Homework 5

CSCI 4332

Due Dec. 6, 2015 at 11:55 PM

Summary

In this homework, you will do some things with transactions.

The programming portion of this homework is based on the bookmarky server we have been developing in class.

Your task is to modify the program to have tags as a separate entity, with a `bookmark_tag` joining table linking bookmarks to tags. There are a few pieces for this.

Getting Started

Download the zip file `bookmarky-a5.zip` **from the assignment**. This contains the data, import scripts, and final code from Lecture 9. Unpack this and use it to get started.

Part 1 (10 points): Modifying the Schema

Your first task is to modify the schema for this new feature.

Right now, our `bm_tag` table contains the bookmark ID and the tag text.

You need to:

- Create a new table to store tags, and associate them with IDs. Tags are shared across all users (that is, all users with the tag 'animal' use the same tag entity and tag ID for 'animal').
- Modify `bm_tag` to link bookmarks and tags. There is one small wrinkle we want to support here: users may enter tags with different capitalization, but we want to simplify that for the tag objects. So the `bm_tag` *also* needs to have a field for the tag *as entered by the user*. When a tag is added, `bm_tag` will get the exact tag, and `tag` will get the lowercase version of the tag.

Your tables must have suitable foreign keys and `UNIQUE` constraints.

Draw an entity-relationship diagram of the logical model final bookmarky schema with your changes, and included it as a PDF file `schema.pdf`.

Part 2 (30 points): Updating Bookmark Update Code

Modify the bookmarky code (you should just need to change `bookmarky/bookmarks.py`).

Your new code needs to:

1. Create a new entry for each tag to be added in the tag table for the *lowercased* version of the tag, if necessary. You can lowercase a tag by calling `tag.lower()` in Python.
2. Add, remove, or update the `bm_tag` entries as necessary. If the user re-entered one of the same tags, but with different capitalization, you should update the relevant row of `bm_tag`.

Change both the `add_bookmark` and `update_bookmark` functions.

Use transactions appropriately. It is easiest to have correct transaction behavior if you use the `SERIALIZABLE` isolation level. If the transaction fails to commit, try again up to 5 times.

Part 3 (20 points): Isolation Behavior

This problem uses the Charity database from HW3, and the isolation level material from Lecture 12 and the PostgreSQL manual section on 'Transaction Isolation'.

Assume that the `gift` table contains the following:

gift_id	donor_id	date
3920	42	2014-04-01

and `gift_fund_allocation`:

gift_id	fund_id	amount
3920	1	35.0
3920	2	45.0

Consider the following two transactions, happening in parallel:

Time	Transaction A	Transaction B
1	`START TRANSACTION;`	`START TRANSACTION;`
2	`INSERT INTO gifts (donor_id, gift_date) VALUES (42, '2014-11-01') RETURNING gift_id;` returns 3947	
3	`INSERT INTO gift_fund_allocation (gift_id, fund_id, amount) VALUES (3947, 1, 100.0)`	
4		`SELECT donor_id, SUM(amount) FROM gift JOIN gift_fund_allocation USING (gift_id) GROUP BY donor_id`
5	`INSERT INTO gift_fund_allocation (gift_id, fund_id, amount) VALUES (3947, 1, 100.0)`	
6	`COMMIT`	

Time	Transaction A	Transaction B

7		`SELECT donor_id, fund_id, amount`
		`FROM gift`
		`JOIN gift_fund_allocation`
		`  USING (gift_id)`

8		`COMMIT`

Write in a plain text or PDF (*not* Word) file called ‘Part3.pdf’ (or ‘Part3.txt’) the results of the Transaction B queries at time stamps 4 and 7 under each of the following isolation levels:

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

Note: PostgreSQL does not have `READ UNCOMMITTED` as a distinct isolation level; it treats it as equivalent to `READ COMMITTED`. For the purposes of this assignment, treat `READ UNCOMMITTED` as a true read-uncommitted isolation level, as allowed by ANSI SQL, permitting dirty reads.

Submitting

Submit a ZIP file containing your modified code, schema, drawing, and your answers to Part 3.