**Лабораторная работа №1. Основы работы в SQL Server Management Studio**

ДЭиВИ. 2 курс. 9 группа. Косс Валерия. 10 вариант

1. Создать базу данных с помощью утилиты **SQL Server Management Studio.** Имя базы − фамилия студента и слово **ПРОДАЖИ**. Изучить файлы, которые при этом создаются.



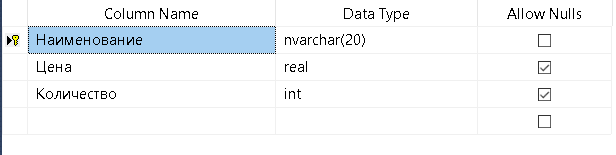
2. Создать таблицу ТОВАРЫ, содержащую поля:

**Наименование** (nvarchar(20))**,**

**Цена** (real)**,**

**Количество** (int).

Сделать первый столбец первичным ключом.



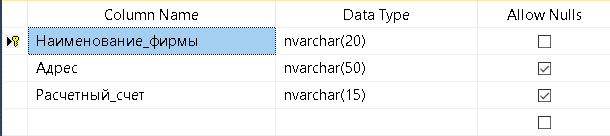
3. Создать таблицу ЗАКАЗЧИКИ, содержащую поля:

**Наименование*\_*фирмы** (nvarchar(20)),

**Адрес** (nvarchar(50)),

**Расчетный\_счет** (nvarchar(15)).

Сделать первый столбец первичным ключом.



4. Создать таблицу ЗАКАЗЫ, содержащую поля:

**Номер\_заказа** (nvarchar(10)),

**Наименование\_товара** (nvarchar(20)),

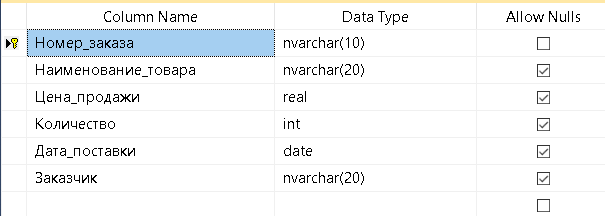
**Цена\_продажи** (real),

**Количество** (int),

**Дата\_поставки** (date),

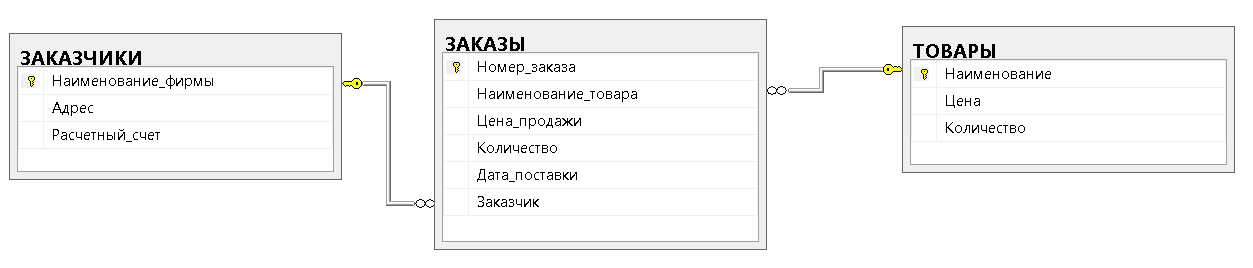
**Заказчик** (nvarchar(20)).

Сделать первый столбец первичным ключом.



5. Установить связи между таблицами ЗАКАЗЧИКИ и ЗАКАЗЫ по полям **Наименование\_фирмы** и **Заказчик.**

Установить связи между таблицами ТОВАРЫ и ЗАКАЗЫ по полям **Наименование** и **Наименование\_ товара**.



6. Сформировать следующие запросы и проанализировать результаты:

– определить товары, поставки которых должны осуществиться после некоторой даты;

SELECT Наименование\_товара, Дата\_поставки

FROM ЗАКАЗЫ

WHERE (Дата\_поставки > CONVERT(DATETIME, '2022-03-03 00:00:00', 102))

ORDER BY Дата\_поставки

– найти товары, цена которых находится в некоторых пределах;

SELECT Наименование, Цена

FROM ТОВАРЫ

WHERE (Цена > 700) AND

(Цена < 2000)

– определить названия фирм, заказавших конкретный товар;

SELECT Наименование\_товара, Заказчик

FROM ЗАКАЗЫ

WHERE (Наименование\_товара = N'Гитара Cort')

– найти заказы определенной фирмы по ее названию, отсортировать их по датам поставки.

SELECT Дата\_поставки, Наименование\_товара, Заказчик

FROM ЗАКАЗЫ

WHERE (Заказчик = N'МузШок')

ORDER BY Дата\_поставки

Сохранить запросы в sql-скрипте.

7. Ознакомиться с **sql**-скриптами, сгенерировать скрипты для создания таблиц.

USE [Koss\_SALES]

GO

/\*\*\*\*\*\* Object: Table [dbo].[ЗАКАЗЧИКИ] Script Date: 21.02.2022 21:58:42 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[ЗАКАЗЧИКИ](

[Наименование\_фирмы] [nvarchar](20) NOT NULL,

[Адрес] [nvarchar](50) NULL,

[Расчетный\_счет] [nvarchar](15) NULL,

CONSTRAINT [PK\_ЗАКАЗЧИКИ] PRIMARY KEY CLUSTERED

(

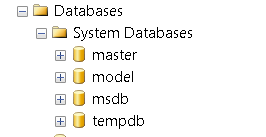
[Наименование\_фирмы] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

8. Найти системные БД, просмотреть в них таблицы.



**Лабораторная работа №2. Проектирование баз данных. Нормализация**

1.Изучить способ приведения информации к *первой нормальной форме*, проанализировав пример в правой части.

Чтобы таблица соответствовала **1-й нормальной форме** (1NF), необходимо, чтобы все значения ее полей были неделимыми и не вычисляемыми, а все записи – уникальными (не должно быть полностью совпадающих строк).

2.Изучить способ приведения ко *второй нормальной форме*, проанализировав пример в правой части.

Чтобы таблица соответствовала **2-й нормальной форме** (2NF), необходимо, чтобы она находилась в 1-й нормальной форме и все не ключевые поля полностью зависели от ключевого.

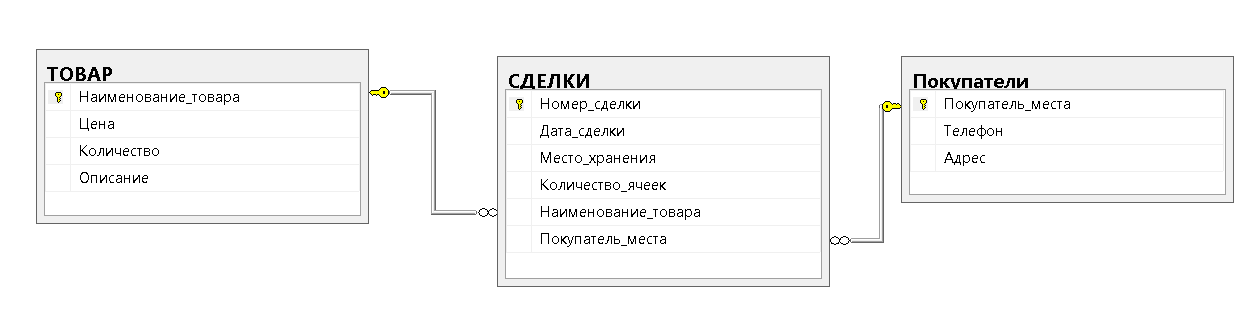
3.Ознакомиться с *третьей нормальной формой*.

Для перехода к **3-й нормальной форме** (3NF), необходимо обеспечить, чтобы все таблицы находились во 2-й нормальной форме и все не ключевые поля в таблицах не зависели взаимно друг от друга.

4. Определить *типы данных* для полей нормализованных таблиц базы данных **ПРОДАЖИ**.

5. В соответствии со своим вариантом (вариант 10), номер которого определяет преподаватель, провести нормализацию исходной информации из таблицы, представленной ниже, создав как минимум три таблицы. При необходимости использовать дополнительные поля.

**Склад.** Компания предоставляет услугу хранения товара на складе. Информационные поля: Наименование товара, Цена, Количество, Описание, Место хранения, Покупатель, Телефон, Адрес, Дата сделки, Количество ячеек.



**Лабораторная работа №3. T-SQL − язык реляционной базы данных**

1. Удалить базу данных **X\_MyBASE**, созданную с помощью команд **Server Management Studio** и вновь создать с помощью языка T**-**SQL.

2. Разработать сценарии для создания в базе данных **X\_MyBASE** нужных таблиц.

Использовать ограничения целостности.

Установить связи между полями.

Просмотреть структуры таблиц с помощью команды **Проект** (**Design**) в контекстном меню таблиц.

use STORAGE

CREATE TABLE ТОВАР

(

Наименование\_товара nvarchar(50) primary key,

Цена real,

Количество int,

Описание nvarchar(50)

);

CREATE TABLE ПОКУПАТЕЛИ

(

Покупатель\_места nvarchar(50) primary key,

Телефон nvarchar(50),

Адрес nvarchar(50)

);

CREATE TABLE СДЕЛКИ

(

Номер\_сделки int primary key,

Дата\_сделки date,

Место\_хранения nvarchar(10),

Количество\_ячеек int,

Наименование\_товара nvarchar(50) foreign key references ТОВАР(Наименование\_товара),

Покупатель\_места nvarchar(50) foreign key references ПОКУПАТЕЛИ(Покупатель\_места)

);

3. Опробовать процедуру внесения изменения в структуру одной из таблиц с помощью оператора **ALTER** добавив столбец.

С использованием **ALTER** добавить некоторые ограничения целостности. Просмотреть новую структуру и удалить добавленный столбец

ALTER Table ПОКУПАТЕЛИ ADD Город\_покупателя nvarchar(15);

ALTER Table ПОКУПАТЕЛИ ADD CONSTRAINT Город DEFAULT 'Minsk' FOR Город\_покупателя;

ALTER Table ПОКУПАТЕЛИ DROP CONSTRAINT Город;

ALTER Table ПОКУПАТЕЛИ DROP Column Город\_покупателя;

4. С помощью оператора **INSERT** заполнить все таблицы информацией.

INSERT into ТОВАР (Наименование\_товара, Цена, Количество, Описание)

Values ('Ariel', 13, 7, 'Стиральный порошок'),

('Azelit', 13, 17, 'Моющая пена'),

('Mr.Proper', 9, 20, 'Моющая жидкость'),

('Persil', 12, 5, 'Стиральный порошок'),

('Sorti', 4, 25, 'Моющее средство'),

('TIDE', 14, 11, 'Стиральный порошок');

INSERT into ПОКУПАТЕЛИ (Покупатель\_места, Телефон, Адрес)

VALUES ('БытХим', 336703059, 'Свердлова, 4'),

('КОСМО', 336707055, 'Свердлова, 17'),

('Мила', 336918055, 'Свердлова, 1'),

('МойМыл', 335647895, 'Каменногорская, 10'),

('Остров Чистоты', 296542312, 'Каменногорская, 12');

INSERT into СДЕЛКИ (Номер\_сделки, Дата\_сделки, Место\_хранения, Количество\_ячеек, Наименование\_товара, Покупатель\_места)

VALUES (105, '2022.02.15', 'A', 8, 'Persil', 'Остров чистоты'),

(107, '2022.02.15', 'B', 10, 'Ariel', 'Мила'),

(109, '2022.02.16', 'С', 12, 'TIDE', 'КОСМО'),

(110, '2022.02.19', 'A', 10, 'Sorti', 'МойМыл'),

(102, '2022.02.20', 'C', 12, 'Mr.Proper', 'БытХим')

5. Вывести все строки и столбцы одной из таблиц.

Написать оператор **SELECT**, выбирающий все строки для двух столбцов таблицы.

Подсчитать количество строк в таблице.

Опробовать запросы на поиск информации с использованием **Where, Distinct,** **Top.**

SELECT \* From ПОКУПАТЕЛИ;

SELECT Покупатель\_места, Телефон From ПОКУПАТЕЛИ;

SELECT count(\*) From ПОКУПАТЕЛИ;

SELECT Наименование\_товара[Товары с ценой меньше 10 руб.] From ТОВАР WHERE Цена<10;

SELECT Top(3) Покупатель\_места, Телефон From ПОКУПАТЕЛИ;

SELECT DISTINCT Цена FROM ТОВАР;

6. Разработать запросы на поиск информации с использованием различных предикатов.

SELECT Номер\_сделки, Наименование\_товара, Покупатель\_места from СДЕЛКИ Where Дата\_сделки Between '2022.02.13' And '2022.02.17'

SELECT Наименование\_товара from ТОВАР where Наименование\_товара Like 'A%'

SELECT Наименование\_товара, Количество from ТОВАР where Количество IN(20,25)

7. Внести изменения в сценарий создания базы данных **X\_MyBASE** с тем, чтобы файлы размещались в определенных местах памяти.

use master;

CREATE database Koss\_lab3 on primary

( name = N'Koss\_lab3\_mdf', filename = N'E:\БД\_лаба3\Koss\_lab3\_mdf.mdf',

size = 10240Kb, maxsize=UNLIMITED, filegrowth = 1024Kb),

( name = N'Koss\_lab3\_ndf', filename = N'E:\БД\_лаба3\Koss\_lab3\_ndf.ndf',

size = 10240Kb, maxsize=1Gb, filegrowth = 25%),

filegroup FG1

( name = N'Koss\_lab3\_fg1\_1', filename = N'E:\БД\_лаба3\Koss\_lab3\_fgq-1.ndf',

size = 10240Kb, maxsize=1Gb, filegrowth = 25%),

( name = N'Koss\_lab3\_fg1\_2', filename = N'E:\БД\_лаба3\Koss\_lab3\_fgq-2.ndf',

size = 10240Kb, maxsize=1Gb, filegrowth = 25%)

log on

( name = N'Koss\_lab3\_log', filename = N'E:\БД\_лаба3\Koss\_lab3\_log.ldf',

size = 10240Kb, maxsize=2048Gb, filegrowth = 10%)

8. Разместить таблицы базы данных **X\_MyBASE** в файловых группах.

use Koss\_lab3\_STORAGE

CREATE TABLE ТОВАР

(

Наименование\_товара nvarchar(50) primary key,

Цена real,

Количество int,

Описание nvarchar(50)

) on FG1;

CREATE TABLE ПОКУПАТЕЛИ

(

Покупатель\_места nvarchar(50) primary key,

Телефон nvarchar(50),

Адрес nvarchar(50)

) on FG1;

CREATE TABLE СДЕЛКИ

(

Номер\_сделки int primary key,

Дата\_сделки date,

Место\_хранения nvarchar(10),

Количество\_ячеек int,

Наименование\_товара nvarchar(50) foreign key references ТОВАР(Наименование\_товара),

Покупатель\_места nvarchar(50) foreign key references ПОКУПАТЕЛИ(Покупатель\_места)

) on FG1;

**Лабораторная работа №4. Многотабличные SELECT-запросы**

1.На основе таблиц **AUDITORIUM\_ TYPE** и **AUDITORIUM** сформировать перечень кодов аудиторий (столбец **AUDITORUM.AUDITORIUM**) и соответствующих им наименований типов аудиторий (столбец **AUDITORIUM\_ TYPE.AUDITORIUM\_ TYPENAME**). Примечание: использовать соединение таблиц INNER JOIN.

use UNIVER\_lab4

SELECT AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME, AUDITORIUM.AUDITORIUM

FROM AUDITORIUM\_TYPE Inner Join AUDITORIUM

ON AUDITORIUM\_TYPE.AUDITORIUM\_TYPE = AUDITORIUM.AUDITORIUM\_TYPE

2.На основе таблиц **AUDITORIUM\_TYPE** и **AUDITORIUM** сформировать перечень кодов аудиторий (столбец **AUDITORIUM.AUDITORIUM**) и соответствующих им наименований типов аудиторий (столбец **AUDITORIUM\_ TYPE.AUDITO-RIUM\_TYPENAME**).

При этом следует выбрать только те аудитории, в наименовании которых присутствует подстрока **компьютер**.

Примечание: использовать соединение таблиц INNER JOIN и предикат LIKE.

use UNIVER\_lab4

SELECT AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME, AUDITORIUM.AUDITORIUM

FROM AUDITORIUM\_TYPE Inner Join AUDITORIUM

ON AUDITORIUM\_TYPE.AUDITORIUM\_TYPE = AUDITORIUM.AUDITORIUM\_TYPE And AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME Like 'компьютер%'

3.Написать два SELECT-запроса, формирующих результирующие наборы аналогичные запросам из заданий 1 и 2, но без применения INNER JOIN.

use UNIVER\_lab4

SELECT AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME, AUDITORIUM.AUDITORIUM

FROM AUDITORIUM\_TYPE, AUDITORIUM

WHERE AUDITORIUM\_TYPE.AUDITORIUM\_TYPE = AUDITORIUM.AUDITORIUM\_TYPE

SELECT T1.AUDITORIUM\_TYPENAME, T2.AUDITORIUM

FROM AUDITORIUM\_TYPE AS T1, AUDITORIUM AS T2

WHERE T1.AUDITORIUM\_TYPE = T2.AUDITORIUM\_TYPE

4.На основе таблиц **PRORGESS**, **STUDENT**, **GROUPS**, **SUBJECT**, **PULPIT** и **FACULTY** сформировать перечень студентов, получивших экзаменационные оценки (столбец **PROGRESS.NOTE**) от 6 до 8.

Результирующий набор должен содержать столбцы: **Факультет**, **Кафедра**, **Специальность**, **Дисциплина**, **Имя Студента**, **Оценка**. В столбце **Оценка** должны быть записаны экзаменационные оценки прописью: **шесть**, **семь**, **восемь**.

Результирующий набор отсортировать в порядке возрастания по столбцам **FACULTY.FACULTY**, **PULPIT.PULPIT**, **PROFESSION.PROFESSION**, **STUDENT. STUDENT\_NAME** и в порядке убывания по столбцу **PROGRESS.NOTE**.

Примечание: использовать соединение INNER JOIN, предикат BETWEEN и выражение CASE.

use UNIVER\_lab4

SELECT FACULTY.FACULTY,PULPIT.PULPIT,GROUPS.PROFESSION,SUBJECT.SUBJECT,STUDENT.NAME,PROGRESS.NOTE,

Case

When (PROGRESS.NOTE = 8) THEN 'Восемь'

When (PROGRESS.NOTE = 7) THEN 'Семь'

When (PROGRESS.NOTE = 6) THEN 'Шесть'

end Оценка

FROM ((((((PROGRESS INNER JOIN STUDENT

ON STUDENT.IDSTUDENT = PROGRESS.IDSTUDENT)

INNER JOIN GROUPS ON STUDENT.IDGROUP = GROUPS.IDGROUP)

INNER JOIN PROFESSION ON PROFESSION.PROFESSION = GROUPS.PROFESSION)

INNER JOIN FACULTY ON FACULTY.FACULTY = PROFESSION.FACULTY)

INNER JOIN PULPIT ON FACULTY.FACULTY = PULPIT.FACULTY)

INNER JOIN SUBJECT ON PULPIT.PULPIT = SUBJECT.PULPIT)

WHERE (PROGRESS.NOTE BETWEEN 6 AND 8)

ORDER BY FACULTY.FACULTY, PULPIT.PULPIT,STUDENT.NAME,GROUPS.PROFESSION ASC,PROGRESS.NOTE DESC

5.Переписать запрос, реализующий задание 4 таким образом, чтобы в результирующем наборе сортировка по экзаменационным оценкам была следующей: сначала выводились строки с оценкой **7**, затем строки с оценкой **8** и далее строки с оценкой **6**.

Примечание: использовать выражение CASE в секции ORDER BY.

use UNIVER\_lab4

SELECT FACULTY.FACULTY,PULPIT.PULPIT,GROUPS.PROFESSION,SUBJECT.SUBJECT,STUDENT.NAME,PROGRESS.NOTE,

Case

When (PROGRESS.NOTE = 8) THEN 'Восемь'

When (PROGRESS.NOTE = 7) THEN 'Семь'

When (PROGRESS.NOTE = 6) THEN 'Шесть'

end Оценка

FROM ((((((PROGRESS INNER JOIN STUDENT

ON STUDENT.IDSTUDENT = PROGRESS.IDSTUDENT)

INNER JOIN GROUPS ON STUDENT.IDGROUP = GROUPS.IDGROUP)

INNER JOIN PROFESSION ON PROFESSION.PROFESSION = GROUPS.PROFESSION)

INNER JOIN FACULTY ON FACULTY.FACULTY = PROFESSION.FACULTY)

INNER JOIN PULPIT ON FACULTY.FACULTY = PULPIT.FACULTY)

INNER JOIN SUBJECT ON PULPIT.PULPIT = SUBJECT.PULPIT)

WHERE (PROGRESS.NOTE BETWEEN 6 AND 8)

ORDER BY

(Case

when(PROGRESS.NOTE=6) then 3

when(PROGRESS.NOTE=7) then 1

when(PROGRESS.NOTE=8) then 2

end

)

6.На основе таблиц **PULPIT** и **TEACHER** получить полный перечень кафедр (столбец **PULPIT.PULPIT\_ NAME)** и преподавателей (столбец **TEACHER.TEA-CHER\_NAME**) на этих кафедрах. Результирующий набор должен содержать два столбца: **Кафедра** и **Преподаватель**. Если на кафедре нет преподавателей, то в столбце **Преподаватель** должна быть выведена строка **\*\*\*.**

Примечание: использовать соединение таблиц LEFT OUTER JOIN и функцию **isnull**.

use UNIVER\_lab4

SELECT PULPIT.PULPIT\_NAME, isnull(TEACHER.TEACHER\_NAME,'\*\*\*')[TEACHER\_NAME]

FROM PULPIT Left Outer JOIN TEACHER

ON PULPIT.PULPIT=TEACHER.PULPIT

**7.** В запросе, реализующем пункт 6, поменять порядок таблиц в выражении LEFT OUTER JOIN. Объяснить: почему в столбце **Кафедра** не может быть значения NULL.

Переписать запрос таким образом, чтобы получился аналогичный результат, но применялось соединение таблиц RIGHT OUTER JOIN.

use UNIVER\_lab4

SELECT PULPIT.PULPIT\_NAME, isnull(TEACHER.TEACHER\_NAME,'\*\*\*')[TEACHER\_NAME]

FROM TEACHER Left Outer JOIN PULPIT

ON PULPIT.PULPIT=TEACHER.PULPIT

use UNIVER\_lab4

SELECT PULPIT.PULPIT\_NAME, isnull(TEACHER.TEACHER\_NAME,'\*\*\*')[TEACHER\_NAME]

FROM PULPIT Right Outer JOIN TEACHER

ON PULPIT.PULPIT=TEACHER.PULPIT

8.Показать на примере, что соединение FULL OUTER JOIN двух таблиц:

− является коммутативной операцией;

− является объединением LEFT OUTER JOIN и RIGHT OUTER JOIN соединений этих таблиц;

− включает соединение INNER JOIN этих таблиц.

Примечание: создать две таблицы, заполнить их данными. Разработать SELECT-запросы, реализующие задания.

Создать три новых запроса:

− запрос, результат которого содержит данные левой (в операции FULL OUTER JOIN) таблицы и не содержит данные правой;

− запрос, результат которого содержит данные правой таблицы и не содержащие данные левой;

− запрос, результат которого содержит данные правой таблицы и левой таблиц;

Примечание: использовать в запросах выражение IS NULL и IS NOT NULL.

use UNIVER\_lab4

--Коммутативность OUTER JOIN: результирующий набор не зависит от порядка, в котором указаны исходные таблицы.

SELECT \*

FROM PULPIT FULL OUTER JOIN TEACHER

ON PULPIT.PULPIT = TEACHER.PULPIT

WHERE TEACHER\_NAME IS NULL

SELECT \*

FROM TEACHER FULL OUTER JOIN PULPIT

ON PULPIT.PULPIT = TEACHER.PULPIT

WHERE TEACHER\_NAME IS NULL

--1 содержит данные левой (в операции FULL OUTER JOIN) и не содержит данные правой

SELECT \*

FROM PULPIT FULL OUTER JOIN TEACHER

ON PULPIT.PULPIT = TEACHER.PULPIT

WHERE TEACHER\_NAME IS NULL

ORDER BY PULPIT\_NAME,FACULTY

--2 содержит данные правой таблицы и не содержащие данные левой

SELECT PULPIT.PULPIT,PULPIT.PULPIT\_NAME,PULPIT.FACULTY

FROM TEACHER RIGHT OUTER JOIN PULPIT

ON PULPIT.PULPIT = TEACHER.PULPIT

WHERE TEACHER\_NAME IS NULL

--3 содержит данные правой таблицы и левой

SELECT PULPIT.PULPIT,PULPIT.PULPIT\_NAME,PULPIT.FACULTY, TEACHER.TEACHER\_NAME

FROM PULPIT FULL OUTER JOIN TEACHER

ON PULPIT.PULPIT = TEACHER.PULPIT

WHERE TEACHER\_NAME IS NOT NULL

ORDER BY PULPIT\_NAME,FACULTY

9.Разработать SELECT-запрос на основе CROSS JOIN-соединения таблиц **AUDITORIUM\_TYPE** и **AUDITORIUM**, формирующего результат, аналогичный результату, полученному при выполнении запроса в задании 1.

use UNIVER\_lab4

SELECT AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME, AUDITORIUM.AUDITORIUM

FROM AUDITORIUM\_TYPE Cross Join AUDITORIUM

where AUDITORIUM\_TYPE.AUDITORIUM\_TYPE = AUDITORIUM.AUDITORIUM\_TYPE

--запрос 1 inner join

--use UNIVER\_lab4

--SELECT AUDITORIUM\_TYPE.AUDITORIUM\_TYPENAME, AUDITORIUM.AUDITORIUM

--FROM AUDITORIUM\_TYPE Inner Join AUDITORIUM

--ON AUDITORIUM\_TYPE.AUDITORIUM\_TYPE = AUDITORIUM.AUDITORIUM\_TYPE

**Лабораторная работа №5. Использование подзапросов**

**1.** На основе таблиц **FACULTY**, **PULPIT** и **PROFESSION** сформировать список наименований кафедр (столбец **PULPIT\_NAME**), которые находятся на факультете (таблица **FACULTY**), обеспечивающем подготовку по специальности, в наименовании (столбец **PROFESSION\_ NAME**) которого содержится слово ***технология*** или ***технологии***. Примечание: использовать в секции WHERE предикат IN c некоррелированным подзапросом к таблице **PROFESSION**.

use UNIVER\_lab4

SELECT FACULTY.FACULTY\_NAME, PULPIT.PULPIT\_NAME

FROM FACULTY, PULPIT

WHERE PULPIT.FACULTY=FACULTY.FACULTY

AND PULPIT.FACULTY IN (SELECT PROFESSION.FACULTY FROM PROFESSION

WHERE (PROFESSION\_NAME LIKE '%технологи%'))

SELECT PROFESSION.FACULTY, PROFESSION\_NAME FROM PROFESSION

WHERE (PROFESSION\_NAME LIKE '%технологи%')

**2.** Переписать запрос пункта 1 таким образом, чтобы тот же подзапрос был записан в конструкции INNER JOIN секции FROM внешнего запроса. При этом результат выполнения запроса должен быть аналогичным результату исходного запроса.

use UNIVER\_lab4

SELECT FACULTY.FACULTY\_NAME, PULPIT.PULPIT\_NAME

FROM FACULTY INNER JOIN PULPIT

ON PULPIT.FACULTY=FACULTY.FACULTY

AND PULPIT.FACULTY IN (SELECT PROFESSION.FACULTY FROM PROFESSION

WHERE (PROFESSION\_NAME LIKE '%технологи%'))

SELECT PROFESSION.FACULTY, PROFESSION\_NAME FROM PROFESSION

WHERE (PROFESSION\_NAME LIKE '%технологи%')

**3.** Переписать запрос, реализующий 1 пункт без использования подзапроса. Примечание: использовать соединение INNER JOIN трех таблиц.

use UNIVER\_lab4

SELECT distinct PULPIT.PULPIT\_NAME, PROFESSION.FACULTY

FROM PROFESSION Inner Join PULPIT

ON (PROFESSION.FACULTY = PULPIT.FACULTY and PROFESSION\_NAME like '%технологи%')

Inner Join FACULTY

ON PULPIT.FACULTY=FACULTY.FACULTY

**4.** На основе таблицы **AUDITORIUM** сформировать список аудиторий самых больших вместимостей (столбец **AUDITORIUM\_CAPACITY**) для каждого типа аудитории (**AUDITORIUM\_TYPE**). При этом результат следует отсортировать в порядке убывания вместимости. Примечание: использовать коррелируемый подзапрос c секциями TOP и ORDER BY.

use UNIVER\_lab4

SELECT AUDITORIUM\_CAPACITY, AUDITORIUM\_TYPE

FROM AUDITORIUM a

where AUDITORIUM\_CAPACITY=(select top(1) AUDITORIUM\_CAPACITY

from AUDITORIUM aa

where aa.AUDITORIUM\_TYPE=a.AUDITORIUM\_TYPE

order by AUDITORIUM\_CAPACITY desc)

**5.** На основе таблиц **FACULTY** и **PULPIT** сформировать список наименований факультетов (столбец **FACULTY\_NAME**) на котором нет ни одной кафедры (таблица **PULPIT**). Примечание: использовать предикат EXISTS и коррелированный подзапрос.

USE UNIVER\_lab4

SELECT FACULTY.FACULTY\_NAME

FROM FACULTY

WHERE NOT EXISTS (SELECT\* FROM PULPIT

WHERE FACULTY.FACULTY=PULPIT.FACULTY)

**6.** На основе таблицы **PROGRESS** сформировать строку, содержащую средние значения оценок (столбец **NOTE**) по дисциплинам, имеющим следующие коды: **ОАиП**, **БД** и **СУБД**. Примечание: использовать три некоррелированных подзапроса в списке SELECT; в подзапросах применить агрегатные функции AVG.

USE UNIVER\_lab4

SELECT TOP 1 (SELECT AVG(NOTE) from PROGRESS

WHERE SUBJECT LIKE 'ОАиП')[ОАиП],

(SELECT AVG(NOTE) from PROGRESS

WHERE SUBJECT LIKE 'КГ')[КГ],

(SELECT AVG(NOTE) from PROGRESS

WHERE SUBJECT LIKE 'СУБД')[СУБД]

FROM PROGRESS

**7.** Разработать SELECT-запрос, демонстрирующий принцип применения ALL совместно с подзапросом.

USE UNIVER\_lab4

SELECT \* FROM PROGRESS

WHERE NOTE >= ALL(SELECT NOTE

FROM PROGRESS

WHERE SUBJECT='ОАиП')

8. Разработать SELECT-запрос, демонстрирующий принцип применения ANY совместно с подзапросом.

USE UNIVER\_lab4

SELECT \* FROM PROGRESS

WHERE NOTE > ANY(SELECT NOTE

FROM PROGRESS

WHERE SUBJECT='ОАиП')

**Лабораторная работа №6. Группировка данных**

1.На основе таблицы **AUDITORIUM** разработать SELECT-запрос, вычисляющий максимальную, минимальную и среднюю вместимость аудиторий, суммарную вместимость всех аудиторий и общее количество аудиторий.

use UNIVER\_lab4

SELECT min(AUDITORIUM\_CAPACITY) [min.capacity],

max(AUDITORIUM\_CAPACITY) [max.capacity],

count(\*) [aud.count],

sum(AUDITORIUM\_CAPACITY) [overall capacity],

avg(AUDITORIUM\_CAPACITY) [average capacity]

FROM AUDITORIUM

2.На основе таблиц **AUDITORIUM** и **AUDITORIUM\_TYPE** разработать запрос, вычисляющий для каждого типа аудиторий максимальную, минимальную, среднюю вместимость аудиторий, суммарную вместимость всех аудиторий и общее количество аудиторий данного типа. Результирующий набор должен содержать столбец с наименованием типа аудиторий (столбец **AUDITORIUM\_TYPE.AU-DITORIUM\_TYPENAME**) и столбцы с вычисленными величинами. Использовать внутреннее соединение таблиц, секцию GROUP BY и агрегатные функции.

use UNIVER\_lab4

SELECT AUDITORIUM\_TYPENAME,

min(AUDITORIUM\_CAPACITY) [min.capacity],

max(AUDITORIUM\_CAPACITY) [max.capacity],

count(\*) [aud. count],

sum(AUDITORIUM\_CAPACITY) [overall capacity],

avg(AUDITORIUM\_CAPACITY) [average capacity]

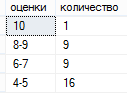
FROM AUDITORIUM INNER JOIN AUDITORIUM\_TYPE

ON AUDITORIUM.AUDITORIUM\_TYPE=AUDITORIUM\_TYPE.AUDITORIUM\_TYPE

GROUP BY AUDITORIUM\_TYPENAME

3.Разработать запрос на основе таблицы **PROGRESS**, который содержит количество экзаменационных оценок в заданном интервале. При этом учесть, что сортировка строк должна осуществляться в порядке, обратном величине оценки; сумма значений в столбце **количество** должна быть равна количеству строк в таблице **PROGRESS**.

Использовать подзапрос в секции FROM, в подзапросе применить GROUP BY, сортировку осуществить во внешнем запросе. В секции GROUP BY, в SELECT-списке подзапроса и в ORDER BY внешнего запроса применить CASE.



USE UNIVER\_lab4;

SELECT \*

FROM (SELECT CASE

WHEN NOTE between 4 and 5 then '4-5'

WHEN NOTE between 6 and 7 then '6-7'

WHEN NOTE between 8 and 9 then '8-9'

WHEN NOTE=10 then '10'

END [оценки],

Count(\*) [количество]

FROM PROGRESS GROUP BY CASE

WHEN NOTE between 4 and 5 then '4-5'

WHEN NOTE between 6 and 7 then '6-7'

WHEN NOTE between 8 and 9 then '8-9'

WHEN NOTE=10 then '10'

END) AS T

ORDER BY CASE [оценки]

WHEN '10' THEN 1

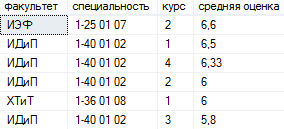
WHEN '8-9' THEN 2

WHEN '6-7' THEN 3

WHEN '4-5' THEN 4

END

4. Разработать SELECT-запроса на основе таблиц **FACULTY**, **GROUPS**, **STUDENT** и **PROGRESS**, который содержит среднюю экзаменационную оценку для каждого курса каждой специальности. Строки отсортировать в порядке убывания средней оценки.



При этом следует учесть, что средняя оценка должна рассчитываться с точностью до двух знаков после запятой. Использовать внутреннее соединение таблиц, агрегатную функцию AVG и встроенные функции CAST и ROUND.

Переписать SELECT-запрос, разработанный в задании 4 так, чтобы в расчете среднего значения оценок использовались оценки только по дисциплинам с кодами **БД** и **ОАиП**. Использовать WHERE.

SELECT FACULTY.FACULTY\_NAME [ФАКУЛЬТЕТ],

GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

STUDENT.IDGROUP [ГРУППА],

ROUND(AVG(CAST(PROGRESS.NOTE AS FLOAT(2))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM FACULTY INNER JOIN GROUPS

ON FACULTY.FACULTY=GROUPS.FACULTY

INNER JOIN STUDENT

ON GROUPS.IDGROUP=STUDENT.IDGROUP

INNER JOIN PROGRESS

ON PROGRESS.IDSTUDENT=STUDENT.IDSTUDENT

GROUP BY FACULTY.FACULTY\_NAME,

GROUPS.PROFESSION,

STUDENT.IDGROUP

ORDER BY [СРЕДНЯЯ ОЦЕНКА] Desc

SELECT FACULTY.FACULTY\_NAME [ФАКУЛЬТЕТ],

GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

STUDENT.IDGROUP [ГРУППА],

ROUND(AVG(CAST(PROGRESS.NOTE AS FLOAT(2))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM FACULTY INNER JOIN GROUPS

ON FACULTY.FACULTY=GROUPS.FACULTY

INNER JOIN STUDENT

ON GROUPS.IDGROUP=STUDENT.IDGROUP

INNER JOIN PROGRESS

ON PROGRESS.IDSTUDENT=STUDENT.IDSTUDENT

WHERE PROGRESS.SUBJECT='БД' or PROGRESS.SUBJECT='ОАиП'

GROUP BY FACULTY.FACULTY\_NAME,

GROUPS.PROFESSION,

STUDENT.IDGROUP

ORDER BY [СРЕДНЯЯ ОЦЕНКА] Desc

5. На основе таблиц **FACULTY**, **GROUPS**, **STUDENT** и **PROGRESS** разработать SELECT-запрос, в котором выводятся специальность, дисциплины и средние оценки при сдаче экзаменов на факультете ТОВ. Использовать группировку по полям FACULTY, PROFESSION, SUBJECT.

Добавить в запрос конструкцию **ROLLUP** и проанализировать результат.

SELECT GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

ROUND(AVG(CAST(PROGRESS.NOTE AS FLOAT(2))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM STUDENT INNER JOIN GROUPS

ON STUDENT.IDGROUP=GROUPS.IDGROUP

INNER JOIN PROGRESS

ON PROGRESS.IDSTUDENT=STUDENT.IDSTUDENT

WHERE GROUPS.FACULTY='ИТ'

GROUP BY GROUPS.FACULTY,

GROUPS.PROFESSION,

PROGRESS.SUBJECT

SELECT GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

ROUND(AVG(CAST(PROGRESS.NOTE AS FLOAT(2))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM STUDENT INNER JOIN GROUPS

ON STUDENT.IDGROUP=GROUPS.IDGROUP

INNER JOIN PROGRESS

ON PROGRESS.IDSTUDENT=STUDENT.IDSTUDENT

WHERE GROUPS.FACULTY='ИТ'

GROUP BY ROLLUP (

GROUPS.PROFESSION,

PROGRESS.SUBJECT)

**6.** Выполнить исходный SELECT-запрос п.5 с использованием **CUBE**-группировки. Проанализировать результат.

SELECT GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

ROUND(AVG(CAST(PROGRESS.NOTE AS FLOAT(2))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM STUDENT INNER JOIN GROUPS

ON STUDENT.IDGROUP=GROUPS.IDGROUP

INNER JOIN PROGRESS

ON PROGRESS.IDSTUDENT=STUDENT.IDSTUDENT

WHERE GROUPS.FACULTY='ИТ'

GROUP BY CUBE (

GROUPS.PROFESSION,

PROGRESS.SUBJECT)

7. На основе таблиц **GROUPS**, **STUDENT** и **PROGRESS** разработать SELECT-запрос, в котором определяются результаты сдачи экзаменов.

В запросе должны отражаться специальности, дисциплины, средние оценки студентов на факультете ТОВ.

Отдельно разработать запрос, в котором определяются результаты сдачи экзаменов на факультете ХТиТ.

Объединить результаты двух запросов с использованием операторов UNION и UNION ALL. Объяснить результаты.

SELECT GROUPS.FACULTY [ФАКУЛЬТЕТ],

GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

round(avg(cast(PROGRESS.NOTE as float(4))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM GROUPS Inner Join STUDENT

On GROUPS.IDGROUP=STUDENT.IDGROUP

Inner Join PROGRESS

On STUDENT.IDSTUDENT=PROGRESS.IDSTUDENT

Where GROUPS.FACULTY='ИТ'

Group BY GROUPS.FACULTY,

GROUPS.PROFESSION,

PROGRESS.SUBJECT

UNION

SELECT GROUPS.FACULTY [ФАКУЛЬТЕТ],

GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

round(avg(cast(PROGRESS.NOTE as float(4))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM GROUPS Inner Join STUDENT

On GROUPS.IDGROUP=STUDENT.IDGROUP

Inner Join PROGRESS

On STUDENT.IDSTUDENT=PROGRESS.IDSTUDENT

Where GROUPS.FACULTY='ХТиТ'

Group BY GROUPS.FACULTY,

GROUPS.PROFESSION,

PROGRESS.SUBJECT

SELECT GROUPS.FACULTY [ФАКУЛЬТЕТ],

GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

round(avg(cast(PROGRESS.NOTE as float(4))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM GROUPS Inner Join STUDENT

On GROUPS.IDGROUP=STUDENT.IDGROUP

Inner Join PROGRESS

On STUDENT.IDSTUDENT=PROGRESS.IDSTUDENT

Where GROUPS.FACULTY='ИТ'

Group BY GROUPS.FACULTY,

GROUPS.PROFESSION,

PROGRESS.SUBJECT

UNION ALL

SELECT GROUPS.FACULTY [ФАКУЛЬТЕТ],

GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

round(avg(cast(PROGRESS.NOTE as float(4))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM GROUPS Inner Join STUDENT

On GROUPS.IDGROUP=STUDENT.IDGROUP

Inner Join PROGRESS

On STUDENT.IDSTUDENT=PROGRESS.IDSTUDENT

Where GROUPS.FACULTY='ХТиТ'

Group BY GROUPS.FACULTY,

GROUPS.PROFESSION,

PROGRESS.SUBJECT

8.Получить пересечение двух множеств строк, созданных в результате выполнения запросов пункта 8. Объяснить результат.

Использовать оператор INTERSECT.

SELECT GROUPS.FACULTY [ФАКУЛЬТЕТ],

GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

round(avg(cast(PROGRESS.NOTE as float(4))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM GROUPS Inner Join STUDENT

On GROUPS.IDGROUP=STUDENT.IDGROUP

Inner Join PROGRESS

On STUDENT.IDSTUDENT=PROGRESS.IDSTUDENT

Where GROUPS.FACULTY='ИТ'

Group BY GROUPS.FACULTY,

GROUPS.PROFESSION,

PROGRESS.SUBJECT

INTERSECT

SELECT GROUPS.FACULTY [ФАКУЛЬТЕТ],

GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

round(avg(cast(PROGRESS.NOTE as float(4))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM GROUPS Inner Join STUDENT

On GROUPS.IDGROUP=STUDENT.IDGROUP

Inner Join PROGRESS

On STUDENT.IDSTUDENT=PROGRESS.IDSTUDENT

Where GROUPS.FACULTY='ХТиТ'

Group BY GROUPS.FACULTY,

GROUPS.PROFESSION,

PROGRESS.SUBJECT

9.Получить разницу между множеством строк, созданных в результате запросов пункта 8.

Объяснить результат.

Использовать оператор EXCEPT.

SELECT GROUPS.FACULTY [ФАКУЛЬТЕТ],

GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

round(avg(cast(PROGRESS.NOTE as float(4))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM GROUPS Inner Join STUDENT

On GROUPS.IDGROUP=STUDENT.IDGROUP

Inner Join PROGRESS

On STUDENT.IDSTUDENT=PROGRESS.IDSTUDENT

Where GROUPS.FACULTY='ИТ'

Group BY GROUPS.FACULTY,

GROUPS.PROFESSION,

PROGRESS.SUBJECT

EXCEPT

SELECT GROUPS.FACULTY [ФАКУЛЬТЕТ],

GROUPS.PROFESSION [СПЕЦИАЛЬНОСТЬ],

PROGRESS.SUBJECT [ДИСЦИПЛИНА],

round(avg(cast(PROGRESS.NOTE as float(4))), 2) [СРЕДНЯЯ ОЦЕНКА]

FROM GROUPS Inner Join STUDENT

On GROUPS.IDGROUP=STUDENT.IDGROUP

Inner Join PROGRESS

On STUDENT.IDSTUDENT=PROGRESS.IDSTUDENT

Where GROUPS.FACULTY='ХТиТ'

Group BY GROUPS.FACULTY,

GROUPS.PROFESSION,

PROGRESS.SUBJECT

10.На основе таблицы **PROGRESS** определить для каждой дисциплины количество студентов, получивших оценки 8 и 9.

Использовать группировку, секцию HAVING, сортировку.

SELECT p1.SUBJECT [ДИСЦИПЛИНА],

p1.NOTE [ОЦЕНКА],

(SELECT COUNT(\*) FROM PROGRESS p2 where p1.SUBJECT = p2.SUBJECT

and p2.NOTE=p1.NOTE) [КОЛ-ВО СТУДЕНТОВ]

FROM PROGRESS p1

GROUP BY p1.NOTE, p1.SUBJECT, p1.IDSTUDENT

HAVING NOTE > 7 and NOTE < 10

**Лабораторная работа №7. Использование представлений**

1.Разработать представление с именем **Преподаватель**. Представление должно быть построено на основе SELECT-запроса к таблице **TEACHER** и содержать следующие столбцы: **код** (**TEACHER**), **имя преподавателя** (**TEACHER\_NAME**), **пол** (**GENDER**), **код кафедры** (**PULPIT**).

create view [Препод]

as select TEACHER[Код],

TEACHER\_NAME[Имя преподавателя],

GENDER[Пол],

PULPIT[Код кафедры] from TEACHER

2.Разработать и создать представление с именем **Количество кафедр**. Представление должно быть построено на основе SELECT-запроса к таблицам **FACULTY** и **PULPIT**.

Представление должно содержать следующие столбцы: **факультет** (**FACULTY.FACULTY\_ NAME**), **количество кафедр** (вычисляется на основе строк таблицы **PULPIT**).

create view [Количество кафедр]

as select FACULTY\_NAME[Факультет],

COUNT (\*) [Количество кафедр]

from FACULTY, PULPIT WHERE

FACULTY.FACULTY=PULPIT.FACULTY

GROUP BY FACULTY\_NAME

3.Разработать и создать представление с именем **Аудитории**. Представление должно быть построено на основе таблицы **AUDITORIUM** и содержать столбцы: **код** (**AUDITORIUM**), **наименование аудитории** (**AUDITORIUM\_NAME**)**.**

Представление должно отображать только лекционные аудитории (в столбце **AUDITORIUM\_ TYPE** строка, начинающаяся с символа **ЛК**) и допускать выполнение оператора INSERT, UPDATE и DELETE.

create view Аудитории(код, наименовани\_аудитории, тип\_аудитории)

as select AUDITORIUM,AUDITORIUM\_NAME, AUDITORIUM\_TYPE from AUDITORIUM

WHERE AUDITORIUM\_TYPE like 'ЛК%' with check option

go

select \*from Аудитории

insert Аудитории values(307-7, 307-7, 'ЛК')

4.Разработать и создать представление с именем **Лекционные\_аудитории**.

Представление должно быть построено на основе SELECT-запроса к таблице **AUDITORIUM** и содержать следующие столбцы: **код** (**AUDITORIUM**), **наименование аудитории** (**AUDITORIUM\_NAME**)**.**

Представление должно отображать только лекционные аудитории (в столбце **AUDITORIUM\_TYPE** строка, начинающаяся с символов **ЛК**).

Выполнение INSERT и UPDATE допускается, но с учетом ограничения, задаваемого опцией WITH CHECK OPTION.

create view [Дисциплины](SUBJECT, SUBJECT\_NAME,PULPIT)

as select top 150 SUBJECT\_NAME,SUBJECT,PULPIT FROM SUBJECT

ORDER BY SUBJECT\_NAME;

5.Разработать представление с именем **Дисциплины**. Представление должно быть построено на основе SELECT-запроса к таблице **SUBJECT**, отображать все дисциплины в алфавитном порядкеи содержать следующие столбцы: **код** (**SUBJECT**), **наименование дисциплины** (**SUBJECT\_NAME**) и **код кафедры** (**PULPIT**). Использовать TOP и ORDER BY.

create view [Количество кафедр\_new] with schemabinding

as select FACULTY\_NAME[Факультет],

COUNT (\*) [Количество кафедр]

FROM dbo.FACULTY join dbo.PULPIT on

FACULTY.FACULTY = PULPIT.FACULTY

GROUP BY FACULTY\_NAME;

**Лабораторная работа №8. Основы программирования на T-SQL**

1. Разработать T-SQL-скрипт, в котором:

− объявить переменные типа char, varchar, datetime, time, int, smallint, tinint, numeric(12, 5);

− первые две переменные проинициализировать в операторе объявления;

− присвоить произвольные значения следующим двум переменным с помощью оператора SET, одной из этих переменных присвоить значение, полученное в результате запроса SELECT;

− одну из переменных оставить без инициализации и не присваивать ей значения, оставшимся переменным присвоить некоторые значения с помощью оператора SELECT;

− значения одной половины переменных вывести с помощью оператора SELECT,значения другой половины переменных распечатать с помощью оператора PRINT.

Проанализировать результаты.

DECLARE @a char='A',

@b varchar(5)='ДЭиВИ',

@c datetime,

@d time(2),

@e int,

@f smallint,

@g tinyint,

@h numeric(12,5);

SET @c=getdate();

SELECT @d='15:13:12.123456';

SELECT @e=10, @g=0, @h=123.45678;--@f не определена

SELECT @a a, @b b, @c c, @d d;

PRINT 'e='+ cast(@e as varchar(2));--в messages

PRINT 'f='+cast(@f as varchar(2));

PRINT 'g='+cast(@g as varchar(1));

PRINT 'h='+cast(@h as varchar(10));

2. Разработать скрипт, в котором определяется общая вместимость аудиторий. Когда общая вместимость превышает 200, то вывести количество аудиторий, среднюю вместимость аудиторий, количество аудиторий, вместимость которых меньше средней, и процент таких аудиторий. Когда общая вместимость аудиторий меньше 200, то вывести сообщение о размере общей вместимости.

use UNIVER\_lab4;

DECLARE @overall\_capacity int=(select cast(sum(AUDITORIUM.AUDITORIUM\_CAPACITY) as int) from AUDITORIUM),

@overall\_count int,

@average\_capacity float,

@average\_less\_capacity float,

@percentage float;

IF @overall\_capacity>200

begin

SELECT @overall\_count=(select cast(count(\*) as int) from AUDITORIUM),

@average\_capacity=(select cast(avg(AUDITORIUM.AUDITORIUM\_CAPACITY) as int) from AUDITORIUM);

SET @average\_less\_capacity=(select cast(count(\*) as int) from AUDITORIUM where AUDITORIUM.AUDITORIUM\_CAPACITY<@average\_capacity);

SET @percentage=@average\_less\_capacity/@average\_capacity \*100;

Select @overall\_capacity 'общая вместимость аудиторий', @average\_capacity 'средняя вместимость аудиторий',

@overall\_count 'общее количество аудиторий', @average\_less\_capacity 'кол-во аудиторий с вместимость меньше средней',

@percentage 'процент таких аудиторий'

end

3. Разработать T-SQL-скрипт, который выводит на печать глобальные переменные:

− @@ROWCOUNT (число обработанных строк);

− @@VERSION (версия SQL Server);

− @@SPID (возвращает системный идентификатор процесса, назначенный сервером текущему подключению);

− @@ERROR (код последней ошибки);

− @@SERVERNAME (имя сервера);

− @@TRANCOUNT (возвращает уровень вложенности транзакции);

− @@FETCH\_STATUS (проверка результата считывания строк результирующего набора);

− @@NESTLEVEL (уровень вложенности текущей процедуры).

Проанализировать результат.

use UNIVER\_lab4;

print '@@ROWCOUNT(число обработанных строк) : '+ cast(@@ROWCOUNT as varchar(12));

print '@@VERSION(версия SQL Server) : '+ cast(@@VERSION as varchar(12));

print '@@SPID (идентификатор процесса, назначенный сервером текущему подключению) : '+ cast(@@SPID as varchar(12));

print '@@ERROR (код последней ошибки) : '+ cast(@@ERROR as varchar(12));

print '@@SERVERNAME (имя сервера) : '+ cast(@@SERVERNAME as varchar(12));

print '@@TRANCOUNT (возвращает уровень вложенности транзакции) : '+ cast(@@TRANCOUNT as varchar(12));

print '@@FETCH\_STATUS (проверка результата считывания строк результирующего набора) : '+ cast(@@FETCH\_STATUS as varchar(12));

print '@@NESTLEVEL(уровень вложенности текущей процедуры) : '+ cast(@@NESTLEVEL as varchar(12));

4. Разработать T-SQL-скрипты, выполняющие:

− вычисление значений переменной **z**



для различных значений исходных данных;

− преобразование полного ФИО студента в сокращенное (например, Макейчик Татьяна Леонидовна в Макейчик Т. Л.);

− поиск студентов, у которых день рождения в следующем месяце, и определение их возраста;

− поиск дня недели, в который студенты некоторой группы сдавали экзамен по СУБД.

DECLARE @x int=7, @z float, @t int=5;

IF (@t>@x) SET @z=SIN(@t)\*SIN(@t)

ELSE IF (@t<@x) SET @z=4\*(@t+@x)

ELSE IF (@t=@x) SET @z=1-EXP(@x-2)

PRINT 'z=' + cast(@z as varchar(5));

-------------------------------------

DECLARE @fio varchar(70) = 'Косс Валерия Александровна'

print substring('Косс Валерия Александровна',1,4) + ' '

+ substring('Косс Валерия Александровна',6,1)+'.'

+ substring('Косс Валерия Александровна',14,1) + '.'

--print substring(@fio,1, charindex(' ',@fio))

--+ substring(@fio, charindex(' ',@fio)+1, 1)+'.'

--+ substring(@fio, charindex(' ',@fio, charindex(' ',@fio)+1) +1, 1)+'.';

-------------------------------------

SELECT NAME as 'Имя', 2022-YEAR(BDAY) as 'Возраст'

FROM STUDENT

WHERE MONTH(BDAY) = MONTH(GETDATE()) + 1;

-------------------------------------

SELECT STUDENT.NAME [Имя студента], STUDENT.IDGROUP [Номер группы], DATENAME(dw,PDATE) [День сдачи экзамена]

FROM PROGRESS INNER JOIN STUDENT

ON PROGRESS.IDSTUDENT = STUDENT.IDSTUDENT

WHERE SUBJECT = 'СУБД' and STUDENT.IDGROUP = 5

5. Продемонстрировать конструкцию IF… ELSE на примере анализа данных таблиц базы данных **Х\_UNIVER**.

USE UNIVER\_lab4;

DECLARE @auditorium\_types\_count int=(SELECT COUNT(\*) FROM AUDITORIUM\_TYPE);

IF (SELECT COUNT(\*) FROM AUDITORIUM\_TYPE) > 4

BEGIN

PRINT 'Количество видов аудиторий больше 4';

PRINT 'Количество='+ cast(@auditorium\_types\_count as varchar(10));

end;

ELSE

BEGIN

PRINT 'Количество видов аудиторий меньше 4';

PRINT 'Количество='+ cast(@auditorium\_types\_count as varchar(10));

end;

6. Разработать сценарий, в котором с помощью CASE анализируются оценки, полученные студентами некоторого факультета при сдаче экзаменов.

SELECT CASE

when NOTE between 0 and 3 then 'В топку'

when NOTE between 4 and 7 then 'Пойдет'

else 'Годно'

end Оценки, count(\*) [Количество оценок]

from PROGRESS

GROUP BY CASE

when NOTE between 0 and 3 then 'В топку'

when NOTE between 4 and 7 then 'Пойдет'

else 'Годно'

end

7. Создать временную локальную таблицу из трех столбцов и 10 строк, заполнить ее и вывести содержимое. Использовать оператор WHILE.

CREATE table #DOG

(

HEAD int,

BODY varchar(200),

TAIL int

);

SET nocount on; --не выводить сообщения о вводе строк

DECLARE @i int=0;

WHILE @i<10

begin

INSERT #DOG(HEAD, BODY, TAIL)

values(floor(15\*rand()), replicate('орган ', 10), floor(16\*rand()));

IF (@i%10=0)

print @i; --вывести сообщение

SET @i=@i+1;

end;

SELECT \* from #DOG;

DROP table #DOG;

8. Разработать скрипт, демонстрирующий использование оператора RETURN.

DECLARE @task int = 2

print @task\*4

print @task+2

return

print @task+1

9. Разработать сценарий с ошибками, в котором используются для обработки ошибок блоки TRY и CATCH. Применить функции ERROR\_NUMBER (код последней ошибки), ERROR\_ES-SAGE (сообщение об ошибке), ERROR\_LINE (код последней ошибки), ERROR\_PROCEDURE (имя процедуры или NULL), ERROR\_SEVERITY (уровень серьезности ошибки), ERROR\_ STATE (метка ошибки). Проанализировать результат.

begin try

DECLARE @delim int=333, @delit int=0, @chastn float;

set @chastn = @delim/@delit;

end try

begin catch

print ERROR\_NUMBER()--код последней ошибки

print ERROR\_MESSAGE()--сообщение об ошибк

print ERROR\_LINE()--код последней ошибки

print ERROR\_PROCEDURE()--имя процедуры или NULL

print ERROR\_SEVERITY()--уровень серьезности ошибки

print ERROR\_STATE()--метка ошибки

end catch

**Лабораторная работа №9. Создание и применение индексов**

1.С помощью SSMS определить все индексы, которые имеются в БД **UNIVER**. Определить, какие из них являются кластеризованными, а какие некластеризованными.

Создать временную локальную таблицу. Заполнить ее данными (не менее 1000 строк).

Разработать SELECT-запрос. Получить план запроса и определить его стоимость.

Создать кластеризованный индекс, уменьшающий стоимость SELECT-запроса.

use UNIVER\_lab4;

exec SP\_HELPINDEX 'AUDITORIUM\_TYPE' --получаем перечень индексов, связ. с табл.

exec SP\_HELPINDEX 'AUDITORIUM'

exec SP\_HELPINDEX 'FACULTY'

exec SP\_HELPINDEX 'GROUPS'

exec SP\_HELPINDEX 'PROFESSION'

exec SP\_HELPINDEX 'PROGRESS'

exec SP\_HELPINDEX 'PULPIT'

exec SP\_HELPINDEX 'STUDENT'

exec SP\_HELPINDEX 'SUBJECT'

exec SP\_HELPINDEX 'TEACHER'

--создать временную локальную таблицу с 1000 строками

CREATE table #TempTableTask1

(

ind int,

field varchar(100)

);

SET nocount on; --не выводить сообщения о вводе строк

DECLARE @i int=0;

WHILE @i<1000

begin

INSERT #TempTableTask1(ind, field)

values (floor(20000\*RAND()), REPLICATE('орган ', 4));

IF (@i % 100=0) print @i;

SET @i=@i+1;

end;

--план запроса и его стоимость

SELECT \* FROM #TempTableTask1 where ind between 300 and 800 order by ind;

--чтобы объективно оценить время

checkpoint; --фиксация БД; позволяет записать образы страниц из буферного кэша в файлы БД

DBCC DROPCLEANBUFFERS; --очистить буферный кэш

--уменьшение стоимости=времени выполнения, а после сканирование не по всей таблице, а по части кластеризованного индекса

CREATE clustered index #TempTableCl on #TempTableTask1(ind asc)

2.Создать временную локальную таблицу. Заполнить ее данными (10000 строк или больше).

Разработать SELECT-запрос. Получить план запроса и определить его стоимость.

Создать *некластеризованный* неуникальный *составной* индекс.

Оценить процедуры поиска информации.

CREATE table #TempTableTask2

( ind int,

cc int identity(1, 1),

field varchar(100)

);

set nocount on;

declare @m int=0;

while @m < 20000 -- добавление в таблицу 20000 строк

begin

INSERT #TempTableTask2(ind, field) values(floor(30000\*RAND()), replicate('орган ', 4));

set @m = @m + 1;

end;

SELECT count(\*)[количество строк] from #TempTableTask2;

SELECT \* from #TempTableTask2

--составной неуникальный, некластеризованный индекс по двум столбцам

CREATE index #TempTableTask2NonCl on #TempTableTask2(ind, cc)

--планы запросов и их стоимость(0.011),

--Этот индекс не применяется оптимизатором ни при фильтрации, ни при сортировке

SELECT \* from #TempTableTask2 where ind > 1500 and cc < 4500;

SELECT \* from #TempTableTask2 order by ind, cc;

--если хотя бы одно из индексируемых значений зафиксировать, то

--оптимизатор применит индекс(0.033)

SELECT \* from #TempTableTask2 where ind = 26236 and cc > 3;

--drop index #TempTableTask2NonCl on #TempTableTask2

3.Создать временную локальную таблицу. Заполнить ее данными (не менее 10000 строк).

Разработать SELECT-запрос. Получить план запроса и определить его стоимость.

Создать *некластеризованный индекс покрытия*, уменьшающий стоимость SELECT-запроса.

CREATE table #TempTableTask3

(

ind int,

cc int identity(1,1),

field varchar(100)

);

set nocount on;

declare @n int=0;

while @n < 10000 -- добавление в таблицу 10000 строк

begin

INSERT #TempTableTask3(ind, field) values(floor(30000\*RAND()), replicate('орган ', 3));

set @n = @n + 1;

end;

--планы запроса и его стоимость(0.05)

SELECT CC from #TempTableTask3 where ind > 15000

--некластеризованный индекс покрытия, уменьш. стоимость

CREATE index #TempTableTask3NonCl on #TempTableTask3(ind) INCLUDE (cc);

--планы запроса и его стоимость(0.019)

SELECT CC from #TempTableTask3 where ind > 15000

4.Создать и заполнить временную локальную таблицу.

Разработать SELECT-запрос, получить план запроса и определить его стоимость.

Создать *некластеризованный фильтруемый индекс*, уменьшающий стоимость SELECT-запроса.

create table #TempTableTask4

(

ind int,

cc int identity(1,1),

field varchar(100)

);

set nocount on;

declare @j int=0;

while @j < 20000 -- добавление в таблицу 20000 строк

begin

INSERT #TempTableTask4(ind, field) values(floor(30000\*RAND()), replicate('орган ', 3));

set @j = @j + 1;

end;

SELECT ind from #TempTableTask4 where ind between 5000 and 19999;

SELECT ind from #TempTableTask4 where ind > 15000 and ind < 20000

SELECT ind from #TempTableTask4 where ind = 17000;

--некластеризованный фильтруемый индекс, стоимость уменьшится

CREATE index #TempTableTask4Cl on #TempTableTask4(ind) where (ind>=15000 and ind < 20000);

SELECT ind from #TempTableTask4 where ind between 5000 and 19999;

SELECT ind from #TempTableTask4 where ind > 15000 and ind < 20000

SELECT ind from #TempTableTask4 where ind = 17000;

--drop table #TempTableTask4

5.Заполнить временную локальную таблицу.

Создать некластеризованный индекс. Оценить уровень *фрагментации индекса*.

Разработать сценарий на T-SQL, выполнение которого приводит к уровню фрагментации индекса выше 90%. Оценить уровень фрагментации индекса.

Выполнить процедуру *реорганизации* индекса, оценить уровень фрагментации.

Выполнить процедуру *перестройки* индекса и оценить уровень фрагментации индекса.

--оценить уровень фрагментации

CREATE index #TempTableTask4Ind ON #TempTableTask4(ind);

--фрагменатация 37,5

use tempdb;

SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)]

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(N'TEMPDB'),

OBJECT\_ID(N'#TempTableTask4'), NULL, NULL, NULL) ss JOIN sys.indexes ii

on ss.object\_id = ii.object\_id and ss.index\_id = ii.index\_id

WHERE name is not null;

--после втсавки 93,3-97.7

INSERT top(10000) #TempTableTask4(ind, field) select ind, field from #TempTableTask4;

SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)]

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(N'TEMPDB'),

OBJECT\_ID(N'#TempTableTask4'), NULL, NULL, NULL) ss JOIN sys.indexes ii

on ss.object\_id = ii.object\_id and ss.index\_id = ii.index\_id

WHERE name is not null;

--РЕОРГАНИЗАЦИЯ

--снижается уровень фрагментации(3 число 16,4)

ALTER index #TempTableTask4Ind on #TempTableTask4 reorganize;

SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)]

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(N'TEMPDB'),

OBJECT\_ID(N'#TempTableTask4'), NULL, NULL, NULL) ss JOIN sys.indexes ii

on ss.object\_id = ii.object\_id and ss.index\_id = ii.index\_id

WHERE name is not null;

--ПЕРЕСТРОЙКА

--убирает полностью(3 число 1.49)

ALTER index #TempTableTask4Ind on #TempTableTask4 rebuild with (online = off);

SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)]

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(N'TEMPDB'),

OBJECT\_ID(N'#TempTableTask4'), NULL, NULL, NULL) ss JOIN sys.indexes ii

on ss.object\_id = ii.object\_id and ss.index\_id = ii.index\_id

WHERE name is not null;

6.Разработать пример, демонстрирующий применение параметра FILLFACTOR при создании некластеризованного индекса.

DROP index #TempTableTask4Ind on #TempTableTask4;

--fillfactor процент заполнения индексных страниц нижнего уровня

CREATE index #TempTableTask4Ind on #TempTableTask4(ind) with (fillfactor = 65);

use tempdb;

SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)]

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(N'TEMPDB'),

OBJECT\_ID(N'#TempTableTask4'), NULL, NULL, NULL) ss JOIN sys.indexes ii

ON ss.object\_id = ii.object\_id and ss.index\_id = ii.index\_id

WHERE name is not null;

--INSERT top(50)percent INTO #TempTableTask4(ind, field)

INSERT top(50) INTO #TempTableTask4(ind, field)

SELECT ind, field FROM #TempTableTask4;

use tempdb;

SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)]

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(N'TEMPDB'),

OBJECT\_ID(N'#TempTableTask4'), NULL, NULL, NULL) ss JOIN sys.indexes ii

ON ss.object\_id = ii.object\_id and ss.index\_id = ii.index\_id

WHERE name is not null;

**Лабораторная работа №10. Обработка результатов запросов с помощью курсоров**

1.Разработать сценарий, формирующий список дисциплин на кафедре ИСиТ. В отчет должны быть выведены краткие названия (поле SUBJECT) из таблицы SUBJECT в одну строку через запятую.

Использовать встроенную функцию RTRIM.

DECLARE @subj char(20), @disc char(300)=''

--объявление курсора

DECLARE DiciplineListCurs CURSOR

for SELECT SUBJECT FROM SUBJECT WHERE PULPIT='ИСиТ' ;

--открытие курсора

OPEN DiciplineListCurs;

--считывает одну строку из результирующего набора

--и продвигает указатель на следующую строку.

FETCH DiciplineListCurs INTO @subj;

print 'Список дисциплин на кафедре ИСиТ';

while @@fetch\_status=0

begin

set @disc=RTRIM(@subj)+','+@disc;

FETCH DiciplineListCurs into @subj;

end;

print @disc;

--закрытие курсора

CLOSE DiciplineListCurs;

2. Разработать сценарий, демонстрирующий отличие глобального курсора от локального на примере базы данных X\_UNIVER.

---------------------ЛОКАЛЬНЫЙ КУРСОР----------------------

--применятеся в рамках одного пакета, ресурсы освобождаются сразу после

--завершения работы пакета

DECLARE DiciplinePulpitCurs CURSOR LOCAL

for SELECT SUBJECT, PULPIT FROM SUBJECT;

DECLARE @subj char(20), @pulpit char(20);

OPEN DiciplinePulpitCurs;

fetch DiciplinePulpitCurs into @subj, @pulpit;

print '1. '+@subj+@pulpit;

fetch DiciplinePulpitCurs into @subj, @pulpit;

print '2. '+@subj+@pulpit;

-------------------------ГЛОБАЛЬНЫЙ КУРСОР-----------------

--курсор может быть объявлен, открыт и использован в разных пакетах.

--ресурсы освобождаются только после выполнения оператора DEALLOCATE

DECLARE FacultyPulpitFaculty CURSOR GLOBAL

for SELECT FACULTY, PULPIT FROM PULPIT;

DECLARE @faculty char(20), @pulp char(20);

OPEN FacultyPulpitFaculty;

fetch FacultyPulpitFaculty into @faculty, @pulp;

print '1. '+@faculty+@pulp;

go

DECLARE @faculty char(20), @pulp char(20);

fetch FacultyPulpitFaculty into @faculty, @pulp;

print '2. '+@faculty+@pulp;

close FacultyPulpitFaculty;

deallocate FacultyPulpitFaculty;

go

3. Разработать сценарий, демонстрирующий отличие статических курсоров от динамических на примере базы данных X\_UNIVER.

------------------------СТАТИЧЕСИКИЙ КУРСОР---------------------------

DECLARE @aud\_type char(10), @aud\_capacity int, @aud\_name char(10);

DECLARE AuditoriumInfoCursor CURSOR LOCAL DYNAMIC

for SELECT AUDITORIUM\_TYPE, AUDITORIUM\_CAPACITY, AUDITORIUM\_NAME

FROM AUDITORIUM where AUDITORIUM\_TYPE = 'ЛK';

open AuditoriumInfoCursor;

print 'Количество строк : '+cast(@@CURSOR\_ROWS as varchar(5));

UPDATE AUDITORIUM set AUDITORIUM\_NAME = '411-4', AUDITORIUM = '411-4' where AUDITORIUM\_CAPACITY = 1;

DELETE AUDITORIUM where AUDITORIUM\_CAPACITY = 10;

INSERT AUDITORIUM (AUDITORIUM, AUDITORIUM\_TYPE, AUDITORIUM\_CAPACITY, AUDITORIUM\_NAME)

values ('412-4', 'ЛК', 30, '412-4');

FETCH AuditoriumInfoCursor into @aud\_type, @aud\_capacity, @aud\_name;

while @@fetch\_status = 0

begin

print @aud\_type + ' '+ @aud\_capacity + ' '+ @aud\_name;

fetch AuditoriumInfoCursor into @aud\_type, @aud\_capacity, @aud\_name;

end;

CLOSE AuditoriumInfoCursor;

--------------------------------------------------------------------------------

DECLARE @sub char(10), @id char(40), @note char(1);

DECLARE Progress CURSOR LOCAL STATIC

for SELECT SUBJECT, IDSTUDENT, NOTE from PROGRESS

where SUBJECT='КГ';

open Progress;

print 'Количество строк: '+cast(@@CURSOR\_ROWS as varchar(5));

UPDATE PROGRESS set NOTE=8 where IDSTUDENT=1019;

DELETE PROGRESS where IDSTUDENT=1022;

INSERT PROGRESS (SUBJECT, IDSTUDENT, PDATE, NOTE) values ('КГ', 1022, '2013-05-06', 5);

FETCH Progress into @sub, @id, @note;

while @@FETCH\_STATUS=0

begin

print @sub+''+@id+''+@note;

fetch Progress into @sub, @id, @note;

end;

CLOSE Progress;

4. Разработать сценарий, демонстрирующий свойства навигации в результирующем наборе курсора с атрибутом SCROLL на примере базы данных X\_UNIVER.

Использовать все известные ключевые слова в операторе FETCH.

DECLARE @rowNumber int, @note int;

DECLARE NotesCurs cursor local dynamic SCROLL

for SELECT row\_number() over (order by NOTE) N, NOTE FROM PROGRESS

OPEN NotesCurs;

FETCH NotesCurs into @rowNumber,@note;

print 'следующая строка: ' + cast(@rowNumber as varchar(3))+' оценка в строке: '+rtrim(@note);

FETCH LAST from NotesCurs into @rowNumber,@note;

print 'последняя строка: ' + cast(@rowNumber as varchar(3)) +' оценка в строке: '+rtrim(@note);

FETCH PRIOR from NotesCurs into @rowNumber,@note;

print 'предыдущая строка от текущей: ' + cast(@rowNumber as varchar(3))+' оценка в строке: '+rtrim(@note);

FETCH ABSOLUTE 3 from NotesCurs into @rowNumber,@note;

print 'третья строка от начала: ' + cast(@rowNumber as varchar(3))+' оценка в строке: '+rtrim(@note);

FETCH ABSOLUTE -3 from NotesCurs into @rowNumber,@note;

print 'третья строка от конца: ' + cast(@rowNumber as varchar(3))+' оценка в строке: '+rtrim(@note);

FETCH RELATIVE 2 from NotesCurs into @rowNumber,@note;

print 'вторая строка вперед от текущей: ' + cast(@rowNumber as varchar(3))+' оценка в строке: '+rtrim(@note);

FETCH RELATIVE -2 from NotesCurs into @rowNumber,@note;

print 'вторая строка назад от текущей: ' + cast(@rowNumber as varchar(3))+' оценка в строке: '+rtrim(@note);

CLOSE NotesCurs;

5. Создать курсор, демонстрирующий применение конструкции CURRENT OF в секции WHERE с использованием операторов UPDATE и DELETE.

DECLARE @groupID int, @namee nvarchar(60)

DECLARE CurrentCurs cursor local dynamic

for SELECT IDGROUP, NAME

FROM STUDENT FOR UPDATE;

OPEN CurrentCurs;

FETCH CurrentCurs into @groupID, @namee;

DELETE STUDENT WHERE CURRENT OF CurrentCurs;--удаляется строка первая строка в таблице STUDENT

FETCH CurrentCurs into @groupID, @namee;

UPDATE STUDENT set IDGROUP=IDGROUP+1 where CURRENT OF CurrentCurs;-- и увеличивается на единицу номер группы в следующей строке.

CLOSE CurrentCurs;

6. Разработать SELECT-запрос, с помощью которого из таблицы PROGRESS удаляются строки, содержащие информацию о студентах, получивших оценки ниже 4 (использовать объединение таблиц PROGRESS, STUDENT, GROUPS).

Разработать SELECT-запрос, с помощью которого в таблице PROGRESS для студента с конкретным номером IDSTUDENT корректируется оценка (увеличивается на единицу).

DECLARE @subject char(10), @studentName char(100), @studentGroup int, @note int;

DECLARE HigherNotes cursor local dynamic

for SELECT SUBJECT, NAME, GROUPS.IDGROUP, NOTE FROM

PROGRESS Inner Join STUDENT

On PROGRESS.IDSTUDENT=STUDENT.IDSTUDENT

Inner Join GROUPS

On STUDENT.IDGROUP=GROUPS.IDGROUP

WHERE PROGRESS.NOTE<6;--<4 и <5 нет

OPEN HigherNotes;

FETCH HigherNotes into @subject, @studentName, @studentGroup, @note;

DELETE PROGRESS WHERE CURRENT OF HigherNotes;

CLOSE HigherNotes;

SELECT \* FROM PROGRESS;

-----------------------------------------------------------------------

DECLARE @studentID char(6), @studentNote int;

DECLARE IdStudentHigherNote cursor local dynamic

for SELECT IDSTUDENT,NOTE

FROM PROGRESS FOR UPDATE;

OPEN IdStudentHigherNote;

FETCH IdStudentHigherNote into @studentID, @studentNote;

UPDATE PROGRESS SET NOTE=NOTE+1

WHERE IDSTUDENT=1022;--первоначально была 6

CLOSE IdStudentHigherNote;

**Лабораторная работа №11. Особенности использования транзакций**

1. Разработать сценарий, демонстрирующий работу в режиме *неявной* транзакции.

Проанализировать пример, приведенный справа, в котором создается таблица Х, и создать сценарий для другой таблицы.

--Разработать сценарий, демонстрирующий работу в режиме неявной транзакции.

set nocount on

if exists (select \* from SYS.OBJECTS -- таблица X есть?

where OBJECT\_ID= object\_id(N'DBO.X') )

drop table TableOne;

declare @c int, @flag char = 'c'; -- commit или rollback?

SET IMPLICIT\_TRANSACTIONS ON -- включ. режим неявной транзакции

CREATE table TableOne(K int ); -- начало транзакции

INSERT TableOne values (1),(2),(3);

set @c = (select count(\*) from TableOne);

print 'количество строк в таблице TableOne: ' + cast( @c as varchar(2));

if @flag = 'c' commit; -- завершение транзакции: фиксация

else rollback; -- завершение транзакции: откат

SET IMPLICIT\_TRANSACTIONS OFF -- выключ. режим неявной транзакции

if exists (select \* from SYS.OBJECTS -- таблица X есть?

where OBJECT\_ID= object\_id(N'DBO.TableOne') )

print 'таблица "TableOne" есть';

else print 'таблицы "TableOne" нет'

2.Разработать сценарий, демонстрирующий свойство *атомарности* *явной* транзакции на примере базы данных X\_UNIVER.

В блоке CATCH предусмотреть выдачу соответствующих сообщений об ошибках.

Опробовать работу сценария при использовании различных операторов модификации таблиц.

--Явная. Разработать сценарий, демонстрирующий свойство атомарности явной транзакции.

begin try

begin tran --начало явной транзакции

insert PROFESSION values ('1-47 01 02', 'ИТ', 'Дизайн электронных и веб-изданий', 'программист-дизайнер');

insert PROGRESS values ('ОАиП', 1002, '2013-01-10', 8);

commit tran; --фиксация транзакции

end try

begin catch

print 'ошибка: '+case

when error\_number()=2627 and patindex('%PROFESSION\_PK%', error\_message())>0 --позиция первого символа подстроки

then 'дублирование записи'

else 'неизвестная ошибка: '+cast(error\_number() as varchar(5))+error\_message()

end;

if @@TRANCOUNT>0 rollback tran; --возвращает уровень вложенности транзакции, если>0, то транзакция не завершена

end catch;

3.Разработать сценарий, демонстрирующий применение оператора SAVE TRAN на примере базы данных X\_UNIVER.

В блоке CATCH предусмотреть выдачу соответствующих сообщений об ошибках.

Опробовать работу сценария при использовании различных контрольных точек и различных операторов модификации таблиц.

--Разработать сценарий, демонстрирующий применение оператора SAVE TRAN

declare @point varchar(150); --макс. длина имени 32

begin try

begin tran --начало явной транзакции

set @point='p1'; save tran @point; --контрольная точка p1

insert PROGRESS values ('ОАиП', 1025, '2013-01-10', 10);

set @point='p2'; save tran @point; --контрольная точка p2

insert FACULTY values ('ЛиД', 'Факультет лесной инженерии, материаловедения и дизайна');

commit tran; --фиксация транзакции

end try

begin catch

print 'ошибка: '+case when error\_number()=547

and patindex ('%FACULTY\_FK%', error\_message())>0

then 'конфликт с ограничением FOREIGN KEY'

else 'неизвестная ошибка: '+cast(error\_number() as varchar(5)) +error\_message()

end;

if @@TRANCOUNT>0

begin

print 'контрольная точка: '+@point;

rollback tran @point; --откат к контрольной точке

commit tran; --фиксация изменений, выполненных до контрольной точки

end;

end catch;

select \* from FACULTY;

4.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарий A представляет собой явную транзакцию с уровнем изолированности READ UNCOMMITED, сценарий B – явную транзакцию с уровнем изолированности READ COMMITED (по умолчанию).

Сценарий A должен демонстрировать, что уровень READ UNCOMMITED допускает неподтвержденное, неповторяющееся и фантомное чтение.

--Сценарий A представляет собой явную транзакцию с уровнем изолированности READ UNCOMMITED,

--сценарий B – явную транзакцию с уровнем изолированности READ COMMITED (по умолчанию).

--Сценарий A должен демонстрировать, что уровень READ UNCOMMITED допускает неподтвержденное,

--неповторяющееся и фантомное чтение.

---A---

set transaction isolation level READ UNCOMMITTED

begin transaction

-------------------------t1--------------------------

select @@SPID, 'insert FACULTY' 'результат', \* from FACULTY --возвращает системный идентификатор процесса, назначенный сервером текущему подключению

where FACULTY='ФЗО';

select @@SPID, 'update FACULTY' 'результат', \* from FACULTY

where FACULTY='ПИМ';

commit;

ДРУГОЙ ФАЙЛ

---B---

begin transaction

select @@SPID

insert FACULTY values ('ФЗО','Факультет заочного образования')

update FACULTY set FACULTY='ПИМ'

where FACULTY='ПиМ';

-------------------------t1--------------------------

-------------------------t2--------------------------

rollback;

5.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарии A и В представляют собой явные транзакции с уровнем изолированности READ COMMITED.

Сценарий A должен демонстрировать, что уровень READ COMMITED не допускает неподтвержденного чтения, но при этом возможно неповторяющееся и фантомное чтение.

--Сценарии A и В представляют собой явные транзакции с уровнем изолированности READ COMMITED.

--Сценарий A должен демонстрировать, что уровень READ COMMITED не допускает неподтвержденного чтения,

--но при этом возможно неповторяющееся и фантомное чтение.

---A---

set transaction isolation level READ COMMITTED

begin transaction

select count(\*) from FACULTY where FACULTY = 'ИТ';

--------------------------t1------------------

--------------------------t2-----------------

select 'update FACULTY' 'результат', count(\*)

from FACULTY where FACULTY = 'ИТ';

commit;

---B---

begin transaction

---------------------------t1--------------------

update FACULTY set FACULTY = 'ИТ'

where FACULTY = 'ФИТ'

commit;

---------------------------t2--------------------

6.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарий A представляет собой явную транзакцию с уровнем изолированности REPEATABLE READ. Сценарий B – явную транзакцию с уровнем изолированности READ COMMITED.

Сценарий A должен демонстрировать, что уровень REAPETABLE READ не допускает неподтвержденного чтения и неповторяющегося чтения, но при этом возможно фантомное чтение.

--Сценарий A представляет собой явную транзакцию с уровнем изолированности REPEATABLE READ.

--Сценарий B – явную транзакцию с уровнем изолированности READ COMMITED.

--Сценарий A должен демонстрировать, что уровень REAPETABLE READ не допускает неподтвержденного чтения и неповторяющегося чтения,

--но при этом возможно фантомное чтение.

---A---

set transaction isolation level REPEATABLE READ

begin transaction

select NOTE from PROGRESS where SUBJECT='СУБД';

-------------------------t1--------------------------

-------------------------t2--------------------------

select case

when NOTE=6 then 'insert PROGRESS' else ''

end 'результат', NOTE from PROGRESS where SUBJECT='СУБД';

commit;

---B---

begin transaction

-------------------------t1--------------------------

insert PROGRESS values ('СУБД', 1030, '2021-12-30', 8);

commit;

-------------------------t2--------------------------

delete PROGRESS where IDSTUDENT=1030

7.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарий A представляет собой явную транзакцию с уровнем изолированности SERIALIZABLE.

Сценарий B – явную транзакцию с уровнем изолированности READ COMMITED.

Сценарий A должен демонстрировать отсутствие фантомного, неподтвержденного и неповторяющегося чтения.

--Сценарий A представляет собой явную транзакцию с уровнем изолированности SERIALIZABLE.

--Сценарий B – явную транзакцию с уровнем изолированности READ COMMITED.

--Сценарий A должен демонстрировать отсутствие фантомного, неподтвержденного и неповторяющегося чтения.

---A---

set transaction isolation level SERIALIZABLE

begin transaction

--delete PROGRESS where SUBJECT='СУБД';

insert PROGRESS values ('СУБД', 1017, '2021-05-05', 10);

select IDSTUDENT from PROGRESS where SUBJECT='СУБД';

-------------------------t1--------------------------

select IDSTUDENT from PROGRESS where SUBJECT='СУБД';

-------------------------t2--------------------------

commit;

---B---

begin transaction

--delete PROGRESS where SUBJECT='СУБД';

insert PROGRESS values ('СУБД', 1017, '2021-05-05', 10);

select IDSTUDENT from PROGRESS where SUBJECT='СУБД';

-------------------------t1--------------------------

commit;

select IDSTUDENT from PROGRESS where SUBJECT='СУБД';

-------------------------t2--------------------------

insert PROGRESS values ('ОАиП', 1005, '2013-10-01', 8);

8.Разработать сценарий, демонстрирующий свойства *вложенных* транзакций, на примере базы данных X\_UNIVER.

begin tran --внешняя транзакция

insert PROGRESS values ('КГ', 1024, '2013-06-04', 8);

begin tran --внутрення транзакция

update PROGRESS set NOTE=8 where IDSTUDENT=1019;

commit;--внутрення транзакция

if @@TRANCOUNT>0 rollback; --внешняя транзакция

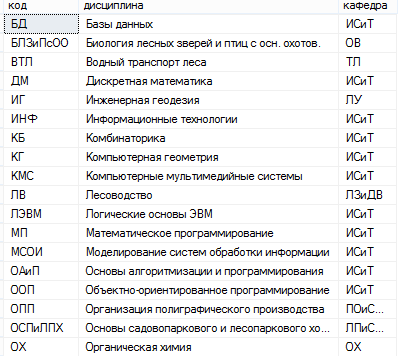
select

(select count(\*) from PROGRESS where IDSTUDENT=1019 and NOTE=8) 'внутрення транзакция',

(select count(\*) from PROGRESS where IDSTUDENT=1024) 'внешняя транзакция';

**Лабораторная работа №12. Разработка хранимых процедур**

1.Разработать хранимую процедуру без параметров с именем **PSUBJECT**. Процедура формирует результирующий набор на основе таблицы **SUBJECT**, аналогичный набору, представленному на рисунке:

****

К точке вызова процедура должна возвращать количество строк, выведенных в результирующий набор.

--Разработать хранимую процедуру без параметров с именем PSUBJECT.

--К точке вызова процедура должна возвращать количество строк, выведенных в результирующий набор.

use UNIVER\_lab4

CREATE PROCEDURE PSUBJECT --создание процедуры

as --хранится в БД

begin

declare @k int=(select count(\*) from SUBJECT);

select \* from SUBJECT;

return @k; --код возврата

end;

declare @n int=0;

EXEC @n=PSUBJECT; --вызов процедуры

print 'кол-во дисциплин='+cast(@n as varchar(3));

--drop procedure PSUBJECT;

--сначала запускать процедуру, потом declare и обращение к процедуре через EXEC

2.Найти процедуру **PSUBJECT** с помощью обозревателя объектов (Object Explorer) SSMS и через контекстное меню создать сценарий на изменение процедуры оператором ALTER.

Изменить процедуру **PSUBJECT**, созданную в задании 1, таким образом, чтобы она принимала два параметра с именами **@p** и **@c**. Параметр **@p** является входным, имеет тип VARCHAR(20) и значение по умолчанию NULL. Параметр **@с** является выходным, имеет тип INT.

Процедура **PSUBJECT** должна формировать результирующий набор, аналогичный набору, представленному на рисунке выше, но при этом содержать строки, соответствующие коду кафедры, заданному параметром **@p**. Кроме того, процедура должна формировать значение выходного параметра **@с**, равное количеству строк в результирующем наборе, а также возвращать значение к точке вызова, равное общему количеству дисциплин (количеству строк в таблице **SUBJECT**).

--Изменить процедуру PSUBJECT, добавив 2 параметра.

--Процедура PSUBJECT должна формировать результирующий набор, соответствующий заданному параметром @p.

--Кроме того, процедура должна формировать значение выходного параметра @с, равное количеству строк в результирующем наборе,

--а также возвращать значение к точке вызова, равное общему количеству дисциплин (количеству строк в таблице SUBJECT).

alter procedure PSUBJECT

@p varchar(20)=NULL,

@c int output

as

begin

declare @k int=(select count(\*) from SUBJECT);

print 'параметры: @p = '+@p+', @c = '+cast(@c as varchar(3));

select \* from SUBJECT where PULPIT=@p;

set @c=@@ROWCOUNT;--значение выходного параметра

return @k;

end;

declare @m int=0, @n int=0;

exec @m=PSUBJECT @p='ИСиТ', @c=@n output;

print 'кол-во дисциплин всего = '+cast(@m as varchar(3));

print 'кол-во дисциплин, соответствующих кафедре ИСиТ = ' + cast(@n as varchar(3));

3.Создать временную локальную таблицу с именем **#SUBJECT**. Наименование и тип столбцов таблицы должны соответствовать столбцам результирующего набора процедуры **PSUBJECT**, разработанной в задании 2.

Изменить процедуру **PSUBJECT** таким образом, чтобы она не содержала выходного параметра.

Применив конструкцию INSERT… EXECUTE с модифицированной процедурой **PSUBJECT**, добавить строки в таблицу **#SUBJECT**.

--Создать временную локальную таблицу с именем #SUBJECT.

--Изменить процедуру PSUBJECT таким образом, чтобы она не содержала выходного параметра.

--Применив конструкцию INSERT… EXECUTE с модифицированной процедурой PSUBJECT,

--добавить строки в таблицу #SUBJECT.

alter procedure PSUBJECT @p varchar(20)=NULL

as begin

declare @k int=(select count(\*) from SUBJECT);

select \* from SUBJECT where PULPIT=@p;

end;

CREATE table #TemporaryTableSUBJECT

(

SUBJECT char(10) primary key,

SUBJECT\_NAME varchar(100),

PULPIT char(20)

)

INSERT #TemporaryTableSUBJECT exec PSUBJECT @p='ИСиТ';

select \* from #TemporaryTableSUBJECT

4.Разработать процедуру с именем **PAUDITORIUM\_INSERT**. Процедура принимает четыре входных параметра: **@a**, **@n**, **@c** и **@t**. Параметр **@a** имеет тип CHAR(20), параметр **@n** имеет тип VARCHAR(50), параметр **@c** имеет тип INT и значение по умолчанию **0**, параметр **@t** имеет тип CHAR(10).

Процедура добавляет строку в таблицу **AUDITORIUM**. Значения столбцов **AUDITORIUM**, **AUDITORIUM\_NAME, AUDITORIUM\_CAPACITY** и **AUDITORIUM\_TYPE** добавляемой строки задаются соответственно параметрами **@a**, **@n**, **@c** и **@t**.

Процедура **PAUDITORIUM\_INSERT** должна применять механизм TRY/CATCH для обработки ошибок. В случае возникновения ошибки, процедура должна формировать сообщение, содержащее код ошибки, уровень серьезности и текст сообщения в стандартный выходной поток.

Процедура должна возвращать к точке вызова значение **-1** в том случае, если произошла ошибка и **1**, если выполнение успешно.

Опробовать работу процедуры с различными значениями исходных данных, которые вставляются в таблицу.

--Процедура PAUDITORIUM\_INSERT должна при-менять механизм TRY/CATCH для обработки ошибок.

--В случае возникновения ошибки, процедура должна формировать сообщение, содержащее код ошибки,

--уровень серьезности и текст сообщения в стандартный выходной поток.

create procedure PAUDITORIUM\_INSERT

@audit CHAR(30),

@auditName NVARCHAR(50),

@auditCapacity INT=0,

@auditType CHAR(10)

as declare @rc int = 1;

begin try

insert into AUDITORIUM (AUDITORIUM, AUDITORIUM\_NAME, AUDITORIUM\_CAPACITY, AUDITORIUM\_TYPE)

values (@audit , @auditName, @auditCapacity, @auditType)

return @rc;

end try

begin catch --обработка ошибки

print 'номер ошибки : '+ cast(error\_number() as varchar(6));

print 'сообщение : '+ error\_message();

print 'уровень : '+cast(error\_severity() as varchar(6));

print 'метка : '+cast(error\_state() as varchar(8));

print 'номер строки : '+cast(error\_line() as varchar(8));

if error\_procedure() is not null

print 'имя процедуры: '+error\_procedure();

return -1;

end catch;

declare @rc1 INT;

exec @rc1=PAUDITORIUM\_INSERT @audit ='101-1', @auditName='101-1', @auditCapacity=40, @auditType='ЛБ-К';

print 'код ошибки: '+cast(@rc1 as varchar(3));

--delete AUDITORIUM where AUDITORIUM='101-1'

drop procedure PAUDITORIUM\_INSERT

5.Разработать процедуру с именем **SUBJECT\_REPORT**, формирующую в стандартный выходной поток отчет со списком дисциплин на конкретной кафедре. В отчет должны быть выведены краткие названия (поле SUBJECT) из таблицы SUBJECT в одну строку через запятую (использовать встроенную функцию RTRIM). Процедура имеет входной параметр с именем **@p** типа CHAR(10), который предназначен для указания кода кафедры.

В том случае, если по заданному значению **@p** невозможно определить код кафедры, процедура должна генерировать ошибку с сообщением **ошибка в параметрах**.

Процедура **SUBJECT\_REPORT** должна возвращать к точке вызова количество дисциплин, отображенных в отчете.

--В отчет должны быть выведены краткие названия (поле SUBJECT) из таблицы SUBJECT в одну

--строку через запятую (использовать встроенную функцию RTRIM).

create procedure SUBJECT\_REPORT @p CHAR(50)

as

declare @rc int=0;

begin try

declare @tv char(20), @t char(300)='';

declare SubjCurs CURSOR for

select SUBJECT from SUBJECT where PULPIT=@p;

if not exists (select SUBJECT from SUBJECT where PULPIT=@p)

raiserror('ошибка',11,1);

else

open SubjCurs;

fetch SubjCurs into @tv;

print 'Дисциплины: ';

while @@FETCH\_STATUS=0

begin

set @t=rtrim(@tv)+', '+@t;

set @rc=@rc+1;

fetch SubjCurs into @tv;

end;

print @t;

close SubjCurs;

return @rc;

end try

begin catch

print 'ошибка в параметрах'

if error\_procedure() is not null

print 'имя процедуры: '+error\_procedure();

return @rc;

end catch;

declare @rc1 int;

exec @rc1=SUBJECT\_REPORT @p='ИСиТ';

print 'Количество дисциплин = '+cast(@rc1 as varchar(3));

--drop procedure SUBJECT\_REPORT

6.Разработать процедуру с именем **PAUDITORIUM\_INSERTX**. Процедура принимает пять входных параметров: **@a**, **@n**, **@c**, **@t** и **@tn**.

Параметры **@a**, **@n**, **@c**, **@t** аналогичны параметрам процедуры **PAUDITORIUM\_INSERT**. Дополнительный параметр **@tn** является входным, имеет тип VARCHAR(50), предназначен для ввода значения в столбец **AUDITORIUM\_TYPE**.**AUDITORIUM\_TYPENAME**.

Процедура добавляет две строки. Первая строка добавляется в таблицу **AUDITORIUM\_TYPE**. Значения столбцов **AUDITORIUM\_TYPE** и **AUDITORIUM\_ TYPENAME** добавляемой строки задаются соответственно параметрами **@t** и **@tn**. Вторая строка добавляется путем вызова процедуры **PAUDITORIUM\_INSERT**.

Добавление строки в таблицу **AUDITORIUM\_TYPE** и вызов процедуры **PAUDITORIUM\_INSERT** должны выполняться в рамках одной явной транзакции с уровнем изолированности SERIALIZABLE.

В процедуре должна быть предусмотрена обработка ошибок с помощью механизма TRY/CATCH. Все ошибки должны быть обработаны с выдачей соответствующего сообщения в стандартный выходной поток.

Процедура **PAUDITORIUM\_INSERTX** должна возвращать к точке вызова значение **-1** в том случае, если произошла ошибка и **1**, если выполнения процедуры завершилось успешно.

--Процедура добавляет две строки.

--Добавление строки в таблицу AUDITORI-UM\_TYPE и вызов процедуры PAUDITORI-UM\_INSERT должны выполняться

--в рамках одной явной транзакции с уровнем изолированности SERIALIZABLE.

create procedure PAUDITORIUM\_INSERTX

@audit char(30),

@auditName nvarchar(50),

@auditCapacity int=0,

@auditType char(10),

@auditTypeName nvarchar(50)

as declare @rc int=1;

begin try

set transaction isolation level SERIALIZABLE;

begin tran

insert into AUDITORIUM\_TYPE (AUDITORIUM\_TYPE, AUDITORIUM\_TYPENAME)

values (@auditType, @auditTypeName)

exec @rc=PAUDITORIUM\_INSERT @audit, @auditName, @auditCapacity, @auditType;

commit tran;

return @rc;

end try

begin catch

print 'номер ошибки: '+cast(error\_number() as varchar(6));

print 'сообщение: '+error\_message();

print 'уровень: '+cast(error\_severity() as varchar(6));

print 'метка: '+cast(error\_state() as varchar(8));

print 'номер строки: '+cast(error\_line() as varchar(8));

if error\_procedure() is not null

print 'имя процедуры: '+error\_procedure();

if @@TRANCOUNT>0

rollback tran;

return -1;

end catch;

declare @r int;

exec @r=PAUDITORIUM\_INSERTX @audit='205-4', @auditName='205-4', @auditCapacity=200,

@auditType='ЛА', @auditTypeName='Лабораторная аудитория';

print 'код ошибки = '+cast(@r as varchar(3));

drop procedure PAUDITORIUM\_INSERTX

**Лабораторная работа №13. Разработка и использование функций**

1. Разработать *скалярную* функцию с именем **COUNT\_STUDENTS**, которая вычисляет количество студентов на факультете, код которого задается параметром типа VARCHAR(20) с именем **@faculty**. Использовать внутреннее соединение таблиц FACULTY, GROUPS, STUDENT. Опробовать работу функции.

Внести изменения в текст функции с помощью оператора ALTER с тем, чтобы функция принимала второй параметр **@prof** типа VARCHAR(20),обозначающий специальность студентов. Для параметров определить значения по умолчанию NULL. Опробовать работу функции с помощью SELECT-запросов.

--Разработать скалярную функцию с именем COUNT\_STUDENTS, которая вычисляет количество

--студентов на факультете, код которого задается параметром типа VARCHAR(20) с именем @faculty.

--Использовать внутреннее соединение таблиц FACULTY, GROUPS, STUDENT.

use UNIVER\_lab4

go

create function COUNT\_STUDENTS(@faculty nvarchar(40)) returns int

as begin declare @rc int = 0;

set @rc=(select count(IDSTUDENT)

from STUDENT join GROUPS

on STUDENT.IDGROUP = GROUPS.IDGROUP

join FACULTY

on GROUPS.FACULTY = FACULTY.FACULTY

where FACULTY.FACULTY = @faculty);

return @rc;

end;

go

declare @f int=dbo.COUNT\_STUDENTS('ИТ');

print 'Количество студентов на факультете ИТ = ' + cast(@f as varchar(4));

select FACULTY [Факультеты],

dbo.COUNT\_STUDENTS(FACULTY) [Количество студентов на факультетах] from FACULTY;

--drop function COUNT\_STUDENTS

--Внести изменения в текст функции с помощью оператора ALTER с тем, чтобы функция принимала

--второй параметр @prof типа VARCHAR(20), обозначающий специальность студентов. Для параметров

--определить значения по умолчанию NULL.

go

alter function COUNT\_STUDENTS(@faculty nvarchar(20)=NULL, @prof varchar(20)=NULL) returns int

as begin declare @rc int = 0;

set @rc=(SELECT count(IDSTUDENT)

from STUDENT join GROUPS

on STUDENT.IDGROUP = GROUPS.IDGROUP

join FACULTY

on GROUPS.FACULTY = FACULTY.FACULTY

where FACULTY.FACULTY = isnull(@faculty, FACULTY.FACULTY) and

GROUPS.PROFESSION=isnull(@prof, GROUPS.PROFESSION));

return @rc;

end;

--вызов скалярной функции без применения параметров по умолчанию

go

select FACULTY.FACULTY [Факультеты],

GROUPS.PROFESSION [Специальности],

dbo.COUNT\_STUDENTS(FACULTY.FACULTY, GROUPS.PROFESSION) [Количество студентов на факультетах]

from FACULTY join GROUPS on FACULTY.FACULTY=GROUPS.FACULTY

group by FACULTY.FACULTY, GROUPS.PROFESSION order by [Количество студентов на факультетах] desc

--вызов скалярной функции с применением значения по умолчанию для второго параметра

select FACULTY [Факультеты],

dbo.COUNT\_STUDENTS(FACULTY, default) [Количество студентов на факультетах] from FACULTY;

--вызов функции со значениями по умолчанию для двух параметров

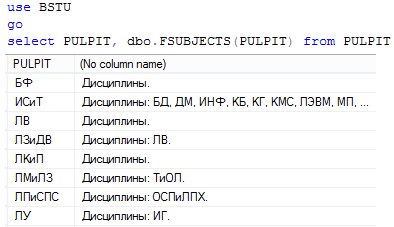
print 'Всего студентов: '+ cast(dbo.COUNT\_STUDENTS(default, default) as varchar(4));

2. Разработать *скалярную* функцию с именем **FSUBJECTS,** принимающую параметр **@p** типа VARCHAR(20), значение которого задает код кафедры (столбец **SUBJECT.PULPIT**).

Функция должна возвращать строку типа VARCHAR(300) с перечнем дисциплин в отчете.

Создать и выполнить сценарий, который создает отчет, аналогичный представленному ниже.

Примечание: использовать локальный статический курсор на основе SELECT-запроса к таблице **SUBJECT**.



--Разработать скалярную функцию с именем FSUBJECTS, принимающую параметр @p типа VARCHAR(20),

--значение которого задает код кафедры (столбец SUBJECT.PULPIT).

--Функция должна возвращать строку типа VARCHAR(300) с перечнем дисциплин в отчете.

go

create FUNCTION FSUBJECTS(@p nvarchar(20)) returns varchar(300)

as begin

declare @tv char(20);

declare @t varchar(300)='Дисциплины: ';

declare SubjectPulpitCursor CURSOR LOCAL static

for select SUBJECT from SUBJECT where PULPIT=@p;

open SubjectPulpitCursor;

fetch SubjectPulpitCursor into @tv;

while @@FETCH\_STATUS=0

begin

set @t = @t + ' ' + rtrim(@tv);

fetch SubjectPulpitCursor into @tv;

end;

return @t;

end;

--drop FUNCTION FSUBJECTS

go

select PULPIT, dbo.FSUBJECTS(PULPIT) [Дисциплины] from SUBJECT

select distinct PULPIT, dbo.FSUBJECTS(PULPIT)[Дисциплины] from SUBJECT

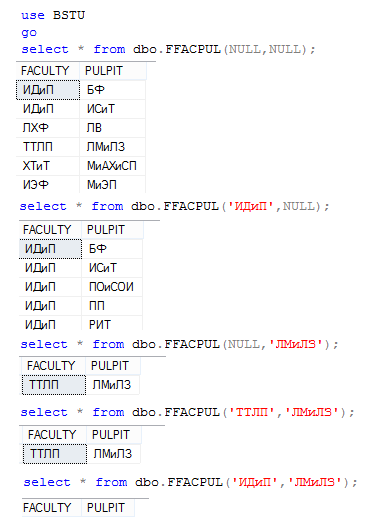
3. Разработать *табличную* функцию **FFACPUL**, результаты работы которой продемонстрированы на рисунке ниже.

Функция принимает два параметра, задающих код факультета (столбец **FACULTY.FACULTY**) и код кафедры (столбец **PULPIT.PULPIT**). Использует SELECT-запрос c левым внешним соединением между таблицами **FACULTY** и **PULPIT**.

Если оба параметра функции равны NULL, то она возвращает список всех кафедр на всех факультетах.

Если задан первый параметр (второй равен NULL), функция возвращает список всех кафедр заданного факультета.

Если задан второй параметр (первый равен NULL), функция возвращает результирующий набор, содержащий строку, соответствующую заданной кафедре.

****

Если заданы два параметра, функция возвращает результирующий набор, содержащий строку, соответствующую заданной кафедре на заданном факультете.

Если по заданным значениям параметров невозможно сформировать строки, функция возвращает пустой результирующий набор.

--Разработать табличную функцию FFACPUL, результаты работы которой продемонстрированы на рисунке ниже.

--Функция принимает два параметра, задающих код факультета (столбец FACULTY.FACULTY)

--и код кафедры (столбец PULPIT.PULPIT). Использует SELECT-запрос c левым внешним соединением между таблицами FACULTY и PULPIT.

go

create function FFACPUL(@f varchar(50), @p varchar(50)) returns table

as return

select FACULTY.FACULTY, PULPIT.PULPIT

from FACULTY left outer join PULPIT

on FACULTY.FACULTY=PULPIT.FACULTY

where FACULTY.FACULTY=isnull(@f, FACULTY.FACULTY)

and

PULPIT.PULPIT=isnull(@p, PULPIT.PULPIT);

go

--список всех кафедр на всех факультетах

select \* from dbo.FFACPUL(NULL, NULL);

--список всех кафедр заданного факультета

select \* from dbo.FFACPUL('ИЭФ', NULL);

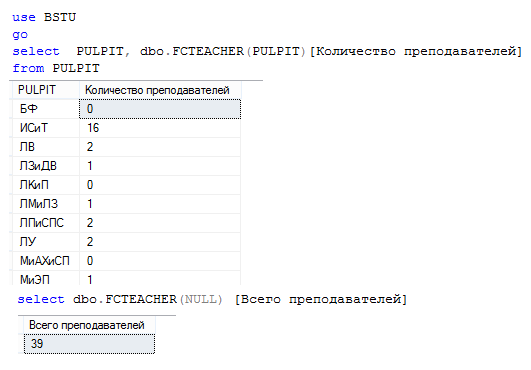
--набор, содерж. строку, соотв. заданной кафедре

select \* from dbo.FFACPUL(NULL, 'МиЭП');

--строка, соотв. заданной кафедре на заданном факультете

select \* from dbo.FFACPUL('ИЭФ', 'МиЭП');

4. На рисунке ниже показан сценарий, демонстрирующий работу *скалярной* функции **FCTEACHER**. Функция принимает один параметр, задающий код кафедры. Функция возвращает количество преподавателей на заданной параметром кафедре. Если параметр равен NULL, то возвращается общее количество преподавателей.

****

Разработать функцию **FCTEACHER**.

--Разработать функцию FCTEACHER.

--Функция принимает один параметр, задающий код кафедры. Функция возвращает количество преподавателей

--на заданной параметром кафедре. Если параметр равен NULL, то возвращается общее количество преподавателей.

create function FCTEACHER(@p varchar(50)) returns int

as

begin

declare @rc int=(select count(\*) from TEACHER

where PULPIT=isnull(@p, PULPIT));

return @rc;

end;

go

select PULPIT, dbo.FCTEACHER(PULPIT)[Количество преподавателей]

from PULPIT

select dbo.FCTEACHER(NULL)[Всего преподавателей]

6. Изменить эту функцию так, чтобы количество кафедр, количество групп, количество студентов и количество специальностей вычислялось отдельными скалярными функциями.

-- Изменить функцию так, чтобы количество кафедр, количество групп, количество студентов и количество специальностей

--вычислялось отдельными скалярными функциями.

use UNIVER\_lab4

--изначальная многооператорная табличная функция FACULTY\_REPORT

go

create function FACULTY\_REPORT(@c int) returns @fr table

([Факультет] varchar(50), [Количество кафедр] int, [Количество групп] int,

[Количество студентов] int, [Количество специальностей] int)

as begin

declare cc CURSOR static for

select FACULTY from FACULTY

where dbo.COUNT\_STUDENTS(FACULTY, default) > @c;

declare @f varchar(30);

open cc;

fetch cc into @f;

while @@fetch\_status = 0

begin

insert @fr values(@f, (select count(PULPIT) from PULPIT where FACULTY = @f),

(select count(IDGROUP) from GROUPS where FACULTY = @f), dbo.COUNT\_STUDENTS(@f, default),

(select count(PROFESSION) from PROFESSION where FACULTY = @f) );

fetch cc into @f;

end;

return;

end;

--количество кафедр

go

create function PULPIT\_COUNT(@faculty varchar(50)) returns int

as begin

declare @rc int=(select count(PULPIT) from PULPIT

where PULPIT.FACULTY = @faculty);

return @rc;

end;

--количество групп

go

create function GROUP\_COUNT(@faculty varchar(50)) returns int

as begin

declare @rc1 int=(select count(IDGROUP) from GROUPS

where GROUPS.FACULTY = @faculty);

return @rc1;

end;

--количество специальностей

go

create function PROFESSION\_COUNT(@faculty varchar(50)) returns int

as begin

declare @rc2 int=(select count(PROFESSION) from PROFESSION

where PROFESSION.FACULTY = @faculty);

return @rc2;

end;

--количество студентов

go

create function STUDENTS\_COUNT(@faculty nvarchar(20)=NULL, @prof varchar(20)=NULL) returns int

as begin

declare @rc2 int=(select count(IDSTUDENT)

from STUDENT join GROUPS

on STUDENT.IDGROUP = GROUPS.IDGROUP

join FACULTY

on GROUPS.FACULTY = FACULTY.FACULTY

where FACULTY.FACULTY = isnull(@faculty, FACULTY.FACULTY) and

GROUPS.PROFESSION=isnull(@prof, GROUPS.PROFESSION));

return @rc2;

end;

--измененная функция со скалярными функциями

go

create function FACULTY\_REPORT1(@c int) returns @fr table

([Факультет]varchar(50), [Количество кафедр]int, [Количеcтво групп]int,

[Количество студентов]int, [Количество специальностей]int)

as begin

declare cc CURSOR static for

select FACULTY from FACULTY

where dbo.STUDENTS\_COUNT(FACULTY, default)>@c;

declare @f varchar(30);

open cc;

fetch cc into @f;

while @@fetch\_status=0

begin

insert @fr values(@f, dbo.PULPIT\_COUNT(@f), dbo.GROUP\_COUNT(@f), dbo.STUDENTS\_COUNT(@f,default), dbo.PROFESSION\_COUNT(@f))

fetch cc into @f;

end;

close cc;

return;

end;

go

select \* from dbo.FACULTY\_REPORT1(-1); --в скобках количество студентов

**Лабораторная работа №14. Применение триггеров**

1.С помощью сценария, представленного на рисунке, создать таблицу **TR\_AUDIT**.

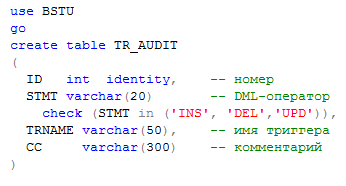
****

Таблица предназначена для добавления в нее строк триггерами.

В столбец **STMT** триггер должен поместить событие, на которое он среагировал, а в столбец **TRNAME −** собственное имя.

Разработать AFTER-триггер с именем **TR\_TEACHER\_INS** для таблицы **TEACHER**, реагирующий на событие **INSERT**. Триггер должен записывать строки вводимых данных в таблицу **TR\_AUDIT**. В столбец **СС** помещаются значения столбцов вводимой строки.

--Создать таблицу TR\_AUDIT.

--Разработать AFTER-триггер с именем TR\_TEACHER\_INS для таблицы TEACHER, реагирующий на событие INSERT.

--Триггер должен записывать строки вводимых данных в таблицу TR\_AUDIT.

--В столбец СС помещаются значения столбцов вводимой строки.

go

create table TR\_AUDIT

(

ID int identity, --номер

--В STMT триггер должен поместить событие, на которое он среагировал

STMT varchar(20)

check (STMT in ('INS', 'DEL', 'UPD')), --DML-оператор

TRNAME varchar(50), --имя триггера

CC varchar(300) --комментарий

)

go

create trigger TR\_TEACHER\_INS on TEACHER after INSERT

as declare @a1 char(10), @a2 varchar(100), @a3 char(1), @a4 char(20), @in varchar(300);

print 'Операция вставки';

set @a1=(select [TEACHER] from INSERTED);

set @a2=(select [TEACHER\_NAME] from INSERTED);

set @a3=(select [GENDER] from INSERTED);

set @a4=(select [PULPIT] from INSERTED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

insert into TR\_AUDIT(STMT, TRNAME, CC) values ('INS', 'TR\_TEACHER\_INS', @in);

return;

go

insert into TEACHER values ('НВСЛСК', 'Новосельская Ольга Александровна', 'ж', 'ИСиТ');

select \* from TR\_AUDIT;

--delete TEACHER from TEACHER where TEACHER.TEACHER='НВСЛСК'

--drop table TR\_AUDIT;

--drop trigger TR\_TEACHER\_INS

2.Создать AFTER-триггер с именем **TR\_TEACHER\_DEL** для таблицы **TEA-CHER**, реагирующий на событие **DELETE**. Триггер должен записывать строку данных в таблицу **TR\_AUDIT** для каждой удаляемой строки. В столбец **СС** помещаются значения столбца **TEACHER** удаляемой строки.

--Создать AFTER-триггер с именем TR\_TEACHER\_DEL для таблицы TEA-CHER, реагирующий на событие DELETE.

--Триггер должен записывать строку данных в таблицу TR\_AUDIT для каждой удаляемой строки. В столбец СС

--помещаются значения столбца TEACHER удаляемой строки.

go

create table TR\_AUDIT

(

ID int identity, --номер

--В STMT триггер должен поместить событие, на которое он среагировал

STMT varchar(20)

check (STMT in ('INS', 'DEL', 'UPD')), --DML-оператор

TRNAME varchar(50), --имя триггера

CC varchar(300) --комментарий

)

go

create trigger TR\_TEACHER\_DEL

on TEACHER after DELETE

as

declare @a1 char(10), @a2 varchar(100), @a3 char(1), @a4 char(20), @in varchar(300);

print 'Операция удаления';

set @a1=(select [TEACHER] from DELETED);

set @a2=(select [TEACHER\_NAME] from DELETED);

set @a3=(select [GENDER] from DELETED);

set @a4=(select [PULPIT] from DELETED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values ('DEL', 'TR\_TEACHER\_DEL', @in);

return;

go

delete TEACHER where TEACHER='НВСЛСК';

select \* from TR\_AUDIT;

--drop trigger TR\_TEACHER\_DEL

3.Создать AFTER-триггер с именем **TR\_TEACHER\_UPD** для таблицы **TEA-CHER**, реагирующий на событие **UPDATE**. Триггер должен записывать строку данных в таблицу **TR\_AUDIT** для каждой изменяемой строки. В столбец **СС** помещаются значения столбцов изменяемой строки до и после изменения.

--Создать AFTER-триггер с именем TR\_TEACHER\_UPD для таблицы TEACHER, реагирующий на событие UPDATE.

--Триггер должен записывать строку данных в таблицу TR\_AUDIT для каждой изменяемой строки.

--В столбец СС помещаются значения столбцов изменяемой строки до и после изменения.

go

create table TR\_AUDIT

(

ID int identity, --номер

--В STMT триггер должен поместить событие, на которое он среагировал

STMT varchar(20)

check (STMT in ('INS', 'DEL', 'UPD')), --DML-оператор

TRNAME varchar(50), --имя триггера

CC varchar(300) --комментарий

)

go

create trigger TR\_TEACHER\_UPD

on TEACHER after UPDATE

as declare @a1 char(10), @a2 varchar(100), @a3 char(1), @a4 char(20), @in varchar(300);

print 'Операция изменения';

set @a1=(select [TEACHER] from INSERTED);

set @a2=(select [TEACHER\_NAME] from INSERTED);

set @a3=(select [GENDER] from INSERTED);

set @a4=(select [PULPIT] from INSERTED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

set @a1=(select [TEACHER] from DELETED);

set @a2=(select [TEACHER\_NAME] from DELETED);

set @a3=(select [GENDER] from DELETED);

set @a4=(select [PULPIT] from DELETED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values ('UPD', 'TR\_TEACHER\_UPD', @in);

return;

go

update TEACHER set TEACHER='НВСЛСК\_О\_А' where TEACHER='НВСЛСК';

select \* from TR\_AUDIT;

--drop trigger TR\_TEACHER\_UPD

4.Создать AFTER-триггер с именем **TR\_TEACHER** для таблицы **TEACHER**, реагирующий на события **INSERT**, **DELETE**, **UPDATE**.

Триггер должен записывать строку данных в таблицу **TR\_AUDIT** для каждой изменяемой строки. В коде триггера определить событие, активизировавшее триггер и поместить в столбец **СС** соответствующую событию информацию.

Разработать сценарий, демонстрирующий работоспособность триггера.

--Создать AFTER-триггер с именем TR\_TEACHER для таблицы TEACHER, реагирующий на события INSERT, DELETE, UPDATE.

go

create table TR\_AUDIT

(

ID int identity, --номер

--В STMT триггер должен поместить событие, на которое он среагировал

STMT varchar(20)

check (STMT in ('INS', 'DEL', 'UPD')), --DML-оператор

TRNAME varchar(50), --имя триггера

CC varchar(300) --комментарий

)

go

create trigger TR\_TEACHER

on TEACHER after INSERT, DELETE, UPDATE

as declare @a1 char(10), @a2 varchar(100), @a3 char(1), @a4 char(20), @in varchar(300);

declare @ins int=(select count(\*) from inserted),

@del int=(select count(\*) from deleted);

if @ins>0 and @del=0

begin

print 'Событие: INSERT';

set @a1=(select [TEACHER] from INSERTED);

set @a2=(select [TEACHER\_NAME] from INSERTED);

set @a3=(select [GENDER] from INSERTED);

set @a4=(select [PULPIT] from INSERTED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values ('INS', 'TR\_TEACHER\_INS', @in);

end;

else

if @ins=0 and @del>0

begin

print 'Событие: DELETE';

set @a1=(select [TEACHER] from DELETED);

set @a2=(select [TEACHER\_NAME] from DELETED);

set @a3=(select [GENDER] from DELETED);

set @a4=(select [PULPIT] from DELETED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values ('DEL', 'TR\_TEACHER\_DEL', @in);

end;

else

if @ins>0 and @del>0

begin

print 'Событие: UPDATE';

set @a1=(select [TEACHER] from INSERTED);

set @a2=(select [TEACHER\_NAME] from INSERTED);

set @a3=(select [GENDER] from INSERTED);

set @a4=(select [PULPIT] from INSERTED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

set @a1=(select [TEACHER] from DELETED);

set @a2=(select [TEACHER\_NAME] from DELETED);

set @a3=(select [GENDER] from DELETED);

set @a4=(select [PULPIT] from DELETED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values ('UPD', 'TR\_TEACHER\_UPD', @in);

end;

return;

insert into TEACHER values ('НВСЛСК', 'Новосельская Ольга Александровна', 'ж', 'ИСиТ');

update TEACHER set TEACHER='НВСЛСК\_О\_А' where TEACHER='НВСЛСК';

delete TEACHER where TEACHER='НВСЛСК';

select \* from TR\_AUDIT;

--drop trigger TR\_TEACHER

5.Разработать сценарий, который демонстрирует на примере базы данных X\_UNIVER, что проверка ограничения целостности выполняется до срабатывания AFTER-триггера.

--Разработать сценарий, который демонстрирует, что проверка ограничения

--целостности выполняется до срабатывания AFTER-триггера.

alter table TEACHER add constraint GENDER check(GENDER in ('м','ж'));

go

update TEACHER set GENDER='трансгендер' where GENDER='ж';

6.Создать для таблицы **TEACHER** три AFTER-триггера с именами: **TR\_TEACHER\_ DEL1**, **TR\_TEACHER\_DEL2** и **TR\_TEA-CHER\_ DEL3**. Триггеры должны реагировать на событие DELETE и формировать соответствующие строки в таблицу **TR\_AUDIT**. Получить список триггеров таблицы **TEACHER**.

Упорядочить выполнение триггеров для таблицы **TEACHER**, реагирующих на событие **DELETE** следующим образом: первым должен выполняться триггер с именем **TR\_TEA-CHER\_DEL3**, последним – триггер **TR\_TEACHER\_DEL2**.

Примечание: использовать системные представления **SYS.TRIGGERS** и **SYS.TRIG-GERS\_ EVENTS**, а также системную процедуру **SP\_SETTRIGGERORDERS**.

--Создать для таблицы TEACHER три AFTER-триггера с именами: TR\_TEACHER\_ DEL1, TR\_TEACHER\_DEL2 и TR\_TEACHER\_ DEL3.

--Триггеры должны реагировать на событие DELETE и формировать соответствующие строки в таблицу TR\_AUDIT.

--Получить список триггеров таблицы TEACHER.

go

create table TR\_AUDIT

(

ID int identity, --номер

--В STMT триггер должен поместить событие, на которое он среагировал

STMT varchar(20)

check (STMT in ('INS', 'DEL', 'UPD')), --DML-оператор

TRNAME varchar(50), --имя триггера

CC varchar(300) --комментарий

)

go

create trigger TR\_TEACHER\_DEL1 on TEACHER after DELETE

as declare @a1 char(10), @a2 varchar(100), @a3 char(1), @a4 char(20), @in varchar(300);

print 'TR\_TEACHER\_DEL1';

set @a1=(select [TEACHER] from DELETED);

set @a2=(select [TEACHER\_NAME] from DELETED);

set @a3=(select [GENDER] from DELETED);

set @a4=(select [PULPIT] from DELETED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values ('DEL', 'TR\_TEACHER\_DEL1', @in);

return;

go

create trigger TR\_TEACHER\_DEL2 on TEACHER after DELETE

as declare @a1 char(10), @a2 varchar(100), @a3 char(1), @a4 char(20), @in varchar(300);

print 'TR\_TEACHER\_DEL2';

set @a1=(select [TEACHER] from DELETED);

set @a2=(select [TEACHER\_NAME] from DELETED);

set @a3=(select [GENDER] from DELETED);

set @a4=(select [PULPIT] from DELETED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values ('DEL', 'TR\_TEACHER\_DEL2', @in);

return;

go

create trigger TR\_TEACHER\_DEL3 on TEACHER after DELETE

as

declare @a1 char(10), @a2 varchar(100), @a3 char(1), @a4 char(20), @in varchar(300);

print 'TR\_TEACHER\_DEL3';

set @a1=(select [TEACHER] from DELETED);

set @a2=(select [TEACHER\_NAME] from DELETED);

set @a3=(select [GENDER] from DELETED);

set @a4=(select [PULPIT] from DELETED);

set @in=@a1+' '+@a2+' '+@a3+' '+@a4;

insert into TR\_AUDIT(STMT, TRNAME, CC)

values ('DEL', 'TR\_TEACHER\_DEL3', @in);

return;

go

--список триггеров

select t.name, e.type\_desc

from sys.triggers t join sys.trigger\_events e

on t.object\_id = e.object\_id

where OBJECT\_NAME(t.parent\_id) = 'TEACHER'

--Упорядочить выполнение триггеров для таблицы TEACHER, реагирующих на событие DELETE следующим образом:

--первым должен выполняться триггер с именем TR\_TEACHER\_DEL3, последним – триггер TR\_TEACHER\_DEL2.

exec SP\_SETTRIGGERORDER @triggername = 'TR\_TEACHER\_DEL',

@order = 'First', @stmttype = 'DELETE';

exec SP\_SETTRIGGERORDER @triggername = 'TR\_TEACHER\_DEL2',

@order = 'Last', @stmttype = 'DELETE';

--insert into TEACHER values ('НВСЛСК', 'Новосельская Ольга Александровна', 'ж', 'ИСиТ');

--delete TEACHER where TEACHER='НВСЛСК';

7.Разработать сценарий, демонстрирующий на примере базы данных X\_UNIVER утверждение: AFTER-триггер является частью транзакции, в рамках которого выполняется оператор, активизировавший триггер.

--Разработать сценарий, демонстрирующий утверждение: AFTER-триггер является частью транзакции,

--в рамках которого выполняется оператор, активизировавший триггер.

go

create trigger FacultyTrigger

on FACULTY after INSERT, DELETE, UPDATE

as declare @c int = (select count (\*) from FACULTY);

if (@c>7)

begin

raiserror('Количество факультетов не может быть > 7', 10, 1);

rollback;

end;

return;

insert into PULPIT(PULPIT) values ('Военная кафедра');

--drop trigger FacultyTrigger

8.Для таблицы **FACULTY** создать **INSTEAD OF**-триггер, запрещающий удаление строк в таблице.

Разработать сценарий, который демонстрирует на примере базы данных X\_UNIVER, что проверка ограничения целостности выполнена, если есть INSTEAD OF-триггер.

С помощью оператора DROP удалить все DML-триггеры, созданные в этой лабораторной работе.

--Для таблицы FACULTY создать INSTEAD OF-триггер, запрещающий удаление строк в таблице.

go

create trigger INSTEAD\_OF

on FACULTY instead of DELETE

as raiserror(N'Удаление запрещено', 10, 1);

return;

delete from FACULTY where FACULTY='ИЭФ';

drop trigger TR\_TEACHER\_INS, TR\_TEACHER\_DEL, TR\_TEACHER\_UPD, TR\_TEACHER, TR\_TEACHER\_DEL1,

TR\_TEACHER\_DEL2, TR\_TEACHER\_DEL3, FacultyTrigger;

9.Создать DDL-триггер, реагирующий на все DDL-события в БД **UNIVER**. Триггер должен запрещать создавать новые таблицы и удалять существующие. Свое выполнение триггер должен сопровождать сообщением, которое содержит: тип события, имя и тип объекта, а также пояснительный текст, в случае запрещения выполнения оператора.

Разработать сценарий, демонстрирующий работу триггера.

--Создать DDL-триггер, реагирующий на все DDL-события в БД UNIVER. Триггер должен запрещать

--создавать новые таблицы и удалять существующие.

go

create trigger DDL\_UNIVER\_lab4 on database

for DDL\_DATABASE\_LEVEL\_EVENTS

as

declare @t varchar(50) = EVENTDATA().value('(/EVENT\_INSTANCE/EventType)[1]', 'varchar(50)');

declare @t1 varchar(50) = EVENTDATA().value('(/EVENT\_INSTANCE/ObjectName)[1]', 'varchar(50)');

declare @t2 varchar(50) = EVENTDATA().value('(/EVENT\_INSTANCE/ObjectType)[1]', 'varchar(50)');

if @t1='FACULTY'

begin

print 'Тип события: '+@t;

print 'Имя объекта: '+@t1;

print 'Тип объекта: '+@t2;

raiserror( N'Операции с базой данных UNIVER\_lab4 запрещены', 16, 1);

rollback;

end;

drop table FACULTY;

drop trigger UNIVER\_lab4;

**Лабораторная работа №15. Использование XML**

1.Разработать сценарий создания XML-документа в режиме PATH из таблицы **TEACHER** для преподавателей кафедры ИСиТ.

use UNIVER\_lab4

go

select [Кафедра].FACULTY[Факультет/@код],

[Кафедра].PULPIT\_NAME [Факультет/Кафедра/@код],

[Преподаватель].TEACHER\_NAME [Факультет/Кафедра/Преподаватель/data()],

[Преподаватель].GENDER [Факультет/Кафедра/Преподаватель/data()]

from TEACHER [Преподаватель] join PULPIT [Кафедра]

on [Преподаватель].PULPIT=[Кафедра].PULPIT

where [Кафедра].PULPIT='ИСиТ' for xml PATH(''),

root('Список\_преподавателей'), elements;

2. Разработать сценарий создания XML-документа в режиме AUTO на основе SELECT-запроса к таблицам **AUDITORIUM** и **AUDITORIUM\_TYPE,** которыйсодержит следующие столбцы: наименование аудитории, наименование типа аудитории и вместимость. Найти только лекционные аудитории.

--Разработать сценарий создания XML-документа в режиме AUTO на основе SELECT-запроса к таблицам

--AUDITORIUM и AUDITORIUM\_TYPE, который содержит следую-щие столбцы: наименование аудитории,

--наименование типа аудитории и вместимость. Найти только лекционные аудитории.

use UNIVER\_lab4

select [Аудитория].AUDITORIUM\_NAME[Наименование\_аудитории],

[Тип\_Aудитории].AUDITORIUM\_TYPE [Тип\_аудитории],

[Аудитория].AUDITORIUM\_CAPACITY [Вместимость\_аудитории]

from AUDITORIUM [Аудитория] join AUDITORIUM\_TYPE [Тип\_Aудитории]

on [Аудитория].AUDITORIUM\_TYPE=[Тип\_Aудитории].AUDITORIUM\_TYPE

where [Аудитория].AUDITORIUM\_TYPE='ЛК' for xml AUTO,

root('Лекционные\_аудитории'),elements;

select [Аудитория].AUDITORIUM\_NAME[Наименование\_аудитории],

[Аудитория].AUDITORIUM\_TYPE [Тип\_аудитории],

[Аудитория].AUDITORIUM\_CAPACITY[Вместимость]

from AUDITORIUM Аудитория join AUDITORIUM\_TYPE

on [Аудитория].AUDITORIUM\_TYPE=AUDITORIUM\_TYPE .AUDITORIUM\_TYPE

where [Аудитория].AUDITORIUM\_TYPE='ЛК' for xml AUTO,

root('Лекционные\_аудитории'),elements;

3.Разработать XML-документ, содержащий данные о трех новых учебных дисциплинах, которые следует добавить в таблицу **SUBJECT**.

Разработать сценарий, извлекающий данные о дисциплинах из XML-документа и добавляющий их в таблицу **SUBJECT**.

При этом применить системную функцию **OPENXML** и конструкцию INSERT… SELECT.

--Разработать XML-документ, содержащий данные о трех новых учебных дисциплинах, которые следует добавить в таблицу SUBJECT.

--Разработать сценарий, извлекающий данные о дисциплинах из XML-документа и добавляющий их в таблицу SUBJECT.

--При этом применить системную функцию OPENXML и конструкцию INSERT… SELECT.

use UNIVER\_lab4

go

declare @h int=0,

@x varchar(2000)='<?xml version="1.0" encoding="windows-1251" ?>

<дисциплины>

<дисциплина код="КГиД" название="Компьютерная графика и дизайн" кафедра="ИСиТ"/>

<дисциплина код="КГиГ" название="Компьютерная геометрия и графика" кафедра="ИСиТ"/>

<дисциплина код="ТЦ" название="Теория цвета" кафедра="ИСиТ"/>

</дисциплины>';

exec sp\_xml\_preparedocument @h output, @x; -- подготовка документа

insert SUBJECT select\* from openxml (@h, '/дисциплины/дисциплина',0)

with([код] char(20), [название] varchar(100), [кафедра] char(20));

exec sp\_xml\_removedocument @h;

select \* from SUBJECT;

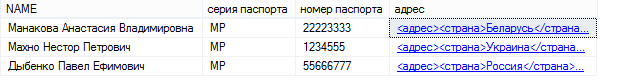
delete from SUBJECT where SUBJECT.SUBJECT='КГиД' or SUBJECT.SUBJECT='КГиГ' or SUBJECT.SUBJECT='МП'

4.Используя таблицу **STUDENT** разработать XML-структуру, содержащую паспортные данные студента: серию и номер паспорта, личный номер, дата выдачи и адрес прописки.

Разработать сценарий, в который включен оператор INSERT, добавляющий строку с XML-столбцом.

Включить в этот же сценарий оператор UPDATE, изменяющий столбец **INFO** у одной строки таблицы **STUDENT** и оператор SELECT, формирующий результирующий набор, аналогичный представленному на рисунке.

В SELECT-запросе использовать методы QUERY и VALUEXML-типа.



--Используя таблицу STUDENT разработать XML-структуру, содержащую паспортные данные студента:

--серию и номер паспорта, личный номер, дата выдачи и адрес прописки.

--Разработать сценарий, в который включен оператор INSERT, добавляющий строку с XML-столбцом.

--Включить в этот же сценарий оператор UPDATE, изменяющий столбец INFO у одной строки таблицы STUDENT и оператор SELECT,

--формирующий результирующий набор, аналогичный представленному на рисунке.

--В SELECT-запросе использовать методы QUERY и VALUEXML-типа.

use UNIVER\_lab4

insert into STUDENT(IDGROUP, NAME, BDAY, INFO)

values (1, 'Манакова Анастасия Владимировна', '11.08.1999',

'<студент>

<паспорт серия="MP" номер="22223333"/>

<адрес>

<страна>Беларусь</страна>

<город>Минск</город>

<улица>Лобанка</улица>

<дом>62</дом>

<квартира>47</квартира>

</адрес>

</студент>');

select IDGROUP, NAME, BDAY, INFO from STUDENT where BDAY='11.08.1999';

delete STUDENT where BDAY='11.08.1999';

update STUDENT

set INFO='<студент>

<паспорт серия="MP" номер="22223333"/>

<адрес>

<страна>Беларусь</страна>

<город>Минск</город>

<улица>Лобанка</улица>

<дом>62</дом>

<квартира>47</квартира>

</адрес>

</студент>'

where INFO.value('(/студент/паспорт/@номер)[1]','varchar(10)')='22223333';

select IDSTUDENT,

INFO.value('(/студент/паспорт/@серия)[1]','varchar(10)') [серия паспорта],

INFO.value('(/студент/паспорт/@номер)[1]','varchar(10)') [номер паспорта],

INFO.query('/студент/адрес') [адрес]

from STUDENT where BDAY='11.08.1999';

5. Изменить (ALTER TABLE) таблицу **STUDENT** в базе данных **UNIVER** таким образом, чтобы значения *типизированного* столбца с именем **INFO** контролировались коллекцией XML-схем (XML SCHEMACOLLECTION), представленной в правой части.

Разработать сценарии, демонстрирующие ввод и корректировку данных (операторы INSERT и UPDATE) в столбец **INFO** таблицы **STUDENT**, как содержащие ошибки, так и правильные.

Разработать другую XML-схему и добавить ее в коллекцию XML-схем в БД UNIVER**.**

--Изменить (ALTER TABLE) таблицу STUDENT в базе данных UNIVER таким образом, чтобы значения типизированного

--столбца с именем INFO контролировались коллекцией XML-схем (XML SCHEMACOLLECTION), представленной в правой части.

--Разработать сценарии, демонстрирующие ввод и корректировку данных (операторы INSERT и UPDATE) в столбец INFO таблицы STUDENT,

--как содержащие ошибки, так и правильные.

use UNIVER\_lab4

go

create xml schema collection Student as

N'<?xml version="1.0" encoding="utf-16" ?>

<xs:schema attributeFormDefault="unqualified"

elementFormDefault="qualified"

xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="студент">

<xs:complexType><xs:sequence>

<xs:element name="паспорт" maxOccurs="1" minOccurs="1">

<xs:complexType>

<xs:attribute name="серия" type="xs:string" use="required" />

<xs:attribute name="номер" type="xs:unsignedInt" use="required"/>

</xs:complexType>

</xs:element>

<xs:element name="адрес">

<xs:complexType><xs:sequence>

<xs:element name="страна" type="xs:string" />

<xs:element name="город" type="xs:string" />

<xs:element name="улица" type="xs:string" />

<xs:element name="дом" type="xs:string" />

<xs:element name="квартира" type="xs:string" />

</xs:sequence></xs:complexType> </xs:element>

</xs:sequence></xs:complexType>

</xs:element>

</xs:schema>';

alter table STUDENT alter column INFO xml(Student);

select Name, INFO from STUDENT where BDAY='11.08.1999';

--drop XML SCHEMA COLLECTION Student;

insert into STUDENT(IDGROUP, NAME, BDAY, INFO)

values (1, 'Манакова Анастасия Владимировна', '11.08.1999',

'<студент>

<паспорт серия="MP" номер="22223333"/>

<адрес>

<страна>Беларусь</страна>

<город>Минск</город>

<улица>Лобанка</улица>

<дом>62</дом>

<квартира>47</квартира>

</адрес>

</студент>');

update STUDENT

set INFO='<студент>

<паспорт серия="MP" номер="22223333"/>

<адрес>

<страна>Беларусь</страна>

<город>Минск</город>

<улица>Лобанка</улица>

<дом>62</дом>

<квартира>47</квартира>

</адрес>

</студент>'

where INFO.value('(/студент/паспорт/@номер)[1]','varchar(10)')='22223333';