

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Кузбасский государственный технический университет  
имени Т. Ф. Горбачева»

Институт информационных технологий, машиностроения и  
автотранспорта  
Кафедра информационных и автоматизированных  
производственных систем

Олег Николаевич Ванеев

## **УПРАВЛЕНИЕ ДАННЫМИ**

### **ЧАСТЬ I**

Методические материалы к лабораторным работам

Рекомендовано учебно-методической комиссией  
направления подготовки 09.03.02 «Информационные системы и  
технологии» в качестве электронного издания для использования  
в учебном процессе

Кемерово 2024

Рецензенты: Чичерин И. В. – кандидат тех наук, доцент, заведующий кафедрой информационных и автоматизированных производственных систем ФГБОУ ВО «Кузбасский государственный технический университет имени Т. Ф. Горбачева»  
Сыркин И. С. – доцент, кандидат тех. наук кафедры информационных и автоматизированных производственных систем ФГБОУ ВО «Кузбасский государственный технический университет имени Т. Ф. Горбачева»

**Ванеев, О.Н. Управление данными Часть I:** методические материалы по выполнению лабораторных работ для обучающихся направления подготовки 09.03.02 «Информационные системы и технологии»/ сост. О. Н. Ванеев, Кузбасский государственный технический университет имени Т. Ф. Горбачева. – Кемерово, 2024. – Текст : электронный.

В данных методических материалах изложено содержание по выполнению лабораторных работ, цель работы, теоретические положения, порядок выполнения, задания для выполнения, контрольные вопросы.

©Кузбасский государственный  
технический университет  
имени Т. Ф. Горбачева, 2024  
© Ванеев О. Н.  
составление 2024

## ОГЛАВЛЕНИЕ

ЛАБОРАТОРНАЯ РАБОТА 1. ИЗУЧЕНИЕ СРЕДЫ РАБОТЫ С СУБД SQL SERVER. РАЗРАБОТКА ДАТАЛОГИЧЕСКОЙ МОДЕЛИ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ.....	6
1. Цель работы.....	6
2. Теоретические положения.....	6
3. Порядок выполнения работы .....	12
4. Задания для выполнения.....	12
5. Контрольные вопросы.....	16
6. Литература.....	16
ЛАБОРАТОРНАЯ РАБОТА 2. СОЗДАНИЕ ТАБЛИЦ СРЕДСТВАМИ ЯЗЫКА SQL.....	18
1. Цель работы.....	18
2. Теоретические положения .....	18
3. Порядок выполнения и задание на лабораторную работу	24
4. Контрольные вопросы.....	24
5. Литература.....	25
ЛАБОРАТОРНАЯ РАБОТА 3. ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ.....	26
1. Цель работы.....	26
2. Теоретические положения .....	26
3. Порядок выполнения и задание на лабораторную работу	38
4. Контрольные вопросы.....	38
5. Литература.....	39
ЛАБОРАТОРНАЯ РАБОТА 4. ВЫБОРКА ДАННЫХ ИЗ СВЯЗАННЫХ ТАБЛИЦ.....	40
1. Цель работы.....	40
2. Теоретические положения .....	40
3. Порядок выполнения и задание для лабораторной работы	42
4. Контрольные вопросы.....	43
5. Литература.....	44
ЛАБОРАТОРНАЯ РАБОТА 5. ИСПОЛЬЗОВАНИЕ ОКОННЫХ ФУНКЦИЙ .....	45

1. Цель работы.....	45
2. Теоретические положения. ....	45
3. Задание для выполнения .....	51
4. Литература.....	51
ЛАБОРАТОРНАЯ РАБОТА 6. НОРМАЛИЗАЦИЯ РЕЛЯЦИОННЫХ ОТНОШЕНИЙ .....	53
1. Цель работы.....	53
2. Теоретические положения .....	53
3. Порядок выполнения работы .....	59
4. Варианты заданий.....	59
5. Литература.....	62
ЛАБОРАТОРНАЯ РАБОТА 7. МЕХАНИЗМЫ РАБОТЫ С БД. ОГРАНИЧЕНИЯ.....	63
1. Цель и задачи работы .....	63
2. Теоретические сведения .....	63
3. Задание для выполнения .....	65
4. Контрольные вопросы.....	66
5. Требования к отчёту .....	66
6. Литература.....	66
ЛАБОРАТОРНАЯ РАБОТА 8. МЕХАНИЗМЫ РАБОТЫ С БАЗОЙ ДАННЫХ. ИСПОЛЬЗОВАНИЕ ХРАНИМЫХ ПРОЦЕДУР И ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИЙ. ....	67
1. Цель работы.....	67
2. Теоретические положения .....	67
3. Порядок выполнения работы .....	74
4. Литература.....	75
ЛАБОРАТОРНАЯ РАБОТА 9. ОРГАНИЗАЦИЯ БИЗНЕС- ЛОГИКИ ИС НА СТОРОНЕ СЕРВЕРА. ИСПОЛЬЗОВАНИЕ ТРИГГЕРОВ. ....	76
1. Цель работы.....	76
2. Теоретические положения. ....	76
3. Задание и порядок выполнения.....	81
4. Литература.....	82
ЛАБОРАТОРНАЯ РАБОТА 10. ИСПОЛЬЗОВАНИЕ КУРСОРОВ .....	83
1. Цель работы.....	83
2. Теоретические положения .....	83

3. Порядок выполнения и задания для работы .....	92
ПРИЛОЖЕНИЕ.....	94
Простейшие операторы манипуляции данными .....	94
Стандартные функции SQL Server .....	94
Пример отчета по лабораторной №1 .....	97
Пример выполнения Лабораторной работы №5 .....	100
Работа с курсорами. Пример сведения нескольких записей в одну с помощью курсора.....	103

# ЛАБОРАТОРНАЯ РАБОТА 1.

## ЛАБОРАТОРНАЯ РАБОТА 1. ИЗУЧЕНИЕ СРЕДЫ РАБОТЫ С СУБД SQL SERVER. РАЗРАБОТКА ДАТАЛОГИЧЕСКОЙ МОДЕЛИ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

### 1. ЦЕЛЬ РАБОТЫ

Получить навыки разработки баз данных в среде MS SQL SERVER Management Studio 2008 (2012).

В связи с этим задачами работы является изучение архитектуры СУБД MS SQL SERVER.

- Знакомство с принципом работы в среде MS SQL SERVER Management Studio 2008 (2012)
- Изучение принципов создания модели базы данных на основе анализа и выявления объектов предметной области.
- Создание базы данных в соответствии с индивидуальным заданием.

### 2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ.

Базовым элементом баз данных, построенных на основе реляционной модели, является отношение. Отношение реализуется в среде различных СУБД как таблица (0).

Таким образом, *таблица* это объект, предназначенный для хранения информации в реляционной БД.

Информация о единичном экземпляре данных в таблице представляется как *запись (кортеж)* или строка.

*Поля (атрибуты)* объекта представляются как – столбцы в табличном виде.

Поля в реляционных базах данных характеризуются следующими свойствами:

1. *Имя поля* – идентификатор поля, по которому организуется программный доступ к нему.
2. *Тип поля* – тип данных, находящихся в этом поле.

Примеры типов:



Рисунок 1.1. Таблица – основной элемент базы данных.

3. *Размер поля* - величина в байтах, выделяемая для хранения данных в поле. Например: если тип поля **СТРОКОВЫЙ**, а размер будет равен 10-ти, то это значит, что в ячейку такого поля нельзя будет записать строку более 10 символов. Если задать **ЦЕЛЫЙ ЧИСЛОВОЙ** тип и установить размер в 4 байта, то числа в ячейке будут принимать значения от 0 до 65535

4. *Инкрементность (счетчик)* – автозаполнение поля в добавленной записи неким значением (как правило числового целого типа).

5. *Ключ* – уникальный идентификатор, характеризующий запись.

6. *Необходимость заполнения* – если поле не обязательно для заполнения, то при добавлении записи (в случае отсутствия данных в поле) оно автоматически заполняется значением по умолчанию, если таковое имеется. Если значения по умолчанию нет, записывается псевдопустое значение “NULL”, которое определено в системе специальным идентификатором.

## Microsoft SQL Server

Microsoft SQL Server [1] — это масштабируемая высокопроизводительная система управления базами данных (СУБД) для платформ на базе Windows. Она разработана с учетом требований к современным распределенным клиент-серверным вычислениям и тесно интегрирована с серверными продуктами семейства Microsoft Office.

Включает в себя библиотеки и службы ядра сервера СУБД. Служба MSSQLSERVER представляет собой движок СУБД, обрабатывающий все запросы, приходящие на сервер.

В стандартный пакет Microsoft SQL Server входят несколько приложений, служащих для администрирования и разработки клиент-серверных приложений:

Для разработки таблиц и серверных механизмов используется приложение MS SQL SERVER Management Studio (MSSMS) различных версий.

При запуске приложения открывается окно соединения приложения с сервером. Приложение можно использовать для работы серверами, установленными независимо от MS SQL SERVER Management Studio.

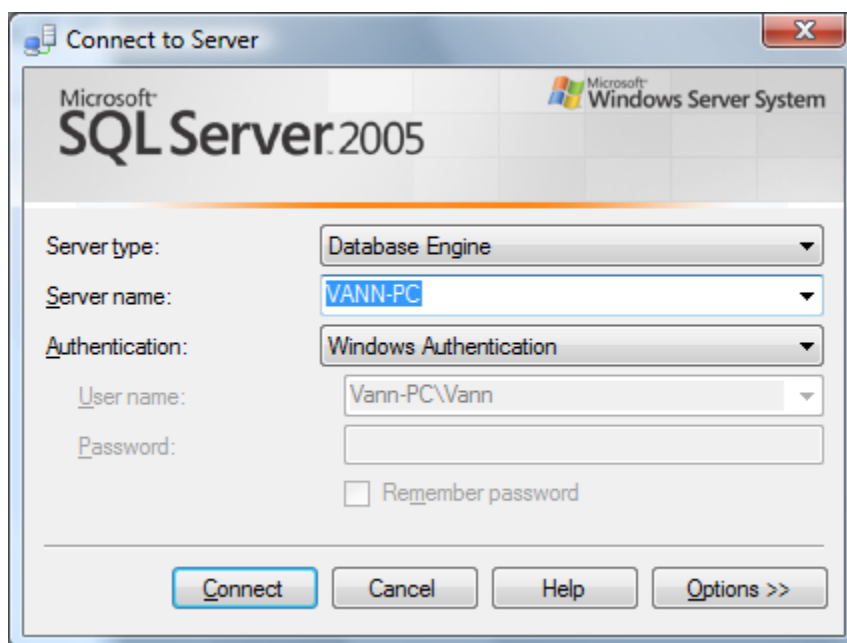


Рисунок 1.2. Окно соединения с сервером.

Для соединения с сервером необходимо знать его имя, имя



записи, зарегистрированной на сервере и пароль для этой записи. Если используется авторизация на основе учетной записи Windows, данная учетная запись должна быть зарегистрирована на сервере БД

После соединения с сервером открывается окно приложения MS SQL SERVER Management Studio (Рисунок 1.2).

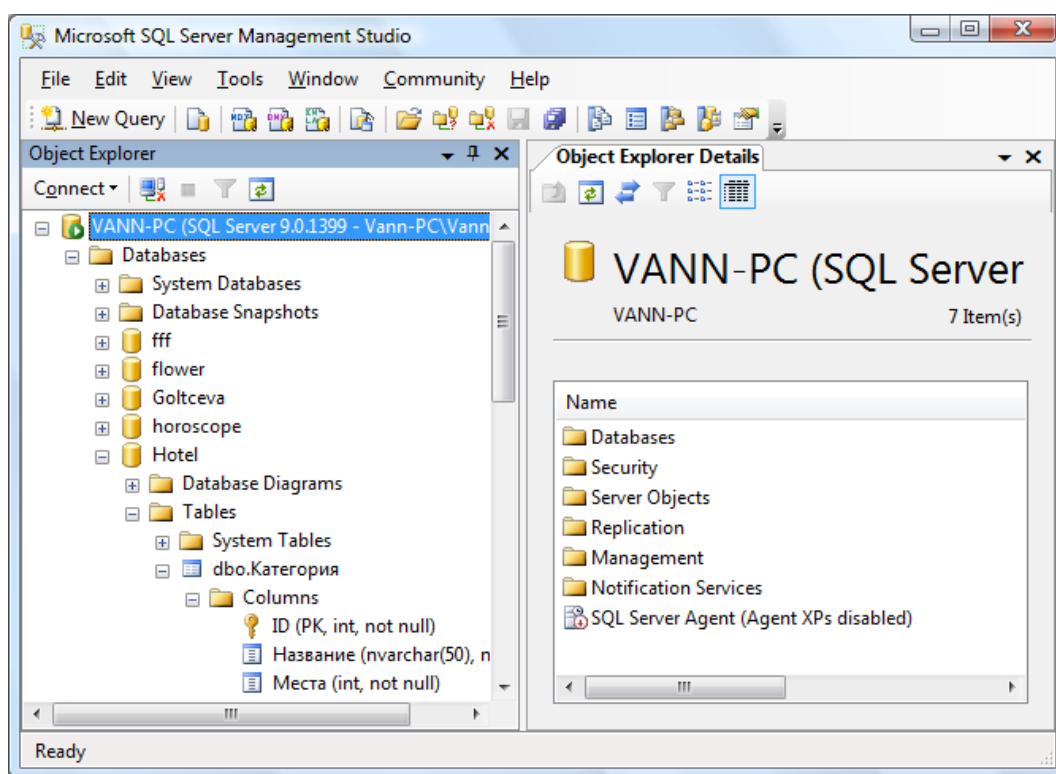


Рисунок 1.3. Рабочее окно MS SQL SERVER Management Studio 2005, 2008

Левую часть окна занимает рабочее окно обозревателя объектов сервера. Объекты сервера представлены в виде древовидной структуры. Корнем дерева является соединение. Management Studio может быть одновременно соединено с несколькими серверами. Работа с любыми объектами сервера может осуществляться через контекстное меню на соответствующем узле дерева (Рисунок 1.4).

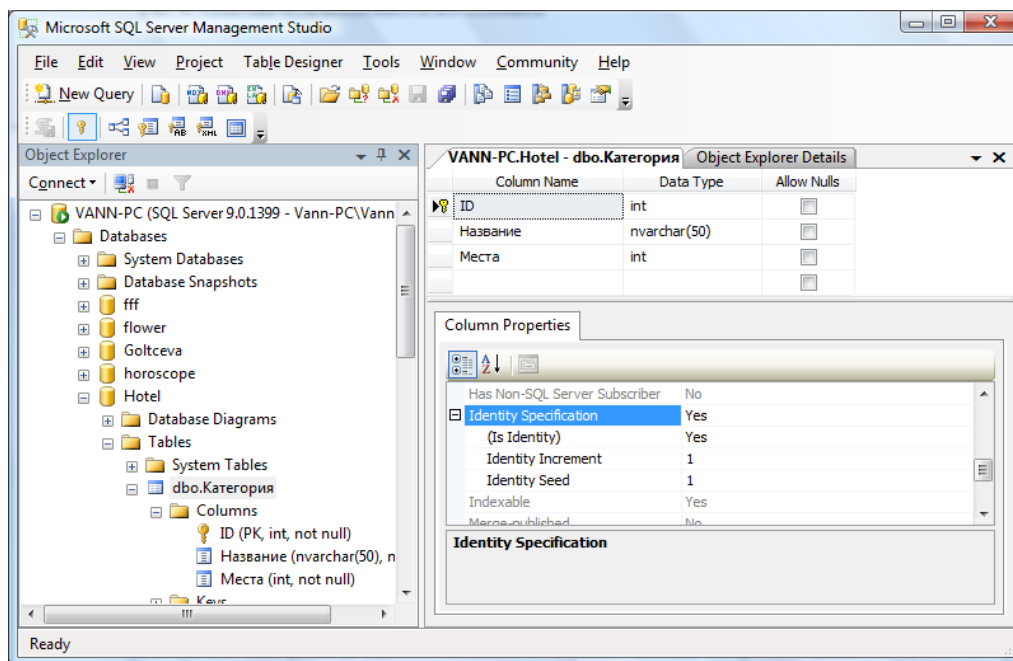


Рисунок 1.4. Работа с таблицей в режиме её модификации.

**Выявление отношений в базе данных.** Обычно качестве отношений реляционной базы данных отображаются объекты предметной области, обеспечивающие получение информации, определенной в требованиях к системе.

Для выявления объектов предметной области необходимо её проанализировать и выявить сущности, обладающие свойствами, на основе которых может быть получена информация, определённая в требованиях к системе для которой проектируется база данных. Состав объектов должен быть достаточным, но не избыточным. Обычно выделяются объекты оперативные и справочные.

Оперативные объекты содержат некоторую текущую информацию, они часто обновляются. Это могут быть данные о единичной покупке, например:

- *покупка(Дата, Покупатель, КодТовара, количество, цена)*

Справочные объекты содержат информацию которая может использоваться в качестве значений атрибутов для оперативных объектов. Например данные о товаре

*ДанныеО\_Товаре(КодТовара, Наименование, цена, произ-*

водитель)

Атрибуты справочных таблиц могут определяться значениями, других справочных таблиц, например атрибут производитель в таблице *ДанныеО\_Товаре* может определяться значениями таблицы *данныеО\_Производителе*(*Наименование*, *номерСчёта*, *юрАдрес*)

Для выявленных отношений устанавливаются атрибуты и требования к ним.

Для каждого отношения необходимо сформулировать бизнес – правила соответствующей предметной области.

Бизнес – правила характеризуют поведение объекта в предметной области, значение его атрибутов.

Необходимо проанализировать атрибуты, выявленные для отношений, на предмет их атомарности. Не атомарный атрибут подразумевает некоторое множество составных атрибутов, а следовательно его можно представить в виде другого отношения.

*Например:*

*Студент – объект, выполняющий обучение на предметах.*

*Характеризуется:*

*фамилией, именем, отчеством (отдельные атрибуты типа строка);*

*номером зачетной книжки (атрибут целого типа).*

*Студент обучается на учебном курсе (учебный курс – это отдельное отношение, так как может иметь свои характеристики).*

Для выявленных объектов и их атрибутов необходимо выявить бизнес правила, определяющие требования целостности сущности, то есть обязательность значения данного атрибута, уникальность значения данного атрибута, его допустимые значения.

*Например:*

*Фамилия студента состоит из символов, это обязательный атрибут.*

*Номер зачетной книжки – число, минимальное значение - 10000,*

*Максимальное 99999.*

Для выявленных отношений необходимо определить бизнес-правила их функционирования в предметной области определяющие их с другими отношениями

В бизнес-правилах, характеризующих связи должна быть дана следующая информация:

- содержания связи;
- множественность связи с одной и другой стороны;
- обязательности и дополнительных ограничений, ограничений накладываемых на связь.

Например отношение Покупка связано с отношением товар так как покупка должна всегда содержать товар. Данная связь имеет множественность «один к многим», так как одна покупка может содержать много товаров.

Каждое выявленное бизнес-правило реализуются в виде фрагмента ER диаграммы.

### 3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Выделить отношения согласно заданию, описать отношения, и их атрибуты.

2. Описать связи между отношениями с точки зрения их множественности с одной и другой стороны, обязательности, ответственности бизнес правилу предметной области.

3. Построить в среде SQL server Manadgment Sydudio таблицы в соответствии с заданием.

4. построить диаграмму отношений в среде SQL server Manadgment Sydudio.

5. Произвести заполнение отношений тестовыми данными.

### 4. ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ.

В качестве заданий выдается примерная формулировка темы для курса лабораторных работ, в результате которых должна быть создана база данных архитектуры «сервер баз данных», то есть база данных должна быть дополнена серверными механизмами для работы с ней. База данных созданная на курсе лабора-

торных работ должна содержать не менее 4 таблиц.

Для задания типов атрибутов в этой работе необходимо использовать.

Для задания целочисленных значений – тип `int`

Для задания значений подразумевающих возможность дробных значений – `float`

Для задания текстовых данных `nvarchar(n)`, где `n` – максимально - возможная длина строки.

Для задания даты и времени `DateTime`

Примеры вариантов заданий.

1. Разработка информационной системы обеспечения хранения, накопления и выборки данных о рейсах междугородних автобусов автовокзала

2. Разработка информационной системы обеспечения хранения, накопления и выборки данных об охотничьих угодьях Кемеровской области, их ресурсах и выдаче лицензий на охоту.

3. Разработка информационной системы обеспечения хранения, накопления и выборки данных о садовых участках кемеровского района, расположении, владельцах, данные об участке, наименование кооператива, председатель кооператива.

4. Разработка информационной системы обеспечения хранения, накопления и выборки данных об аппаратном обеспечении персональных компьютеров и его поставщиков.

5. Разработка информационной системы обеспечения хранения, накопления и выборки данных о цветах, букетах цветочного магазина.

6. Разработка информационной системы обеспечения хранения, накопления и выборки данных о студентах, учебных группах, успеваемости (база деканат).

7. Разработка информационной системы обеспечения хранения, накопления и выборки данных о состоянии сданной в ремонт компьютерной техники.

8. Разработка информационной системы обеспечения хранения, накопления и выборки данных об алкогольной продукции, продаваемой в Кемеровской области.

9. Разработка информационной системы обеспечения хранения, накопления и выборки данных о тарифах и услугах сото-

вых операторов.

10. Разработка информационной системы обеспечения хранения, накопления и выборки данных о продажах сотовых телефонов в магазине сотовой связи с учетом остатков, типа продаж.

11. Разработка информационной системы обеспечения хранения, накопления и выборки данных о зоологических особенностях животных.

12. Разработка информационной системы обеспечения хранения, накопления и выборки данных о составе и комплектации компьютерного оборудования.

13. Разработка информационной системы обеспечения хранения, накопления и выборки данных о деятельности гостиницы. Клиенты, номера, проживание клиентов в номерах.

14. Разработка информационной системы обеспечения хранения накопление выборки данных материального обеспечения учебного процесса кафедры «Прикладная механика» КузГТУ.

15. Разработка информационной системы обеспечения хранения накопление выборку данных об индивидуальных прогнозах личностей (гороскоп).

16. Разработка информационной системы обеспечения хранения накопление выборку данных о музыкальных направлениях, и произведениях

17. Разработка ИС обеспечения хранения накопление выборки данных о Интернет-провайдерах, их услугах и пользователях.

18. Разработка ИС обеспечения хранения накопление выборки данных маршрутах средств общественного транспорта.

19. Разработка ИС обеспечения хранения накопление выборки данных о поставке, лицах, осуществляющих поставку и затратах на разгрузку товара в торговый комплекс.

20. Разработка ИС обеспечения хранения накопление выборки данных о соревнованиях Формула 1.

21. Разработка ИС обеспечения хранения накопление выборки данных об анкетировании студентов.

22. Разработка ИС обеспечения хранения накопление выборки данных о выполнении графика подготовки спортсменов - лыжников к соревнованиям.

23. Разработка ИС обеспечения хранения, накопления и вы-

борки данных обеспечивающих работу агентства недвижимости.

24.Разработка ИС обеспечения хранения накопление выборки данных об игроках в футбол команд высшей и первой лиги.

25.Разработка ИС обеспечения хранения накопление выборки данных о лекарственных средствах, имеющихся в наличии в аптеках города.

26.Разработка ИС обеспечения хранения накопление выборки данных обеспечивающих работу автомагазина.

27.Разработка ИС обеспечения хранения накопление выборки данных об иероглифах и их сочетаниях китайского языка (китайский словарь)

28.Разработка ИС обеспечения хранения накопление выборки данных о музыкальных магазинах г. Кемерово, наличии в них аудио, видео дисков, их содержании и исполнителях.

29.Разработка ИС обеспечения хранения накопление выборки данных о странах мира, их основных характеристиках, граничащих странах.

30.Разработка ИС обеспечения хранения, накопление и выборки данных о результатах игр сезона по футболу.

31.Разработка ИС обеспечения хранения, накопление и выборки данных об игровом компьютерном клубе: игроки, игры, результаты.

32.Разработка ИС обеспечения хранения, накопление и выборки данных об оборудовании на угледобывающем карьере. Его характеристиках и размещении, состоянии на определённую дату.

33.Разработка ИС обеспечения хранения, накопление и выборки данных о заселении студентов в общежитие.

34.Разработка ИС обеспечения хранения накопление выборки данных о делах, ведомых в ГУВД, фигурантах дел.

35.Разработка ИС обеспечения хранения накопление выборки данных об авто-аксессуарах, продаваемы в магазине.

36.Разработка ИС обеспечения хранения накопление выборки данных обеспечивающих работу автопредприятия, тип транспортного средства, грузоподъемность, состояние.

37.Разработка ИС обеспечения хранения, накопление и выборки данных о начислении зарплаты работникам предприятия.

Работник. Дата. Начислено. Необходимо данные об отделах, в которых работают работники.

38.Разработка ИС обеспечения хранения, накопление и выборки данных о кулинарных рецептах.

39.Разработка ИС обеспечения хранения накопление выборки данных высаженных культурах, исполнителях, проведенных работах, истории посадок на садовом участке.

40.Разработка ИС обеспечения хранения накопление выборки данных о нотных записях и текстах музыкальных произведений.

41.Разработка ИС обеспечения хранения накопление выборки данных о чемпионате России по баскетболу.

42.Разработка ИС обеспечения хранения, накопление и выборки данных о репертуаре театра на сезон.

43.Разработка ИС обеспечения хранения, накопление и выборки данных о данных соревнованиях по велоспорту.

44.Разработка ИС обеспечения хранения, накопление и выборки данных о товарах ружейного магазина (характеристики оружия, боеприпасы, аксессуары)

## 5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое реляционное отношение?
2. Что такое атрибут отношения?
3. Что такое кортеж реляционного отношения?
4. Фундаментальные свойства реляционных отношений?
5. Что такое первичный ключ отношения?
6. Что такое внешний ключ отношения?
7. Архитектура СУБД MsSQLServer, какие еще типы архитектур СУБД существуют?
8. Какие компоненты существуют для работы с базами данных под управлением SQL server ?

## 6. ЛИТЕРАТУРА

1. Что такое SQL Server Management Studio (SSMS)?  
<https://learn.microsoft.com/ru-ru/sql/ssms/sql-server-management->



studio-ssms?view=sql-server-ver16

## 2. Скачивание SQL Server Management Studio (SSMS)

<https://learn.microsoft.com/ru-ru/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16>

## **ЛАБОРАТОРНАЯ РАБОТА 2. СОЗДАНИЕ ТАБЛИЦ СРЕДСТВАМИ ЯЗЫКА SQL**

### **1. ЦЕЛЬ РАБОТЫ**

Получить практический навык создания заполнения объектов БД средствами языка SQL .

В связи с этим задачами работы являются

- Изучение основных операторов языка, определения и манипулирования данными
- Создание и выполнение запросов в среде Management Studio, обеспечивающих создание и заполнение таблиц данными.

### **2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ**

SQL (Structured Query Language, структурированный язык запросов) — это язык программирования, предназначенный для выборки и обработки информации, содержащейся в реляционной базе данных. SQL является стандартным языком для работы с реляционными базами данных, его основа реляционная алгебра и реляционное исчисление. SQL содержит набор стандартных операторов доступа к данным и манипулирования [1]

Существуют следующие версии SQL:

- SQL1 (принята в 1986 году, дополнена в 1989 году стандарт ANSI) ,
- SQL2 (SQL-92 принята в 1992 году),
- SQL3 (SQL-99) расширяет SQL2 за счет включения поддержки некоторых объектно-реляционных возможностей.
- SQL2003 введены возможности работы с XML данными.
- SQL2006 расширены возможности работы с XML данными.
- SQL2008 улучшена возможности оконных функций.
- SQL 2011 поддержка хронологических баз данных конструкции fetch, SQL 2016.

Существуют версии SQL предлагаемые основными поставщиками СУБД, они, как правило, удовлетворяют требованиям ANSI, реализуют многие возможности SQL2 и имеют некоторые

особенности.

SQL – это слабо структурированный язык, особенно, по сравнению с такими высокоструктурированными языками, как C, Pascal или Java. В нем нет инструкции IF..THEN для проверки условий, нет инструкции GOTO для организации переходов и нет инструкций DO или FOR для создания циклов.

SQL обеспечивает независимость от конкретных СУБД: реляционную базу данных и программы, которые с ней работают, можно перенести с одной СУБД на другую с минимальными доработками и переподготовкой персонала. Все ведущие поставщики СУБД используют SQL. Однако поставщики СУБД предлагают различные диалекты SQL, имеющие некоторые отличия. Например, СУБД Oracle использует PL/SQL MS SQLSERVER использует Transact-SQL. В этих диалектах стандартный SQL дополнен инструкциями IF..THEN, GOTO и др., однако эти диалекты не получили статус стандарта и являются частными разработками отдельных компаний (PL/SQL применяется в СУБД Oracle, а Transact SQL - в СУБД MS SQL Server). В данной работе будет рассматриваться Transact SQL [3]

SQL не является отдельным программным продуктом. SQL – это неотъемлемая часть СУБД ее *Манипуляционная часть*, инструмент, с помощью которого осуществляется связь пользователя с БД.

### **Типы данных В SQL**

Типы данных в SQL Server объединены в следующие категории. Подробное описание их приведено в [1]

- Точные числа
- Символьные строки в Юникоде
- Приблизительные числа
- Двоичные данные
- Дата и время
- Прочие типы данных
- Символьные строки

#### **Точные числа**

Целые (int (4 байта) bigint(8 байт), smallint (2 байта), tinyint (1 байт))

Числа с фиксированным количеством десятичных разрядов

*p* (масштабом) и количеством знаков после запятой (точностью) - *s* (decimal ( **decimal**[ (*p*[ ,*s* ] )] и **numeric**[ (*p*[ ,*s* ] )] )

Тип для задания логических данных bit (1 байт - 0,1,null)

Денежный тип (может использоваться денежный знак \$ и другие) ( money (8 байт), smallmoney (4 байта) )

### **Приблизительные числа**

float(*n*) – в зависимости от *n* 4 байта (*n* =1-24) или 8 , real

### **Дата и время**

date

datetimeoffset

datetime2

smalldatetime

datetime time

### **Символьные строки**

Фиксированной длины - char (*n*) – *n* - длина строки.

Переменной длины varchar(*n*) - *n* – максимальная длина строки.

### **Символьные строки в Юникоде**

фиксированной длины - nchar(*n*)

Переменной длины nvarchar(*n*)

### **Двоичные данные (binary, varbinary, image)**

### **Прочие типы данных**

Значение NULL и его применение.

Атрибутам отношения или переменным SQL допускает присвоение специального значения NULL.

Значение NULL имеет следующий смысл:

- значение не известно, то есть когда создается новый кортеж, а значение некоторого атрибута явно не задается и не задано по умолчанию, то ему присваивается данное значение;
- значение не может быть задано, то есть когда значения некоторого атрибута быть не может (например атрибут Супруг, для некоторого кортежа отношения Сотрудники, когда рассматриваемый сотрудник не женат);
- значение умалчивается, то есть, если значение атрибута, выдается по запросу, но данный атрибут запрещен для просмотра для данного источника запроса.

Если атрибут или выражение со значением NULL участвует

в арифметической операции, то результат операции будет иметь значение NULL.

При сравнении выражения, имеющего значение NULL с другим выражением с помощью операций сравнения (=, !=, <>, <, >, >=, <=, !=>, !=<) результат будет иметь значение *unknown*.

Для проверки выражения на значение NULL операция сравнения не используется. Для Этого необходимо использовать специальный предикат IS NULL (IS NOT NULL), он будет рассмотрен ниже.

### **Структура языка SQL**

Различают несколько групп операторов (подязыки):

I. Язык определения данных DDL.

К языку запросов относятся операторы

CREATE TABLE - создания нового отношения;

DROP TABLE - удаление отношения;

ALTER TABLE - изменение структуры таблицы;

CREATE VIEW - создания представления;

DROP VIEW - удаления представления;

CREATE INDEX - удаление индексов.

II. Язык манипулирования данными DM (команды, DELETE, INSERT, UPDATE )

III. Язык запросов DQL (оператор SELECT)

IV. Средства управления транзакциями.

V. Средства администрирования данными.

**Операторы языка определения данных ddl и модификации отношений.**

*Оператор создания таблицы (задания схемы отношения).*

Оператор создаёт таблицу и столбцы таблицы. Или говоря терминологией реляционных отношений задаёт схему отношения. То есть само отношение и его атрибуты.

Общий формат оператора

Create table ИмяТаб(Аtr1 ТипАtr [ЗначПоУмолч] [ОгрАtr]  
[,Аtr2 ...]

.....

ОгрКортежа), ГДЕ

- Аtr1, Аtr2 ... - идентификаторы (имена) атрибутов отно-

шения;

- *ЗначПоУмолч* – значение, присваиваемое атрибуту по умолчанию;
- *ОгрАтр* – ограничения на значение атрибута (будут рассмотрены позже ;
- *ОгрКортежа* - ограничения на значение кортежа (будут рассмотрены позже ;

Например

создание отношения Pers(kodPers int, FamPers nvarchar(15))

```
create table Pers(kodPers int primary key, FamPers nvarchar(15))
```

### **Оператор удаление отношения**

*Drop table ИмяОтн*

### **Модификация отношений.**

Модификация отношения может быть следующих разновидностей.

Удаление атрибута -

*Alter table ИмяОтн drop ИмяАмп1* – из отношения ОТН будет удалён атрибут именем *ИмяАмп1*;

□□□□ Вставка атрибута -

*Alter table ИмяОтн add ИмяАмп1 типАмп1 ЗначПоУмолч]..*,

где *ИмяАмп1 типАмп1 [ЗначПоУмолч]..* – описание атрибута, аналогичное используемому в операторе *Create table*.

### **Операторы SQL манипулирования данными.**

Операторы данной группы позволяют модифицировать существующие кортежи отношений. То есть, вставлять новые кортежи отношения, удалять, изменять значений атрибутов.

### **Вставка кортежей (INSERT).**

*insert into ИмяОтн(ИмяАмп1, ИмяАмп2, ..) values(знач1, знач2,...)*

В результате выполнения данной команды в отношение с именем *ИмяОтн* будет вставлен кортеж, при этом атрибутам с именами *ИмяАмп1, ИмяАмп2...* будут присвоены значения *знач1, знач2,...*. Атрибутам, не перечисленным в списке, будет присвоено значение по умолчанию. Если значения по умолчанию не заданы, то система попытается присвоить им значения NULL.

### Удаление кортежей

`delete from имяОтн where условие`

При выполнении данной команды из отношения с именем *имяОтн* будут удалены кортежи, значение атрибутов которых будет соответствовать условию.

### Модификация (обновление) кортежей.

`update отн set ИмяАтриб1=знач1, ИмяАтриб1=знач1 [...]  
where условие`

При выполнении данной команды кортежам, отвечающим заданным условиям будет изменено значение заданных атрибутов.

Изменение атрибутов будет отменено, если они противоречат условиям целостности базы данных, или другим ограничениям.

Для работы с объектами СУБД MS SQL Server Management Studio предоставляет набор команд контекстного меню, в частности для работы с таблицам может использоваться иерархия контекстных меню, показанная на рисунке (Рисунок 2.1).

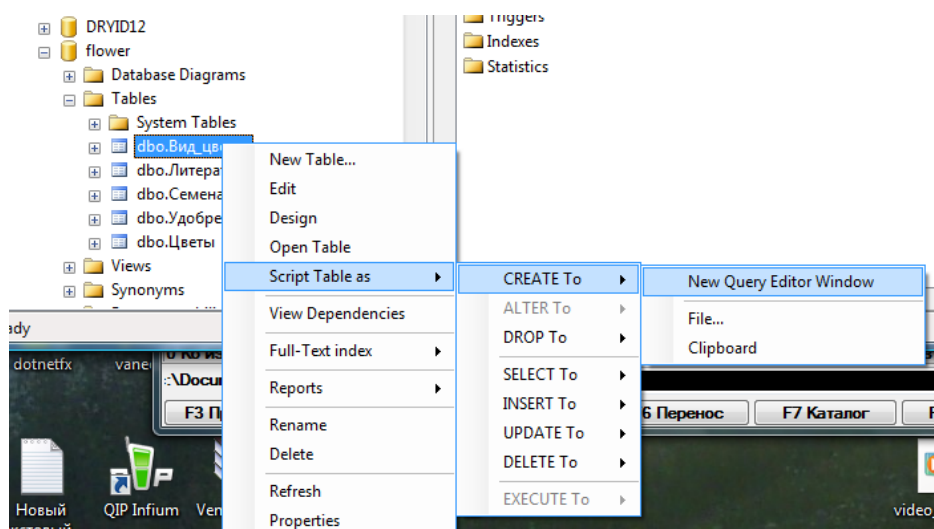


Рисунок 2.1. Иерархия команд контекстного меню для работы с таблицами

Команда меню *Script table as* позволяет создавать шаблоны команд для создания, удаления, вставки кортежей и для выполнения других действий с таблицами.

Созданные команды можно сохранять во внешних файлах, с

расширением SQL.

### 3. ПОРЯДОК ВЫПОЛНЕНИЯ И ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Заданием на лабораторную работу является формирование команд SQL, обеспечивающих создание таблиц, аналогичных полученным в результате выполнения лабораторной работы №1 и заполнение этих таблиц данными.

Примерный порядок выполнения работы может быть следующим.

1. Написать запросы для создания таблиц в соответствии с заданием лабораторной работы 1. Имена таблиц создаваемых командами должны отличаться от имён таблиц, созданных в лабораторной 1, дополнительным символом. Например подчеркиком. Например – исходная таблица pers. Таблица созданная командой pers\_.

2. Создать запросы для вставки в таблицы значений.

3. Сформировать два файла запросов, первый содержит команды по созданию таблиц, второй по заполнению.

4. Создать файл, содержащий команды по удалению созданных таблиц.

5. Выполнить запросы создания, заполнения. Показать преподавателю результаты. Выполнить запрос по удалению. Показать преподавателю результаты.

6. Подготовить отчет. В отчет включить описание процесса создания запросов и тексты запросов и скриншоты отображающие структуру и состав полученных таблиц.

### 4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Основные операторы языка определения данных?

2. Формат оператора SQL, используемого для создания отношений?

3. Какие операторы SQL позволяют менять состав атрибутов отношений?

4. Основные операторы языка модификации отношений?



5. Каким образом с помощью операторов SQL изменить тип атрибута в заполненной таблице?

## 5. ЛИТЕРАТУРА

1. SQL <https://ru.wikipedia.org/wiki/SQL>
2. Справочник по Transact-SQL (ядро СУБД)  
<https://learn.microsoft.com/ru-ru/sql/t-sql/language-reference?view=sql-server-ver16>
3. Инструкция CREATE TABLE (Transact-SQL)  
<https://learn.microsoft.com/ru-ru/sql/t-sql/statements/create-table-transact-sql?view=sql-server-ver16>
4. Инструкция INSERT (Transact-SQL)  
<https://learn.microsoft.com/ru-ru/sql/t-sql/statements/insert-transact-sql?view=sql-server-ver16>
5. Типы данных (Transact-SQL) <https://learn.microsoft.com/ru-ru/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15>
6. Типы данных и функции даты и времени (Transact-SQL)  
<https://learn.microsoft.com/ru-ru/sql/t-sql/functions/date-and-time-data-types-and-functions-transact-sql?view=sql-server-ver16>

## **ЛАБОРАТОРНАЯ РАБОТА 3. ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ**

### **1. ЦЕЛЬ РАБОТЫ**

Освоить принципы выборки информации из таблиц реляционных баз данных.

Задачами работы в связи с этим является.

- Изучение конструкций языка SQL, обеспечивающих выборку данных. Параметров, используемых для приведения результатов выборки к требуемому виду.
- Построение запросов по выборке данных из таблиц, сформированных и заполненных согласно индивидуального задания.

### **2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ**

#### **Оператор SELECT.**

Оператор SELECT осуществляет выборку кортежей из базы данных и имеет наиболее сложную структуру среди всех операторов языка SQL.

Общий формат

SELECT [ALL/DISTINCT] СписокПолей FROM имяТаблицы  
[WHERE предикатВыбора]  
[GROUP BY СписокПолейГруппирования]  
[HAVING условиеГруппирования]  
[ORDER BY списокПолейУпорядочивания]

где,

ALL/DISTINCT - вывод всех кортежей, попавших в выборку (ALL)/ или только не дублирующихся (DISTINCT)

СписокПолей – выводимые поля, разделитель – запятая, если указан знак «\*» - выводятся все поля;

предикатВыбора – условия выбора кортежей, составляется из операций сравнения на основе атрибутов, соединённых логическими операциями.

Простейший пример

SELECT \* FROM PC

Осуществляется выборка всех записей из объекта БД табличного типа с именем PC. При этом выведутся все поля таблицы, в порядке, определяемом СУБД. Чтобы указать в каком порядке следует выводить атрибуты, их следует перечислить через запятую в нужном порядке, например:

```
SELECT price as Цена, speed, hd, ram as Память, cd, model,
code
FROM PC;
```

Для поля может быть указан синоним с помощью ключевого слова as. В данном случае имя столбца Price будет отображено как «Цена»

Ниже приводится результат выполнения этого запроса.

Цена	speed	hd	Память	cd	model	code
600.0	500	5	64	12x	1232	1
850.0	750	14	128	40x	1121	2
600.0	500	5	64	12x	1233	3
850.0	600	14	128	40x	1121	4
850.0	600	8	128	40x	1121	5

Рисунок 3.1. Результат выполнения запроса с выводом всех атрибутов

Вертикальную проекцию таблицы PC можно получить, если перечислить только необходимые поля. Например, чтобы получить информацию только о скорости процессора и объеме оперативной памяти компьютеров, следует выполнить запрос:

```
SELECT speed, ram FROM PC
```

Запрос вернёт данные в следующем виде (Рисунок 3.2):

speed	ram
-------	-----

500	64
750	128
500	64
600	128
600	128

Рисунок 3.2. Запрос с использованием проекции на заданные атрибуты (вывод части атрибутов)

*Обратите внимание в результирующем наборе существуют одинаковые кортежи, они возникают вследствие того, что происходит выборка не всех столбцов, а только заданных. Подавление одинаковых кортежей возможно с помощью слова **DISTINCT**, рассмотренного ниже.*

#### **Ключевое слово *DISTINCT***

Вертикальная выборка может содержать дубликаты строк в том случае, если она не содержит ключа, однозначно определяющего запись. В приведенном выше результирующем наборе имеются дубликаты строк (например, строки 1 и 3). Если требуется получить уникальные строки (скажем, нас интересуют только различные комбинации скорости процессора и объема памяти, а не характеристики всех имеющихся компьютеров), то можно использовать ключевое слово **DISTINCT**:

**SELECT DISTINCT speed, ram FROM Pc**

Запрос вернёт результат (Рисунок 3.3):

speed	ram
500	64

750	128
600	128

Рисунок 3.3. Использование ключевого слова DISTINCT

Помимо DISTINCT может применяться также ключевое слово ALL (все строки), которое принимается по умолчанию.

### *Ключевое слово ORDER BY*

Чтобы упорядочить строки результирующего набора, можно выполнить сортировку по любому количеству полей, указанных в предложении SELECT. Для этого используется предложение ORDER BY СписокПолейУпорядочивания

Данное предложение всегда является последним предложением в операторе SELECT. При этом в списке полей могут указываться как имена полей, так и их порядковые позиции в списке предложения SELECT. Так если требуется упорядочить результирующий набор по объему оперативной памяти в порядке убывания, можно записать

```
SELECT DISTINCT speed, ram FROM Pc ORDER BY ram
DESC
```

Результат, приведенный ниже, будет таким же, как в исходной выборке, так как она отвечает заданным требованиям сортировки.

Сортировку можно проводить по возрастанию (параметр ASC принимается по умолчанию) или по убыванию (параметр DESC).

Сортировка может выполняться по нескольким полям, например по двум, сначала по объёму памяти (убывание), потом, для случая, если объем памяти одинаковый, по скорости (тоже по убыванию):

```
SELECT DISTINCT speed, ram FROM Pc ORDER BY ram
DESC, speed DESC
```

Запрос выдаст следующий результат (Рисунок 3.4):

speed	ram
750	128
600	128
500	64

Рисунок 3.4. Результат запроса с использованием сортировки.

### *Предложение WHERE*

Данное предложение реализует горизонтальную выборку. Общий формат предложения.

WHERE предикатВыбора

Данное предложение записывается после предложения FROM. При этом в результирующий набор попадут только те строки из источника записей, для каждой из которых значение предиката выбора равно TRUE. То есть предикат проверяется для каждой записи. Например, запрос «получить информацию о частоте процессора и объеме оперативной памяти для компьютеров с ценой ниже \$500» можно сформулировать следующим образом:

SELECT DISTINCT speed, ram FROM Pc WHERE price<500  
ORDER BY 2 DESC

speed	ram
450	64
450	32
500	32

Рисунок 3.5. Результат запроса с условием и сортировкой по убыванию.

В последнем запросе использовался предикат сравнения с использованием операции сравнения «<» (меньше чем). Кроме этой операции сравнения могут использоваться: «=» (равно), «>»

(больше), «>=» (больше или равно), «<=» (меньше или равно) и «<>» (не равно). Выражения в предикатах сравнения могут содержать любые поля из таблиц, указанных в предложении FROM. Символьные строки и константы типа дата/время записываются в апострофах.

Примеры простых предикатов сравнения приведены в таблице 3.1.

Таблица 3.1. Примеры простых предикатов сравнения

price < 1000	Цена меньше \$1000.
type = 'laptop'	Типом продукции является ПК-блокнот.
cd = '24x'	24-скоростной CD-ROM.
color <>'y'	Не цветной принтер.
ram - 128 >0	Объем оперативной памяти свыше 128 Mb.
price <=speed*2	Цена не превышает удвоенной частоты процессора.

### *Использование LIKE и знаков подстановки*

Ключевое слово LIKE обеспечивает пользователю гибкость при размещении условий в запросе. При этом можно использовать знаки подстановки. Знак подстановки “%” - означает любые символы. Знак подстановки “\_” – он означает один любой символ.

Пусть необходимо составить запрос с использованием таблицы, из которой необходимо выбрать данные столбцов LAST\_NAME, FIRST\_NAME и STATE (фамилии, имена и информацию о месте проживания всех служащих) фамилия которых начинается на символы «Ст».

```
SELECT LAST_NAME, FIRST_NAME, STATE FROM EMPLOYEE_TBL WHERE LAST_NAME LIKE 'Ст%';
```

## Предикаты

Предикаты представляют собой выражения результатов которых является логическое значение. Они могут представлять собой как одно выражение, так и любую комбинацию из неограниченного количества выражений, построенную с помощью булевых операторов AND, OR или NOT. Кроме того, в этих комбинациях может использоваться SQL-оператор IS, а также круглые скобки для конкретизации порядка выполнения операций.

Предикат в языке SQL может принимать одно из трех логических значений

TRUE (истина),

FALSE (ложь)

UNKNOWN (неизвестно).

### *Предикаты сравнения*

Предикат сравнения представляет собой два выражения, соединяемых оператором сравнения. Имеется шесть традиционных операторов сравнения: =, >, <, >=, <=, <>. Данные типа NUMERIC (числа) сравниваются в соответствии с их алгебраическим значением.

Данные типа CHARACTER STRING (символьные строки) сравниваются в соответствии с их алфавитной последовательностью. Если  $a_1a_2...a_n$  и  $b_1b_2...b_n$  - две последовательности символов, то первая «меньше» второй, если  $a_1 < b_1$ , или  $a_1 = b_1$  и  $a_2 < b_2$  и т.д. Считается также, что  $a_1a_2...a_n < b_1b_2...b_m$ , если  $n < m$  и  $a_1a_2...a_n = b_1b_2...b_n$ , т.е. если первая строка является префиксом второй. Например, 'folder' < 'for', т.к. первые две буквы этих строк совпадают, а третья буква строки 'folder' предшествует третьей букве строки 'for'. Также справедливо неравенство 'bar' < 'barber', поскольку первая строка является префиксом второй.

Данные типа DATETIME (дата/время) сравниваются в хронологическом порядке.

Данные типа INTERVAL (временной интервал) преобразуются в соответствующие типы, а затем сравниваются как обычные числовые значения типа NUMERIC.

Пример. Получить информацию о компьютерах, имеющих частоту процессора не менее 500 Мгц и цену ниже \$800:

```
SELECT * FROM Pc WHERE speed >= 500 AND price < 800;
```



Запрос возвращает следующие данные(Таблица 3.2. ):

Таблица 3.2. Результаты запроса

code	model	speed	ram	hd	cd	price
1	1232	500	64	5	12x	600.0
3	1233	500	64		12x	600.0
7	1232	500	32	10	12x	400.0
10	1260	500	32	10	12x	350.0

### *Предикат BETWEEN*

Предикат BETWEEN проверяет, попадают ли значения проверяемого выражения в диапазон, задаваемый пограничными выражениями, соединяемыми служебным словом AND. Естественно, как и для предиката сравнения, выражения в предикате BETWEEN должны быть совместимы по типам.

Синтаксис BETWEEN

exp1 BETWEEN exp2 AND exp3

*Пример. Пусть требуется найти модель и частоту процессора компьютеров стоимостью от \$400 до \$600:*

```
SELECT      model,      speed      FROM      Pc
WHERE price BETWEEN 400 AND 600
```

Результат выражение приведён в таблице (Таблица 3.3. )

Таблица 3.3. Результат выборки

model	speed
1232	500
1233	500
1232	500

### **Предикат IN**

Предикат IN определяет, будет ли значение проверяемого

выражения обнаружено в наборе значений, который либо явно определен, либо получен с помощью табличного подзапроса. Табличный подзапрос это обычный оператор SELECT, который создает одну или несколько строк для одного столбца, совместимого по типу данных со значением проверяемого выражения. Если целевой объект эквивалентен хотя бы одному из указанных в предложении IN значений, истинностное значение предиката IN будет равно TRUE. Если для каждого значения X в предложении IN целевой объект  $\neq$  X, истинностное значение будет равно FALSE. Если подзапрос выполняется, и результат не содержит ни одной строки (пустая таблица), предикат принимает значение FALSE. Когда не соблюдается ни одно из упомянутых выше условий, значение предиката равно UNKNOWN.

Синтаксис IN

ПроверяемоеВыражение [NOT] IN (Подзапрос)  
|(ВыражениеДляВычисленияЗначения,...)

*Пример. Найти модель, частоту процессора и объем жесткого диска тех компьютеров, которые комплектуются накопителями 10 или 20 Мб:*

SELECT model, speed, hd FROM Pc WHERE hd IN (10, 20);

Таблица 3.4. Результат выборки

model	speed	hd
1233	750	20
1232	500	10
1232	450	10
1260	500	10

*Пример. Найти модель, частоту процессора и объем жесткого диска тех компьютеров, которые комплектуются накопителями 10 или 20 Мб и выпускаются производителем А (Таблица 3.5):*

SELECT model, speed, hd FROM Pc WHERE hd IN (10, 20)  
AND model IN (SELECT model FROM product

WHERE maker = 'A');

Таблица 3.5

model	speed	hd
1233	750	20
1232	500	10
1232	450	10

### Получение итоговых значений. Агрегатные функции.

Агрегатные функции позволяют рассчитывать итоговые показатели по определённому столбцу выборки. Стандартом предусмотрены следующие агрегатные функции(0):

Таблица 3.6 Агрегатные функции

Функция	Описание
COUNT(*)	Возвращает количество строк источника записей.
COUNT(<имя поля>)	Возвращает количество значений в указанном столбце.
SUM(<имя поля>)	Возвращает сумму значений в указанном столбце.
AVG(<имя поля>)	Возвращает среднее значение в указанном столбце.
MIN(<имя поля>)	Возвращает минимальное значение в указанном столбце.
MAX(<имя поля>)	Возвращает максимальное значение в указанном столбце.

Все эти функции возвращают единственное значение. При этом функции COUNT, MIN и MAX применимы к любым типам данных, в то время как SUM и AVG используются только для

числовых полей. Разница между функцией COUNT(\*) и COUNT(ИмяПоля) состоит в том, что вторая при подсчете не учитывает NULL-значения.

*Пример. Найти минимальную и максимальную цену на персональные компьютеры:*

```
SELECT MIN(price) AS Min_price, MAX(price) AS Max_price  
FROM PC;
```

Результатом будет единственная строка, содержащая агрегатные значения:

Min_price	Max_price
350.0	980.0

*Пример. Найти имеющееся в наличии количество компьютеров, выпущенных производителем A:*

```
SELECT COUNT(*) AS Qty FROM PC WHERE model IN  
(SELECT model FROM Product WHERE maker = 'A');
```

В будет следующим:

Qty
7

*Пример. Пусть необходимо найти количество различных моделей, выпускаемых производителем A, то запрос можно сформулировать следующим образом (пользуясь тем фактом, что в таблице Product каждая модель записывается один раз):*

```
SELECT COUNT(model) AS Qty_model FROM Product  
WHERE maker = 'A';
```

Для того, чтобы при получении статистических показателей использовались только уникальные значения, при аргументе агрегатных функций можно использовать параметр DISTINCT. По умолчанию используется параметр ALL он предполагает подсчет всех возвращаемых значений в столбце. Оператор,

```
SELECT COUNT(DISTINCT model) AS Qty  
FROM PC WHERE model IN
```

(SELECT model FROM Product WHERE maker = 'A');  
даст следующий результат:

Qty
2

### Предложение GROUP BY

Предложение GROUP BY используется для определения групп выходных строк, к которым могут применяться агрегатные функции (COUNT, MIN, MAX, AVG и SUM). При использовании GROUP BY в параметрах SELECT могут указываться или поля по которым производится группирование, или агрегатные функции применённые к другим полям.

Простой пример:

```
SELECT model, COUNT(model) AS Qty_model, AVG(price)
AS Avg_price
FROM PC
GROUP BY model;
```

В этом запросе для каждой модели ПК определяется их количество и средняя стоимость. Все строки с одинаковыми значениями model (номер модели) образуют группу, и на выходе SELECT вычисляются количество значений и средние значения цены для каждой группы. Результатом выполнения запроса будет следующая таблица (Таблица 3.7):

Таблица 3.7

model	Qty_model	Avg_price
1121	3	850.0
1232	4	425.0
1233	3	843.33333333333337
1260	1	350.0

### Предложение HAVING

Если предложение WHERE определяет предикат для филь-

трации строк, то предложение HAVING применяется после группировки для определения аналогичного предиката, фильтрующего группы по значениям агрегатных функций. Это предложение необходимо для проверки значений, которые получены с помощью агрегатной функции не из отдельных строк источника записей, определенного в предложении FROM, а из групп таких строк. Поэтому такая проверка не может содержаться в предложении WHERE.

Пример. Получить количество ПК и среднюю цену для каждой модели при условии, что средняя цена менее \$800:

```
SELECT model, COUNT(model) AS Qty_model, AVG(price)
AS Avg_price
FROM PC
GROUP BY model
HAVING AVG(price) < 800;
```

В результате выполнения запроса получим:

model	Qty_model	Avg_price
1232	4	425.0
1260	1	350.0

### 3. ПОРЯДОК ВЫПОЛНЕНИЯ И ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

При выполнении работы необходимо для таблиц созданной базы сделать примеры на каждую функцию, приведенную в теоретическом материале.

В отчете отобразить текст инструкции и результаты ее выполнения в виде копий экрана

### 4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Общий формат оператора Select, его назначение.
2. Каким образом задать требуемые поля для вывода, и необходимый порядок? Назначение ключевого слова Distinct? Когда возникает смысл его использования?

3. Назначение предложения ORDER BY? Какие дополнительные параметры указываются при его использовании?
4. Назначение предложение Where? Как задается предикат выбора?
5. Зачем используется конструкция LIKE? Какие знаки подстановки могут применяться для формирования шаблона сравнения?
6. Что такое предикат в операторе выбора? Какие значения могут принимать предикаты? Какие типы предикатов используются?
7. Какие функции для получения итоговых значений используются в SQL? Особенности этих функций?
8. Назначение предложения GROUP BY? Какие параметры могут использоваться с данным предложением? Как связано использование функций получения итоговых значений с предложением GROUP BY?
9. Назначение предложения HAVING? Совместно с каким предложением оно используется?

## 5. ЛИТЕРАТУРА

1. SQL <https://ru.wikipedia.org/wiki/SQL>
2. Справочник по Transact-SQL (ядро СУБД) <https://learn.microsoft.com/ru-ru/sql/t-sql/language-reference?view=sql-server-ver16>
3. SELECT (Transact-SQL) <https://learn.microsoft.com/ru-ru/sql/t-sql/queries/select-transact-sql?view=sql-server-ver16>
4. Типы данных и функции даты и времени (Transact-SQL) <https://learn.microsoft.com/ru-ru/sql/t-sql/functions/date-and-time-data-types-and-functions-transact-sql?view=sql-server-ver16>

## **ЛАБОРАТОРНАЯ РАБОТА 4. ВЫБОРКА ДАННЫХ ИЗ СВЯЗАННЫХ ТАБЛИЦ**

### **1. ЦЕЛЬ РАБОТЫ**

Получение практических навыков выборки данных из нескольких таблиц с организацией их соединения в запросе выборки. С использованием языка SQL.

Задачами работы в связи с этим является

- Изучение возможных типов связей, реализуемых средствами SQL
- Изучение принципов построения связей между таблицами в запросах SQL
- Построение запросов на выборку с использованием связей между таблицами, согласно заданию работы.

### **2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ**

При выполнении выборки данных из нескольких отношений, необходимо учитывать, каким образом логически организована связь между ними.

Для отображения связей таблиц в предложении FROM может быть указана явная операция соединения двух и более таблиц.

Среди ряда операций соединения, описанных в стандарте языка SQL, многими серверами баз данных поддерживается лишь операция соединения по предикату. Синтаксис соединения по предикату имеет вид:

```
SELECT имяТабл1.ИмяПоле1,..., имяТабл2.ИмяПоле1,..  
FROM   имяТабл1 {[INNER] | {LEFT | RIGHT | FULL }  
[OUTER]} JOIN имяТабл2  
[ON предикат]
```

Соединение может быть

- либо внутренним (INNER используется по умолчанию),
- либо одним из внешних (OUTER).

Служебные слова INNER и OUTER можно опускать, по-



сколько внешнее соединение однозначно определяется его типом:

LEFT (левое),  
RIGHT (правое),  
FULL (полное).

Отдельное слово JOIN (без INNER и OUTER) будет означать внутреннее соединение.

Тип соединения может уточняться некоторыми дополнительными ключевыми словами. Например слово CROSS, указывает на выполнение прямого произведения между кортежами отношений.

Таблица 4.1. Описание соединений, реализуемых в запросах SQL на выборку.

Ключевые слова	Описание реализуемого соединения отношений
CROSS JOIN	Возвращает каждую строку из первой таблицы, объединенную с каждой строкой из второй таблицы. Количество строк результата равно произведению числа строк в каждой таблице
INNER JOIN	Возвращает все строки из каждой таблицы, удовлетворяющие критерию отбора, который задан в предложении WHERE, при условии совпадения значений в объединяемых полях, указанных в предложении ON
LEFT [OUTER] JOIN	Возвращает все строки таблицы, которые находятся слева в объединении, удовлетворяющие критерию отбора предложения WHERE, и только те строки таблицы, расположенные справа в объединении, в которых существует совпадение по объединяемым полям, заданным в предложении ON
RIGHT [OUTER] JOIN	Возвращает все строки таблицы, которые находятся справа в объединении, удовлетворяющие критерию отбора предложения WHERE, и только те строки таблицы, расположенные слева в объ-

	единении, в которых существует совпадение по объединяемым полям, заданным предложением ON
FULL [OUTER] JOIN	Возвращает все строки из каждой таблицы, удовлетворяющие критерию отбора предложения WHERE, в которых нет совпадения по объединяемым полям, заданным предложением ON

Предикат ON определяет условие соединения строк из разных таблиц. При этом INNER JOIN означает, что в результирующий набор попадут только те соединения строк двух таблиц, для которых значение предиката равно TRUE. Как правило, предикат определяет соединение по внешнему и первичному ключам соединяемых таблиц, хотя это не обязательно.

### 3. ПОРЯДОК ВЫПОЛНЕНИЯ И ЗАДАНИЕ ДЛЯ ЛАБОРАТОРНОЙ РАБОТЫ

Выполнить задания лабораторной работы 3 «Извлечение информации из таблиц», но применив задания к нескольким таблицам, соединённым в запросе.

- с join без условия,
- На условие (Where) - в условии соединяются сравнения по атрибутам первой и второй соединяемой таблицы,
- с условием и использованием агрегатной функции (условие по атрибутам одной таблицы, агрегатная функция по атрибутам другой таблицы),
- на группировку (группировка по атрибутам одной таблицы, вывод атрибутов другой таблицы через агрегатную функцию),
- с использованием на Having,
- на предикаты (like, between)

Выполнить запрос RIGHT [OUTER] JOIN, LEFT [OUTER] JOIN, FULL [OUTER] JOIN к какой либо паре логически связанных таблиц без условий.

Дополнительные задания к лабораторной работе.

1. Напишите запрос для выборки данных из связанных таблиц студенты(NZCH, Name), задания (nzch, kodzadachi, DateVydach), задачи(kodzadachi, soderzhane). Запрос должен выводить фамилию студентов, содержание выданных задач, дату выдачи,

2. Напишите запрос для выборки данных из связанных таблиц клиенты(кодКлиента, имяКлиента), заказы(КодЗаказа, кодКлиента, кодУслуги, ДатаЗаказа), услуги(кодУслуги, СодержаниеЕслуги, Стоимость). запрос должен выводить имяКлиента, датуЗаказа, услуги, включенные в заказ.

3. Исходные отношения персона(кодПерс, имя), лечМеропр(КодМеропр,Наименование), выдачаМеропр(кодПерс, кодМеропр, Дата, Длительность), ДанныеДавления(кодПерс, давление). Найти среднее давление для лиц, принимавших лечебное мероприятие с определенным кодом)

4. Отношения ингредиенты(кодИнг, наименование), блюда(кодБлюда, наименование) составБлюд(кодБлюда, кодИнг, колИнгр). Вывести количество блюд использующие своем составе ингредиент заданного наименование ('картошка'). Запрос построить с использованием предиката in.

#### 4. КОНТРОЛЬНЫЕ ВОПРОСЫ.

1. Назовите типы соединения таблиц, какие ключевые слова используются для обозначения связи.

2. Какие разновидности внешних соединений используются?

3. Какое соединение реализует ключевые слова CROSS JOIN. Какой теоретико-множественной операции соответствует данное соединение.

4. Охарактеризуйте чем отличаются левое внешнее, правое внешнее соединение и полное?

5. Зачем используется предикат ON, что он задаёт?

## 5. ЛИТЕРАТУРА

1. Соединения (SQL Server) <https://learn.microsoft.com/ru-ru/sql/relational-databases/performance/joins?view=sql-server-ver16>
2. Join (SQL) [https://ru.wikipedia.org/wiki/Join\\_\(SQL\)](https://ru.wikipedia.org/wiki/Join_(SQL))

## ЛАБОРАТОРНАЯ РАБОТА 5. ИСПОЛЬЗОВАНИЕ ОКОННЫХ ФУНКЦИЙ

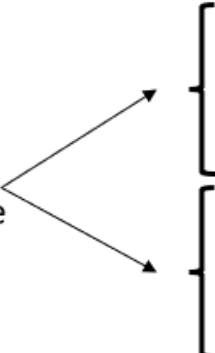
### 1. ЦЕЛЬ РАБОТЫ

Получить практический навык использования оконных функций.

### 2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ.

Оконная функция производит вычисление над заданным набором строк, объединенных каким-то одним значением атрибута (столбца) таблицы. Такой набор называется окном или партицией.

Партиции  
оконной функции  
(в данном примере  
по полю **Имя**)



Имя	Предмет	Оценка
Петя	матем	3
Петя	рус	4
Петя	физ	5
Петя	история	4
Маша	матем	4
Маша	рус	3
Маша	физ	5
Маша	история	3

Рисунок 5.1. Выделение партиций


Партиции формируются аналогично группировкам (group by), но при использовании GROUP BY с агрегирующими функциями результат сокращает количество строк в запросе, строки одной группировки сливаются в одну.

Например, при выполнении команды выборки с группировкой

```
select Имя, avg(оценка) as СрОценка from студОценки group  
Имя
```

Произойдёт слияние строк попадающих в одну группу с одинаковым значением столбца Имя.

Имя	Предмет	Оценка
Петя	матем	3
Петя	рус	4
Петя	физ	5
Петя	история	4
Маша	матем	4
Маша	рус	3
Маша	физ	5
Маша	история	3



Имя	Средняя оценка
Петя	4
Маша	3,75

Рисунок 5.2. Слияние строк при использовании группировки с агрегатными функциями


При использовании оконных функций количество строк в запросе не уменьшается по сравнению с исходной таблицей.

Например:

`select Имя, Предмет, Оценка, avg(Оценка) over (partition by Имя) as [Средняя оценка] from СтудОценки`

Ключевое слово `over` задаёт использование оконной функции

Имя	Предмет	Оценка
Петя	матем	3
Петя	рус	4
Петя	физ	5
Петя	история	4
Маша	матем	4
Маша	рус	3
Маша	физ	5
Маша	история	3



Имя	Предмет	Оценка	Средняя оценка
Петя	матем	3	4
Петя	рус	4	4
Петя	физ	5	4
Петя	история	4	4
Маша	матем	4	3,75
Маша	рус	3	3,75
Маша	физ	5	3,75
Маша	история	3	3,75

Рисунок 5.3. Использование оконной функции

### Порядок расчета оконных функций в SQL запросе

Пусть будет представлен в общем виде запрос с использованием оконных функций (**ОконныеФункции**)

*Select списокСтолбцов, **ОконныеФункции** FROM табл/view  
WHERE услФильтр GROUP BY столбцыГруппир HAVING усл-  
НаГруппы ORDER BY стлбцСорт*

- Сначала выполняется подготовка данных. То есть производится выборка таблиц, их объединения и возможные подзапросы под командой FROM.

- Далее выполняются условия фильтрации WHERE, группировки GROUP BY и возможная фильтрация с HAVING
- Только потом применяется команда выборки столбцов SELECT и расчет оконных функций под выборкой.
- После этого идет условие сортировки ORDER BY, где тоже можно указать столбец расчета оконной функции для сортировки.

Важно уточнить, что партии или окна оконных функций создаются после разделения таблицы на группы с помощью команды GROUP BY, если эта команда используется в запросе.

### Синтаксис оконных функций

Общий формат оконной функции включает следующие элементы

ИмяФункции([АтрФунк]) [FILTER(WHERE услФильтра)]  
OVER (PARTITION BY(АтрФрмОкна) [ORDER BY(АтрСотр)] [услФормФрейма] )

Где,

ИмяФункции([АтрФунк]) – имя оконной функции

FILTER(WHERE условиеФильтра) – указание на использования фильтра данных

OVER(..) ключевое слово, определяющее создаваемое окно данных.

ORDER BY(АтрСотр)] – задание сортировки данных окна по атрибуту АтрСотр

PARTITION BY(АтрФрмОкна) - задание параметров разбиения окна по атрибуту АтрФрмОкна

Оконные функции можно прописывать как под командой SELECT, так и в отдельном ключевом слове WINDOW, где окну дается алиас (псевдоним), к которому можно обращаться в SELECT выборке. Использование псевдонима окна позволяет сокращать запись, в том случае, если требуется использовать несколько функций к одному окну.

дубли определения окна

<pre>select name, subject, grade, row_number() over (partition by name order by grade desc), rank() over (partition by name order by grade desc), dense_rank() over (partition by name order by grade desc) from student_grades;</pre>	=	<pre>select name, subject, grade, row_number() over name_grade, rank() over name_grade, dense_rank() over name_grade from student_grades window name_grade as (partition by name order by grade desc);</pre>
--	---	--

Рисунок 5.4. Использование псевдонима окна при вызове

нескольких оконных функций к одному окну.

Ключ ORDER BY(АтрСотр)] задаёт, как отмечалось ранее, сортировку данных при вычислении оконной функции. При использовании агрегирующей функций значение функции будет вычисляться для всех предшествующих значений в полученной упорядоченной последовательности относительно текущей строки.

Например, конструкция приведённая на примере (Листинг 5.1) указывает на то что будет вычисляться сумма оценок для строк с одинаковыми значениями столбца Имя, причём сумма будет вычисляться для всех предшествующих строк при их упорядочении по оценке, начиная от текущей строки. То есть в виде накопленной суммы.

#### Листинг 5.1 Пример оконной функции

```
select Имя, Предмет, Оценка, sum(Оценка) over (partition by  
Имя order by Оценка desc) as [Средняя оценка] from СтудОценки
```

Результат будет выглядеть следующий образом

Результаты		Сообщения		
	Имя	Предмет	Оценка	Средняя оценка
1	Маша	физика	5	5
2	Маша	математика	4	9
3	Маша	русский	3	15
4	Маша	история	3	15
5	Петя	физика	5	5
6	Петя	история	4	13

Рисунок 5.5. Применение в оконной функции накопления результата по заданному параметру.

Существующие оконные функции можно разделять на 3 класса:

- Агрегирующие (Aggregate)
- Ранжирующие (Ranking)
- Функции смещения (Value)

**Агрегирующие оконные функции** подразумевают применение агрегирующих функций - SUM, AVG, COUNT, MIN, MAX (Листинг 5.2)



### Листинг 5.2 Агрегирующая оконная функция

```
select name, subject, grade,
       sum(grade) over (partition by name) as sum_grade,
       avg(grade) over (partition by name) as avg_grade,
       count(grade) over (partition by name) as count_grade,
       min(grade) over (partition by name) as min_grade,
       max(grade) over (partition by name) as max_grade
from student_grades;
```

abc name	abc subject	123 grade	123 sum_grade	123 avg_grade	123 count_grade	123 min_grade	123 max_grade
Маша	история	3	15	3,75	4	3	5
Маша	математика	4	15	3,75	4	3	5
Маша	русский	3	15	3,75	4	3	5
Маша	физика	5	15	3,75	4	3	5
Петя	математика	3	16	4	4	3	5
Петя	русский	4	16	4	4	3	5
Петя	физика	5	16	4	4	3	5
Петя	история	4	16	4	4	3	5

Рисунок 5.6. Использование агрегирующих функций

### Ранжирующие оконные функции.

В ранжирующих функциях под ключевым словом OVER обязательным идет указание условия ORDER BY, по которому будет происходить сортировка ранжирования.

**ROW\_NUMBER()** - функция вычисляет последовательность ранг (порядковый номер) строк внутри партии, НЕЗАВИСИМО от того, есть ли в строках повторяющиеся значения или нет.

**RANK()** - функция вычисляет ранг каждой строки внутри партии. Если есть повторяющиеся значения, функция возвращает одинаковый ранг для таких строчек, пропуская при этом следующий числовой ранг.

**DENSE\_RANK()** - то же самое что и RANK, только в случае одинаковых значений DENSE\_RANK не пропускает следующий числовой ранг, а идет последовательно.

### Листинг 5.3 Ранжирующая оконная функция.

```
select name, subject, grade,
       row_number() over (partition by name order by grade desc),
       rank() over (partition by name order by grade desc),
```

*dense\_rank() over (partition by name order by grade desc)*  
*from student\_grades;*

name	subject	grade	row_number	rank	dense_rank
Маша	физика	5	1	1	1
Маша	математика	4	2	2	2
Маша	история	3	3	3	3
Маша	русский	3	4	3	3
Петя	физика	5	1	1	1
Петя	русский	4	2	2	2
Петя	история	4	3	2	2
Петя	математика	3	4	4	3

↑  
Пропуск значения «3»

↑  
Без пропуска значения

Рисунок 5.7. Использование ранжирующих функций

Для SQL пустые NULL значения будут определяться одинаковым рангом

### Функции смещения.

Это функции, которые позволяют перемещаясь по выделенной партии таблицы обращаться к предыдущему значению строки или крайним значениям строк в партии.

*LAG()* - функция, возвращающая предыдущее значение столбца по порядку сортировки.

*LEAD()* - функция, возвращающая следующее значение столбца по порядку сортировки.

На простом примере видно, как можно в одной строке получить текущую оценку, предыдущую и следующую оценки Пети в четвертях.

name	quartal	subject	grade
Петя	1 четверть	физика	4
Петя	2 четверть	физика	3
Петя	3 четверть	физика	4
Петя	4 четверть	физика	5

Рисунок 5.8. Таблица для применения функций смещения

Ниже приведён запрос с использованием вывода предыдущей оценки и следующей оценки в

```
select name, quartal, subject, grade,
lag(grade) over (order by quartal) as previous_grade,
lead(grade) over (order by quartal) as next_grade
from grades_quartal;
```

ABC name	ABC quartal	ABC subject	123 grade	123 previous_grade	123 next_grade
Петя	1 четверть	физика	4	[NULL]	3
Петя	2 четверть	физика	3	4	4
Петя	3 четверть	физика	4	3	5
Петя	4 четверть	физика	5	4	[NULL]

Рисунок 5.9. Пример приведения функций смещения.

FIRST\_VALUE()/LAST\_VALUE() - функции возвращающие первое или последнее значение столбца в указанной партии. В качестве аргумента указывает столбец, значение которого нужно вернуть. В оконной функции под словом OVER обязательное указание ORDER BY условия.

### 3. ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ

Скорректировать данные для одной из своих таблиц так, чтобы в них была возможность группировки и применения оконных функций.

Сделать запросы с использованием агрегирующих функций, ранжирующих и функций смещения.

### 4. ЛИТЕРАТУРА

1. Оконные функции SQL простым языком с примерами - <https://habr.com/ru/articles/664000/>

2. Оконные функции - <https://thisisdata.ru/blog/uchimsya-primenyat-okonnyye-funktsii/>

3. Справочник по Transact-SQL (ядро СУБД) <https://learn.microsoft.com/ru-ru/SQL/t-sql/language-reference?view=azure-sqldw-latest>

4. Типы данных (Transact-SQL) <https://learn.microsoft.com/ru-ru/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15>

5. Типы данных и функции даты и времени (Transact-SQL)

<https://learn.microsoft.com/ru-ru/sql/t-sql/functions/date-and-time-data-types-and-functions-transact-sql?view=sql-server-ver16>

## **ЛАБОРАТОРНАЯ РАБОТА 6. НОРМАЛИЗАЦИЯ РЕЛЯЦИОННЫХ ОТНОШЕНИЙ**

### **1. ЦЕЛЬ РАБОТЫ**

Получение практических навыков построения нормализованных отношений

Задачи работы:

- Изучение 3 уровней нормализации и принципов приведения к ним отношений
- Практическое применение изученных принципов нормализации для построения нормализованных отношений согласно индивидуальному заданию.

### **2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ**

Для удобства модификации отношений, избегания избыточности информации, аномалий модификации, исходные отношения обычно приводят к особому виду, называемому нормализованной формой.

Существует несколько уровней (форм) нормализации, на практике обычно достаточно бывает первых трех форм 1NF, 2NF, 3NF.

#### **Первая нормальная форма (1 NF)**

Первая нормальная форма (1NF) - это обычное реляционное отношение. Требования к нему это свойства рел отношения., любое отношение автоматически уже находится в 1NF. Свойства 1NF – это свойства обычных реляционных отношений:

- в отношении нет одинаковых кортежей;
- кортежи не упорядочены;
- атрибуты не упорядочены и различаются по наименованию;
- все значения атрибутов атомарные.

На первом шаге разработки модели БД, то есть в ходе логического моделирования, обычно предполагается хранить данные в одном отношении. В том виде, в котором пользователю требуется получать основные данные.

Пусть требуется хранить и сопровождать информацию о студентах, оценках студентов по учебным предметам, учебных предметах, преподавателях.

Например, данные о студентах, учебных предметах, оценках студентов по предметам, преподавателях.

Таблица 6.1. Пример талицы 1НФ

НомЗачет-ки	ФамСтуд	Пред-мет	Пре-под	УчСте-пень	Оценка
1111	Сидоров	Химия	Фро-лов	ктн	4.5
1111	Сидоров	Физика	Хло-пов	дтн	2
2222	Петров	Иняз	Кторов	безСтеп	5
2222	Петров	Физика	Хло-пов	дтн	5
Null	Null	Пение	Null	Null	Null

Данный пример приведен в ненормализованном виде, в нем очевидна избыточность информации, очевидны и возможные аномалии удаления, обновления, вставки.

Избыточность информации заключается уже в том, что описание ученой степени преподавателя будет повторяться столько же раз, сколько преподаватель будет встречаться в перечне, аналогично с данными о преподавателе.

Аномалия вставки заключается в необходимости вводить пустые значения(NULL) для всех атрибутов, когда появляется студент, не сдававший ещё экзамены.

Аномалия удаления заключается в опасности удалить данные о преподавателе, при удалении данных о студенте.

Аномалия обновления заключается в необходимости корректировки всех записей относящихся к студенту, если он изменит фамилию.

Для исключения этих недостатков исходное отношение подвергается нормализации по второй и третьей нормальной форме.

## **2НФ - Вторая Нормальная Форма.**

Отношение находится во второй нормальной форме (2НФ)

тогда и только тогда, когда отношение находится в 1НФ и нет *неключевых* атрибутов, зависящих от *части сложного ключа*. То есть нет частичных зависимостей.

Частичная зависимость это зависимость не ключевого атрибута от части ключа.

Под зависимостью подразумевается возможность определить значение некоторого атрибута (зависимого) по значению другого атрибута (атрибута детерминанта).

Не ключевой атрибут – это атрибут, не входящий в состав никакого потенциального ключа.

Для рассмотрения данного материала требуется вернуться к понятию первичного ключа и вообще ключа отношения.

При рассмотрении реляционной модели данных обычно выделяется понятие *потенциального ключа*.

Потенциальный ключ это подмножество атрибутов отношения, обладающее свойствами *уникальности* и *неизбыточности*.

Согласно свойству *уникальности* не может быть двух различных кортежей, с одинаковым значением атрибутов, образующих потенциальный ключ.

Согласно свойству *неизбыточности* в потенциальном ключе нельзя выделить подмножество атрибутов, обладающее также свойством уникальности.

Любое отношение имеет, по крайней мере, один потенциальный ключ, так как, согласно основному свойству отношений, в отношении нет одинаковых кортежей, то есть любые кортежи различаются, по крайней мере, полным набором атрибутов. Следовательно, полный набор атрибутов отношения образует ключ.

В большинстве случаев потенциальный ключ включает только часть атрибутов отношения. Возможен случай, когда потенциальный ключ образуется одним атрибутом.

Потенциальный ключ, состоящий из одного атрибута, называется *простым*.

Потенциальный ключ, состоящий из нескольких атрибутов, называется *составным*.

Отношение может иметь несколько потенциальных ключей. Традиционно, один из потенциальных ключей объявляется первичным, то есть используемым при реализации отношения в базе

данных для идентификации кортежей, а остальные потенциальные ключи - альтернативными.

Понятие потенциального ключа отражает некоторый смысл (трактовку) понятий из конкретной предметной области, то есть это некоторые характеристики объектов, по которым их обычно различают.

Для приведения отношения к 2НФ необходимо выявить ключ отношения, то есть разделить атрибуты отношения на ключевые и не ключевые и установить зависимости неключевых атрибутов от атрибутов, входящих в потенциальный ключ.

!!Если потенциальный ключ простой, то отношение автоматически находится в 2НФ.(зависимостей от *части* ключа не будет, так как ключ не имеет частей)

Для выполнения нормализации обычно используется *диаграмма зависимостей*.

Диаграмма зависимостей отображает в виде последовательных блоков атрибуты отношений. При этом ключевые атрибуты на ней отображаются выделенным фоном или шрифтом. Так же на ней в виде линий указываются зависимости атрибутов ().

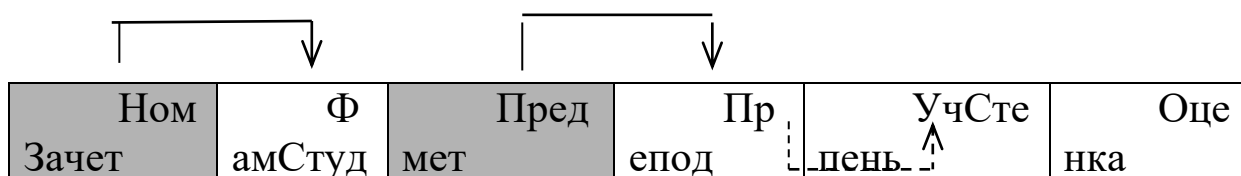


Рисунок 6.1. Диаграмма зависимостей.

В рассматриваемом примере в качестве потенциального ключа приняты атрибуты (НомЗачетки,Предмет). По значению данных атрибутов можно определить любой кортеж (строку, запись).

Можно было выделить и другие варианты потенциальных ключей, например (ФамСтуд, Препод). Однако данный вариант по смысловому содержанию не отражал бы особенности предметной области.

Для выполнения преобразования к 2НФ на диаграмме зависимостей указываются зависимости неключевых атрибутов от атрибутов, входящих в потенциальный ключ.



В общем случае, под зависимостями подразумевается, что по значению некоторых атрибутов можно определить значения других атрибутов. Те атрибуты, которые определяют значения других, называются *детерминантами* в данной зависимости.

Зависимости неключевых атрибутов от атрибутов, входящих в потенциальный ключ, называются частичными (зависимость от части ключа). Частичные зависимости на диаграмме зависимостей изображены в виде сплошных линий.

В рассматриваемом примере выделены две частичные зависимости:

НомерЗачетки  $\rightarrow$  Фамилия,

Предмет  $\rightarrow$  Препод

По смыслу зависимость Предмет  $\rightarrow$  Препод

показывает, что по предмету можно найти преподавателя, который его вёл.

Транзитивной зависимостью называется зависимость неключевых атрибутов от других неключевых. Транзитивной такая зависимость названа, потому что в данном случае зависимый атрибут связывается с некоторым ключевым через промежуточные (транзитивные) неключевые атрибуты. Ключевой атрибут, с которым связана данная транзитивная зависимость в данной работе будет обозначаться как базовый для транзитивной зависимости. Транзитивные зависимости на диаграмме зависимостей обозначены штриховыми линиями.

В рассматриваемом примере выделена одна транзитивная зависимость:

Препод  $\rightarrow$  УченаяСтепень.

Данная зависимость подразумевает то, что ученая степень может быть определена по данным о преподавателе, что соответствует условиям предметной области.

Приведение отношения к 2НФ подразумевает декомпозицию исходного отношения на несколько, соответствующих выявленным частичным зависимостям. Ключами в данных отношениях будут ключевые атрибуты исходного отношения – детерминанты в данной зависимости.



Рисунок 6.2. Декомпозиция отношения по частичным зависимостям

При данной декомпозиции транзитивно зависимые атрибуты выносятся вместе с теми атрибутами от которых они зависят.

В исходном отношении останутся атрибуты первичного ключа и неключевые атрибуты, не связанные частичной и транзитивной зависимостями.

Отношения, полученные в результате декомпозиции из исходного, будут связаны через свои ключи с остатками исходного отношения, внешними ключами в оставшемся отношении для них будут те частичные ключи, на основании которых были построены данные отношения.

### 3НФ – Третья Нормальная форма

Приведение отношения к 3НФ подразумевает, чтобы отношение было нормализовано по 2НФ и в отношении не существовало бы транзитивных зависимостей. Другими словами третья нормальная форма повышает требования второй нормальной формы: оно не ограничивается составными первичными ключами, а требует, чтобы ни один неключевой столбец не зависел от другого неключевого столбца. Любой неключевой атрибут должен зависеть только от первичного ключа.

В полученном отношении (Предмет, Препо, УчСтепень) осталась транзитивная зависимость Препо → УчСтепень, ее наличие скажется в избыточности данных и аномалиях, будет много раз повторяться данные о учёной степени, в том случае, если учёная степень одинакова у различных преподавателей.

Для приведения к 3NF отношение декомпозируется таким образом, чтобы транзитивная зависимость была вынесена в отдельное отношение.

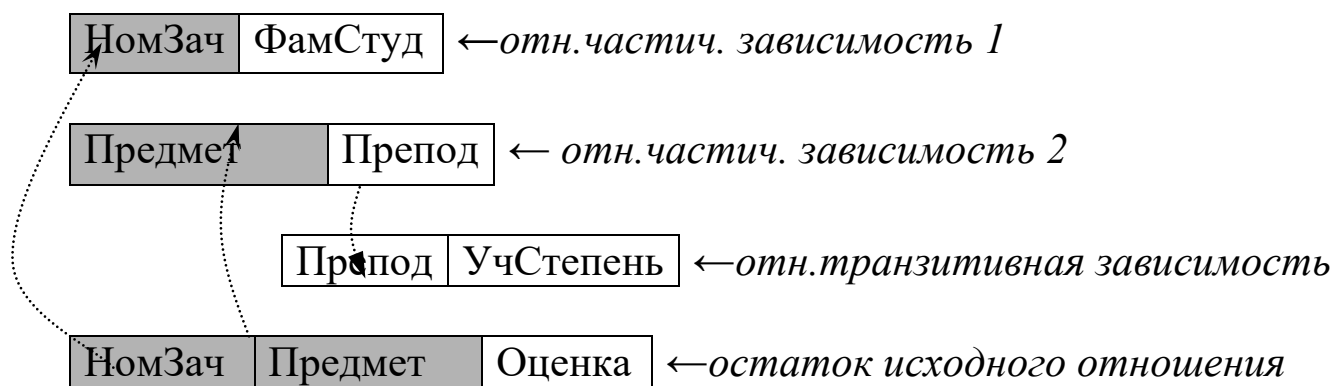


Рис.3. Вынесение транзитивной зависимости в отдельное отношение преобразованная к 3НФ

### 3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Выделить потенциальный ключ.
2. Построить диаграмму зависимостей.
  - а. Выявить частичные зависимости.
  - б. Выявить транзитивные зависимости.
  - с. Отобразить выявленные зависимости на диаграмме зависимостей.
3. Преобразовать в форму 2НФ, избавившись от частичных зависимостей.
4. Преобразовать в форму 3НФ, избавившись от транзитивных зависимостей.
5. Построить ER модель.
6. Построить заданные отношения в среде Microsoft SQL Server.
7. Сформировать запрос, результатом которого является соединение всех отношений, полученных в результате нормализации в один набор данных, соответствующий исходному отношению. Для формирования запроса можно использовать конструктор, используемый при создании представлений (new view) в среде management studio.

### 4. ВАРИАНТЫ ЗАДАНИЙ

В качестве задание выдается отношение в ненормализованном виде. Студент должен выполнить нормализацию, построить

ER диаграмму для сформированных отношений и сформировать запрос к связанным таблицам, позволяющий получить данные в исходном виде.

Заданием для данной лабораторной работы является отношение, сформированное из отношений, полученных в результате выполнения лабораторной работы №1. В данном отношении необходимо соединить все атрибуты отдельных отношений. Полученное общее отношение должно быть утверждено преподавателем.

В некоторых случаях отношение для выполнения лабораторной работы может быть взято из ниже приведённых вариантов.

СотрудникОтделыПроекты(Н\_СОТР, ФАМ, Н\_ОТД, ТЕЛ, Н\_ПРО, ПРОЕКТ, Н\_ЗАДАН)

Н\_СОТР - табельный номер сотрудника

ФАМ - фамилия сотрудника

Н\_ОТД - номер отдела, в котором числится сотрудник

ТЕЛ - телефон сотрудника

Н\_ПРО - номер проекта, над которым работает сотрудник

ПРОЕКТ - наименование проекта, над которым работает сотрудник

Н\_ЗАДАН - номер задания, над которым работает сотрудник

каждый сотрудник в каждом проекте выполняет ровно одно задание,

Товар(кодТовара, наименованиеТовара, КодПроизводителя, НаименованиеПроизводителя, Код Страны производителя, СтранаПроизводителя, ЦенаТовараДанногоПроизвдителя) -

СотрудникДолжность(ТабНомер, Фамилия, КодДолжности, НАименованиеДолжности, ОкладПоДаннойДолжности, ДатаНазначения)

Дела(КодДела, НаименованиеДела, КодГлавногоИсполнителя, ИмяИсполнителя, ТрудоёмкостьДела, КодТребуемогоРесурса, НаименованиеРесурса, ОбъёмРесурса)

СоставИнгр(КодБлюда, НаименованиеБлюда, КодИнгредиента, НаименованиеИнгредиента, КоличествоИнгреВБлюде, КодОперацииНадИнгрдВ\_блюде, НаименованиеОперации, ДлительностьОперации)

РаботаНадПроектомВыполнСотрудн(НомерПроекта, НаимеНоваНиеПроекта, КодПредприятияЗаказчика, ТелДиректораЗаказчика, ФамилияДиректораЗаказчика, табНомерСотрудника, ФамилияСотрудника, НомерОтделаСотрудника, НаименованиеОделаСотрудника, КодВидаРаботКоторВыполнСотрНадПроект, НаименовВидаРаботКоторыеВыполняетСОтрудник, СтоимостьЧасаОпределённогоВидаРабот, отработанныеЧасыОперделённогоВидаРАбот) - сотрудник выполняет определённые виды работ над проектом, выполняемым для определённого предприятия.

УспеваемостьСтудендтов(НЗачКн, Фам, Групп, УчКурс, Часов,Семестр,Оценка)

ДанныеНагрузкеПреподователей(ТабНомПреподователя, ФамПрепод, НаимКаф, НомТелКаф, АудКафедры, КодПредметаКоторыйВедётПреопод, НаименованиеПредметВедомыйПреп, НаименованиСпециальностьДляКоторойВедёт, КодСпециальности, ЧасовНаПредмет, ЛабораторияВКоторой ведётся предмет)

Данные о измерении показателей на контролируемых объектах ЗначПоказНаОъектах(КодОбъекта, НаименованеОъекта, ДатаИзмерения, КодПоказателя, НаименованиеПоказателя, значениеПоказателя, КодЕдиницыИзмеренияПокзателя, ЕдиницаИзмеренияПоказателя, значениеПоказателя) - для каждого показателя своя одна единица измерения. Она опеределяется показателем (Температура -градусы цельсия, давление – паскалях..)

## 5. КОНТРОЛЬНЫЕ ВОПРОСЫ.

1. Что такое первичный ключ отношения, потенциальный ключ?
2. Что из себя представляет отношение нормализованное по 1NF?
3. Что такое частичная зависимость?
4. Что из себя представляет отношение, нормализованное по 2NF? 3NF
5. Что такое транзитивная зависимость?
6. Что из себя представляет отношение, нормализованное по 3NF?

## 5. ЛИТЕРАТУРА

1. Роб П., Коронел К. Системы баз данных: проектирование, реализация и управление. – 5-е изд., перераб. и доп. Пер. с англ. – Спб.: БХВ- Петербург, 2004.-1040 с.: ил.

2. Нормальная форма  
[https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F\\_%D1%84%D0%BE%D1%80%D0%BC%D0%B0](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0)

3. Нормализация <https://metanit.com/sql/tutorial/2.1.php>

## **ЛАБОРАТОРНАЯ РАБОТА 7. МЕХАНИЗМЫ РАБОТЫ С БД. ОГРАНИЧЕНИЯ**

### **1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ**

Целью работы является получение практических навыков создания различного типа ограничений, используемых в база данных.

В связи с этим задачами работы являются

- Изучение назначения ограничений. Их типов и видов задания.
- Изучение основных операторы языка SQL создания ограничений.
- Создание ограничений и анализ их работы.

### **2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Под механизмами подразумеваются действия над данными, выполняемые автоматически, обеспечивающие сохранение целостности данных и реализующие логику работы предметной области.

Среди механизмов можно выделить отдельную группу - механизмы обеспечения целостности. Они реализуются в виде ограничений и более сложных элементов – триггеров (об этом позже)

#### **Ограничения**

Ограничения позволяют обеспечить целостность БД с точки зрения соответствия их основным свойствам реляционных отношений

- уникальность первичных ключей,
- сохранение связей по внешним ключам.

Кроме того ограничения позволяют реализовать обеспечение целостности с точки зрения соответствия требованиям предметной области (запись в качестве значения атрибутов только тех данных, которые отвечают предъявленным к ним ограничениям)

Ограничения можно разделить на следующие группы по виду накладываемого ограничения:

- ограничения первичного ключа;
- уникальность значения атрибута;
- ограничение внешнего ключа;
- ограничение типа check;
- not NULL;
- ограничения общего вида.

#### ***Задание ограничения может выполняться***

- на уровне атрибута, то есть при описании атрибута в операторах создания или модификации отношения
- на уровне кортежа.

На уровне атрибута могут задаваться ограничения первичного ключа, уникальности, типа check, not NULL.

Пример задания на уровне атрибута.

Create table myTable(at1 int primary key, ....)

На уровне кортежа ограничения задаются в командах создания нового отношения или модификации существующего. При задании на уровне кортежа ограничению может быть присвоено имя.

Задание ограничений на уровне кортежа.

Ограничения первичного ключа

constraint ИмяОгр primary Key [(Атр1[Атр2])]

ИмяОгр, имя, присваиваемое ограничению, его можно использовать при модификации отношений для удаления ограничения.

Удаление ограничения выполняется при модификации таблицы.

ALTER TABLE nameTab drop CONSTRAINT nameConstraint

Ограничения уникальности задаются аналогично, ключевое слово UNIQUE – ограничение внешнего ключа.

Ограничение внешнего ключа задает целостность отношений по ссылкам, то есть, чтобы на каждый кортеж, в котором задан внешний ключ, существовал кортеж связанном с ним отношении.

Задание ограничений внешнего ключа.

constraint ИмяОгранич Foreign Key [(Атр1[Атр2]...)] references БазОтн (АтрБ1[АтрБ2]...) [on delete {cascade/ no action }]



[on update {cascade/no action}]

АтрБ1[АтрБ2]...– атрибуты базового отношения, на которые ссылается внешний ключ.

Ограничение внешнего ключа обеспечивает целостность по ссылкам следующими способами:

1. Отмена выполняемых действий - no action
2. Каскадная модификация - cascade, то есть, если модифицируется атрибут внешнего ключа, то производится модификация соответствующего ключа атрибута базового отношения.

Ограничение check.

Данное ограничение позволяет задавать произвольные условия, контролируемые значение вводимого атрибута. В ограничении может быть несколько условий, соединённых логическими операциями.

[constrant ИмяОгр] check (условие)

Условие в одном ограничении может быть задано только на один атрибут. Если нужно на два атрибута задаём два ограничения.

Например:

```
ALTER TABLE [dbo].[Client] ADD CONSTRAINT  
[CK_Client_1] CHECK ((([phone]>=(1000))))  
GO
```

Ограничение NOT NULL позволяет исключить возможность ввода значения NULL в атрибут отношения.

### 3. ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ

1. Задать ограничение уникальности для одной из таблицы.
2. Задать ограничения внешнего ключа одной из таблиц своих таблиц (при необходимости сначала удалите его из диаграммы, сохраните диаграмму).
3. Убедиться на примере, что ограничение работает.
4. Задать ограничения типа check согласно логике предметной области. Сделать проверку работы данных ограничения.
5. Удалить созданные ограничения показать на примерах что ограничения не действуют.

#### 4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего используются ограничения.
2. Какие виды ограничений используются в СУБД SQL SERVER?
3. Каким образом могут быть созданы ограничения?
4. На каком уровне могут быть созданы ограничения?
5. Как задаётся ограничение внешнего ключа?
6. Какие дополнительные параметры задаются при задании ограничения внешнего ключа?

#### 5. ТРЕБОВАНИЯ К ОТЧЁТУ

В отчёте должно быть отображены тексты запросов для создания ограничений и приведены копии экрана, подтверждающие работу ограничения.

#### 6. ЛИТЕРАТУРА

1. Атрибуты и ограничения столбцов и таблиц  
<https://metanit.com/sql/sqlserver/3.4.php>
2. Ограничения уникальности и проверочные ограничения  
<https://learn.microsoft.com/ru-ru/sql/relational-databases/tables/unique-constraints-and-check-constraints?view=sql-server-ver16>

## **ЛАБОРАТОРНАЯ РАБОТА 8. МЕХАНИЗМЫ РАБОТЫ С БАЗОЙ ДАННЫХ. ИСПОЛЬЗОВАНИЕ ХРАНИМЫХ ПРОЦЕДУР И ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИЙ.**

### **1. ЦЕЛЬ РАБОТЫ**

Получить практических навык организации бизнес-логики на стороне сервера, на основе использования хранимых процедур и пользовательских функций

В связи с этим задачами работы является:

1. Изучение принципов работы и создание хранимых процедур и пользовательских функций в среде MS SQL Server;
2. Создание хранимых процедур и пользовательских функций в соответствии с заданием к лабораторной работе.

### **2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ**

Для реализации бизнес логики на стороне сервера используются, так называемые «механизмы сервера», включающие хранимые процедуры, пользовательские функции, триггеры, и другие объекты их состав может быть различен для СУБД различных производителей.

Наиболее распространенное средство реализации бизнес-логики – хранимые процедуры сервера.

Хранимые процедуры – это модули, хранящиеся непосредственно в базе данных в откомпилированном виде и которые могут запускаться пользователями или приложениями, работающими с базой данных. Хранимые процедуры обычно пишутся либо на специальном процедурном расширении языка SQL (например, PL/SQL для ORACLE или Transact-SQL для MS SQL Server), или на некотором универсальном языке программирования, например, C++, с включением в код операторов SQL в соответствии со специальными правилами такого включения. Основное назначение хранимых процедур - реализация бизнес-процессов предметной области на стороне сервера.

Процедура подразумевает выполнение не только обычных

запросов SQL, но и более сложную обработку, связанную с вычислениями, переходами, сравнениями ветвлениями. Для этого существует расширение SQL включающее описание переменных, операторы ветвления, циклы и пр.

### **Создание хранимой процедуры**

Создание хранимой процедуры, как и других объектов данных выполняется командой CREATE.

```
CREATE PROCedure [владелец.]имя_процедуры  
    списокПараметров  
AS  
Операторы_SQL
```

Где, списокПараметров – параметры функции разделённые знаком ‘,’.

Отдельный параметр - @имяПараматра тип\_данных [=default] [OUTPUT]

[=default] – значение которое, получает параметр, при отсутствии передаваемого значения (по умолчанию).

[OUTPUT] – ключевое слово, указывающее, что при изменении параметра в ходе выполнения процедуры новое значение возвращается в переменную, используемую для вызова этой процедуры.

Длина имени хранимой процедуры вместе не может превышать 20 символов. Одна процедура может вызывать другую процедуру, уровень вложенности не может превышать 16, текущий уровень вложенности можно узнать из глобальной переменной @@NESTLEVEL

Вызов хранимой процедуры осуществляется оператором EXEC

```
EXEC имяПроцедуры фактЗнПараметра
```

Где фактЗнПараметра – фактическое значение параметра.

В качестве фактического значения параметра может выступать как переменная, так и константа

*Пример: использование константы в качестве фактического значения параметра:*

EXEC СводСотрудников 1111, 'Инженер'

В приведённом примере производится вызов хранимой процедуры СводСотрудников. При вызове в качестве фактических параметров передаётся целое число 1111 и строка 'Инженер'

*Использование переменной в качестве параметра:*

DECLARE @Должность varchar(10), @TabNamb int

SET @Должность = 'Инженер'

SET @TabNamb = 1111

EXEC СводСотрудников @TabNamb, @Должность

Изменение хранимой процедуры производится, как и других объектов базы данных, командой ALTER. Формат команды совпадает с форматом команды CREATE

ALTER PROCEDURE [владелец.]имя\_процедуры

    списокПараметров

AS

Операторы\_SQL

### **Удаление хранимой процедуры**

DROP PROCEDURE имя\_процедуры

### **Особенности процедурного SQL**

Для создания тела процедуры используется расширенная версия SQL процедурный SQL (PSM SQL). Данное расширение SQL позволяет использовать переменные, условные операторы, циклы и другие конструкции. Реализация PSM SQL в каждом диалекте СУБД может различаться.

Для использования переменных их необходимо объявить с помощью ключевого слова DECLARE

DECLARE имяПерем типДанных

Где имяПерем - идентификатор переменной. Он должен быть без пробелов. В качестве первого символа должен использоваться символ «@». В среде SQL server могут быть созданы глобальные переменные, видимые на уровне сервера, данные переменные начинаются с двух символов «@@». Локальные пере-

менные, начинающиеся с одного символа «@» являются локальные и видимы только внутри одной транзакции.

Для присвоения значений переменным используется оператор Set или Select.

Формат операторов SET и Select при присвоении значений.

SET имяПерем = значение

или

SELECT имяПерем значение

Для обращения к значению переменной в запросе используется та же конструкция SELECT

SELECT имяПерем as синоним.

### **Основные алгоритмические конструкции процедурного расширения языка SQL**

*Блок операторов* – набор команд воспринимаемых как одно целое

Begin

....

end

*Конструкция ветвления в SQL*

IF логическоеВыражение

{ операторSQL | блокОпреаторовSQL }

[ ELSE

{ операторSQL | блокОпреаторовSQL } ]

*Конструкция цикла*

WHILE <логическое\_выражение>

{ операторSQL | блокОпреаторовSQL }

[ BREAK ]

{ операторSQL | блокОпреаторовSQL }

[ CONTINUE ]

Для вывода результатов вычислений или значений переменных можно использовать оператор print

Print выражениеТипаСтрока

T-SQL имеет набор функций, позволяющих выполнять набор математических, строковых, системных операций и операций преобразования (См. приложение).

### **Преимущества использования процедур**

Использовать хранимые процедуры целесообразнее, чем отдельные операторы SQL по следующим причинам:

- Операторы хранимой процедуры всегда находятся в базе данных.
- Операторы хранимой процедуры уже проверены и находятся в готовом для использования виде
- Возможность использования процедур позволяет использовать модульное программирование
- Хранимые процедуры могут вызывать другие процедуры и функции.
- Сохраненные процедуры могут вызываться любыми приложениями.
- При использовании сохраненных процедур результат от ответа от базы данных как правило получается быстрее.
- Использование процедур принципиально упрощено в СУБД.

### **Пользовательские функции UDF (user-defined function)**

Основное отличие UDF от хранимых процедур заключается в том, что функция обязательно должна вернуть хотя бы какое-то значение. UDF имеют некоторые ограничения. Нельзя, например, изменять данные в таблицах БД, выводить данные с помощью команд print и select. Внутри UDF нельзя использовать недетерминированные функции типа GETDATE(). В SQL Server имеется 3 типа пользовательских функций: Scalar, Multi-statement Table, InLine.

Scalar – скалярная функция может возвращать значения любого скалярного типа данных, кроме данных типа text, image, table. Такая функция может объединять несколько команд языка T-SQL, находящихся в блоке BEGIN...END.

При задании заголовка функции инструкцией Returns указывается тип параметра, который возвращает функция. Само возвращение данных из функции выполняется командой Return Воз-

возвращаемоеВыражение. Тип возвращаемого выражения должен соответствовать типу продекларированному в заголовке.

Пример скалярной пользовательской функции приведён в листинге (Листинг 8.1)

*Листинг 8.1 Скалярная пользовательская функция.*

```
CREATE FUNCTION FunState
(@a varchar(20))
RETURNS varchar(20)
as
BEGIN
IF @a IS NULL
SET @a = "Not Applicable"
RETURN @a
END
```

Обратиться к такой функции можно с помощью конструкции select, указав вместо поля таблицы значение функции с параметром:

```
SELECT pub_name, City, dbo.FunState(state) AS State FROM
dbo.publishers
```

***Multi-Statement Table*** – ***много командная табличная*** функция. Функция возвращает тип данных table, то есть таблицу. Тело такой функции может быть достаточно сложным и включать несколько (множество) операторов, находящихся между заключённых между ключевыми словами BEGIN...END.

Тип временной таблицы, которую возвращает функция, задаётся в заголовке инструкцией Returns. Имя временной таблицы должно начинаться с символа «@». Формат описания возвращаемой таблицы такой же, как и команде создания таблицы. В ниже приведённом примере создается функция, которая должна возвращать таблицу, состоящую из двух столбцов FirstName nvarchar(20), LastName nvarchar(20). В качестве данных этих столбцов заносятся атрибуты au\_fname и au\_lname из таблицы authors, если в качестве входного параметра @b указано слово 'author'. Если в качестве входного параметра указано слово



'employee', то заносятся атрибуты fname, lname из таблицы employee.

*Листинг 8.2 Пример табличной много командной функции.*

```
CREATE FUNCTION FunMult (@b nvarchar(8)) RETURNS
@Fun_Auth table (FirstName nvarchar(20) not null, LastName nvar-
char(20) not null)
AS
BEGIN
IF @b = 'author'
INSERT @Fun_Auth SELECT au_fname, au_lname
FROM authors
ELSE IF @b = 'employee'
INSERT @Fun_Auth SELECT fname, lname
FROM employee
RETURN
END
```

InLine функция. Функции данного типа также возвращают значение типа table, но отличается от Multi-Statement Table-valued тем, что может состоять только из одной команды select. В функциях данного типа не требуется описания возвращаемой таблицы.

*Листинг 8.3 Пример табличной inline функции.*

```
CREATE FUNCTION FunInLine
(@State nvarchar(30))
RETURNS table
AS
RETURN (      SELECT pub_name, city
FROM Pubs.dbo.publishers
WHERE state = @State
)
```

Обращение к табличным функциям (много командным и in-line-функциям) происходит командой select. Они указываются в качестве таблиц.

SELECT \* FROM FunInLine('Texas')

### **Удаление функции**

DROP FUNCTION Имя функции

Особенностью функции InLine является то, что код функции при выполнении программы вставляется непосредственно в исполняемый набор команд. Другими словами, происходит не вызов функции, а встраивание в текущую команду.

В среде SQL server хранимую процедуру или функцию можно создать в приложении Management Studio в разделе хранимых процедур с помощью контекстного меню или с помощью отдельного запроса.

## **3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ**

1. Уточнить задание применительно к сформированным отношениям.

2. Схематично описать необходимые запросы по заданию для своей БД.

3. Утвердить у преподавателя эскиз лабораторной работы.

4. Создать хранимые процедуры и функции с использованием описанных операторов в методических указаниях (также использовать по одной из стандартных ф. по выбору преподавателя).

5. Подготовить и предоставить контрольные примеры с использованием хранимых процедур

## **5. ЗАДАНИЕ.**

1. Оформить запросы, выполненные в ЛР № 3 и ЛР №4 в виде хранимых процедур с использованием входных параметров в условиях выбора WHERE и HAVING.

2. Создать хранимую процедуру, в которой будет выполняться гибкий запрос на группировку по различным полям в зависимости от входного параметра

3. Создать пользовательскую функцию, которая возвращает

значение, вычисленное, на основе одного или нескольких атрибутов отношения по заданным критериям поиска.

4. Создать хранимую процедуру, использующую созданную пользовательскую функцию.

5. Создать хранимые процедуры для вставки, изменения и удаления данных, переданных через входные параметры.

## 6. КОНТРОЛЬНЫЕ ВОПРОСЫ.

1. Что такое хранимая процедура, общий формат описания и вызов?

2. Каков формат описания переменных в хранимых процедурах?

3. Тип параметров функций и процедур?

4. Описать типы пользовательских функций?

5. Формат описания конструкций цикла и ветвления?

6. Чем отличается хранимая процедура от пользовательской функции?

7. Что дает использование хранимых процедур при разработке БД?

## 4. ЛИТЕРАТУРА

1. SQL/PSM <https://ru.wikipedia.org/wiki/SQL/PSM>

2. Хранимые процедуры (ядро СУБД).  
<https://learn.microsoft.com/ru-ru/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver16>

3. Хранимые процедуры  
<https://metanit.com/sql/sqlserver/11.1.php>

## **ЛАБОРАТОРНАЯ РАБОТА 9. ОРГАНИЗАЦИЯ БИЗНЕС-ЛОГИКИ ИС НА СТОРОНЕ СЕРВЕРА. ИСПОЛЬЗОВАНИЕ ТРИГГЕРОВ.**

### **1. ЦЕЛЬ РАБОТЫ**

Получить практический навык использования триггеров для организации бизнес-логики на стороне сервера.

В связи с этим задачами работы являются.

- Изучить принципы работы и особенности построения триггеров в среде MS SQL Server.
- Создать триггеры, выполняющие действия, согласно индивидуальному заданию.

### **2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ.**

Триггер – это откомпилированная процедура, используемая для выполнения действий, инициируемых происходящими в базе данных событиями. Такими событиями являются запросы к базе данным, генерируемые операторами языка манипуляции данными — DML (INSERT, DELETE или UPDATE). С помощью триггеров можно отменять транзакции, а также модифицировать данные одних таблиц и читать данные других даже из других баз данных. В общем случае, результатом работы триггеров является обеспечение целостности базы данных.

Чаще всего триггеры использовать очень удобно, однако, их использование приводит к значительному увеличению числа операций ввода-вывода. Триггеры не следует использовать тогда, когда ограничение, хранимая процедура или клиентская программа может добиться тех же результатов с меньшими накладными расходами.

В Microsoft SQL Server синтаксис оператора для создания триггера выглядит следующим образом.

```
CREATE TRIGGER имяТриггера  
ON имяТаблицыПредставления  
{FOR|AFTER|INSTEAD OF} {[INSERT], [DELETE], [UP-
```

```
DATE]]  
AS  
операторыSQL  
[IF UPDATE (ПОЛЕ) [{AND|OR} UPDATE (ПОЛЕ)]] и тд...
```

Как видно из шаблона команды, в среде СУБД SQL Server триггер может выполняться вместо или после операторов вызвавших срабатывание триггера.

Изменение триггера производится командой ALTER TRIGGER имяТриггера. Формат данной команды совпадает с форматом команды CREATE.

Удаление триггера выполняется командой DROP TRIGGER имяТриггера.

Триггер может быть создан только в текущей базе данных, но допускается обращение внутри триггера к другим базам данных, в том числе и расположенным на удаленном сервере.

Имя триггера должно быть уникальным в пределах базы данных. Дополнительно можно указать имя владельца.

Работа триггера строится на предварительном контроле вводимых, удаляемых или модифицируемых данных с помощью временных таблиц. Действия, выполняемые с таблицами, предварительно выполняются с временными таблицами. При вставке строк данные предварительно помещаются во временную таблицу inserted, при удалении - удаляемые данные помещаются в таблицу deleted. Данные таблицы имеют такую же структуру, как и таблицы, на которые создаются соответствующие триггеры.

При модификации данных создаются две временных таблицы inserted и deleted. Так как модификация подразумевает удаление старых данных и вставку на их место новых.

Использование триггера для реализации ограничений на значение.

Пусть в базе данных есть две таблицы

Таблица товаров tov(idtov int, nameTov varchar(12), kolTov int) и таблица движения товаров dvTov(idTov int, kolDvTov int, dateDv). Таблица tov задаёт товары присутствующие на складе (код и наименование) и остаток товара (kolTov). Таблица dvTov

задаёт количество поступления или расхода (если знак -) товара и дату движения.

Необходимо чтобы при выполнении движения, то есть при добавлении строки в таблицу dvTov выполнялась проверка на наличие необходимого остатка.

Команда вставки записи в таблицу dvTov может быть, например, такой:

```
INSERT INTO Сделка VALUES (1,-5,'01/01/2024')
```

Создаваемый триггер должен отреагировать на ее выполнение следующим образом: необходимо отменить команду, если в таблице Tov величина остатка товара оказалась меньше продаваемого количества товара с введенным кодом (в примере код товара=2). Во вставляемой записи количество товара указывается со знаком «+», если товар поставляется, и со знаком «-», если он продается. Представленный триггер настроен на обработку только одной добавляемой записи.

*Листинг 9.1 Триггер на ограничение вставки с учётом данных логически связанной таблицы*

```
alter TRIGGER Триггер_ins ON dvTov FOR INSERT AS
IF @@ROWCOUNT=1
BEGIN
declare @kodTov int, @kolTov int, @ostTov int
select @kodTov=idTov, @kolTov=kolDvTov from inserted
set @ostTov=(select kolTov from tov where idtov=@kodTov)
if @ostTov+@kolTov<0
begin
rollback transaction
print 'остаток товара с кодом '+convert(varchar,@kodTov)+'
меньше чем требуется OstTov='+ convert(varchar,@ostTov)
end
end
```

Данный триггер включает действие по выборке данных из временной таблицы inserted с помощью команды select. В данной команде производится присвоение переменной @kodTov атрибу-

та idTov (код товара добавление которого производится). Также переменной @kolTov присваивается значение атрибута kolDvTov (поставляемое или расходуемое количество товара). Затем производится выборка остатка данного товара (товара с кодом @kodTov) и присвоение остатка переменной @ostTov. Затем выполняется проверка на случай не достаточности остатка для выполнения расхода ( $@ostTov + @kolTov < 0$ ). Если суммарная величина остатка и величины движения товара меньше 0, то производится откат транзакции и выводится сообщение пользователю.

Например пусть текущее состояние таблицы tov соответствует данным приведённым на рисунке (Рисунок 9.2)

	idtov	nameTov	kolTov
►	1	Макарон...	5
	2	Пряники	2
	5	Печенье	9
*	NULL	NULL	NULL

Рисунок 9.2. Текущее состояние таблицы Tov

Ниже показано команды по вставке данных и реакция триггера (Рисунок 9.3).

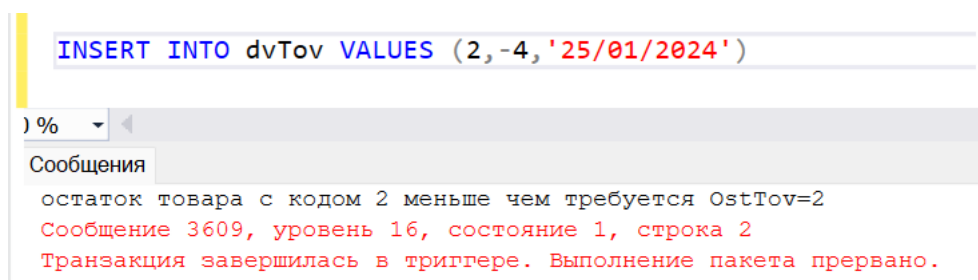


Рисунок 9.3. Реакция триггера на вставку данных выходящих за заданные пределы.

При вставке данных, в которых расход не выходит за пределы остатка. Триггер не отменяет транзакцию.



Рисунок 9.4. Вставка данных, не выходящих за пределы остатка

В приведённом выше примере используется не лучшая реализация триггера с точки зрения производительности. В нём объявляются переменные, и кроме того, выполняется две выборки.

Более производительным будет вариант с использованием соединения таблицы inserted с таблицей Tov с проверкой существования строк в которых остаток выходит за заданные пределы (Листинг 9.2)

*Листинг 9.2 Более рациональный вариант реализации триггера ограничения на основе данных логически связанной таблицы.*

```
alter TRIGGER Триггер_ins ON dvTov FOR INSERT AS
IF @@ROWCOUNT=1
BEGIN
  IF NOT EXISTS(SELECT * FROM inserted WHERE -
inserted.kolDvTov <=ALL (SELECT tov.kolTov FROM inserted join
tov on inserted.idTov= tov.idTov))
  begin
    rollback transaction
    print 'остаток товара с кодом меньше чем требуется OstTov'
  end
end
```

Рассмотренный выше пример предметной области будет иметь смысл, если остаток в таблице товаров соответствует выполненным операциям движения. То есть при добавлении данных в таблицу dvTov необходимо производить пересчёт в таблице Tov остатка соответствующего товара. Данное действие логично выполнять с помощью рассмотренного выше триггера.

Для товара, код которого указан в таблицы добавления за-



писи, необходимо откорректировать остаток на складе. С учётом данного изменения триггер будет выглядеть следующим образом (Листинг 9.3). В данной версии нельзя обойтись без использования переменных и вспомогательных выборов.

*Листинг 9.3 Модификация триггера с добавлением корректировки логически связанной таблицы.*

```
alter TRIGGER Триггер_ins ON dvTov FOR INSERT AS
IF @@ROWCOUNT=1
BEGIN
declare @kodTov int, @kolDvTov int, @ostTov int
IF NOT EXISTS(SELECT * FROM inserted
WHERE      -inserted.kolDvTov<=ALL(SELECT      tov.kolTov
FROM inserted join tov
on inserted.idTov= tov.idTov))
begin
rollback transaction
print 'остаток товара с кодом меньше чем требуется OstTov'
end
else
begin
select @kodTov=idTov, @koldvTov=kolDvTov from inserted
set @ostTov=(select koltov from tov where idtov=@kodTov)
UPDATE tov SET kolTov=KolTov-@kolDvTov WHERE
idTov=@kodTov
end
end
```

Аналогичные действия также необходимо предусмотреть для случая удаления строк из таблицы движения и их модификации.

### 3. ЗАДАНИЕ И ПОРЯДОК ВЫПОЛНЕНИЯ

Задаaniem для работы является создание триггеров на таблицы базы данных по заданным требованиям.

1. Создать триггер, для самой модифицируемой таблицы. Триггер должен вести журнал изменений (журнал - отдельная таблица. `Jornal(TypeWork varchar(12), dann int, dateWork datetime)`) в журнал необходимо занести выполняемое действие (вставка, удаление, изменение), дата выполнения операции.

2. Создать триггеры, препятствующие вставке в отношение определенных кортежей, не отвечающих заданным требованиям, предъявляемым к значению каких либо атрибутов.

3. Создать триггер препятствующий изменению численного атрибута отношения выходящего за определенные установленные рамки.

4. Создать триггер, выполняющий каскадное удаление кортежей с внешними ключей из таблицы, с которыми связаны удаленные кортежи с первичными ключами в исходной таблице, если в ней таких кортежей больше не осталось.

#### 4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое триггер? На какие события создаются триггеры СУБД MS SQL Server?

2. Какие действия может выполнять триггер?

3. Общий формат команды создания триггера?

4. На каком принципе построена логика работы триггеров?

5. Как выявляются данные, вызвавшие срабатывание триггера?

6. Какие временные таблицы создаются при срабатывании триггера на модификацию таблицы?

#### 5. ЛИТЕРАТУРА

1. CREATE TRIGGER (Transact-SQL)

<https://learn.microsoft.com/ru-ru/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver16>

2. Триггеры. Определение триггеров

<https://metanit.com/sql/sqlserver/12.1.php>

## ЛАБОРАТОРНАЯ РАБОТА 10. ИСПОЛЬЗОВАНИЕ КУРСОРОВ

### 1. ЦЕЛЬ РАБОТЫ

Целью работы является получение практических навыков задания бизнес-логики на стороне сервера с использованием курсоров. В связи с этим задачами работы являются следующие:

- Изучением принципов работы и особенностей построений курсоров сервера;
- Созданием хранимых процедур с использованием сервера на основе задания.

### 2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Курсор представляет собой механизм организации упорядоченного набора данных, в котором можно выполнять операции над отдельными строками. Этим он отличается от стандартных результирующих наборов, в которых операции выполняются сразу над всеми строками. Курсоры позволяют *построчно* обрабатывать данные, содержащиеся в некоторой таблице или в результате выборки. С помощью курсоров SQL Server можно перемещаться по результирующему набору.

Курсоры можно использовать:

- в блоке операторов управления выполнением программы;
- внутри хранимой процедуры;
- внутри триггера.

При использовании курсора необходимо выполнить следующие действия.

- Объявить курсор;
- Открыть;
- Осуществить выборку данных из курсора;
- Обработать данные;
- Закрыть курсор.

## Объявление курсора

Для объявления курсора используется оператор DECLARE. В данном операторе задаётся имя, применяемое для ссылки на набор данных, организуемый курсором, и запрос, с помощью которого создаётся набор данных курсора.

### Синтаксис объявления курсора

*Синтаксис в стандарта SQL-92:*

DECLARE имяКурсора [INSENSITIVE] [SCROLL] CURSOR

FOR операторSELECT

[FOR {READ ONLY | UPDATE [OF списокСтолбцов]}]

Значение отдельных элементов объявления курсора согласно стандарту SQL-92 приведено в таблице (3)

В MS SQL SERVER используется расширенный синтаксис оператора:

```
DECLARE          имя_курсора          CURSOR
[          LOCAL          |          GLOBAL          ]
[          FORWARD_ONLY          |          SCROLL          ]
[  STATIC  |  KEYSET  |  DYNAMIC  |  FAST_FORWARD  ]
[  READ_ONLY  |  SCROLL_LOCKS  |  OPTIMISTIC  ]
[          TYPE_WARNING          ]
FOR
                                select_statement
[ FOR UPDATE [ OF список_столбцов] ]
```

Значение отдельных элементов синтаксиса объявления курсора используемого в Transact SQL приведено в таблице (4).

### 3. Значения синтаксических элементов, принятых в SQL 92

Параметр	Описание
имя курсора	Имя курсора
INSENSITIVE	Указывает, что изменения в источнике данных не будут отражаться в курсоре. При выборе этого параметра обновления курсора не разрешены
SCROLL	Разрешает использование следующих FETCH-

	команд: PRIOR, FIRST, LAST, ABSOLUTE л и RELATIVE л
Оператор SE- LECT	Оператор SQL SELECT. При использовании следующих команд курсор объявляется как IN- SENSITIVE: DISTINCT. UNION, GROUP BY и/или HAVING
READ ONLY	Запрещает обновления курсора
UPDATE [OF список столбцов]	Разрешает обновления курсора. Необязатель- ное предложение [OF список столбцов] опреде- ляет, какие столбцы в курсоре можно обновлять

4. Расширенный синтаксис оператора объявле-  
ния  
курсора используемый в Transact SQL

Параметр	Описание
FORWARD ONLY	Указывает, что для навигации курсора будет исполь- зоваться только команда FETCH NEXT. Используйте этот параметр для минимизации ресурсов и блокиро- вок курсора
STATIC	Указывает на то, что модификации курсора запреще- ны и изменения, внесенные в базовые таблицы после его открытия, не будут отражены в <i>курсоре</i> . Исполь- зуйте этот параметр в случае, когда курсор предна- значен только для чтения, а также для минимизации ресурсов и блокировок курсора
KEYSET	Указывает на фиксированный порядок членов кур- сора. При этом в курсоре используются только ключи базовых таблиц. KEYSET-курсор переходит в STATIC-курсор, если базовая таблица (таблицы) не содержит уникальных индексов или первичных ключей
DYNAMIC	Указывает на то, что изменения, внесенные в базовую таблицу после открытия курсора, будут отражаться в курсоре. Используйте этот параметр для достижения максимальной согласованности данных. Однако сле- дует помнить, что этот параметр требует значитель- ных затрат ресурсов по сравнению с такими типами

	курсоров, как KEYSET и STATIC
FAST_FORWARD	Определяет курсор как FORWARD ONLY и READ ONLY. Этот параметр добавлен с целью оптимизации и достижения максимальной производительности
SCROLL LOCKS	Указывает, что в результирующем наборе <i>курсора</i> , будет использоваться блокировка данных. Блокировка выполняется при чтении данных в курсор. Этот параметр гарантирует, что обновления или удаления строк, связанных с курсором, всегда будут выполнены успешно, поскольку данные заблокированы курсором. Следует избегать применения этого параметра при одновременном доступе к данным курсора со стороны нескольких пользователей
OPTIMISTIC	Указывает на то, что при чтении в курсор данные не блокируются. Обновления или удаления строк данных, связанных с курсором, могут завершиться неудачей, если данные, лежащие в основе курсора, <i>были</i> изменены после их прочтения в курсор. Этот параметр следует использовать при одновременном доступе к данным курсора со стороны нескольких пользователей

### Примеры объявления курсора:

*a. Стандартный курсор*

```
DECLARE pub_crsr CURSOR
FOR
SELECT pub_id, pub_name
FROM publishers
```

*b. Курсор, предназначенный только для чтения*

```
DECLARE pub_crsr CURSOR
FOR
SELECT pub_id, pub_name
FROM publishers
FOR READ_ONLY
```

с. *Курсор, допускающий обновления*  
 DECLARE pub\_crsr CURSOR  
 FOR  
 SELECT pub\_id, pub\_name  
 FROM publishers  
 FOR UPDATE

### Открытие курсора

После объявления курсор нужно открыт оператором open  
 Синтаксис оператора open  
 OPEN [GLOBAL] имяКурсора

Пример

OPEN pub\_crsr

### Выборка данных из курсора

Из открытого курсора можно прочесть данные построчно.  
 Для этого используется оператор FETCH (выталкивание из курсора данных в переменные).

### Синтаксис оператора FETCH

```

FETCH [ [ NEXT | PRIOR | FIRST | LAST
        | ABSOLUTE { n | @nvar }
        | RELATIVE { n | @nvar }
      ]
      FROM
      { [ GLOBAL ] имяКурсора }
[ INTO @имяПеременной ]
  
```

В таблице 10.3 приведено описание элементов синтаксиса команды FETCH.

5. Описание элементов оператора FETCH

Элемент	Описание
NEXT	Считывает следующую строку
PRIOR	Считывает предыдущую строку

FIRST	Считывает первую строку
LAST	Считывает последнюю строку
ABSOLUTE n	Считывает строку с указанным абсолютным номером из результирующего набора
RELATIVE л	Считывает строку, номер которой указан относительно текущей позиции внутри результирующего набора
<i>имя курсора</i>	Имя открытого курсора
INTO @имя переменной, @имя переменной и т.д.	Копирует содержимое столбцов в указанные переменные

При использовании аргумента *absolute* или *relative* величина перемещения в обратном направлении задается с помощью отрицательных чисел. Причем для аргумента *absolute* строки отсчитываются назад, начиная с последней строки набора записей, а для аргумента *relative* – начиная с текущей строки набора записей.

Пример 1. Прочитать следующую строку из результирующего набора:

```
FETCH NEXT FROM pub_crsr
```

Пример 2. Прочитать строку 5 из результирующего набора:

```
FETCH ABSOLUTE 5 FROM pub_crsr
```

Пример 3. Вывести содержимое следующей строки в переменные:

```
FETCH NEXT FROM pub_crsr INTO @pub_id, @pub_name
```

Выборка данных используется в основном вместе с циклом WHILE (а).

*а. Организация выборки данных в цикле*

```
Fetch next from pub_crsr -- первая выборка
-- устанавливает значение @@Fetch_status
While @@Fetch_status=0 --выбираем в цикле строки
-- курсора
```



```
Begin
    Fetch next from pub_crsr —выбираем следующую --
строку
End
```

### **Закрытие и освобождение курсора**

Завершив обработку данных курсора, необходимо закрыть его (с помощью оператора CLOSE) и освободить занимаемые им системные ресурсы (с помощью оператора DEALLOCATE). Оператор CLOSE закрывает курсор, но не освобождает используемые им структуры данных. Применяйте этот оператор, если вы планируете в дальнейшем повторно открывать курсор. Оператор DEALLOCATE закрывает курсор и освобождает структуры используемых им данных.

Синтаксис операторов CLOSE и DEALLOCATE

CLOSE [GLOBAL] имя\_курсора

DEALLOCATE [GLOBAL] имя\_курсора

Пример 1. Закрывать курсор

CLOSE pub\_crsr

Пример 2. Освободить курсор

DEALLOCATE pub\_crsr

### **Модификация таблиц с помощью курсоров**

Помимо считывания информации из курсора, можно выполнять построчное обновление и удаление содержащихся в нем данных. При модификации курсора соответствующие изменения будут автоматически распространяться и на источник данных курсора. Такие действия выполняются позиционными операторами UPDATE и DELETE.

Синтаксис операторов UPDATE

UPDATE имяТаблицы

SET имяСтолбца1 = (выражение1 | NULL | (операторSelect)

[, имя\_столбца 2 = {выражение2 | NULL | (операторSelect)}...]

WHERE CURRENT OF имяКурсора

В данном случае будет модифицироваться значения атрибутов таблицы имяТаблицы но только у строки, на которую спозиционирован курсор. Служебная переменная CURRENT указывает на текущую запись набора курсора.

Синтаксис оператора delete, используемый в курсоре аналогичен update.

DELETE FROM имяТаблицы WHERE CURRENT OF имяКурсора

**Пусть требуется обновить столбец pub\_name в таблице publishers**

*b. В данном примере выполняется обновление текущей строки в курсоре:*

```
UPDATE publishers  
SET pub_name = 'XYZ publisher'  
WHERE CURRENT OF pub_crsr
```

**Пусть требуется удалить строку в таблице publishers**

*В данном примере будет удалена текущая строка в курсоре:*

```
DELETE FROM publishers WHERE CURRENT OF pub_crsr
```

**Глобальные переменные, работающие с курсорами**

Для контроля за состоянием курсора можно использовать две глобальные переменные:

@@Fetch\_\_Status и @@Cursor\_Rows.

В переменной @@Fetch\_Status сохраняется состояние последней команды FETCH. Ниже приведены возможные значения данной переменной Transact-SQL.

-1 - неудачное выполнение выборки или попытка прочитать данные за пределами результирующего набора

-2 - Выбираемая строка выпадает из набора данных

Переменная @@Fetch\_Status обычно используется для организации цикла по строкам курсора.

!!Для установки начального значения @@Fetch\_Status пер-

вая команды fetch должна быть выполнена перед проверкой условия завершения цикла (при организации цикла for до начала цикла)

Fetch from имяКурсора into переменные..

WHILE @@Fetch\_Status = 0

Fetch from имяКурсора into переменные..

В переменной @@Cursor\_Rows сохраняется количество строк в результирующем наборе курсора. Эту переменную можно использовать *после* открытия курсора. Ниже приведены возможные значения переменной @@Cursor\_Rows.

-n Курсор в текущий момент загружается данными. Возвращаемое число означает текущее количество строк в ключевом результирующем наборе, однако это число продолжает увеличиваться по мере того, как SQL Server обрабатывает оператор SELECT;

n Количество строк в результирующем наборе

0 Результирующий набор не содержит строк

-1 Динамический тип курсора, поэтому количество записей неизвестно

### **Пример использования курсоров. Циклический просмотр таблицы**

Ниже приведен код, в котором используются различные компоненты курсора

(DECLARE, OPEN, FETCH и DEALLOCATE) для циклического просмотра таблицы publishers. При каждом выполнении оператора FETCH происходит обращение к глобальной переменной @@Fetch\_Status. После того как указатель записей достигнет конца результирующего набора, переменная @@Fetch\_Status станет равной -1, и поэтому выполнение кода внутри цикла WHILE @@Fetch\_Status = 0 прекратится.

```
-- Объявляем курсор, содержащий столбцы pub_id и pub_name, из таблицы publishers
```

```
DECLARE pub_crshr CURSOR
```

```
FOR
```

```
SELECT pub_id, pub_name FROM publishers
```

```

-- ОТКРЫВАЕМ курсор
OPEN pub_crsr
-- Выбираем первую строку из курсора для установки --
значения @@Fetch_Status = 0
FETCH NEXT FROM pub_crsr
-- Выбираем в цикле строки курсора
WHILE @@Fetch_Status = 0
BEGIN
-- Выбираем следующую строку
FETCH NEXT FROM pub_crsr
END
-- Закрываем курсор и освобождаем память
DEALLOCATE pub_crsr

```

## 6. Результат выполнения приведенного кода

<b>pub_id</b>	<b>pub_name</b>
0736	New Moon Books
0877	Binnet & Hardley
1389	Algodata Infosys- tems
1622	Five Lakes Publish- ing
1756	Ramona Publishers
9901	GGG&G
9952	Scootney Books
9999	Lucerne Publishing

## 3. ПОРЯДОК ВЫПОЛНЕНИЯ И ЗАДАНИЯ ДЛЯ РАБОТЫ

1. Создать курсоры – сведение нескольких записей в одну. Например, для отчетности: составить сводную таблицу, в которой у каждого человека будут указаны все его должности по совместительству через запятую. Пример сведения записей в одну строку приведён в приложении.

2. Создать курсоры для вычисления разницы атрибутов у

кортежей (строк) таблицы, отсортированных по определенному критерию.

3. Осуществить с помощью курсоров поиск информации в таблице с помощью курсоров по определенным критериям, последовательным просмотром строк некоторого набора.

4. Осуществить с помощью курсора модификацию информации в кортеже, найденном по требуемому условию.

#### 4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие функции для расширения возможностей обработки данных поддерживают курсоры?

2. Какая последовательность действий при работе с курсором?

3. Описать синтаксис операторов используемых при работе с курсором?

4. Как контролировать работу курсора серверными переменными?

#### 5. ЛИТЕРАТУРА

1. Курсоры (SQL Server). <https://learn.microsoft.com/ru-ru/sql/relational-databases/cursors?view=sql-server-ver16>

2. Дейт, К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом «Вильямс», 2005. — 1328 с.: ил.

3. Пример курсора в SQL Server

[https://sql-ex.com/blogs/?/Primer\\_kursora\\_v\\_SQL\\_Server.html](https://sql-ex.com/blogs/?/Primer_kursora_v_SQL_Server.html)

## ПРОСТЕЙШИЕ ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ

### ***Вставка строки***

INSERT INTO имяТабл ([имяПоле1], [имяПоле2],[...])  
VALUES(1значение1, значение>[,...])

### ***Изменение значения поля в строке:***

update имяТаблицы  
set имяСтолбца = значение  
[where условие];

### ***Удаление строки:***

delete from имя\_таблицы  
[where условие];

## СТАНДАРТНЫЕ ФУНКЦИИ SQL SERVER

### **Аггрегативные функции**

Возвращают суммарные значения.

AVG	COUNT(*)	MIN
COUNT	MAX	SUM

### **Функции манипуляции датой и временем:**

DATEADD	добавляет к дате отрезок
DATENAME	возвращает часть даты строкой
GETDATE	текущая дата
DATEDIFF	вычисл разницу
DATEPART	возвращает определенную часть заданной даты

## Математические функции

ABS	модуль
RAND	Генерация случайного числа
ROUND	округление
FLOOR	округление в меньшую сторону
CEILING	округление в большую сторону
SQRT	Корень квадратный
POWER	Возведение в степень
EXP	вычисление экспоненты
LOG	натуральный логарифм
LOG10	
PI	
DEGREES	возвращает градусы из радиан
RADIANS	Преобразует градусы в радианы
SIN, COS, TAN, COT	Тригонометрические функции
ACOS, ASIN, ATAN , ATN2	
SIGN	возвращает

## Niladic-функции

Эти функции возвращают различные системные значения.

CURRENT_TIMESTAMP- текущая дата	SYSTEM_USER R (сист.пользов)
CURRENT_USER – текущий пользователь (например DBO)	USER= current_user
SESSION_USER -сессия	

## Функции для манипуляции со строками

ASCII	переводит в соответствующую кодировку
STR	конвертирует в строку
SPACE	Возвращает пробелы
CHAR	Возвращает номер символа в кодировке
REPLICATE	удваивает строку
STUFF	вставляет строку в другую
CHARINDEX	порядковый номер символа,
REVERSE	зеркалирует строку
SUBSTRING	Возвращает подстроку
RIGHT	возвращает правую часть строки
UPPER	Возвращает строку в верхнем регистре
LOWER	Возвращает строку в нижнем регистре
LTRIM	Удаляет пробелы из начала строки
RTRIM	Удаляет пробелы из конца строки

## Системные функции

	Возвращаемое значение
HOST_NAME	имя сервера
DB_ID ([name DB])	номер БД(default текущей)
DB_NAME	Имя текущей БД
COL_LENGTH	Длина колонки в таблице
IDENT_SEED	
INDEX_COL	
DATALength	возвращает размер данных поля
ISNULL	Проверяет на значение NULL



## Функции для преобразования различных типов данных.

CAST	конвертирует типы данных
CONVERT(требТип, Данных)	Конвертирует данное (Данных) в требуемый тип (требТип)

### ПРИМЕР ОТЧЕТА ПО ЛАБОРАТОРНОЙ №1

**Задание: Разработать БД, отображающую музыкальные произведения, их исполнителей, авторов и музыкальные стили**

#### **Объекты:**

- 1) Произведение - play; Соответствует музыкальному произведению как таковому, может не исполненному никем;
- 2) Person – содержит данные об авторах, исполнителях, вообще всех персонах, как то связанных с музыкальными произведениями;
- 3) Стил – style - содержит данные об музыкальных стилях;
- 4) Ispolnenie – содержит данные об исполнении музыкального произведения каким либо исполнителем.

#### **Атрибуты:**

- Произведения (proizv):
  1. id\_play – числовой тип integer; код произведения искусственный атрибут, введен для идентификации произведения
  2. Название (play) – текстовый тип varchar(50);
  3. id\_avts – числовой тип integer; код автора слов используется для указания на автора, сочинившего слова
  4. id\_avtm – числовой тип integer. ; код автора музыки используется для указания на автора, сочинившего музыку
- Стил (style):
  1. id\_style – искусственный атрибут, используется для обозначения стиля числовой тип integer;
  2. Название (style) – текстовый тип varchar(50).
- ispolnenie
  1. id\_play – код исполнения – искусственный атрибут, введен для обозначения произведения ;
  2. id\_avt – код исполнителя, ссылка на таблицу person– чис-

ловой тип integer;

3. id\_style – числовой тип integer ссылка на значение кодов персон в табл. person;

### **Связи:**

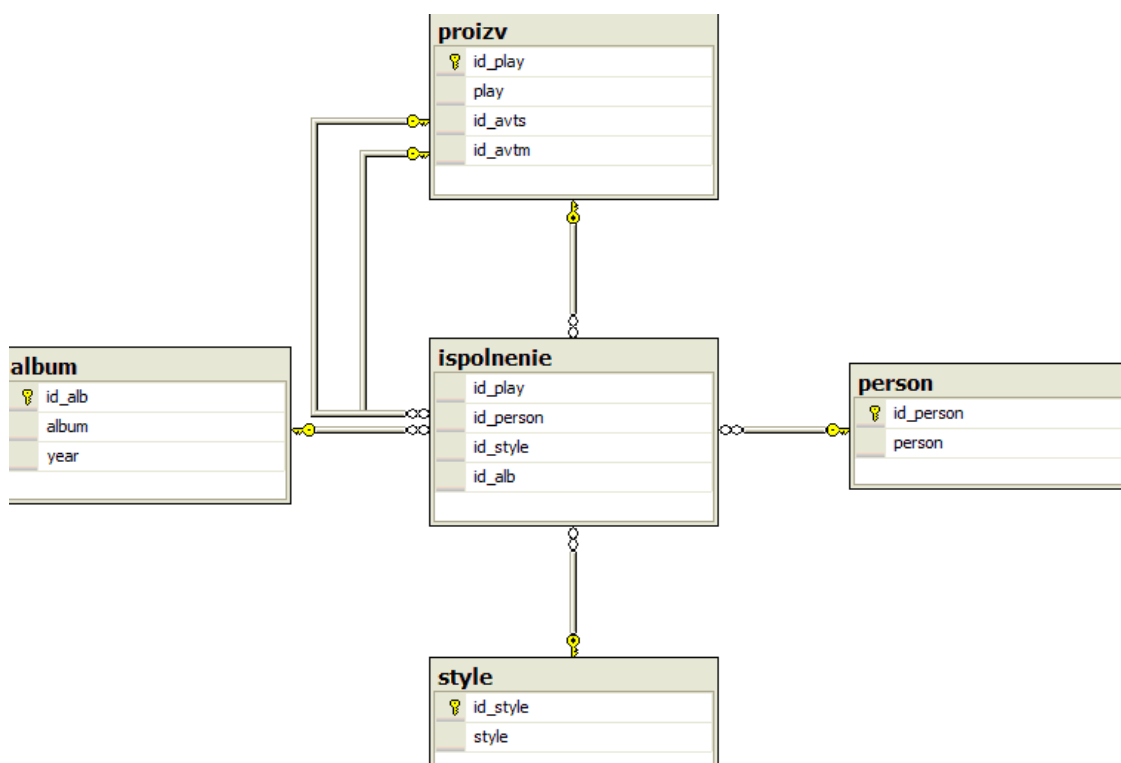
1. Произведение связано с исполнение связью один к многим, так как одно произведение может иметь много вариантов исполнений.

2. Группа связана с исполнение связью один к многим, так как одна группа может записать много исполнений произведений;

3. Стиль связан с исполнение связью один к многим, так как в одном стиле может быть сделано много исполнений;

4. Альбом связан с исполнение связью один к многим, так как один альбом может содержать много исполнений.

Диаграмма связей в БД.



Таблицы, соответствующие выделенным отношениям.

Таблица album

Table - dbo.proizv	Table - dbo.ispolnenie	Table - dbo.person	Table - dbo.style	Table - dbo.album
	id_alb	album	year	
	1	Стихия огня	2006	
	2	Смутное время	1997	
	3	Неформат	2000	
	4	See You The Other Side	1996	
	5	Aska	2005	
	6	Крылья	2005	
►*	NULL	NULL	NULL	

Таблица person

Table - dbo.proizv	Table - dbo.ispolnenie	Table - dbo.person	Table - dbo.style	Table - dbo.dl
	id_person	person		
►	1	Aska		
	2	Catharsis		
	3	Легион		
	4	Ария		
	5	Korn		
	6	Маврин		
*	NULL	NULL		

Таблица proizv

Table - dbo.proizv	Table - dbo.ispolnenie	Table - dbo.person	Table - dbo.style	Table - dbo.dl
	id_play	play	id_avts	id_avtm
►	1	Вольная птица	3	3
	2	Я свободен	4	6
	3	Стая	6	6
	4	Twisted Transistor	5	5
	5	Angels of war	1	1
	6	Крылья	2	2
*	NULL	NULL	NULL	NULL

Таблица style

Table - dbo.proizv	Table - dbo.ispolnenie	Table - dbo.person	Table - <b>dbo.style</b>	Table - dbo.album
	id_style	style		
▶	1	Heave metall		
	2	Classik rock		
	3	Melodik metall		
	4	NU metall		
*	NULL	NULL		

Таблица ispolnenie

Table - dbo.proizv	Table - <b>dbo.ispolnenie</b>	Table - dbo.person	Table - dbo.style	Table - dbo.album
	id_play	id_person	id_style	id_alb
▶	1	3	1	1
	2	4	1	2
	3	6	1	3
	4	5	4	4
	5	1	2	5
	6	2	3	6
*	NULL	NULL	NULL	NULL

ПРИМЕР ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ №5  
Дана информация об установленном оборудовании

Отдел	НачОтдела	ЦехИнвНомер	Модель	Стоимость	СрокСлужбы
Цех1	Сидоров	2	1К62	160000	4.5
Цех1	Сидоров	3	1Т62	160000	2
Цех2	Петров	2	2Ф14	260000	5

1. Строим диаграмму зависимостей

А) В качестве ключа приняты атрибуты (Отдел, ЦехИнвНомер). По значению данных атрибутов можно определить любой кортеж (строку, запись).

В) Частичные зависимости:

Отдел → НачОтдела,

ЦехИнвНомер → Модель

Данная зависимость  $\text{Отдел} \rightarrow \text{НачОтдела}$  показывает, что по номеру отдела можно определить его начальника. Зависимость  $\text{ЦехИнвНомер} \rightarrow \text{Модель}$  показывает, что по цеховому номеру станка можно определить его модель, то есть цеховой номер станка подразумевает некоторый конкретный станок, который имеет некоторую модель.

С) Транзитивная зависимость:

$\text{Модель} \rightarrow \text{Стоимость}$ .

Данная зависимость подразумевает то, что стоимость станка определяется его моделью, что соответствует условиям предметной области.



2. Для приведения ко 2НФ выделяем объекты “Отдел” и “Модель” в отдельные отношения.

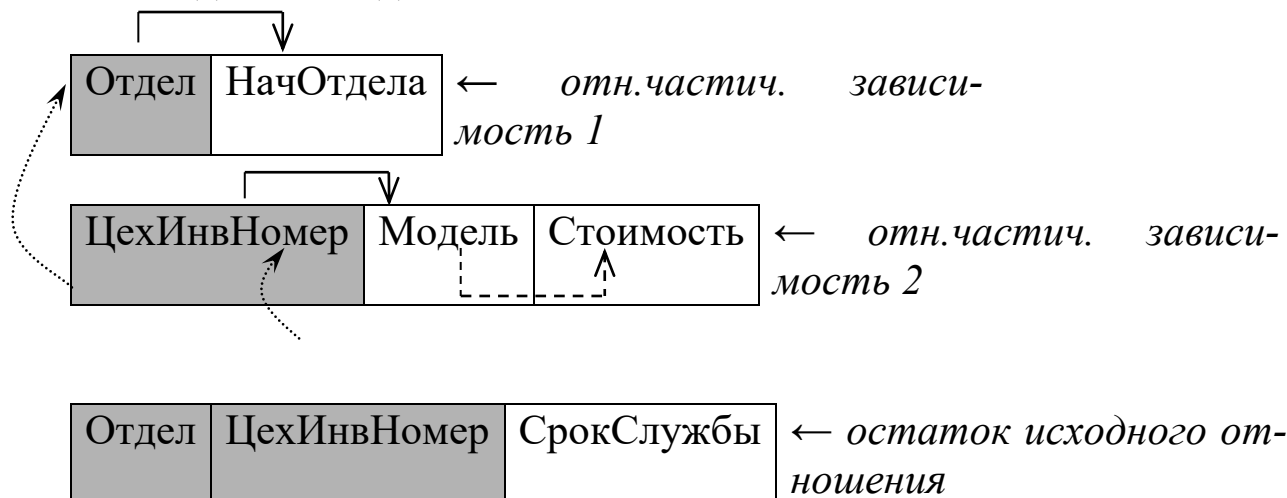
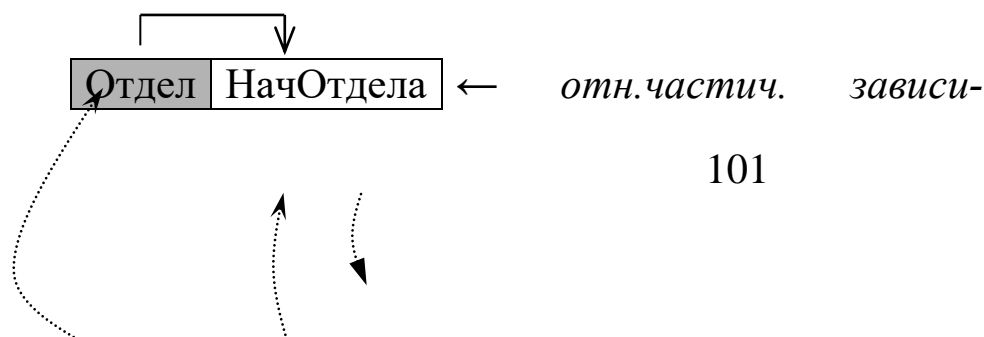


Рисунок П.1. Преобразованная диаграмма зависимостей.

3. Для приведения к 3NF выделяем транзитивную зависимость “Модель-Стоимость” в отдельные отношения.



		мость 1
--	--	---------

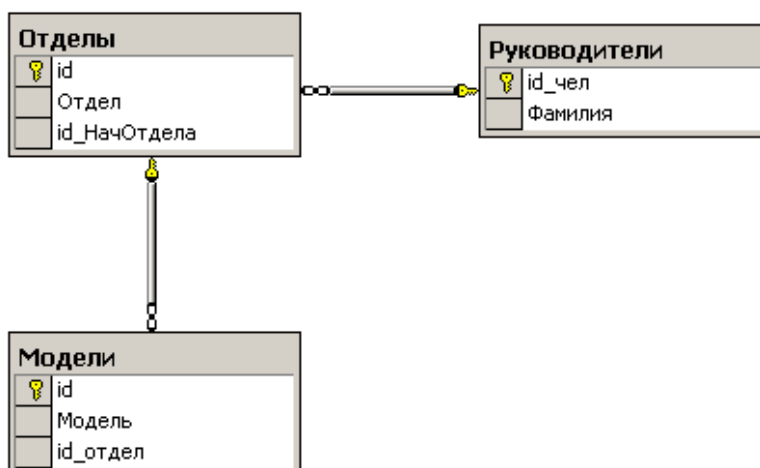
ЦехИнвНомер	Модель	← отн.частич. зависимость 2
-------------	--------	--------------------------------

Модель	Стоимость	← отн.транзитивная зависимость
--------	-----------	--------------------------------

Отдел	ЦехИнвНомер	СрокСлужбы	← остаток исходного отношения
-------	-------------	------------	-------------------------------

Рисунок П2. Преобразованная диаграмма зависимостей.

#### 4. Строим связь отношений в Enterprise Manager



5. Формируем запрос, позволяющий получать требуемую информацию

```

SELECT M.Модель, O.Отдел, R.Фамилия
FROM dbo.Модели M INNER JOIN Отделы O
ON M.id_отдел = O.id
INNER JOIN Руководители R
ON O.id_НачОтдела = R.id_чел
  
```

Результат:

1K62	Цех1	Иванов
1T65	Цех2	Петров
C1E12	Цех2	Петров

## РАБОТА С КУРСОРАМИ. ПРИМЕР СВЕДЕНИЯ НЕСКОЛЬКИХ ЗАПИСЕЙ В ОДНУ С ПОМОЩЬЮ КУРСОРА.

Пусть даны таблицы `ostTov(idtov, nameTov, kolTov)` и `dvTov(idTov, kolTov, idDvig)`. Таблица `ostTov` показывает остатки определённого товара, таблица `dvTov` показывает движение для определённого `idTov`.

	idtov	nameTov	KolTov
► 1	1	Молоко	3
	3	Сливочки	2
	4	Макарон...	10
	5	КрупМан	1
	6	Йогур	10
	7	Сметана	5
	8	Карачин...	10
	13	Сливы	5
*	NULL	NULL	NULL

	idtov	kolTov	idDvig
► 1	1	2	2
	1	-2	3
	1	-2	4
	2	3	5
	2	-1	6
	1	2	7
	2	1	8

Рисунок П.1 Таблицы для сведения записей в одну.

Пусть необходимо вывести данные о движении товаров так, чтобы для каждого товара были выведены `kolTov` из всех строк таблицы `dvig`, относящихся к данному товару.

*Курсор для сведения нескольких строк в одну.*

```
declare @idT int, @idT0 int, @nameT nvarchar(12), @kolT int,
@idDv int
declare @strkoldv nvarchar(100)
```

```
declare myCur2 cursor
for select idTov, kolTov, idDvig from dvig order by idTov
open myCur2
fetch next from myCur2 into @idT, @kolT, @idDv
--Задаём начальные условия для первой строки
```

```

set @idT0=0
while @@FETCH_STATUS=0
begin
--Проверка на новый товар.
if @idT0<>@idT
--Если товар новый, то выводим накопленные данные для
товара @idT0
begin
print 'Код товара -'+convert(nvarchar,@idT0)+'Количества
Поставки- '+@strkoldv
-- и задаём новый текущий код - idT0
set @idT0=@idT
set @strkoldv=convert(nvarchar, @kolT )
end
else
-- иначе наращиваем строку движения
set @strkoldv=@strkoldv+', '+convert(nvarchar, @kolT )
fetch next from myCur2 into @idT, @kolT, @idDv
end
print 'Код товара -'+convert(nvarchar,@idT)+'Количества По-
ставки- '+@strkoldv

close myCur2
deallocate myCur2

```

```

exec svStr

```

%

Сообщения

```

Код товара -1Количества Поставки- 2,-2,-2,2
Код товара -2Количества Поставки- 1,3,-1

```

Время выполнения: 2024-01-29T16:51:36.2737717+07:00

Рисунок П.2 Результат работы курсора