.

OS/390 Users: Be sure to add the correct IN clause to your CREATE TABLE statements. Ask the instructor if you forgot which IN clause to use.

Required Materials

- Student handout
- SQL Reference

Exercise Instructions

Harvey needs your help to create a database for his test environment. He has accurately defined the requirements, but he does not know the SQL syntax. You should help him to do the subsequent steps.

Problem 1

Create the table EMPDEPT with these columns:

- EMPNO
- LASTNAME
- SALARY
- DEPTNO
- DEP_NAME

The data types and null characteristics for these columns should be the same as for the columns with the same names in the EMPLOYEE and DEPARTMENT tables. These tables are described in our course data model.

The definition of the table should limit the values for the yearly salary (SALARY) column to ensure that:

- The yearly salary for employees in department E11 (operations) must not exceed 28000.
- No employee in any department may have a yearly salary that exceeds 50000.

The values in the EMPNO column should be unique. The uniqueness should be guaranteed via a unique index.

Create the table HIGH_SALARY_RAISE with the following columns:

- EMPNO
- PREV_SAL
- NEW_SAL

The data type for column EMPNO is CHAR(6). The other columns should be defined as DECIMAL(9,2). All columns in this table should be defined with NOT NULL.

Problem 2

After creating the table, you should add referential constraints.

The primary key for the EMPDEPT table should be EMPNO.

The EMPDEPT table should only allow values in column EMPNO which exist in the EMPLOYEE table. If an employee is deleted from the EMP table, the corresponding row in the EMPDEPT table should also be immediately deleted.

The EMPDEPT table should only allow values in column DEPTNO which exist in the DEPARTMENT table. It should not be possible to delete a department from the DEPARTMENT table as long as a corresponding DEPTNO exists in the EMPDEPT table.

Note: In DB2 for OS/390 the used sample tables are originally defined as views. You can not have a foreign key reference a view. Ask your instructor for the name and the qualifier of the tables which should be referenced.

Problem 3

Klaus must update the yearly salaries for the employees of the EMPDEPT table. If the new value for a salary exceeds the previous value by 10 percent or more, Harvey wants to insert a row into the HIGH_SALARY_RAISE table. The values in this row should be the employee number, the previous salary, and the new salary. Create something in DB2 that will ensure that a row is inserted into the HIGH_SALARY_RAISE table whenever an employee of the EMPDEPT table gets a raise of 10 percent or more.

Problem 4

Now, you should insert data in the EMPDEPT table. Use the combined contents of tables EMPLOYEE and DEPARTMENT as the source for your data.

Did your insert work?

If not, correct your INSERT statement so that you get only rows which satisfy the check constraints on the EMPDEPT table.

Problem 5

Harvey wants to test the table-level check constraint on the EMPDEPT table.

Ethel Schneider works in the operations department. Her department number is E11, and her employee number is 000280. Try to set her yearly salary to the value of 30000. Does it work?

Problem 6

Harvey wants to see if the trigger works.

Elizabeth Pianka, whose employee number is 000160, has been given a raise. Set her yearly salary to 25000. Inspect the HIGH_SALARY_RAISE table to see if the trigger worked.

Problem 7

Create a view named VEMPPAY that contains one row for each employee in the company. Each row should contain employee number, last name, department number, and total earnings for the corresponding employee. Total earnings means salary plus bonus plus commission for the employee. Then, determine the average of the earnings for the departments by using the view you just created.

Problem 8

Create a view named VEMP1 containing employee number, last name, yearly salary, and work department based on your **TESTEMP** table. Only employees with a yearly salary less than 50000 should be displayed when you use the view.

Note: It is very important that you base this view on the **TESTEMP** table that was created for you or you created with the CRTABS member. Otherwise, you may get incorrect results in a later lab.

Display the rows in the view in employee number sequence.

Our employee with the employee number 000020 (Thompson) changed jobs and will get a new salary of 51000. Update the data for employee number 000020 using the view VEMP1.

Display the view again, arranging the rows in employee number sequence.

What happened? Is Thompson still in the view?

Query the row of employee number 000020 in your TESTEMP table.

Did the update work?

Problem 9

Reset the salary of employee Thompson (empno = '000020') to the value of 41250.

Create a view named VEMP2 which has the same definition as in problem 8, but add a CHECK OPTION. Again, base the view on your **TESTEMP** table.

Display the rows in the view in employee number sequence.

Our employee with the employee number 000050 (Geyer) also changed jobs and will have a new salary of 55000. Update the data for employee number 000050 using the view VEMP2. Does the UPDATE statement work?

Display the view again, arranging the rows in employee number sequence.

Query Geyer's row in your TESTEMP table.

Did the data in the base table change?

END OF LAB