Транзакции firebird 2.5

1. Исследование уровня транзакций Snapshot

T1 – Tn – транзакции

T1: уровень Snapshot

```
select * from types where type_id=0
```

T2: уровень Read Commited

```
update types set title_type='Contract' where type_id=0
```

Запустим на выполнение T1, и T2 и подтверждая T2, получим:

TYPE_ID	TITLE_TYPE
0	New title

Это показывает то что T1 использует старую информацию, ее слепок до выполнения всех транзакций. Вернем значение обратно в New title.

2. Исследование Read Commited

T1: read commited

T2: rw stability

```
update types set title_type='Contract' where type_id=0
```

Запускаем T2 не подтверждая, получаем результаты первой транзакции:

TYPE_ID	TITLE_TYPE
0	New title

Попробуем записать первой транзакцией что-нибудь в это таблицу.

Messages

Unsuccessful execution caused by system error that does not preclude successful execution of subsequent statements.
lock conflict on no wait transaction.

Это говорит о том что доступ к этой таблице возможен только для чтения, а на запись она заблокирована транзакцией T2 которая стартовала как RW stability.

Теперь подтвердим T2. Получим результат первой транзакции:

TYPE_ID	TITLE_TYPE
0	Contract

Как видно данные обновились, тогда как в случае с Snapshot нам показывалась старая информация, можно сказать что read commited показывает все подтвержденные транзакции, а снимок - соответственно показывает данные которые были подтверждены до запуска транзакции.

3. RW Table Stability

Данные уровни существуют для предотвращения одновременного доступа к ресурсу, например одной и той же записи в таблице. Если запустить транзакцию в режиме **RW stability** то это приведет к тому, что другие транзакции не смогут получить доступ к данной таблице для записи, а только для чтения. Продемонстрируем:

T1: RW stability

```
update types set title_type='Contract' where type_id=0
```

T2: Read commited

```
update types set title_type='Others 5' where type_id=4
```

После выполнения T2 и не завершения T1 получим:

Messages

Unsuccessful execution caused by system error that does not preclude successful execution of subsequent statements.
lock conflict on no wait transaction.

T2:

```
select first 2 * from types
```

TYPE_ID	TITLE_TYPE
0	Contract_5
1	Contracct_additional

Как видно, доступ на чтение есть.

T2:

```
update types set title_type='Others 5' where type_id=4
```

Завершим T1 и запустим T2, получим:

TYPE_ID	TITLE_TYPE
0	Contract_5
1	Contracct_additional
2	TZ
3	Fixes
4	Others 5

Как видно данные обновились, тоесть T2 получило доступ к таблице после окончания транзакции T1.

T1: RW stability

```
select * from types where type_id=0
```

T2: RO stability

```
select first 2 * from types
```

TYPE_ID	TITLE_TYPE
0	Contract_5
1	Contracct_additional

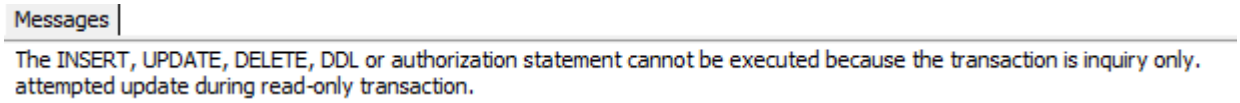
Таким образом операции выборки из БД не влияют на доступ к таблицам в режиме RW stability.

4. RO Table Stability

T1: **RO stability**

```
update types set title_type='Contract_8' where type_id=0
```

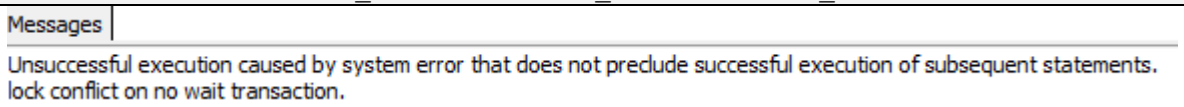
Результат:



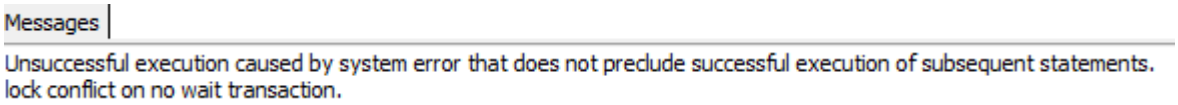
Что говорим о том, что это должна быть операция чтения.

T2: RW stability

```
update types set title_type='Contract_5' where type_id=0
```



T2: Read commited



Собственно до закрытия транзакции T1 нет возможности ни писать, ни читать из базы.

T1: **RO Stability**

```
select first 2 * from types
```

T2: RO stability, RW stability

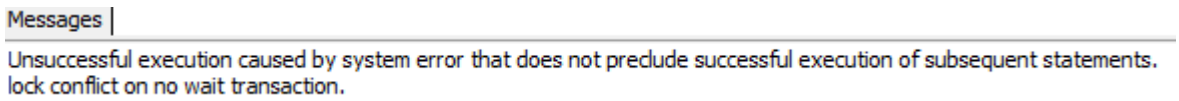
```
select * from types where type_id=0
```

Данные варианты возможны.

T2: RW Stability, Read commited, Snapshot

```
update types set title_type='Contract_8' where type_id=0
```

Это невозможно.



Выводы:

В работе исследовались различные уровни изоляций для firebird 2.5. В качестве вывода можно сформулировать некоторые правила по использованию различных уровней изоляции.

1. Когда требуются актуальные данные, необходимо использовать Read commited. Но данный тип транзакции не обеспечивает повторимости чтения, так как подтвержденные данные становятся видны этой транзакции.
2. Когда необходимо предотвратить изменение одних и тех же данных, можно выставить на эту транзакцию уровень изоляции RW stability, это позволит запретить запись в таблицу других процессов, но оставит чтение.
3. Слепок может обеспечит повторимость чтения, но обладает не самой актуальной информацией, так как создает список транзакций завершенных до ее старта.

Разработка триггеров

1. Триггер который заполняет ключевое поле

В отличие от mysql в firebird отсутствует возможность использования автоинкремента «из коробки» для этого можно создать генератор или последовательность и создать триггеры который будет высчитывать новый идентификатор перед записью в таблицу.

```
CREATE SEQUENCE GENERATORID;
ALTER SEQUENCE GENERATORID RESTART WITH 0;

// создаем триггера для автоинкремента
CREATE OR ALTER TRIGGER TYPES_AUTOINCREMENT FOR TYPES
ACTIVE BEFORE INSERT POSITION 0
as
begin
  if ((new.TYPE_ID is null) or (new.TYPE_ID = 0)) then
  begin
    new.TYPE_ID = gen_id(GENERATORID, 1);
  end
end
^
SET TERM ; ^
```

Пример использования

```
insert into types (title_type, cat) values('Smth', '/category')
```

Здесь триггер самостоятельно рассчитает идентификатор для новой записи.
Результат:

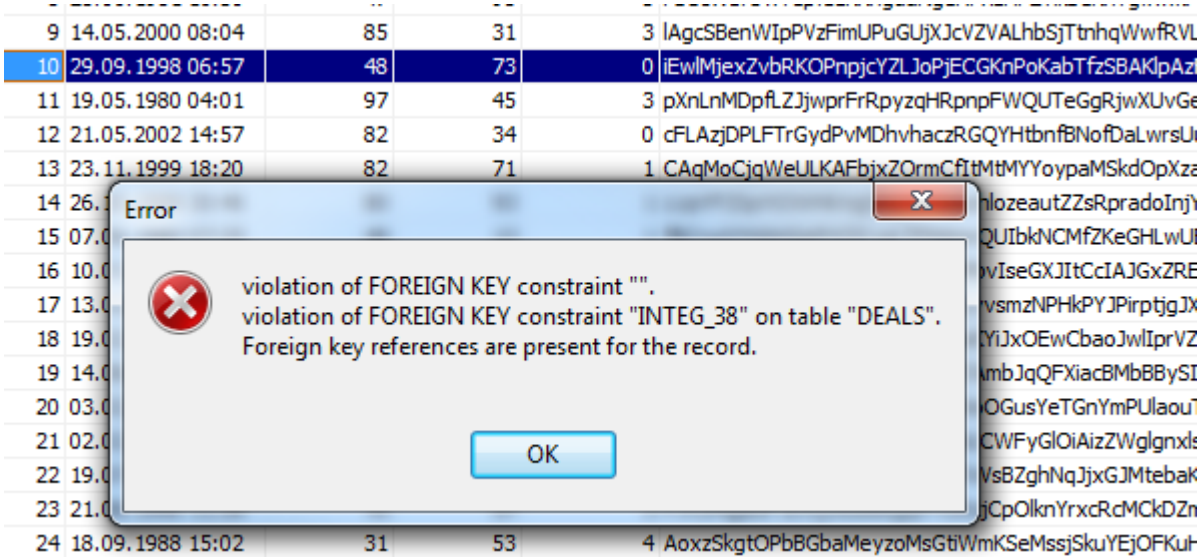
TYPE_ID	TITLE_TYPE
0	Contract_5
1	Contracct_additional
2	TZ
3	Fixes
4	Others 5
5	Smth

2. Триггер для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице

При удалении документа происходит нарушение целостности, так как запись в таблице сделок становится бессмысленной если не будет существовать документа с идентификатором который указан в этой записи, для этого создадим 2 триггера на удаление и на обновление данных. Если удаляется документ – удаляем сделку, если у документа меняется идентификатор – меняем идентификатор документа в таблицу сделок.

Пример:

Выключим триггер, попробуем удалить запись о документе.



Включим триггер:

До удаления документа.

```
select * from deals where d_id='10'
```

DEAL_ID	W_ID	C_ID	S_ID	D_ID	DEADLINE	START_DATE	MONEY
14	17	24	3	10	30.05.2000 17:19	12.12.1999 05:14	127 561
31	52	72	0	10	06.10.1992 21:47	14.03.1998 11:07	657 991

После удаления документа и подтверждения транзакции.

DEAL_ID	W_ID	C_ID	S_ID	D_ID	DEADLINE	START_DATE	MONEY
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>

Как видим сделки удалились.

Продемонстрируем изменение о идентификаторе документа.

```
select * from deals where d_id='6'
```

DEAL_ID	W_ID	C_ID	S_ID	D_ID	DEADLINE	START_DATE	MONEY
18	87	65	3	6	<null>	04.03.2004 21:17	35 155
184	44	82	2	6	<null>	<null>	99 267

Изменим значение документа на 666666, и проверим эти записи в таблице сделок.

```
select * from deals where deal_id='18'
```

DEAL_ID	W_ID	C_ID	S_ID	D_ID	DEADLINE	START_DATE	MONEY
18	87	65	3	666 666	<null>	04.03.2004 21:17	35 155

Как видно триггера работает верно.

Разработка индивидуальной хранимой процедуры

Желаем расформировать какой-либо департамент, и все встречи которые назначены сотрудникам из этого департамента переназначить на какого-то одного сотрудника.

```
SET TERM ^ ;
create or alter procedure NEW_PROCEDURE (
    WORKER integer,
    SEAT integer)
as
declare variable WORKERS_IDS integer;
begin
  for select WORK_ID
      from WORKERS
      where S_ID = :SEAT
      into  :WORKERS_IDS
      do
      update meetings set W_MAN = :worker where W_MAN= :workers_ids;
end^

SET TERM ; ^
GRANT SELECT ON WORKERS TO PROCEDURE NEW_PROCEDURE;
GRANT SELECT,UPDATE ON MEETINGS TO PROCEDURE NEW_PROCEDURE;
GRANT EXECUTE ON PROCEDURE NEW_PROCEDURE TO SYSDBA;
```

Соберемся расформировать например 4 департамент а встречи назначим на 10 работника. Проверим на 3х сотрудниках из 4 департамента, это сотрудники 8, 18, 19

Record: 21 — 21 records

MEET_ID	DATUM	CL_MAN	W_MAN	ADDIT
120	<null>	63	19	<null>
561	24.10.1981 21:35	37	8	<null>
869	06.03.1984 08:58	7	8	<null>
227	27.11.1984 10:16	23	19	<null>
292	08.12.1984 12:07	79	8	<null>
372	22.04.1986 10:42	55	8	<null>
405	22.07.1988 02:21	32	19	iddKevanuOwnslWaMHBVzLuvcAhtKrbSueCyADLUrihKueKgLmdYwGbWNuHfwAzAdYaGKVYTcJngttSGvdESKFPyaxDwXdXabYGLTEFWoScHmVoIZSA
686	08.09.1991 23:04	62	19	nLmuNCdqepWCLDtPJMhqvjSZHQunqEgEptZdWcuasrDdUXTDJaUfkMeRCzfTdmxtFwxBySBRjfUdcNHLNBQXoVPQuUJyggSLDpmbhOTrTNuVBcHpEyr
762	28.12.1992 04:54	15	8	qcZVqhdOIXXhMjtBelqxYLjDluBtRFnhIndYuhNdeKkRUEtYPJvOvFsGZTaRznZhBcgVjUYoEjgyOzxEjTtEyLkyFkpeypmartWBnVPReVpTvNyEGRifCtDiETr
424	20.03.1995 09:54	83	19	<null>

Получили 21 запись.

Запустим процедуру, в результате ее выполнения в встречах с идентификаторами например 561, 869 , 227 проверим изменение w_man.

```
execute procedure new_procedure(10, 4)
```

```
select * from meetings where w_man in (8, 18, 19) order by datum
```

MEET_ID	DATUM	CL_MAN	W_MAN	ADDIT
<null>	<null>	<null>	<null>	<null>

Проверим встречи с номерами.

MEET_ID	DATUM	CL_MAN	W_MAN	ADDIT
227	27.11.1984 10:16	23	10	<null>
561	24.10.1981 21:35	37	10	<null>
869	06.03.1984 08:58	7	10	<null>

Как видно теперь ответственный за эту встречу назначен человек который был указан в параметрах процедуры.

Выводы:

В результате выполнения заданий по разработке хранимых процедур можно сказать следующее: хранимые процедуры удобны для частого использования, достаточно один раз создать процедуру, что помогает экономить время. Хранимые процедуры и язык psql предоставляет довольно гибкие средства для реализации сложных процедур.

Необходимо отметить что в хранимых процедурах возможно не только изменение данных в БД, но так же проводить различные вычисления.

Дорогов Алексей, 4081/11, 2011г.