Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе

Дисциплина: «Базы данных»
Тема: «Изучение работы транзакций»

Работу выполнила:
Мартюшева Надежда
Группа: 43501/3
Преподаватель:
Мяснов Александр Владимирович

Санкт-Петербург
2017

1 Цели работы

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

2 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.

3 Ход работы

Транзакция - это атомарное действие над БД, переводящее ее из одного целостного состояния в другое целостное состояние. Другими словами, транзакция - это последовательность операций, которые должны быть или все выполнены или все не выполнены.

В Firebird начало транзакции может задаваться явно командой SET TRANSACTION, либо предполагаться неявным, т.е. очередная транзакция открывается автоматически сразу же после удачного или неудачного завершения предыдущей.

Для завершения транзакции используют следующие команды SQL:

- COMMIT - подтвердить транзакцию, то есть подтвердить изменения, совершенные над БД в рамках данной транзакции;
- ROLLBACK - откатить транзакцию, то есть вернуть БД в состояние, в котором она находилась на момент начала транзакции.

Запуск и подтверждение транзакций

Запустим два сеанса связи с базой данных. Осуществим выборку всех строк из таблицы JUDGE в первом сеансе (транзакция 1). Во втором сеансе (транзакция 2) добавим новую запись в таблицу, при выводе отобразилась таблица с уже измененными данными. Подтвердим транзакцию 2 (рисунок 2).

В транзакции 1 выведем таблицу JUDGE. В ней отображаются старые данные. Теперь подтвердим транзакцию 1 и снова выведем таблицу JUDGE. Отобразилась измененная таблица (рисунок 1)

Откат транзакций

В первом сеансе неявно начата транзакция 4, а во втором – транзакция 3. В рамках транзакции 3 удалим из таблицы JUDGE запись и выведем таблицу, которая содержит актуальные данные. Затем откатим транзакцию 3 и снова выведем таблицу JUDGE. Данные вернулись в исходное состояние (рисунок 3).

В рамках транзакции 4 выведем таблицу JUDGE до начала и после завершения транзакции 3. В обоих случаях данные одинаковы (рисунок 4).

Таким образом мы видим, что транзакции видны лишь те изменения, фиксация которых произошла не позднее момента старта этой транзакции. Чтобы изменения, внесенные и подтвержденные другой транзакцией, нужно завершить транзакцию (подтвердить её или выполнить полный откат) и запустить транзакцию заново (начать новую транзакцию).

При использовании транзакций необходимо оградить данные, которые принимают в ней участие от других пользователей, чтобы они не могли их повредить. 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;

```
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14

SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14

SQL> commit;
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            1 2017-01-13
```

Рис. 1: Сеанс 1, транзакция 1. Наблюдение изменений данных в таблице JUDGE

```
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14

SQL> insert into judge values (25, 8595, 1, '13.01.2017');
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            1 2017-01-13

SQL> commit;
```

Рис. 2: Сеанс 2, транзакция 2. Добавление записи в таблицу JUDGE и подтвнрждение транзакции

- 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.

Для борьбы с ошибками существуют различные уровни изоляции. Уровень изолированности транзакций — это степень изолированности одной транзакции от другой. Изменения, внесённые некоторым оператором, будут видны всем последующим операторам, запущенным в рамках этой же транзакции, независимо от её уровня изолированности, а вот изменения произведённые в рамках другой транзакции остаются невидимыми для текущей транзакции до тех пор пока они не подтверждены. Уровень изолированности, а иногда, другие атрибуты, определяет, как транзакции будут взаимодействовать с

```
SQL> delete from judge where judge_id = 25;
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14

SQL> rollback;
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            1 2017-01-13
```

Рис. 3: Сеанс 2, транзакция 3. Внесение изменений в таблицу JUDGE и их откат

```
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            1 2017-01-13

SQL> commit;
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            1 2017-01-13
```

Рис. 4: Сеанс 1, транзакция 4. Вывод таблицы JUDGE до начала и после завершения транзакции 3

другой транзакцией, которая хочет подтвердить изменения.

Уровень изоляции - это самая важная характеристика транзакции, которая определяет её поведение по отношению к другим одновременно выполняющимся транзакциям. Существует три уровня изолированности транзакции:

- SNAPSHOT,
- SNAPSHOT TABLE STABILITY,
- READ COMMITTED с двумя уточнениями (NO RECORD_VERSION и RECORD_VERSION).

Рассмотрим каждый из этих уровней изоляции на примерах.

Уровень изолированности SNAPSHOT

Уровень изолированности SNAPSHOT (используется по умолчанию) означает, что этой транзакции видны лишь те изменения, фиксация которых произошла не позднее момента старта этой транзакции.

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 следующим образом: выполнение второй транзакции приостанавливается до момента завершения первой транзакции. если после снятия блокировки оказывается, что данные которые мы собираемся изменить, только что были изменены, то выдается ошибка о конфликте.

Потеряное обновление

В транзакции 1 обнавляем строку в таблице. В транзакции 2 пытаемся обновить ту же строку, видим, что выполнение команды обновления приостановлено. Завершаем транзакцию 1 командой COMMIT (рисунок 5). Транзакция 2 завершается ошибкой (рисунок 6).

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
Commit current transaction (y/n)?y
Committing.
SQL> update JUDGE set rank = 2 where judge_id = 25;
SQL> commit;
```

Рис. 5: SNAPSHOT. Транзакция 1. Потеряное обновление

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
Commit current transaction (y/n)?y
Committing.
SQL> update JUDGE set rank = 1 where judge_id = 25;
Statement failed, SQLSTATE = 40001
deadlock
-update conflicts with concurrent update
-concurrent transaction number is 8274
```

Рис. 6: SNAPSHOT. Транзакция 2. Потеряное обновление

«Грязное» чтение

В транзакции 1 обнавляем строку в таблице. В транзакции 2 выводим эту таблицу, видим изначальные данные. Откатываем транзакцию 1 командой ROLLBACK (рисунок 7). В транзакции 2 отображаются корректные данные (рисунок 8).

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
SQL> update JUDGE set rank = 1 where judge_id = 25;
SQL> rollback;
```

Рис. 7: SNAPSHOT. Транзакция 1. «Грязное» чтение

Неповторяющееся чтение

Начинаем транзакцию 1, выводим таблицу. Начинаем транзакцию 2, изменяем данные в таблице, подтверждаем транзакцию (рисунок 10). В транзакции 1 снова выводим таблицу, данные остались неизменными (рисунок 9).

Фантомная вставка

Начинаем тринзакцию 1, выводим количество строк в таблице. Начинаем транзакцию 2, добавляем новые данные в таблицу, подтверждаем транзакцию (рисунок 12). В транзакции 1 снова выводим количество строк в таблице, данные остались неизменными (рисунок 11).

Уровень изолированности SNAPSHOT TABLE STABILITY

Уровень изоляции транзакции SNAPSHOT TABLE STABILITY позволяет, как и в случае SNAPSHOT, также видеть только те изменения, фиксация которых произошла не позднее момента старта этой транзакции. При этом после старта такой транзакции в других клиентских транзакциях невозможно выполнение изменений ни в каких таблицах этой базы данных, уже каким-либо образом измененных первой транзакцией. Все такие попытки в параллельных транзакциях приведут к исключениям базы данных. Просматривать любые данные другие транзакции могут совершенно свободно.

Данный уровень изоляции позволяет избежать всех возможных ошибок.

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            2 2017-01-13

SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            2 2017-01-13

SQL> commit;
```

Рис. 8: SNAPSHOT. Транзакция 2. «Грязное» чтение

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            2 2017-01-13

SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            2 2017-01-13

SQL> commit;
```

Рис. 9: SNAPSHOT. Транзакция 1. Неповторяющееся чтение

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
SQL> update JUDGE set rank = 1 where judge_id = 25;
SQL> commit;
```

Рис. 10: SNAPSHOT. Транзакция 2. Неповторяющееся чтение

Потеряное обновление

В транзакции 1 обнавляем строку в таблице. В транзакции 2 пытаемся обновить ту же строку, видим, что выполнение команды обновления приостановлено. Завершаем транзакцию 1 командой COMMIT (рисунок 13). Транзакция 2 завершается ошибкой (рисунок 14).

«Грязное» чтение

В транзакции 1 обнавляем строку в таблице. В транзакции 2 выводим эту таблицу, видим, что выполнение команды вывода таблицы приостановлено. Откатываем транзакцию 1 командой ROLLBACK (рисунок 15). В транзакции 2 отображаются корректные данные (рисунок 16).

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
SQL> select count(*) from judge;

       COUNT
============
           8

SQL> select count(*) from judge;

       COUNT
============
           8

SQL> commit;
```

Рис. 11: SNAPSHOT. Транзакция 1. Фантомная вставка

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
SQL> insert into judge values (35, 9077, 1, '13.01.2017');
SQL> commit;
```

Рис. 12: SNAPSHOT. Транзакция 2. Фантомная вставка

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY;
SQL> update JUDGE set rank = 3 where judge_id = 25;
SQL> commit;
```

Рис. 13: SNAPSHOT TABLE STABILITY. Транзакция 1. Потеряное обновление

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY;
SQL> update JUDGE set rank = 1 where judge_id = 25;
Statement failed, SQLSTATE = 40001
deadlock
-update conflicts with concurrent update
-concurrent transaction number is 8296
```

Рис. 14: SNAPSHOT TABLE STABILITY. Транзакция 2. Потеряное обновление

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY;
SQL> update JUDGE set rank = 2 where judge_id = 25;
SQL> rollback;
```

Рис. 15: SNAPSHOT TABLE STABILITY. Транзакция 1. «Грязное» чтение

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY;
SQL> select * from judge;

    JUDGE_ID       ASH_ID         RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            3 2017-01-13
          35         9077            1 2017-01-13

SQL> commit;
```

Рис. 16: SNAPSHOT TABLE STABILITY. Транзакция 2. «Грязное» чтение

Неповторяющееся чтение

Начинаем транзакцию 1, выводим таблицу. Начинаем транзакцию 2, изменяем данные в таблице, видим, что выполнение команды обновления приостановлено. В транзакции 1 снова выводим таблицу, данные остались неизменными. Завершаем транзакцию 1 (рисунок 17). После завершения транзакции 1 в транзакции 2 выполнилась команда изменения данных. Завершаем транзакцию 2 (рисунок 18).

Фантомная вставка

Начинаем тринзакцию 1, выводим количество строк в таблице. Начинаем транзакцию 2, удаляем данные из таблицы, видим, что выполнение команды обновления приостановлено. В транзакции 1 снова выводим количество строк в таблице, данные остались неизменными. Завершаем транзакцию 1 (рисунок 19). После завершения транзакции 1 в транзакции 2 выполнилась команда изменения данных. Завершаем транзакцию 2 (рисунок 20).

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY;
SQL> select * from judge;

    JUDGE_ID       ASH_ID         RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            3 2017-01-13
          35         9077            1 2017-01-13

SQL> select * from judge;

    JUDGE_ID       ASH_ID         RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            3 2017-01-13
          35         9077            1 2017-01-13

SQL> commit;
```

Рис. 17: SNAPSHOT TABLE STABILITY. Транзакция 1. Неповторяющееся чтение

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY;
SQL> update JUDGE set rank = 1 where judge_id = 25;
SQL> commit;
```

Рис. 18: SNAPSHOT TABLE STABILITY. Транзакция 2. Неповторяющееся чтение

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY;
SQL> select count(*) from judge;

       COUNT
============
           9

SQL> select count(*) from judge;

       COUNT
============
           9

SQL> commit;
```

Рис. 19: SNAPSHOT TABLE STABILITY. Транзакция 1. Фантомная вставка

```
SQL> SET TRANSACTION ISOLATION LEVEL SNAPSHOT TABLE STABILITY;
SQL> delete from judge where ash_id = 9077;
SQL> commit;
```

Рис. 20: SNAPSHOT TABLE STABILITY. Транзакция 2. Фантомная вставка

Уровень изолированности READ COMMITTED

Уровень изолированности READ COMMITTED позволяет в транзакции без её перезапуска видеть все подтверждённые изменения данных базы данных, выполненные в других параллельных транзакциях. Неподтверждённые изменения не видны в транзакции и этого уровня изоляции. Для получения обновлённого списка строк интересующей таблицы необходимо лишь повторное выполнение оператора SELECT в рамках активной транзакции READ COMMITTED без её перезапуска.

Данный уровень изоляции позволяет избежать ошибок потерянного обновления и «грязного» чтения.

Потеряное обновление

В транзакции 1 обнавляем строку в таблице. В транзакции 2 пытаемся обновить ту же строку, видим, что выполнение команды обновления приостановлено. Завершаем транзакцию 1 командой COMMIT (рисунок 21). В транзакции 2 выполняется команда обновления строки. Завершим транзакция 2 (рисунок 23). Перезапустим транзакцию 1 и выведем данные. Данные содержат изменения внесенные в транзакции 2 (рисунок 22).

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SQL> update JUDGE set rank = 2 where judge_id = 25;
SQL> commit;
```

Рис. 21: READ COMMITTED. Транзакция 1. Потеряное обновление

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            1 2017-01-13

SQL> commit;
```

Рис. 22: READ COMMITTED. Транзакция 1, перезапуск. Потеряное обновление

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SQL> update JUDGE set rank = 1 where judge_id = 25;
SQL> commit;
```

Рис. 23: READ COMMITTED. Транзакция 2. Потеряное обновление

«Грязное» чтение

В транзакции 1 обнавляем строку в таблице. В транзакции 2 выводим эту таблицу, видим, что выполнение команды вывода приостановлено. Откатываем транзакцию 1 командой ROLLBACK (рисунок 24). В транзакции 2 отображаются корректные данные (рисунок 25).

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SQL> update JUDGE set rank = 2 where judge_id = 25;
SQL> rollback;
```

Рис. 24: READ COMMITTED. Транзакция 1. «Грязное» чтение

```
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ==========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            1 2017-01-13

SQL> commit;
```

Рис. 25: READ COMMITTED. Транзакция 2. «Грязное» чтение

Неповторяющееся чтение

Начинаем транзакцию 1, выводим таблицу. Начинаем транзакцию 2, изменяем данные в таблице, подтверждаем транзакцию (рисунок 27). В транзакции 1 снова выводим таблицу, отображаются уже измененные данные (рисунок 26).

Фантомная вставка

Начинаем тринзакцию 1, выводим количество строк в таблице. Начинаем транзакцию 2, добавляем новые данные в таблицу, подтверждаем транзакцию (рисунок 29). В транзакции 1 снова выводим количество строк в таблице, отображаются уже измененные данные (рисунок 28).

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ===========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            1 2017-01-13

SQL> select * from judge;

    JUDGE_ID       ASH_ID          RANK  DATE_VALID
============ ============ ============ ===========
           1          627            1 2009-04-04
           2         5993            3 2016-01-29
           3          502            1 2009-01-10
           4         1185            1 2009-07-21
           5         1161            3 2012-09-30
           6         2207            1 2011-12-12
          10         2767            3 2016-02-14
          25         8595            2 2017-01-13

SQL> commit;
```

Рис. 26: READ COMMITTED. Транзакция 1. Неповторяющееся чтение

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SQL> update JUDGE set rank = 2 where judge_id = 25;
SQL> commit;
```

Рис. 27: READ COMMITTED. Транзакция 2. Неповторяющееся чтение

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SQL> select count(*) from judge;

       COUNT
============
           8

SQL> select count(*) from judge;

       COUNT
============
           9

SQL> commit;
```

Рис. 28: READ COMMITTED. Транзакция 1. Фантомная вставка

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SQL> insert into judge values (35, 9077, 1, '13.01.2017');
SQL> commit;
```

Рис. 29: READ COMMITTED. Транзакция 2. Фантомная вставка

4 Вывод

В ходе работы я познакомилась с основами проектирования баз данных, созданием SQL-диаграмм.

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 и SERIALIZABLE.

COMMITTED READ реализуется соответственно стандарту, обеспечивая защиту от потерянного обновления и «грязного» чтения при помощи блокировки записи отдельных ячеек.

SNAPSHOT создает «слепок» базы данных на момент начала транзакции и блокирует только ячейки, в которые была произведена запись. Благодаря этому обеспечивается защита от потерянного обновления и «грязного» чтения, а так же от неповторяемого чтения и «фантомной» вставки. По стандарту уровень изоляции 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 четырех видов ошибок, но и обеспечивает последовательное выполнение транзакций, работающих с одними и теми же таблицами. 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.