Transactions control

Databases 2022

Transaction Control - recap

- The following commands are used to control transactions.
  - **COMMIT** – to save the changes.
  - **ROLLBACK** – to roll back the changes.
  - **SAVEPOINT** – creates points within the groups of transactions in which to ROLLBACK.
  - **SET TRANSACTION** – sets the characteristics of the current transaction.

Anonymous block and Exception handling

```
do
-- variable declaration section
begin
    set transaction ISOLATION LEVEL [configuration];
    -- some code
    commit;
exception
    when [exception_type] then
        -- code to be executed when the exception occurs
    when [exception_type] then
        -- code to be executed when the exception occurs
end$$;
```

Exercise 1 (Banking transactions)

- Create a table of accounts
- Each account should have:
  - A unique ID
  - Name
  - Credit
  - Currency
- Generate and insert 3 accounts into the table, each account has 1000 Rub.
- Create Transactions:
  - T1 : Account 1 send 500 RUB to Account 3
  - T2: Account 2 send 700 RUB to Account 1
  - T3: Account 2 send to 100 RUB to Account 3
  - Return Credit for all Account
- Create Rollback for T1,T2,T3.

Exercise 1 (Banking transactions)

- Add this field:
  - BankName
- Account 1 & 3 is SberBank, Account 2 is Tinkoff.
- Define the following conditions for each transaction
  - Internal transaction’s fee is 0 RUB.
  - External transaction’s fee is 30 RUB.
- Fees Should be saved in new Record (Account 4).
- Create Transactions:
  - T1 : Account 1 send 500 RUB to Account 3
  - T2: Account 2 send 700 RUB to Account 1
  - T3: Account 2 send to 100 RUB to Account 3
  - Return the amount Credit for all Account
- Create Rollback for T1,T2,T3.

Exercise 1 (Banking transactions)

- Create new Table Called Ledger to show all transactions:
  - ID (unique)
  - From (ID)
  - To (ID)
  - Fee (RUB)
  - Amount (RUB)
  - TransactionDateTime (DateTime)
- Modify Exercise 1 & 2 To save all transaction inside this table
- This is How Bitcoin BlockChain Works….

Exercise 2 – Isolation level

- Create a table account as the following example:

username	fullname	balance	Group_id
jones	Alice Jones	82	1
bitdiddl	Ben Bitdiddle	65	1
mike	Michael Dole	73	2
alyssa	Alyssa P. Hacker	79	3
bbrown	Bob Brown	100	3

Exercise 2 – Isolation level

- Test with **`Read committed, Repeatable read`** isolation levels
- Connect to your database using postgres CLI from 2 different sessions to:

Step No.	Terminal 1	Step No.	Terminal 2
1	Start a transaction and display the accounts information.	2	Start a transaction and update the username for “Alice Jones” as “ajones”
3	Display again the accounts table	4	Display again the accounts table
Do both terminals show the same information? Explain the reason			
		5	Commit the changes and compare again both sessions.
		6	Start a new transaction
7	Update the balance for the Alice’s account by +10.	8	Update the balance for the Alice’s account by +20
Explain the output form the second terminal			
9	Commit the changes.	10	Rollback

Hint: `set transaction isolation level` **`read uncommitted;`**

Exercise 2 – Isolation level

- Test with **`Read committed, Repeatable read`** isolation levels:
  - Start a transaction (T1 & T2)
  - Read accounts with group_id=2 (T1).
  - Move Bob to group 2(T2).
  - Read accounts with group_id=2 (T1).
  - Update selected accounts balances by +15 (T1).
  - Commit transaction (T1 & T2).
- Explain the result for both isolation levels.

Note: make sure that bob is assigned to the group 3 at the beginning of each experiment and no other transaction is in progress at each session.

Exercise 3 – Optional

- Test with Repeatable read , Serializable isolation levels:
  - Start a transaction in both terminals.
  - Set the same transaction isolation level (T1 & T2).
  - Read the sum of accounts balances with group_id=2 (T1).
  - Move Bob to group 2 (T2).
  - Read accounts with group_id=2 (T1).
  - Update selected accounts' balances by +sum (T1).
  - Read accounts with group_id=2 (T1 & T2).
  - Commit (T1).
  - Commit (T2).
- Explain the result for both isolation levels.

Note: make sure that bob is assigned to the group 3 at the beginning of each experiment and no other transaction is in progress at each session.

For next lab – Installing mongoDB

- Binaries are available for all major platforms (Linux, OS X, Windows)
  - Easy way
  - https://docs.mongodb.com/tutorials/
- You can also build from sources
  - https://github.com/mongodb/mongo
  - https://github.com/mongodb/mongo/wiki/Build-Mongodb-From-Source
  - Hard way, but you can grab the latest version
- Go ahead and install MongoDB on your laptop
- Run with `mongod`

For next lab – Importing data

- Once you are done, import data from the following link:
  - https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json
- With the following command:
  - `mongoimport --db test --collection restaurants --drop --file ~/downloads/primer-dataset.json`
- To import data into a mongod instance running on a different host or port, specify the hostname or port by including the `--host` and the `--port` options in your mongoimport command

References

- https://habr.com/en/company/postgrespro/blog/467437/
- https://www.gatevidyalay.com/concurrency-problems-in- transaction/

See you next week ☺