University of Ottawa School of IT and Engineering
CSI3130- Database II

Project Description

Posted Monday October 10th

Consider a **Banking** environment, with the following database schema. In this scenario, a client may have many accounts and there are one or more clients linked to an account. An account is associated with a branch and it follows that there are many accounts at each branch.

Client (Client-number, Lastname, Firstname, Marital-status, Postal-code, Phone, City, ...)
Owns(Client-number, Acc-number)
Account(Acc-number, Dollar-balance, Branch-number)
Branch(Branch-number, Branch-name, City)

Suppose that a **large** volume of concurrent transactions are executed against this database. Refer to the MS Excel spreadsheets, which contain the data you should use for this project.

Recall that PostgreSQL uses **a multi-version concurrency control** algorithm. Unlike traditional database systems which use locks for concurrency control, PostgreSQL maintains data consistency by using a **Multiversion Concurrency Control** model, MVCC). This means that while querying a database each transaction sees a **snapshot** of data (a database version) as it was some time ago, regardless of the current state of the underlying data. This protects the transaction from viewing inconsistent data that could be caused by (other) concurrent transaction updates on the same data rows, providing transaction **isolation** for each database session. According to the PostgreSQL designers, the main advantage to using the MVCC model of concurrency control rather than locking is that in MVCC locks acquired for querying (reading) data do not have conflicts with locks acquired for writing data, and so reading never blocks writing and writing never blocks reading.

However, table and row-level locking facilities are also available in PostgreSQL. According to the PostgreSQL team, "**proper use of MVCC will generally provide better performance than locks**". In this project, we are going to invest if this claim is, indeed, true! The aim of this project is

thus to study the effects, in terms of consistency and isolation, when varying (a) the **level of isolation** and (b) the level of locking.

Your task

Your task is to (1) **create** a database, populate it with data and implement the transactions, (2) **study** the impact of varying the concurrency control levels and locking mechanisms in PostgreSQL in terms of concurrency control and transaction throughput.

List of transactions

As a first step, you are required to **implement** the following transactions. **Create** your own database schema and populate it with the data we have supplied you with, in order to **test** these queries.

T1. A client named Igor Rasmussen **opens** a new account (account 447712) at the East Side Vancouver branch and deposits $300.00 into it. His current marital status is Single and he lives in Victoria, British Columbia.

T2. Igor **withdraws** $275.00 from account 447712.

T3. Igor **transfers** $2,800.00 from account 424244, which is at the Downtown branch in Calgary and has a current balance of $7,000.00, into account 447712. (Assume account 424244 already exists and Igor is the sole owner.)

T4. Igor's friend, Isis Erickson, **deposits** $1,000,000.00 into account 447712. (Assume that Isis has an account *445128* in downtown Toronto. This account currently has a balance of $3,000,000.00. She **transfers** the money from this account. She lives in Toronto.)

T5. Igor gets married to Isis and relocates to Toronto. So, he **moves** his account 447712 from the East Side Vancouver to the downtown Toronto branch. He also **changes** his marital status, postal code and phone number. (Suppose that he keeps the same account number, and last name, for account 447712. Furthermore, he chooses to **keep** his other account (424244) at the Downtown branch in Calgary.)

T6. The manager at the downtown Toronto branch **calculates** the **sum** of the balances of all the customers that have an account at his branch.

T7. The manager at the Downtown branch in Calgary **calculates** the **average** balance of all customers that have an account at his branch.
T8. The bank decides to give all clients, with an account balance higher than $400,000.00, a **bonus** of 7%.
T9. Isis and Igor **open** a new joint account (account *425450)* at the downtown Toronto branch and Isis transfers $600,000.00 from her current account (i.e. account *445128*) into this new joint account.
T10. Igor **withdraws** $550,000.00, in cash, from the newly opened joint account.

More information

As a first step, read through the PostgreSQL documentation about concurrency control, as contained in Chapter 13 (http://www.postgresql.org/docs/9.5/static/mvcc.html.)

PostgreSQL allows for only two different levels of isolation (**read committed data and serializable**). Recall that "read committed" may lead to unrepeatable reads and the phantom problem. This is the default setting in PostgreSQL, due to the overhead associated with serializable isolation.

About explicit locking, also note the following. In PostgreSQL, there are several distinct types of lockable objects, **whole relations**, **individual pages** of relations, **individual tuples** of relations, **etc**. Also, the right to extend a relation is represented as a **separate** lockable object. PostgreSQL sets these locks automatically and you can acquire any of these locks at any time by using the **LOCK** command. For example, all updates require a **ROW EXCLUSIVE** lock to be set. Similarly, the **LOCK TABLE** command enforces an **ACCESS EXCLUSIVE** lock.

Your **first** task is to collect a number of transaction-related statistics and then present it to the user in a condensed, integrated form, when running multiple sessions using the default settings. **Secondly**, you need to explicitly set the lock levels of all tables or pages accessed by a transaction, and collect the same statistics.

- You may need to study the PostgreSQL System **Catalog**, in order to identify the relevant sources of the statistics you need to collect.

- Refer to http://www.postgresql.org/docs/9.5/static/catalogs.html and also to http://www.postgresql.org/docs/9.5/static/monitoring-stats.html for more information.

- To determine and set the actual level of explicit table-level locks, use the LOCK command. (See http://www.postgresql.org/docs/9.5/static/explicit-locking.html.)

- On a low level, the lmgr.c and utility.c files deal with the LOCK table command and call a function in lockcmd.c to lock a relation according to its identifier.

- To examine the list of the currently outstanding locks in a database server, you may want to use the information provided by **`pg_locks`**, which gives one row per active lockable object, requested lock mode, and relevant transaction. (See http://www.postgresql.org/docs/9.5/static/view-pg-locks.html.)

- To display the information about transactions that are currently prepared for two-phase commit, you may use **`pg_prepared_xacts.`** (See http://www.postgresql.org/docs/9.5/static/view-pg-prepared-xacts.html).

- You can **monitor** the disk usage in one of three ways, as discussed in http://www.postgresql.org/docs/9.5/static/diskusage.html

- The default page size is **8kB** and you may want to read the following page about the page lay-out: http://www.postgresql.org/docs/9.5/static/storage-page-layout.html

- You will have to **delve** into the PostgreSQL source code to set the exclusive page-level locks.

Project Deliverables

Part A: Database Creation, Population (5%)

Complete in a group of 2-3 students; **due on October 16th at 22h55**

You are required to create the database schema, populate the database with the data as provided to you, using *the same sort order* as in the MS Excel files. It is important to finish this part before the next lab (see posted supplementary sheet and instructions).

Part B: Implementing Transactions (15 %)

Complete in a group of 2-3 students; **due on October 24th at 22h55**

Implement the SQL code for the transactions listed above. Upload your SQL code to blackboard before the due date. Each group should submit a copy.

Part C: Building PostgreSQL (5 %)

Complete in a group of 2-3 students; **due on October 30th at 22h55**

Here, your task is to build the PostgreSQL backend server from the source code. Upload the screen print in a MS Word file to the Virtual Campus before the due date. Each group should submit a copy.

Part D: Monitoring Concurrency Control (75%)

Complete in a group of 2-3 students**; due on December 4th at 23h55, with partial submissions indicated as bellow.**

You are required to open two concurrency sessions, and run the following two sets of transactions at the same time, against your database. Note that these should be handled as separate transactions, i.e. each transaction should have its own *begin* and *end* transactions statement.

Session 1	T1, T2, T4, T7, T10
Session 2	T3, T5, T6, T8, T9

For each one of the isolation levels shows in the table below, you are required to run a *Pass* consisting of these two sessions.

	Due Date	Isolation	Locking Level
Pass 1	**November 13th**	Read committed (Default)	Default
Pass 2	**November 20th**	Serializable	Default
Pass 3		Read committed (Default)	Table level locking: Exclusive
Pass 4		Serializable	Table level locking: Exclusive
Pass 5		Read committed (Default)	Table level locking: Access Exclusive
Pass 6	**December 4th**	Read committed (Default)	Page level locking: Exclusive

During each of the ***Passes***, you should collect the following relevant transaction statistics into your own "*transaction statistics log*" and then display it at the end of the two sessions.

a. the **names** of the actual disk pages used, as they are accessed by each transaction,

b. the **actual locks**, in any, as that are being held, and the transaction-id of the transaction that holds it,

c. the **level** of explicit locking used, if any,

d. the **status** of a transaction as it is executed, including active, prepared and commit (or abort), together with the initial time it started, and

e. whether a **deadlock** was detected and which transaction was **aborted** (if any).

You are also required to display the **end** result of the sessions and then determine whether **serializability** has been ensured. That is, you need to determine whether the end results leave the database in a consistent state, i.e. no unrepeatable reads and phantoms have occurred. (Refer to Chapters 20 and 21 of B1 text book or Chapters 16 and 17 of B2 text book.)

Mark allocation for this part**:**

1. **Understanding** of the project (15%)

2. Collect the above **transaction-related statistics** (6 marks each) into the transaction statistics log and display it on the screen, when running the sessions using the default settings (**Pass 1**), i.e. when the isolation level is read committed. (5*6=30%) (Due: November 13th)

3. Collect the above **transaction statistics** (3 marks each) into the transaction statistics log and display it on the screen, when running the two sessions using the settings as in **Passes 2 to 5.** (3*5=15%) (Due: November 20th)

4. Display the **end results** at the end of each session and indicate whether **serializability** has been violated. (5%)

Next, you are required to change the level of locking of all disk pages as they are accessed, to **exclusive** page-level locking.
(See http://www.postgresql.org/docs/9.5/static/explicit-locking.html.)

5. Collect the above **transaction statistics** into the transaction statistics log and display it on the screen, when running the two sessions using **page-level locking (Pass 6)**.[1] (25%) (Due: December 4th)

Deliverables:

All groups are required to submit all their source files via the Blackboard. You are **also** required to demonstrate your project to the TA during time slot on Week of December 5th . Project demonstration and interview will be scheduled. **Each individual group member will be interviewed to test his/her understanding of the project (15%).**

[1] Part D is out of 75%, but you are able to earn 90%. That is, students who successfully complete question 5 will earn an additional 15% "bonus marks".