

## **ЛАБОРАТОРНАЯ РАБОТ 3. ВЫПОЛНЕНИЕ ЗАПРОСОВ К БАЗЕ ДАННЫХ**

**Цель работы:** получить навыки формирования SQL запросов на добавление, изменение, извлечение и удаление данных на примере созданной согласно варианту базы данных. Изучить основы создания простейших триггеров.

### **3.1 Задание**

- 1) Заполнить БД, созданную в Лабораторной работе №2 используя запросы
- 2) Создать запросы на извлечение данных (*Требования:* запросы должны отражать потребности реальных пользователей, например, найти самую дорогую книгу, самую покупаемую вещь, определить наиболее частых клиентов и т.д.)
- 3) Создать подзапросы и вложенные запросы (такие же требования как и в П2)
- 4) Создать триггеры с помощью запросов
- 5) Отобразить созданные запросы в отчете с комментариями

### **3.2 Ход работы**

1. Создать базу данных используя мастер создания БД в SQL Server Management Studio согласно схеме, представленной на рисунке 3.1.
2. Написать SQL запросы на добавление данных в таблицы. Данные представлены на рисунках 3.2 – 3.5.
3. Изучить 28 примеров простых и вложенных запросов на извлечение данных из раздела 3.4 данного методического пособия. Протоколирую результат выполнения запросов в отчет о проделанной работе.
4. На свое усмотрения создать три триггера из примеров раздела 3.5 Протестировать их и результаты теста привести в отчете.
5. Для своей схемы БД (созданной во второй лабораторной работы) написать 25 запросов различной степени сложности (аналогично проделанным выше примерам). Результаты выполнения представить в отчете.
6. Составить отчет о проделанной работе. Структура отчета:
  - титульный лист;
  - задание;

- описание хода выполнения работы (написать запросы и прикрепить скрины результатов работы по каждому запросу);
- заключение;

### 3.3 Подготовка к выполнению лабораторной работы

Перед тем как приступить к выполнению лабораторной работы номер 3 Вам необходимо создать базу данных используя мастер создания БД в MSSQL Server 20XX согласно ниже представленной схеме.

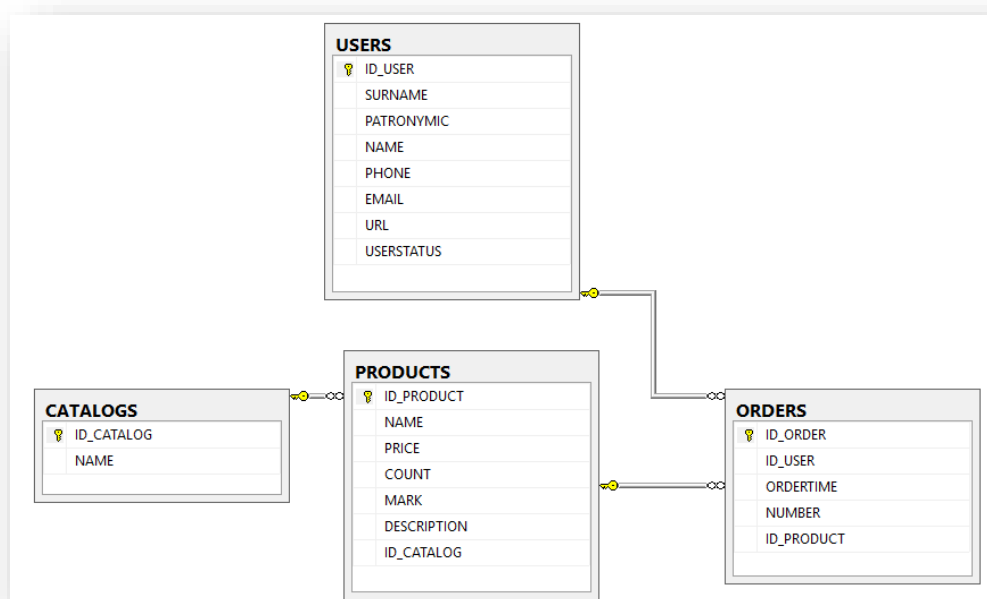


Рис. 3.1 – Схема базы данных

Следующим шагом будет заполнение БД данными. Ниже представлены данные, которыми необходимо наполнить базу данных используя SQL запросы.

	ID_CATALOG	NAME
1	1	Процессоры
2	2	Материнские платы
3	3	Видеоадаптеры
4	4	Жесткие диски
5	5	Оперативная память

Рис. 3.2 – Данные в таблице CATALOGS

	ID_PRODUCT	NAME	PRICE	COUNT	MARK	DESCRIPTION	ID_CATALOG
1	1	Celeron 1.8	1595.00	10	3,6	Процессор Celeron® 1.8GHz, 128kb, 478-PGA, 400Mhz, OEM 0.18	1
2	2	Celeron 2.0GHz	1969.00	2	3,7	Процессор Celeron® 2.0GHz, 128KB, 478-PGA, 400Mhz, OEM	1
3	3	Celeron 2.4GHz	2109.00	4	3,9	Процессор Celeron® 2.4GHz, 128kb, 478-PGA, 400Mhz, OEM	1
4	4	Celeron D 320 2.4GHz	1962.00	1	4,1	Процессор Celeron® D 320 2.4GHz, 256kb, 478-PGA, 533Mhz, OEM	1
5	5	Celeron D 325 2.53GHz	2747.00	6	4,1	Процессор Celeron® D 325 2.53GHz, 256kb, 478-PGA, 533Mhz, OEM	1
6	6	Celeron D 315 2.26GHz	1880.00	6	4,1	Процессор Celeron® D 315 2.26GHz, 256kb, 478-PGA, 533Mhz, OEM	1
7	7	Intel Pentium 4 3.2GHz	7259.00	5	4,5	Процессор Intel® Pentium®4 3.2GHz, 1Mb, 478-PGA, 800Mhz, Hyper-Thre...	1
8	8	Intel Pentium 4 3.0GHz	6147.00	1	4,6	Процессор Intel® Pentium®4 3.0GHz, 512Kb, 478-PGA, 800Mhz, Hyper-Thr...	1
9	9	Intel Pentium 4 3.0GHz	5673.00	12	4,5	Процессор Intel® Pentium®4 3.0GHz, 1Mb, 478-PGA, 800Mhz, Hyper-Trea...	1
10	10	Gigabyte GA-8I848P-RS	1896.00	4	3,9	Материнская плата SOCKET-478 Gigabyte GA-8I848P-RS i848, (800Mhz), ...	2
11	11	Gigabyte GA-8IG1000	2420.00	2	3,8	Материнская плата SOCKET-478 Gigabyte GA-8IG1000 i865g,FSB800/53...	2
12	12	Gigabyte GA-8IPE1000G	2289.00	6	3,7	Материнская плата Socket-478 Gigabyte GA-8IPE1000G i865PE(800/533...	2
13	13	Asustek P4C800-E Delux	5395.00	4	4,1	Материнская плата Socket-478 Asustek P4C800-E Delux i875P,FSB800/5...	2
14	14	Asustek P4P800-VM\Л i865G	2518.00	6	4	Материнская плата Socket-478 Asustek P4P800-VM\Л i865G FSB800/53...	2
15	15	Epoх EP-4PDA3I	2289.00	5	4	Материнская плата Socket-478 Epoх EP-4PDA3I i865PE(800Mhz), 2chDD...	2
16	16	ASUSTEK A9600XT/TD	5156.00	2	4,7	Видеоадаптер ASUSTEK A9600XT/TD 128Mb DDR SDRAM, 2x400MHz ...	3
17	17	ASUSTEK V9520X	1602.00	6	4	Видеоадаптер ASUSTEK V9520X 128Mb DDR SDRAM, 400MHz DAC, AG...	3
18	18	SAPPHIRE 256MB RADEON 9550	2730.00	3	3,8	ВИДЕОКАРТА SAPPHIRE 256MB RADEON 9550, TV-out, DVI, OEM	3
19	19	GIGABYTE AGP GV-N59X128D	5886.00	6	3,6	ВИДЕОКАРТА GIGABYTE AGP GV-N59X128D FX5900XT OEM	3
20	20	Maxtor 6Y120P0	2456.00	6	4,5	Винчестер 120 GB Maxtor 6Y120P0, UDMA-133, 7200rpm, 8MB	4
21	21	Maxtor 6B200P0	3589.00	4	4	Винчестер 200 GB Maxtor 6B200P0, UDMA-133, 7200rpm, 8Mb	4
22	22	Samsung SP0812C	2093.00	5	4	Винчестер 80 GB Samsung SP0812C, SATA, 7200rpm SpinPoint P80 Serial...	4
23	23	Seagate Barracuda ST3160023A	3139.00	3	4,1	Винчестер 160 GB Seagate Barracuda ST3160023A, UDMA-100, 7200rpm,...	4
24	24	Seagate ST3120026A	2468.00	8	4,2	Винчестер 120 GB Seagate ST3120026A, UDMA-100, 7200rpm, 8MB	4
25	25	DDR-400 256MB Kingston	1085.00	20	4,8	Оперативная память DDR-400 256MB Kingston	5
26	26	DDR-400 256MB Hynix Original	1179.00	15	4,6	Оперативная память DDR-400 256MB Hynix Original	5
27	27	DDR-400 256MB PQI	899.00	10	4,2	Оперативная память DDR-400 256MB PQI	5
28	28	DDR-400 512MB Kingston	1932.00	20	4,8	Оперативная память DDR-400 512MB Kingston	5
29	29	DDR-400 512MB PQI	1690.00	12	4,2	Оперативная память DDR-400 512MB PQI	5
30	30	DDR-400 512MB Hynix	1717.00	8	4,5	Оперативная память DDR-400 512MB Hynix	5

Рис. 3.3 – Данные в таблице PRODUCTS

ID_USER	SURNAME	PATRONYMIC	NAME	PHONE	EMAIL	URL	USERSTATUS
1	Иванов	Валерьевич	Александр	58-98-78	ivanov@email.ru	NULL	active
2	Прокопчук	Иванович	Сергей	9057777777	pro@email.ru	NULL	passive
3	Семенов	Вячеславович	Игорь	9056666100	simdyanov@softtime.ru	http://www.softtime.ru	active
4	Петров	Валерьевич	Максим	NULL	kuznetsov@softtime.ru	http://www.softtime.ru	active
5	Лосев	Юрьевич	Анатолий	NULL	losev@email.ru	NULL	lock
6	Корнеев	Александрович	Александр	89-78-36	komeev@domen.ru	NULL	gold

Рис. 3.4 – Данные в таблице USERS

ID_ORDER	ID_USER	ORDERTIME	NUMBER	ID_PRODUCT
4	3	2005-04-01 10:39:38.000	1	8
5	6	2005-10-02 09:40:29.000	2	10
6	1	2005-02-18 13:41:05.000	4	20
7	3	2005-10-03 18:20:00.000	1	20
8	3	2005-03-17 19:15:36.000	1	20

Рис. 3.5 – Данные в таблице ORDERS

### 3.4 Примеры запросов на извлечение данных

Ниже представлен перечень (24) простых запросов к БД (схема которой описана Выше. Данные примеры покрывают большой спектр конструкций языка SQL, начиная от простых запросов, кончая запросов с использованием функций и сортировок. Для выполнения лабораторной работы, Вам необходимо проделать все 24 запроса и привести результат выполнения в виде скриншота. Все конструкции языка SQL подробно описаны в лекционном материале.

#### Пример 1. Вывод данных таблицы CATALOGS

```
SELECT ID_CATALOG, NAME FROM CATALOGS -- Здесь выводятся два поля  
ID_CATALOG и NAME из таблицы CATALOGS  
SELECT * FROM CATALOGS -- Здесь выводятся все поля из таблицы CATALOGS
```

#### Пример 2. Вывод данных таблицы CATALOGS с присвоением псевдонима

```
SELECT  
    ID_CATALOG AS 'Идентификатор категории', -- команда AS изменяет  
имя столбца в результирующей таблице на имя, указанное после этой  
команды в кавычках (ID_CATALOG изменяется на 'Идентификатор  
категории')  
    NAME AS 'Имя категории'  
FROM CATALOGS
```

**Пример 3.** Добавление данных с помощью SELECT в результирующую таблицу

```
SELECT NAME, ID_CATALOG, 5, 'COMMENTS' FROM CATALOGS
```

**Пример 4.** Извлечение из таблицы CATALOGS записи, чей первичный ключ ID\_CATALOG больше 2

```
SELECT * FROM CATALOGS
WHERE ID_CATALOG > 2 -- здесь WHERE это условие. Т.е. мы указываем,
что нам нужно выбрать все данные из всех столбцов (*) таблицы CATALOGS
и вывести те строки, где значение поля ID_CATALOG будет больше 2
```

**Пример 5.** Составное условие: извлечение из таблицы CATALOGS записи, чей первичный ключ ID\_CATALOG больше 2, но меньше или равен 4

```
SELECT * FROM CATALOGS
WHERE ID_CATALOG > 2 AND ID_CATALOG <= 4 -- AND позволяет создавать
составные условия, т.е. после WHERE мы можем указать несколько условий
для выборки, разделяя их AND
```

```
SELECT * FROM CATALOGS
WHERE ID_CATALOG BETWEEN 3 AND 4 -- Здесь записана упрощенная версия
запроса выше. BETWEEN буквально означает «между», т.е. весь запрос
можно воспринять так: Выбрать все столбцы из таблицы CATALOGS, и
вывести те строки, где значение поля ID_CATALOG находится между 3 и 4
```

**Пример 5.** Противоположная конструкция, которая выводит из таблицы CATALOGS записи, чей первичный ключ ID\_CATALOG меньше 3, но больше 4.

```
SELECT * FROM CATALOGS
WHERE ID_CATALOG NOT BETWEEN 3 AND 4 -- NOT перед BETWEEN означает,
что по условию требуется что бы выводились строки где ID_CATALOG вне
указанного далее диапазона т.е. то, что находится между 3 и 4 не
выводилось
```

**Пример 6.** Вывод записей, удовлетворяющих не диапазону, а списку

```
SELECT * FROM CATALOGS
WHERE ID_CATALOG IN (1, 2, 5) -- IN указывает, что необходимо в
результатирующую таблицу поместить только те строки, значения поля
ID_CATALOG которых равны значениям указанным в скобках после IN
```

**Пример 7.** Вывод записей, удовлетворяющих условию, заданному текстом:  
вывести все записи, содержащие слово процессор

```
SELECT * FROM CATALOGS
WHERE NAME = 'процессоры' -- выводятся все строки, значения поля NAME
в которых равно «процессоры».
```

**Пример 8.** Вывод записей, удовлетворяющих условию, заданному текстом:  
вывести все записи, не содержащие слово процессор

```
SELECT * FROM CATALOGS
WHERE NOT NAME = 'процессоры' -- то же, что и в предыдущем примере,
только выводится все, кроме «процессоры»
```

**Пример 9.** Вывод записей, удовлетворяющих условию, заданному частью текста

```
SELECT * FROM USERS
WHERE SURNAME LIKE 'И%' -- LIKE дает возможность указать какую-либо
часть значения. В данном случае мы получаем все строки, значения поля
SURNAME которых будет начинаться с буквы И
```

**Пример 10.** Работа с датой: извлечение из таблицы ORDERS записи,  
соответствующие сделкам, осуществленным за февраль 2005 г.

```
SELECT * FROM orders
WHERE ORDERTIME >= '2005-02-01' AND ORDERTIME < '2005-03-01';
```

### Пример 11. Сортировка по значению одного из столбцов

```
SELECT * FROM CATALOGS
ORDER BY ID_CATALOG -- сортируем все строки по полю ID_CATALOG (по
возрастанию)
```

```
SELECT * FROM CATALOGS
ORDER BY NAME -- сортируем все строки по полю NAME
```

**Пример 12.** Извлечение из таблицы PRODUCTS записи товаров, количество которых COUNT на складе от 4 до 8 с сортировкой по полю COUNT и полю MARK (для краткости выведем только столбцы COUNT и MARK)

```
SELECT COUNT, MARK FROM PRODUCTS
WHERE COUNT BETWEEN 4 AND 8 ORDER BY COUNT, MARK -- выбираем строки,
значение поля COUNT которого находятся между 4 и 8, затем сортируем
полученные строки сначала по COUNT затем по MARK
```

**Пример 13.** Изменение порядка сортировки (по умолчанию, сортировка производится в прямом порядке (ASC))

```
SELECT ORDERTIME FROM ORDERS
ORDER BY ORDERTIME DESC -- DESC означает сортировка по убыванию
```

**Пример 14.** Извлечение первых пяти записей с обратной сортировкой по полю COUNT

```
SELECT TOP 5 ID_PRODUCT, COUNT FROM PRODUCTS
ORDER BY COUNT DESC
```

### Пример 15. Подсчет количества проданных ТОВАРОВ

```
SELECT SUM (NUMBER) AS 'Всего продано' -- SUM означает сумму значений  
поля NUMBER  
FROM ORDERS
```

### Пример 16. Подсчет среднего количества товаров в одном заказе

```
SELECT AVG (NUMBER) AS 'Среднее количество' -- AVG означает среднее  
значение поля NUMBER  
FROM ORDERS
```

### Пример 17. Подсчет числа строк в таблице, значения столбца которых отличны от NULL

```
SELECT COUNT (ID_ORDER) -- COUNT выводит количество строк поля ID_ORDER  
значения которого не равны NULL  
FROM ORDERS
```

### Пример 18. Подсчет числа строк в таблице, значения столбца которых отличны от NULL с присвоением псевдонима

```
SELECT COUNT (ID_ORDER) AS TOTAL  
FROM ORDERS
```

### Пример 19. Извлечение максимального значения столбца ID\_CATALOG

```
SELECT MAX (ID_CATALOG) -- MAX позволяет извлечь максимальное значение  
поля из столбца ID_CATALOG  
FROM CATALOGS
```

```
SELECT TOP 1 * FROM CATALOGS -- TOP 1 означает вывод первой строки из  
полученной выборки. После TOP может стоять любое число, соответственно  
число строк будет другое  
ORDER BY ID_CATALOG
```



**Пример 20.** Извлечение минимального значения столбца ID\_CATALOG

```
SELECT MIN (ID_CATALOG)
FROM CATALOGS
SELECT TOP 1 * FROM CATALOGS
ORDER BY ID_CATALOG DESC
```

**Пример 21.** Вывод числа уникальных значений ID\_CATALOG (сравните результат с SELECT COUNT (ID\_CATALOG) FROM PRODUCTS)

```
SELECT COUNT(DISTINCT ID_CATALOG) -- DISTINCT означает вывод только
уникальных значений этого поля
FROM PRODUCTS
```

**Пример 22.** Вывод числа записей, соответствующих каждому из уникальных значений ID\_CATALOG

```
SELECT ID_CATALOG, COUNT(ID_CATALOG)
FROM PRODUCTS
GROUP BY ID_CATALOG ORDER BY ID_CATALOG
```

**Пример 23.** Вывод числа записей, соответствующих каждому из уникальных значений ID\_CATALOG больше двух

```
SELECT ID_CATALOG, COUNT(ID_CATALOG)
FROM PRODUCTS
WHERE ID_CATALOG > 2
GROUP BY ID_CATALOG -- группировка записи по какому-либо полю,
оставляя при этом только одну запись с каждым значением
ORDER BY ID_CATALOG
```

**Пример 24.** Выбрать категории товаров, для которых добавлено более пяти товаров (ограничение выборки по результатам функции)

```
SELECT ID_CATALOG, COUNT(ID_CATALOG) AS TOTAL
FROM PRODUCTS
GROUP BY ID_CATALOG
HAVING ID_CATALOG > 5
ORDER BY ID_CATALOG
```

*Примечание: HAVING аналогичен WHERE за исключением того, что строки отбираются не по значениям столбцов, а строятся из значений столбцов, указанных в GROUP BY, и значений агрегатных функций, вычисленных для каждой группы, образованной GROUP BY.*

### 3.5 Подзапросы и вложенные запросы

Вложенные запросы — это запросы, которые расположены внутри других запросов, таких как SELECT, INSERT, UPDATE или DELETE. К тому же, подзапросы MySQL могут быть расположены внутри других подзапросов.

#### 3.5.1 Подзапрос MySQL в условии WHERE

Вы можете использовать операторы сравнения =, >, < и т.д., чтобы сравнить одно значение возвращенное подзапросом с выражением в условии WHERE.

**Пример 25.** Добавим вложенный запрос для получения самого дорогого товара из таблицы PRODUCTS.

```
SELECT NAME,
PRICE,
COUNT
FROM PRODUCTS
WHERE PRICE = (
SELECT MAX(PRICE)
FROM PRODUCTS
);
```

*Пояснение: Вложенные запросы следует рассматривать снизу вверх, т.е. здесь мы сначала получаем строку с максимальным значением поля PRICE из таблицы PRODUCTS, затем делаем выборку, где выводим строки с полями NAME, PRICE, COUNT где во вложенной выборке мы нашли максимальный PRICE*

### 3.5.2 Вложенный запрос с операторами IN и NOT IN

Если подзапрос возвращает более одного значения, вы можете использовать операторы IN и NOT IN в условии WHERE.

**Пример 26.** Используя оператор NOT IN, найдем пользователей, которые не заказали ни одного товара.

```
SELECT NAME, SURNAME
FROM USERS
WHERE ID_USER NOT IN (
    SELECT DISTINCT ID_USER
    FROM ORDERS
);
```

### 3.5.3 Вложенный запрос с EXISTS И NOT EXISTS

Когда подзапрос используется с операторами EXISTS или NOT EXISTS, то такой подзапрос возвращает булево значение: TRUE или FALSE. В данном случае вложенный запрос действует как проверка на существование.

**Пример 27.** Найдем всех пользователей, которые купили товары с ценой больше 1000.

```
SELECT NAME, SURNAME
FROM USERS
WHERE EXISTS
    (SELECT NAME
     FROM PRODUCTS, ORDERS, USERS
     WHERE ID_PRODUCT = ORDERS.ID_PRODUCT
     AND ORDERS.ID_USER = USERS.ID_USER
     AND PRICE > 1000)
```

*Примечание:* Конструкция `WHERE <имя_столбца> = <имя_столбца>` для соединения таблиц является старым стилем и в настоящее время заменяется конструкцией с оператором `JOIN`, речь о котором пойдет в следующей лабораторной работе.

### 3.5.4 Подзапрос MySQL в условии FROM

Когда вы используете вложенный запрос в условии FROM, то результат возвращенный этим подзапросом будет таблицей, которая называется производной.

**Пример 28.** Найдем максимальное, минимальное и среднее количество элементов в заказе.

```
SELECT
    MAX(items),
    MIN(items),
    FLOOR(AVG(items))
FROM
    (
        SELECT
            NUMBER,
            COUNT(NUMBER) AS items
        FROM
            ORDERS
        GROUP BY
            NUMBER
    ) AS LINEITEMS;
```

## 3.6 Триггеры

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, а также для отката транзакций.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на

операции ввода/вывода. В том случае, когда тех же результатов (с гораздо меньшими непроизводительными затратами ресурсов) можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно.

Триггеры – особый инструмент SQL-сервера, используемый для поддержания целостности данных в базе данных. С помощью ограничений целостности, правил и значений по умолчанию не всегда можно добиться нужного уровня функциональности. Часто требуется реализовать сложные алгоритмы проверки данных, гарантирующие их достоверность и реальность. Кроме того, иногда необходимо отслеживать изменения значений таблицы, чтобы нужным образом изменить связанные данные. Триггеры можно рассматривать как своего рода фильтры, вступающие в действие после выполнения всех операций в соответствии с правилами, стандартными значениями и т.д.

Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

Создает триггер только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Триггер представляет собой весьма полезное и в то же время опасное средство. Так, при неправильной логике его работы можно легко уничтожить целую базу данных, поэтому триггеры необходимо очень тщательно отлаживать.

В отличие от обычной подпрограммы, триггер выполняется неявно в каждом случае возникновения триггерного события, к тому же он не имеет аргументов. Приведение его в действие иногда называют запуском триггера. С помощью триггеров достигаются следующие цели:

- проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно,

поддерживать с помощью ограничений целостности, установленных для таблицы;

- выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;
- накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;
- поддержка репликации.

Ниже приведен синтаксис создания триггера:

```
CREATE TRIGGER trigger_name trigger_time trigger_event  
ON tbl_name FOR EACH ROW trigger_stmt
```

**trigger\_name** — название триггера;

**trigger\_time** — время срабатывания триггера:

BEFORE — перед событием;

AFTER — после события.

**trigger\_event** — Событие:

**insert** — событие активируется операторами insert, data load, replace;

**update** — событие активируется оператором update

**delete** — событие активируется операторами delete, replace.

Операторы DROP TABLE и TRUNCATE не активируют выполнение триггера

**tbl\_name** — название таблицы;

**trigger\_stmt** - выражение, которое выполняется при активации триггера

Рассмотрим несколько простых примеров создания триггеров для существующей базе данных. В рамках лабораторной работы необходимо создать и протестировать любых три триггера (из рассмотренных примеров ниже, изучив их работу)

**Пример 1.** Триггер на добавление записи в таблицу USERS. Данный триггер в случае успешного добавления данных выводит в «Запись добавлена»

```
CREATE TRIGGER INSERT_INDICATION      --определение имени функции
ON USERS                             --для какой таблицы создается триггер
AFTER INSERT                          --когда выполнять триггер
--INSERT - при создании записи в таблице,
--DELETE - при удалении записи в таблице,
--UPDATE - при изменении записи в таблице,
--AFTER - после выполнения операции,
--INSTEAD OF - вместо выполнения операции
AS
BEGIN --тело триггера
    SET NOCOUNT ON;
    PRINT 'Запись добавлена'
END
GO
```

#### *Тестирование работы триггера*

```
INSERT INTO USERS VALUES
('Громова', 'Валерьевна', 'Анна', '55-66-89', NULL, NULL, 'active'),
('Кремнева', 'Александровна', 'Александра', '9058956458',
'cremneva@mail.ru', NULL, 'passive')
```

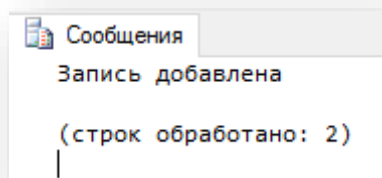


Рис. 3.6 – Результат работы триггера

## Пример 2. Триггер на изменение записи в таблицу USERS

```
CREATE TRIGGER UPDATE_INDICATION
    ON USERS
    AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    PRINT 'Запись изменена'
END
```

### *Тестирование работы триггера*

```
UPDATE USERS
    SET URL = 'cremnera.tomsk.ru'
    WHERE SURNAME = 'Кремнева'
```

## Пример 3. Триггер на удаление записи из таблицы USERS

```
CREATE TRIGGER DELETE_INDICATION
    ON USERS
    AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;
    PRINT 'Запись удалена'
END
GO
```

### *Тестирование работы триггера*

```
DELETE FROM USERS
    WHERE SURNAME = 'Громова'
```



#### Пример 4. Триггер, демонстрирующий откат

```
CREATE TRIGGER ROLLBACK_EXAMPLE
    ON ORDERS
    AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    IF (SELECT NUMBER FROM inserted) < 1
        ROLLBACK
    PRINT 'Вы не можете создать заказ с количеством меньше 1'
END
GO
```

#### *Тестирование работы триггера*

```
INSERT INTO ORDERS VALUES (2, '2005-01-06 12:39:38', 0, 20)
```

**Пример 5.** Триггер на изменение количества товаров при их заказе. Количество проданного товара должно быть не меньше, чем его остаток из таблицы PRODUCTS

```
CREATE TRIGGER NUMBER_UPDATE
    ON ORDERS
    AFTER INSERT
AS
DECLARE @X INT, @Y INT
BEGIN
    SET NOCOUNT ON;
    IF NOT EXISTS(SELECT * FROM inserted
        WHERE inserted.NUMBER <= ALL (SELECT PRODUCTS.COUNT FROM
PRODUCTS WHERE inserted.ID_PRODUCT = PRODUCTS.ID_PRODUCT))
    BEGIN
        ROLLBACK TRAN
        PRINT 'откат! товара нет '
    END
    SELECT @Y = O.ID_PRODUCT, @X=O.NUMBER
    FROM inserted O
    UPDATE PRODUCTS
```

```
SET PRODUCTS.COUNT = PRODUCTS.COUNT - @X
WHERE PRODUCTS.ID_PRODUCT = @Y
END
GO
```

*Тестирование работы триггера*

```
INSERT INTO ORDERS VALUES ( 4, '2005-01-04 18:39:38', 12, 28)
```

## **Заключение**

В данной лабораторной работе были даны теоретические знания и практические навыки по созданию простых запросов, вложенных запросов, используя при этом различные дополнительные параметры как условия, сортировки и так далее. Также были приведены практические примеры по созданию выборок и триггеров.