Федеральное государственное образовательное бюджетное учреждение
высшего образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)
Новороссийский филиал

Кафедра «Информатика, математика и общегуманитарные науки»

Д.В. Тимшина

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

Базы данных

Направление подготовки: 38.03.05 Бизнес-информатика
Направленность(профиль): ИТ-менеджмент в бизнесе
Программа подготовки: академическая
Форма обучения: заочная
Квалификация (степень) выпускника: Бакалавр

Новороссийск
2017

Тимшина Д.В. Базы данных. Методические рекомендации предназначены для студентов, обучающихся по направлению 38.03.05 «Бизнес-информатика», профиль ИТ-менеджмент в бизнесе (программа подготовки бакалавра, заочная форма обучения) – Новороссийск: Новороссийский филиал Финуниверситета, кафедра «Информатика, математика и общегуманитарные науки», 2017. – 105 с.

Методические рекомендации содержат комплекс требований и методические материалы для освоения дисциплины «Базы данных».

СОДЕРЖАНИЕ

ЦЕЛЬ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Цель освоения дисциплины – формирование знаний, умений и практических навыков создания и эксплуатации баз данных (БД). В результате изучения дисциплины у обучающихся формируются профессиональные компетенции.

В ходе семинарских (практических) занятий может использоваться следующее ПО:

- табличный процессор MS EXCEL;
- текстовый процессор MS Word;
- свободная объектно-реляционная СУБД PostgreSQL (программы: Reload Configuration, Restart Server, Start Server, Stop Server, программа-клиент SQL Shell (psql) (стандартный терминальный клиент), pgAdmin – программа работы с графическим пользовательским интерфейсом)

или MySQL, свободная реляционная система управления базами данных, dbForge Studio for MySQL, универсальное решение для разработки, администрирования и управления базами данных MySQL и MariaDB. Данный продукт позволяет создавать и выполнять запросы, разрабатывать и отлаживать процедуры и функции, а также автоматизировать управление объектами баз данных MySQL с помощью удобного пользовательского интерфейса. dbForge Studio также содержит инструменты для сравнения, синхронизации, создания резервных копий баз данных по графику, а также для анализа и создания отчетов по данным таблиц MySQL

или СУБД MS SQL Server с компонентами Analysis Services.

В начале семинарского (практического) занятия преподаватель объясняет цель занятия, объявляет форму выполнения занятия, требования к форме и содержанию отчета, дает рекомендации по технологии выполнения заданий, используемому ПО и источникам.

Лекционные занятия проводятся в соответствии с тематическим планом, при изложении материала рекомендуется использовать презентации в среде MS PowerPoint.

В ходе интерактивных занятий следует проводить разбор конкретных примеров, максимально приближенных к реальным данным, соответствующих экономической и финансовой информации.

Основное внимание при проведении семинарских (практических) занятий следует уделять развитию навыков формирования рациональных схем данных предметной области, реализации этих схем в среде современных СУБД с использованием языка структурированных запросов – SQL, формирования сложных содержательных запросов по выбору данных, использования методов и алгоритмов анализа данных.

При этом задача состоит в обучении профессиональным навыкам разработки и использования баз данных и анализа данных в среде современных СУБД.

Проведение семинарских (практических) занятий осуществляется в компьютерных классах и включает в себя реализацию всех этапов создания и использования баз данных в среде СУБД (создание базы данных, реализация схемы базы данных, включая все таблицы, связи, ограничения, индексы, ввод и редактирование данных, создание и отладка сложных логических прикладных запросов, анализ и оптимизация выполнения сформированных запросов), также использования методов анализа данных (подготовки исходных данных, создания структур для анализа, использования альтернативных алгоритмов обработки для созданных структур). Следует обратить внимание, что примеры данных в таблицах должны обеспечить получение корректных и полных результатов запросов.

Семинарские (практические) занятия в компьютерных классах позволяют студентам сформировать навыки работы с современными СУБД.

Внедрение активных и интерактивных элементов в проведение занятий по дисциплине может осуществляться разными методами: семинар с групповым обсуждением, опрос, компьютерный эксперимент, работа студентов над проектами в составе групп (2-3 человека) и др.

Интерактивная форма проведения занятий способствует формированию профессиональных компетенций для успешного освоения основных дисциплин блока программы. Реализация интерактивной формы обеспечивается базой данных прикладной предметной области, коллективной работой над решениями задач, отсутствием единственного решения, единой целью в поиске решения. Конечная цель - выработать у студентов умение реализовывать и оценивать альтернативные варианты различных аспектов функционирования современных СУБД и аналитических систем.

Учебным планом по дисциплине «Базы данных» предусмотрено выполнение домашнего творческого задания (ДТЗ).

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ДОМАШНЕГО ТВОРЧЕСКОГО ЗАДАНИЯ

1. Цель выполнения домашнего творческого задания и его структура

В соответствии с учебным планом в процессе изучения дисциплины «Базы данных» студенты, обучающиеся по направлению 38.03.05 «Бизнес-информатика», профиля «ИТ-менеджмент в бизнесе», выполняют домашнее творческое задание (ДТЗ).

Цель ДТЗ – закрепить и систематизировать знания и навыки, полученные в ходе изучения дисциплины, и научиться применять полученные знания проектирования и ведения баз данных.

Студент должен изучить рекомендуемую учебную литературу и ознакомится с методическими рекомендациями по выполнению домашнего творческого задания.

Для выполнения ДТЗ студенту рекомендуется подобрать и изучить литературу, изданную не ранее последних пяти лет от года выполнения ДТЗ. Это могут быть учебные пособия и учебники, рекомендованные по направлению подготовки бакалавров в высшей школе, а также монографии, статьи из журналов, интернет-ресурсы и др.

Структура ДТЗ следующая:

Титульный лист
Содержание
Введение
< Основная часть:>
1. Номер варианта и тема ДТЗ. Постановка задачи предметной области
2. Проектирование БД. ER-модель базы данных и описание алгоритма ее построения
3. Формирование базы данных в СУБД. Описание алгоритма заполнения БД исходной информацией с помощью языка SQL
4. Результаты выполнения SQL-запросов к базе данных
Заключение
Список литературы

Приложения

Титульный лист является первой страницей ДТЗ, однако он не нумеруется. Образец титульного листа ДТЗ приведен в Приложении 1 данного раздела.

В ***содержании*** необходимо привести все заголовки структурных частей ДТЗ с указанием страниц, с которых они начинаются. Последнее слово каждого заголовка соединяется отточием с соответствующим ему номером страницы в правом столбце содержания.

Во ***введении*** студент должен привести краткие характеристики ПК (объем оперативной памяти ПК, тактовая частота и тип процессора и т.п.) и программного обеспечения (ОС, СУБД и проч., названия и версии приложений и фирм-разработчиков), использованного для выполнения и оформления ДТЗ

Заключение должно содержать собственные *выводы* студента, полученные в результате проведенной работы.

Литературные источники – это учебники и учебные пособия, рекомендованные для студентов высших учебных заведений, журналы, электронные издания и др., указанные в списке использованной литературы, оформленные в соответствии с правилами и относящиеся к последним пяти годам.

В ***списке литературы*** студент приводит литературу, использованную им в процессе написания ДТЗ. В список должны включаться только те литературные источники, на которые имеются ссылки, приведенные в ДТЗ.

В ***приложениях*** приводят материалы, которые дополняют работу. По форме данные материалы – это скриншоты результатов запросов, схемы ER-моделей и пр. Каждое приложение должно начинаться с новой страницы с указанием в правом верхнем углу слова «Приложение» и номера, а также должно иметь тематический заголовок. При наличии в работе более одного приложения необходимо нумеровать их арабскими цифрами. Например:

Приложение 2

ER-модель
< Материалы приложения >
. . . . . .

Связь основного текста с приложениями осуществляется через ссылки.

2. Требования к оформлению отчета по домашнему творческому заданию

ДТЗ оформляется на ПК с использованием текстового процессора Microsoft Word на листах формата А4, ориентация – книжная.

Следует установить следующие размеры полей страницы: левое поле – 3 см, правое, верхнее и нижнее – 2 см.

Требования к оформлению текста ДТЗ:

– отступ первой строки (абзацный отступ) – 1,25 см;

– междустрочный интервал – 1,5 строки;

– гарнитура шрифта – Times New Roman;

– кегль шрифта (размер) – 14 пунктов;

– форматирование текста (выравнивание) – по ширине.

Текст должен быть оформлен в соответствии с ГОСТ 7.32-2001 «Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления» - http://protect.gost.ru/document.aspx?control=7&id=130946.

Каждую структурную часть ДТЗ нужно начинать с нового листа. Точка в конце заголовка структурной части работы не ставится.

Каждая цитата, заимствованные цифры, факты должны сопровождаться ссылкой на источник, описание которого приводится в списке использованной литературы. В ссылке указывается номер источника по списку и номера страниц, например: [7, С.45-46].

Список литературы оформляется в соответствии с требованиями ГОСТ Р 7.0.5 – 2008 «Библиографическая ссылка».

Все аббревиатуры и сокращения слов должны быть расшифрованы в тексте ДТЗ при первом употреблении.

Рисунки необходимо снабжать подрисуночной подписью, например:

< Рисунок >

. . . . . .

Рисунок 3 – Результат выполнения запроса

В конце подрисуночной подписи точку не ставят.

Все схемы и рисунки имеют одинарную сквозную нумерацию. Нельзя располагать подрисуночную подпись и рисунок на разных страницах. На все рисунки необходимо сделать ссылки в тексте ДТЗ.

Табличный материал (таблица) оформляется следующим образом. В левом верхнем углу пишут слово «Таблица» и ее порядковый номер в работе и ставится тире «-». Таблица должна иметь тематический заголовок, который располагают после «-» без точки в конце, например:

Таблица 3 - Структура таблицы базы данных

< Таблица >

. . . . . .

Допускается использование в таблице кегля шрифта (размера) – 12 пунктов.

На последней странице ДТЗ студент обязан поставить дату сдачи ДТЗ на регистрацию и свою подпись.

Отчет по выполнению ДТЗ должен быть сброшюрован по левому краю.

Образец титульного листа ДТЗ приведен в Приложении 1 этого раздела.

Объем ДТЗ не более 15 страниц, включая титульный лист и список литературы. Приложения в общем объеме ДТЗ не учитываются.

3. Варианты выполнения домашнего творческого задания

3.1 Цель выполнения домашнего творческого задания

3.1.1 Проектирование БД предметной области (ПрО) варианта. Построение ER-модели ПрО.

3.1.2 Создание БД и структуры таблиц БД с помощью языка SQL.

3.1.3 Формирование SQL-запросов и транзакций к базе данных.

3.2 Предметная область и варианты домашнего творческого задания

3.2.1 Для выполнения ДТЗ студент выбирает вариант предметной области. Варианты выполнения ДТЗ и наименование предметной области, соответствующей номеру варианта, приведены в таблице 1. Номер варианта ДТЗ определяется по номеру студента в списке журнала группы.

Таблица 1 – Варианты выполнения ДТЗ и наименование предметной области

Вариант	**Тематика задания**
1	**2**
1	Учет продовольственных товаров на рынке
2	Кинотеатр
3	Рекламное агентство
4	Книжный магазин
5	Салон красоты
6	Кафе
7	Кондитерский цех
8	Производство жалюзи
9	Фармацевтический цех
10	Магазин электронных товаров
11	Парикмахерская
12	Центр косметологии
13	Ресторан
14	Больница
15	Магазин женской одежды
16	Мебельный салон
17	Туристическая фирма
18	Молочный комбинат
19	Компьютерная фирма
20	Типография
21	Цветочный магазин
22	Центр (служба) занятости населения
23	Станция техобслуживания
24	Поликлиника
25	Риэлтерская фирма

Приложение 1

Образец титульного листа домашнего творческого задания

Федеральное государственное образовательное бюджетное учреждение высшего образования

«Финансовый университет при Правительстве Российской Федерации»

Новороссийский филиал

Кафедра «Информатика, математика и общегуманитарные науки»

Домашнее творческое задание

по дисциплине **«Базы данных»**

Исполнитель: студент
<Фамилия И.О.>
направление:
«Бизнес-информатика»
группа: <номер группы>
номер зачетной книжки:
11флб00838
Руководитель:
<уч. степень, должность
Фамилия И.О.>

Новороссийск 201_

ЛЕКЦИОННЫЙ МАТЕРИАЛ И КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ САМОПОДГОТОВКИ

Лекция 1. Информационные системы и системы баз данных. Архитектура систем баз данных

План

1. Основные понятия систем баз данных. Архитектура систем баз данных
2. Системы управления базами данных (СУБД) и базы данных (БД)
3. Архитектуры доступа к данным
4. Функции и обзор современных СУБД. Современная СУБД, как интегрированная платформа обработки информации

1. Основные понятия систем баз данных. Архитектура систем баз данных

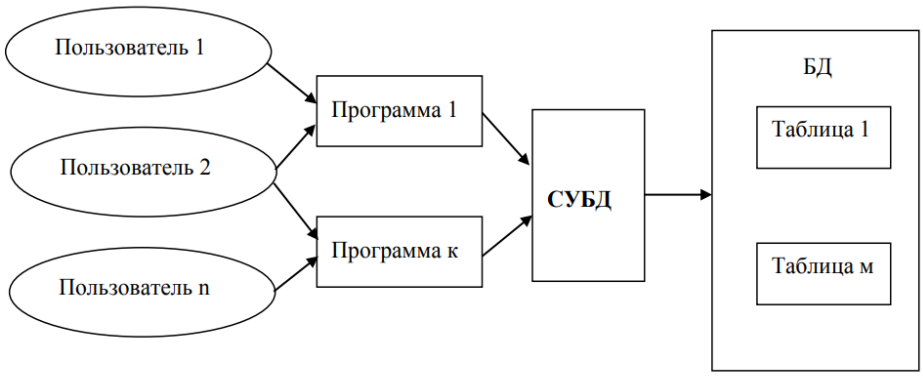
С начала развития вычислительной техники (ВТ) одним из основных направлений ее использования стало создание ***автоматизированных информационных систем*** (***АИС***).

Появление АИС связано с появлением ***жестких дисков большой емкости, обеспечивающих произвольный доступ к данным***. Это предопределило развитие АИС разного назначения и масштаба, обеспечивающих оперативную обработку в реальном времени огромных потоков данных, что привело к появлению новой ИТ интегрированного хранения и обработки данных – ***концепции баз данных***.

Причины перехода к использованию баз данных

1. Большая избыточность данных.
2. Зависимость программ пользователя от данных.
3. Сложность обеспечения целостности и непротиворечивости данных.
4. Отсутствие централизованного доступа к данным.

Слайд Использование баз данных



Основные понятия: понятие базы данных и системы баз данных, СУБД

Система баз данных (СБД - database system) – это система специально организованных данных (баз данных), программных, технических (аппаратных) и языковых средств, а также организационно-методического обеспечения для централизованного накопления и коллективного многоцелевого использования данных.

Термин «***система баз данных***» широко используется в современной англоязычной литературе для определения человеко-машинной системы, включающей **базы данных** (БД), **системы управления базами данных** (СУБД), оборудование и персонал.

В системах баз данных (СБД) иногда выделяют в качестве отдельного объекта пользователя СБД. Ранее вместо СБД использовалось название «банк данных».

База данных

1) База данных (БД) – именованная совокупность данных, отображающая состояние объектов и их отношений в рассматриваемой предметной области.

В ранних определениях БД указывалось **на отсутствие дублирования данных.** Однако дублирование может быть вызвано спецификой модели данных или технологическими причинами (обеспечение надежности, сокращение времени реакции). Но это должно быть отслеживаемое и управляемое дублирование.

2) База данных – организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей. (Кагаловский М.Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2002. -800 с.)

Системы управления данными

1) Система управления базами данных (СУБД) – совокупность программных и лингвистических (языковых) средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

2) Система управления базами данных (СУБД) – это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Иногда в составе СБД выделяют систему управления архивами.

Под оперативным управлением СУБД находится часть данных, остальные данные (архивы) располагаются на носителях, неуправляемых СУБД.

Основные требования к СБД:

• адекватность отображения предметной области (полнота, целостность, непротиворечивость и актуальность данных);

• возможность взаимодействия пользователей разных категорий, обеспечение высокой эффективности доступа;

• дружественность интерфейса;

• обеспечение секретности и конфиденциальности;

• обеспечение взаимной независимости программ и данных;

• обеспечение надежности – защита данных от случайного и преднамеренного разрушения, возможность восстановления данных в случае сбоев в системе;

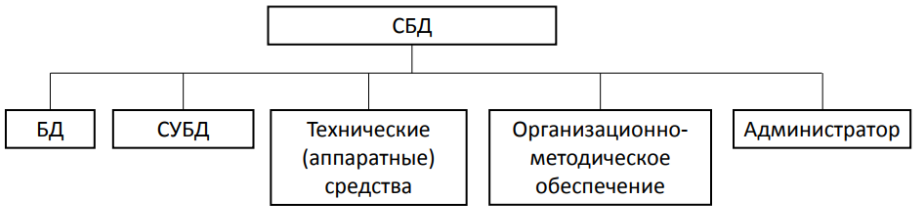
• распределенная обработка данных и обеспечение эффективного доступа пользователей к данным в любой точке сети.

Компоненты системы баз данных (СБД). Структура СБД

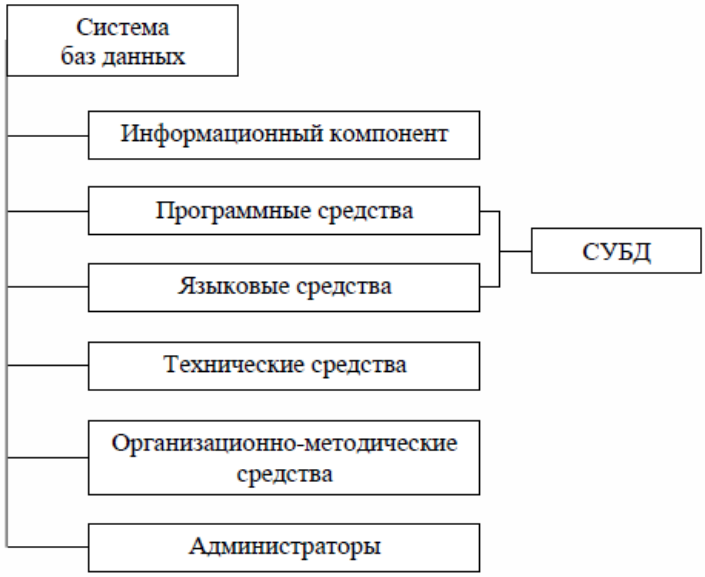
СБД – сложная человеко-машинная система, включает различные взаимосвязанные и взаимозависимые компоненты (подсистемы):

Слайд

Структура системы баз данных (СБД)



Слайд **Компоненты СБД**



Данные, отражающие состояние предметной области и используемые АИС, принято называть **информационной базой (информационный компонент).**

Информационная база включает: собственно данные; метаданные (описания этих данных).

Языковые средства СУБД являются важнейшим компонентом СБД, т.к. обеспечивают интерфейс пользователей разных категорий с СБД:

Слайд Языковые средства современных СУБД

Языковые средства современных СУБД относятся к 4-му поколению.

К 1-му поколению относят **машинные языки**

ко 2-му – **языки ассемблера**

к 3-му – **алгоритмические языки типа PL и Cobol**, которые назывались языками высокого уровня, но уровень которых гораздо ниже, чем у языков 4-го поколения.

К 5-му поколению относят языки систем искусственного интеллекта (**Prolog**).

Для выражения обобщенного взгляда на данные применяют **язык описания данных (ЯОД)**, включаемый в состав СУБД.

ЯОД позволяет определять схемы БД, характеристики хранимых данных, параметры хранения их в памяти и может включать средства поддержки целостности, ограничения доступа, секретности. Одна БД на ЯОД разных СУБД может описываться по-разному.

Язык манипулирования данными (ЯМД) включает в себя средства запросов к БД и поддержания БД (добавление, удаление, обновление данных, создание и уничтожение БД, обеспечение запросов к справочнику БД).

ЯМД разделяются на:

• процедурные;

• непроцедурные (декларативные).

При пользовании процедурными языками надо указать, какие действия и над какими объектами необходимо выполнить, чтобы получить результат. В непроцедурных языках указывается, что надо получить в ответе, а не как этого достичь.

Процедурные языки различаются по основным информационным единицам, которыми они манипулируют:

• языки, ориентированные на **позаписную обработку данных;**

• языки, ориентированные на **операции над множеством записей.**

Примеры непроцедурных языков, основанных на реляционном исчислении

- язык запросов SQL;
- табличный язык QBE.

По форме представления различают следующие языковые средства:

• аналитические;

• табличные;

• графические.

В рамках одной СУБД могут использоваться языки разных типов. Во многих СУБД для манипулирования данными могут использоваться:

• табличный язык запросов типа QBE;

• аналитический язык запросов SQL;

• процедурный язык программирования.

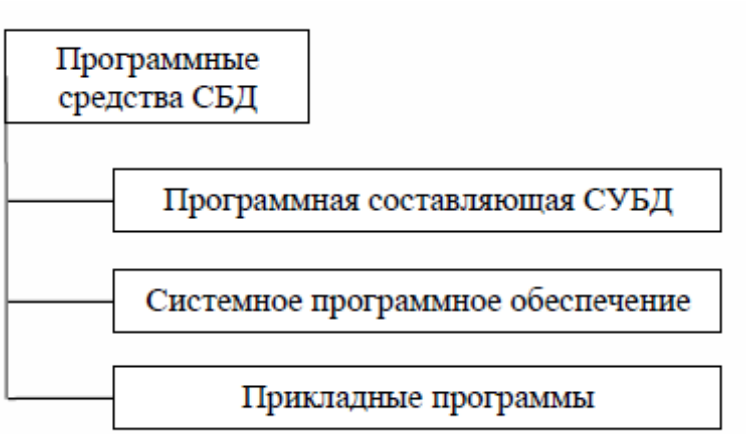
Также системы включают генераторы экранных форм, отчетов и приложений, а также язык разветвленной иерархической системы «меню» для выбора действий пользователем.

Наиболее распространенный язык - SQL (Structured Query Language). Предоставляет средства обработки запросов и функции по созданию, обновлению и управлению доступом.

SQL соединяет ЯОД и ЯМД и не является полноценным языком программирования. Для доступа к БД из прикладных программ SQL-выражения встраиваются в конструкции базового языка.

Программные средства СБД - сложный комплекс, обеспечивающий взаимодействие всех частей системы:

Слайд Программные средства СУБД



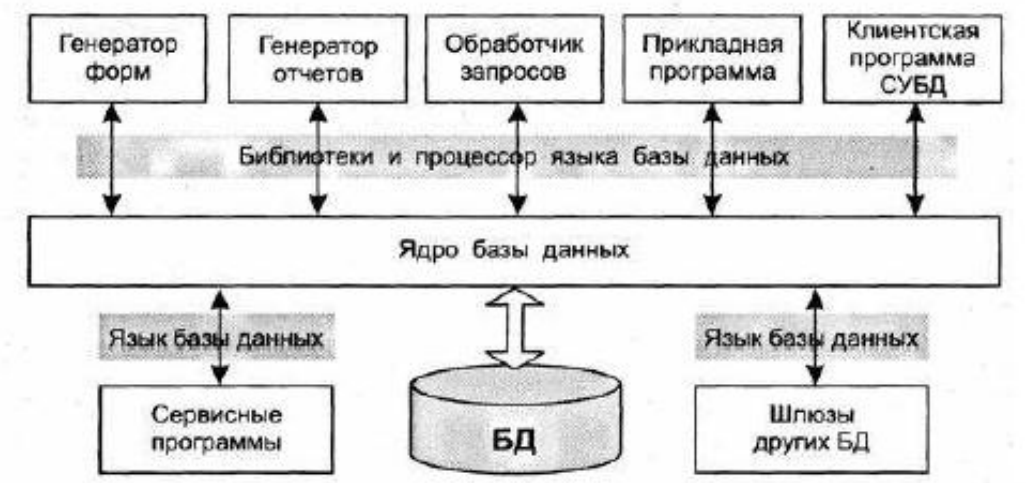
Программная составляющая СУБД осуществляет обработку данных и взаимодействие с ОС и прикладными программами.

Необходима координация между ОС и СУБД. Многопользовательские приложения, обработка распределенных запросов требуют эффективного использования ресурсов, которыми управляет ОС. Управление доступом и обеспечение защиты также интегрируются с соответствующими средствами ОС.

Взаимосвязь компонентов программных средств:

Слайд

Программная составляющая СУБД



Программная составляющая СУБД включает компоненты:

• **ядро**, обеспечивающее управление данными во внешней и оперативной памяти и протоколирование изменений;

• **процессор языка БД** обеспечивает обработку и оптимизацию запросов на выборку и изменение данных;

• **подсистема (библиотека) поддержки программных вызовов** обслуживает прикладные программы управления данными, взаимодействующие с СУБД через средства пользовательского интерфейса;

• **сервисные программы** (**системные и внешние утилиты**), обеспечивающие настройку СУБД, восстановление после сбоев и др.

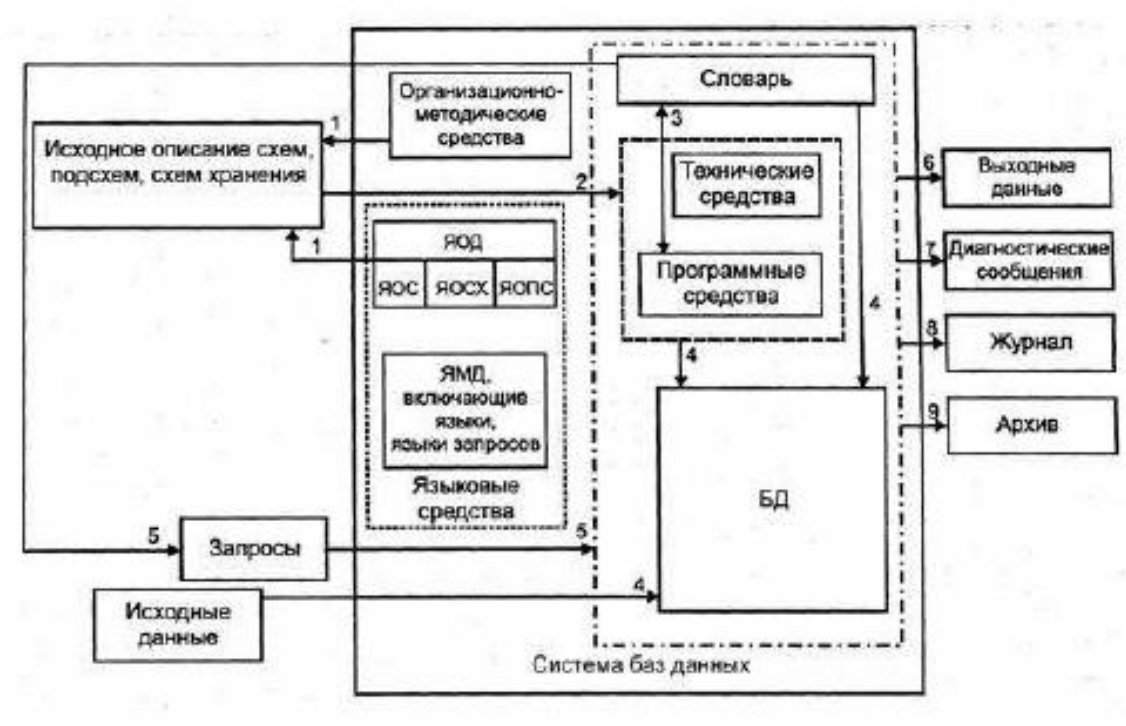
К **техническим средствам СБД** относятся: ЭВМ, периферийные средства ввода информации в БД, средства хранения данных и отображения выводимой информации Для сетевых СБД необходимы коммуникационные средства.

Взаимодействие компонентов системы баз данных

На слайде представлена схема взаимодействия компонентов СБД в процессе создания и эксплуатации:

Слайд

Схема взаимодействия компонентов СБД



(1). Создание БД начинается с проектирования и описания на ЯОД. Могут использоваться методики ручного проектирования и CASE-средства, автоматически генерирующие описания БД.

(2, 3). Описания вводятся в СБД и запоминаются в соответствии с требованиями конкретной СУБД.

(4). После сохранения описания БД, в БД вводятся данные. СУБД использует метаинформацию, зафиксированную в словаре данных.

Заполненная БД используется для извлечения необходимой пользователям информации (5). При формулировании запросов используется информация, содержащаяся в схемах и подсхемах.

В результате выполнения запроса пользователю выдаются выходные данные (6). Кроме данных выдается диагностическая информация (7).

Для обеспечения надежности необходимо осуществлять журнализацию выполняемых действий (8) и регулярно архивировать данные (9).

Классификация БД

СБД - сложные системы, их классификация может быть произведена для всей СБД и для каждого компонента отдельно.

Центральный компонент СБД – БД и большинство классификационных признаков относится к БД.

По форме представления информации различают визуальные, системы аудио- и мультимедиа. Классификация показывает, в каком виде информация хранится в БД и выдается пользователям.

По характеру организации данных: БД неструктурированные (БД, организованные в виде семантических сетей), частично структурированные (текст или

гипертекстовые системы) и структурированные (требуют предварительного проектирования и описания структуры).

Структурированные БД по типу используемой модели делятся: на иерархические, сетевые, реляционные, смешанные и мультимодельные. Эта классификация распространяется и на СУБД.

В структурированных БД различают несколько уровней информационных единиц (ИЕ), входящих одна в другую. Большинство систем поддерживает:

- поле – наименьшая семантическая единица информации;
- совокупность полей (или более сложных ИЕ) образует запись;
- множество однотипных записей представляет файл базы данных.

Многие СУБД в явном виде поддерживают и уровень базы данных как совокупности взаимосвязанных файлов БД.

По типу хранимой информации БД делятся на фактографические, документальные и лексикографические.

В **фактографических БД** хранится информация фактического характера – **числовые или текстовые характеристики объектов**, **представленные в формализованном виде**. В ответ на запрос выдается информация об интересующем объекте.

В **документальных БД единицей хранения является документ** и пользователю выдается ссылка на документ или сам документ.

Документальные БД организуются без хранения и с хранением документа на машинных носителях. К первому типу относятся библиографические, реферативные и БД-указатели, отсылающие к источнику информации. Системы, хранящие полный текст документа, называются полнотекстовыми. Их разновидностью являются БД форм документов, в которых документ ищется для использования его в качестве шаблона.

К **лексикографическим БД** относятся различные словари (**классификаторы, многоязычные словари, словари основ слов** и т. п.).

По характеру организации хранения данных и обращения к ним различают локальные (персональные), общие (интегрированные, централизованные) и распределенные БД.

Персональная БД предназначена для локального использования одним пользователем. Локальные БД могут создаваться каждым пользователем самостоятельно, а могут извлекаться из общей БД.

Интегрированные и распределенные БД предполагают возможность одновременного обращения к информации нескольких пользователей (многопользовательский режим доступа). Части распределенных БД физически расположены на разных ЭВМ, но логически представляют собой единое целое.

Распределяться по узлам сети могут и другие компоненты СБД. Сама БД при этом может быть нераспределенной. Поэтому различают:

• **распределенные БД;**

• **распределенные СБД** (в которых распределен хотя бы один компонент).

В некоторых источниках упоминают экстенсиональные и интенсиональные БД. Первые строятся с помощью явного хранения данных в БД, вторые – с помощью правил, определяющих их содержание.

БД классифицируются по объему. (***Очень большие БД. Для больших БД*** по-иному ставятся вопросы обеспечения эффективности хранения информации и обеспечения ее обработки).

Классификация СУБД

СУБД классифицируются:

По языкам общения: открытые, замкнутые и смешанные.

В открытых системах для обращения к БД используются универсальные языки. Замкнутые системы имеют собственные языки общения с пользователями СБД.

По выполняемым функциям: информационные и операционные.

Информационные позволяют организовать хранение информации и доступ к ней. Для более сложной обработки необходимы специальные программы. Операционные выполняют сложную обработку и могут менять алгоритмы обработки.

По сфере возможного применения: универсальные и специализированные (проблемно-ориентированные СУБД).

***По ориентации на преобладающую категорию пользователей*:** СУБД для разработчиков и для конечных пользователей.

Первые должны иметь качественные компиляторы и позволять создавать отчуждаемые программные продукты, обладать развитыми средствами отладки, включать средства документирования. Вторые удобный интерфейс, высокий уровень языковых средств, интеллектуальные модули подсказок, защиту от ошибок и т. п.

По мощности: настольные (Dbase, FoxBase/FoxPro, Clipper, Paradox, Access, Approach и др.) и корпоративные (серверные) (Oracle, Microsoft SQL Server, Sybase, Informix и др.).

Для первых характерны невысокие требования к техническим средствам, ориентация на конечного пользователя и сравнительно низкая стоимость; небольшой объем обрабатываемых данных, малое количество пользователей, относительно упрощенная архитектура, функционируют в режиме файл-сервер, поддерживают не все функции СУБД. Например, в таких СУБД может не вестись журнал транзакций, отсутствовать возможность автоматического восстановления базы данных после сбоев и т. п.

• ***по модели данных*** (иерархические, сетевые, реляционные, объектно-ориентированные, объектно-реляционные);

• ***по степени распределенности*** (локальные, распределенные);

• ***по способу доступа к БД***.

В локальных СУБД все части локальной СУБД размещаются на одном компьютере. В распределённых СУБД части СУБД размещаются на двух и более компьютерах.

По способу доступа к БД СУБД делятся на:

• файл-серверные;

• клиент-серверные;

• встраиваемые.

В файл-серверных СУБД файлы данных располагаются централизованно на файл-сервере. СУБД располагается на каждом клиентском компьютере (рабочей станции). Доступ СУБД к данным осуществляется через локальную сеть. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок.

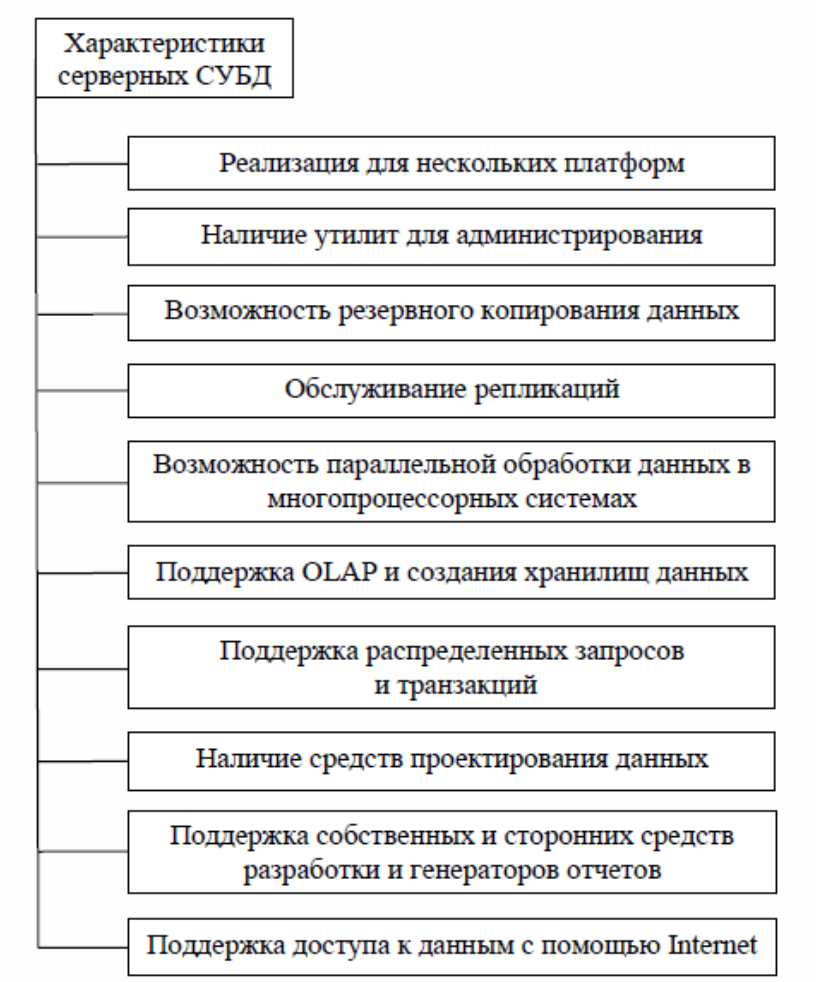
Примеры файл-серверных СУБД: Microsoft Access, Paradox, dBase, FoxPro, Visual FoxPro.

Клиент-серверная СУБД располагается на сервере вместе с БД и осуществляет доступ к БД непосредственно, в монопольном режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно.

Примеры Клиент-серверных СУБД: Oracle, Firebird, Interbase, IBM DB2, Informix, MS SQL Server, Sybase Adaptive Server Enterprise, PostgreSQL, MySQL, Cache, ЛИНТЕР.

Слайд

Характеристики серверных СУБД



Встраиваемая СУБД – СУБД, которая может поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки. Встраиваемая СУБД представляет собой по сути библиотеку данных и предназначена для локального хранения данных своего приложения и не рассчитана на коллективное использование в сети. Примеры встраиваемых СУБД: OpenEdge, SQLite, BerkeleyDB, Firebird, Embedded, Microsoft SQL Server Compak.

Утилиты администрирования. Наличие утилит администрирования - один из решающих факторов при выборе СУБД. Почти все современные СУБД обычно поставляется с подобными утилитами.

Резервное копирование данных и **журналов транзакций** поддерживается всеми коммерческими серверными СУБД.

Обслуживание репликаций. Репликация представляет собой гарантированное копирование информации из одной базы в несколько других и используются для разделения нагрузки между серверами в сети, для перемещения поднаборов данных на вспомогательные серверы, для синхронизации данных на нескольких серверах и многих других целей.

Репликации поддерживаются всеми современными серверными СУБД.

Параллельная обработка данных в многопроцессорных системах. Серверы, поддерживающие параллельную обработку, разрешают нескольким процессорам обращаться к одной БД, что позволяет обеспечить высокую скорость обработки транзакций.

Подавляющее большинство производителей современных серверных СУБД поставляют на рынок версии, поддерживающие параллельную обработку данных.

Поддержка OLAP и создания хранилищ данных. OLAP (On-Line Analytical Processing) - технология построения многомерных хранилищ данных (Data Warehouses), как правило, агрегатных, т.е. являющихся результатом обработки набора данных, нередко состоящего из нескольких таблиц. Такие ХД широко используются в системах поддержки принятия решений.

Многомерные ХД могут быть реализованы в виде набора обычных реляционных таблиц и в виде нереляционной многомерной БД. В последнем случае такое хранилище обычно управляется отдельным сервером. Производители серверных СУБД поставляют такие серверы отдельно (Oracle, Informix), некоторые включают их в состав сервера реляционных БД (Microsoft SQL Server). С целью повышения конкурентоспособности подобные OLAP-системы строят многомерные хранилища на основе данных из других СУБД, как это сделано, например, в **Microsoft SQL Server OLAP Extensions** и в **Sybase Adaptive Server IQ**.

Распределенные запросы и транзакции. Распределенные транзакции и запросы стали особенно актуальны, когда наличие нескольких серверов БД в одной организации стало обычным явлением. Возможности выполнения распределенного запроса или распределенной транзакции поддерживаются сейчас почти всеми серверными СУБД, по крайней мере, в том случае, когда все вовлеченные в транзакцию серверы от одного производителя.

Распределенные транзакции с участием сторонних серверов, как правило, реализуются с помощью ***мониторов транзакций*** или иных подобных сервисов, например, **Microsoft Distributed Transaction Coordinator**.

Средства проектирования данных. Производители серверных СУБД производят также средства анализа бизнес-процессов и проектирования данных - ***универсальные*** (как в случае Sybase DataArchitect) и ***ориентированные на конкретную*** СУБД (как в случае Oracle Designer).

Многие производители СУБД не имеют в своем арсенале собственных средств проектирования данных, ориентируясь на универсальные CASE-средства. Нередко производители СУБД встраивают в административные утилиты несложные средства проектирования данных, позволяющие визуально редактировать схемы данных, например, в Microsoft SQL Server.

Поддержка собственных и сторонних средств разработки и генераторов отчетов. Производители серверных СУБД выпускают также средства разработки и генераторы отчетов.

Иногда данные средства разработки используют тот же язык программирования, что применяется при написании ***триггеров*** и ***хранимых процедур*** (тогда, клиентское приложение должно включать ***интерпретатор этого языка***), что позволяет отлаживать хранимые процедуры, помещая их в клиентское приложение. Типичный пример подобного подхода реализован в Oracle Developer. Чаще средства разработки производителей серверных СУБД используют языки программирования, отличные от языков создания серверного кода.

Практически все производители серверных СУБД делают все возможное для того, чтобы клиентские приложения для их СУБД можно было создавать с помощью других средств разработки. Они предоставляют разработчикам ***описания API клиентской части***, **ODBC-драйверы**, ***OLE DB-провайдеры***, а нередко и объектные модели, позволяя

использовать COM-объекты клиентской части в приложениях (как, например, это сделано в клиентских частях Oracle, Microsoft SQL Server, Informix).

Поддержка доступа к данным с помощью Internet. Без поддержки публикации данных в Internet или получения данных от удаленных Internet-клиентов сегодня не обходится практически ни одна коммерческая СУБД. Производители серверных СУБД поддерживают Web-технологии. Эта поддержка осуществляется с помощью ***Web-серверов собственного производства*** или посредством ***создания расширений для существующих Web-серверов***, или путем включения в комплект поставки ***утилит,*** генерирующих Web-страницы согласно определенному расписанию.

Пользователи систем баз данных

• ***Конечные пользователи***

В зависимости от особенностей создаваемого банка данных круг его конечных пользователей может существенно различаться. Это могут быть случайные пользователи, обращающиеся к базе данных время от времени, а могут быть и регулярные пользователи.

• ***Администраторы СБД***

В зависимости от сложности и объема банка данных, от особенностей используемой СУБД служба администрации банка данных может различаться как по составу и квалификации специалистов, так и по количеству работающих в этой службе.

Контрольные вопросы для самоподготовки

1. Что такое система баз данных?
2. Что такое система управления базами данных?
3. Какие выделяют основные компоненты СБД?
4. Что входит в состав языковых средств СБД?
5. Какие требования предъявляются к современным СУБД?

Лекция 2

Тема: Модели данных. Реляционные базы данных

План

1. Классификация моделей данных. Даталогические модели (иерархическая, сетевая, реляционная)
2. Основные понятия реляционной модели данных (отношение, атрибут, кортеж, тип данных, домен, первичный ключ, внешний ключ, типы связей, целостность данных)
3. Необходимость нормализации схемы отношений. Нормальные формы. Достоинства и недостатки нормализации
4. Реляционная алгебра. Операции реляционной алгебры. Реляционное исчисление

1. Классификация моделей данных. Даталогические модели (иерархическая, сетевая, реляционная) 2. Основные понятия реляционной модели данных (отношение, атрибут, кортеж, тип данных, домен, первичный ключ, внешний ключ, типы связей, целостность данных)

Этапы проектирования баз данных

БД отражает информацию об определенной предметной области (ПрО) – части реального мира, данные о которой надо отразить БД.

Предметную область принято рассматривать в виде следующих представлений:

• представление ПрО в том виде, как она реально существует;

• как ПрО воспринимает человек (проектировщик БД);

• как ПрО может быть описана с помощью символов.

При проектировании БД ***организацию данных*** принято рассматривать на следующих уровнях:

• **информационно-логическом (инфологическом, концептуальном);**

• **даталогическо**м;

• **физическом.**

Этим уровням соответствуют инфологическая, даталогическая и физическая модели ПрО.

Процесс создания БД начинается с определения концептуальных требований будущих пользователей, которые интегрируются в обобщенном представлении, называемом концептуальной моделью.

Концептуальной, или инфологической моделью (информационно- логическая модель - ИЛМ) называется ориентированная на человека и не зависимая от типа СУБД модель предметной области, определяющая совокупности информационных объектов, их атрибутов и отношений между объектами, динамику изменений предметной области, а также характер информационных потребностей пользователей.

Концептуальная модель отражает специфику ПрО, а не структуру БД.

Версия концептуальной модели, которая может быть реализована в конкретной СУБД, называется **даталогической моделью**. Модель отражает логические взаимосвязи между элементами данных безотносительно их содержания и физической организации. Модель строится в терминах информационных единиц, допустимых для СУБД.

Описание логической структуры БД на языке СУБД называется **схемой.**

Т.е. под **даталогической** понимается модель, отражающая логические взаимосвязи между элементами данных безотносительно их содержания и физической организации. Даталогическая модель разрабатывается с учётом конкретной реализации СУБД, с учётом специфики конкретной ПрО на основе ее инфологической модели.

Внутренняя (физическая) модель данных определяет способ размещения данных непосредственно на машинном носителе, учитывает распределение данных, методы доступа и способы индексирования, т.е. определяют способы размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне.

Резюме. В современных прикладных программных средствах этот уровень организации обеспечивается автоматически без вмешательства пользователя. Пользователь оперирует представлениями СУБД. Т.о., основная задача проектирования - создание инфологической и даталогической моделей.

Модель, описывающая логическую структуру БД с точки зрения конкретного пользователя (приложения), называется **внешней**, а ее описание называется **подсхемой.** Внешняя модель представляет отображение концептуальных требований конкретного пользователя.

Пользователь имеет доступ только к данным подсхемы, что является одним из способов защиты информации. Использование подсхем облегчает работу пользователя, т. к. он должен знать структуру только своей части БД.

Взаимосвязь этапов проектирования БД показана на слайде.

Слайд

Взаимосвязь этапов проектирования

Первым шагом является построение инфологической модели.

Предварительная модель строится на предпроектной стадии и уточняется на более поздних стадиях проектирования. Затем на ее основе строится даталогическая модель. Физическая и внешняя модели после этого могут строиться в любой последовательности, в том числе и параллельно.

Инфологическое проектирование баз данных

Для описания ПрО может использоваться естественный язык, но его применение имеет много недостатков - громоздкость описания и неоднозначность его трактовки. Поэтому используют искусственные формализованные языковые средства.

Под инфологической моделью понимают описание ПрО, выполненное с использованием специальных языковых средств.

Основным требованием к ИЛМ является требование адекватного отображения предметной облаПрОти. ИЛМ должна быть непротиворечивой.

ИЛМ должна обладать свойством легкой расширяемости: ввод новых данных без изменения ранее определенных. Аналогично и для удаления данных. ИЛМ должна обеспечивать возможность композиции и декомпозиции модели.

ИМЛ должна легко восприниматься разными пользователей и однозначно всеми специалистами, которые в дальнейшем участвуют в процессе проектирования БД и ПО. ИЛМ - ядро системы проектирования.

Способы описания инфологической модели:

– модель данных Хаммера и Мак-Леона;

– модель данных Шипмана;

– подход, основанный на применении диаграмм «сущность-связь» (ER – Entity Relationship), предложенный Питером Ченом в 1976 году.

Модель «сущность-связь» стала фактическим стандартом при инфологическом моделировании БД. В основе модели лежит деление реального мира на отдельные различимые **сущности**, находящиеся в определенных **связях** друг с другом. Категории – сущность и связь – полагаются первичными понятиями.

Основные понятия ER-модели

На использовании разных вариантов ER-модели основано большинство современных подходов к проектированию БД (главным образом, реляционных). Моделирование ПрО базируется на использовании графических диаграмм. Простота и наглядность представления концептуальных схем БД в ER-модели привели к ее широкому распространению в **CASE-системах**, **поддерживающих автоматизированное проектирование реляционных БД.** Одним из наиболее распространенных CASE-средств является ERwin. Также разработаны методы автоматического преобразования проекта БД из ER-модели в даталогическую модель, соответствующую конкретной СУБД.

Сейчас не существует единой общепринятой системы обозначений для ER-модели и разные CASE-системы используют разные графические нотации. Наиболее популярны:

– нотация Питера Чена;

– IDEF1X;

– Crow's Foot;

– Bachman notation;

– EXPRESS;

– Martin notation.

Основные понятия ER-модели: ***сущность, связь* и *атрибут*.**

Нотация – система условных обозначений, принятая в какой-либо области знаний или деятельности. Включает множество символов, используемых для представления понятий и их взаимоотношений, составляющее алфавит нотации, а также правила их применения.

Описание основных положений инфологического проектирования будет вестись с точки зрения нотации IDEF1X.

Сущность – реальный или представляемый объект, информация о котором должна сохраняться и быть доступной.

В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности – это имя типа, а не некоторого конкретного экземпляра этого типа.

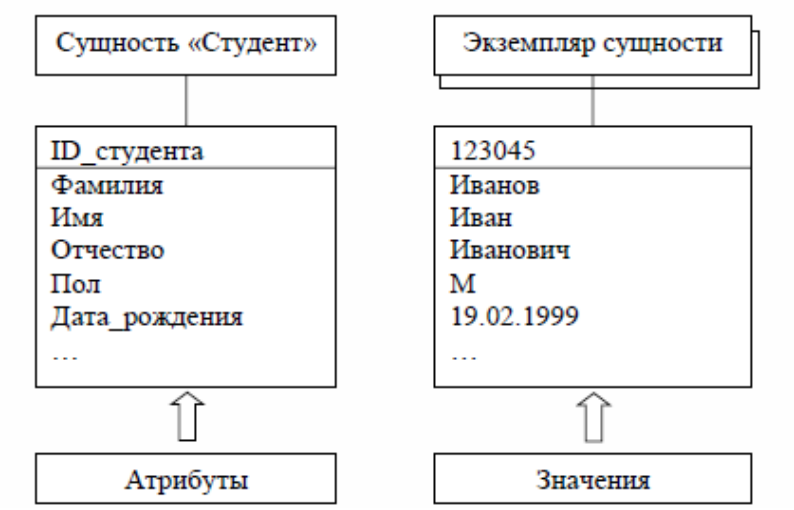
Примеры сущностей: типы объектов «Студент», «Предмет», «Успеваемость».

Экземпляр сущности – это конкретный представитель данной сущности. Например, представителем сущности «Студент» может быть «Студент Иванов». Экземпляры сущностей должны быть различимы, т.е. сущности должны иметь некоторые свойства, уникальные для каждого экземпляра этой сущности.

Атрибут сущности – это именованная характеристика, являющаяся некоторым свойством сущности. Наименование атрибута выражается существительным в единственном числе (возможно, с прилагательными).

Слайд

Сущность и экземпляр сущности



Виды атрибутов:

• ***простой*** – состоит из одного элемента данных;

• ***составной*** – состоит из нескольких элементов данных;

• ***однозначный*** – содержит одно значение для одной сущности;

• ***многозначный*** – содержит несколько значений для одной сущности;

• ***необязательны***й – может иметь пустое (неопределенное) значение;

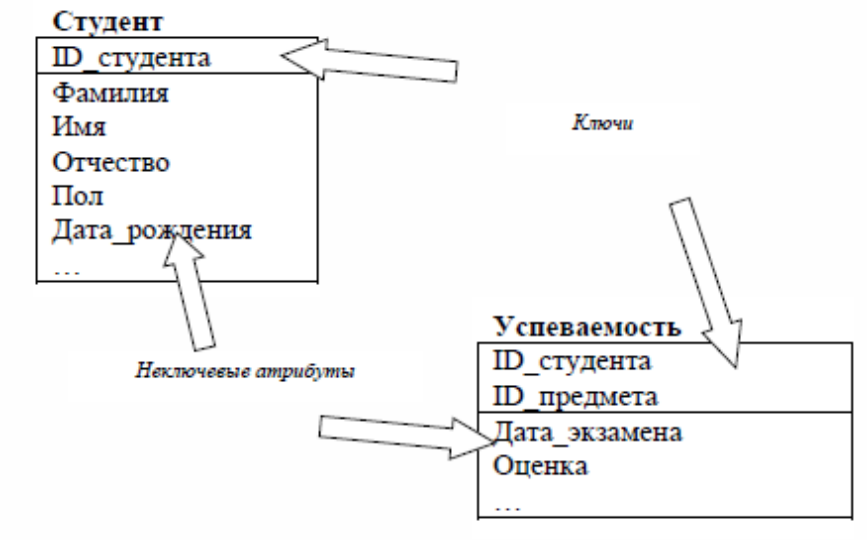
• производный – значение, производное от значения другого атрибута.

Ключ – минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать экземпляр сущности по оставшимся атрибутам.

Ключевые атрибуты в нотациях изображаются на диаграмме подчеркиванием.

Слайд

Ключевые и неключевые атрибуты сущности



Каждая сущность должна иметь следующие свойства:

• иметь уникальное имя;

• обладать одним или несколькими атрибутами, которые принадлежат сущности или наследуются через связь;

• обладать одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности.

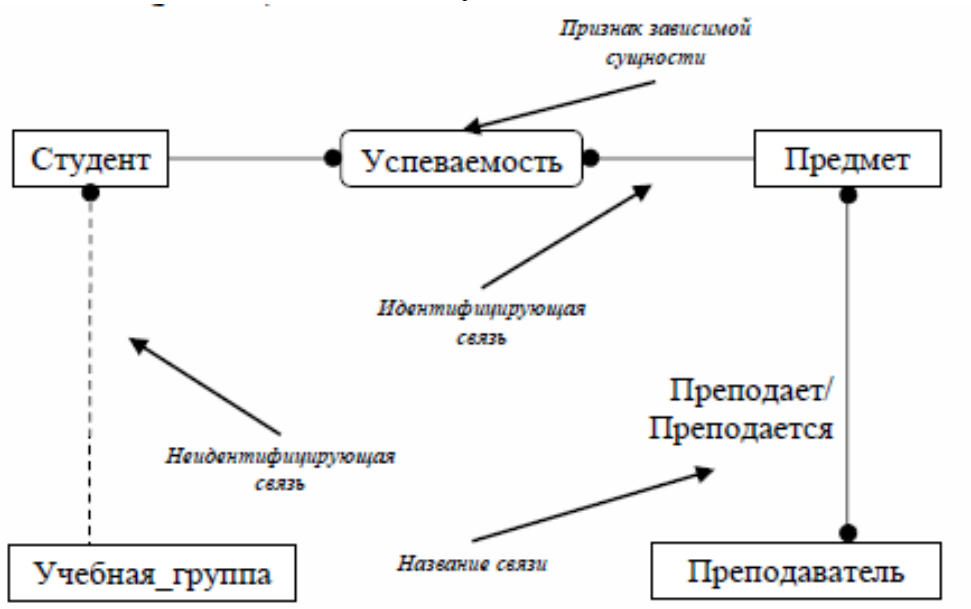
Каждая сущность может обладать любым количеством связей с другими сущностями.

Связь – поименованная ассоциация между сущностями, значимая для рассматриваемой ПрО.

Связь изображается линией, соединяющей две сущности. Каждая связь имеет два конца и одно или два наименования. Каждое из наименований относится к своему концу связи. Иногда наименования не пишутся:

Слайд

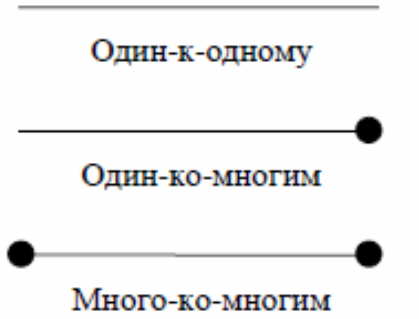
Обозначение сущностей и связей



Каждая связь может принадлежать к следующим типам связи: ***один-к-одному, один-ко-многим, много-ко-многим***:

Слайд

Типы связи в нотации IDEF1X



Связь типа **один-к-одному** означает, что один экземпляр первой сущности (левой) связан с одним экземпляром второй сущности (правой). Связь один-к-одному чаще всего свидетельствует о том, что на самом деле имеется всего одна сущность, неправильно разделенная на две.

Связь типа **один-ко-многим** означает, что один экземпляр первой сущности (левой) связан с несколькими экземплярами второй сущности (правой). Это наиболее часто используемый тип связи.

Левая сущность (со стороны «один») называется ***родительской***, правая (со стороны «много») – ***дочерней***.

Связь типа **много-ко-многим** означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности. Тип связи много-ко-многим – **временный** тип связи, допустимый на ранних этапах разработки модели. В дальнейшем он должен быть заменен двумя связями типа один-ко-многим путем создания промежуточной сущности.

По признаку идентификации сущностей связи типа один-ко-многим делятся на два вида:

- ***Идентифицирующая связь*** – такая связь, при которой экземпляр дочерней сущности идентифицируется через свою ассоциацию с родительской сущностью. Экземпляр дочерней сущности не существует без экземпляра родительской сущности. Атрибуты ключа родительской сущности становятся атрибутами ключа дочерней сущности. Дочерняя сущность является зависимой сущностью.

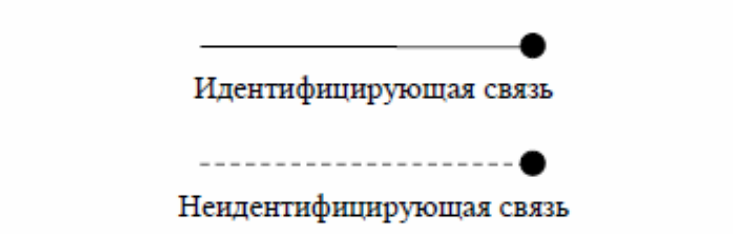
Пример. Связь сущностей «Студент» и «Успеваемость». Экземпляр зависимой сущности «Успеваемость» определяется только через отношение к родительской сущности «Студент», т. е. информация о результатах сдачи экзамена не может быть внесена и не имеет смысла без информации о студенте.

– **Неидентифицирующая связь** – такая связь, при которой экземпляр дочерней сущности не идентифицируется через свою ассоциацию с родительской сущностью. Экземпляр дочерней сущности может существовать без экземпляра родительской сущности. Атрибуты ключа родительской сущности становятся неключевыми атрибутами дочерней. Например, связь сущностей «Студент» и «Учебная группа».

Обозначения различных видов связи и сущностей:

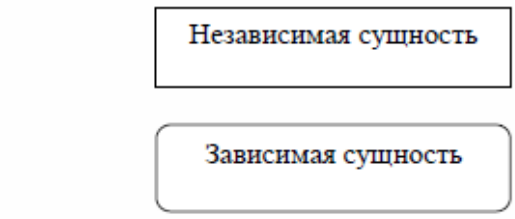
Слайд

Обозначение связей в нотации IDEF1X



Слайд

Обозначение сущностей в нотации IDEF1X



Связь между сущностями обеспечивается за счет миграции атрибутов родительской сущности в дочернюю.

Миграция – перенос атрибутов одной сущности в другую для установления связи между ними. Мигрировавший атрибут называется внешним ключом и помечается на ER-диаграмме символами (**FK**) (Foreign Key).

Мигрировавший атрибут или группа атрибутов могут быть помещены в состав **первичного ключа** сущности или в состав **неключевых атрибутов** в зависимости от типа связи между сущностями.

Действуют следующие правила миграции:

1) если сущности связаны идентифицирующей связью, то все ключевые атрибуты родительской сущности мигрируют в состав первичного ключа дочерней сущности;

2) если сущности связаны неидентифицирующей связью, то все ключевые атрибуты родительской сущности мигрируют в состав неключевых атрибутов дочерней сущности.

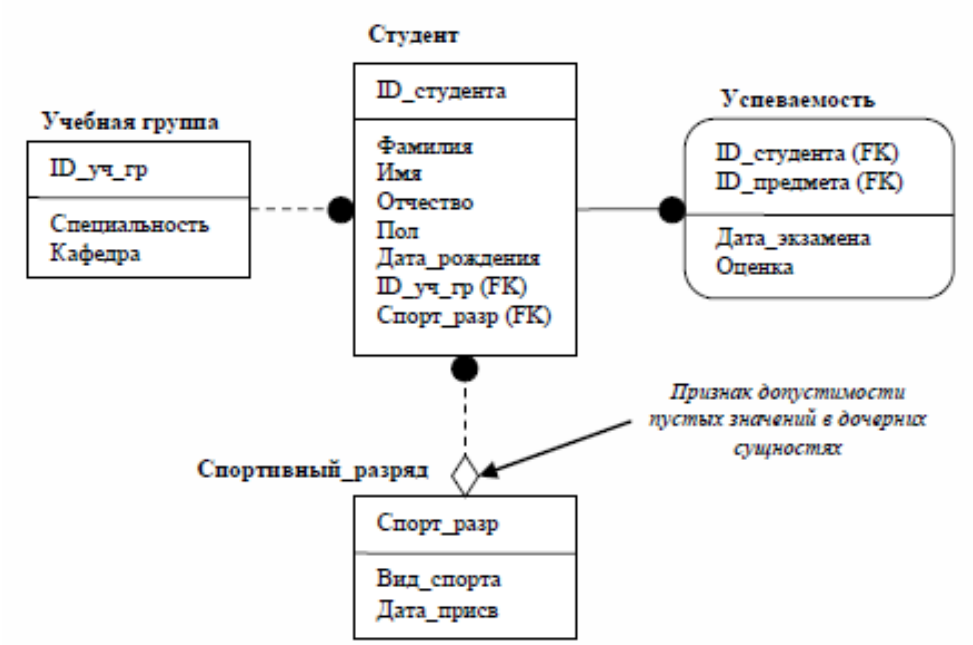
Для второго случая возможны два варианта отношений:

• допускаются пустые значения внешних ключей в дочерней сущности (в некоторых нотациях обозначается специальным символом — знак ромба на неидентифицирующей связи со стороны родительской сущности);

• пустые значения внешних ключей в дочерней сущности не допускаются (отсутствие знака ромба со стороны родительской сущности):

Слайд

Пример миграции атрибутов



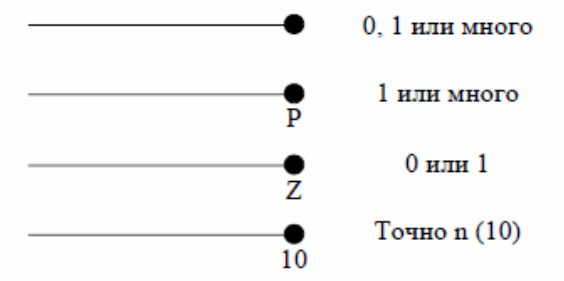
Каждая связь обладает мощностью.

Мощность связи (кардинальность) – характеристика связи между сущностями, предназначенная для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней.

Существует четыре различных типа мощности:

Слайд

Обозначение мощности связи (IDEF1X)



Одному экземпляру родительской сущности соответствуют 0, 1 или много экземпляров дочерней сущности. Не помечается дополнительным значком на диаграмме.

Одному экземпляру родительской сущности соответствуют 1 или много экземпляров дочерней сущности (исключено нулевое значение). На диаграмме помечается значком P.

Одному экземпляру родительской сущности соответствуют 0 или 1 экземпляр дочерней сущности (исключены множественные значения). Помечается значком Z.

Одному экземпляру родительской сущности соответствует заранее заданное число экземпляров дочерней сущности. Помечается цифрой.

Особая разновидность связи - **категориальная связь**. Используется для описания структур, в которых сущность является типом (категорией) другой сущности.

При этом родительская сущность (родовой предок) содержит общие свойства, присущие дочерним (категориальным) сущностям.

Категориальную связь, называемую **иерархией наследования**, создают, когда несколько сущностей имеют общие по смыслу атрибуты или когда сущности имеют общие по смыслу связи.

Для каждой категории можно указать **дискриминатор** – атрибут родового предка, который показывает, как отличить одну категориальную сущность от другой.

Пример. Известно, что при оформлении операций, связанных с перемещением ТМЦ в организации используются два вида накладных: ***приходная*** и ***расходная***. Сущность «Накладная» является родительской, так как объединяет общие для обеих сущностей атрибуты, а сущности «Приходная» и «Расходная» содержат информацию об особенностях накладных каждого вида. Дискриминатором в данном случае будет вид накладной:

Слайд

Категориальные связи делятся на два типа:

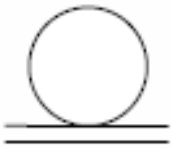
– полные;

– неполные.

Если экземпляру родового предка соответствует экземпляр в каком-либо потомке, то связь является **полной**, на ER-диаграмме изображается с помощью **дискриминатора полной связи:**

Слайд

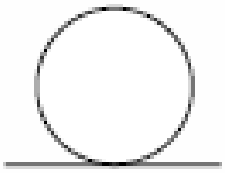
Дискриминатор полной связи



Если категория еще не выстроена полностью и в родовом предке есть экземпляры, для которых нет соответствующих экземпляров в потомках, категория является **неполной,** на ER-диаграмме изображается с помощью **дискриминатора неполной связи**:

Слайд

Дискриминатор неполной связи



Даталогические модели данных

Модель данных есть формальная теория представления и обработки данных в СУБД, которая включает три аспекта:

– **аспект структуры**: методы описания типов и логических структур данных в БД;

– **аспект манипуляции**: методы манипулирования данными;

– **аспект целостности**: методы описания и поддержки целостности БД.

В книге К. Дж. Дейта «Введение в системы баз данных» приводится следующее определение модели данных.

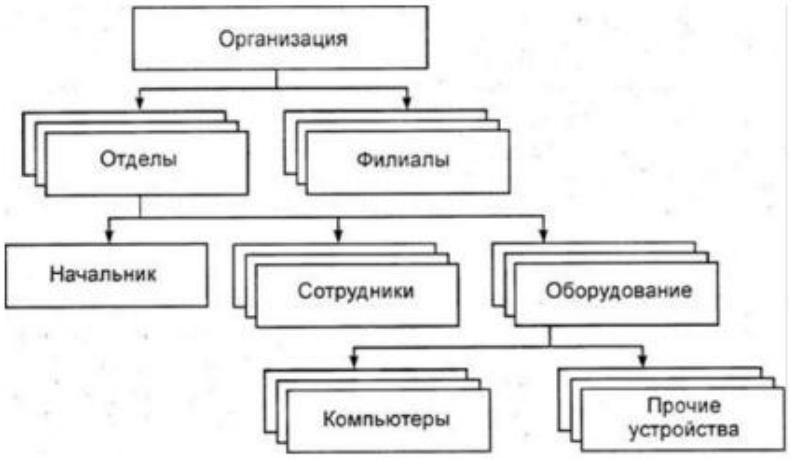
Модель данных – это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы – поведение данных.

В классической теории БД различают иерархическую, сетевую и реляционную модели данных.

Иерархическая модель данных – представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней. Такие объекты находятся в отношении предка к потомку.

Слайд

Иерархическая модель данных



К основным понятиям иерархической структуры относятся: уровень, узел, связь.

Узел (сегмент) – это совокупность атрибутов данных, описывающих некоторый объект. На схеме иерархического дерева узлы представляются вершинами графа. Каждый узел на более низком уровне связан только с одним узлом, находящимся на более высоком уровне.

Иерархическое дерево имеет только одну вершину (**корень дерева**), не подчиненную никакой другой вершине и находящуюся на самом верхнем (нулевом) уровне. Зависимые (подчиненные) узлы находятся на первом, втором, третьем и т.д. уровнях.

Количество деревьев в БД определяется числом корневых записей. К каждой записи БД существует только один (иерархический) путь от корневой записи. Поиск данных всегда начинается с корня. Затем производится спуск с одного уровня на другой пока не будет достигнут искомый уровень. Перемещения от одной записи к другой осуществляются с **помощью ссылок.**

Базовая логическая структура иерархической модели станет понятнее, если провести аналогию с неким производственным процессом. Рассмотрим упрощенную процедуру создания картотечного шкафа.

Картотечный шкаф состоит из нескольких узлов: каркас, набор выдвижных ящиков и направляющие для этих ящиков.

Каждый узел может состоять из множества более мелких сборок, например, у каждого ящика имеются ручка с защелкой, несколько роликов, с помощью которых он перемещается по направляющим, и разделительные пластинки.

Каждая сборка может состоять из нескольких деталей. К примеру, каждый ролик состоит из небольшого колесика, оси и крепежной скобы.

Процесс производства картотечного шкафа состоит в сборке всех частей, взаимосвязи между которыми постоянны во времени.

Когда бы ни был собран данный картотечный шкаф, сегодня или завтра, одни и те же детали собираются вместе тем же способом, образуя сборку, сборки собираются в узел, а все узлы собираются воедино, образуя тем самым картотечный шкаф.

Описание формирования деталей, сборок и узлов помогает представить весь этот логический процесс в виде перевернутого «дерева», которое называют **иерархической структурой**:

Слайд

Иерархическая (древовидная) структура

Древовидная структура, представленная на слайде не может быть непосредственно воспроизведена в компьютерном устройстве хранения. Дерево определяется цепочкой, отображающей маршрут от сегмента-предка к сегменту-потомку начиная слева.

Эта упорядоченная последовательность сегментов, отображающая иерархическую структуру, называется **иерархическим маршрутом**. Например, иерархический маршрут к сегменту с именем «Деталь D» может быть представлен следующим образом:

Изделие → Узел А → Сборка А →Деталь А → Деталь В → Узел В → Узел С → Сборка В → Деталь С → Деталь D

Маршрут обходит все сегменты, включая корневой, начиная с самого левого. Этот «левосторонний» список называется прямым порядком обхода или иерархической последовательностью.

Проектировщики БД должны убедиться, что сегменты и их элементы в этой цепочке, доступ к которым происходит чаще всего, расположены ближе к левому краю дерева с тем, чтобы обеспечить более эффективную обработку данных.

Например, если *Деталь D* используется чаще всего, то необходимо поместить *Узел С* в левую часть *Уровня 1*. Затем, внутри этой перемещенной «ветви» нужно переставить *Деталь D* на место, которое до этого занимала *Деталь С*. Эти изменения сделают иерархическую последовательность более короткой.

Изделие → Узел С → Сборка В → Деталь D

Достоинства иерархической модели: простота описания иерархических структур реального мира, быстрое выполнение запросов, соответствующих структуре данных.

Недостатки иерархической модели: модели часто содержат избыточные данные (это связано с невозможностью реализации связей «много-ко-многим») и не всегда удобно каждый раз начинать поиск нужных данных с корня.

Сетевая модель данных – логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных.

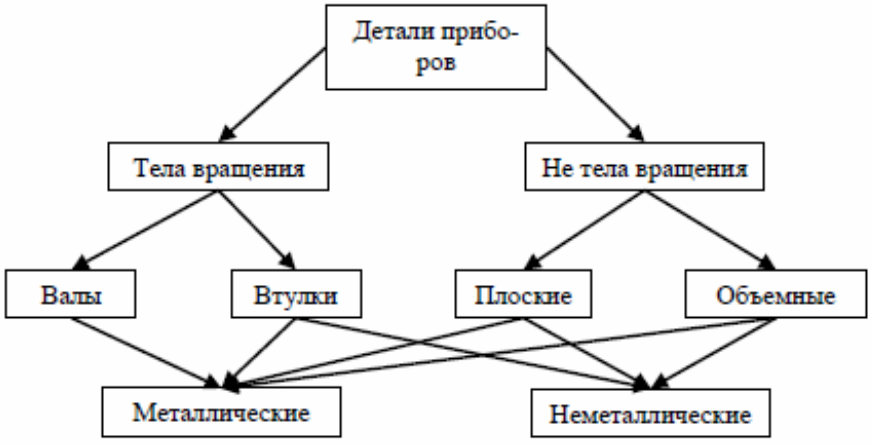
Разница между иерархической моделью данных и сетевой состоит в том, что в иерархических структурах запись* – потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков. ***Запись** – сегмент в иерархических моделях.

Сетевая модель БД сходна с иерархической моделью. Например, , пользователь воспринимает БД как совокупность записей, между которыми существует связь. В отличие от иерархической модели записи сетевой модели могут иметь более одного предка.

Графическое изображение структуры связей сегментов моделей представляет собой сеть:

Слайд

Сетевая структура



Сегменты данных в сетевых БД могут иметь множественные связи с сегментами старшего уровня. При этом направление и характер связи в сетевых БД не являются столь очевидными, как в случае иерархических БД. Поэтому имена и направление связей должны идентифицироваться при описании БД.

В сетевой модели сохранились основные преимущества иерархической модели. В ней были исправлены многие недостатки, присущие иерархической модели. Связь M:N проще реализуется в сетевой модели, чем в иерархической. **Недостатки сетевой модели данных**: высокая сложность и жесткость схемы БД, построенной на ее основе; сложность для понимания и выполнения обработки информации в БД обычным пользователем; ослаблен контроль целостности связей вследствие допустимости установления произвольных связей между записями.

Использование иерархической и сетевой моделей ускоряет доступ к информации, но требует значительных ресурсов памяти, т.к. каждый элемент данных содержит ссылки на другие элементы. Характерна сложность реализации СУБД.

В **реляционной модели** (РМД) информация представляется в виде двумерных таблиц, а операции сводятся к манипуляциям с таблицами. В 1980-х годах она получила широкое распространение, а реляционные СУБД стали промышленным стандартом.

Реляционная модель данных – логическая модель данных. Впервые была предложена британским учёным сотрудником компании IBM Эдгаром Франком Коддом (E. F. Codd) в 1970 году. В реляционной модели достигается гораздо более высокий уровень абстракции данных, чем в иерархической или сетевой. Термин «реляционный» означает, что теория основана на математическом понятии отношение (relation).

Таблицы обладают следующими свойствами:

1. Каждая ячейка таблицы представляет собой один элемент данных, совокупность значений в одном столбце одной строки недопустима.

2. Все столбцы в таблице однородные. Это означает, что элементы столбца имеют одинаковую природу. Столбцам присвоены имена.

3. В таблице нет двух одинаковых строк.

4. Порядок размещения строк и столбцов в таблице может быть произвольным. В операциях с такой таблицей ее строки и столбцы могут просматриваться в любом порядке безотносительно к их информационному содержанию и смыслу.

Причины доминирования РМД обусловлены тем, что имеются:

• развитая теория (реляционная алгебра);

• аппарат сведения других моделей данных к РМД;

• специальные средства ускоренного доступа к информации;

• стандартизированный высокоуровневый язык запросов к БД, позволяющий манипулировать данными без знания физической организации БД.

Свойства модели:

• модель является логической, т.е. отношения являются логическими (абстрактными), а не физическими (хранимыми) структурами;

• для реляционных БД верен информационный принцип: всё информационное наполнение БД представлено одним и только одним способом, а именно явным заданием значений атрибутов в кортежах отношений; в частности, нет никаких указателей (адресов), связывающих одно значение с другим;

• наличие реляционной алгебры позволяет реализовать декларативное программирование и декларативное описание ограничений целостности в дополнение к навигационному (процедурному) программированию и процедурной проверке условий.

Объектно-ориентированная модель начала разрабатываться в 1990-е годы с появлением объектно-ориентированных языков.

Такие БД хранят методы классов, что позволяет интегрировать данные и их обработку в приложениях.

Существует **объектно-реляционная модель данных.**

Контрольные вопросы для самоподготовки

1. Какие выделяют основные этапы проектирования БД?
2. Какая основная цель инфологического проектирования?
3. Дайте понятие сущности, атрибута, связи.
4. Каким образом происходит миграция атрибутов между сущностями?
5. Какие выделяют основные даталогические модели данных?

Основные понятия реляционной модели данных

Основы реляционной модели данных были впервые изложены в статье Э. Кодда в 1970 году. Наиболее распространенная трактовка реляционной модели данных принадлежит К. Дейту.

Согласно Дейту, реляционная модель состоит из трех частей:

- структурной;
- целостной;
- манипуляционной.

Структурная часть описывает, какие объекты рассматриваются реляционной моделью. Единственной структурой данных, используемой в реляционной модели, являются ***нормализованные n-арные отношения*.**

Целостная часть описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных БД. Это целостность сущностей и внешних ключей.

Манипуляционная часть описывает два эквивалентных способа манипулирования реляционными данными – реляционную алгебру и реляционное исчисление.

Структурная часть

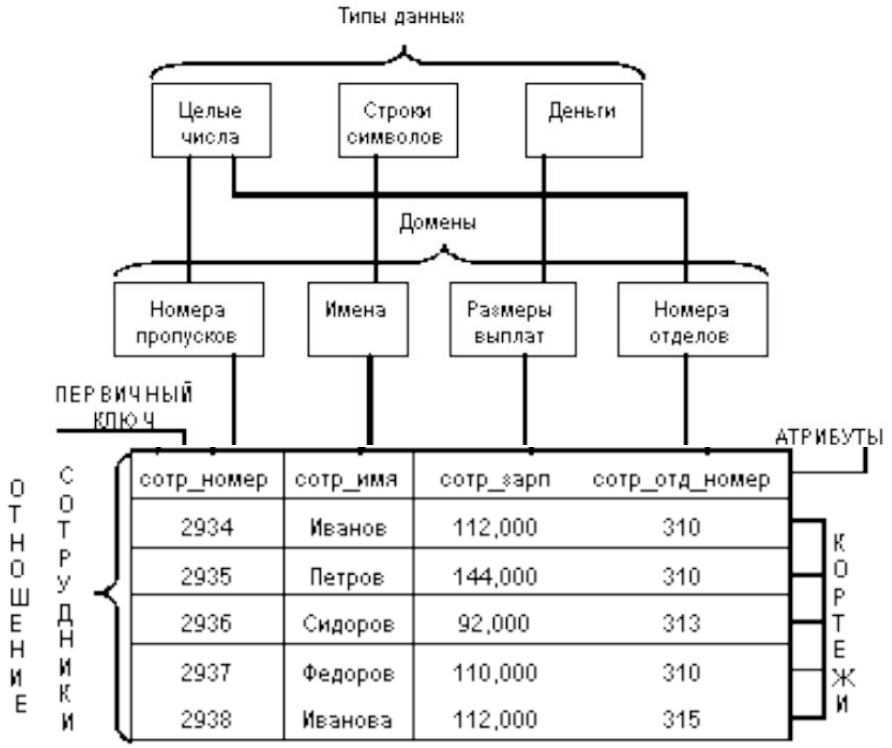
Основной структурой данных в модели является **отношен**ие, именно поэтому модель получила название **реляционной** (от английского relation — отношение).

В любой реляционной СУБД предполагается, что пользователь воспринимает БД как набор отношений (логическая структура данных). Эти отношения описывают соответствующие сущности или объекты. Любое отношение можно изобразить в виде таблицы, но нужно четко понимать, что отношения не являются таблицами. Это близкие, но не совпадающие понятия.

Основными понятиями реляционных БД являются:

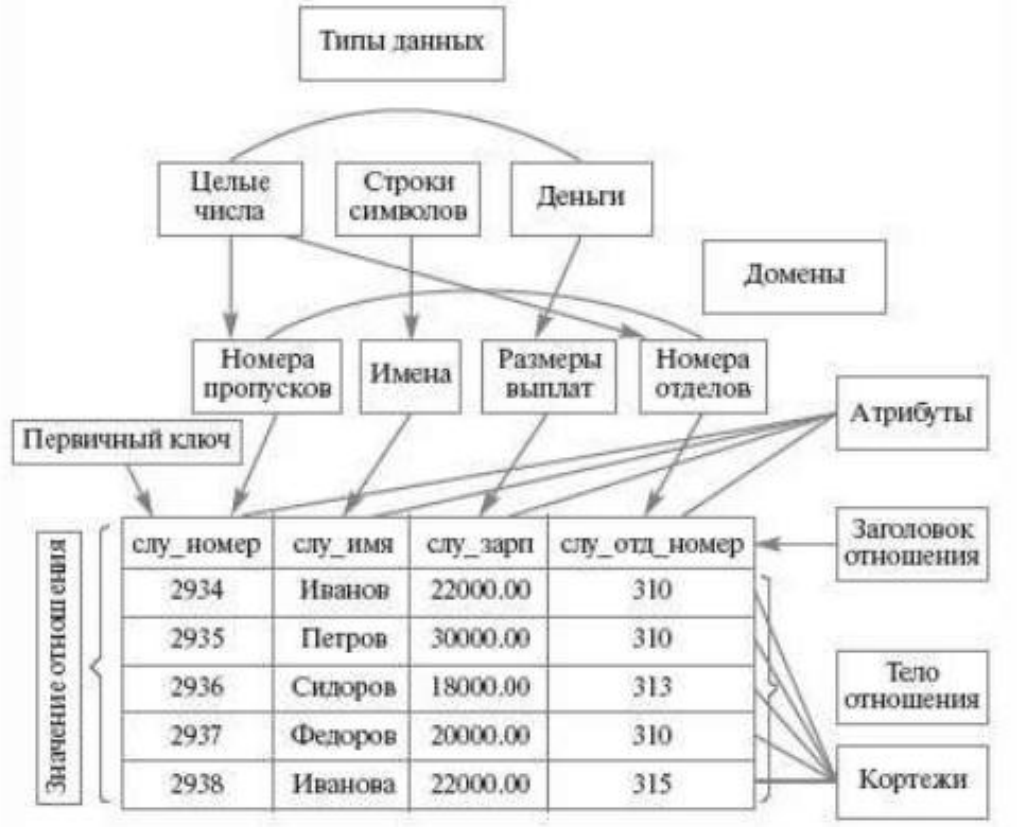
- тип данных
- домен
- атрибут
- кортеж
- ключ
- отношение.

Слайд



Слайд

Структурные компоненты РМД



Тип данных

Значения данных в реляционной БД являются типизированными, т. е. известен тип каждого хранимого значения. Понятие **типа данных** в РМД полностью соответствует понятию типа данных в языках программирования.

Типы данных делятся на группы:

• простые типы данных;

• структурированные типы данных;

• ссылочные типы данных.

Простые (атомарные) типы данных не обладают внутренней структурой. Данные такого типа называют **скалярами.**

К простым типам относятся следующие типы:

• логический;

• строковый;

• численный.

Различные СУБД могут расширять этот список:

• целый;

• вещественный;

• дата;

• время;

• денежный;

• перечислимый;

• интервальный;

• и т.д.…

Понятие атомарности данных является относительным. Строковый тип данных можно рассматривать как одномерный массив символов, а целый тип данных как набор битов. При переходе на такой низкий уровень теряется семантика (смысл) данных. Если строку, выражающую фамилию сотрудника, разложить в массив символов, то при этом теряется смысл такой строки как единого целого.

Структурированные типы данных предназначены для задания сложных структур данных. Они конструируются из составляющих элементов, называемых **компонентами,** которые могут обладать структурой.

***Структурированные типы данных*:**

• массивы,

• записи (структуры).

С математической точки зрения массив представляет собой функцию с конечной областью определения.

Запись (или структура) представляет собой кортеж из некоторого декартового произведения множеств. Объявляя новые типы записей на основе уже имеющихся типов, пользователь может конструировать сложные типы данных.

Резюме. Обычно в современных реляционных БД допускается хранение символьных, числовых данных, битовых строк, специализированных числовых данных (таких как «деньги»), а также специальных «темпоральных» данных (дата, время, временной интервал).

Структурированные типы данных имеют внутреннюю структуру. То есть при работе с массивами или записями можно манипулировать ими как с единым целым (создавать, удалять, копировать целые массивы или записи), так и поэлементно. Для

структурированных типов есть **специальные функции – конструкторы типов**, позволяющие создавать массивы или записи из элементов более простых типов.

С простыми типами данных, например, с числовыми, манипулируем ими как неделимыми целыми объектами.

Числовой тип данных сложен (является набором битов); нужно перейти на более низкий уровень абстракции.

Ссылочный тип данных (указатели) предназначен для обеспечения возможности указания на другие данные. Этот тип используется для обработки сложных изменяющихся структур.

Домен

С понятием тип данных тесно связано понятие домена, которое можно считать уточнением типа данных.

Домен – это семантическое понятие.

Домен – допустимое потенциальное множество значений типа данных. В самом общем виде **домен** определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат «истина», то элемент данных является элементом домена.

Свойства домена:

- Домен имеет уникальное имя (в пределах БД).
- Домен определен на некотором простом типе данных или на другом домене.
- Домен может иметь некоторое логическое условие, позволяющее описать подмножество данных, допустимых для данного домена.
- Домен несет определенную смысловую нагрузку.

Пример. Домен «Оценки на экзамене» может быть определен как целые числа, имеющие набор заданных значений (2, 3, 4, 5).

Отличие домена от понятия подмножества в том, что домен отражает семантику, определенную ПрО. Может быть несколько доменов, совпадающих как подмножества, но несущих различный смысл.

Пример. Домены «Вес детали» и «Имеющееся количество» можно одинаково описать как множество неотрицательных целых чисел, но смысл этих доменов будет различным, и это будут различные домены.

Основное значение доменов состоит в том, что домены ограничивают сравнения. Некорректно сравнивать значения из различных доменов, даже если они имеют одинаковый тип. В этом проявляется смысловое ограничение доменов.

Отношение, атрибут, кортеж

Атрибут – это свойство, характеризующее объект.

Атрибут отношения есть пара вида

<Имя_атрибута : Имя_домена>.

В структуре таблицы каждый атрибут именуется и ему соответствует заголовок некоторого столбца таблицы.

Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов.

Отношение R, определенное на множестве доменов D_1, D_2, …, D_n (не обязательно различных), содержит: ***заголовок*** и ***тело***.

Заголовок отношения (схема отношения) содержит фиксированное количество атрибутов отношения: ($<A_1 : D_1>$, $<A_2 : D_2>$, …, $<A_n : D_n>$).

Тело отношения содержит множество кортежей отношения.

Каждый **кортеж отношения** представляет собой множество пар вида *<Имя атрибута : Значение атрибута>*:

$$(<A_1 : Val_1>, <A_2 : Val_2>, \ldots, <A_n : Val_n>)$$

таких, что значение Val_i атрибута A_i принадлежит домену D_i.

Отношение – это множество кортежей, соответствующих одной схеме отношения.

Отношение обычно записывается в виде:

$$R(<A_1 : D_1>, <A_2 : D_2>, \ldots, <A_n : D_n>),$$

или короче

$$R(A_1, A_2, \ldots, A_n).$$

Число атрибутов в отношении называют **степенью (арностью)** отношения.

Мощность множества кортежей отношения называют **мощностью отношения.**

Схема БД (в структурном смысле) – это набор именованных схем отношений.

Реляционная база данных – это набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

Свойства отношений

Свойства отношений непосредственно следуют из определения отношения.

1. ***В отношении нет одинаковых кортежей.*** Тело отношения есть множество кортежей и, как всякое множество, не может содержать неразличимые элементы. Таблицы в отличие от отношений могут содержать одинаковые строки.

2. ***Кортежи не упорядочены*** (сверху вниз). Тело отношения есть множество, а множество не упорядочено. Нельзя отождествить отношения и таблицы – строки в таблицах упорядочены. Одно и то же отношение может быть изображено разными таблицами, в которых строки идут в различном порядке.

3. ***Атрибуты не упорядочены*** (слева направо). Т.к. каждый атрибут имеет уникальное имя в пределах отношения, то порядок атрибутов не имеет значения. Нельзя отождествить отношения и таблицы – столбцы в таблице упорядочены. Одно и то же отношение может быть изображено разными таблицами, в которых столбцы идут в различном порядке.

4. ***Все значения атрибутов атомарны***. Это следует из того, что лежащие в их основе атрибуты имеют атомарные логически неделимые значения. Это следует из определения домена как потенциального множества значений простого типа данных, т.е. среди значений домена не могут содержаться множества значений (отношения).

Ключ

Ключ – элемент данных, который позволяет уникально идентифицировать отдельные экземпляры некоторого типа сущности (кортежи отношения).

Виды ключей:

- Первичные
- Сложные
- Потенциальные
- Альтернативные
- Внешние.

Первичный ключ (Primary Key) – это атрибут или группа атрибутов, уникально идентифицирующая каждый экземпляр сущности. Первичный ключ однозначно идентифицирует каждый кортеж отношения.

Пример: номер зачетной книжки (или идентификационный номер) студента, однозначно определяющий каждого студента.

Ключ может быть **составным (сложным),** т.е. состоять из нескольких атрибутов.

Каждое отношение обязательно имеет один или комбинацию атрибутов, которая может служить ключом. Ее существование гарантируется тем, что отношение – это множество, которое не содержит одинаковых элементов — кортежей. Если в отношении нет повторяющихся кортежей, а это значит, что, по крайней мере, вся совокупность атрибутов является ключом и обладает свойством однозначной идентификации кортежей отношения.

Пример: в отношении *R (ID_студента, ID_предмета, Оценка)*, хранящего информацию об успеваемости студентов, ключом является комбинация атрибутов (*ID_студента, ID_предмета)* для однозначного определения результатов сдачи экзаменов каждого студента по каждому предмету.

Возможны случаи, когда отношение имеет несколько комбинаций атрибутов, каждая из которых однозначно определяет все кортежи отношения. Все эти комбинации атрибутов являются **потенциальными (возможными) ключами** отношения. Любой из потенциальных ключей может быть выбран как первичный.

Альтернативный ключ (Alternate Key) – это потенциальный ключ, не ставший первичным.

Реляционная модель представляет БД в виде множества взаимосвязанных отношений. Связи между отношениями устанавливаются с помощью **внешних ключей.**

Пусть в отношении *R1* имеется не ключевой атрибут *A*, значения которого являются значениями ключевого атрибута *B* другого отношения *R2*. Тогда говорят, что атрибут *A* отношения *R1* есть **внешний ключ** (**Foreign Key**).

Реляционная модель накладывает на внешние ключи ограничение для обеспечения целостности данных, называемое **ссылочной целостностью**. Это означает, что каждому значению внешнего ключа должны соответствовать строки в связываемых отношениях.

Целостная часть

Логические ограничения, накладываемые на данные, называются **ограничениями целостности**. СУБД должна контролировать соответствие данных заданным ограничениям при переводе БД из одного состояния в другое. Использование ограничений связано также с адекватностью отражения ПрО.

Явные ограничения задаются семантикой ПрО.

Они описывают области допустимых значений атрибутов, соотношение между атрибутами, динамику их изменения и т.д. Внутренние ограничения свойственны собственно модели данных.

Они накладываются на структуру отношений, на связи, на допустимые значения наборов данных, заложенные в выбранной модели данных. Способы реализации внутренних ограничений целостности зависят от СУБД.

В реляционной модели данных фиксируются два базовых внутренних ограничения целостности, которые должны поддерживаться в любой реляционной СУБД: требования ***целостности сущностей*** и ***целостности по ссылкам***.

Целостность сущностей

Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Любой кортеж любого отношения отличим от любого другого кортежа этого отношения, т.е. другими словами, **любое отношение должно обладать первичным ключом**. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

Целостность по ссылкам

При соблюдении нормализованности отношений сложные сущности представляются в реляционной БД в виде нескольких кортежей нескольких отношений.

Требование **целостности по ссылкам**, или **требование внешнего ключа** состоит в том, что ***для каждого значения внешнего ключа***, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, ***должен найтись кортеж с таким же значением первичного ключа***, или значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать).

Существуют три подхода, каждый из которых поддерживает целостность по ссылкам.

Первый подход заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно удалить ссылающиеся кортежи или соответствующим образом изменить значения их внешнего ключа).

При ***втором подходе*** при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным.

Третий подход (**каскадное удаление**) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

Практический подход к представлению данных в таблицах

В **реляционной модели данных** информация хранится в **связанных таблицах. Отдельная таблица (отношение)** обычно представляет совокупность (группу) либо реальных объектов, либо некоторых абстрактных концепций, либо событий одного типа. Каждая запись в таблице идентифицирует один **объект группы.**

Таблица состоит из строк и столбцов, называемых соответственно **записями и полями** (**кортежами и атрибутами**).

Слайд

3. Необходимость нормализации схемы отношений. Нормальные формы. Достоинства и недостатки нормализации

Процесс проектирования БД в терминах реляционной модели данных основывается на методе последовательного приближения к удовлетворительному набору схем отношений.

Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих лучшими свойствами.

Процесс проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая НФ обладает свойствами лучшими, чем предыдущая.

Нормализация – процесс преобразования отношения, имеющего некоторые недостатки, в отношение, которое этих недостатков не имеет.

Каждой НФ соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных БД выделяется следующие НФ:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (3НФ);
- нормальная форма Бойса-Кодда (НФБК);
- четвертая нормальная форма (4НФ);
- пятая нормальная форма, или нормальная форма проекции-соединения (5НФ).

Основные свойства нормальных форм:

• каждая следующая НФ в некотором смысле лучше предыдущей;

• при переходе к следующей НФ свойства предыдущих нормальных свойств сохраняются.

В основе **процесса нормализации** лежит **метод декомпозиции отношения,** находящегося в предыдущей НФ, в два или более отношения, удовлетворяющих требованиям следующей НФ.

Первая нормальная форма (1НФ)

Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, считается, что исходный набор отношений уже соответствует этому требованию.

Отношение находится в 1НФ, если все атрибуты отношения принимают простые значения (атомарные или неделимые), не являющимися множеством или кортежем из более элементарных составляющих.

Пример. Отношение *СТУДЕНТ_КАФЕДРА_СЕКЦИЯ (Код кафедры студента, Номер зачетки, ФИО студента, Телефон кафедры, Секция, Плата)* описывает студенческий спортивный клуб.

Потенциальный ключ отношения – *(Номер зачетки, Секция)*.

Отношение *СТУДЕНТ_КАФЕДРА_СЕКЦИЯ*

Код кафедры	Номер зачетки	ФИО студента	Телефон кафедры	Секция	Плата
101	123400	Васильева А.Н.	12-87	Аэробика	75
101	123408	Петренко Т.С.	12-87	Шейпинг	70
102	123411	Иванов Н.И.	12-32	Борьба	60
102	123411	Иванов Н.И.	12-32	Футбол	65
103	123403	Кононов С.В.	11-06	Борьба	60

Отношение *СТУДЕНТ_КАФЕДРА_СЕКЦИЯ* находится в 1НФ, т. к. соблюдены основные свойства отношения: в отношении нет одинаковых кортежей; кортежи не упорядочены; атрибуты не упорядочены и различаются по наименованию; все значения атрибутов атомарные.

Однако в данном отношении данные хранятся с большой избыточностью. При изменении состояния предметной области и соответствующих изменениях состояния БД возникают проблемы, которые называются аномалиями.

Аномалии – это неадекватность модели данных предметной области, либо некоторые дополнительные трудности в реализации ограничений предметной области средствами СУБД.

Причина возникновения аномалий - наличие в отношении разнородной информации.

Существуют три вида аномалий: вставки, удаления и модификации.

Аномалии вставки. В отношение *СТУДЕНТ_КАФЕДРА_СЕКЦИЯ* нельзя вставить данные о студенте, который пока не посещает секцию. Поскольку атрибут ***Секция*** входит в состав потенциального ключа, и, следовательно, не может содержать null-значений.

Аномалии удаления. В отношении *СТУДЕНТ_КАФЕДРА_СЕКЦИЯ* при удалении некоторых данных может произойти потеря другой информации.

При удалении информации о студенте Кононове теряется информацию о кафедре 103.

Аномалии обновления. В отношении *СТУДЕНТ_КАФЕДРА_СЕКЦИЯ* присутствует повторение данных в некоторых кортежах. Изменения этих данных должны одновременно выполняться во всех местах, иначе отношение станет некорректным. При изменении телефона кафедры 102 необходимо внести эти изменения в нескольких местах. Т. о., обновление БД одним действием реализовать невозможно.

Метод нормализации отношений применяется для устранения указанных аномалий.

Нормализация основана на понятии **функциональной зависимости** атрибутов отношения.

Функциональная зависимость. Пусть R – отношение. Множество атрибутов Y ***функционально зависимо*** от множества атрибутов X (X функционально определяет Y) тогда и только тогда, когда для любого состояния отношения R для любых кортежей $r_1, r_2 \in R$ из того, что $r_1.X = r_2.X$ следует, что $r_1.Y = r_2.Y$. (т.е. во всех кортежах, имеющих одинаковые значения атрибутов X, значения атрибутов Y также совпадают в любом состоянии отношения R).

Символическая запись функциональной зависимости: $\boldsymbol{X \rightarrow Y}$.

Множество атрибутов X называется ***детерминантом*** функциональной зависимости, а множество атрибутов Y называется ***зависимой частью***.

Если атрибуты X составляют потенциальный ключ отношения R, то любой атрибут отношения R функционально зависит от X.

Примеры функциональных зависимостей в отношении *СТУДЕНТ_КАФЕДРА_СЕКЦИЯ*:

1. Зависимость атрибутов от ключа отношения:

{Номер зачетки, Секция} → Код кафедры.

{Номер зачетки, Секция} → ФИО студента.

{Номер зачетки, Секция} → Телефон кафедры.

{Номер зачетки, Секция} → Плата.

2. Зависимость атрибутов, характеризующих студента от номера зачетки студента:

Номер зачетки → Код кафедры.

Номер зачетки → ФИО студента.

Номер зачетки → Телефон кафедры.

3. Зависимость стоимости от названия секции:

Секция → Плата.

Вторая нормальная форма (2НФ)

Отношение находится во 2НФ тогда и только тогда, когда отношение находится в 1НФ, и нет неключевых атрибутов, зависящих от части сложного ключа. Если потенциальный ключ отношения является простым, то отношение автоматически находится во 2НФ.

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

Алгоритм перехода от 1НФ к 2НФ

Отношения находятся в 1НФ.

Шаг 1. Обнаруживаем в отношениях зависимость атрибутов от части сложного ключа.

Шаг 2. Проводим декомпозицию этих отношений на несколько отношений:

- те атрибуты, которые зависят от части сложного ключа, выносятся в отдельное отношение вместе с этой частью ключа;
- в исходном отношении остаются все ключевые атрибуты.

Приведение отношения ко 2НФ в формализованном виде:

Исходное отношение: $R(K_1, K_2, A_1, \ldots, A_n, B_1, \ldots, B_m)$.

Ключ: $\{K_1, K_2\}$ – сложный.

Функциональные зависимости:

$\{K_1, K_2\} \rightarrow \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$ – зависимость всех атрибутов от ключа отношения.

$\{K_1\} \rightarrow \{A_1, \ldots, A_n\}$ – зависимость некоторых атрибутов от части сложного ключа.

Декомпозированные отношения:

$R_1(K_1, K_2, B_1, \ldots, B_m)$ – остаток от исходного отношения. Ключ *{K1, K2}*.

$R_2(K_1, A_1, \ldots, A_n)$ – атрибуты, вынесенные из исходного отношения вместе с частью сложного ключа. Ключ $\{K_1\}$.

В отношении *СТУДЕНТ_КАФЕДРА_СЕКЦИЯ* присутствуют неключевые атрибуты, зависящие от части сложного ключа:

Номер зачетки → Код кафедры.

Номер зачетки → ФИО студента.

Номер зачетки → Телефон кафедры.

Секция → Плата.

Для приведения к 2НФ в соответствии с алгоритмом отношение *СТУДЕНТ_КАФЕДРА_СЕКЦИЯ* декомпозируем на два отношения – *СТУДЕНТ_КАФЕДРА* и *СЕКЦИЯ*.

Отношение *СТУДЕНТ_КАФЕДРА*

Код кафедры	Номер зачетки	ФИО студента	Телефон кафедры	Секция
101	123400	Васильева А.Н.	12-87	Аэробика
101	123408	Петренко Т.С.	12-87	Шейпинг
102	123411	Иванов Н.И.	12-32	Борьба
102	123411	Иванов Н.И.	12-32	Футбол
103	123403	Кононов С.В.	11-06	Борьба

Отношение *СЕКЦИЯ*

Секция	Плата
Аэробика	75
Шейпинг	70
Борьба	60
Футбол	65

Оставшиеся аномалии вставки. В отношение СТУДЕНТ_КАФЕДРА нельзя вставить кортеж {103, 123413, Пушников А.И., 15-08}, так как при этом получится, что два студента одной кафедры под номером 103 (Кононов С.В. и Пушников А.И.) имеют разные номера телефонов кафедры, а это противоречит модели предметной области.

Оставшиеся аномалии обновления. Одни и те же номера телефонов кафедры повторяются в некоторых кортежах отношения.

Поэтому если на кафедре меняется номер телефона, то такие изменения необходимо одновременно выполнить во всех местах, где этот номер телефона встречаются, иначе отношение станет некорректным.

Оставшиеся аномалии удаления. При удалении некоторых данных по-прежнему может произойти потеря другой информации.

Третья нормальная форма (3НФ)

Атрибуты называются ***взаимно независимыми***, если ни один из них не является функционально зависимым от другого.

***Отношение находится в* 3НФ** тогда и только тогда, когда отношение находится во 2НФ и все неключевые атрибуты взаимно независимы.

Если для атрибутов *A*, *B* и *C* некоторого отношения существуют зависимости вида *A* → *B* и *B* → *C*, это означает , что атрибут *C* ***транзитивно зависит*** от атрибута *A* через атрибут *B* (при условии, что атрибут *A* функционально не зависит ни от атрибута *B*, ни от атрибута *C*).

3НФ свободна от транзитивных зависимостей.

Алгоритм перехода от 2НФ к 3НФ

Отношения находятся во 2НФ.

Шаг 1. Обнаруживаем зависимость некоторых неключевых атрибутов от других неключевых атрибутов.

Шаг 2. Проводим декомпозицию этих отношений следующим образом:

- те неключевые атрибуты, которые зависят от других неключевых атрибутов, выносятся в отдельное отношение;

- в новом отношении ключом становится детерминант функциональной зависимости.

Приведение отношения к 3НФ в формализованном виде:

Исходное отношение: $R(K, A1, \ldots, A_n, B_1, \ldots, B_m)$.

Ключ: $\{K_1\}$.

Функциональные зависимости:

$\{K\} \rightarrow \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$ – зависимость всех атрибутов от ключа отношения.

$\{A1, \ldots, An\} \rightarrow \{B1, \ldots, Bm\}$ – зависимость некоторых неключевых атрибутов от других неключевых атрибутов.

Декомпозированные отношения:

R1(K, A1, …, An) – остаток от исходного отношения. Ключ *{K}*.

R2(A1, …, An, B1, …, Bm) – атрибуты, вынесенные из исходного отношения вместе с детерминантом функциональной зависимости. Ключ *{ A1, …, An }*.

В отношении *СТУДЕНТ_КАФЕДРА* присутствует функциональная зависимость неключевых атрибутов: *Код кафедры → Телефон кафедры*.

Для приведения к 3НФ в соответствии с алгоритмом отношение *СТУДЕНТ_КАФЕДРА* декомпозируем на два отношения – *СТУДЕНТ* и *КАФЕДРА*.

Отношение *СТУДЕНТ*

Код кафедры	Номер зачетки	ФИО студента
101	123400	Васильева А.Н.
101	123408	Петренко Т.С.
102	123411	Иванов Н.И.
103	123403	Кононов С.В.

Отношение *КАФЕДРА*

Код кафедры	Телефон кафедры
101	12-87
102	12-32
103	11-06

Декомпозиция отношений

Декомпозиция отношения – это взятие одной или нескольких проекций исходного отношения так, чтобы эти проекции в совокупности содержали (возможно, с повторениями) все атрибуты исходного отношения.

При восстановлении исходного отношения путем соединения проекций не должны появиться новые атрибуты, т.е. необходимо использовать естественное соединение.

Проекция *R[X]* отношения *R* на множество атрибутов называется ***собственной,*** если множество атрибутов *X* является собственным подмножеством множества атрибутов отношения *R* (т.е. множество атрибутов *X* не совпадает с множеством всех атрибутов отношения *R*).

Собственные проекции *R1* и *R2* отношения *R* называются ***декомпозицией без потерь***, если отношение *R* точно восстанавливается из них при помощи естественного соединения для любого состояния отношения *R*: *R1 JOIN R2*.

Декомпозиция без потерь – корректность процедуры нормализации.

Теорема (Хеза)

Пусть *R(A, B, C)* является отношением, и *A, B, C* – атрибуты или множества атрибутов этого отношения. Если имеется функциональная зависимость $A \rightarrow B$, то проекции *R1 = R[A, B]* и *R2 = R[A, C]* образуют ***декомпозицию без потерь*.**

Основной смысл теоремы Хеза заключается в доказательстве того, что при этом не появятся новые кортежи, отсутствовавшие в исходном отношении.

При выполнении декомпозиции и последующем восстановлении отношения при помощи естественного соединения, кортежи исходного отношения не будут потеряны

Нормальная форма Бойса-Кодда (НФБК)

Отношение находится в нормальной форме Бойса-Кодда (**НФБК**) тогда и только тогда, когда детерминанты всех функциональных зависимостей являются потенциальными ключами.

Если отношение находится в НФБК, то оно автоматически находится и в 3НФ.

Алгоритм нормализации (приведение к НФБК)

Шаг 1. Если имеются отношения, содержащие несколько потенциальных ключей, проверяем, имеются ли функциональные зависимости, детерминанты которых не являются потенциальными ключами.

Шаг 2. Если функциональные зависимости имеются, проводим дальнейшую декомпозицию отношений: те атрибуты, которые зависят от детерминантов, не являющихся потенциальными ключами, выносятся в отдельное отношение вместе с детерминантами.

Пример. Требуется хранить данные о получении книг некоторыми кафедрами. Код кафедры и название являются уникальными.

Отношение *КАФЕДРА_КНИГИ* содержит два потенциальных ключа: *{Код кафедры, Код книги}* и *{Название кафедры, Код книги}* и находится в 3НФ.

Отношение *КАФЕДРА_КНИГИ*

Код кафедры	Название кафедры	Код книги	Количество книг
101	Прикладная информатика	23012	12
101	Прикладная информатика	23115	15
101	Прикладная информатика	40108	15
102	Автоматизированные системы управления	40478	20
102	Автоматизированные системы управления	24566	34
103	Информационные технологии	23111	17

Отношение *КАФЕДРА_КНИГИ* не находится в НФБК, так как имеются зависимости *Код кафедры* → *Название кафедры* и *Название кафедры* → *Код кафедры*, детерминанты которых не являются потенциальными ключами.

Проведем декомпозицию, вынося эти детерминанты и зависимые от них части в отдельное отношение. Отношения *КАФЕДРА* и *ПОЛУЧАЕМЫЕ_КНИГИ*, полученные в результате декомпозиции, находятся в НФБК.

Отношение *КАФЕДРА*

Код кафедры	Название кафедры
101	Прикладная информатика
102	Автоматизированные системы управления
103	Информационные технологии

Отношение *ПОЛУЧАЕМЫЕ_КНИГИ*

Код кафедры	Код книги	Количество книг
101	23012	12
101	23115	15
101	40108	15
102	40478	20
102	24566	34
103	23111	17

Многозначная зависимость

Пусть R – отношение, и X, Y, Z – некоторые из его атрибутов (или непересекающиеся множества атрибутов). Тогда атрибуты (множества атрибутов) Y и Z *многозначно зависят* от X (обозначается $X \rightarrow\rightarrow Y|Z$), тогда и только тогда, когда из того, что в отношении R содержатся кортежи $r_1 = (x, y, z_1)$ и $r_2 = (x, y_1, z)$ следует, что в отношении R содержится также и кортеж $r_3 = (x, y, z)$.

Теорема (Фейджина)

Пусть X, Y, Z – непересекающиеся множества атрибутов отношения $R(X, Y, Z)$. Декомпозиция отношения R на проекции $R_1 = R[X, Y]$ и $R_2 = R[X, Z]$ будет ***декомпозицией без потерь*** тогда и только тогда, когда имеется многозначная зависимость $X \rightarrow\rightarrow Y|Z$.

Многозначная зависимость $X \rightarrow\rightarrow Y|Z$ называется ***нетривиальной многозначной зависимостью***, если не существует функциональных зависимостей $X \rightarrow Y$ и $X \rightarrow Z$.

Четвертая нормальная форма (4НФ)

Отношение находится в четвертой нормальной форме **4НФ** тогда и только тогда, когда отношение находится в НФБК и не содержит нетривиальных многозначных зависимостей.

Алгоритм перехода от НФБК к 4НФ

Отношения находятся в НФБК.

Если в отношениях обнаружены нетривиальные многозначные зависимости, проводим декомпозицию для исключения таких зависимостей.

Пример. Пусть требуется учитывать данные об абитуриентах, поступающих в вуз. При анализе предметной области были выделены следующие требования:

- Каждый абитуриент имеет право сдавать экзамены на несколько факультетов одновременно.
- Каждый факультет имеет свой список сдаваемых предметов.
- Один и тот же предмет может сдаваться на нескольких факультетах.
- Абитуриент обязан сдавать все предметы, указанные для факультета, на который он поступает, несмотря на то, что он, может быть, уже сдавал такие же предметы на другом факультете.

Предположим, что нам требуется хранить данные о том, какие предметы должен сдавать каждый абитуриент.

Отношение *АБИТУРИЕНТЫ_ФАКУЛЬТЕТЫ_ПРЕДМЕТЫ*

Абитуриент	Факультет	Предмет
Иванов	Математический	Математика
Иванов	Математический	Информатика
Иванов	Физический	Математика
Иванов	Физический	Физика
Петров	Математический	Математика
Петров	Математический	Информатика

Отношение *АБИТУРИЕНТЫ_ФАКУЛЬТЕТЫ_ПРЕДМЕТЫ* находится в НФБК, т. к. единственным потенциальным ключом этого отношения является набор всех атрибутов.

В отношении *АБИТУРИЕНТЫ_ФАКУЛЬТЕТЫ_ПРЕДМЕТЫ* имеется нетривиальная многозначная зависимость *Факультет*$\rightarrow\rightarrow$*Абитуриент*|*Предмет*.

Отношение *АБИТУРИЕНТЫ_ФАКУЛЬТЕТЫ_ПРЕДМЕТЫ* не находится в 4НФ. Согласно теореме Фейджина, это отношение можно без потерь декомпозировать на отношения *ФАКУЛЬТЕТЫ_АБИТУРИЕНТЫ* и *ФАКУЛЬТЕТЫ_ ПРЕДМЕТЫ*.

Отношение *ФАКУЛЬТЕТЫ_АБИТУРИЕНТЫ*

Факультет	Абитуриент
Математический	Иванов
Физический	Иванов
Математический	Петров

Отношение *ФАКУЛЬТЕТЫ_ПРЕДМЕТЫ*

Факультет	Предмет
Математический	Математика
Математический	Информатика
Физический	Математика
Физический	Физика

Зависимость соединения

Пусть *R* является отношением, а *A, B, ..., Z* – произвольными (возможно пересекающимися) подмножествами множества атрибутов отношения *R*. Тогда отношение *R* удовлетворяет *зависимости соединения *(A, B, ..., Z)* тогда и только тогда, когда оно равносильно соединению всех своих проекций с подмножествами атрибутов *A, B, ..., Z*, т.е. *R=R[A] JOIN R[B] JOIN ... JOIN R[Z]*.

Теорема Фейджина (другая формулировка)

Отношение *R(X, Y, Z)* удовлетворяет зависимости соединения **(XY, XZ)* тогда и только тогда, когда имеется многозначная зависимость $X \rightarrow\rightarrow Y|Z$.

Зависимость соединения **(A, B, ..., Z)* называется ***нетривиальной зависимостью соединения,*** если выполняются два условия:

- одно из множеств атрибутов *A, B, ..., Z* не содержит потенциального ключа отношения *R*;
- ни одно из множеств атрибутов не совпадает со всем множеством атрибутов отношения *R*.

Зависимость соединения **(A, B, ..., Z)* называется ***тривиальной зависимостью соединения***, если выполняется одно из условий:

- либо все множества атрибутов *A, B, ..., Z* содержат потенциальный ключ отношения *R*;
- либо одно из множеств атрибутов совпадает со всем множеством атрибутов отношения *R*.

Пятая нормальная форма (5НФ)

Отношение находится в **5НФ** тогда и только тогда, когда любая имеющаяся зависимость соединения является тривиальной.

Алгоритм перехода от 4НФ к 5НФ

Отношения находятся в 4 НФ.

Если в отношениях обнаружены нетривиальные зависимости соединения, то необходимо провести декомпозицию для исключения таких зависимостей.

Пример. Предположим, что нужно хранить данные об ассортименте нескольких продавцов, торгующих продукцией нескольких фирм (номенклатура товаров фирм может пересекаться).

Отношение *ПРОДАВЦЫ_ФИРМЫ_ТОВАРЫ*

Продавец	Фирма	Товар
Иванов	Samsung	Телевизор
Иванов	Samsung	Пылесос
Петров	Philips	Холодильник
Петров	Philips	Телевизор
Петров	Philips	Пылесос
Петров	Philips	Смартфон
Сидоров	Philips	Телевизор
Сидоров	Philips	Радионаушники
Сидоров	Samsung	Телевизор
Сидоров	Samsung	Монитор
Сидоров	Genius	Клавиатура

Если дополнительных условий нет, то данное отношение, которое находится в 4НФ, является корректным и отражает все необходимые ограничения.

Пусть нужно учесть следующее ограничение: каждый продавец имеет в своём ассортименте ограниченный список фирм и ограниченный список типов товаров и предлагает товары из списка товаров, производимых фирмами из списка фирм.

Т. е. продавец не имеет права торговать какими угодно товарами каких угодно фирм. Если продавец *П* имеет право торговать товарами фирмы *Ф*, и если продавец *П* имеет право торговать товарами типа *Т*, то в этом случае в ассортимент продавца *П* входят товары типа *Т* фирмы *Ф* при условии, что фирма *Ф* производит товары типа *Т*.

Такое ограничение может быть вызвано, например, тем, что список типов товаров продавца ограничен имеющимися у него лицензиями, либо знаниями и квалификацией, необходимыми для их продажи, а список фирм каждого продавца определён партнёрскими соглашениями.

В частности, предполагается, что продавец Иванов имеет право торговать товарами только фирмы «Sumsung», продавец Петров — товарами только фирмы «Philips», зато продавец Сидоров не имеет права торговать пылесосами и холодильниками и т.д.

Отношение *ПРОДАВЦЫ_ФИРМЫ_ТОВАРЫ* не может исключить ситуаций, при которых данное ограничение будет нарушено.

Ничто не препятствует занести данные о торговле товаром, который данная фирма вообще не выпускает, либо данные о торговле товарами той фирмы, которую данный продавец не обслуживает, либо данные о торговле таким типом товара, который данный продавец не имеет права продавать.

Отношение не находится в 5НФ, поскольку в нём есть нетривиальная зависимость соединения **({Продавец, Фирма}, {Фирма, Товар}, {Продавец, Фирма})*, так как подмножества *{Продавец, Фирма}, {Фирма, Товар}, {Продавец, Товар}* не содержат потенциального ключа отношения и не совпадают со всем множеством атрибутов отношения *ПРОДАВЦЫ_ФИРМЫ_ТОВАРЫ*.

В данном случае для приведения к 5НФ отношение должно быть разбито на три:

- *ПРОДАВЕЦ_ФИРМА*,
- *ФИРМА_ТОВАР*,
- *ПРОДАВЕЦ_ТОВАР*.

Отношение *ПРОДАВЕЦ_ФИРМА*

Продавец	Фирма
Иванов	Samsung
Петров	Philips
Сидоров	Philips
Сидоров	Acer
Сидоров	Genius

Отношение *ФИРМА_ТОВАР*

Фирма	Товар
Samsung	Телевизор
Samsung	Пылесос
Samsung	Монитор
Philips	Холодильник
Philips	Телевизор
Philips	Пылесос
Philips	Смартфон
Philips	Радионаушники
Gunius	Клавиатура

Отношение *ПРОДАВЕЦ_ТОВАР*

Продавец	Товар
Иванов	Телевизор
Иванов	Пылесос
Петров	Холодильник
Петров	Телевизор
Петров	Пылесос
Петров	Смартфон
Сидоров	Телевизор
Сидоров	Радионаушники
Сидоров	Телевизор
Сидоров	Монитор
Сидоров	Клавиатура

Контрольные вопросы для самоподготовки

1. Какова основная цель нормализации?
2. Назовите основные свойства отношения.
3. Назовите правило приведения отношения ко 2НФ.
4. Что называется функциональной зависимостью?
5. Назовите правило приведения отношения к 3НФ.
6. Что называется декомпозицией отношения без потерь?
7. Что называется многозначной зависимостью?
8. Назовите правило приведения отношения к НФБК.
9. Назовите правило приведения отношения к 4НФ.
10. Что называется зависимостью соединения?
11. Назовите правило приведения отношения к 5НФ.

4. Реляционная алгебра. Операции реляционной алгебры. Реляционное исчисление

Манипуляционная часть

Для манипулирования данными в реляционной модели используются два формальных аппарата:

– **реляционная алгебра**, основанная на теории множеств;

– **реляционное исчисление**, базирующееся на исчислении предикатов первого порядка.

Реляционная алгебра и реляционное исчисление предложены Э. Коддом (1971 г.) в качестве основы для создания реляционных языков.

Реляционную алгебру можно описать как язык (высокоуровневый процедурный язык), который может быть использован, чтобы сообщить СУБД, как следует построить требуемое отношение на базе одного или нескольких существующих в БД отношений.

Реляционная алгебра – замкнутая система операций над отношениями в реляционной модели данных.

Реляционная алгебра – это процедурный язык, связанный с процессами: для достижения желаемых результатов может потребоваться провести последовательность действия (операций).

Операции можно разделить на две группы.

Реляционное исчисление с неформальной точки зрения представляет собой непроцедурный язык, который можно использовать для определения того, каким будет некоторое отношение, созданное на основе одного или нескольких других отношений БД.

Реляционная алгебра и реляционное исчисление эквивалентны друг другу, т.е. для каждого выражения алгебры существует эквивалентное выражение в реляционном исчислении (и наоборот).

Реляционное исчисление используется для оценки избирательной мощности реляционных языков.

Язык называется реляционно полным, если позволяет получить любое отношение, которое можно вывести с помощью реляционного исчисления. Большинство реляционных языков запросов является реляционно полными.

Реляционная алгебра и реляционное исчисление – формальные языки. В реляционных БД они использовались в качестве основы для разработки других языков управления данными более высокого уровня.

1.Реляционная алгебра – это теоретический язык операций, которые на основе одного или нескольких отношений позволяют создавать другое отношение без изменения самих исходных отношений.

Т.к., оба операнда и результат являются отношениями, следовательно результаты одной операции могут стать исходными данными для другой операции, что позволяет создавать вложенные выражения реляционной алгебры, как создаются вложенные арифметические выражения.

Это **свойство называется замкнутостью**, т.е. отношения покрываются реляционной алгеброй так же, как числа арифметическими операциями.

Реляционная алгебра является языком последовательного использования отношений, в котором все кортежи, возможно взятые даже из разных отношений, обрабатываются одной командой, без организации циклов.

Существует несколько вариантов выбора операций, которые включаются в реляционную алгебру. Изначально Э. Кодд предложил восемь операторов, но потом к ним были добавлены и другие.

Основное множество в реляционной алгебре - множество отношений.

Первую группу составляют **операции над множествами.**

В состав теоретико-множественных операций входят **операции**:

1) объединения отношений;

2) пересечения отношений;

3) взятия разности отношений;

4) взятия декартова произведения отношений.

Три первые операции являются бинарными (участвуют два отношения и требуют эквивалентных схем исходных отношений).

Схемы двух отношений называются эквивалентными, если имеют одинаковую степень и возможно такое упорядочение имен атрибутов в схемах, что на одинаковых местах будут находиться сравнимые атрибуты, т.е. атрибуты, принимающие значения из одного домена.

Вторую группу составляют специальные **операции над отношениями.**

Специальные реляционные операции включают:

1) **выборку** (ограничение отношения);

2) **проекцию** отношения;

3) **соединение** отношений;

4) **деление** отношений.

В состав алгебры включены:

– **операция присваивания**, позволяющая сохранить в БД результаты вычисления алгебраических выражений;

– **операция переименования атрибутов**, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

Теоретико-множественные операции

Объединение двух односхемных отношений *A* и *B* – отношение *C*, построенное по той же схеме и содержащее все кортежи отношения *A* и все кортежи отношения *B*:

$$C = A \ UNION \ B.$$

Слайд

Операция объединения отношений

Студенты первого курса

Студент_ID	ФИО	Курс
213400	Иванов А.А.	1
213408	Петров И.А.	1
213416	Зайцева О.Н.	1
213417	Краснова И.Н.	1

Студенты второго курса

Студент_ID	ФИО	Курс
211813	Колесников С.В.	2
211815	Морозова В.С.	2
211816	Симонов А.С.	2
211818	Ким А.П.	2
211819	Садовников М.С.	2

Студенты младших курсов

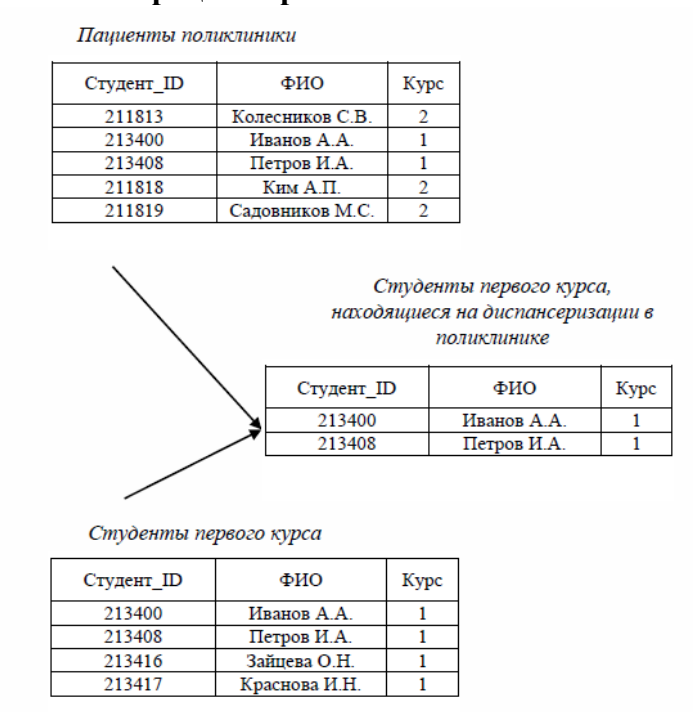
Студент_ID	ФИО	Курс
213400	Иванов А.А.	1
213408	Петров И.А.	1
213416	Зайцева О.Н.	1
213417	Краснова И.Н.	1
211813	Колесников С.В.	2
211815	Морозова В.С.	2
211816	Симонов А.С.	2
211818	Ким А.П.	2
211819	Садовников М.С.	2

Пересечение двух односхемных отношений *A* и *B* – отношение *C*, построенное по той же схеме и содержащее только те кортежи отношения *A*, которые есть в отношении *B*:

C = A INTERSECT B.

Слайд

Операция пересечения отношений



Пациенты поликлиники

Студент_ID	ФИО	Курс
211813	Колесников С.В.	2
213400	Иванов А.А.	1
213408	Петров И.А.	1
211818	Ким А.П.	2
211819	Садовников М.С.	2

Студенты первого курса, находящиеся на диспансеризации в поликлинике

Студент_ID	ФИО	Курс
213400	Иванов А.А.	1
213408	Петров И.А.	1

Студенты первого курса

Студент_ID	ФИО	Курс
213400	Иванов А.А.	1
213408	Петров И.А.	1
213416	Зайцева О.Н.	1
213417	Краснова И.Н.	1

Вычитание двух односхемных отношений *A* и *B* – отношение *C*, построенное по той же схеме и содержащее те кортежи отношения *A*, которых нет в отношении *B*:

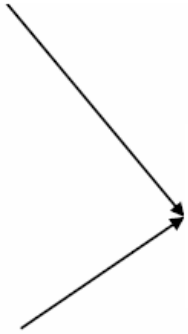
C = A MINUS B.

Слайд

Операция вычитания отношений

Студенты перового курса, прошедшие медосмотр

Студент_ID	ФИО	Курс
213400	Иванов А.А.	1
213408	Петров И.А.	1



Студенты первого курса, не прошедшие диспансеризацию

Студент_ID	ФИО	Курс
213416	Зайцева О.Н.	1
213417	Краснова И.Н.	1

Студенты первого курса

Студент_ID	ФИО	Курс
213400	Иванов А.А.	1
213408	Петров И.А.	1
213416	Зайцева О.Н.	1
213417	Краснова И.Н.	1

Декартово произведение двух отношений *A* и *B* – отношение *C*, схема которого включает все атрибуты отношений *A* и *B*, а тело отношения состоит из всевозможных сцеплений кортежей отношений *A* и *B*:

C = A TIMES B.

Слайд

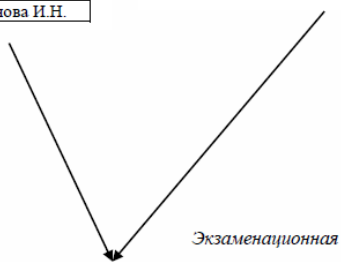
Декартово произведение отношений

Студенты

ФИО
Иванов А.А.
Петров И.А.
Зайцева О.Н.
Краснова И.Н.

Экзамены

Дисциплина	Дата	Оценка
Физика	16.06.16	
Информатика	23.06.16	



Экзаменационная ведомость

ФИО	Дисциплина	Дата	Оценка
Иванов А.А.	Физика	16.06.16	
Иванов А.А.	Информатика	23.06.16	
Петров И.А.	Физика	16.06.16	
Петров И.А.	Информатика	23.06.16	
Зайцева О.Н.	Физика	16.06.16	
Зайцева О.Н.	Информатика	23.06.16	
Краснова И.Н.	Физика	16.06.16	
Краснова И.Н.	Информатика	23.06.16	

Специальные реляционные операции

Выборка на отношении *A* – отношение *C*, построенное по той же схеме, что и отношение *A*, и содержащее подмножество кортежей отношения *A*, удовлетворяющих условию выборки:

C = A WHERE (Курс > 3)

Слайд

Операция выборки на отношении

Студенты

Студент_ID	ФИО	Курс
213400	Иванов А.А.	1
213408	Петров И.А.	1
211813	Колесников С.В.	2
211819	Садовников М.С.	2
206782	Круг М.В.	4
203579	Попов А.А.	5

Студенты старших курсов

Студент_ID	ФИО	Курс
206782	Круг М.В.	4
203579	Попов А.А.	5

Проекция отношения *A* – отношение *C*, схема которого состоит из подмножества атрибутов, по которым производится проекция, а кортежи содержат соответствующие значения из кортежей отношения *A*:

C = A [Номер студенческого билета, ФИО].

Слайд

Операция проекции отношения

Студенты второго курса

Студент_ID	ФИО	Курс
211813	Колесников С.В.	2
211815	Морозова В.С.	2
211816	Симонов А.С.	2
211818	Ким А.П.	2
211819	Садовников М.С.	2

Список студентов

Студент_ID	ФИО
211813	Колесников С.В.
211815	Морозова В.С.
211816	Симонов А.С.
211818	Ким А.П.
211819	Садовников М.С.

Соединение отношений *A* и *B* подобно декартовому произведению отношений, но сцепление кортежей отношений *A* и *B* происходит не каждое с каждым, а по некоторому условию. В зависимости от условия соединение называется: **естественным** – равенство значений общих атрибутов отношений *A* и *B*; э**квисоединением** – равенство значений атрибутов, входящих в условие соединения; **тета-соединением** – другой знак сравнения.

Естественное соединение отношений *A* и *B* – соединение по условию равенства значений некоторого общего атрибута отношений *A* и *B* (чаще всего равенство значений первичного и внешнего ключа):

C = A JOIN B = A TIMES B
WHERE A[Предмет]=B[Предмет].

Слайд

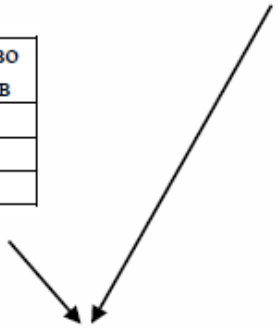
Естественное соединение отношений

Успеваемость студентов

Студент_ID	ФИО	Предмет	Оценка
213400	Иванов А.А.	Физика	3
213408	Петров И.А.	Мат. анализ	4
211813	Колесников С.В.	Физика	4
213400	Иванов А.А.	Ин. яз.	5
213408	Петров И.А.	Физика	4
203579	Попов А.А.	История	5

Профильные дисциплины

Предмет	Ведущий лектор	Кол-во часов
Физика	Козловский Д.А.	144
Мат. анализ	Швец А.С.	72
Лин. алгебра	Барбин П.П.	72



Успеваемость студентов по профильным дисциплинам

Студент_ID	ФИО	Предмет	Оценка	Ведущий лектор	Кол-во часов
213400	Иванов А.А.	Физика	3	Козловский Д.А.	144
213408	Петров И.А.	Мат. анализ	4	Швец А.С.	72
213408	Петров И.А.	Физика	4	Козловский Д.А.	144
211813	Колесников С.В.	Физика	4	Козловский Д.А.	144

Тета-соединение отношений *A* и *B* — соединение по условию неравенства значений атрибутов отношений *A* и *B*:

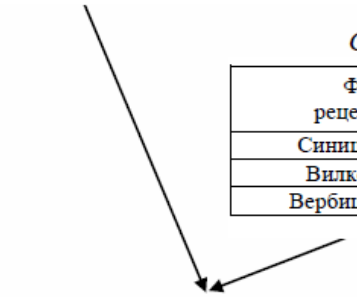
C = A TIMES B WHERE A[ФИО науч. руководителя] <>
B[ФИО рецензента].

Слайд

Тета-соединение отношений

Студенты и научные руководители

ФИО студента	ФИО науч. руководителя
Иванов А.А.	Синицын И.В.
Петров И.А.	Синицын И.В.
Колесников С.В.	Вилков А.Ф.



Список рецензентов

ФИО рецензента	Кафедра
Синицын И.В.	23
Вилков А.Ф.	12
Вербицкий А.В.	34

Возможные пары «Студент-Рецензент»

ФИО студента	ФИО науч. руководителя	ФИО рецензента	Ка-федра
Иванов А.А.	Синицын И.В.	Вилков А.Ф.	12
Иванов А.А.	Синицын И.В.	Вербицкий А.В.	34
Петров И.А.	Синицын И.В.	Вилков А.Ф.	12
Петров И.А.	Синицын И.В.	Вербицкий А.В.	34
Колесников С.В.	Вилков А.Ф.	Синицын И.В.	23
Колесников С.В.	Вилков А.Ф.	Вербицкий А.В.	34

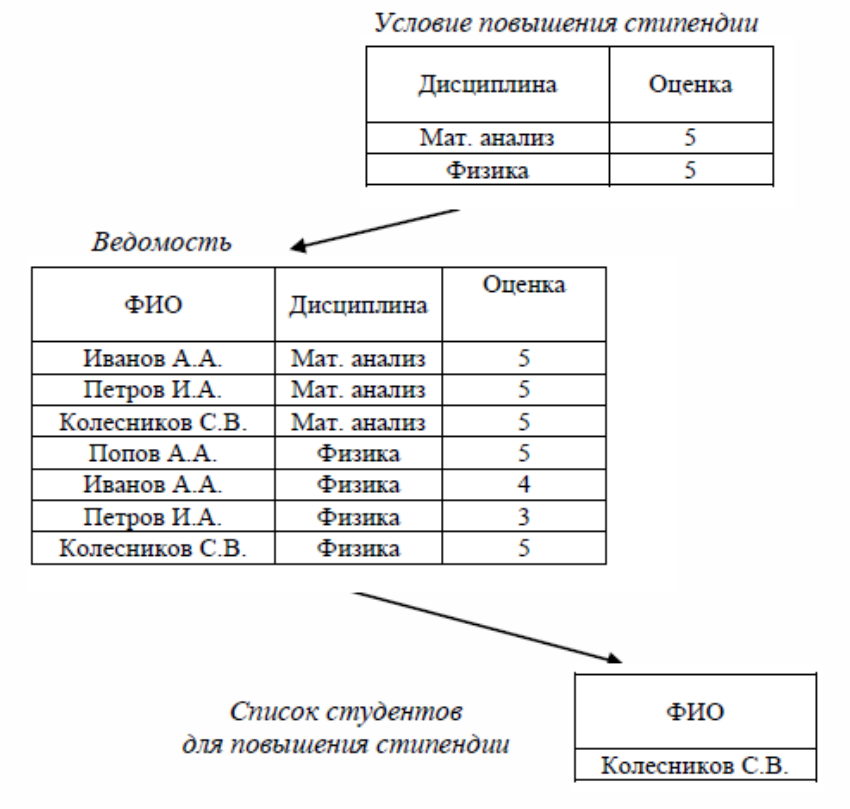
Деление выполняется над двумя отношениями A и B, имеющими в общем случае разные структуры и часть одинаковых атрибутов.

Пусть отношение A, называемое **делимым**, содержит атрибуты $(a_1, a_2, a_3, \ldots, a_n)$. Отношение B – **делитель** – содержит подмножество атрибутов отношения A, например, $(a_1, a_2, \ldots, a_к)$. Результирующее отношение C определено на атрибутах отношения A, которых нет в B, т.е. $(a_{к+1}, a_{к+2}, \ldots, a_n)$. Кортеж включается в отношение C только, если его декартово произведение с отношением B содержится в делимом отношении A:

$$C = A \; DIVIDE \; BY \; B.$$

Слайд

Операция деления



Контрольные вопросы для самоподготовки

1. Что называется доменом в реляционной модели данных?
2. Дайте понятие отношения, атрибута, картежа.
3. Что называется потенциальным ключом отношения?
4. Какие выделяют два базовых внутренних ограничения целостности?
5. Какие теоретико-множественные операции входят в состав реляционной алгебры? Дайте их краткое описание.
6. Какие специальные реляционные операции входят в состав реляционной алгебры? Дайте их краткое описание.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К СЕМИНАРСКИМ (ПРАКТИЧЕСКИМ) ЗАНЯТИЯМ

ПОСТРЕЛЯЦИОННАЯ СУБД – PostgreSQL. УСТАНОВКА СУБД. КОМПОНЕНТЫ СУБД

PostgreSQL является свободно распространяемой объектно-реляционной системой управления базами данных (СУБД) с открытым кодом. В настоящее время существуют разные версии PostgreSQL. Для установки PostgreSQL можно использовать, например, следующие версии: PostgreSQL 10.4, PostgreSQL 9.6.9, PostgreSQL 9.5.13, PostgreSQL 9.4.18 и др. для 32- и 64-разрядных ОС семейства Windows. При работе в 64-разрядной версии Windows должна быть использована 64-разрядная версия PostgreSQL сервера.

Для установки СУБД PostgreSQL использована программа установки PostgreSQL_9.4.18_32bit_Setup для 32-разрядной версии Windows.

1.1 Установка СУБД PostgreSQL

При работе в операционных системах семейства Windows одним из вариантов установки СУБД PostgreSQL является установка с использованием установщика по адресу: https://postgrespro.ru/windows.

Для установки PostgreSQL необходимо выполнить следующие шаги:

– Запустить программу установки PostgreSQL_9.4.18_32bit_Setup.

– После запуска файла установщика необходимо определиться с выбором языка установки. В окне *Installer Language* выбрать *Russian* (русский) и нажать кнопку *OK* (рис.1.1).

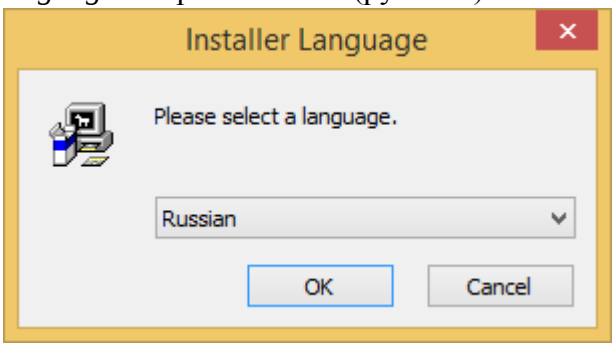


Рисунок 1.1 – Выбор языка установки в окне *Installer Language*

– Интерфейс программы установки выполнен в виде «мастера» (рис. 1.2). Затем нажать кнопку *Далее* (рис. 1.2).

– Принять лицензионное соглашение (рис. 1.3), нажав кнопку *Принимаю* (рис. 1.3).

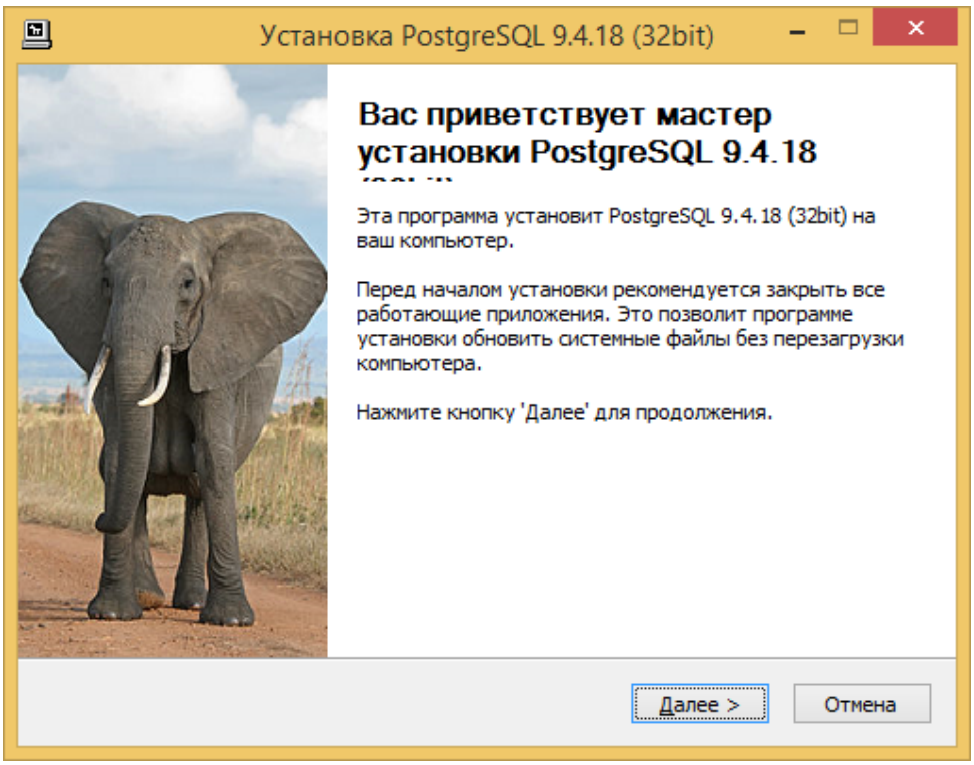


Рисунок 1.2 – Общий вид окна мастера установки PostgreSQL 9.4.18

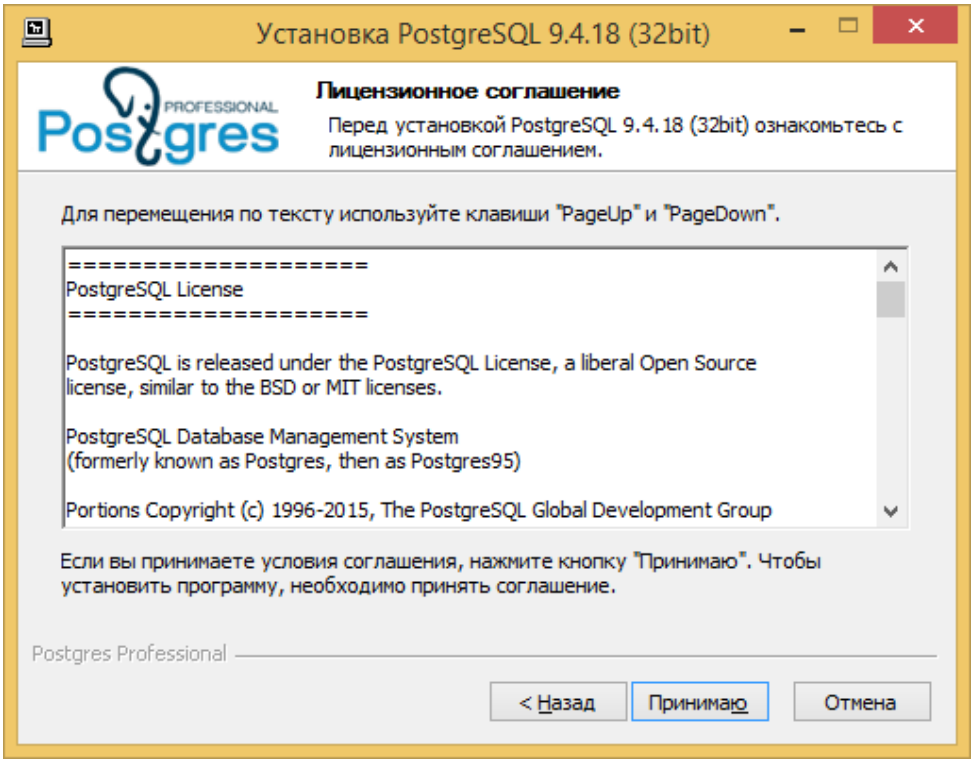


Рисунок 1.3 – Принятие лицензионного соглашения

– Выбрать нужные компоненты устанавливаемой программы и нажать кнопку *Далее* (рис. 1.4).

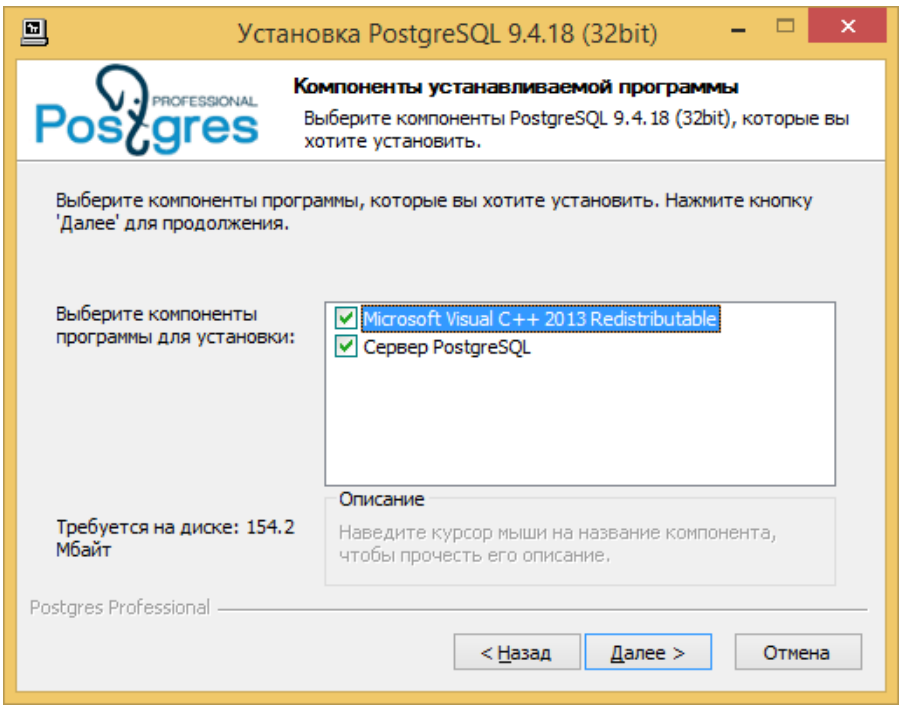


Рисунок 1.4 – Выбор компонентов устанавливаемой программы

Потом необходимо указать путь доступа и папку для установки программы PostgreSQL. По умолчанию установка программы будет произведена в папку C:\Program Files\PostgreSQL\9.4 (рис. 1.5). Нажать кнопку *Далее*.

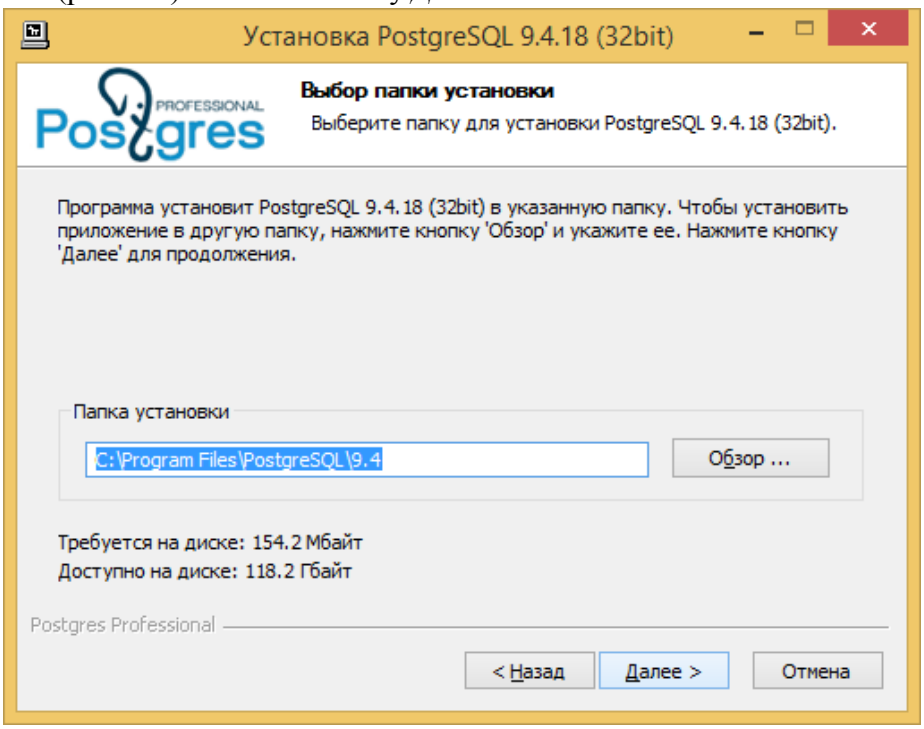


Рисунок 1.5 – Выбор папки для установки PostgreSQL

– Указать каталог для размещения баз данных. По умолчанию БД будут размещены в каталог C:\Program Files\PostgreSQL\9.4\data (рис. 1.6). Для размещения БД в другом каталоге щелкнуть кнопку *Обзор*. Выбрать или создать требуемую папку. Следует убедится, что на диске достаточно места для размещения баз данных. Нажать кнопку *Далее*.

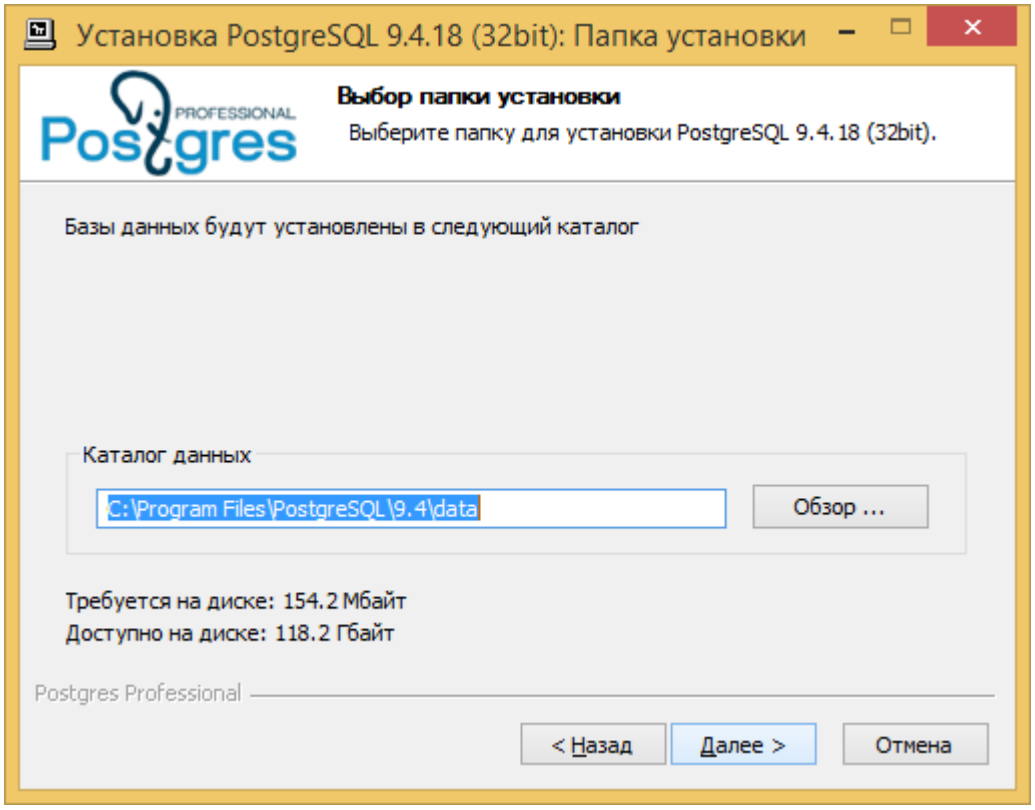


Рисунок 1.6 – Выбор каталога для размещения баз данных

– Следующий шаг – это ввод параметров сервера. Можно оставить значения поля *Порт* и флажка *Адреса* (Разрешать подключения с любых IP-адресов) по умолчанию. Значение поля *Локаль* (региональные параметры) можно оставить по умолчанию – «Настройка ОС», если используются русская локаль в ОС Windows, иначе надо выбрать значение «Russian, Russia», если данные будут храниться в СУБД на русском языке. Значение в поле *Супер-пользователь* можно оставить по умолчанию, а в поля *Пароль* и *Подтверждение* ввести соответственно пароль и его подтверждение. Кроме этого, надо взвести флажок *Настроить переменные среды* для подключения к серверу СУБД PostgreSQL под текущим пользователем операционной системы. Результат ввода параметров сервера приведен на рисунке 1.7. Нажать кнопку *Далее* для оптимизации параметров сервера.

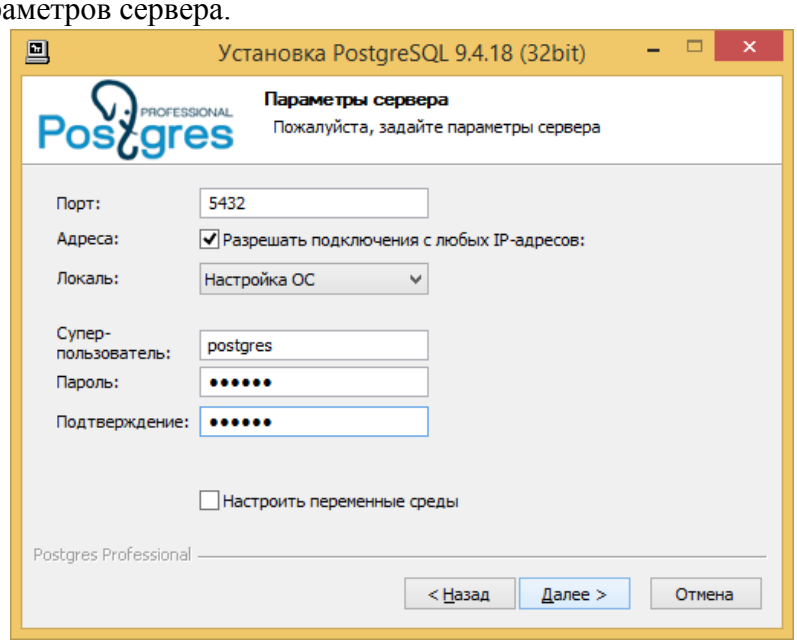


Рисунок 1.7 – Ввод параметров сервера СУБД

В появившемся окне для проведения оптимизации параметров сервера можно оставить значение кнопки *Провести оптимизацию параметров* (рис. 1.8). Если же установить значение кнопки *Использовать параметры по умолчанию*, то можно будет использовать СУБД для целей обучения. В данном случае для размещения СУБД не требуется большой объем оперативной памяти.

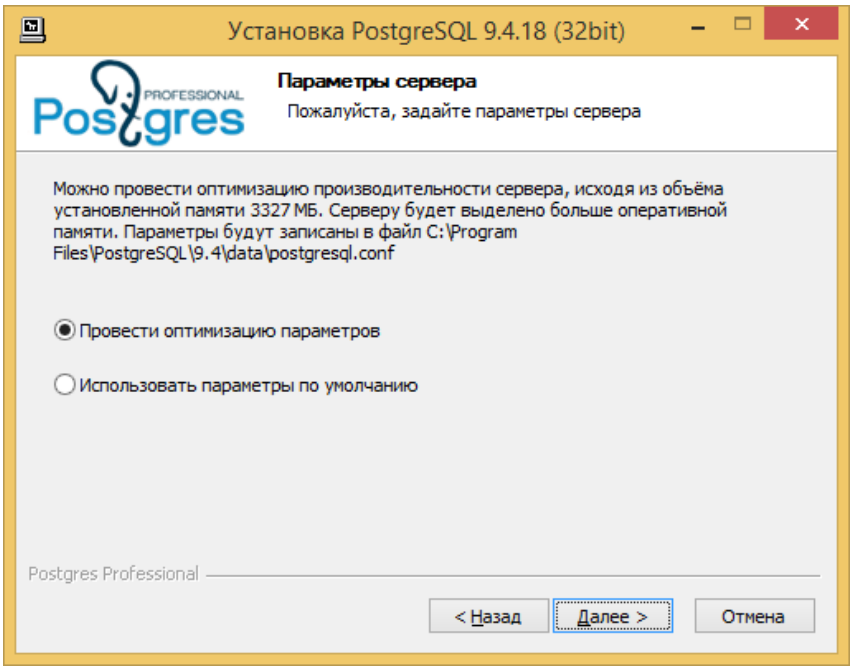


Рисунок 1.8 – Выбор параметра для оптимизации производительности сервера

На данном шаге установки надо выбрать папку в меню *Пуск* для размещения ярлыков программы, например PostgreSQL 9.4.18 (32bit) или задать другое имя папки. После выбора паки для размещения ярлыков нажать кнопку *Установить* (рис. 1.9).

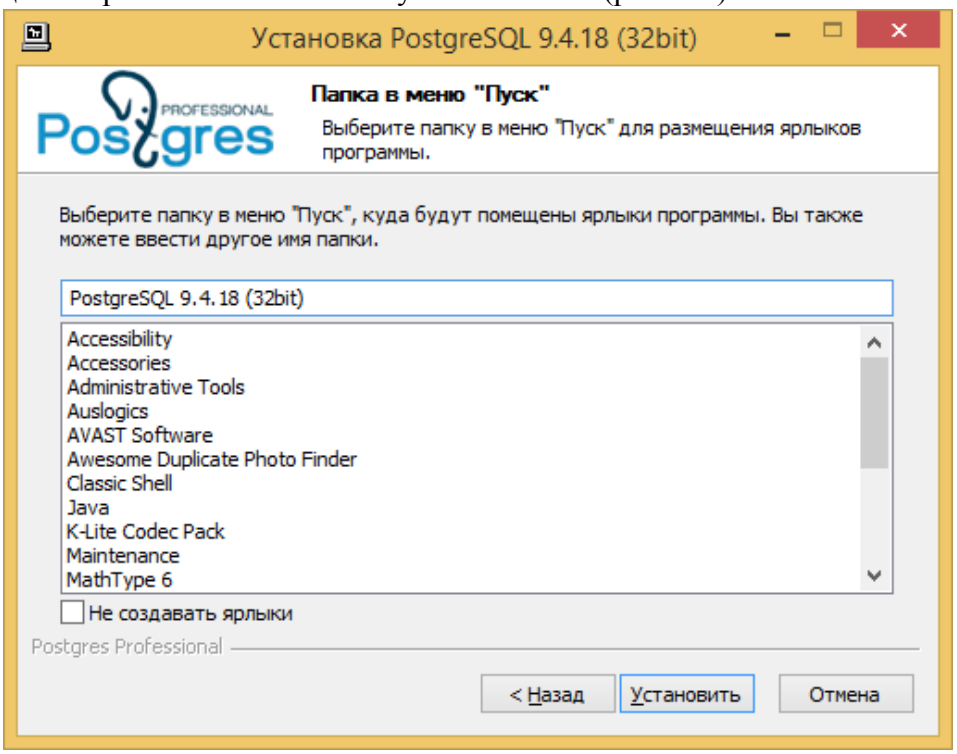


Рисунок 1.9 – Выбор папки PostgreSQL 9.4.18 (32bit)
для размещения ярлыков программы

В процессе установки идет копирование файлов в выбранную папку (рис.1.10).

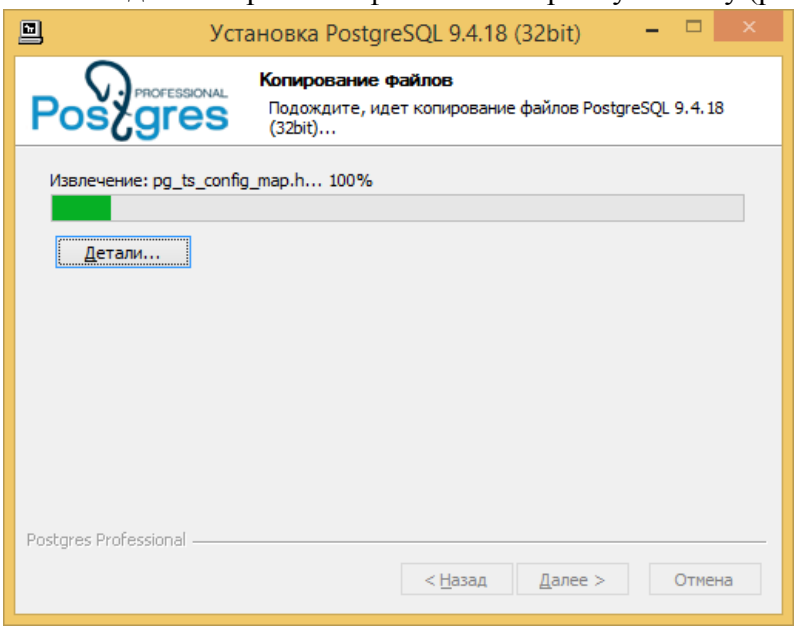


Рисунок 1.10 – Копирование файлов PostgreSQL 9.4.18 (32bit)

Завершение процесса установки сопровождается появлением окна, приведенного на рисунке 1.11, в котором надо нажать кнопку *Готово* для выхода из программы установки.

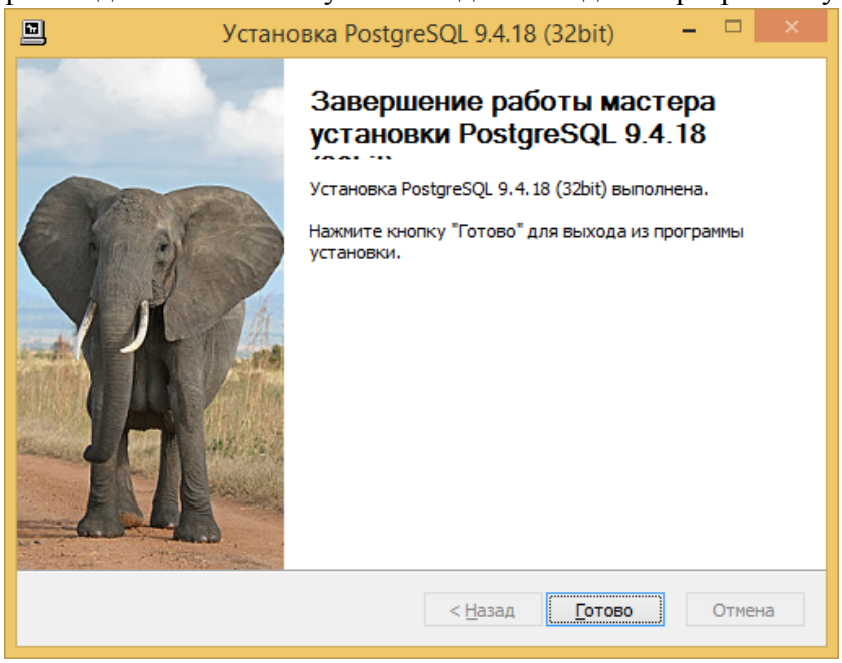


Рисунок 1.11 – Завершение работы мастера установки

После завершения процесса установки в папке PostgreSQL 9.4.18 (32 bit) будут установлены ярлыки программ для сервера, приведенные на рисунке 1.12.

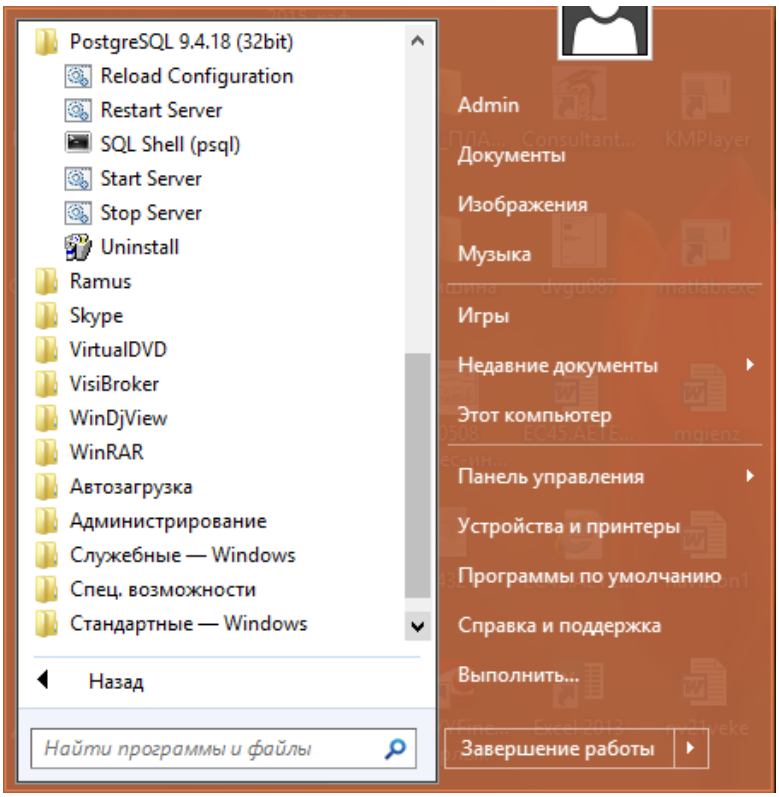


Рисунок 1.12 – Результат установки PostgreSQL

1.2. Компоненты среды СУБД PostgreSQL

Назначение установленных компонентов следующее:

– программа **Reload Configuration** позволяет осуществить перезагрузку конфигурации сервера из файлов конфигурации;

– **Restart Server** – предоставляет возможность выполнить останов и запуск сервера. Программа закрывает все активные соединения с сервером;

– **Start Server** – позволяет запустить сервер;

– **Stop Server** – позволяет выполнить останов сервера. В результате выполнения данной программы все активные соединения с сервером будут закрыты.

– программа **Uninstall** используется для удаления сервера: сервер останавливается, в результате чего все активные соединения с сервером закрываются. Однако, перезагрузка компьютера в данном случае не требуется, и папка с данными не удаляется;

– программа-клиент **SQL Shell (psql)** – это стандартный терминальный клиент, который требуется для работы с сервером СУБД. Psql предоставляет возможность подключиться к серверу СУБД и выполнять команды интерактивно в режиме командной строки. Клиент psql можно использовать для решения определенных задач по администрированию баз данных (БД) и написанию запросов. Терминальный клиент psql, обладая собственными командами, дает возможность работы с объектами БД и представлять информацию из таблиц БД в нужном для пользователя виде.

После установки PostgreSQL в папке C:\Program Files\PostgreSQL\9.4\data (каталог баз данных) будут находятся файлы конфигурации, которые определяют настройки сервера: основной файл postgresql.conf, содержащий значения параметров сервера и файл pg_hba.conf, определяющий настройки доступа. По умолчанию доступ подтверждается паролем и возможен только с локального компьютера.

Для работы с графическим пользовательским интерфейсом можно использовать программу **pgAdmin**.

Запуск программы SQL Shell (psql). Чтобы запустить программу *SQL Shell (psql)* необходимо щелкнуть по кнопке *Пуск*, выбрать пункт меню *Все программы*, выбрать папку *PostgreSQL 9.4.18* (*32 bit*) и запустить на выполнение программу (рис. 1.13).

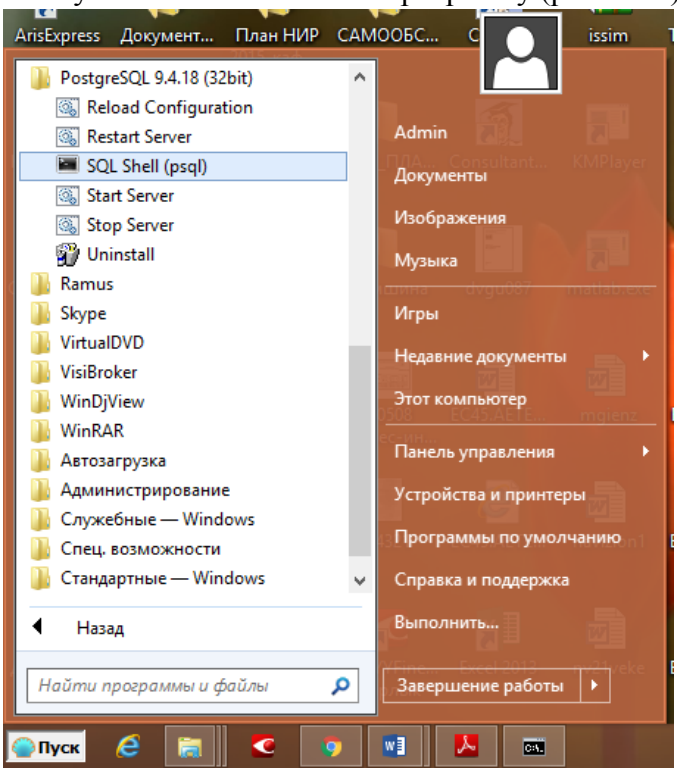


Рисунок 1.13 – Запуск программы *psql* с помощью кнопки *Пуск*

В результате запуска *psql* возникает окно «SQL Shell (psql)», в котором необходимо ввести пароль пользователя, указанный при установке PostgreSQL (рис. 1.14):

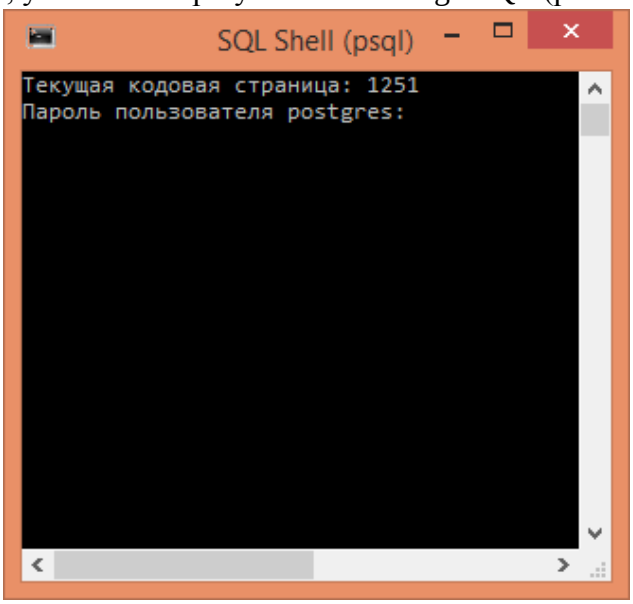


Рисунок 1.14 – Результат запуска *psql* и ввод пароля

После ввода пароля пользователя в окне «SQL Shell (psql)» будет выведено приглашение к работе «postgres=#», где «postgres» – это имя БД, к которой в данный момент подключен пользователь (рис. 1.15). Одновременно пользователь может работать только с одной базой данных, несмотря на то, что один сервер PostgreSQL может обслуживать несколько баз данных.

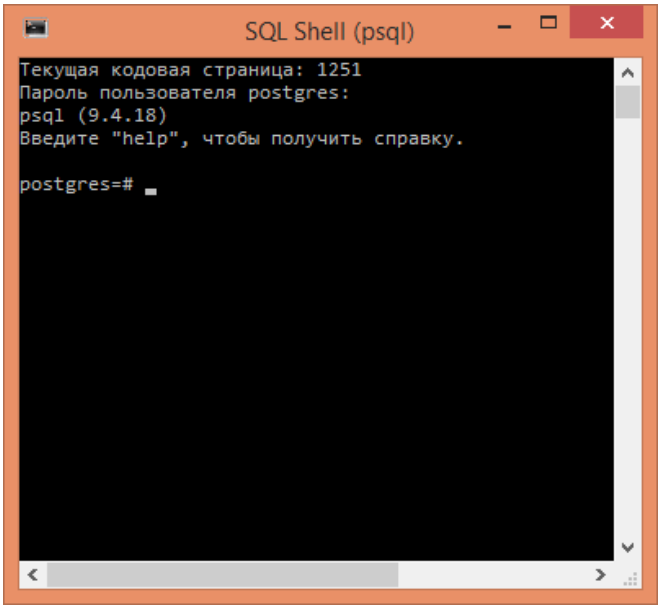


Рисунок 1.15 – Результат ввода пароля и вывод приглашения

При не корректном отображении на экране монитора букв кириллицы необходимо проверить установку шрифта формата TrueType в свойствах окна программы *SQL Shell* (*psql*) – *Consolas* или *Lucida Console* (рис. 1.16).

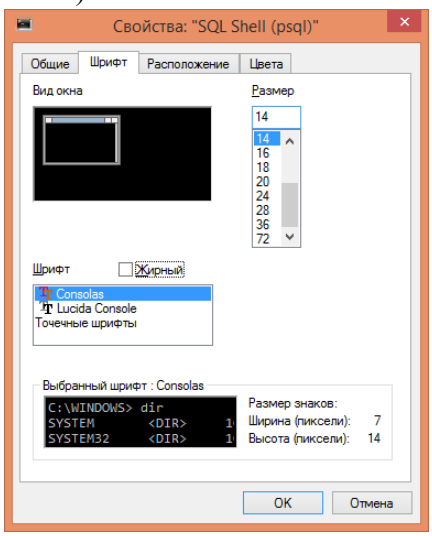


Рисунок 1.16 – Установка шрифта *Consolas*

Для поддержки кириллицы при запуске *psql* надо выполнить команду **chcp 1251**, которая позволит правильно отображать символы кириллицы.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Дайте определение СУБД.
2. Что такое *PostgreSQL*?
3. Каким образом можно выполнить установку *PostgreSQL*?
4. Опишите основные шаги установки *PostgreSQL*.
5. Для чего используется программа *Uninstall*?
6. Какие компоненты будут размещены в папке PostgreSQL 9.4.18 (32 bit) после установки *PostgreSQL*?
7. Для чего используется программа *SQL Shell (psql)* и какие возможности она предоставляет пользователю?

8. Какая команда позволяет правильно отображать символы кириллицы при работе с терминальным клиентом *SQL Shell* (*psql*)?

9. Каков вид приглашения к работе с программой *psql*?

ОСНОВЫ ЯЗЫКА PostgreSQL

Для создания баз данных, заполнения их непосредственно данными, выполнения запросов и т.п. используется язык структурированных запросов – SQL.

2.1. Создание базы данных

Для создания базы данных используется команда:

CREATE DATABASE <имя_базы_данных>;

***Пример* 2.1**. Создам базу данных с именем **dis_un** и подключимся к ней с помощью следующей команды:

\c dis_un;

Набор команд можно выполнять как строчными, так и прописными буквами. Результат выполнения команд приведен на рисунке 2.1.

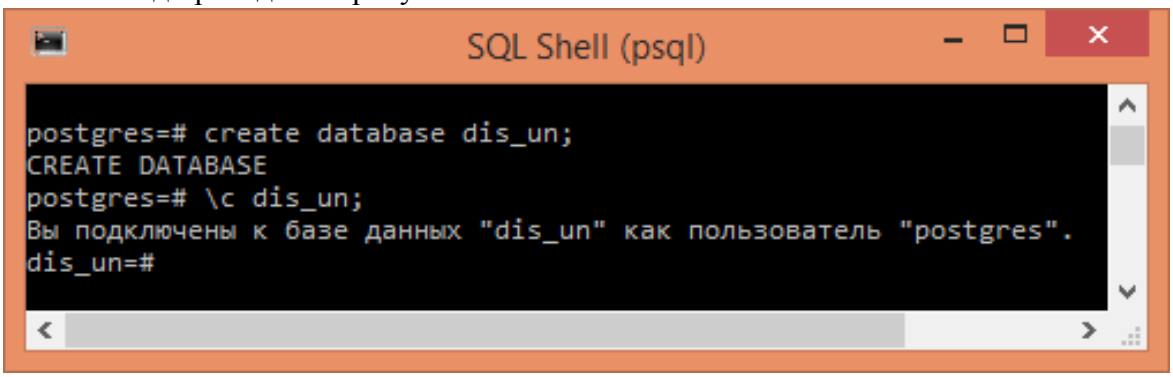


Рисунок 2.1 – Результат выполнения команд

Признаком конца команды является символ «**;**», что позволяет размещать команду на нескольких строках. Команды, начинающиеся с символа «\» – это команды psql, а не языка SQL.

После ввода команды SQL **create database dis_un;** и ее правильного выполнения выводятся сообщение «CREATE DATABASE» и приглашение к работе **postgres=#**. После выполнения команды **\c dis_un** выдается сообщение «Вы подключены к базе данных “dis_un” как пользователь “postgres”» и приглашение меняется на **dis_un=#** (рис. 2.1).

Для вывода полного списка команд программы psql используется сочетание символов «**?**» (рис.2.2).

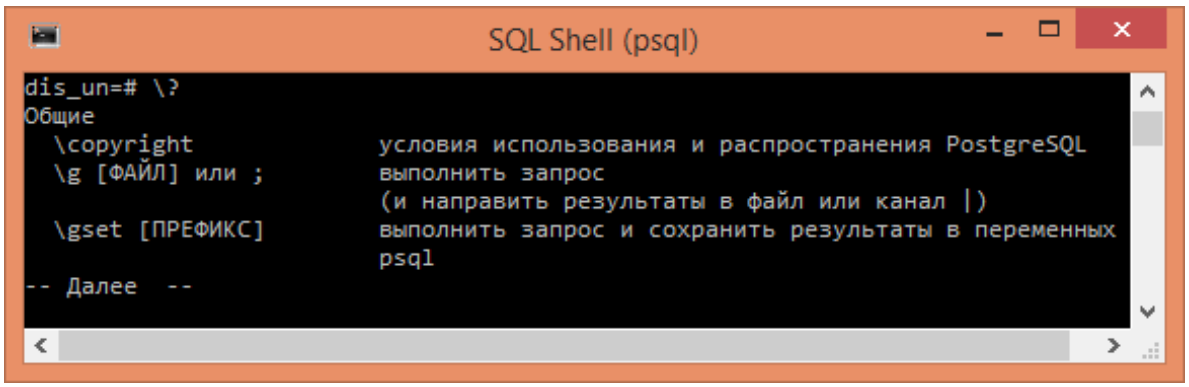


Рисунок 2.2. – Вывод списка команд программы psql

В таблице 2.1 приведен список часто используемых команд psql.

Таблица 2.1 – Список часто используемых команд psql

Команда	Назначение команды
Общие команды	
\h [ИМЯ]	Выдает справку по заданному SQL-оператору
\h *	Выдает справку по всем операторам
\q	Выход из psql
\?	Вывод справки по командам psql
Команды буфера запроса	
\p	Вывести содержимое буфера запросов
\r	Очистить буфер запросов
Информационные команды S – *показывать системные объекты*; + – *дополнительные подробности*	
\d[S+]	Показывать список таблиц, представлений и последовательностей
\d[S+] ИМЯ	Описание таблицы, представления, последовательности или индекса
\df[antw] [S+] [МАСКА]	Список [агрегатных/ нормальных/ триггерных/ оконных] функций
\di[S+] [МАСКА]	Список индексов
\dn[S+] [МАСКА]	Список схем
\dp [МАСКА]	Список прав доступа к таблицам, представлениям, последовательностям
\dt[S+] [МАСКА]	Список таблиц
\dT[S+] [МАСКА]	Список типов данных
\du[+] [МАСКА]	Список ролей
\l[+] [МАСКА]	Список баз данныхdx
\dv[S+] [МАСКА]	Список представлений
\dx[+] [МАСКА]	Список расширений

2.2. Типы данных

В СУБД PosgreSQL используется большое количество встроенных типов данных, а также существуют возможности создания новых типов.

К встроенным типам данных относятся: числовые; денежные; символьные; двоичные; типы даты/времени; логический тип; типы перечислений; геометрические типы; типы, описывающие сетевые адреса; типы, предназначенные для текстового поиска; тип XML; массивы; составные типы; диапазонные типы; псевдотипы и др.

Команда **CREATE TYPE** позволяет создавать новые типы данных. Владельцем типа становится пользователь, создавший тип.

Примечание. Подробнее с различными типами данных и синтаксисом команд ознакомиться можно в документации по PosgreSQL.

Для работы с созданной базой данных будут использованы следующие стандартные типы:

– числовой тип ***integer***, который представляет целые числа в диапазоне -2147483648…+2147483647. Данные представляются точно и занимают по 4 байта оперативной памяти (ОП);

– числовой тип ***real*** представляет вещественные числа с переменной точностью в пределах 6 десятичных цифр. Под каждое число отводится 4 байта ОП;

– логический тип ***boolean***, принимающий значения TRUE (истина), FALSE (ложь). Возможно состояние UNKNOWN, которое представляется SQL-значением NULL (значение не известно). Данное логического типа занимает 1 байт ОП.

Эти типы данных включены в стандарт SQL в отличие от символьного типа – ***text,*** который позволяет хранить строки переменный длины и поддерживается PosgreSQL и другими СУБД SQL.

2.3. Создание таблиц

В реляционных СУБД данные хранятся в таблицах. Каждый столбец в таблице имеет уникальное имя (заголовок) и относится к определенному типу данных. Значения полей в строках таблиц должны удовлетворять этим типам. Данные в таблицах не упорядочены, т.е. строки могут храниться в порядке не соответствующему порядку их добавления в таблицы.

Для создания таблицы базы данных используется следующая команда:

CREATE TABLE <имя_таблицы>;

Примечание. С полным синтаксисом всех команд SQL можно ознакомиться в источнике, а с помощью команды psql – **\h [ИМЯ_КОМАНДЫ]** – с конкретной командой (табл. 2.1).

***Пример* 2.1**. Создание таблицы, содержащей дисциплины для направления подготовки бакалавров «Бизнес-информатика».

Структура таблицы *disciplines* (*дисциплины*) приведена на рисунке 2.3.

Имя столбца (тип)	Имя столбца (тип)	Имя столбца (тип)	Имя столбца (тип)
d_num (text)	title (text)	hours (integer)	credit_units (integer)

Рисунок 2.3 – Структура таблицы *disciplines*

Таблица *disciplines* содержит четыре столбца:

d_num – идентификационный номер дисциплины (первичный ключ);

title – наименование дисциплины;

hours – часы, отводимые на изучение дисциплины;

credit_units – зачетные единицы по ECTS.

Решение. Создадим таблицу *disciplines*. Использование команды **CREATE TABLE** и результат ее работы приведен на рисунке 2.4. При вводе команды изменяется приглашение psql. Изменение приглашения означает, что ввод команды продолжается с новой строки. Каждый столбец поименован и определен его тип данных в соответствии со структурой (рис. 2.3), а также для столбца **d_num** определено ограничение целостности **PRIMARY KEY.** Это ограничение означает, что значения в столбце должны быть уникальными (не повторяющимися) и неопределенные значения не допускаются. Этот столбец используется для однозначной идентификации строк в таблице.

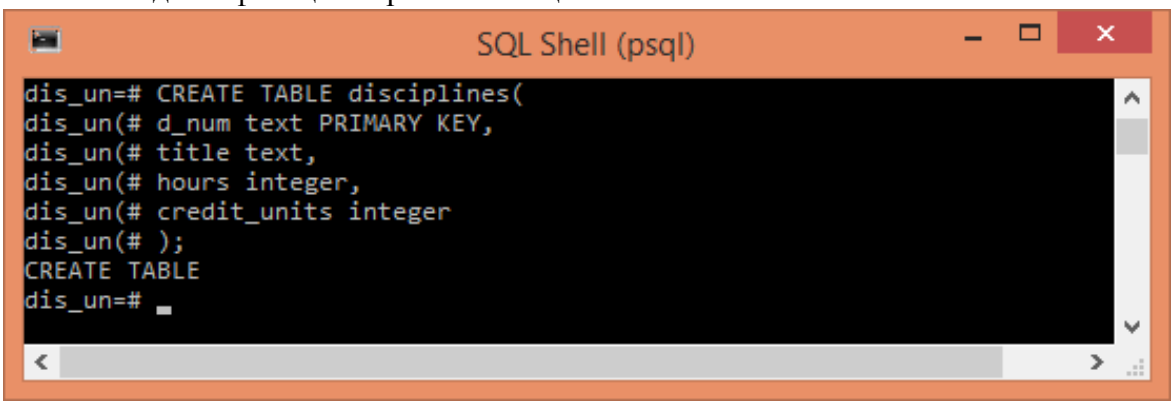


Рисунок 2.4 – Создание таблицы *disciplines*

***Пример* 2.2**. Создать таблицу *students* (*студенты*) и таблицу *examinations* (*экзамены*) в соответствии со структурами таблиц, приведенными на рисунках 2.5 и 2.6 соответственно.

Имя столбца (тип)	Имя столбца (тип)	Имя столбца (тип)	Имя столбца (тип)	Имя столбца (тип)
s_num (integer)	surname (text)	name (text)	year_bd (integer)	year_en (integer)

Рисунок 2.5 – Структура таблицы *students*

s_num (integer)	d_num (text)	marks (integer)	**sd_pk (составной тип)**

Рисунок 2.6 – Структура таблицы *examinations*

Таблица *students* представлена следующими столбцами: **s_num** – номер личного дела студента (первичный ключ); surname – фамилия студента; name – имя студента; year_bd – год рождения студента и year_en – год поступления.

Таблица *examinations* содержит поля: s_num – номер личного дела студента (внешний ключ); d_num – идентификационный номер дисциплины (внешний ключ) (внешний ключ показывает, что значения в одной таблице ссылаются на строки в другой таблице); marks – оценка по дисциплине; **sd_pk –** составной первичный ключ для идентификации оценок студентов по определенным дисциплинам.

Решение. Аналогично создаются таблицы *students* и *examinations* с помощью команд:

CREATE TABLE students
(s_num integer **PRIMARY KEY,**
surname text,
name text,

year_bd integer,
year_en integer);

CREATE TABLE examinations
(s_num integer **REFERENCES** students (s_num),
d_num text **REFERENCES** disciplines (d_num),
marks integer,
CONSTRAINT sd_pk **PRIMARY KEY** (s_num, d_num)
);

С использованием **REFERENCES** определены внешние ключи, а с помощью **CONSTRAINT** каждая запись в таблице *examinations* будет идентифицирована составным ключом, зависящим от s_num и d_num.

Результат выполнения команд представлен на рис. 2.7.

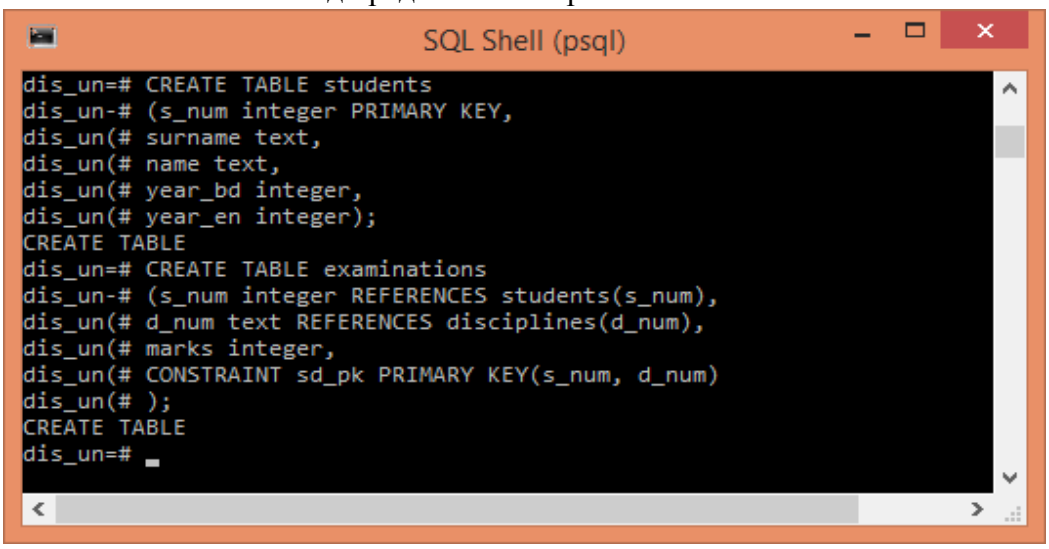


Рисунок 2.7 – Результат создания таблиц *students* и *examinations*

2.3. Добавление строк в таблицы

Для добавления строк в таблицу используется команда **INSERT.**

Пример 2.3. Осуществить ввод исходных данных в таблицы *disciplines*, *students* и *examinations.* Исходные данные приведены в таблицах 2.2, 2.3 и 2.4 соответственно.

Таблица 2.2 – Исходные данные для таблицы *disciplines*

d_num	title	hours	credit_units
Б.1.1.3.7	Архитектура организации	180	5
Б1.1.3.1	Базы данных	216	6
Б1.1.1.7	Введение в специальность	36	1
Б1.1.3.6	Инжиниринг бизнеса	288	8
Б1.1.3.8	ИТ-стандарты	108	3
Б1.1.2.4	Математика	360	10

Таблица 2.3 – Исходные данные для таблицы *students*

s_num	surname	name	year_bd	year_en
1959	Зайцев	Вадим	1997	2017

1936	Михайлов	Павел	1999	2017
2011	Романова	Алина	2000	2018

Таблица 2.4 – Исходные данные для таблицы *examinations*

s_num	d_num	marks
1959	Б1.1.1.7	5
2011	Б1.1.1.7	5
1959	Б1.1.2.4	5
1936	Б1.1.2.4	4
1959	Б1.1.3.8	4
1936	Б1.1.3.8	3
1959	Б1.1.3.1	5
1936	Б1.1.3.1	5
2011	Б1.1.3.7	4
2011	Б1.1.3.6	4

Решение. Исходные данные (табл. 2.2) в таблицу *disciplines* будем вводить по строкам, используя команду:

INSERT INTO disciplines (d_num, title, hours, credit_units)
VALUES (‘Б1.1.3.7’, ‘Архитектура организации’, 180, 5),
(‘Б1.1.3.1’, ‘Базы данных’, 216, 6),
(‘Б1.1.1.7’, ‘Введение в специальность’, 36, 1),
(‘Б1.1.3.6’, ‘Инжиниринг бизнеса’, 288, 8),
(‘Б1.1.3.8’, ‘ИТ-стандарты’, 108, 3),
(‘Б1.1.2.4’, ‘Математика’, 360, 10);

Результат выполнения команды INSERT приведен на рисунке 2.8.

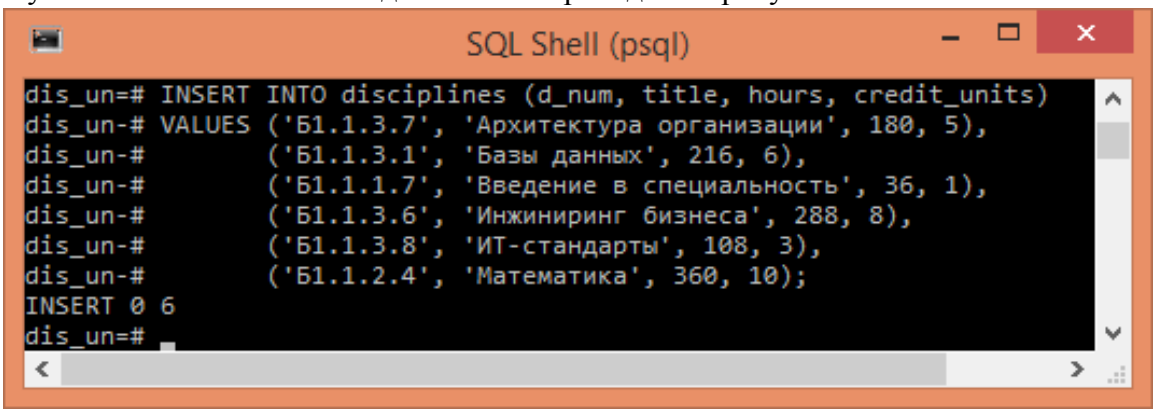


Рисунок 2.8 – Результат выполнения команды INSERT

Если загружаются данные из внешнего файла, то используется команда **COPY**, формат которой приведен в источнике.

Аналогично осуществим ввод исходных данных в таблицы *students* и *examinations* с помощью следующих команд:

INSERT INTO students (s_num, surname, name, year_bd, year_en)
VALUES (1959, ‘Зайцев’, ‘Вадим’, 1997, 2017),
(1936, ‘Михайлов’, ‘Павел’, 1999, 2017),
(2011, ‘Романова’, ‘Алина’, 2000, 2018);

INSERT INTO examinations (s_num, d_num, marks)
VALUES (1959, ‘Б1.1.1.7’, 5),
(2011, ‘Б1.1.1.7’, 5),
(1959, ‘Б1.1.2.4’, 5),
(1936, ‘Б1.1.2.4’, 4),
(1959, ‘Б1.1.3.8’, 4),
(1936, ‘Б1.1.3.8’, 3),
(1959, ‘Б1.1.3.1’, 5),
(1936, ‘Б1.1.3.1’, 5),
(2011, ‘Б1.1.3.7’, 4),
(2011, ‘Б1.1.3.6’, 4);
Результат ввода исходных данных в таблицу *examinations* представлен на рисунке 2.9.

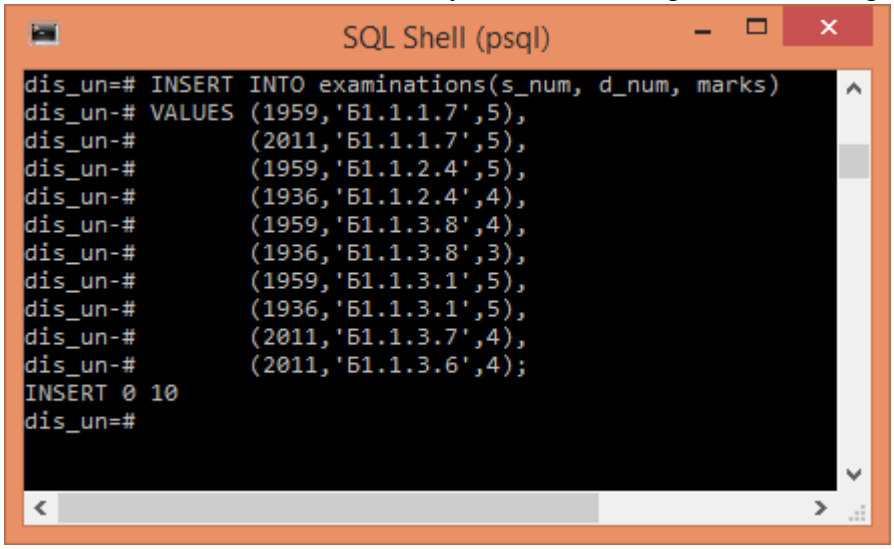


Рисунок 2.9 – Результат ввода исходных данных
в таблицу *examinations*

2.4 Запросы на выборку данных

Выборка данных из таблиц осуществляется оператором **SELECT**. Список оператора **SELECT** (между ключевыми словами **SELECT** и **FROM**) содержит выражения, формирующие выходные строки оператора. Выражения, как правило, обращаются к столбцам (колонкам), вычисленным в предложении **FROM.**

Каждая выходная колонка **SELECT** имеет имя. Чтобы задать имя выходной колонки необходимо записать **AS <выходное_имя>** после выражения колонки, в противном случае PostgreSQL выберет имя автоматически. К значению выходной колонки можно обратиться по имени колонки в предложениях **GROUP BY** и **ORDER BY**. При использовании предложений **HAVING** или **WHERE** вместо имени надо записывать все выражение.

В выходном списке можно использовать символ «*». Тогда будут выведены все столбцы выбранных строк.

Примечание. В последующих примерах ввод операторов SQL и результаты их работы в среде psql будут представлены без окна *SQL Shell (psql)* с использованием другой гарнитуры шрифта для лучшей визуализации рассматриваемого материала.

***Пример* 2.4**. Осуществить вывод всех столбцов из таблицы *disciplines.*

Решение. Для вывода столбцов используем оператор **SELECT**:

```
dis_un=# SELECT * FROM disciplines;
```

```
 d_num   |          title          | hours | credit_units
---------+-------------------------+-------+--------------
 Б1.1.3.7 | Архитектура организации  |   180 |            5
 Б1.1.3.1 | Базы данных              |   216 |            6
 Б1.1.1.7 | Введение в специальность |    36 |            1
 Б1.1.3.6 | Инжиниринг бизнеса       |   288 |            8
 Б1.1.3.8 | ИТ-стандарты             |   108 |            3
 Б1.1.2.4 | Математика               |   360 |           10
(6 строк)
```

***Пример* 2.5**. Осуществите вывод столбцов *title* и *credit_units* из таблицы *disciplines.* Столбец *credit_units* переименуйте.

Решение. Дадим столбцу *credit_units* новое имя – *units*, используя конструкцию **AS**:

```
dis_un=# SELECT title, credit_units AS units
dis_un-# FROM disciplines;
         title          | units
------------------------+-------
 Архитектура организации |     5
 Базы данных             |     6
 Введение в специальность |     1
 Инжиниринг бизнеса      |     8
 ИТ-стандарты            |     3
 Математика              |    10
(6 строк)
```

***Пример* 2.6**. Получить выборку по годам поступления студентов. В полученной выборке года поступления не должны повторяться.

Решение. В результирующей выборке могут появиться одинаковые строки. Это связано с тем, что выводятся не все столбцы таблицы *students*. Например:

```
dis_un=# SELECT year_en FROM students;
 year_en
---------
    2017
    2017
    2018
(3 строки)
```

Для устранения повторяющихся строк используется предложение **DISTINCT** после **SELECT.**

В результате выполнения запроса все повторяющиеся строки исключаются из результирующего набора, то есть из каждой группы дубликатов остается только одна строка:

```
dis_un=# SELECT DISTINCT year_en FROM students;
 year_en
---------
    2017
    2018
```

(2 строки)

Использование **SELECT ALL** позволит сохранить все строки (значение действует по умолчанию).

***Пример* 2.7**. Вычислите следующие выражения: а) 2*7; б) 2^5.

Решение. Если опустить предложение **FROM**, то можно вычислять простые выражения: а) **SELECT** 7*2 **AS** res; б) **SELECT** 2^5 **AS** res;

Тогда получим следующие результаты вычисления выражений:

```
dis_un=# SELECT 7*2 AS res;
 res
-----
  14
(1 строка)

dis_un=# SELECT 2^5 AS res;
 res
-----
  32
(1 строка)
```

Примечание. Если предложение **FROM** отсутствует, то запрос не может обращаться к таблицам БД.

***Пример* 2.8**. Выбрать из таблицы *disciplines* те дисциплины, у которых количество часов на изучение больше или равно 180.

Решение. Для выполнения данного условия используется предложение **WHERE.** Форма предложения **WHERE** следующая:

WHERE <условие>,

где **<условие> -** произвольное выражение, результат которого имеет тип boolean.

Строка удовлетворяет условию, если оно возвращает значение true. Строки, не удовлетворяющие выражению **<условие>**, исключаются из результата.

В выражении **<условие>** могут применяться операторы отношения (сравнения):

= (равно, проверяет равенство);

!= или <> (не равно, проверяет неравенство);

> (больше);

< (меньше);

>= (больше или равно);

<= (меньше или равно), а также использоваться логические операторы **NOT**, **AND** и **OR** и () (круглые скобки) для объединения результатов нескольких условий.

Приоритеты выполнения операторов приведены в приложении 1. Таблица истинности логических операций приведена в приложении 2.

Тогда используем следующее условие фильтрации:

SELECT * **FROM** disciplines **WHERE** hours >= 180;

В результате получим:

```
dis_un=# SELECT * FROM disciplines WHERE hours >=180;
  d_num   |          title          | hours | credit_units
----------+-------------------------+-------+--------------
 Б1.1.3.7 | Архитектура организации |   180 |            5
 Б1.1.3.1 | Базы данных             |   216 |            6
 Б1.1.3.6 | Инжиниринг бизнеса      |   288 |            8
 Б1.1.2.4 | Математика              |   360 |           10
(4 строки)
```

Примечание. Определенность значения (содержит значение NULL или не содержит) проверяется с помощью конструкций типа:

<выражение> IS NULL
<выражение> IS NOT NULL (приложение 1)

или не стандартных, но равнозначных конструкций:

<выражение> ISNULL
<выражение> NOTNULL.

Также могут быть использованы конструкции:

<выражение> IS DISTINCT FROM <выражение>
<выражение> IS NOT DISTINCT FROM <выражение>

и функция **coalesce**, которая возвращает первый аргумент, отличный от NULL. При равенстве всех аргументов NULL результатом также будет NULL.

Логические значения можно проверить с помощью следующих условий:

<выражение> IS TRUE
<выражение> IS NOT TRUE
<выражение> IS FALSE
<выражение> IS NOT FALSE
<выражение> IS UNKNOWN
<выражение> IS NOT UNKNOWN.

***Пример* 2.9**. Выбрать из таблицы *disciplines* те дисциплины, у которых количество часов на изучение больше или равно 180, но меньше 360.

Решение. Условие фильтрации примет вид: hours >= 180 AND hours < 360. Тогда получим следующий результат:

```
dis_un=# SELECT * FROM disciplines WHERE hours >= 180 AND hours <360;
  d_num   |          title          | hours | credit_units
----------+-------------------------+-------+--------------
 Б1.1.3.7 | Архитектура организации |   180 |            5
 Б1.1.3.1 | Базы данных             |   216 |            6
 Б1.1.3.6 | Инжиниринг бизнеса      |   288 |            8
```

(3 строки)

***Пример* 2.10**. Выбрать из таблицы *disciplines* те дисциплины, у которых количество часов на изучение больше или равно 216, но меньше или равно 360.

Решение. Можно вместо операторов сравнения использовать специальную конструкцию **BETWEEN**, которая показывает, что границы интервала также включаются в интервал:

hours **BETWEEN** 216 **AND** 360

Приведенная выше конструкция **BETWEEN** равнозначна выражению:

hours >= 216 **AND** hours <= 360

В результате выполнения запроса будет получен следующий результат:

```
dis_un=# SELECT * FROM disciplines WHERE hours BETWEEN 216 AND 360;
  d_num   |       title        | hours | credit_units
----------+--------------------+-------+--------------
 Б1.1.3.1 | Базы данных        |   216 |            6
 Б1.1.3.6 | Инжиниринг бизнеса |   288 |            8
 Б1.1.2.4 | Математика         |   360 |           10
(3 строки)
```

***Пример* 2.11**. Выбрать из таблицы *disciplines* те дисциплины, у которых количество часов на изучение меньше 216, но больше 288.

Решение. Можно вместо операторов сравнения использовать специальную конструкцию **NOT BETWEEN**:

hours **NOT BETWEEN** 216 **AND** 288

Границы интервала не включаются в интервал. Конструкция **NOT BETWEEN** равнозначна выражению:

hours < 216 **OR** hours > 288.

В результате выполнения запроса будет получен следующий результат:

```
dis_un=# SELECT * FROM disciplines WHERE hours NOT BETWEEN 216 AND 288;
  d_num   |          title           | hours | credit_units
----------+--------------------------+-------+--------------
 Б1.1.3.7 | Архитектура организации  |   180 |            5
 Б1.1.1.7 | Введение в специальность |    36 |            1
 Б1.1.3.8 | ИТ-стандарты             |   108 |            3
 Б1.1.2.4 | Математика               |   360 |           10
(4 строки)
```

2.5 Соединения в запросах

Правильно спроектированная БД не должна содержать избыточных данных. Поэтому для получения всех требуемых данных необходимо в запросе соединять данные из разных таблиц.

***Пример* 2.12**. Получить все данные из таблиц *disciplines* и examinations.

Решение. Для получения необходимой информации надо перечислить имена таблиц в предложении **FROM** команды **SELECT**:

SELECT * **FROM** disciplines, examinations;

Результат выполнения запроса приведен в приложении 3.

Таблицы *disciplines* и *examinations* являются источниками данных для **SELECT**. Результатом является *декартово произведение* всех строк таблиц. К каждой строке таблицы *disciplines* добавляется каждая строка таблицы *examinations*.

Как правило в запрос добавляются уточняющие условия (условие соединения) в предложении **WHERE**, ограничивающие набор строк подмножеством этого произведения.

***Пример* 2.13**. Вывести оценки по всем дисциплинам, сопоставляя дисциплины с теми экзаменами, которые проведены по данной дисциплине.

Решение. Введем команду и получим результат:

```
dis_un=# SELECT disciplines.title, examinations.s_num, examinations.marks
dis_un-# FROM disciplines, examinations
dis_un-# WHERE disciplines.d_num = examinations.d_num;
         title          | s_num | marks
------------------------+-------+-------
 Введение в специальность |  1959 |     5
 Введение в специальность |  2011 |     5
 Математика               |  1959 |     5
 Математика               |  1936 |     4
 ИТ-стандарты             |  1959 |     4
 ИТ-стандарты             |  1936 |     3
 Базы данных              |  1959 |     5
 Базы данных              |  1936 |     5
 Архитектура организации  |  2011 |     4
 Инжиниринг бизнеса       |  2011 |     4
(10 строк)
```

***Пример* 2.14**. Вывести фамилии студентов и их экзаменационные оценки по дисциплине *Базы данных*. Сформировать запрос, используя соединение с предложением **JOIN**.

Решение. Сформируем следующий запрос и получим результат:

```
dis_un=# SELECT students.surname, examinations.marks
dis_un-# FROM students
dis_un-# JOIN examinations
dis_un-# ON students.s_num = examinations.s_num
dis_un-# AND examinations.d_num ='Б1.1.3.1';
 surname  | marks
----------+-------
 Зайцев   |     5
```

```
 Михайлов |     5
(2 строки)
```

Из данного результата следует, что в него не вошли строки исходной таблицы, для которых не нашлось пары в другой таблице, несмотря на то, что условие наложено на дисциплины, из результата исключаются студенты, не сдававшие экзамен.

Для того, чтобы в выборку вошли все студенты, используется операция *внешнего соединения*, которая будет рассмотрена в примере 2.15. Тогда перед оператором **JOIN** указывается одно из ключевых слов, например, **LEFT** или **RIGHT**, которые определяют тип соединения. Перед оператором **JOIN** может стоять ключевое слово **OUTER**, хотя употребление его необязательно. После оператора **JOIN** указывается присоединяемая таблица, а после оператора **ON** следует условие соединения. Например, **LEFT [OUTER] JOIN** – выборка будет содержать все строки из первой или левой таблицы; **RIGHT [OUTER] JOIN** – выборка будет содержать все строки из второй или правой таблицы.

Кроме рассмотренных вариантов соединений возможны также следующие (в [] приведены необязательные параметры):

[INNER] JOIN
FULL [OUTER] JOIN
CROSS JOIN

Типы соединений **INNER** и **OUTER** требуют указания условия соединения – одного из предложений **NATURAL**, **ON <условие соединения>** или **USING (колонка_соединения [, …]).** Для соединения **CROSS JOIN** эти предложения не допустимы.

Предложения **CROSS JOIN** и **INNER JOIN** формирует простое декартово произведение. Предложение **CROSS JOIN** равнозначно предложению **INNER JOIN ON (TRUE)**, никакие строки по условию не удаляются.

FULL OUTER JOIN – возвращает все соединенные строки, а также все строки слева, не имеющие соответствия справа и все строки справа, не имеющие соответствия слева (эти строки дополнены значениями NULL вправо и влево).

***Пример* 2.15**. Вывести фамилии студентов, которые сдали экзамен по дисциплине *Базы данных*, и их экзаменационные оценки. В список также должны войти студенты, которые не сдавали экзамен.

```
dis_un=# SELECT students.surname, examinations.marks
dis_un-# FROM students
dis_un-# LEFT JOIN examinations
dis_un-# ON students.s_num = examinations.s_num
dis_un-# AND examinations.d_num = 'Б1.1.3.1';
 surname  | marks
----------+-------
 Зайцев   |     5
 Михайлов |     5
 Романова |
(3 строки)
```

Как видно в результат добавлены строки из левой таблицы, для которых не нашлось пары в правой таблице. Для столбцов правой таблицы возвращаются неопределенные значения (NULL).

<условие> в предложении **WHERE** применяется к уже готовому результату соединений, и если вынести ограничение на дисциплины из условия соединения, то Романова не попадет в выборку, так как для этой студентки не определен examinations.d_num:

```
dis_un=# SELECT students.surname, examinations.marks
dis_un-# FROM students
dis_un-# LEFT JOIN examinations
dis_un-# ON students.s_num = examinations.s_num
dis_un-# WHERE examinations.d_num ='Б1.1.3.1';
 surname  | marks
----------+-------
 Зайцев   |     5
 Михайлов |     5
(2 строки)
```

2.6 Формирование подзапросов

Оператор **SELECT** формирует таблицу, которая выводится в качестве результата. Данная таблица может использоваться в другой конструкции языка SQL в любом месте, где по смыслу может находиться таблица. Такая вложенная команда **SELECT**, заключенная в круглые скобки, называется *подзапросом*.

Вложенный подзапрос представляет собой подзапрос, который заключается в круглые скобки и вкладывается в фразу **WHERE** или **HAVING** предложения **SELECT**. В свою очередь вложенный подзапрос может содержать в своей фразе **WHERE (HAVING)** другой вложенный подзапрос и т.д. Создается вложенный подзапрос с целью отбора строк таблицы, сформированного основным запросом, а также чтобы можно было использовать данные из других таблиц.

Существуют простые и коррелированные вложенные подзапросы, которые включаются во фразу **WHERE (HAVING)** с помощью условий **IN**, **EXISTS** или одного из условий сравнения (=, <>, <, <=, >, >=). Обработка простых вложенных подзапросов происходит «снизу вверх», то есть первым обрабатывается вложенный подзапрос самого нижнего уровня вложения. Результат его выполнения используется подзапросом более высокого уровня и т.д. В отличие от простых вложенных подзапросов коррелированные обрабатываются в обратном порядке: в первую очередь выбирается первая строка таблицы, сформированной основным запросом, из нее выбираются значения тех столбцов, которые используются во вложенном подзапросе (подзапросах). После этого выбирается вторая строка и т.д., пока в результат не будут включены все строки, удовлетворяющие вложенному подзапросу или последовательности вложенных подзапросов.

***Пример* 2.16.** *Формирование подзапроса*. Если подзапрос возвращает одну строку или столбец, то его можно использовать как скалярное выражение:

```
dis_un=# SELECT surname,
dis_un-# (SELECT marks
dis_un(# FROM examinations
dis_un(# WHERE examinations.s_num = students.s_num
```

```
dis_un(# AND examinations.d_num = 'Б1.1.3.1')
dis_un-# FROM students;
 surname  | marks
----------+-------
 Зайцев   |     5
 Михайлов |     5
 Романова |
(3 строки)
```

Подзапрос, использованный в списке выражений **SELECT**, может не содержать ни одной строки, тогда возвращается неопределенное значение (см. последнею строку результата: Романова).

***Пример* 2.17**. *Формирование скалярного подзапроса в условиях фильтрации*: получить результаты экзаменов, которые сдали студенты, поступившие после 2017 года.

Решение.

```
dis_un=# SELECT *
dis_un-# FROM examinations
dis_un-# WHERE (SELECT year_en
dis_un(#      FROM students
dis_un(#      WHERE students.s_num = examinations.s_num) > 2017;
 s_num |  d_num   | marks
-------+----------+-------
  2011 | Б1.1.1.7 |     5
  2011 | Б1.1.3.7 |     4
  2011 | Б1.1.3.6 |     4
(3 строки)
```

***Пример* 2.18**. *Формирование подзапроса с использованием конструкции IN*. Вывести фамилии студентов, получивших оценки по дисциплине *ИТ-стандарты*.

Решение. Общий вид конструкции **IN** следующий:

<выражение> IN (<подзапрос>).

Выражение круглых скобках **<подзапрос>** должно возвращать ровно один столбец. Вычисленное значение выражения **<выражение>** сравнивается со значениями во всех строках, возвращаемых подзапросом. Результатом конструкции **IN** будет **TRUE**, если строка с таким значением находится и **FALSE** – в противном случае. Если результатом выражения **<выражение>** является **NULL** или равных значений справа не найдено, а хотя бы одно из значений справа равно **NULL**, конструкция **IN** возвращает **NULL**, а не **FALSE**.

Сформируем подзапрос и получим следующий результат:

```
dis_un=# SELECT surname, year_en, year_bd
dis_un-# FROM students
dis_un-# WHERE s_num IN (SELECT s_num
dis_un(#                 FROM examinations
dis_un(#                 WHERE d_num = 'Б1.1.3.8');
 surname  | year_en | year_bd
```

```
----------+---------+---------
 Зайцев   |    2017 |    1997
 Михайлов |    2017 |    1999
(2 строки)
```

***Пример* 2.19**. *Формирование подзапроса с использованием конструкции NOT IN.* Вывести фамилии студентов, не получивших оценки по дисциплине *ИТ-стандарты*.

Решение. Конструкция **NOT IN** возвращает противоположный результат:

```
dis_un=# SELECT surname, year_en, year_bd
dis_un-# FROM students
dis_un-# WHERE s_num NOT IN (SELECT s_num
dis_un(#                     FROM examinations
dis_un(#                     WHERE d_num = 'Б1.1.3.8');
 surname  | year_en | year_bd
----------+---------+---------
 Романова |    2018 |    2000
(1 строка)
```

***Пример* 2.20**. *Формирование подзапроса с использованием конструкции NOT IN.* Вывести список студентов, получивших оценки «4» и «5».

Решение.

```
dis_un=# SELECT surname, name, year_en
dis_un-# FROM students
dis_un-# WHERE s_num NOT IN (SELECT s_num
dis_un(#                     FROM examinations
dis_un(#                     WHERE marks < 4);
 surname  | name  | year_en
----------+-------+---------
 Зайцев   | Вадим |    2017
 Романова | Алина |    2018
(2 строки)
```

***Пример* 2.21**. *Формирование подзапроса с использованием предиката EXISTS.* Вывести список студентов, получивших оценки «4» и «5».

Решение. Формат использования **EXISTS** следующий:

EXISTS (<подзапрос>),

где **<подзапрос>** представлен оператором **SELECT**.

В результате выполнения запроса проверяется возвращаются ли строки. Если возвращается хотя бы одна строка, то результатом **EXISTS** будет **TRUE**, а если не возвращается ни одной строки, то – **FALSE**.

Аналогичный результат (см. пример 2.20) можно получить с помощью **EXISTS**.

```
dis_un=# SELECT surname, name, year_en
dis_un-# FROM students
dis_un-# WHERE NOT EXISTS (SELECT s_num
```

```
dis_un(#                FROM examinations
dis_un(#               WHERE examinations.s_num = students.s_num
dis_un(#             AND marks < 4);
 surname  | name  | year_en
----------+-------+---------
 Зайцев   | Вадим |    2017
 Романова | Алина |    2018
(2 строки)
```

2.7 Использование псевдонимов

Таблицам и ссылкам на них в запросе можно давать временное имя (псевдоним). Тогда существует возможность обращения к ним с помощью *псевдонима* в запросе.

Для определения псевдонима могут использоваться следующие конструкции:

FROM <табличная_ссылка> AS <псевдоним>

FROM <табличная_ссылка> <псевдоним>

Примечание. Псевдоним становится новым именем в текущем запросе и после его назначения исходное имя таблицы нельзя использовать в другом месте запроса.

Псевдонимы необходимо назначать подзапросам.

При назначении псевдонимов столбцам используется следующая конструкция:

FROM <табличная_ссылка> [AS] <псевдоним>
(<столбец1> [, <столбец2> [, …])

***Пример* 2.22**. Вывести список студентов и их оценки по дисциплине *Архитектура организации*, используя псевдонимы таблицы и подзапроса.

Решение. Назначим таблице *students* псевдоним *st*, а подзапросу псевдоним – *pq*:

```
dis_un=# SELECT st.surname, st.name, pq.marks
dis_un-# FROM students st
dis_un-# JOIN (SELECT examinations.*
dis_un(#       FROM disciplines, examinations
dis_un(#       WHERE disciplines.d_num = examinations.d_num
dis_un(#      AND disciplines.title = 'Архитектура организации') pq
dis_un-# ON st.s_num = pq.s_num;
 surname  | name  | marks
----------+-------+-------
 Романова | Алина |     4
(1 строка)
```

***Пример* 2.23**. Используя условие примера 2.22 сформировать запрос, не формируя подзапрос.

Решение. Назначим таблицам *disciplines*, *students* и *examinations* псевдонимы *d*, *s* и *e* соответственно, и выполним следующий запрос:

```
dis_un=# SELECT s.surname, s.name, e.marks
dis_un-# FROM students s, disciplines d, examinations e
dis_un-# WHERE d.d_num = e.d_num
dis_un-# AND d.title ='Архитектура организации'
dis_un-# AND s.s_num = e.s_num;
 surname  | name  | marks
----------+-------+-------
 Романова | Алина |     4
(1 строка)
```

2.8 Сортировка

Использование предложения **ORDER BY** позволяет улучшить представление полученных результатов запроса, так как данные в таблицах не упорядочены. Это предложение со списком выражений определяет порядок выполнения сортировки. После каждого выражения – *ключа сортировки* указывается направление (порядок) сортировки.

Общая форма предложения **ORDER BY** в операторе **SELECT** имеет следующий вид:

```
SELECT <список_выборки>
FROM < табличное_выражение>
ORDER BY выражение_сортировки1 [ASC | DESC] [NULLS {FIRST | LAST}]
  [, выражение_сортировки2 [ASC | DESC] [NULLS {FIRST | LAST}] …]
```

Выражение сортировки – любое выражение, допустимое в списке выборки запроса. Например, это может быть метка столбца или номер выводимого столбца.

При указании нескольких выражений, последующие значения позволяют сортировать строки, в которых совпали все предыдущие значения. Каждое выражение сортировки можно дополнить ключевыми словами:

ASC – сортировка данных по возрастанию (принята по умолчанию);

DESC – сортировка данных по убыванию.

Указания **NULLS FIRST** и **NULL LAST** используются для определения места значений **NULL**:

NULLS FIRST помещает значения **NULL** до значений не **NULL**;

NULL LAST помещает значения **NULL** после значений не **NULL**.

Примечание. По умолчанию значения **NULL** считаются больше любых других значений, поэтому подразумевается **NULLS FIRST** для порядка **DESC** и **NULL LAST** в противном случае.

***Пример* 2.24**. Отсортировать строки сначала по возрастанию оценки, для совпадающих оценок по дисциплинам – по номеру личного дела студента, а при совпадении первых двух ключей по убыванию номера дисциплины.

Решение. Для выполнения сортировки используем оператор SELECT и ключи сортировки – *marks*, *s_num* и *d_num*:

```
dis_un=# SELECT * FROM examinations
dis_un-# ORDER BY marks, s_num, d_num DESC;
 s_num |  d_num   | marks
-------+----------+-------
  1936 | Б1.1.3.8 |     3
  1936 | Б1.1.2.4 |     4
```

```
 1959 | Б1.1.3.8 |     4
 2011 | Б1.1.3.7 |     4
 2011 | Б1.1.3.6 |     4
 1936 | Б1.1.3.1 |     5
 1959 | Б1.1.3.1 |     5
 1959 | Б1.1.2.4 |     5
 1959 | Б1.1.1.7 |     5
 2011 | Б1.1.1.7 |     5
(10 строк)
```

2.9 Агрегатные функции и группировка

Агрегатные функции используются для получения обобщающих значений и их можно применять к *наборам* строк: ко всем строкам таблицы, строкам, определенным в предложении **WHERE** или к группам строк в предложении **GROUP BY**. Предложение **GROUP BY** разделяет таблицу на наборы, а агрегатная функция вычисляет для каждого из них итоговое значение.

Независимо от структуры набора строк, для каждого из них получается *единственное значение*.

Агрегатная функция вычисляет единственное значение, обрабатывая множество строк. Например, существуют следующие агрегатные функции, которые часто используются на практике:

sum(<выражение>) – вычисляет сумму значений выражения **<выражение>** по всем входным данным;

avg(<выражение>) – находит арифметическое среднее для всех входных значений;

count(*) – подсчитывает количество входных строк;

count(<выражение>) – подсчитывает количество входных строк, для которых значение выражения **<выражение>** не равно **NULL** (ненулевых значений);

max(<выражение>) – находит максимальное значение выражения **<выражение>** среди всех входных данных;

min(<выражение>) – находит минимальное значение выражения **<выражение>** среди всех входных данных.

Полное описание агрегатных функций общего назначения, агрегатных функций для статистических вычислений, сортирующих агрегатных функций и гипотезирующих агрегатных функций приведено в источнике.

На практике агрегатные функции чаще используются в комбинации с предложением **GROUP BY**.

***Пример* 2.25**. Выдать информацию о количестве проведенных экзаменов, количестве студентов, сдавших эти экзамены, а также о среднем экзаменационном балле.

Решение. Для получения нужного результата используем агрегатные функции **count(*)**, **count(<выражение>)** и **avg(<выражение>)**:

```
dis_un=# SELECT count(*), count(DISTINCT s_num), avg(marks)
dis_un-# FROM examinations;
 count | count |        avg
-------+-------+--------------------
    10 |     3 | 4.4000000000000000
(1 строка)
```

***Пример* 2.26**. Выдать информацию, аналогичную примеру 2.25, но с учетом выполнения группировки в разрезе идентификационных номеров дисциплин.

Решение. Для получения нужного результата используем те же агрегатные функции, а группировку выполним с помощью предложения **GROUP BY**, в котором укажем необходимый ключ группировки – *d_num* (идентификационный номер дисциплины):

```
dis_un=# SELECT d_num, count(*), count(DISTINCT s_num), avg(marks)
dis_un-# FROM examinations
dis_un-# GROUP BY d_num;
  d_num   | count | count |        avg
----------+-------+-------+--------------------
 Б1.1.1.7 |     2 |     2 | 5.0000000000000000
 Б1.1.2.4 |     2 |     2 | 4.5000000000000000
 Б1.1.3.1 |     2 |     2 | 5.0000000000000000
 Б1.1.3.6 |     1 |     1 | 4.0000000000000000
 Б1.1.3.7 |     1 |     1 | 4.0000000000000000
 Б1.1.3.8 |     2 |     2 | 3.5000000000000000
(6 строк)
```

Часто в запросах, выполняющих группировку данных, возникает необходимость провести фильтрацию на основании результатов агрегирования. Для этой цели может быть использовано предложение **HAVING** с помощью которого можно задать условия отбора в предложении **GROUP BY**.

Примечание. Отличие использования предложений **HAVING** и **WHERE** в том, что с помощью **WHERE** сначала выбираются строки, затем выполняется их группировка и вычисляются агрегатные функции, а при использовании **HAVING** отбор строк групп происходит после группировки и вычисления агрегатных функций.

***Пример* 2.27**. Выдать информацию о студентах, получивших на экзаменах по дисциплинам оценку «5» более двух раз.

Решение. Сформируем запрос с использованием предложения **HAVING**, в котором укажем условие фильтрации **count(*) > 2**:

```
dis_un=# SELECT students.surname
dis_un-# FROM students, examinations
dis_un-# WHERE students.s_num = examinations.s_num AND examinations.marks =5
dis_un-# GROUP BY students.surname
dis_un-# HAVING count(*) > 2;
 surname
---------
 Зайцев
(1 строка)
```

2.10 Изменение и удаление данных в таблицах

Изменение существующих данных. Оператор **UPDATE** предназначен для изменения существующих в таблице данных. В операторе **UPDATE** нужно указать изменяемые строки и

их новые значения, которые могут быть константами, выражениями или могут быть получены из других таблиц.

Упрощенный синтаксис команды **UPDATE**, изменяющий выбранные строки следующий:

UPDATE имя_таблицы
SET имя_столбца = выражение
[**WHERE** условие]

Ключевое слово **UPDATE** предшествует названию таблицы. В каждом операторе **UPDATE** можно изменять данные только одной таблицы.

В предложении **SET** определяются столбцы и указываются их новые значения, а в предложении **WHERE**, если оно используется, можно указать изменяемые строки. Если же предложение **WHERE** не используется, то при выполнении команды **UPDATE** будут изменены значения во всех строках столбцов, указанных в предложении **SET.**

Примечание. При обновлении данных со значениями столбцов можно выполнять математические преобразования.

***Пример* 2.28**. Увеличить в два раза число часов, отводимых на изучение дисциплины *ИТ-стандарты*.

Решение. Для изменения данных в таблице *disciplines* можно воспользоваться следующим оператором:

```
dis_un=# UPDATE disciplines
dis_un-# SET hours = hours*2
dis_un-# WHERE d_num = 'Б1.1.3.8';
UPDATE 1
```

Удаление данных. Команда **DELETE** позволяет удалять строки.

Синтаксис оператора **DELETE** следующий:

DELETE FROM имя_таблицы
WHERE условие

В предложении **WHERE** определяются строки, подлежащие удалению.

Примечание. Если в операторе **DELETE** будет отсутствовать предложение **WHERE,** то будут удалены все строки таблицы.

***Пример* 2.29**. Удалить из таблицы *examinations* строки с оценками ниже «4».

Решение. Для удаления данных в таблице *examinations* можно воспользоваться следующим оператором:

```
dis_un=# DELETE FROM examinations
dis_un-# WHERE marks < 4;
DELETE 1
```

В результате будет удалена одна строка:

```
dis_un=# SELECT * FROM examinations;
 s_num |  d_num   | marks
```

```
-------+----------+-------
  1959 | Б1.1.1.7 |     5
  2011 | Б1.1.1.7 |     5
  1959 | Б1.1.2.4 |     5
  1936 | Б1.1.2.4 |     4
  1959 | Б1.1.3.8 |     4
  1959 | Б1.1.3.1 |     5
  1936 | Б1.1.3.1 |     5
  2011 | Б1.1.3.7 |     4
  2011 | Б1.1.3.6 |     4
(9 строк)
```

ТРАНЗАКЦИИ

3.1 Выполнение транзакций

Транзакция – логическая единица работы СУБД. Управление транзакциями обеспечивает интерпретацию некоторой последовательности операторов SQL, как единого блока (неделимого объекта) и гарантирует, что будут выполнены все операции из некоторой группы (транзакция) или не будет выполнена ни одна из них. Механизм транзакций используется для контроля совпадений, то есть устранения конфликтов пользователей, которые пытаются одновременно обратиться к БД и восстановления БД в нормальное состояние после сбоя аппаратного или программного обеспечения.

В PostgreSQL транзакция – это список команд SQL, которые находятся внутри блока, начинающегося командой **BEGIN** и заканчивающегося командой **COMMIT**.

Если в процессе выполнения транзакции не требуется фиксировать ее изменения, то можно выполнить команду **ROLLBACK** вместо **COMMIT**, и тогда все изменения будут отменены. Управление операторами в транзакции возможно на более детальном уровне. Тогда для целей управления используются *точки сохранения*, которые позволяют выборочно отменять некоторые части транзакции и фиксировать остальные части. Определяются точки сохранения с помощью **SAVEPOINT**. Возврат к точке сохранения осуществляется с помощью команды **ROLLBACK TO**. Изменения в БД, которые произошли после точки сохранения и до момента отката, отменяются, а ранее произведенные изменения сохраняются.

Для рассмотрения дальнейших примеров изменим структуру базы данных: введем в состав схемы БД еще одну таблицу *groups* (группы) с целью отнесения каждого студента к определенной группе. Необходимо в каждой группе назначить старосту.

***Пример* 2.30**. Создать таблицу *groups* в соответствии с рекомендациями, описанными выше и использовать ограничение целостности, которое запрещает неопределенные значения.

Решение. Создадим таблицу *groups*:

```
dis_un=# CREATE TABLE groups(
dis_un(# g_id text PRIMARY KEY,
dis_un(# head integer NOT NULL REFERENCES students(s_num)
dis_un(# );
CREATE TABLE
```

***Пример* 2.31**. Добавить в таблицу *students* поле *g_id* (*идентификационный номер группы*) в соответствии в соответствии с изменениями в структуре базы данных, произведенными в примере 2.30.

Решение. Для добавления в таблицу *students* поля *g_id* используем команду **ALTER TABLE**, которая меняет определение существующей таблицы. Синтаксис команды подробно описан в документации. В результате добавления нового столбца получим:

```
dis_un=# ALTER TABLE students
dis_un-# ADD g_id text REFERENCES groups(g_id);
ALTER TABLE
```

Чтобы просмотреть какие таблицы теперь находятся в базе данных и какие поля определены в таблице *students* воспользуемся командами psql, приведенными в таблице 2.1.

Выполним команду:

```
dis_un=# \d
```

Результат выполнения командыр sql **\d**:

```
             Список отношений
 Схема |     Имя      |   Тип   | Владелец
--------+--------------+---------+----------
 public | disciplines  | таблица | postgres
 public | examinations | таблица | postgres
 public | groups       | таблица | postgres
 public | students     | таблица | postgres
(4 строки)
```

Выполним следующую команду:

```
dis_un=# \d students
```

Результат выполнения команды **\d students**:

```
    Таблица "public.students"
 Колонка |   Тип   | Модификаторы
---------+---------+--------------
 s_num   | integer | NOT NULL
 surname | text    |
 name    | text    |
 year_bd | integer |
 year_en | integer |
 g_id    | text    |
Индексы:
    "students_pkey" PRIMARY KEY, btree (s_num)
Ограничения внешнего ключа:
    "students_g_id_fkey" FOREIGN KEY (g_id) REFERENCES groups(g_id)
Ссылки извне:
```

TABLE "examinations" CONSTRAINT "examinations_s_num_fkey" FOREIGN KEY (s_num) REFERENCES students(s_num)
TABLE "groups" CONSTRAINT "groups_head_fkey" FOREIGN KEY (head) REFERENCES students(s_num)

***Пример* 2.32.** *Формирование транзакции*: выполнение двух операций одновременно – создание группы *ББИ-17* и назначение старостой группы *ББИ-17* студента Зайцева.

Решение. Эти две операции надо выполнить одновременно, так как они не имеют смысла одна без другой (составляют логическую единицу работы СУБД – транзакцию).

Начнем транзакцию командой **BEGIN**:

```
dis_un=# BEGIN;
BEGIN
```

После выполнения команды добавим группу и старосту Зайцева:

```
dis_un=# INSERT INTO groups(g_id, head)
dis_un-# SELECT 'ББИ-17', s_num
dis_un-# FROM students
dis_un-# WHERE surname = 'Зайцев';
INSERT 0 1
```

Затем необходимо открыть новое окно psql (щелкнуть по кнопке *Пуск*, выбрать пункт меню *Все программы*, выбрать папку *PostgreSQL 9.4.18* (*32 bit*) и запустить на выполнение программу *SQL Shell (psql)*) и запустить еще один процесс, работающий параллельно с первым. Результат приведен на рисунке 2.10.

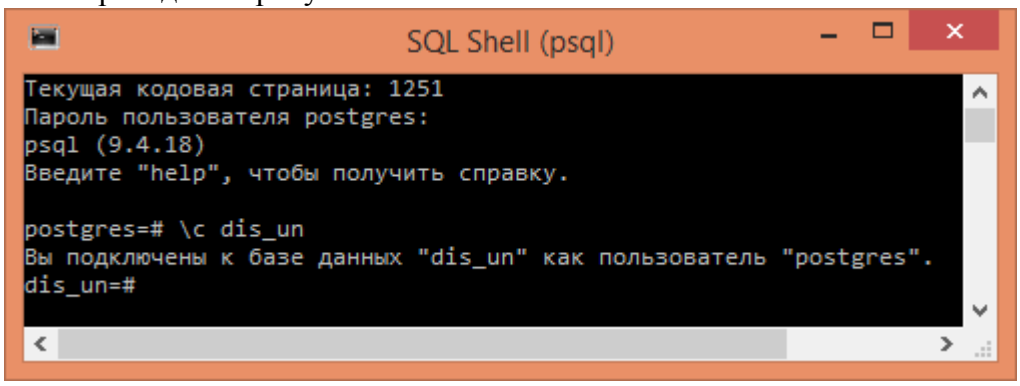


Рисунок 2.10 – Результат запуска второго процесса

Продолжим ввод в новом окне и получим следующий результат:

```
dis_un=# SELECT *
dis_un-# FROM groups;
 g_id | head
------+------
(0 строк)
```

Внесенные изменения не видны, так как транзакция еще не завершена. Вернемся в первое окно и обновим результаты командой **UPDATE**:

```
dis_un=# UPDATE students
dis_un-# SET g_id = 'ББИ-17';
UPDATE 3
```

Продолжим второй сеанс, в результате которого видны согласованные данные, которые актуальны на начало еще не оконченной транзакции:

```
dis_un=# SELECT *
dis_un-# FROM students;
 s_num | surname  | name  | year_bd | year_en | g_id
-------+----------+-------+---------+---------+------
  1959 | Зайцев   | Вадим |    1997 |    2017 |
  1936 | Михайлов | Павел |    1999 |    2017 |
  2011 | Романова | Алина |    2000 |    2018 |
(3 строки)
```

Выполним завершение транзакции в первом сеансе (окне) с помощью команды **COMMIT** и сохраним все изменения:

```
dis_un=# COMMIT;
COMMIT
```

И только сейчас второму сеансу доступны все изменения, проведенные транзакцией:

```
dis_un=# SELECT *
dis_un-# FROM groups;
  g_id  | head
--------+------
 ББИ-17 | 1959
(1 строка)

dis_un=# SELECT *
dis_un-# FROM students;
 s_num | surname  | name  | year_bd | year_en |  g_id
-------+----------+-------+---------+---------+--------
  1959 | Зайцев   | Вадим |    1997 |    2017 | ББИ-17
  1936 | Михайлов | Павел |    1999 |    2017 | ББИ-17
  2011 | Романова | Алина |    2000 |    2018 | ББИ-17
(3 строки)
```

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Что такое транзакция?
2. С какой целью используется команда COMMIT?
3. Когда в транзакциях необходимо выполнить команду **ROLLBACK** вместо команды **COMMIT?**

Приложение 1
Приоритет операторов в порядке убывания

Оператор	Очередность выполнения	Примечание
.	слева-направо	Разделитель имен таблицы и колонки
::	слева-направо	Приведение типов в PostgreSQL
[]	слева-направо	Выбор элемента массива
+, -	справа-налево	Унарный плюс, унарный минус
^	слева-направо	Возведение в степень
*, /, %	слева-направо	Умножение, деление, остаток от деления
+, -	слева-направо	Сложение, вычитание
IS		IS TRUE, IS FALSE, IS NULL и т.д.
IS NULL		Проверка на значение NULL
NOT NULL		Проверка на значение не NULL
Все остальные операторы	слева-направо	Все другие встроенные и пользовательские операторы
IN		Проверка членства
BETWEEN		Проверка диапазона
OVERLAPS		Проверка пересечения временных интервалов
LIKE ILIKE SIMILAR		Сравнение строк по шаблону
<, >		Меньше, больше
=	справа-налево	Проверка равенства, присваивание
NOT	справа-налево	Логическое отрицание
AND	слева-направо	Конъюнкция
OR	слева-направо	Дизъюнкция

Приложение 2
Таблица истинности логических операций

a	*b*	NOT *a*	*a* AND *b*	*a* OR *b*
FALSE	-	TRUE	-	-
TRUE	-	FALSE	-	-
NULL	-	NULL	-	-
FALSE	FALSE	-	FALSE	FALSE
FALSE	TRUE	-	FALSE	TRUE
TRUE	FALSE	-	FALSE	TRUE
TRUE	TRUE	-	TRUE	TRUE
FALSE	NULL	-	FALSE	NULL
TRUE	NULL	-	NULL	TRUE
NULL	NULL	-	NULL	NULL

Приложение 3

Результат выполнения запроса (пример 2.12)

```
dis_un=# SELECT * FROM disciplines, examinations;
  d_num   |          title          | hours | credit_units | s_num |  d_num   | marks
----------+-------------------------+-------+--------------+-------+----------+-------
 Б1.1.3.7 | Архитектура организации |   180 |            5 |  1959 | Б1.1.1.7 |     5
 Б1.1.3.1 | Базы данных             |   216 |            6 |  1959 | Б1.1.1.7 |     5
 Б1.1.1.7 | Введение в специальность |    36 |            1 |  1959 | Б1.1.1.7 |     5
 Б1.1.3.6 | Инжиниринг бизнеса      |   288 |            8 |  1959 | Б1.1.1.7 |     5
 Б1.1.3.8 | ИТ-стандарты            |   108 |            3 |  1959 | Б1.1.1.7 |     5
 Б1.1.2.4 | Математика              |   360 |           10 |  1959 | Б1.1.1.7 |     5
 Б1.1.3.7 | Архитектура организации |   180 |            5 |  2011 | Б1.1.1.7 |     5
 Б1.1.3.1 | Базы данных             |   216 |            6 |  2011 | Б1.1.1.7 |     5
 Б1.1.1.7 | Введение в специальность |    36 |            1 |  2011 | Б1.1.1.7 |     5
 Б1.1.3.6 | Инжиниринг бизнеса      |   288 |            8 |  2011 | Б1.1.1.7 |     5
 Б1.1.3.8 | ИТ-стандарты            |   108 |            3 |  2011 | Б1.1.1.7 |     5
 Б1.1.2.4 | Математика              |   360 |           10 |  2011 | Б1.1.1.7 |     5
 Б1.1.3.7 | Архитектура организации |   180 |            5 |  1959 | Б1.1.2.4 |     5
 Б1.1.3.1 | Базы данных             |   216 |            6 |  1959 | Б1.1.2.4 |     5
 Б1.1.1.7 | Введение в специальность |    36 |            1 |  1959 | Б1.1.2.4 |     5
 Б1.1.3.6 | Инжиниринг бизнеса      |   288 |            8 |  1959 | Б1.1.2.4 |     5
 Б1.1.3.8 | ИТ-стандарты            |   108 |            3 |  1959 | Б1.1.2.4 |     5
 Б1.1.2.4 | Математика              |   360 |           10 |  1959 | Б1.1.2.4 |     5
 Б1.1.3.7 | Архитектура организации |   180 |            5 |  1936 | Б1.1.2.4 |     4
 Б1.1.3.1 | Базы данных             |   216 |            6 |  1936 | Б1.1.2.4 |     4
 Б1.1.1.7 | Введение в специальность |    36 |            1 |  1936 | Б1.1.2.4 |     4
```

Продолжение приложения 3

```
 Б1.1.3.6 | Инжиниринг бизнеса      |   288 |            8 |  1936 | Б1.1.2.4 |     4
 Б1.1.3.8 | ИТ-стандарты            |   108 |            3 |  1936 | Б1.1.2.4 |     4
 Б1.1.2.4 | Математика              |   360 |           10 |  1936 | Б1.1.2.4 |     4
```

Б1.1.3.7 | Архитектура организации | 180 | 5 | 1959 | Б1.1.3.8 | 4
Б1.1.3.1 | Базы данных | 216 | 6 | 1959 | Б1.1.3.8 | 4
Б1.1.1.7 | Введение в специальность | 36 | 1 | 1959 | Б1.1.3.8 | 4
Б1.1.3.6 | Инжиниринг бизнеса | 288 | 8 | 1959 | Б1.1.3.8 | 4
Б1.1.3.8 | ИТ-стандарты | 108 | 3 | 1959 | Б1.1.3.8 | 4
Б1.1.2.4 | Математика | 360 | 10 | 1959 | Б1.1.3.8 | 4
Б1.1.3.7 | Архитектура организации | 180 | 5 | 1936 | Б1.1.3.8 | 3
Б1.1.3.1 | Базы данных | 216 | 6 | 1936 | Б1.1.3.8 | 3
Б1.1.1.7 | Введение в специальность | 36 | 1 | 1936 | Б1.1.3.8 | 3
Б1.1.3.6 | Инжиниринг бизнеса | 288 | 8 | 1936 | Б1.1.3.8 | 3
Б1.1.3.8 | ИТ-стандарты | 108 | 3 | 1936 | Б1.1.3.8 | 3
Б1.1.2.4 | Математика | 360 | 10 | 1936 | Б1.1.3.8 | 3
Б1.1.3.7 | Архитектура организации | 180 | 5 | 1959 | Б1.1.3.1 | 5
Б1.1.3.1 | Базы данных | 216 | 6 | 1959 | Б1.1.3.1 | 5
Б1.1.1.7 | Введение в специальность | 36 | 1 | 1959 | Б1.1.3.1 | 5
Б1.1.3.6 | Инжиниринг бизнеса | 288 | 8 | 1959 | Б1.1.3.1 | 5
Б1.1.3.8 | ИТ-стандарты | 108 | 3 | 1959 | Б1.1.3.1 | 5
Б1.1.2.4 | Математика | 360 | 10 | 1959 | Б1.1.3.1 | 5
Б1.1.3.7 | Архитектура организации | 180 | 5 | 1936 | Б1.1.3.1 | 5
Б1.1.3.1 | Базы данных | 216 | 6 | 1936 | Б1.1.3.1 | 5
Б1.1.1.7 | Введение в специальность | 36 | 1 | 1936 | Б1.1.3.1 | 5
Б1.1.3.6 | Инжиниринг бизнеса | 288 | 8 | 1936 | Б1.1.3.1 | 5

Продолжение приложения 3

Б1.1.3.8 | ИТ-стандарты | 108 | 3 | 1936 | Б1.1.3.1 | 5
Б1.1.2.4 | Математика | 360 | 10 | 1936 | Б1.1.3.1 | 5
Б1.1.3.7 | Архитектура организации | 180 | 5 | 2011 | Б1.1.3.7 | 4
Б1.1.3.1 | Базы данных | 216 | 6 | 2011 | Б1.1.3.7 | 4
Б1.1.1.7 | Введение в специальность | 36 | 1 | 2011 | Б1.1.3.7 | 4
Б1.1.3.6 | Инжиниринг бизнеса | 288 | 8 | 2011 | Б1.1.3.7 | 4
Б1.1.3.8 | ИТ-стандарты | 108 | 3 | 2011 | Б1.1.3.7 | 4
Б1.1.2.4 | Математика | 360 | 10 | 2011 | Б1.1.3.7 | 4

Б1.1.3.7 | Архитектура организации | 180 | 5 | 2011 | Б1.1.3.6 | 4
Б1.1.3.1 | Базы данных | 216 | 6 | 2011 | Б1.1.3.6 | 4
Б1.1.1.7 | Введение в специальность | 36 | 1 | 2011 | Б1.1.3.6 | 4
Б1.1.3.6 | Инжиниринг бизнеса | 288 | 8 | 2011 | Б1.1.3.6 | 4
Б1.1.3.8 | ИТ-стандарты | 108 | 3 | 2011 | Б1.1.3.6 | 4
Б1.1.2.4 | Математика | 360 | 10 | 2011 | Б1.1.3.6 | 4
(60 строк)

ТЕСТОВЫЕ ЗАДАНИЯ ДЛЯ САМОПОДГОТОВКИ

1. _____ модель данных определяет способ размещения данных непосредственно на машинном носителе, учитывает распределение данных, методы доступа и способы индексирования, т.е. определяют способы размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне.

2. Модель _____ стала фактическим стандартом при инфологическом моделировании БД.

3. _____ – такая связь, при которой экземпляр дочерней сущности не идентифицируется через свою ассоциацию с родительской сущностью.

4. В реляционных БД к простым типам относятся следующие типы: _____; строковый; численный.

5. Специальные реляционные операции над отношениями включают: выборку, _____, соединение, деление.

6. _____ модель предметной области отражает логические взаимосвязи между элементами данных безотносительно их содержания и физической организации.

7. Основные понятия ER-модели**: _____, _____, _____.**

8. _____ – перенос атрибутов одной сущности в другую для установления связи между ними.

9. В реляционных базах данных к структурированным типам данных относятся _____, записи (структуры).

10. Логические ограничения, накладываемые на данные, называются ограничениями _____.

11. Описание логической структуры БД на языке СУБД называется _____.

12. _____ – система условных обозначений, принятая в какой-либо области знаний или деятельности.

13. _____ атрибут называется внешним ключом и помечается на ER-диаграмме символами.

14. В реляционных базах данных к структурированным типам данных относятся массивы, _____.

15. Отношение находится в _____ нормальной форме, если все атрибуты отношения принимают простые значения (атомарные или неделимые), не являющимися множеством или кортежем из более элементарных составляющих.

16. _____ моделью называется ориентированная на человека и не зависимая от типа СУБД модель предметной области, определяющая совокупности информационных объектов, их атрибутов и отношений между объектами, динамику изменений предметной области, а также характер информационных потребностей пользователей.

17. В нотации IDEF1X _____ – реальный или представляемый объект, информация о котором должна сохраняться и быть доступной. В диаграммах ER-модели _____ представляется в виде прямоугольника, содержащего имя.

18. _____ – характеристика связи между сущностями, предназначенная для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней.

19. _____ представляет собой кортеж из некоторого декартового произведения множеств.

20. _____ – это неадекватность модели данных предметной области, либо некоторые дополнительные трудности в реализации ограничений предметной области средствами СУБД.

21. Языки манипулирования данными разделяются на: процедурные и _____.

22. _____ – это именованная характеристика, являющаяся некоторым свойством сущности.

23. Если одному экземпляру родительской сущности соответствуют 1 или много экземпляров дочерней сущности (исключено нулевое значение), то на диаграмме (IDEF1X) это помечается _____.

24. _____ – допустимое потенциальное множество значений типа данных.

25. Существуют три вида аномалий моделей данных предметной области: вставки, _____ и модификации.

26. При проектировании БД организацию данных принято рассматривать на следующих уровнях: _____, _____, _____.

27. _____ атрибут – состоит из одного элемента данных.

28. Если одному экземпляру родительской сущности соответствуют 0 или 1 экземпляр дочерней сущности (исключены множественные значения), то на диаграмме (IDEF1X) это помечается _____.

29. _____ отношения есть пара вида: <Имя_атрибута : Имя_домена>.

30. Существуют три вида аномалий моделей данных предметной области: _____, удаления и модификации.

31. Языковые средства СУБД являются важнейшим компонентом СБД, т.к. обеспечивают интерфейс пользователей разных категорий с СБД и включают: _____.

32. _____ атрибут – состоит из нескольких элементов данных.

33. Если одному экземпляру родительской сущности соответствует заранее заданное число экземпляров дочерней сущности, то на диаграмме (IDEF1X) это помечается _____.

34. _____ – это множество кортежей, соответствующих одной схеме отношения.

35. Существуют три вида аномалий моделей данных предметной области: вставки, удаления и _____.

36. Данные, отражающие состояние предметной области и используемые в автоматизированных информационных системах, принято называть _____.

37. _____ атрибут – содержит одно значение для одной сущности.

38. _____ связь используется для описания структур, в которых сущность является типом (категорией) другой сущности.

39. Число атрибутов в отношении называют _____ отношения.

40. Отношение находится в _____ нормальной форме тогда и только тогда, когда отношение находится в 1НФ, и нет неключевых атрибутов, зависящих от части сложного ключа. Если потенциальный ключ отношения является простым, то отношение автоматически находится в _____ нормальной форме.

41. К клиент-серверным СУБД относятся: ____, _____, _____, ____, _____.

42. _____ атрибут – содержит несколько значений для одной сущности.

43. _____ – атрибут родового предка, который показывает, как отличить одну категориальную сущность от другой.

44. Мощность множества кортежей отношения называют мощностью _____.

45. _____ атрибут – это атрибут, не входящий в состав никакого потенциального ключа.

46. Структурированные БД по типу используемой модели делятся: на_____, _____, _____, _____ и _____.

47. _____ атрибут – может иметь пустое (неопределенное) значение.

48. Если экземпляру родового предка соответствует экземпляр в каком-либо потомке, то связь является _____, на ER-диаграмме изображается с помощью дискриминатора _____ связи.

49. Схема БД (в структурном смысле) – это набор именованных схем _____.

50. Отношение находится в _____ нормальной форме тогда и только тогда, когда отношение находится во 2НФ и все неключевые атрибуты взаимно независимы.

51. _____ – наименьшая семантическая единица информации в структурированной БД.

52. _____ атрибут – значение, производное от значения другого атрибута.

53. Если категория еще не выстроена полностью и в родовом предке есть экземпляры, для которых нет соответствующих экземпляров в потомках, категория является _____, на ER-диаграмме изображается с помощью дискриминатора _____ связи.

54. Реляционная база данных – это набор _____, имена которых совпадают с именами схем _____ в схеме БД.

55. Специальные реляционные операции над отношениями включают: выборку, проекцию, _____, деление.

56. В структурированных БД различают несколько уровней информационных единиц, входящих одна в другую: _____, запись, _____.

57. _____ – минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности.

58. _____ модель данных – представление базы данных в виде древовидной структуры, состоящей из объектов (данных) различных уровней. Такие объекты находятся в отношении предка к потомку.

59. _____ – элемент данных, который позволяет уникально идентифицировать отдельные экземпляры некоторого типа сущности (кортежи отношения).

60. Специальные реляционные операции над отношениями включают: выборку, проекцию, соединение, _____.

61. Программная составляющая СУБД включает компоненты: _____, _____, _____, _____.

62. _____ атрибуты в нотациях изображаются на диаграмме подчеркиванием.

63. _____ модель данных – логическая модель данных, в которой у потомка может иметься любое число предков.

64. В реляционной БД существуют следующие виды ключей: первичные, сложные, потенциальные, альтернативные, _____.

65. _____ двух односхемных отношений *A* и *B* – отношение *C*, построенное по той же схеме и содержащее все кортежи отношения *A* и все кортежи отношения *B*.

66. К программным средствам СУБД относятся: программная составляющая СУБД, системное ПО и _____.

67. _____ – поименованная ассоциация между сущностями, значимая для рассматриваемой предметной области.

68. В _____ модели информация представляется в виде двумерных таблиц, а операции сводятся к манипуляциям с таблицами.

69. В реляционной БД существуют следующие виды ключей: _____, сложные, потенциальные, альтернативные, внешние.

70. _____ двух односхемных отношений *A* и *B* – отношение *C*, построенное по той же схеме и содержащее только те кортежи отношения *A*, которые есть в отношении *B*.

71. Непроцедурные языки, относящиеся к языковым средствам СБД и основанные на реляционном исчислении: табличный язык QBE, _____.

72. _____ на ER-диаграмме изображается линией, соединяющей две сущности.

73. _____ модель данных состоит из трех частей: структурной; целостной; манипуляционной.

74. В реляционной БД существуют следующие виды ключей: первичные, сложные, потенциальные, _____, внешние.

75. _____ двух односхемных отношений *A* и *B* – отношение *C*, построенное по той же схеме и содержащее те кортежи отношения *A*, которых нет в отношении *B*.

76. Процедурные языки манипулирования данными в СБД различаются по основным информационным единицам, которыми они манипулируют: языки, ориентированные на позаписную обработку данных и языки, ориентированные на _____.

77. Каждая связь в ER-диаграмме может принадлежать к следующим типам связи: _____, _____, _____.

78. Манипуляционная часть реляционной модели описывает два эквивалентных способа манипулирования реляционными данными – _____ алгебру и _____ исчисление.

79. В реляционной БД существуют следующие виды ключей: первичные, сложные, _____, альтернативные, внешние.

80. _____ двух отношений *A* и *B* – отношение *C*, схема которого включает все атрибуты отношений *A* и *B*, а тело отношения состоит из всевозможных сцеплений кортежей отношений *A* и *B*.

81. Языковые средства современных СУБД относятся к _____ поколению.

82. Связь типа _____ означает, что один экземпляр первой сущности (левой) связан с одним экземпляром второй сущности (правой).

83. Основной структурой данных в реляционной модели является _____.

84. _____ _____ – это атрибут или группа атрибутов, уникально идентифицирующая каждый экземпляр сущности.

85. _____ на отношении *A* – отношение *C*, построенное по той же схеме, что и отношение *A,* и содержащее подмножество кортежей отношения *A*, удовлетворяющих условию выборки.

86. _____ – это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

87. Связь типа _____ означает, что один экземпляр первой сущности (левой) связан с несколькими экземплярами второй сущности (правой).

88. Основными понятиями реляционных БД являются: _____, _____, _____, _____, _____, _____.

89. _____ _____ однозначно идентифицирует каждый кортеж отношения.

90. _____ отношения *A* – отношение *C*, схема которого состоит из подмножества атрибутов, по которым производится данная операция, а кортежи содержат соответствующие значения из кортежей отношения *A*.

91. Система управления базами данных (СУБД) – совокупность программных и лингвистических (языковых) средств общего или специального назначения, обеспечивающих управление созданием и использованием _____.

92. Связь типа _____ означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности.

93. Основными понятиями реляционных БД являются: тип данных, атрибут, домен, _____, ключ, кортеж.

94. _____ ключ состоит из нескольких атрибутов.

95. _____ отношений *A* и *B* подобно декартовому произведению отношений, но сцепление кортежей отношений *A* и *B* происходит не каждое с каждым, а по некоторому условию.

96. Система баз данных включает следующие взаимосвязанные и взаимозависимые компоненты: базу данных, технические средства, организационно-методические средства, администратора и _____.

97. Тип связи много-ко-многим – _____ тип связи, допустимый на ранних этапах разработки модели. В дальнейшем он должен быть заменен двумя связями типа _____ путем создания промежуточной сущности.

98. Основными понятиями реляционных БД являются: тип данных, атрибут, домен, отношение, _____, кортеж.

99. _____ ключ – это потенциальный ключ, не ставший первичным.

100. _____ отношений *A* и *B* – соединение по условию неравенства значений атрибутов отношений *A* и *B*.

101. Специальные реляционные операции, входящие в состав реляционной алгебры: _____.

102. _____ – такая связь, при которой экземпляр дочерней сущности идентифицируется через свою ассоциацию с родительской сущностью.

103. В реляционной БД простые (атомарные) типы данных не обладают внутренней структурой. Данные такого типа называют _____.

104. Реляционная модель представляет БД в виде множества взаимосвязанных отношений. Связи между отношениями устанавливаются с помощью _____ ключей.

105. _____ соединение отношений *A* и *B* – соединение по условию равенства значений некоторого общего атрибута отношений *A* и *B* (чаще всего равенство значений первичного и внешнего ключа).

ВОПРОСЫ ДЛЯ ПОДГОТОВКИ К ЭКЗАМЕНУ

1. Понятие «система баз данных» (СБД). Требования к СБД. Основные компоненты СБД.
2. Языковые средства СБД и их классификация.
3. Понятие «база данных».
4. Архитектура систем баз данных.
5. Понятия «система управления базами данных» (СУБД) и «базы данных» (БД).
6. Программная составляющая системы баз данных.
7. Технические средства систем баз данных.
8. Структурная схема взаимодействия компонентов СБД.
9. Классификация баз данных. Приведите примеры.
10. Классификация СУБД. Приведите примеры.
11. Функции и обзор современных СУБД.
12. Современная СУБД, как интегрированная платформа обработки информации.
13. Основные характеристики серверных СУБД.
14. Этапы проектирования баз данных и их взаимосвязь.
15. Понятия «информационно-логическая модель» (концептуальная) и «даталогическая модель» данных.
16. Способы описания инфологической модели.
17. Модель «сущность-связь» (ER-модель) - стандарт при инфологическом моделировании баз данных.
18. Понятие «физическая модель» данных.
19. ER-модель и основные понятия модели (сущность, экземпляр сущности, атрибут сущности, связь).
20. Виды атрибутов ER-модели. Ключевые и неключевые атрибуты сущности.
21. Нотации для представления ER-модели.
22. Понятие «связь» в ER-модели. Типы связей (один-к-одному, один-ко-многим, много-ко-многим) и обозначение типов связи в нотации IDEF1X.
23. Идентифицирующая связь. Неидентифицирующая связь. Мощность связи. Зависимая и независимая сущности. Обозначение связей, мощности связи и сущностей в нотации IDEF1X.
24. Понятие «миграция». Ключевые и неключевые атрибуты сущности.
25. Понятие категориальной связи сущностей и ее типы. Понятие дискриминатора. Дискриминаторы полной и неполной связи и их обозначение на ER-диаграммах.
26. Иерархическая, сетевая и реляционная модели данных в базах данных. Основные понятия иерархической структуры.
27. Основные понятия реляционной модели данных (отношение, атрибут, домен, кортеж, первичный ключ отношения, внешний ключ, связь отношений, контроль целостности связей).
28. Основные понятия реляционных баз данных: тип данных, домен, атрибут, кортеж, ключ, отношение.
29. Реляционные базы данных. Типы данных: простые типы данных, структурированные типы данных, ссылочные типы данных. Приведите примеры.
30. Понятия «отношение», «атрибут», «кортеж», «степени отношения», «мощности отношения», «схема базы данных» в реляционных базах данных. Свойства отношений.
31. Понятие «ключ». Виды ключей: первичные, сложные, потенциальные, альтернативные, внешние.

32. Понятия «целостность сущностей» и «целостность по ссылкам».

33. Реляционная алгебра. Теоретико-множественные операции. Операция объединения отношений. Приведите пример.

34. Реляционная алгебра. Теоретико-множественные операции. Операция пересечения отношений. Приведите пример.

35. Реляционная алгебра. Теоретико-множественные операции. Операция взятия разности отношений (вычитание). Приведите пример.

36. Реляционная алгебра. Теоретико-множественные операции. Операция взятия декартова произведения отношений (декартово произведение). Приведите пример.

37. Специальные реляционные операции: выборка на отношении. Приведите пример.

38. Специальные реляционные операции: проекция отношения. Приведите пример.

39. Специальные реляционные операции: соединение отношений. Приведите пример. Естественное соединение отношений. Тета-соединение отношений.

40. Специальные реляционные операции: деление. Приведите пример.

41. Реляционное исчисление. Реляционное исчисление кортежей и реляционное исчисление доменов в контексте баз данных. Понятие предиката. Логические операторы AND, OR, NOT.

42. Операции реляционной алгебры (по Кодду). Типы операций соединения отношений.

43. Понятия «нормализация отношений» и «нормальная форма». Основные свойства нормальных форм.

44. Первая нормальная форма (1НФ). Функциональная зависимость между атрибутами отношения (полная, частичная, транзитивная).

45. Декомпозиция схемы отношения. Нормальные формы. Влияние степени нормализации на производительность работы СУБД.

46. Декомпозиция схемы отношения. Алгоритм перехода от первой нормальной формы ко второй нормальной форме.

47. Декомпозиция схемы отношения. Алгоритм перехода от второй нормальной формы к третьей нормальной форме.

48. Нормальная форма Бойса-Кодда (НФБК). Алгоритм нормализации (приведение к НФБК).

49. Четвертая нормальная форма (4НФ). Алгоритм перехода от НФБК к 4НФ.

50. Пятая нормальная форма (5НФ). Алгоритм перехода от 4НФ к 5НФ.

51. Индексирование записей. Повышение производительности обработки индексированных данных.

52. Типы данных в Postgresql. Правила и функции преобразования типов.

53. Язык SQL. Существующие стандарты SQL-языка.

54. Язык SQL. Команда CREATE DATABASE и ее синтаксис.

55. Язык SQL. Команда CREATE TABLE и ее синтаксис.

56. Язык SQL. Команда INSERT INTO и ее синтаксис.

57. Язык SQL. Выборка данных. Оператор SELECT и его синтаксис.

58. Язык SQL. Соединения. Операция LEFT JOIN.

59. Язык SQL. Создание подзапросов.

60. Язык SQL. Сортировка по возрастанию и убыванию.

61. Язык SQL. Группировка и агрегатные функции.

62. Язык SQL. Использование условия WHERE (до группировки) и предложения HAVING (после группировки). Их использование в запросах.

63. Язык SQL. Изменение и удаление данных. Оператор UPDATE и его синтаксис.

64. Язык SQL. Изменение и удаление данных. Оператор DELETE и его синтаксис.

65. Язык SQL. Оператор ALTER TABLE и добавление столбцов в таблицу.
66. Язык SQL. Транзакции. Использование BEGIN и COMMIT в транзакциях.
67. СУБД Postgrisql и основные команды приложения psql.