Санкт-Петербургский политехнический университет

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

ОТЧЕТ

По лабораторным работам

по дисциплине "Базы данных"

Выполнил: студент гр. 43501/3 Бабарицкий П.А.

Проверил: Мяснов А.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/подпись преподавателя, дата/

Санкт-Петербург

2016

1. **Изучение работы транзакций**
   1. **Цель работы**

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

* 1. **Программа работы**
     + Изучить основные принципы работы транзакций.
     + Провести эксперименты по запуску, подтверждению и откату транзакций.
     + Разобраться с уровнями изоляции транзакций в Firebird.
     + Спланировать и провести эксперименты, показывающие основные возможности транзакций с различным уровнем изоляции.
  2. **Выполнение работы**

Транзакция — это совокупность операций, которая должна либо быть выполнена целиком, либо не выполнена вообще. Транзакция заканчивается либо когда будет подтверждена (commit) и все изменения, совершенные ей, запишутся в базу данных, либо когда будет откачена (rollback) и никаких изменений в базу данных внесено не будет.

Любая последовательность действий в Firebird неявно считается транзакцией, от начала программы до commit или rollback.

При использовании транзакций желательно полностью оградить данные, которые принимают в ней участие от других пользователей, чтобы они не могли их испортить. Потенциально при одновременной работе двух и более транзакций с одними данными могут возникать следующие ошибки:

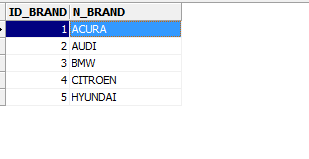
* Потерянное обновление — одна из транзакций изменяет данные, а вторая изменяет эти же данные сразу за первой транзакцией, при этом результат первой транзакции теряется
* «Грязное» чтение — одна из транзакций изменяет данные, вторая читает эти данные. После этого первая транзакция откатывает изменения и данные, полученные второй транзакцией, оказываются неверны.
* Неповторяющееся чтение — одна из транзакций читает данные, а вторая после этого их изменяет. Тогда повторное чтение первой транзакцией тех же данных приведет к другим результатам.
* Фантомная вставка — одна из транзакций выполняет оператор, использующий несколько полей, затем другая транзакция вставляет строку. После этого, будучи заново вызванным в первой транзакции, оператор может вернуть другое значение.

ID\_BRAND N\_BRAND

Проверка неявных транзакций:

Запущено два сеанса связи с базой данных. Первый сеанс читает таблицу.

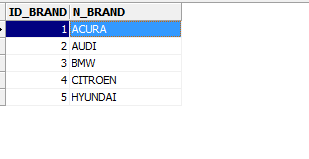
SELECT \* from brends



Второй пишет в нее.

insert into BRENDS values (6,'HONDA');Первый читает эту таблицу снова и получает старые данные

SELECT \* from brends



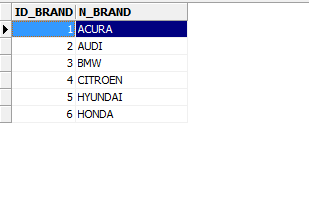
Даже после подтверждения второй транзакции

commit;

Первый читает старые данные. И только когда он сам подтверждает транзакцию чтения, он начинает видеть новые данные.

commit;

SELECT \* from brends



Проверка откатов транзакций:

Запущено два сеанса связи с базой данных. Первый сеанс читает таблицу.

Второй пишет в нее и откатывает транзакцию.

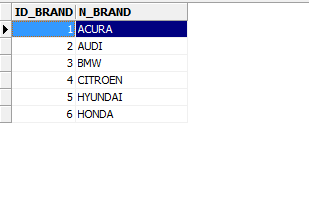
insert into BRENDS values (7,'TOYOTA');

rollback;

После этого подтверждаем первую транзакцию и проверяем из нее содержимое таблицы

commit;

select \* from brends;



Как ожидалось, таблица не изменилась после завершения обоих транзакций.

Рассмотрение различных уровней изоляции:

Для борьбы с ошибками существуют различные уровни изоляции. Чем большая изоляция осуществляется, тем больше требуется ресурсов.

READ UNCOMMITED

Самая слабая защита, защищает только от одновременной записи данных и их порчи. Данные доступны всем, как только были изменены, до подтверждения. В Firebird не реализована.

READ COMMITTED

Устанавливается как:

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

Устанавливает блокировку доступа к данным со стороны других транзакций, которые были изменены, но еще не были подтверждены. Позволяет избежать ошибок потерянного обновления и «грязного» чтения.

Потерянное обновление

Транзакция 1

update clients set tel = '89621609490' where id\_client = 1;

Транзакция 2

update clients set tel = '89621609490' where id\_client = 1;

Вторая транзакция начинает выполнять обновление, но блокируется до окончания первой транзакции, соответственно, ошибка потерянного обновления исключается.

Аналогичная ситуация с «грязным» чтением, т.е. чтением между изменением данных и их откатом.

Транзакция 1

update clients set tel = '89621609490' where id\_client = 1;

Транзакция 2

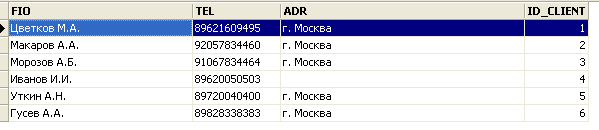
select \* from clients;

Транзакция 1

rollback;

Транзакция 2

select \* from clients;

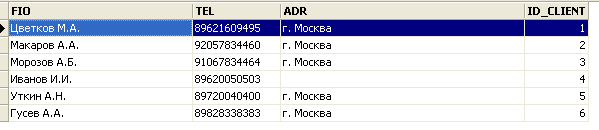


В связи с тем, что данные блокируются только при изменении, а при чтении не блокируются, от ошибок неповторяющееся чтение и фантомная вставка READ COMMITTED не защищает.

Неповторяющееся чтение:

Транзакция 2

select \* from clients;



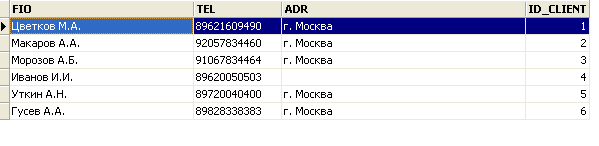
Транзакция 1

update clients set tel = '89621609490' where id\_client = 1;

commit;

Транзакция 2

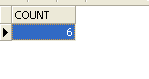
SQL> select \* from clients;



Фантомная вставка:

Транзакция 2

select count(\*) from clients;



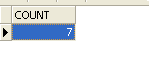
Транзакция 1

insert into Clients values('Мягков А.Б.','9106783000', 'г. Москва', 7)

commit;

Транзакция 2

select count(\*) from clients;



SNAPHOT

Устанавливается как

SET TRANSACTION ISOLATION LEVEL SNAPSHOT

Не блокирует память, а создает «снимок» базы на момент начала трансляции и все данные при запросах берутся из него. Поэтому «грязное» чтение, неповторяющееся чтение и фантомная вставка не возникают. Проблема с потерянным обновлением решается так же, как и в READ COMMITTED, но если после снятия блокировки оказывается, что данные которые мы собираемся изменить, только что были изменены, то выдается ошибка о конфликте.

Потерянное обновление:

Транзакция 1

update clients set tel = '89621609491' where id\_client = 1;

Транзакция 2

update clients set tel = '89621609492' where id\_client = 1;

Транзакция 1

commit;

Транзакция 2

Statement failed, SQLSTATE = 40001

deadlock

-update conflicts with concurrent update

-concurrent transaction number is 143

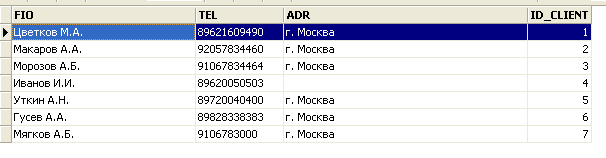
«Грязное» чтение:

Транзакция 1

update clients set tel = '89621609499' where id\_client = 4;

Транзакция 2

SQL> select \* from clients;

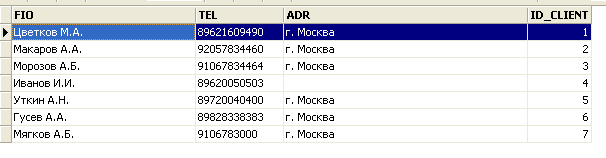


Транзакция 1

rollback;

Транзакция 2

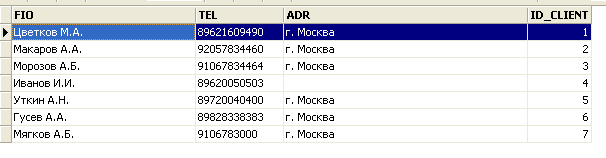
select \* from clients;



Неповторяющееся чтение:

Транзакция 2

select \* from clients;



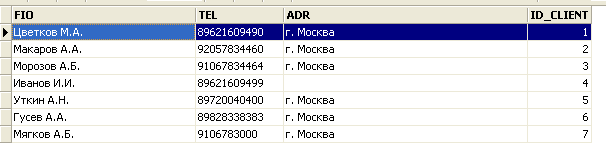
Транзакция 1

update clients set tel = '89621609499' where id\_client = 4;

SQL> commit;

Транзакция 2

SQL> select \* from clients;



SNAPHOT TABLE STABILITY

Устанавливается как

SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY

Отличается от предыдущего тем, что полностью блокирует изменение данных в таблице, что и избавляет от всех ошибок, блокируя записывающие транзакции, и не изменяя таблицу.

Потерянное обновление:

Транзакция 1

update clients set tel = '89621609499' where id\_client = 4;

Транзакция 2

update clients set tel = '89621609491' where id\_client = 4;

Транзакция 1

SQL> commit;

Транзакция 2

Statement failed, SQLSTATE = 40001

deadlock

-update conflicts with concurrent update

-concurrent transaction number is 143

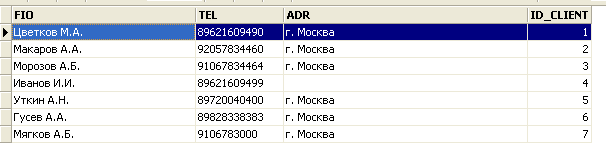
«Грязное» чтение:

Транзакция 1

update clients set tel = '89621609400' where id\_client = 4;

Транзакция 2

SQL> select \* from clients;



commit;

Транзакция 1

Statement failed, SQLSTATE = 40001

deadlock

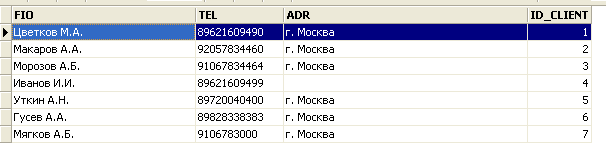
-update conflicts with concurrent update

-concurrent transaction number is 443

Неповторяющееся чтение:

Транзакция 2

SQL> select \* from clients;



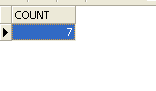
Транзакция 1

update clients set tel = '89621609400' where id\_client = 4;

Фантомная вставка:

Транзакция 2

SQL> select count(\*) from clients;



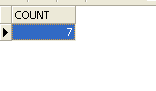
Транзакция 1

insert into Clients values('Мягков А.Б.','9106783000', 'г. Москва', 8);

SQL> commit;

Транзакция 2

SQL> select count(\*) from clients;



Таким образом уровни изоляции транзакций можно привести в виде таблицы, определяющей от каких ошибок они защищают:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Уровень изоляции | неповторяющееся чтение | "грязное" чтение | потерянное обновление | фантомная вставка |
| READ UNCOMMITED | - | - | - | - |
| READ COMMITED | - | + | + | - |
| SNAPHOT | + | + | + | - |
| SNAPHOT TABLE STABILITY | + | + | + | + |

* 1. **Выводы**

По стандарту существуют 4 вида транзакций: UNCOMMITTED READ, COMMITTED READ, REPEATABLE READ, SERIALIZABLE.

В Firebird есть три уровня изоляции: COMMITTED READ, SNAPSHOT и SNAPSHOT TABLE STABILITY, которые соответственно реализуют COMMITTED READ, REPEATABLE READ, SERIALIZABLE.

COMMITTED READ реализуется прямо по стандарту, обеспечивая блокировкой записи отдельных ячеек защиту от потерянного обновления и «грязного» чтения.

SNAPSHOT создает «слепок» базы данных на момент начала транзакции и блокирует только ячейки, в которые была произведена запись. Благодаря этому также обеспечивается защита от потерянного обновления и «грязного» чтения, а так же от неповторяемого чтения и «фантомной» вставки. Стандартный уровень REPEATABLE READ требует только защиты от неповторяемого чтения, так что SNAPSHOT реализует этот уровень даже с избытком.

SNAPSHOT TABLE STABILITY не только создает «слепок», но и обеспечивает блокирование используемых таблиц целиком, что не только защищает от всех 4 видов ошибок, но и обеспечивает последовательное выполнение транзакций, работающих с одними и теми же таблицами, то есть тех, которые могут повлиять друг на друга. Соответственно, этот уровень удовлетворяет требованиям уровня SERIALIZABLE.

Выбор уровня должен исходить из требований к необходимости параллельной обработки запросов и структуры системы (например, отсутствие возможности возникновения некоторых ошибок). Слишком высокий уровень изоляции замедлит работу системы, слишком низкий может привести к ошибкам в работе.

При использовании уровней изоляции с блокировкой отдельных записей и тем более таблиц целиком, не стоит забывать о возможности взаимной блокировки (deadlock) и пытаться избегать ее.