Санкт-Петербургский государственный политехнический университет

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №6**

**Дисциплина**: Базы данных

**SQL-программирование: Триггеры, вызовы процедур**

Выполнил студент гр. 43501/3 Муравьев Ф.Э.

Преподаватель: Мяснов А. В.

Санкт-Петербург

2015

**Цели работы**

Познакомить студентов с возможностями реализации более сложной обработки данных на стороне сервера с помощью хранимых процедур и триггеров.

## Программа работы

1. Создать два триггера: один триггер для автоматического заполнения ключевого поля, второй триггер для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице
2. Создать триггер в соответствии с **индивидуальным заданием**, полученным у преподавателя
3. Создать триггер в соответствии с **индивидуальным заданием**, вызывающий хранимую процедуру
4. Выложить скрипт с созданными сущностями в svn
5. Продемонстрировать результаты преподавателю

**Индивидуальное задание:**

Реализовать следующие триггеры:

1. При добавлении контракта подставлять свободную ячейку из наиболее заполненного хранилища.
2. При изъятии всех ценностей из ячейки завершать контракт.

**Триггер**— это [хранимая процедура](https://ru.wikipedia.org/wiki/%D0%A5%D1%80%D0%B0%D0%BD%D0%B8%D0%BC%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0) особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определенном столбце заданной таблицы [реляционной базы данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Триггеры применяются для обеспечения целостности данных и реализации сложной[бизнес-логики](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0). Триггер запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в [транзакции](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D1%8F), в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

Момент запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанного с ним события; например, до добавления записи) или AFTER (после события). В случае, если триггер вызывается до события, он может внести изменения в модифицируемую событием запись (конечно, при условии, что событие — не удаление записи). Некоторые СУБД накладывают ограничения на операторы, которые могут быть использованы в триггере (например, может быть запрещено вносить изменения в таблицу, на которой «висит» триггер, и т. п.).

Кроме того, триггеры могут быть привязаны не к таблице, а к [представлению](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) (VIEW). В этом случае с их помощью реализуется механизм «обновляемого представления». В этом случае ключевые слова BEFORE и AFTER влияют лишь на последовательность вызова триггеров, так как собственно событие (удаление, вставка или обновление) не происходит.

**Выполнение работы:**

1. Создать два триггера: один триггер для автоматического заполнения ключевого поля, второй триггер для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице

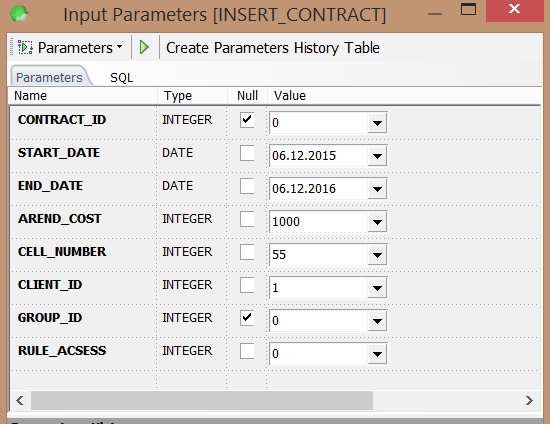
* Создан триггер для автоматического заполнения поля contract\_ID таблицы(Contracts) (имитация автоинкремента).

**Листинг 1. Триггер автоинкремента**

|  |
| --- |
| create generator create\_id;  set generator create\_id to 8;  create trigger auto\_create\_id for contracts before insert  as  begin  new.contract\_ID=gen\_id(create\_id, 1);  end |

Проверим работу триггера:

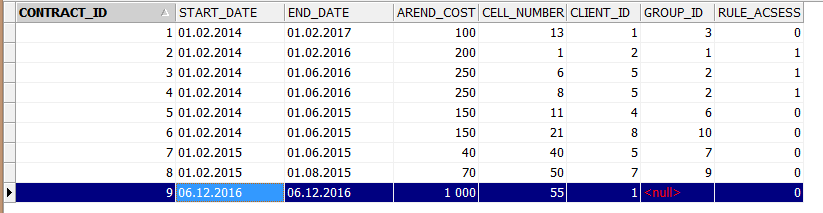
Воспользуемся процедурой insert\_contracts:



Как видно поле contract\_ID оставлено со значением NULL.

Запустим процедуру на выполнение.

Полученные результаты:



Поле contract\_id получило значение 9. Триггер работает.

* Создан триггер, который проверяет данные на целостность: при попытке удаления или изменения записи в таблице предметов, на которую присутствуют внешние ссылки, он выдает ошибку:

**Листинг 2. Триггер контроля целостности**

|  |
| --- |
| create exception error\_items 'Error. Cant delete or update in table items';  create trigger control\_items for items before delete or update  as  begin  if(old.item\_id in (select item\_id from history\_cell)) then  exception error\_items;  end |

Проверка работы триггера:

Таблица items:

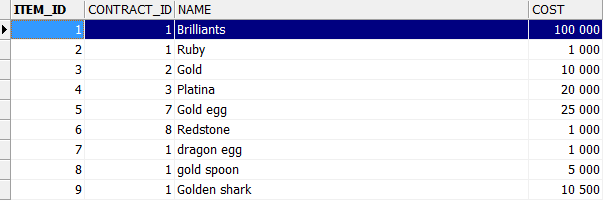
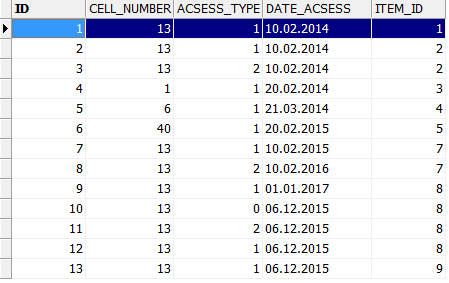


Таблица history\_cell:



Попробуем удалить запись из таблицы items с item\_id=1:

|  |
| --- |
| delete from items where item\_id=1; |

Результат:



Триггер работает.

1. Создать триггер в соответствии с **индивидуальным заданием**, полученным у преподавателя

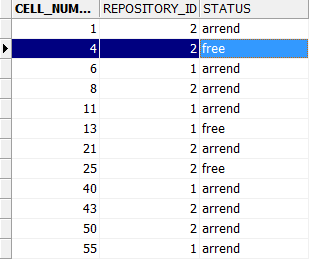
* При добавлении контракта подставлять свободную ячейку из наиболее заполненного хранилища.

**Листинг 3. Триггер, подставляющий свободную ячейку из наиболее заполненного хранилища.**

|  |
| --- |
| create exception dosent\_have\_free\_cells 'Error. Cant find free cell in banks';  create trigger insert\_cell\_n for contracts before insert  as  declare variable rep\_id integer;  declare variable num integer;  begin  for select repository.repository\_id, count(cell\_number) as kol from repository, cells  where(repository.repository\_id=cells.repository\_id)and(cells.status='arrend')  group by repository\_id order by kol desc into :rep\_id, :num do  begin  num=-1;  select min(cell\_number) from cells  where (repository\_id=:rep\_id)and(status='free') into :num;  if (num<>-1) then  begin  new.cell\_number=num;  update cells set status='arrend' where cell\_number=:num;  exit;  end  end  exception dosent\_have\_free\_cells;  end |

Проверка работы триггера:

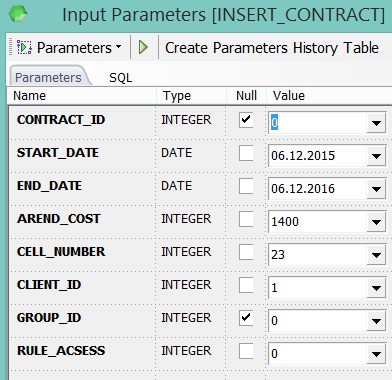
Таблица cells:



|  |  |
| --- | --- |
| Репозиторий | Количество занятых ячеек |
| 1 | 4 |
| 2 | 5 |

Триггер должен выбрать ячейку из 2 репозитория, проверим это.

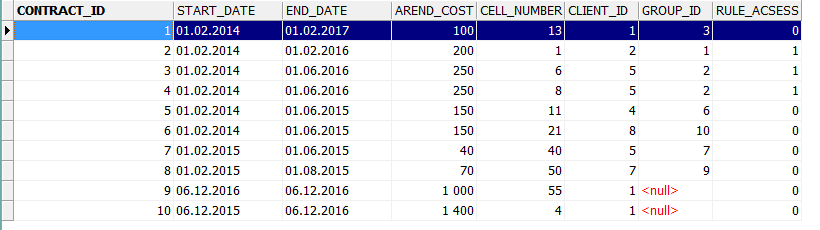
Запускаем процедуру insert\_contract:



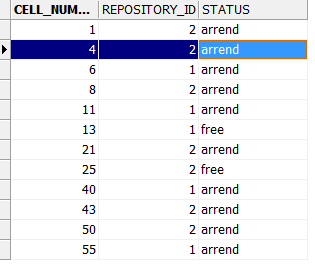
Значение cell\_number=23

Выполняем процедуру.

Проверим таблицы contracts и cells:



Видим что cell\_number=4 в контракте с id=10



Ячейка №4 теперь арендована

Триггер работает.

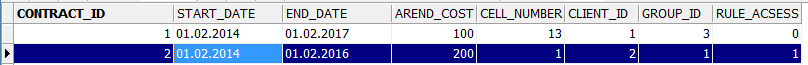
* При изъятии всех ценностей из ячейки завершать контракт.

**Листинг 4. Триггер, завершающий контракт при изъятии всех ценностей из ячейки.**

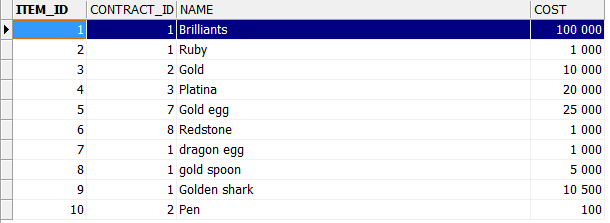
|  |
| --- |
| create trigger fin\_contract for history\_cell after insert  as  declare variable contract\_i integer;  declare variable item\_i integer;  declare variable flag integer;  begin  if(new.acsess\_type=2) then  begin  select contract\_id from contracts  where cell\_number=new.cell\_number  into :contract\_i;  flag=2;  for select item\_id from items  where contract\_id=:contract\_i into item\_i do  begin  select first 1 acsess\_type from history\_cell  where (item\_id=:item\_i)and(cell\_number=new.cell\_number)  order by date\_acsess desc into :flag;  if(flag<>2) then  begin  exit;  end  end  update contracts set end\_date=current\_date where contract\_ID=:contract\_i;  update cells set status='free' where cell\_number=new.cell\_number;  end  end |

Проверка работы триггера:

Контракт(таблица contracts):

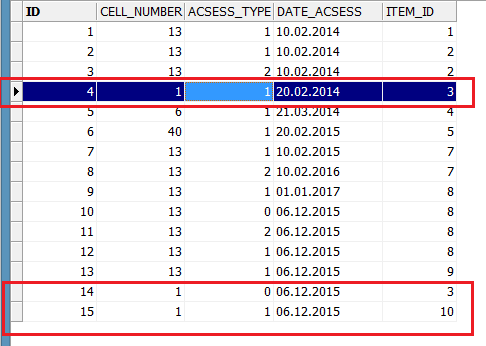


Хранимые предметы:



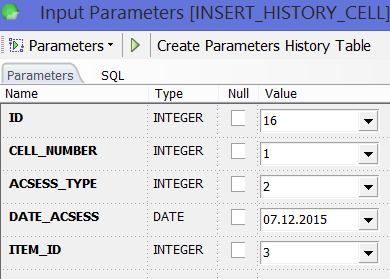
Предметы с id=3 и id=10

Оба предмета сейчас находятся в ячейке:



Попробуем извлечь один из предметов:

Запускаем процедуру insert\_history\_cell



Результаты:

Таблица history\_cell: 

Таблица contracts:

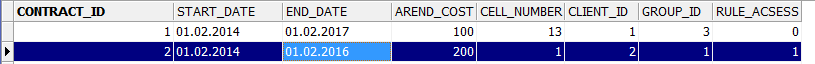
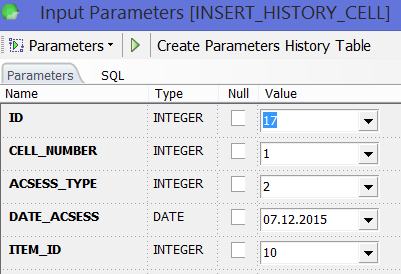


Таблица cells:



Триггер не сработал(т.к. в ячейке еще остались предметы).

Извлекаем последний предмет:



Результаты:

Таблица history\_cell:

Таблица contracts:

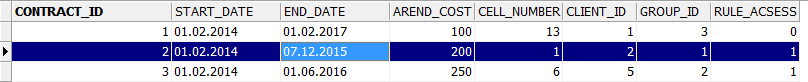
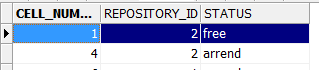


Таблица cells:



Как видим триггер сработал. Контракт на ячейку завершился при изъятии всех предметов из ячейки. Status ячейки в таблице Cells изменился на ‘free’.

**Выводы:**

В данной работе были изучены триггеры и генераторы.

Триггер – это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных.

Триггеры предназначены для выполнения простых операций по поддержанию целостности базы. Например, они могут генерировать индексы для таблицы, что является самым безопасным способом получения индекса. Также они позволяют каскадно удалять или изменять поля, зависящие от других таблиц.

Генератор – это специальный объект базы данных, который генерирует уникальные последовательные числа. Одним из применений генераторов является их использование в триггерах автоинкрементирования ключей. В таких триггерах необходимо использовать генераторы, так как они обеспечивают уникальность генерируемых значений даже при параллельной обработке нескольких запросов.

Можно придумать различные способы полезного применения механизма триггеров, но принято считать, что основными областями использования этого механизма являются следующие.

* *Журнализация и аудит*. С помощью триггеров можно отслеживать изменения таблиц, для которых требуется поддержка повышенного уровня безопасности. Данные об изменении таблиц могут сохраняться в других таблицах и включать, например, идентификатор пользователя, от имени которого выполнялась операция обновления; временную метку операции обновления; сами обновляемые данные и т. д.
* *Согласование и очистка данных*. С любым простым оператором SQL, обновляющим некоторую таблицу, можно связать триггеры, производящие соответствующие обновления других таблиц. Например, с операцией вставки новой строки в таблицу EMP (прием на работу нового служащего) можно было связать триггер, модифицирующий значения столбцов DEPT\_EMP\_NO и DEPT\_TOTAL\_SAL[162)](http://citforum.ru/database/advanced_intro/75.shtml" \l "footnote162)строки таблицы DEPT со значением столбца DEPT\_NO, которое соответствует номеру отдела нового служащего.
* *Операции, не связанные с изменением базы данных*. В триггерах могут выполняться не только операции обновления базы данных. Стандарт SQL позволяет определять хранимые процедуры (которые могут вызываться из триггеров), посылающие электронную почту, печатающие документы и т. д.

При условии правильного использования триггеры могут стать очень мощным механизмом. Основное их преимущество заключается в том, что стандартные функции сохраняются внутри базы данных и согласованно активизируются при каждом ее обновлении. Это может существенно упростить приложения. Тем не менее следует упомянуть и о присущих триггеру недостатках:

* сложность: при перемещении некоторых функций в базу данных усложняются задачи ее проектирования, реализации и администрирования;
* скрытая функциональность: перенос части функций в базу данных и сохранение их в виде одного или нескольких триггеров иногда приводит к сокрытию от пользователя некоторых функциональных возможностей. Хотя это в определенной степени упрощает его работу, но, к сожалению, может стать причиной незапланированных, потенциально нежелательных и вредных побочных эффектов, поскольку в этом случае пользователь не в состоянии контролировать все процессы, происходящие в базе данных;
* влияние на производительность: перед выполнением каждой команды по изменению состояния базы данных СУБД должна проверить триггерное условие с целью выяснения необходимости запуска триггера для этой команды. Выполнение подобных вычислений сказывается на общей производительности СУБД, а в моменты пиковой нагрузки ее снижение может стать особенно заметным. Очевидно, что при возрастании количества триггеров увеличиваются и накладные расходы, связанные с такими операциями.

Неправильно написанные триггеры могут привести к серьезным проблемам, таким, например, как появление "мертвых" блокировок. Триггеры способны длительное время блокировать множество ресурсов, поэтому следует обратить особое внимание на сведение к минимуму конфликтов доступа.

**Индекс** — объект [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путем последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск.