МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

СЕВЕРО-КАВКАЗСКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ

СРЕДНЕПРОФЕССИОНАЛЬНЫЙ КОЛЛЕДЖ

З. С. Шовкарова

УП.07.01 Учебная практика
Методические указания по организации практики модуля ПМ.07. Соадминистрирование баз данных и серверов для специальности 09.02.07 Информационные системы и программирование

Черкесск
2026

УДК 004
ББК 16.33:32.973
Ш 78

Рассмотрено на заседании ЦК «Информационные и естественнонаучные дисциплины».
Протокол № 1 от «01» 09. 2025 г.
Рекомендовано к изданию редакционно-издательским советом СКГА.
Протокол № 28 от «28» 09. 2025 г.

Рецензенты: Черных Л. А. – председатель ЦК «Информационные дисциплины»

Ш78 **Шовкарова, З. С.** УП.07.01 Учебная практика: методические указания по организации практики модуля ПМ.07. Соадминистрирование баз данных и серверов для специальности 09.02.07 Информационные системы и программирование / З. С. Шовкарова. – Черкесск: БИЦ СКГА, 2026. – 32 с.

Данные методические указания содержат теоретический и практический материал, изучение которого позволит студенту применить полученные знания в процессе учебной практики.

УДК 004
ББК 16.33:32.973

© Шовкарова З. С., 2026
© ФГБОУ ВО СКГА, 2026

ВВЕДЕНИЕ

ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ОТЧЕТА УП.07.01 УЧЕБНАЯ ПРАКТИКА 4
ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ И РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЗАДАНИЙ УП.07.01 Учебная практика 5
1. Постановка целей и задач практики 5
2. Анализ предметной области 5
3. Составление словаря данных. Словарь данных: назначение, структура, префиксы 5
4. Построение схемы базы данных применительно к разрабатываемому заданию 7
5. Рассмотрение вопросов, связанных с созданием запросов, процедур и триггеров. Создание запросов и процедур на изменение структуры базы данных 8
6. Разработка требований к корпоративной сети, применительно к разрабатываемому заданию. 16
7. Решение вопросов, связанных с конфигурированием сети, применительно к разрабатываемому заданию 17
8. Разработка хранимых процедур, применительно к создаваемой учебной базе данных. 23
9. Разработка триггеров, применительно к создаваемой учебной базе данных 28

ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ОТЧЕТА УП.07.01 УЧЕБНАЯ ПРАКТИКА

Оформлять текст отчета следует с соблюдением следующих требований:

– набирается шрифтом Times New Roman строчными буквами (заглавные буквы применяются в заголовках 1-го – 2-го уровней);

– поля: верхнее – 2, нижнее – 2, левое – 3, правое – 1;

– отступ первой строки – 1,25 см, оформление абзацев вручную (пробелом) не допускается;

– размер основного шрифта – 14 pt, вспомогательного (подписи к рисункам, сноски и т.д.) – 12 pt;

– интервал междустрочный – одинарный;

– заголовки отделяются от текста сверху на один интервал больше;

– в тексте должен быть установлен автоматический перенос слов;

– между словами не допускается использование более одного пробела, перед знаком препинания пробел не ставится;

– таблицы следует делать в режиме таблиц, все таблицы нумеруют арабскими цифрами в пределах всего текста;

– ширина таблицы должна соответствовать ширине текстового блока отчета;

– формулы должны быть выведены шрифтом такого же размера и начертания, что и основной текст;

– номера формул заключаются в круглые скобки и выравниваются по правому краю печатного листа;

– рисунки, диаграммы, схемы, графики и чертежи, иллюстрации размещаются в тексте после абзацев, содержащих ссылку на них, и должны быть пронумерованы в последовательности, соответствующей упоминанию их в тексте, и номерами привязаны к подрисуночным подписям (точка в конце названия рисунка не ставится);

– ширина рисунков, схем, диаграмм, графиков, чертежей и ил должна соответствовать ширине текстового блока отчета;

– расположение номера страниц – внизу по центру (нумерация страниц на первом листе (титульном) не ставится).

Печатный вариант отчета формируется в скоросшивателе и сдается руководителю практики вместе с дневником.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ И РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЗАДАНИЙ УП.07.01 Учебная практика

1. Постановка целей и задач практики

Во **введении** прописывается:

– информация о целях учебной практики,

– описание перечня задач, которые планируется решить в ходе прохождения практики в соответствии с рабочей программой,

– указание сроков прохождения учебная практики в соответствии с приказом.

2. Анализ предметной области:

В отчете необходимо:

– описать предметную область,

– перечислить сущности (таблицы) и атрибуты (столбцы) предметной области,

– поставить задачи для разработки приложения.

3. Составление словаря данных. Словарь данных: назначение, структура, префиксы.

Словарь данных – это набор реляционных таблиц и представлений которые содержат информацию о таблицах или метаданные (данные о данных).

Структура словаря данных. В состав словаря данных базы данных входят: Базовые таблицы Основу словаря данных составляет совокупность базовых таблиц, хранящих информацию о базе данных.

Создание словаря базы данных в SQL Server Management Studio (SSMS) включает в себя документирование структуры базы данных, включая таблицы, столбцы, типы данных, ограничения и взаимосвязи. Это помогает лучше понять архитектуру базы данных и облегчает её поддержку и развитие. Вот шаги, которые помогут вам разработать словарь базы данных:

Шаги по разработке словаря базы данных:

1. **Определите объекты базы данных**:

Перечислите все таблицы, представления, хранимые процедуры, функции и индексы, которые вы планируете использовать в базе данных.

2. **Соберите информацию о таблицах**:

Для каждой таблицы соберите следующую информацию:

- **Имя таблицы**: Уникальное имя таблицы.
- **Описание**: Краткое описание назначения таблицы.
- **Столбцы**: Перечислите все столбцы с их именами, типами данных и ограничениями (например, NOT NULL, PRIMARY KEY).
- **Индексы**: Укажите, есть ли индексы на таблице и какие столбцы они охватывают.

– **Внешние ключи**: Перечислите внешние ключи и связи с другими таблицами.

3. Создайте документ словаря:

– Создайте документ в формате Word где вы будете хранить всю собранную информацию.

– Используйте таблицы для структурирования данных, чтобы они были легко читаемыми.

Пример словаря базы данных

Вот пример, как может выглядеть словарь базы данных для простой базы данных, содержащей таблицы "Customers" и "Orders":

Таблица: Customers

Поле	**Тип данных**	**Ограничения**	**Описание**
CustomerID	INT	PRIMARY KEY, IDENTITY	Уникальный идентификатор клиента
FirstName	VARCHAR(50)	NOT NULL	Имя клиента
LastName	VARCHAR(50)	NOT NULL	Фамилия клиента
Email	VARCHAR(100)		Электронная почта клиента
CreatedAt	DATETIME	DEFAULT GETDATE()	Дата создания записи

Таблица: Orders

Поле	**Тип данных**	**Ограничения**	**Описание**
OrderID	INT	PRIMARY KEY, IDENTITY	Уникальный идентификатор заказа
CustomerID	INT	FOREIGN KEY	Идентификатор клиента (внешний ключ)
OrderDate	DATETIME	NOT NULL	Дата заказа
TotalAmount	DECIMAL(10, 2)	NOT NULL	Общая сумма заказа

Пример SQL-запросов для получения информации о таблицах

Вы также можете использовать SQL-запросы для извлечения информации о структуре базы данных:

Получение списка всех таблиц:

```
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE';
```

Получение информации о столбцах конкретной таблицы:

```
SELECT
   COLUMN_NAME,            -- Имя столбца
   DATA_TYPE,              -- Тип данных столбца
   CHARACTER_MAXIMUM_LENGTH,    -- Максимальная длина для строковых типов данных
   IS_NULLABLE                  -- Указывает, может ли столбец содержать NULL
FROM
   INFORMATION_SCHEMA.COLUMNS
WHERE
 TABLE_NAME = 'WorkingCalendar';   -- Указываем имя таблицы
```

Получение информации о внешних ключах:

```
SELECT fk.name AS ForeignKeyName, -- Имя внешнего ключа tp.name
AS ParentTable, -- Имя родительской таблицы cp.name
AS ParentColumn, -- Имя столбца в родительской таблице tr.name
AS ReferencedTable, -- Имя связанной таблицы cr.name
AS ReferencedColumn -- Имя столбца в связанной таблице
FROM sys.foreign_keys AS fk -- Таблица с внешними ключами
INNER JOIN
sys.foreign_key_columns AS fkc ON fk.object_id = fkc.constraint_object_id -- Соединяем с колонками внешних ключей
INNER JOIN sys.tables AS tp ON fkc.parent_object_id = tp.object_id -- Получаем родительскую таблицу
INNER JOIN sys.columns AS cp ON fkc.parent_object_id = cp.object_id AND fkc.parent_column_id = cp.column_id -- Получаем родительский столбец
INNER JOIN sys.tables AS tr ON fkc.referenced_object_id = tr.object_id -- Получаем связанную таблицу
INNER JOIN sys.columns AS cr ON fkc.referenced_object_id = cr.object_id AND fkc.referenced_column_id = cr.column_id -- Получаем связанный столбец
 WHERE tp.name = 'Orders'; -- Указываем имя таблицы, для которой получаем
```

4. Построение схемы базы данных применительно к разрабатываемому заданию.

1. **Анализ требований задания:**

– Изучите спецификацию задания: определите основные объекты (сущности, например, пользователи, товары, заказы), их атрибуты (столбцы, например, имя, цена, дата).

– Определите функциональные требования: какие данные хранить, как связывать (отношения 1:1, 1:N, N:M).

2. **Проектирование схемы:**

– Идентификация сущностей и атрибутов: Составьте список таблиц и их столбцов с типами данных (например, VARCHAR(255) для строк, INT для

целых, DATE для дат, DECIMAL для чисел с плавающей точкой). Назначьте первичные ключи (PRIMARY KEY, часто автоинкрементные).

– Установление отношений: Определите связи (FOREIGN KEY) между таблицами. Для N:M используйте промежуточные таблицы (например, таблица связи для "пользователи" и "роли").

3. Реализация в SQL:

– Создайте базу данных: `CREATE DATABASE your_db_name; USE your_db_name;` (MySQL) или `CREATE DATABASE your_db_name;` (PostgreSQL).

– Напишите скрипт для таблиц: Используйте `CREATE TABLE` с определением столбцов, ключей и ограничений. Пример структуры скрипта (адаптируйте под задание):

– Для MySQL: `CREATE TABLE table_name (column1 INT PRIMARY KEY AUTO_INCREMENT, column2 VARCHAR(255) NOT NULL, FOREIGN KEY (fk_column) REFERENCES other_table(pk_column));`

– Для PostgreSQL: Аналогично, но используйте `SERIAL` для автоинкремента и `REFERENCES` для FK.

– Добавьте индексы для оптимизации: `CREATE INDEX idx_name ON table_name(column);` (для часто запрашиваемых столбцов).

– Выполните скрипт в клиенте СУБД.

В отчет диаграмма сриншот SQL+ описание.

5. Рассмотрение вопросов, связанных с созданием запросов, процедур и триггеров. Создание запросов и процедур на изменение структуры базы данных.

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

Ход работы

1. Создать базу данных используя мастер создания БД в SQL Server Management Studio согласно схеме, представленной на рисунке 1.
2. Написать SQL запросы на добавление данных в таблицы. Данные представлены на рисунках 2-5.
3. Проделать 24 примера на извлечение данных из раздела 4 данного методического пособия. Протоколирую результат выполнения запросов в отчет о проделанной работе.
4. На свое усмотрения создать три триггера из примеров. 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.

Составить отчет о проделанной работе.

Подготовка к выполнению лабораторной работы

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

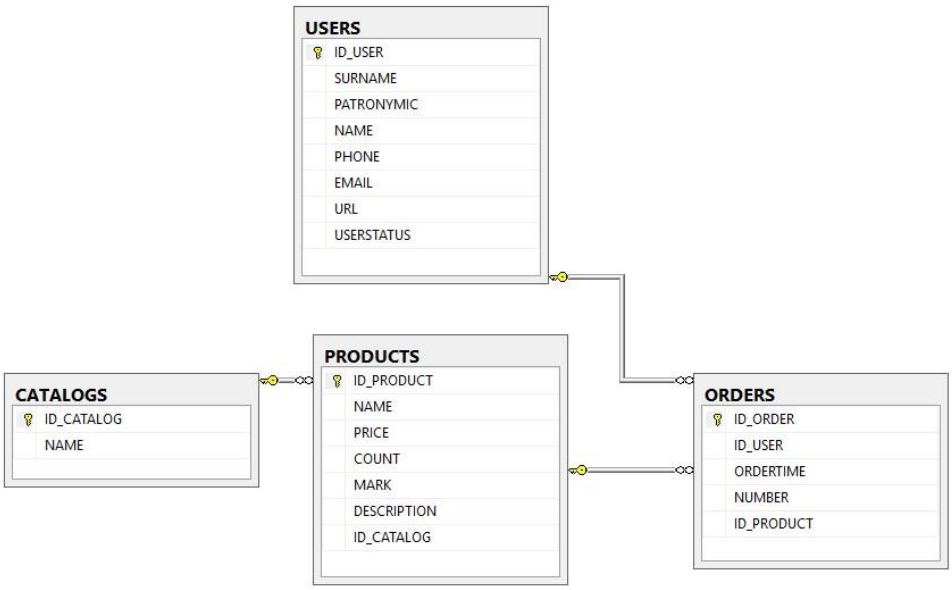


Рисунок 1 – Схема базы данных

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

Рисунок 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\\L i865G	2518.00	6	4	Материнская плата Socket-478 Asustek P4P800-VM\\L i865G FSB800/53...	2
15	15	Epox EP-4PDA3I	2289.00	5	4	Материнская плата Socket-478 Epox 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

Рисунок 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	korneev@domen.ru	NULL	gold

Рисунок 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

Рисунок 5 – Данные в таблице ORDERS

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
SELECT * FROM CATALOGS
```

Пример 2. Вывод данных таблицы CATALOGS с присвоением псевдонима

```
SELECT
ID_CATALOG AS '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
```

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
SELECT * FROM CATALOGS
WHERE ID_CATALOG BETWEEN 3 AND 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
```

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

```
SELECT * FROM CATALOGS
WHERE ID_CATALOG IN (1, 2, 5)
```

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

```
SELECT * FROM CATALOGS
WHERE NAME = 'процессоры'
```

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

```
SELECT * FROM CATALOGS
WHERE NOT NAME = 'процессоры'
```

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

```
SELECT * FROM USERS
WHERE SURNAME LIKE '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
SELECT * FROM CATALOGS
ORDER BY 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
```

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

```
SELECT ORDERTIME FROM ORDERS
ORDER BY ORDERTIME DESC
```

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'
FROM ORDERS
```

Пример 16. Подсчет среднего количества товаров в одном заказе

```
SELECT AVG (NUMBER) AS 'Среднее количество'
FROM ORDERS
```

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

```
SELECT COUNT(ID_ORDER)
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)
FROM CATALOGS
SELECT TOP 1 * FROM CATALOGS
ORDER BY ID_CATALOG
```

Пример 20. Извлечение минимального 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)
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
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 TOTAL > 5
ORDER BY ID_CATALOG
```

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.

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,
                                 --и как его выполнять
                                 --AFTER – после выполнения операции,
```

```
AS                          --INSTEAD OF – вместо выполнения операции
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')
```

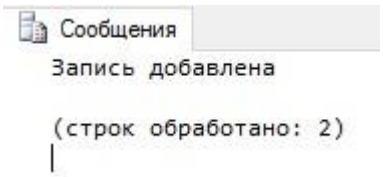


Рисунок 6 – Результат работы триггера

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)
```

6. Разработка требований к корпоративной сети, применительно к разрабатываемому заданию.

Студент должен описать свой вариант построения локальной сети в придуманной фирме, с обоснование всех её недостатков и достоинств и с расчётом стоимости её построения.

Учащимся предлагается описать один из видов компьютерных сетей, рассчитать её стоимость и сделать вывод о том, где целесообразнее её использовать.

Схема локальной сети		
Недостатки		
Преимущества		
Принцип действия:		
Оборудование, необходимое для создания сети и его стоимость	оборудование	стоимость
Общая стоимость создания локальной сети		
Выводы:		

Разработка требований к корпоративной сети

Проанализируйте информация на тему требования к корпоративным сетям.

Составьте краткий доклад по вариантам:

1	Требования к производительности
2	Требования к архитектуре
3	Требования к интерфейсам между компонентами корпоративной сети
4	Требования к управлению и диагностированию
5	Требования к используемому программному и аппаратному обеспечению
6	Требования к электроснабжению
7	Требования к идентификации
8	Требования к восстановлению работоспособности

7. Решение вопросов, связанных с конфигурированием сети, применительно к разрабатываемому заданию.

Конфигурация сети (локальной и глобальной)

Возможно наиболее важным требованием к конфигурации NFS-сервера является обеспечение достаточной полосы пропускания и степени готовности сети. Это требование на практике трансформируется в необходимость создания конфигурации с соответствующим количеством и типом сетей и интерфейсов.

Сетевая среда, определяемая профилем приложения

Как отмечалось ранее, наиболее важным фактором, определяющим выбор конфигурации сети, является доминирующий тип операций NFS, используемых приложениями. Для приложений с интенсивной нагрузкой по данным требуется относительно небольшое количество сетей, но эти сети должны иметь большую полосу пропускания, как например, в сетях FDDI или CDDI. Эти требования могут удовлетворяться также с помощью сетей 100baseT (Ethernet 100 Мбит/с) или ATM (Asynchronous Transfer Mode 155 Мбит/с). Большинство интенсивных по атрибутам приложений работают и при наличии менее дорогой инфраструктуры, хотя может потребоваться большое количество сетей.

Принять решение по выбору типа сети сравнительно просто. Если для работы индивидуального клиента требуется агрегатированная скорость передачи данных, превышающая 1 Мбайт/с, или если для одновременной работы нескольких клиентов необходима полоса пропускания сети, превышающая 1 Мбайт/с, то такие приложения требуют применения высокоскоростных сетей. Реально эта цифра (1 Мбайт/с) искусственно завышена, поскольку она характеризует скорость передачи данных, которую вы гарантируете не превышать. Обычно более разумно рассматривать

скорость сети Ethernet равной примерно 440 Кбайт/с, а не 1 Мбайт/с. (Обычно пользователи воспринимают Ethernet как "неотвечающую" уже примерно при 35% загрузке сети. Приведенная здесь цифра 440 Кбайт/с соответствует 35%-ной загрузке линии с пропускной способностью 1.25 Мбайт/с).

Если приложение в установившемся режиме работы не требует широкой полосы пропускания, то возможно будет достаточна менее скоростная сетевая среда типа Ethernet или TokenRing. Эта среда обеспечивает достаточную скорость передачи данных при выполнении операций lookup и getattr, которые доминируют в приложениях с интенсивным использованием атрибутов, а также относительно легкий трафик данных, связанный с таким использованием.

Использование высокоскоростных сетей для предотвращения перегрузки

Высокоскоростные сети наиболее полезны для обслуживания больших групп клиентов с интенсивной нагрузкой по данным скорее из-за более низкой стоимости инфраструктуры, а не по причине обеспечения максимальной пропускной способности при взаимодействии одной системы с другой. Причиной этого является текущее состояние протокола NFS, который в настоящее время работает с блоками данных длиной 8 Кбайт и обеспечивает предварительную выборку только 8 Кбайт (т.е. в одной операции с сервером можно определить максимально 16 Кбайт данных).

Общий эффект такой организации проявляется в том, что максимальная скорость передачи данных между клиентом и сервером, которые взаимодействуют через кольцо FDDI, составляет примерно 2.7 Мбайт/с. (Эта скорость достигается только при добавлении в файл /etc/system на клиенте оператора set nfs: nfs_async_threads = 16. Клиенты SunOS 4.1.x должны запускать 12 демонов biod, а не 8, как это делается по умолчанию). Эта скорость всего в три раза превосходит максимальную скорость, которую обеспечивает Ethernet несмотря на то, что скорость среды FDDI в десять раз больше. (NFS представляет собой протокол прикладного уровня (уровня 7 в модели OSI). Протоколы более низких уровней, такие как TCP и UDP могут работать с гораздо более высокими скоростями, используя те же самые аппаратные средства. Большая часть времени тратится на ожидание ответов и другую обработку прикладного уровня. Другие протоколы прикладного уровня, которые не рассчитаны на немедленное получение ответа и/или подтверждения, также могут эффективно использовать значительно более высокую скорость среды). Пиковая скорость при использовании 16 Мбит/с Token Ring составляет примерно 1.4 Мбайт/с. Сравнительно недавно была анонсирована новая версия протокола NFS+, которая устраняет этот недостаток, разрешая работу с блоками значительно больших размеров. NFS+ допускает пересылку блоков данных почти произвольных размеров. Клиент и сервер договариваются о максимальном размере блока при каждом монтировании файловой системы. При этом размер блока может увеличиваться до 4 Гбайт.

Главное преимущество 100-Мбитных сетей при работе с обычными версиями NFS заключается в том, что эти сети могут поддерживать много одновременных передач данных без деградации. Когда сервер пересылает по Ethernet данные клиенту со скоростью 1 Мбайт/с, то такая передача потребляет 100% доступной полосы пропускания сети. Попытки передачи по этой сети большего объема данных приводят к более низкой пропускной способности для всех пользователей. Те же самые клиент и сервер могут осуществлять пересылки данных со скоростью 2.7 Мбайт/с по кольцу FDDI, но в более высокоскоростной сети эта транзакция потребляет только 21% доступной полосы пропускания. Сеть может поддерживать пять или шесть пересылок одновременно без серьезной деградации.

Эту ситуацию лучше всего можно сравнить со скоростной магистралью. Когда движение небольшое (легкий трафик) скоростная магистраль с двумя полосами и ограничением скорости в 90 км в час почти так же хороша, как и восьмиполосная супермагистраль с ограничением скорости 120 км в час. Но когда движение очень интенсивное (тяжелый трафик) супермагистраль гораздо менее чувствительна к перегрузке.

Сеть FDDI также немного (примерно на 5%) более эффективна по сравнению с Ethernet и Token Ring в среде с интенсивной пересылкой данных, поскольку в ее пакете можно разместить больший объем полезных данных (4500 байт по сравнению с 1500 байт у Ethernet и 2048 байт у Token Ring). При пересылках данных объемом 8 Кбайт требуется обработать всего два пакета по сравнению с пятью-шестью для Token Ring или Ethernet. Но все эти рассуждения имеют смысл только для среды с интенсивной передачей данных, поскольку объем атрибутов при обработке соответствующих запросов настолько мал (по 80-128 байт), что для их передачи требуется только один пакет независимо от типа используемой сети. Если существующая на предприятии проводка сети заранее исключает возможность применения оптоволоконной среды FDDI, то существуют стандарты "FDDI по медным проводам" (CDDI), которые обеспечивают возможность предотвращения перегрузки сети при сохранении существующей разводки на основе витой пары.

Хотя ATM до сих пор не превратилась в повсеместно применяемую технологию, возможно в будущем она станет основным средством для среды с интенсивной пересылкой данных, поскольку она обеспечивает более высокую скорость передачи данных (в настоящее время определены скорости передачи данных 155 Мбит/с, 622 Мбит/с и 2.4 Гбит/с), а также использует топологию точка-точка, в которой каждое соединение клиент-хаб может работать со своей определенной скоростью среды.

NFS и глобальные сети

В реальной жизни часто возникают ситуации, когда клиенты и серверы NFS могут располагаться в разных сетях, объединенных маршрутизаторами. Топология сети может существенно отразиться на ощущаемой пользователем производительности сервера NFS и обеспечиваемого им сервиса. Эффективность обеспечения NFS-сервиса через комплексные сети должна

тщательно анализироваться. Но по крайней мере известно, что можно успешно сконфигурировать сети и приложения в глобальных (wide-area) топологиях NFS.

Возможно наиболее важным вопросом в этой ситуации является задержка выполнения операций: время, которое проходит между выдачей запроса и получением ответа. Задержка выполнения операций в локальных сетях не столь критична, поскольку связанные с такими сетями сравнительно короткие расстояния не могут вызвать значительных задержек в среде передачи данных. В глобальных сетях задержки выполнения операций могут происходить просто при транспортировке пакетов из одного пункта в другой. Задержка передачи пакетов складывается из нескольких составляющих:

Задержка маршрутизатора: маршрутизаторы затрачивают некоторое конечное (и часто существенное) время на выполнение собственно маршрутизации пакетов из одной сети в другую. Заметим, что при построении большинства глобальных сетей (даже при прокладке линий между двумя соседними зданиями) используются по крайней мере два маршрутизатора. На рисунке 4.6 представлена топология типичного университетского кампуса, в котором обычно между клиентом и сервером устанавливаются три или даже четыре маршрутизатора.

Задержка передачи по сети: физическая среда, используемая для передачи пакетов через глобальные сети, часто может вносить свою собственную существенную задержку, превосходящую по величине задержку маршрутизаторов. Например, организация спутниковых мостов часто связана с появлением очень больших задержек.

Ошибочные передачи: глобальные сети возможно на порядок величины более восприимчивы к ошибкам передачи, чем большинство локальных сетей. Эти ошибки вызывают значительный объем повторных пересылок данных, что приводит как к увеличению задержки выполнения операций, так и к снижению эффективной пропускной способности сети. Для сетей, сильно подверженных ошибкам передачи, размер блока данных NFS часто устанавливается равным 1 Кбайт, вместо нормальных 8 Кбайт. Это позволяет сократить объем данных, которые должны повторно передаваться в случае появления ошибки.

Если обеспечивается приемлемый уровень безошибочных передач данных, файловый сервис по глобальным сетям вполне возможен. Наиболее часто в конфигурации таких сетей используются высокоскоростные синхронные последовательные линии связи точка-точка, которые подсоединяются к одной или нескольким локальным сетям на каждом конце. В США такие последовательные линии связи обычно имеют скорость передачи данных 1.544 Мбит/с (линия T1) или 56 Кбит/с. Европейские коммуникационные компании предлагают немного большие скорости: 2.048 Мбит/с (линия E1) или 64 Кбит/с соответственно. Доступны даже более высокоскоростные линии передачи данных. Эти арендуемые линии, известные под названием T3, обеспечивают скорость передачи до 45 Мбит/с

(5.3 Мбайт/с). На сегодня большинство линий T3 частично используются для передачи данных.

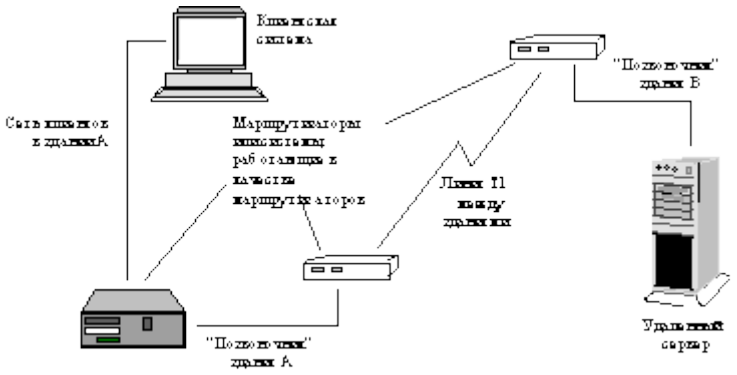


Рисунок 7 – Типичная топология сети при организации связи между зданиями

На первый взгляд кажется, что эти линии значительно более медленные по сравнению с локальными сетями, к которым они подсоединяются. Однако в действительности быстрые последовательные линии (T1) обеспечивают пропускную способность гораздо более близкую к реальной пропускной способности локальных сетей. Это происходит потому, что последовательные линии могут использоваться почти со 100% загрузкой без чрезмерных накладных расходов, в то время как сети Ethernet обычно насыщаются уже примерно при 440 Кбайт/с (3.5 Мбит/с), что всего примерно вдвое превышает пропускную способность линии T1. По этой причине файловый сервис по высокоскоростным последовательным линиям связи возможен и позволяет передавать данные с приемлемыми скоростями. В частности, такая организация оказывается полезной при передаче данных между удаленными офисами. В приложениях с интенсивной обработкой атрибутов работа NFS по глобальным сетям может быть успешной, если задержка выполнения операций не является критичной. В глобальной сети короткие пакеты передаются через каждый сегмент достаточно быстро (при высокой пропускной способности), хотя задержки маршрутизации и самой среды часто вызывают значительную задержку выполнения операций.

Выводы:

Для реализации глобальных сервисов NFS подходят последовательные линии T1, E1 или T3.

Для большинства применений NFS линии со скоростями передачи 56 и 64 Кбит/с обычно оказываются недостаточно быстрыми.

При организации NFS через глобальные сети существуют проблемы с задержками сети и маршрутизации. Пропускная способность сети обычно не вызывает проблем.

Для существенного сокращения трафика по глобальной сети, можно использовать на клиентских системах кэширующую файловую систему (CFS), если только в этом трафике не доминируют операции записи NFS.

Выбор типа сети и количества клиентов

Учитывая вышеизложенные соображения, для определения надлежащего типа и числа сетей могут быть использованы следующие эмпирические правила:

Если в приложении доминируют операции с данными, следует выбрать сеть FDDI или какую-нибудь другую высокоскоростную сеть. Если по материально-техническим причинам прокладка оптоволоконных кабелей не представляется возможной, следует рассмотреть возможность реализации FDDI на витых парах. При создании новой системы следует иметь в виду, что для сетей ATM используются те же самые кабели, что и для FDDI.

В конфигурации сети необходимо предусмотреть одно кольцо FDDI для каждых 5-7 клиентов, одновременно полностью активных в смысле NFS и интенсивно работающих с данными. Следует помнить, что очень немногие интенсивные по данным приложения непрерывно генерируют запросы к серверу NFS. В типичных интенсивных по данным приложениях автоматизации проектирования электронных устройств и системах исследования земных ресурсов это часто позволяет иметь до 25-40 клиентов на кольцо.

В системах с интенсивным использованием данных, где существующая система кабелей вынуждает использовать Ethernet, следует предусмотреть отдельную сеть Ethernet для каждых двух активных клиентов и максимально 4-6 клиентов на одну сеть.

Если приложение связано с интенсивной обработкой атрибутов, то вполне достаточно построения сетей Ethernet или Token Ring.

В среде с интенсивным использованием атрибутов следует иметь одну сеть Ethernet на 8-10 полностью активных клиентов. Неблагоразумно превышать уровень 20-25 клиентов на Ethernet независимо от требований из-за резкой деградации, возникающей в случае активности многих клиентов. В качестве контрольной точки с точки зрения здравого смысла можно считать, что Ethernet способен поддерживать 250-300 NFS-операций в секунду на тесте SPECsfs_97 (LADDIS) даже с высоким уровнем коллизий. Неразумно превышать уровень 200 операций NFS в секунду в установившемся режиме.

Следует конфигурировать одну сеть TokenRing для каждых 10-15 полностью активных клиентов в среде с интенсивным использованием атрибутов. Если необходимо, к сети Token Ring можно подключать 50-80 клиентов благодаря превосходным характеристикам этого типа сети по устойчивости к деградации при тяжелой нагрузке (по сравнению с Ethernet).

Для систем, которые обеспечивают сервис нескольким классам пользователей имеют смысл смешанные конфигурации сетей. Например, и FDDI, и Token Ring подходят для сервера, который поддерживает как приложения, связанные с отображением документов (интенсивные по данным), так и группу ПК, выполняющих приложение финансового анализа (возможно интенсивное по атрибутам).

8. Разработка хранимых процедур, применительно к создаваемой учебной базе данных.

Нередко операция с данными представляет набор инструкций, которые необходимо выполнить в определенной последовательности. Например, при добавлении данных покупки товара необходимо внести данные в таблицу заказов. Однако перед этим надо проверить, а есть ли покупаемый товар в наличии. Возможно, при этом понадобится проверить еще ряд дополнительных условий. То есть фактически процесс покупки товара охватывает несколько действий, которые должны выполняться в определенной последовательности. И в этом случае более оптимально будет инкапсулировать все эти действия в один объект - **хранимую процедуру** (stored procedure).

То есть по сути хранимые процедуры представляют набор инструкций, которые выполняются как единое целое. Тем самым хранимые процедуры позволяют упростить комплексные операции и вынести их в единый объект. Изменится процесс покупки товара, соответственно достаточно будет изменить код процедуры. То есть процедура также упрощает управление кодом.

Также хранимые процедуры позволяют ограничить доступ к данным в таблицах и тем самым уменьшить вероятность преднамеренных или неосознанных нежелательных действий в отношении этих данных.

И еще один важный аспект - производительность. Хранимые процедуры обычно выполняются быстрее, чем обычные SQL-инструкции. Все потому что код процедур компилируется один раз при первом ее запуске, а затем сохраняется в скомпилированной форме.

Для создания хранимой процедуры применяется команда **CREATE PROCEDURE** или **CREATE PROC**.

Таким образом, хранимая процедура имеет три ключевых особенности: упрощение кода, безопасность и производительность.

Например, пусть в базе данных есть таблица, которая хранит данные о товарах:

```
CREATE TABLE Products
(
    Id INT IDENTITY PRIMARY KEY,
    ProductName  NVARCHAR(30)  NOT
NULL,
    Manufacturer  NVARCHAR(20)  NOT
NULL,
    ProductCount INT DEFAULT 0,
    Price MONEY NOT NULL
);
```

Создадим хранимую процедуру для извлечения данных из этой таблицы:

```
USE productsdb;
```

GO

CREATE PROCEDURE ProductSummary AS

SELECT ProductName AS Product, Manufacturer, Price

FROM Products

Поскольку команда CREATE PROCEDURE должна вызываться в отдельном пакете, то после команды USE, которая устанавливает текущую базу данных, используется команда **GO** для определения нового пакета.

После имени процедуры должно идти ключевое слово AS.

Для отделения тела процедуры от остальной части скрипта код процедуры нередко помещается в блок BEGIN...END:

USE productsdb;

GO

CREATE PROCEDURE ProductSummary AS

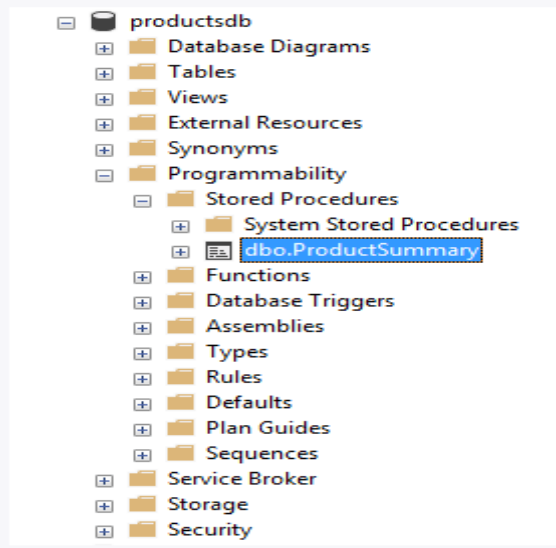
BEGIN

SELECT ProductName AS Product, Manufacturer, Price

FROM Products

END;

После добавления процедуры мы ее можем увидеть в узле базы данных в SQL Server Management Studio в подузле **Programmability -> Stored Procedures**:



И мы сможем управлять процедурой также и через визуальный интерфейс.

Выполнение процедуры

Для выполнения хранимой процедуры вызывается команда **EXEC** или **EXECUTE**:

```
1 EXEC ProductSummary
```

```
1  USE productsdb;
2
3  EXEC ProductSummary
```

100 %

Results | Messages

	Product	Manufacturer	Price
1	iPhone 6	Apple	36000,00
2	iPhone 6S	Apple	41000,00
3	iPhone 7	Apple	52000,00
4	Galaxy S8	Samsung	46000,00
5	Galaxy S8 Plus	Samsung	56000,00
6	Mi 5X	Xiaomi	26000,00
7	OnePlus 5	OnePlus	38000,00

Удаление процедуры

Для удаления процедуры применяется команда **DROP PROCEDURE**:

```
1        DROP PROCEDURE ProductSummary
```

Параметры в процедурах

Процедуры могут принимать параметры. Параметры бывают входными – с их помощью в процедуру можно передать некоторые значения. И также параметры бывают выходными - они позволяют возвратить из процедуры некоторое значение.

Например, пусть в базе данных будет следующая таблица Products:

```
USE productsdb;
CREATE TABLE Products
(
   Id INT IDENTITY PRIMARY KEY,
   ProductName NVARCHAR(30) NOT NULL,
   Manufacturer NVARCHAR(20) NOT NULL,
   ProductCount INT DEFAULT 0,
   Price MONEY NOT NULL
);
```

Определим процедуру, которая будет добавлять данные в эту таблицу:

```
USE productsdb;
GO
CREATE PROCEDURE AddProduct
   @name NVARCHAR(20),
   @manufacturer NVARCHAR(20),
   @count INT,
   @price MONEY
AS
INSERT INTO Products(ProductName, Manufacturer, ProductCount, Price)
VALUES(@name, @manufacturer, @count, @price)
```

После названия процедуры идет список входных параметров, которые определяются также как и переменные - название начинается с символа **@**, а

после названия идет тип переменной. И с помощью команды INSERT значения этих параметров будут передаваться в таблицу Products.

Используем эту процедуру:

```
USE productsdb;

DECLARE @prodName NVARCHAR(20), @company NVARCHAR(20);
DECLARE @prodCount INT, @price MONEY
SET @prodName = 'Galaxy C7'
SET @company = 'Samsung'
SET @price = 22000
SET @prodCount = 5
 EXEC AddProduct @prodName, @company, @prodCount, @price
 SELECT * FROM Products
```

Здесь передаваемые в процедуру значения определяются через переменные. При вызове процедуры ей через запятую передаются значения. При этом значения передаются параметрам процедуры **по позиции**. Так как первым определен параметр @name, то ему будет передаваться первое значение – значение переменной @prodName. Второму параметру – @manufacturer передается второе значение – значение переменной @company и так далее. Главное, чтобы между передаваемыми значениями и параметрами процедуры было соответствие по типу данных.

```
1  USE productsdb;
2
3  DECLARE @prodName NVARCHAR(20), @company NVARCHAR(20);
4  DECLARE @prodCount INT, @price MONEY
5  SET @prodName = 'Galaxy C7'
6  SET @company = 'Samsung'
7  SET @price = 22000
8  SET @prodCount = 5
9
10 EXEC AddProduct @prodName, @company, @prodCount, @price
11
12 SELECT * FROM Products
```

00 %

Results | Messages

	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	2	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi 5X	Xiaomi	2	26000,00
7	7	OnePlus 5	OnePlus	6	38000,00
8	1004	Galaxy C7	Samsung	5	22000,00

Также можно было бы передать непосредственно значения:

```
EXEC AddProduct 'Galaxy C7', 'Samsung', 5, 22000
```

Также значения параметрам процедуры можно передавать по имени:

```
USE productsdb;

DECLARE @prodName NVARCHAR(20), @company NVARCHAR(20);
SET @prodName = 'Honor 9'
SET @company = 'Huawei'

EXEC AddProduct @name = @prodName,
        @manufacturer=@company,
        @count = 3,
        @price = 18000
```

При передаче параметров по имени параметру процедуры присваивается некоторое значение.

Необязательные параметры

Параметры можно отмечать как необязательные, присваивая им некоторое значение по умолчанию. Например, в случае выше мы можем автоматически устанавливать для количества товара значение 1, если соответствующее значение не передано в процедуру:

```
USE productsdb;
GO
CREATE PROCEDURE AddProductWithOptionalCount
   @name NVARCHAR(20),
   @manufacturer NVARCHAR(20),
   @price MONEY,
   @count INT = 1
AS
INSERT INTO Products(ProductName, Manufacturer, ProductCount, Price)
VALUES(@name, @manufacturer, @count, @price)
```

При этом необязательные параметры лучше помещать в конце списка параметров процедуры.

```
DECLARE @prodName NVARCHAR(20), @company NVARCHAR(20), @price MONEY
SET @prodName = 'Redmi Note 5A'
SET @company = 'Xiaomi'
SET @price = 22000

EXEC AddProductWithOptionalCount @prodName, @company, @price

SELECT * FROM Products
```

И в этом случае для параметра @count в процедуру можно не передавать значение.

9. Разработка триггеров, применительно к создаваемой учебной базе данных.

Триггеры для операций INSERT, UPDATE, DELETE

Для рассмотрения операций с триггерами определим следующую базу данных productsdb:

```
CREATE DATABASE productsdb;
GO
USE productsdb;
CREATE TABLE Products
(
   Id INT IDENTITY PRIMARY KEY,
   ProductName NVARCHAR(30) NOT NULL,
   Manufacturer NVARCHAR(20) NOT NULL,
   ProductCount INT DEFAULT 0,
   Price MONEY NOT NULL
);
CREATE TABLE History
(
   Id INT IDENTITY PRIMARY KEY,
   ProductId INT NOT NULL,
   Operation NVARCHAR(200) NOT NULL,
   CreateAt DATETIME NOT NULL DEFAULT GETDATE(),
);
```

Здесь определены две таблиц: Products - для хранения товаров и History – для хранения истории операций с товарами.

Добавление

При добавлении данных (при выполнении команды **INSERT**) в триггере мы можем получить добавленные данные из виртуальной таблицы **INSERTED**.

Определим триггер, который будет срабатывать после добавления:

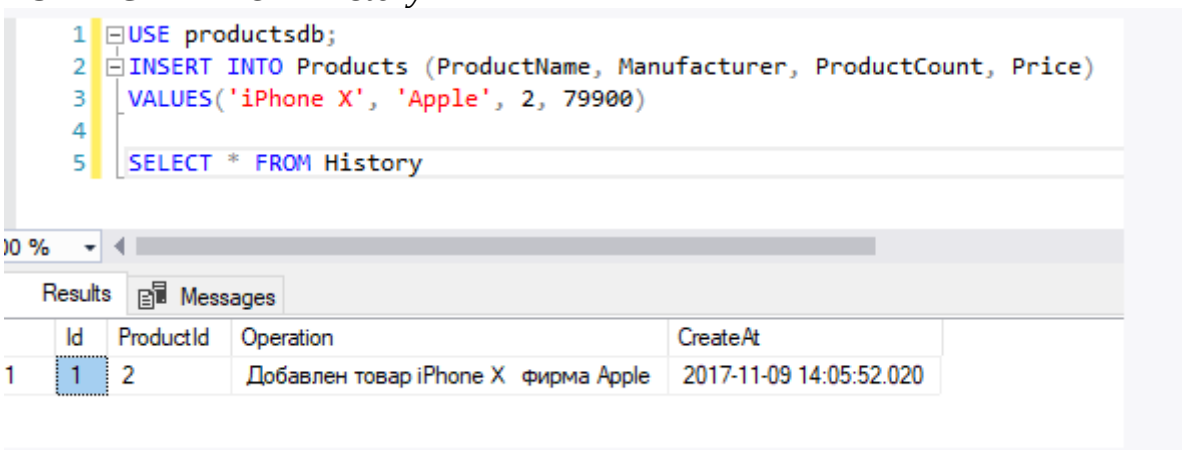
```
USE productsdb
GO
CREATE TRIGGER Products_INSERT
ON Products
AFTER INSERT
AS
INSERT INTO History (ProductId, Operation)
SELECT Id, 'Добавлен товар ' + ProductName + ' фирма ' + Manufacturer
FROM INSERTED
```

Этот триггер будет добавлять в таблицу History данные о добавлении товара, которые берутся из виртуальной таблицы INSERTED.

Выполним добавление данных в Products и получим данные из таблицы History:

USE productsdb;
INSERT INTO Products (ProductName, Manufacturer, ProductCount, Price)
VALUES('iPhone X', 'Apple', 2, 79900)

*SELECT * FROM History*



Удаление данных

При удалении все удаленные данные помещаются в виртуальную таблицу **DELETED**:

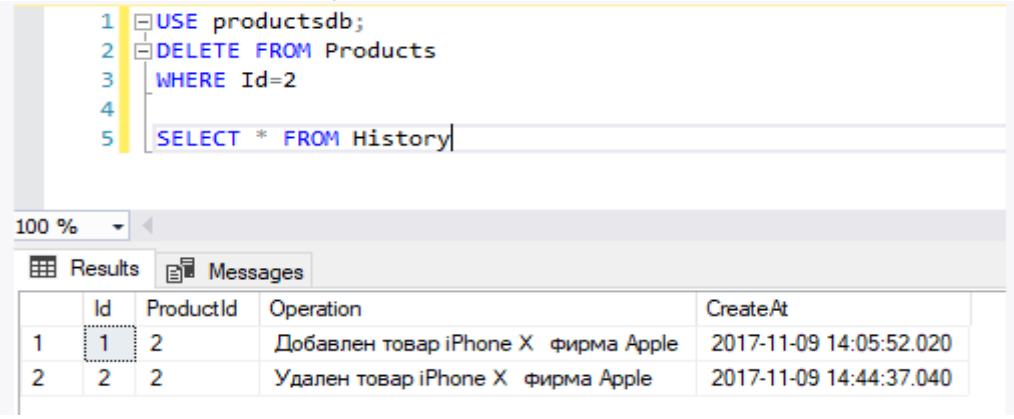
USE productsdb
GO
CREATE TRIGGER Products_DELETE
ON Products
AFTER DELETE
AS
INSERT INTO History (ProductId, Operation)
SELECT Id, 'Удален товар ' + ProductName + ' фирма ' + Manufacturer
FROM DELETED

Здесь, как и в случае с предыдущим триггером, помещаем информацию об удаленных товарах в таблицу History.

Выполним команду на удаление:

USE productsdb;
DELETE FROM Products
WHERE Id=2

*SELECT * FROM History*



Изменение данных

Триггер обновления данных срабатывает при выполнении операции UPDATE. И в таком триггере мы можем использовать две виртуальных таблицы. Таблица INSERTED хранит значения строк после обновления, а таблица DELETED хранит те же строки, но до обновления.

Создадим триггер обновления:

```
USE productsdb
GO
CREATE TRIGGER Products_UPDATE
ON Products
AFTER UPDATE
AS
INSERT INTO History (ProductId, Operation)
SELECT Id, 'Обновлен товар ' + ProductName + ' фирма ' + Manufacturer
FROM INSERTED
```

И при обновлении данных сработает данный триггер:

```
1  USE productsdb;
2  INSERT INTO Products(ProductName, Manufacturer, ProductCount, Price)
3  VALUES('C350', 'Motorola', 10, 2100);
4
5  UPDATE Products SET Manufacturer='Moto'
6  WHERE Manufacturer='Motorola';
7
8  SELECT * FROM History
```

100 %

Results | Messages

	Id	ProductId	Operation	CreateAt
1	1	2	Добавлен товар iPhone X фирма Apple	2017-11-09 14:05:52.020
2	2	2	Удален товар iPhone X фирма Apple	2017-11-09 14:44:37.040
3	3	3	Добавлен товар C350 фирма Motorola	2017-11-09 15:00:18.130
4	4	3	Обновлен товар C350 фирма Moto	2017-11-09 15:02:04.733

ШОВКАРОВА Зарина Сейтбиевна

УП.07.01 Учебная практика
Методические указания по организации практики модуля ПМ.07. Соадминистрирование баз данных и серверов для специальности 09.02.07 Информационные системы и программирование

Корректор Чагова О. Х.
Редактор Чагова О. Х.

Сдано в набор 27. 02. 2026 г.
Формат 60х84/16
Бумага офсетная.
Печать офсетная.
Усл. печ. л. 1,86
Заказ № 5285
Тираж 100 экз.

Оригинал-макет подготовлен
в Библиотечно-издательском центре СКГА
369000, г. Черкесск, ул. Ставропольская, 36