Триггеры

Триггеры в существующей базе данных (EMPLOYEE).

Задание 1.В базе данных EMPLOYEE есть 4 триггера, решающих различные задачи. SET_CUST_NO и SET_EMP_NO используются совместно с генераторами для автоматического формирования суррогатных первичных ключей. SAVE_SALARY_CHANGE автоматически заносит информацию об изменении оклада сотрудника. POST_NEW_ORDER генерирует событие при добавлении нового счета.Ознакомьтесь с этими триггерами.

Задание 2. Таблица EMPLOYEE, кроме двух явных триггеров, которые можно увидеть на вкладке Triggers, имеет еще два триггера, связанных с ограничением на значение поля salary. Чтобы увидеть их текст необходимо перейти на вкладку Dependencies.

Создание триггеров

Замечание. Чтобы контролировать правильность срабатывания триггеров рекомендуется включать в них операцию записи текстовой строки в таблицу LOG_FILE.Чтобы результаты записывались в таблицу LOG_FILE всегда, соответствующую команду нужно выполнять в автономной транзакции: IN AUTONOMOUS TRANSACTION DO. Все задания выполняются в своих рабочих базах данных.

Задание 1. Пусть одно из деловых ограничений требует, чтобы название фирмы NAME_AG, заносимое в таблицу-справочник AGENT, всегда записывалось заглавными буквами. Чтобы не обременять пользователя лишними инструкциями по вводу, а прикладную программу дополнительным кодом, определим триггер

```
create trigger BI_AGENT for AGENT
    before insert position 0
as
begin
    NEW.NAME_AG = UPPER(NEW.NAME_AG);
end^
```

Создайте триггер для таблицы AGENT в визуальном редакторе, параметры триггера определяются в соответствующих окнах, текст вводится, как и текст процедуры.

ВНИМАНИЕ!!! Для правильного срабатывания названий, задаваемых в кириллице должны быть правильно установлены кодировки. У базы данных на сервере UTF-8, в сеансе соединения из IBExpert – win1251.

Протестируйте работу триггера, введя несколько записей в таблицу.

Обратите внимание, что триггер будет работать только при выполнении той операции, которая его вызвала. Для полной реализации функциональности ограничений на столбцы необходимо создать два триггера BEFORE INSERT и BEFORE UPDATE или объединить оба действия в одном триггере. Создайте второй триггер для реализации того же делового ограничения.

Задание 2. Деловые правила могут запрещать выполнять операции, нарушающие ограничения целостности. Например, при редактировании поля PAY (оклад) в таблице SOTR (сотрудники) можно ввести ограничение, что оклад может быть только повышен. Создайте в своей базе соответствующую простую таблицу или дополните нужными полями таблицу AGENT. Создайте необходимые исключение и триггер, аналогичные описанным в примере.

```
create exception ERROR_PAY 'Запрещено!!!';

create trigger AU_SOTR for SOTR
    after update position 0
as
begin
    if (NEW.PAY<OLD.PAY) then
        exceptionERROR_PAY;
end^
```

Если операция изменения оклада не будет его повышать, триггер завершится генерацией исключения. При этом вызвавшая его команда UPDATE будет откачена оператором ROLLBACK. Если меняются другие поля – оплата не будет изменяться.

Проверьте, как работает ограничение.

Задание 3. Определение значения по умолчанию для поля.

Поскольку большинство поставщиков, с которыми осуществляется работа, расположены в Ростове-на-Дону, определим его для подстановки значения по умолчанию, если при добавлении или редактировании таблицы AGENT в поле TOWN указывается значение NULL. Создайте для реализации этого делового ограничения триггер.

```
create trigger BIU_AGENT for AGENT
    before insert or update position 1
as
begin
    if NEW.TOWN is NULL then
        NEW.TOWN = 'Ростов-на-Дону';
end^
```

Проверьте, как работает этот триггер.

ВНИМАНИЕ !!! На самом дела для этого можно использовать значение default при создании/изменении схемы таблицы.

Задание 4. Проверка значения в таблице на основе значений из другой таблицы.

Создайте в своей базе данных две таблицы. Таблица «Счет за услуги телефонной связи» ORDERS будет содержать данные:

номер счета ACC,
дата,
имя получателя,
номер телефона,
сумма к оплате SUMMA.

Таблица «Услуги» SERVICES будет содержать:

номер счета ID_ACC,
наименование услуги,
стоимость COST.

Каждый счет может содержать сведения более чем об одной услуге. Опишите таблицы, не вводя никаких ограничений (ключей). Создайте триггер для правильного заполнения суммы по счету.

```
create trigger BIU_ORDERS for ORDERS
    before insert or update position 0
as
declare variable S NUMERIC(17,2);
begin
    select sum(COST)
        from SERVICES
        where ID_ACC = NEW.ACC into :S;
    NEW.SUMMA = S;
end^
```

Работа с таблицами ORDERS и SERVICES должна осуществляться в таком порядке: сначала заводятся данные в таблицу SERVICES – несколько строк, в которых указан одинаковый номер счета ID_ACC, затем, счет с таким номером вводится в таблицу ORDERS и поле SUMMA в нем не заполняется (или вводится произвольное значение). Протестируйте работу триггера.

Задание 5. Создайте триггер, который бы при любых изменениях в таблице SERVICES (добавление новой строки, изменение суммы в существующей) находит нужную запись в таблице ORDERS и меняет в ней сумму. Считать в этом случае, что порядок работы с таблицами другой – сначала заводится счет в таблице ORDERS , а потом в него добавляются строки в таблице SERVICES.

ВНИМАНИЕ !!! Перед выполнением этого задания отключите триггер, созданный в предыдущем задании.

Задание 6. Существующие в Firebird генераторы уникальных значений могут быть использованы для реализации автоинкрементных первичных ключей.

Создайте в тестовой базе данных таблицу MYTABLE, у которой поле первичного ключа PK_MYTABLE. Создайте генератор и триггер для автоинкрементного заполнения поля первичного ключа.

```
create generator GEN_PK;

create trigger BI_MYTABLE for MYTABLE
    before insert
as
begin
        if (NEW.PK_MYTABLE is NULL) then
            NEW.PK_MYTABLE=GEN_ID(GEN_PK,1);
end^
```

Приведенное выше решение не снимает проблему появления дублирующих значений первичного ключа, если ввод ключа осуществляется и с помощью генератора, и ручным вводом. Протестируйте и убедитесь в этом. Измените триггер так, чтобы он использовал генератор корректно в любой ситуации.

Функция GEN_ID может использовать любое значение для автоинкремента ключа. Проверьте это!

Такое поведение генератора не соответствует стандарту, поэтому, начиная с версии 2.0, имеется возможность использования последовательностей – синтаксически это эквивалентные описания (посмотрите DDL вкладку для генераторов). Для получения следующего значения в последовательности используется оператор NEXT VALUE FOR.

Задание 7. Реализуем ссылочную целостность для следующих двух таблиц.

Справочника (таблицы соответствия) LOOKUP

```
create table LOOKUP
(L_ID integer NOT NULL UNIQUE,
. . .
);
```

И таблицы, использующей данные из справочника через ссылку на ID соответствующей строки в таблице LOOKUP.

```
create  table REQUESTOR
(ID integer NOT NULL PRIMARY KEY,
 LOOKUP_ID integer,
. . .
);
```

Таблицы можно создавать и через команды и через оконный интерфейс.

Чтобы реализовать действие NOACTION при удалении или редактировании строки в таблице LOOKUP, создадим исключение

```
create exception NOT_VALID 'Ошибка!!!';
COMMIT;
```

Триггер, который проверяет ограничение ссылочной целостности на стороне дочерней таблицы, в нашем случае таблицы REQUESTOR.

```
create trigger BA_REQ for REQUESTOR
active before INSERT OR UPDATE
as
begin
    if (NEW.LOOKUP_ID IS NOT NULL
        AND
        NOT EXISTS(
        select L_ID from LOOKUP
```

```
            where L_ID = NEW.LOOKUP_ID)
        then exception NOT_VALID;
    end^
```

Второй триггер проверяет возможность удалять и редактировать строки в таблице соответствия LOOKUP.

```
    create trigger BA_L for LOOKUP
    active before DELETE OR UPDATE
    as
    begin
        if (UPDATING AND (NEW.L_ID<>OLD.L_ID)
            OR DELETING)
        then
            if (EXISTS(
                select LOOKUP_ID
                from REQUESTOR
                where LOOKUP_ID=OLD.L_ID))
            then exception NOT_VALID;
    end^
```

Задание 8. Реализовать возможность каскадного изменения записей в дочерней таблице при изменении первичного ключа в родительской таблице. Воспользуемся в этом задании теми же таблицами, что и в предыдущем. Изменения вносим в триггер для таблицы LOOKUP.

```
    create trigger BU_L for LOOKUP
    active before UPDATE
    as
    begin
        if (NEW.L_ID<>OLD.L_ID)
        then
            if (EXISTS(
                select LOOKUP_ID
```

```
                from REQUESTOR
                where LOOKUP_ID=OLD.L_ID))
            then
                update REQUESTOR
                    set LOOKUP_ID=NEW.L_ID
                    where LOOKUP_ID=OLD.L_ID;
end^
```

Прежде, чем тестировать работу этого триггера, не забудьте сделать триггер, созданный в предыдущем задании неактивным.

Задание 9. Реализовать возможность каскадного удаления записей в дочерней таблице при удалении строки в родительской таблице. Воспользуемся в этом задании теми же таблицы, что и в предыдущих заданиях. Изменения вносим в триггер для таблицы LOOKUP.

```
create trigger BD_L for LOOKUP
active before DELETE
as
begin
    if (EXISTS(
            select LOOKUP_ID
            from REQUESTOR
            where LOOKUP_ID=OLD.L_ID))
    then
    delete from REQUESTOR
        where LOOKUP_ID=OLD.L_ID;
end^
```

Задание 10. При каждом добавлении записи в таблицу делать запись в таблицу-журнал. Сначала познакомимся с возможность вести журнал средствами IBExpert.

Для активизации журнала воспользуемся командой вызова утилиты логирования Logmanager. Появляющееся при этом окно Logmanager

содержит информацию обо всех настройках журнала. При первом вызове Logmanager может появиться окно, сообщающее, что для ведения журналов нужно создать вспомогательные таблицы. Ответьте на сообщение в этом окне согласием. Отмечаем, что нам необходим журнал по выполнению операции INSERT. Будет сгенерирован триггер аналогичный следующему

```
create trigger IBE$PROJECT_AI for AGENT
ACTIVE AFTER INSERT POSITION 32767
as
declare variable tid integer;
begin
tid = gen_id(ibe$log_tables_gen,1);
insert into ibe$log_tables (id, table_name,
            operation, date_time, user_name)
values (:tid, 'AGENT', 'I', 'NOW', user);
end^
```

Выполните теперь вставку одной строки в таблицу AGENT. Зафиксируйте транзакцию. Запись в журнале можно посмотреть, используя обыкновенный запрос к таблице ibe$log_tables. Или воспользоваться вкладкой Logging окна работы с таблицей AGENT в IBExpert.

Если в окне Logmanager дополнительно отметить поля таблицы, то в триггере для каждого поля появится приблизительно такой оператор (для поля NAME_AG)

```
if (new.name_ag is not null) then
insert into ibe$log_fields (log_tables_id,
            field_name, old_value, new_value)
values (:tid, 'NAME_AG', null,
                    new.name_ag);
```

Задание 11. По аналогии с предыдущим заданием, создайте свои собственные таблицы для журналов операций, напишите и проверьте соответствующие триггеры.

Задание 12. Триггеры можно использовать для реализации сложных правил проверок защиты данных от несанкционированного доступа. Например, можно запретить вносить изменения (INSERT, UPDATE, DELETE) в таблицу OPERATION в праздничные дни. Для упрощения проверки, предположим, что нас интересует только один праздничный день – 1 января.

Создайте исключение

```
CREATE EXCEPTION HOLIDAY
    'May not change operation table during a holiday';
```

Создайте триггер

```
CREATE TRIGGER OPERATION_ALL FOR operation
ACTIVE BEFORE INSERT and UPDATE AND DELETE POSITION 0
AS
declare variable dt integer;
declare variable mn integer;
begin
dt = extract(day from current_date);
mn = extract(month from current_date);
if  (dt=1 and mn=1) then
exception HOLIDAY;
end^
```

Для проверки работы этого триггера можно изменить дату выходного дня на любую.

Задание 13. Создайте вспомогательную таблицу «Справочник дней, когда запрещено работать». Внесите соответствующие изменения в триггер из предыдущего задания.

Задание 14. Добавьте в триггер `OPERATION_ALL` возможность журнализации попыток работать с таблицей `OPERATION` в четверг (использовать для определения дня недели функцию EXTRACT). Операции записи выполнять в автономных транзакциях.

Задание 15. Для проверки очередности срабатывания триггеров, создайте для одной из таблиц и каждой из фаз BEFORE и AFTER несколько триггеров. Предусмотрите возможность нормального завершения и выбрасывания исключения в каждом из триггеров. Каждый из триггеров должен в начале своей работы делать две записи в таблицу LOG_FILE, одна из которых должна выполняться в автономной транзакции.