COSI 127B: Introduction to Database Systems

Programming Assignment II

Database Management and Applications

Due: 3:30 p.m., Monday, March 19, 2018

1   Introduction

The purpose of PA1 was to let you get familiar with SQL by composing several complex queries. PA2 is a complementary assignment designed to let you get familiar with two important skills required of a DBA: writing integrity constraints and implementing triggers. For this assignment, you will act as DBA for a manufacturing company that must monitor its inventory of parts, the suppliers for those parts, orders made for those parts and their fulfillments. You will be provided with the schema and data to populate the very large database maintained by the manufacturing company. Your job will be to formulate and construct integrity constraints and triggers to ensure data integrity within this large application.

2   Background

The data you will be working with is drawn from the TPC-H benchmark[1] database, which models the data needs of a manufacturing company. Information maintained in this database includes an inventory of parts, suppliers for those parts, orders for parts and order fulfillments. Your job will be to tune this database by adding integrity constraints and triggers.

2.1   The TPC-H Database

The database you will be working with consists of eight tables, each with several attributes. Each attribute name is prefixed by a unique prefix (derived from the name of the table where it is contained), to ensure the uniqueness of attribute names across the database. Throughout this document, we will refer to an attribute without its prefixes if the table containing the attribute is understood. The eight tables, their primary keys, and their assumed attribute prefixes are shown in on the following page.

---

[1]A benchmark consists of a schema, data population and a suite of queries and transactions that is used to compare the performance of different database systems.

2.1.1 Tables

Table Name	Prefix	Primary Key
region	r	regionkey
nation	n	nationkey
supplier	s	suppkey
part	p	partkey
partsupp	ps	partkey, suppkey
customer	c	custkey
lineitem	l	orderkey, linenumber
orders	o	orderkey

Table 1: The tables of the TPC-H Database
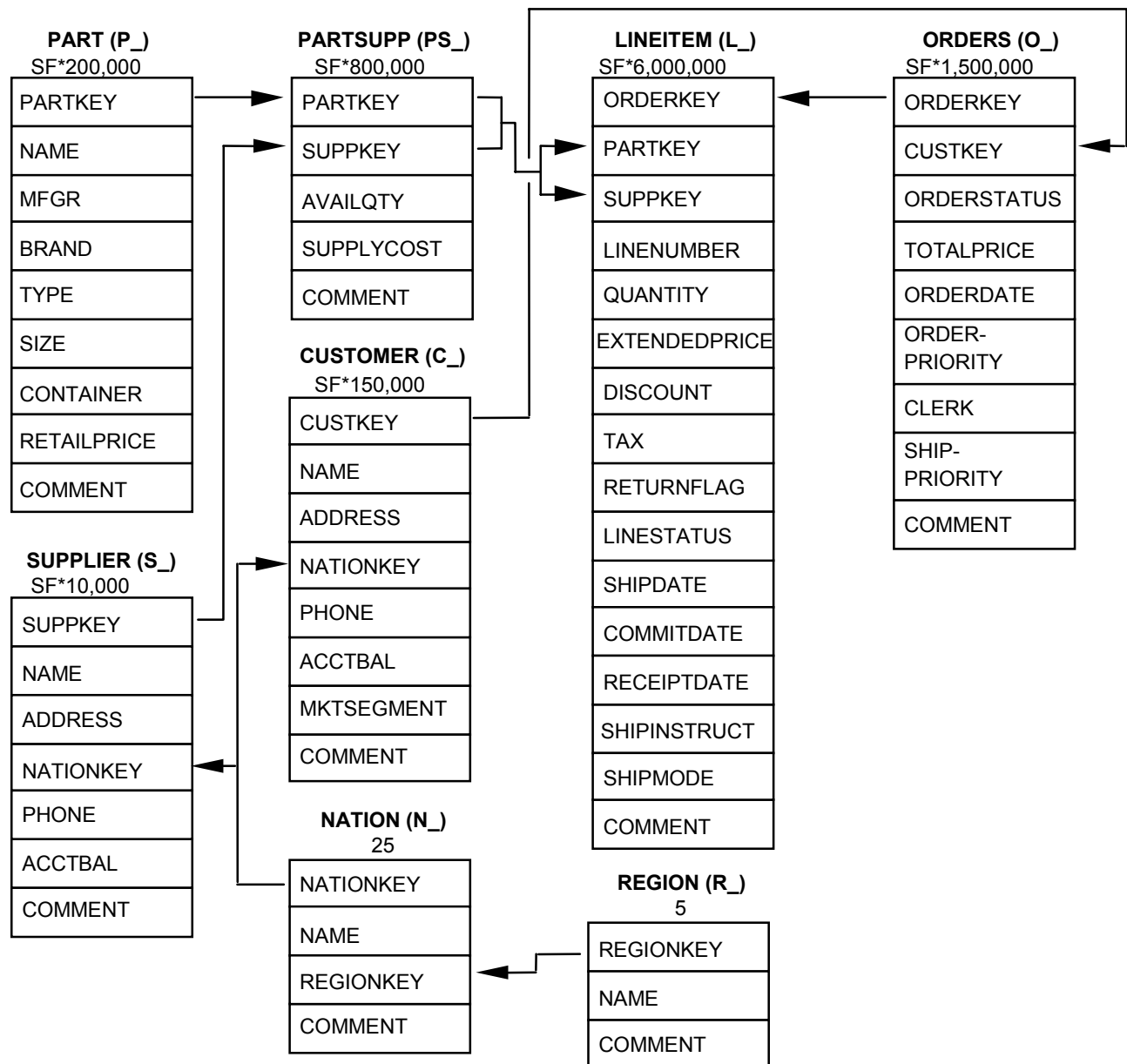
2.1.2   Schema

Figure 1: The Schema of the TPC-H Database

2.1.3   Description

region: This table lists the 5 regions (i.e., continents) in the world the data in this database pertains to. Each region has a name (name) and a comment permitting annotations for the region (comment).

nation: This table lists all nations the manufacturing company deals with. Every nation has a name (name), a region in which it is contained (regionkey) and a comment (comment).

supplier: This table lists all suppliers of interest for the manufacturing company. Data for a supplier includes its name (name), address (address), the nation where it is located (nationkey), phone (phone), account balance (acctbal) and comments about the supplier (comment).

part: This table holds information about all parts needed by the manufacturing company. Relevant data for a part includes its name (name), the parts manufacturer (mfgr), and its retail price (retailprice).

partsupp: A row in this table represents a given part (partkey) that is supplied to the manufacturer by a given supplier (suppkey). For each part and supplier, additional data includes the number of parts available from the supplier (availqty) and the suppliers cost for the part (supplycost).

customer: This table contains information regarding the customers of the manufacturing company. Information stored for each customer includes his/her name (name), address (address) and nation he/she resides in (nationkey), as well as a contact phone number (phone), and the balance of the customers account with the company (acctbal).

lineitem: This table maintains each individual part order contained in a purchase order. Every row represents some ordered part (identified by the part (partkey) and the supplier it was ordered from (suppkey)) the quantity ordered (quantity), the total price paid for the parts (extendedprice) before tax and a discount, the date when the parts were shipped (shipdate), the date when the parts were promised by the supplier (commitdate), the date when the parts were received (receiptdate), the discount received (discount) and so on. As well, each lineitem contains an attribute (linestatus) that has a value of CO if the part order is still open and CF if the part order has been fulfilled.

order: This table contains details about every purchase order (or just, order). An order has a single customer (though a customer can maintain multiple orders) and is made up of one or more lineitems. An order status (orderstatus) is set to CO if it is open (i.e., if all lineitems it contains have an open status), CF if the order is fulfilled (i.e., every lineitem it contains is fulfilled), and CP if it is partially fulfilled (i.e., if some but not all lineitems in the order have been fulfilled). Order records also include the total price of the order (totalprice) and the date the order was placed (orderdate).

3   Getting Started

This section will instruct you on how to setup your development environment where you can write your queries.

Please proceed in order through the following sections, and refer back to them as needed throughout the assignment.

3.1   Environment

All queries written as part of this assignment will be executed on one of the CS public machines. . They are accessible via SSH and are able to connect to the database servers vis PSQL. The individual server you will use are the same as the one you used for PA1.

To begin, first download the assignment tar file from latte. Next use scp to move the tar to your cs account.

scp [path to tar] [your username]@[cs public machine name].cs.brandeis.edu:.

Next, simply log onto the server using SSH:

ssh [your username]@[cs public machine name].cs.brandeis.edu

Then proceed to decompress and untar the archive using `tar`:

tar -xvzf PA2_2018.tar.gz

This will create a directory called "MyPA2" containing all of the files you will need for this assignment. This will be your working directory for the remainder of this guide.

3.2   Database

To log onto your database you will need to set up your environment variables. These will be read by PSQL to authenticate and log you in to the right database. Simply execute the following set of commands in your shell[2]:

```
$ export PGHOST=[your database host]
$ export PGUSER=[your user name]
$ export PGDATABASE=[your user name]pa2
$ export PGPASSWORD=[your database password]
```

3.2.1   Loading TPC-H

Once you set up these variables, you will want to populate your database with TPC-H data. This task along with many other tasks in this assignment will be automated using `make` - a *nix tool to execute a predefined batch of commands.

To create a new database and load it with TPC-H data, execute the following command:

```
$ make initialize-db
```

This may take about a minute. Once it is done, enter `psql` and type `\d` to show all tables.
You should see the following output:

```
        List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+---------
 public | customer | table | [your user name]
 public | lineitem | table | [your user name]
 public | nation   | table | [your user name]
 public | orders   | table | [your user name]
 public | part     | table | [your user name]
 public | partsupp | table | [your user name]
 public | region   | table | [your user name]
 public | supplier | table | [your user name]
(8 rows)
```

Ensure the data is there by selecting from one of the tables.

3.2.2   Database Reset

If ever you feel like you've corrupted the database by doing an exercise incorrectly, you can always start again with a fresh, uncorrupted database by running the following reset command:

```
$ make reset-db.
```

The above should also be used for several exercises in this assignment, which require that you reset to a fresh database so as to erase the effects of previously completed exercises.

---

[2]You can also add these to your `.bashrc` to make these variables persistent.

4   Constraints and Triggers

4.1   Introduction

Your task for this assignment is to add integrity constraints and triggers to the TPC-H database to ensure that data consistency is maintained. Although triggers have not been discussed in class, you can read about triggers at:

`http://www.postgresql.org/docs/8.4/static/sql-createtrigger.html`

or

`http://www.postgresql.org/docs/8.4/static/plpgsql-trigger.html`

In short, triggers are statements that a database executes automatically as a side effect of certain updates to the database. When registering a trigger with the system, you specify what events (expressed as conditions) trigger execution of the trigger, and what actions (expressed in the Postgres trigger language) are to take place as a result. For example, a trigger could be used to automatically calculate the value of derived attributes, every time one of the attributes from which it is derived is updated. Triggers can be used to enforce data integrity constraints that cannot be expressed with the PRIMARY KEY, UNIQUE, CHECK, ASSERTION and FOREIGN KEY constructs of SQL because they require manipulating data in more complex ways than can be achieved using keys, checks, assertions, and foreign keys. In Postgres, you will be writing the trigger actions in a procedural programming language that resembles SQL, but adds features such as control structure, variables, etc. This language is called PL/pgSQL. The documentation for PL/pgSQL can be found at `http://www.postgresql.org/docs/8.4/static/plpgsql.html`. Note that assertions are not yet implemented in PostgreSQL. You should avoid using assertions.

We will also go over it during the help session.

4.2   Key and Foreign Key Constraints

4.2.1   Excercises

Listed below are the primary and foreign key requirements for every table in the TPC-H database. Your task is to add these constraints to the database using the appropriate SQL commands. You should do this using `ALTER TABLE` commands only (i.e., not as part of a `CREATE TABLE` command). For example, to add a primary key on an attribute 'thekey' in a table called 'mytable',you would type:

`ALTER TABLE mytable ADD PRIMARY KEY (thekey)`

To add a foreign key (calling it fk1) from attribute otherkey of mytable, referencing the primary key attributes of yourtable, you would type:

```
ALTER TABLE mytable
ADD CONSTRAINT fk1 FOREIGN KEY (otherkey) references yourtable
```

For the foreign keys below, do not give any action the default action will be NO ACTION which will reject any update that violates the constraint.

Keep your SQL instructions for creating primary and foreign key constraints in a separate file, as you will need to recreate these constraints every time you reset the database for all exercises that follow. The primary and foreign keys to add are as follows:

Table Customer :

    PRIMARY KEY (C_CUSTKEY)

    FOREIGN KEY (C_NATIONKEY) referencing NATION

Table Lineitem :

    PRIMARY KEY (L_ORDERKEY,L_LINENUMBER)

    FOREIGN KEY (L_PARTKEY,L_SUPPKEY) referencing PARTSUPP

    FOREIGN KEY (L_ORDERKEY) referencing ORDERS

Table Nation :

    PRIMARY KEY (N_NATIONKEY)

    FOREIGN KEY (N_REGIONKEY) referencing REGION

Table Orders :

    PRIMARY KEY (O_ORDERKEY)

    FOREIGN KEY (O_CUSTKEY) referencing CUSTOMER

Table Part :

    PRIMARY KEY (P_PARTKEY)

Table Partsupp :

    PRIMARY KEY (PS_PARTKEY,PS_SUPPKEY)

    FOREIGN KEY (PS_SUPPKEY) referencing SUPPLIER

    FOREIGN KEY (PS_PARTKEY) referencing PART

Table Region :

    PRIMARY KEY (R_REGIONKEY)

Table Supplier :

    PRIMARY KEY (S_SUPPKEY)

    FOREIGN KEY (S_NATIONKEY) referencing NATION

For more documentation, see:

- `http://www.postgresql.org/docs/8.4/static/sql-altertable.html` for the `ALTER TABLE` syntax.

- `http://www.postgresql.org/docs/8.4/static/sql-createtable.html` for table constraint syntax.

4.3   Integrity Constraints and Triggers

4.3.1   Overview

The data integrity constraints achieved through primary and foreign keys are powerful, but we will see that in complex databases such as the one we are working on, there are more complicated data integrity requirements that require more powerful tools to be expressed. For each data integrity requirement listed below, write the appropriate SQL integrity constraint using ALTER TABLE commands. If (and only if) the constraint cannot be expressed with an SQL integrity constraint, express the constraint with a trigger. We will provide you with details on how to test to see that your integrity constraint is working.

Note #1 You should work on a fresh database for every problem in this section of the assignment. (I.e., run the reset script from a shell prompt before beginning each new problem). However, youll need to add the primary and foreign key constraints you formulated in Section 4.2.1 everytime you reset the database, so you are well-advised to keep the SQL instructions that add these constraints in a separate file.

Note #2 When testing out triggers and their associated functions, Postgres does NOT check the syntax of a function at creation time only at execution time. To test triggers and functions, you will need to drop them each time; drop the trigger first, then the function:

    DROP TRIGGER foo ON table;
    DROP FUNCTION some_function();

4.3.2 Exercises

1. If `nation` is updated to change a nationkey for a given nation, nationkeys should be modified across every table in the database.

   To Test Your Solution: Your solution can be verified by issuing the following queries:

   ```
   SELECT c_custkey, c_nationkey FROM customer WHERE c_nationkey = 99;
   SELECT s_suppkey, s_nationkey FROM supplier WHERE s_nationkey = 99;
   ```

   Both queries should return tables with 0 rows.
   Next, issue the following update:

   ```
   UPDATE nation SET n_nationkey = 99 WHERE n_name = 'UNITED STATES';
   ```

   Now issue the initial two queries again. If you've implemented the constraint or trigger correctly, these queries should now return 184 and 11 rows respectively. After youve successfully tested your constraint or trigger, reset the database and recreate the primary key and foreign key constraints you formulated in Section 4.2.1. **Note:** If you need to run this test again, make sure that you reset the database and recreate the primary and foreign key constraints from Section 4.2.1 beforehand. For all exercises in Part 1, you will need to reset the database and recreate the key constraints if you want to test your constraint more than once.

2. If a part's retail price changes, the supply cost for any row in `partsupp` containing this part should change by the same amount. Do not worry about updating prices in `lineitem`.

   To Test Your Solution: Issue the query below to see the current costs of this part from various suppliers:

   ```
   SELECT ps_supplycost FROM partsupp WHERE ps_partkey = 1;
   ```

   Next, issue the following update query to increase the parts retail price by 10%:

   ```
   UPDATE part SET p_retailprice = p_retailprice * 1.1
   WHERE p_partkey = 1;
   ```

   Now, issue the 1st query again to see if the corresponding supply costs in `partsupp` have changed by the appropriate amounts. After you've successfully tested your constraint or trigger, reset the database and recreate the primary key and foreign key constraints you formulated in Section 4.2.1.

3. Customers cannot have more than 14 open orders. Reject any additional orders for customers who already have 14 orders. Be careful, Postgres is case sensitive! An open order is an order with `o_orderstatus` set to 'O',not 'o'.

   To Test Your Solution: The customer with `custkey = 112` already has 13 open orders. Insert another order for this customer:

   ```
   INSERT INTO orders VALUES
   (99098, 112, 'O', 99.00, NOW(), '5-LOW', 'Clerk#99', 0, 'IWillPass');
   ```

   Now that this customer has 14 open orders, trying to insert another order should fail. After you've successfully tested your constraint or trigger, reset the database and recreate the primary key and foreign key constraints you formulated in Section 4.2.1.

4. Upon deletion of a customer, delete any orders (and its lineitems) the customer had.

   To Test Your Solution: Determine the orders/lineitems the customer with custkey = 203 has by issuing the query:

   ```
   SELECT l_orderkey, l_linenumber
   FROM lineitem
   WHERE l_orderkey IN (SELECT o_orderkey FROM orders WHERE o_custkey = 203);
   ```

   This should return:

l_orderkey	l_linenumber
80996	1
80996	2
65280	1
14945	1
14945	2
14945	3
14945	4
14945	5
14945	6
148071	1
148071	2
148071	3
148071	4
136678	1
136678	2
136678	3
136678	4

(17 rows)

Now issue a deletion query to delete this customer:

```
DELETE FROM customer WHERE c_custkey = 203;
```

The first query should now return 0 rows.
After you've successfully tested your constraint or trigger, **reset** the database and recreate the primary key and foreign key constraints you formulated in Section 4.2.1.

5. An order can have status 'F' (fulfilled), 'O' (open) or 'P' (partial). If a new lineitem is inserted check all lineitems for that order, and update orders. **o_orderstatus** appropriately. Note: If all lineitems have the same (line) status, the order also has this (order) status. Otherwise the order status is 'P'.

   To Test Your Solution: Issue the following query:

   ```
   SELECT o_orderstatus FROM orders WHERE o_orderkey = 7;
   ```

   This order should have a status of 'O'. Now issue the update:

   ```
   INSERT INTO LINEITEM VALUES
   (7, 1, 2, 8, 1, 99.00, 0.00, 0.05, 'N', 'F', NOW(), NOW(), NOW(), 'NONE', 'MAIL', 'No Comment');
   ```

   When you issue the 1st query again, it should report that the order with **o_orderkey = 7** has a status of 'P'. After youve successfully tested your constraint or trigger, reset the database and recreate the primary key and foreign key constraints you formulated in Section 4.2.1.

6. If you delete a region from the database, all nations within that region, and all suppliers located in any of those nations also should be deleted. However, customers from those nations should not be deleted. Any **partsupp** record associated with deleted suppliers should also be deleted, but **lineitem** records referencing those **partsupp** records should not be deleted.

   To Test Your Solution: Delete the region with regionkey 3 (Europe) from the region table with the following update:

   ```
   DELETE FROM region WHERE r_regionkey = 3;
   ```

   A good way to test your solution for this exercise involves counting how many rows you deleted or affected, although you may wish to verify manually the contents of some of the tables to confirm your solution. Run the following queries:

   ```
   SELECT COUNT(*) FROM nation WHERE n_regionkey = 3;
   SELECT COUNT(*) FROM supplier;
   SELECT COUNT(*) FROM customer WHERE c_nationkey IS NULL;
   ```

```
SELECT COUNT(*) FROM partsupp;
SELECT COUNT(*) FROM lineitem WHERE l_partkey IS NULL AND l_suppkey IS NULL;
```

If your constraint or trigger was expressed correctly, the queries should return 0, 229, 859, 18407 and 42154 rows respectively. (Before the trigger been executed, the queries would have returned 5, 299, 0, 24021 and 0 rows respectively.)

After you've successfully tested your constraint or trigger, reset the database and recreate the primary key and foreign key constraints you formulated in Section 4.2.1.

7. The table below shows the changes in supply costs for all parts issued by a supplier, when that supplier moves from one region to another. For example, if a supplier moves from Asia to Europe, the costs of parts sold by that supplier should rise by 10%.

From/New	America	Asia	Europe
America	0%	-20%	5%
Asia	20%	0%	10%
Europe	-5%	-10%	0%

Update the costs of parts sold by a supplier according to the rates listed in this table every time a supplier changes its location (`nationkey`).

To Test Your Solution: The supplier with `suppkey = 1` is located in the USA. Issue the following query to determine the total costs of parts sold by this supplier:

```
SELECT SUM(ps_supplycost) FROM partsupp WHERE ps_suppkey = 1;
```

The answer should be $37704.83. Now, issue the following update to move the supplier's location to France.

```
UPDATE supplier SET s_nationkey = 6 WHERE s_suppkey = 1;
```

Issue the 1st query again. The sum returned should now have changed by the amount indicated by the table above (+5%). Note that Postgres does some rounding in computing the answer to the precision specified by the column type, and for this example the updated supply cost would show $39590.11. Try this test for other suppliers to test all other percentages in the table.

After you've successfully tested your constraint or trigger, reset the database and recreate the primary key and foreign key constraints you formulated in Section 4.2.1.

4.4   To Hand In

For Part 1 of PA2, you will need to submit a file containing the following:

1. The SQL code that adds to the TPC-H database, all of the primary keys and foreign keys described in Section 4.2.1

2. The SQL constraint commands and triggers used to enforce the 7 constraints described in Section 4.3.2.

Each of your primary and foreign key constraints should be written as files under the following directory structure:

`./sql/keys/[primary / foreign].sql`

Where all primary key definitions go in the `primary.sql` file and all foreign key definitions go in the `foreign.sql` file. You shoule be able to pipe them in that order to PSQL, which is how they will be graded.

Each of your constraint or trigger used to enforce constraints 1-7 should be written as files under the following directory structure:

`./sql/triggers/q_[question number].sql`.

5   Submission

To submit your solutions, you should first use cd to move to the directory containing the MyPA2 directory. Next, compress the directory using the command:

tar -czvf [your last name]_[your first name]_127b_pa2.tar.gz MyPA2/

Next, from your local machine (while not connected using ssh) execute the following command to transfer the compressed file from the server to your machine.

$ scp [your username]@[cs public machine name].cs.brandies.edu: /MyPA2/*_*_127b_pa2.tar.gz .

The file should then be transferred to the current directory.

Upload the file to LATTE.