Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе**

**Дисциплина**: Базы данных

**Тема**: Изучение транзакций

Выполнил студент гр. 43501/3 Супронович П.В.

(подпись)

Руководитель Мяснов А.В.

(подпись)

Санкт -Петербург

2016

**Цель работы**

Познакомиться с механизмом транзакций, возможностями ручного управления транзакциями, уровнями изоляции транзакций.

**Программа работы**

1. Изучить основные принципы работы транзакций.
2. Провести эксперименты по запуску, подтверждению и откату транзакций.
3. Разобраться с уровнями изоляции транзакций в Firebird.
4. Спланировать и провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции.
5. Продемонстрировать результаты преподавателю, ответить на контрольные вопросы.

Работа проводится в IBExpert. Для проведения экспериментов параллельно запускается несколько сессий связи с БД, в каждой сессии настраивается уровень изоляции транзакций. Выполняются конкурентные операции чтения/изменения данных в различных сессиях, а том числе приводящие к конфликтам.

**Ход работы**

1. Изучить основные принципы работы транзакций

Транзакция - задача с определенными характеристиками, включающая любое количество различных операций (SELECT, INSERT, UPDATE, или DELETE) над данными в БД. Транзакция должна рассматриваться как единая операция над данными, т.е. она выполняется либо полностью, либо никак. Так как возможны случаи возникновения ошибок во время выполнения транзакции, приводящих к невозможности ее закончить, необходимы особые операции для подтверждения (COMMIT) или отмены транзакции (ROLLBACK).

Все выполняемые транзакции должны удовлетворять следующим свойствам (ACID критериям):

1. *Атомарность.* Выполнение по принципу "`все или ничего"';
2. *Согласованность.* В результате транзакции система переходит из одного абстрактного корректного состояния в другое;
3. *Изолированность.* Данные, находящиеся в несогласованном состоянии, не должны быть видны другим транзакциям, пока изменения не будут завершены;
4. *Долговечность.* Если транзакция зафиксирована, то ее результаты должны быть долговечными. Новые состояния всех объектов сохранятся даже в случае аппаратных или системных сбоев.

Транзакция обычно начинается автоматически при подключении клиента к БД и продолжается до выполнения команд COMMIT или ROLLBACK, либо до отключения клиента или сбоя сервера.

1. Провести эксперименты по запуску, подтверждению и откату транзакций

Рассмотрим команды COMMIT и ROLLBACK на примере добавления записей в таблицу:

/\*таблица для теста\*/

SQL> **select** \* **from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

/\*Добавим запись\*/

SQL> **insert into** type **values** (3, 'another', 150.0)**;**

SQL> **select** \* **from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

3 another 150.00000

/\*Отмена транзакции\*/

SQL> **rollback;**

SQL> **select** \* **from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

/\*Добавим запись снова, но подтвердим транзакцию\*/

SQL> **insert into** type **values** (3, 'another', 150.0)**;**

SQL> **commit;**

SQL> **select** \* **from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

3 another 150.00000

/\*Удалим транзакцию\*/

SQL> **delete from** type **where** id = 3**;**

SQL> **select** \* **from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

/\*Создадим точку сохранения и добавим запись снова\*/

SQL> **savepoint** first**;**

SQL> **insert into** type **values** (3, 'another', 150.0)**;**

/\*Повторим процесс\*/

SQL> **savepoint** sec**;**

SQL> **insert into** type **values** (4, 'clothes', 50.0)**;**

SQL> **select** \* **from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

3 another 150.00000

4 clothes 50.000000

/\*Откатим транзакцию на 2 точку сохранения\*/

SQL> **rollback to** sec**;**

SQL> **select** \* **from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

3 another 150.00000

/\*Откатим транзакцию на первую точку сохранения\*/

SQL> **rollback to** first**;**

SQL> **select** \* **from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

/\*Завершение транзакции\*/

SQL> **commit;**

1. Разобраться с уровнями изоляции транзакций в Firebird

Каждая транзакция имеет свой уровень изоляции, который устанавливается при запуске и остается неизменным в течение всей ее жизни. Firebird SQL предусматривает 3 уровня изоляции:

* *READ COMMITTED* (читать подтвержденные данные)

Уровень изоляции READ COMMITTED используется, когда мы хотим видеть все подтвержденные результаты параллельно выполняющихся (в рамках других транзакций) действий. Этот уровень изоляции гарантирует, что мы не сможем прочитать неподтвержденные данные, измененные в других транзакциях, и делает возможным прочитать подтвержденные данные.

* *SNAPSHOT* (моментальный снимок)

Этот уровень изоляции используется для создания "моментального" снимка базы данных. Все операции чтения данных, выполняемые в рамках транзакции с уровнем изоляции SNAPSHOT, будут видеть только состояние базы данных на момент начала запуска транзакции. Все изменения, сделанные в параллельных транзакциях, не видны в этой транзакции. В то же время SNAPSHOT не блокирует данные, которые он не изменяет.

* *SNAPSHOT TABLE STABILITY*

Этот уровень изоляции также создает "моментальный" снимок базы данных, но одновременно блокирует на запись данные, задействованные в операциях, выполняемые данной транзакцией. Это означает, что если транзакция SNAPSHOT TABLE STABILITY изменила данные в какой-нибудь таблице, то после этого данные в этой таблице уже не могут быть изменены в других параллельных транзакциях. Кроме того, транзакции с уровнем изоляции SNAPSHOT TABLE STABILITY не могут получить доступ к таблице, если данные в них уже изменяются в контексте других транзакций.

1. Спланировать и провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции
   1. READ COMMITED

SQL> **select** \* **from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

3 another 150.00000

SQL> **set transaction isolation level read** committed**;**

**Commit** current **transaction** (y/n)?y

Committing.

SQL> **select** \* **from** type**;**

/\*Запрос “висит” до команды commit на втором терминале\*/

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

3 new type 150.00000

На другом терминале выполняем:

SQL> **update** type **set** name = 'new type' **where** id = 3**;**

SQL> **commit;**

В ходе эксперимента выполняем неподтвержденное изменение данных таблицы AWARDS. При этом параллельно выполняется транзакция с уровнем изоляции READ COMMITTED, содержащая запрос SELECT. Транзакция не выполняется ("`зависает"'), т.к. она обращается к неподтвержденным данным, но после подтверждения изменения мы увидим измененное содержимое таблицы.

* 1. SNAPSHOT

SQL> **select** \* **from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

3 new type 150.00000

SQL> **set transaction isolation level snapshot;**

**Commit** current **transaction** (y/n)?y

Committing.

/\*Выполняем после COMMIT на втором терминале\*/

SQL> **select \* from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

3 new type 150.00000

SQL> **update** type **set** name = 'new type' **where** id = 3**;**

Statement failed, **SQLSTATE** = 40001

deadlock

-**update** conflicts with concurrent **update**

-concurrent **transaction number** is 2799

На другом терминале выполняем:

SQL> **update** type **set** name = 'another' **where** id = 3**;**

SQL> **commit;**

Видим, что транзакция, содержащая SELECT, вывела "`старые"' данные, подтвержденные на момент старта транзакции. При таком уровне изоляции даже подтвержденные изменения данных из других транзакций не видны. Можно заметить, что при изменении данных в рамках сделанного снимка базы возникает конфликт между двумя транзакциями. В конечном итоге в силу вступили изменения из второго терминала.

* 1. SNAPSHOT TABLE STABILITY

SQL> **select \* from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

3 new type 150.00000

SQL> **set transaction isolation level snapshot table stability;**

**Commit** current **transaction** (y/n)?y

Committing.

SQL> **update** type **set** name = 'another' **where** id = 3**;**

SQL> **commit;**

SQL> **select \* from** type**;**

ID NAME COST

============ ==================== ==============

0 build matherials 100.00000

1 furniture 200.00000

2 technics 300.00000

3 another 150.00000

На другом терминале выполняем:

SQL> **update** type **set** name = 'new type' **where** id = 3**;**

/\*Запрос “висит” до команды COMMIT на первом терминале\*/

Statement failed, **SQLSTATE** = 40001

deadlock

-**update** conflicts with concurrent **update**

-concurrent **transaction number** is 2801

Так же, как и в предыдущем случае, возник конфликт между двумя транзакциями, но в этот раз транзакция с уровнем изоляции SNAPSHOT TABLE STABILITY запретила изменение таблицы из других транзакций. В этом случае в силу вступили изменения первого терминала.

**Выводы**

В ходе выполнения работы было изучено создание и управление транзакциями, а также на примерах рассмотрены уровни изоляции.

Транзакции позволяют выполнять контроль целостности данных при выполнении различных запросов, но если требовать последовательной обработки транзакций, то возможно уменьшение производительности, т.к. при параллельном выполнении транзакций возможны следующие проблемы:

* Грязное чтение (Dirty Read) - чтение данных, добавленных или измененных транзакцией, которая впоследствии не подтвердится (откатится);
* Размытое чтение (Fuzzy Read) - при повторном чтении в рамках одной транзакции, ранее прочитанные данные оказываются измененными;
* Фантомное чтение (Phantom Read) - одна транзакция в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям, другая транзакция в интервалах между этими выборками добавляет или удаляет строки или изменяет столбцы некоторых строк, используемых в критериях выборки первой транзакции, и успешно заканчивается; в результате получится, что одни и те же выборки в первой транзакции дают разные множества строк.

Для достижения компромисса между быстродействием существуют несколько уровней изоляции транзакций. При этом можно выделить соответствие между уровнями изоляции и возможными конфликтами:

|  |  |  |  |
| --- | --- | --- | --- |
| **Уровень изоляции** | **Dirty Read** | **Fuzzy Read** | **Phantom Read** |
| READ UNCOMMITED | Возможно | Возможно | Возможно |
| READ COMMITED | Невозможно | Возможно | Возможно |
| REPEATABLE READ | Невозможно | Невозможно | Возможно |
| SERIALIZABLE | Невозможно | Невозможно | Невозможно |

Таким образом, сериализуемость - это способность к упорядочению параллельной обработки транзакций. Также можно установить соответствие между уровнями изоляции по SQL-стандарту и уровнями изоляции транзакций в Firebird SQL:

|  |  |
| --- | --- |
| **Уровни изоляции по SQL** | **Уровни изоляции транзакций в Firebird** |
| READ COMMITED | READ COMMITED |
| REPEATABLE READ | SNAPSHOT |
| SERIALIZABLE | SNAPSHOT TABLE STABILITY |