PURDUE UNIVERSITY

PURDUE POLYTECHNIC INSTITUTE
Department of Computer and Information Technology

CNIT 27200: Lab #8

25 pts

Due Date

- **Part A** is due **via Blackboard on Wednesday, April 8th, by 11:59pm**. 10 pts.
- **Part B** is due **via Blackboard on Monday, April 13th, by 11:59 p.m.** 15 pts.

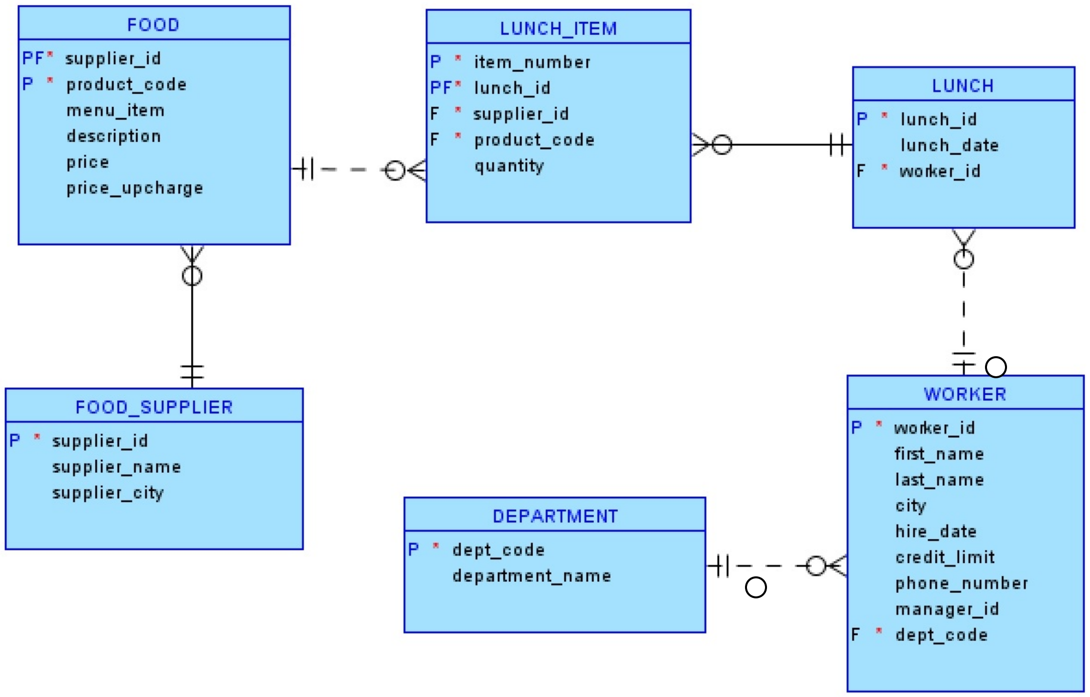
PART A (10 points)

SPECIFIC LEARNING OBJECTIVES:

- Be able to change existing data in a database with the UPDATE SET command
- Be able to add new rows to a database with the INSERT INTO command
- Be able to delete existing rows from a database with the DELETE FROM command
- Be able to populate a new table from an existing table
- Understand transactions and be able to use the COMMIT and ROLLBACK commands appropriately

Instructions:

1. Complete the following questions using SQL Developer.
2. Show the SQL statement that performs the data manipulation.
3. In most cases you are requested to **execute a SQL query before and after** the data change to verify that the change took place.
4. As instructed, **include the pre and post SQL results** for verification along with the SQL that updates the data.
5. **Pay particular attention to whether COMMIT's and ROLLBACK's are requested**.
6. Your PART A answer template should follow a standard format, to be compiled in a .sql file with your name in the file as **studentnameLab8A.sql**. This is for saving your FINAL SQL statements and the output. Fill in the header information and insert the question number to separate the responses – we will be using comments to store all our non-statement content such as question numbers, results, etc.
7. You can comment in SQL by beginning the line with a --
8. Alternatively, for multi-line comments, surround the block by /* and */.



Use the Lunches Database for this in-lab activity. The sql script is provided in Blackboard. You may want to rerun the Lunches script at the end of part A.

Question 1. Add new food supplier and food item records to the database.

Step 1. First confirm that there is no food supplier in the database with a supplier ID of **Sky**. Run a SQL statement in the FOOD_SUPPLIER table and filter by the supplier_id of **Sky**. (should be no rows selected)

Step 2. Add a new food supplier to the FOOD_SUPPLIER table with the following details:

Supplier ID	Sky
Supplier_name	Sky Rise Market
Supplier_city	Chicago

<u>Step 3</u>. Add a new food item to the FOOD table with the following details:

Supplier ID	Sky
Product Code	Sp
Menu Item	2
Description	Broccoli Soup
Price	4.75
Price Upcharge	.6

<u>Step 3.</u> Create a new food item of your own for supplier ID **Sky**.

<u>Step 4</u>***.*** **Verify the new food items** by selecting supplier ID, supplier name, product code, and price for items with a supplier ID of **Sky**. Use the FOOD and FOOD_SUPPLIER tables. (2 rows selected)

Question 2.

<u>Step 1</u>. List the supplier ID, Supplier Name and Supplier City for the supplier ID **Hsd**. (FOOD_SUPPLIER table) 1 row

<u>Step 2</u>. The supplier Hsd has changed owners and moved the business. The new supplier name is **Hanover Market**, and the supplier city is new **Skokie**. Update the database by changing the name and city for supplier ID Hsd.

<u>Step 3</u>. Issue a COMMIT.

<u>Step 4</u>. To verify the update, list the supplier ID, Supplier Name and Supplier City for the supplier ID **Hsd***.* 1 row

Question 3.

<u>Step 1</u>. Run a query that lists the supplier ID, description, and price for each soup item on the lunch menu. The soup items are designated with a product code of **Sp** (FOOD table). (5 rows selected)

<u>Step 2</u>. It is determined that the price of soup must be increased on the menu. Increase the price by **2** additional dollars for all of the soup items selected above.

(**DON'T** Issue a COMMIT here.)

<u>Step 3</u>. To verify, rerun the query from step 1, and verify that the price values are now increased by 2 dollars. (5 rows)

Question 4.

Part A: Create a new lunch record for worker ID **231**.

Step 1. Run a SQL statement and search for worker ID **231** in the LUNCH table. (2 rows selected)

Step 2. Add a lunch to the LUNCH table with the following details:

Lunch ID	48
Lunch Date	12-JAN-2020
Worker ID	231

Step 3**.** **Verify the new lunch** by selecting all columns for worker ID 231 in the LUNCH table*. (3 rows selected)*

Part B: What lunch items did worker ID 231 order for Lunch ID 48 on January 12th, 2020?

1. Create 3 records in the LUNCH_ITEM table. Your choice of lunch items.
2. Look at the SQL script for the CreateLunch database for ideas. Go to the insert statements for the FOOD table. What food is stored in the database? Go to the LUNCH_ITEM table. What is the Item Number?
3. Add three items for Lunch ID 48
4. To verify, show worker ID, lunch ID, lunch date, item number, supplier_ID, Product Code and quantity for Worker ID 231 (you will have to run an inner join with LUNCH and LUNCH_ITEM) (8 rows selected)

NOW, Issue a **Rollback**;

Part C: Run queries for each item (and also report output results):

- Verify that the 3 lunch items for worker ID 231 were deleted.
- Verify that the 1 lunch (48) was deleted.
- Verify that the change in Question 3 was also rolled back. *(3 records, all prices back to original amounts)*
- Verify that the change made in Question 2 was NOT rolled back
- Verify that the added food item was NOT rolled back in Question 1

Question 5.

Step 1. List the Supplier ID, Supplier name of the food suppliers with no food items (hint: use NOT IN with a nested subquery). (4 records selected)

Step 2. Remove these suppliers from the database. (4 records deleted)

Step 3. Verify that they have been removed by rerunning step 1.

Step 4. Issue a **COMMIT** statement.

Step 5. Run a query that lists the food suppliers left. Include the supplier ID, supplier name, and supplier city. (11 rows selected)

Question 6.

Step 1: Create a new table called **EmpComp** for workers that take clients out to lunch or dinner as part of their job. This table explains the columns and constraints found in the table you are creating:

Column Name	Datatype	Constraint
Worker_ID	Char(3)	Primary key includes Worker_ID and Comp_Date
Comp_Date	Date	
Comp_Limit	Number(5,2)	
Participants	VarChar2(100)	

Step 2: Alter the EmpComp table by adding a foreign key constraint. The EmpComp table references the Worker table via the Worker_ID.

Step 3: Using data from the Worker table for only the employees that work in Sales (**Sal**), insert rows into EmpComp setting the credit limit at 40% higher than what is stored in the Worker table. Set the Comp_date to the system date (SYSDATE). Hint: Review how to use INSERT INTO with a select statement, and also use the calculation for the credit limit: (Credit_Limit + Credit_limit*.40).

Show the CREATE TABLE and INSERT statements and Empcomp data. (2 rows inserted). Issue a **COMMIT**;

Step 4: Verify the new table. Run a query that joins the Worker table and the Empcomp table. List the worker id, the department code, credit limit, and the increased comp. limit. 2 rows.

PART B (15 points)

SPECIFIC LEARNING OBJECTIVES:

- Be able to change existing data in a database with the UPDATE command
- Be able to add new rows to a database with the INSERT INTO command
- Be able to delete existing rows from a database with the DELETE FROM command
- Be able to populate a new table from an existing table
- Understand transactions and be able to use the COMMIT and ROLLBACK commands appropriately

❒ **Using the Answer Template:**
Modify your template to include your name, and the remaining header data. Use SQL to answer each question or request for information and copy the statement **and** the results into the Answer file. For each question, your answer must follow these instructions:

Instructions:

1. Use same answer template from previous labs. Check the formatting of your **studentnameLab8B.sql** file. If the outputs are wrapping around, reduce the margins and use font size when necessary. To line up the output columns, change the font of the whole document to Courier New. Please use a simple text editor instead of MS Word due to formatting.
2. Part B utilizes the Eagle Database.
3. Complete the following questions using SQL Developer.
4. Show the SQL statement that performs the data manipulation.
5. As required in the instructions, **execute the required SQL before and after the data change to verify that the change took place**.
6. As instructed, **include the pre and post SQL results** for verification along with the SQL that updates the data.
7. **Pay particular attention to whether COMMIT's and ROLLBACK's are requested**.
8. It is your responsibility to show all documentation on how you achieved each step. **Provide as much documentation and explanation of your steps as possible**.

SHOW ALL OF YOUR WORK FOR EACH STEP IN PART B

As a reminder, this is an individual assignment.

Do your own work on your own computer. Do not share work.

Question 1:

A. Customer **Jennifer Kmec** has updated her contact information.
B. Generate a result set with the customerid, companyname, address, city, state, postal code, and email address BEFORE the change for just customer **Jennifer Kmec**.
C. Make the following changes to her customer record:
   a. New company name – **York Ballet School**
   b. New address – **492 Harold Blvd**
   c. New city – **York**
   d. New state – **PA**
   e. New postal code - **17401**
   f. New email address – **jkballet@gmail.com**
D. Make the changes in SQL, and remember to paste the SQL statement and results into your answer template.
E. Finally, generate a result set with the customerid, companyname, address, city, state, postal code, and email address BEFORE the change for just customer **Jennifer Kmec**. (1 record updated)
F. Issue a **COMMIT**.
G. Explain: Why would you document the query before you make a change, and then run it again after the update?

Question 2:

A. Add two new suppliers using **SE-203** and **SE-204** as the Supplier IDs. Fill in the rest of the fields with appropriate values of your choosing. It cannot have the same values as any of the other suppliers in the list. You will utilize these records later in this assignment.
B. Run a query with all columns (Select *) from the SUPPLIER table and filter by the two new supplier IDs.
C. Then run a Count of all records in the SUPPLIER table.
D. Issue a **COMMIT**.
E. Explain: Why not just use the same supplier ID again for a new record?

Question 3:

A. Run a query that counts of the number of inventory parts per category. (11 rows)
B. The company no longer wants to categorize items with a **POW** category, so update the records by **changing any inventory parts with a POW category to a HOME category**. (Note: 3 parts will be updated.)
C. To **verify**, again show the count of inventory parts per category.
D. Now issue a **Rollback**; statement.

E. **Verify** (through COUNT) that the category has returned back to like it was before.
F. Perform the **UPDATE statement again** to set the parts with **POW** to **HOME**.
G. Issue a **Commit**;
H. Remove the categoryid **POW** from the CATEGORY table.
I. Issue a **Commit;**
J. Explain the difference between COMMIT and ROLLBACK.

Question 4. (worth 3 points)

One of your new suppliers (**SE-203**) is ready to supply parts to Eagle Electronics.

A. Create two new records for your supplier in the SUPPLIEDPART table.
    a. Review existing data in the database for guidance for the two new supplied parts.
    b. Use catalog numbers **400-5109** and **400-6148**. Catalog number and Supplier ID make up the composite primary key.
    c. Use part numbers **ADT-004** and **ADT-005** from the INVENTORYPART table.
    d. Report the SQL statements to make the additions.
B. Document all of your values for each field.
C. To confirm, run a query that joins the SUPPLIER, SUPPLIEDPART, and INVENTORYPART tables. List the supplierid, companyname, partnumber, catalognumber, unitcost, and stockprice for your two new records.
D. Issue a **COMMIT.**
E. Can supplierIDs be added to just the SUPPLIEDPART table and not the SUPPLIER table? Explain how this could happen in a database without referential integrity.

Question 5. (worth 3 points)

There is a table named TIMECARDLINE which currently has no records.

A. Confirm that the TIMECARDLINE table has no records (show count of records).
B. Run a query from the EMPLOYEE table to list the Employee ID, hire date, and salary wage for each employee with a salary wage **less than 18** and a type of **Temporary**. (6 rows selected)
C. In the TIMECARDLINE table, populate records for these 6 employees using a SELECT clause:
    a. You are populating only two fields in each record:
        i. Use the **Employee ID** from the Employee table
        ii. Use **SysDate** for the DateTimeStart value
    b. Include the SQL statement used to make the addition, and the result.
D. To confirm, produce a list of each Employee ID and the DateTimeStart in “fmMonth DDth, YYYY, Day“ format (called TimeCard) for all rows in the updated TimeCardLine table (should be 25 rows selected).

E. Use TO_DATE when inserting time cards for employees **250120 and 251088**. Use **02-20-2020** as the date and use the format MM-DD-YYYY to enter in the date (since this is not in the Oracle default format, you need to use TO_DATE).
F. Run a SQL statement with Employee ID and DateTimeStart from TIMECARDLINE. (8 rows in result).
G. What is the difference between TO_DATE and TO_CHAR?

Question 6.

A. Increase the Unit Cost in the Supplied Part table by 10% for all suppliers from the state of Indiana (**IN**).
B. Show the Supplier ID, Catalog Number, Part Number, and Unit Cost **BEFORE** the change.
C. Show the SQL statement to make this **change**.
D. Confirm the change by listing the Supplier ID, Catalog Number, Part Number, and Unit Cost **AFTER** the change.
E. Note: 3 records are updated.

Issue a **COMMIT**.

F. Explain how you constructed this SQL statement to make the update.

Question 7. Remove customer **C-200045** from the database.

A. You will have to delete from 6 tables in order to remove this customer due to customer orders, shipped orders, packing, etc.
B. Run and document the select statements first for each of the tables in order to isolate the data you need to delete. **SHOW ALL OF YOUR WORK**.
C. Then run the delete from commands for each of the tables. HINT: Work backwards with the child tables first.
D. Explain why you must delete in a particular order. And also explain what order you used.

Issue a **COMMIT**.

Question 8. Run a sql statement that displays your userID and the system time stamp (SYSTIMESTAMP). No explanation required for this question.