А.А. Рузаков

УПРАВЛЕНИЕ ДАННЫМИ

Челябинск

2015

Министерство образования и науки
Российской федерации

Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Челябинский государственный педагогический университет»

А.А. Рузаков

Управление данными

Учебное пособие

Челябинск
2015

УДК 681.14(021)
ББК 32.973.2-018Я73
Р 83

Рузаков, А.А. Управление данными: учеб. пособие / А.А. Рузаков. – Челябинск: Изд-во Челяб. гос. пед. ун-та, 2015. – 132 с.

ISBN 978-5-906777-40-9

Пособие раскрывает содержание курса «Управление данными»: основные понятия теории баз данных, проектирование баз данных и применение языка баз данных SQL. Все команды языка баз данных SQL демонстрируются на спроектированной и реализованной базе данных в СУБД Microsoft Office Access 2013. Каждый раздел завершается вопросами для самопроверки и списком рекомендуемой литературой.

Адресована студентам направления подготовки 09.03.02 «Информационные системы и технологии», а также преподавателям соответствующих дисциплин.

Рецензенты:

А.Б. Кузнецов, канд. пед. наук, доцент

Н.В. Лапикова, канд. пед. наук

ISBN 978-5-906777-40-9

СОДЕРЖАНИЕ

Введение

Бакалавр по направлению «Информационные системы и технологии» должен:

- знать основные положения теории баз данных, хранилищ данных, витрин данных, баз знаний, концептуальные, логические и физические модели данных;
- уметь разрабатывать информационно-логическую, функциональную и объектно-ориентированную модели информационной системы, модели данных информационных систем;
- владеть методами и средствами представления данных и знаний о предметной области, методами и средствами анализа информационных систем, технологиями реализации, внедрения проекта информационной системы;
- владеть инструментальными средствами обработки информации.

Эти знания и навыки приобретаются в процессе изучения информатики, информационных технологий, управления данными, методов и средств проектирования информационных систем и технологий, и в процессе изучения других учебных дисциплин учебного плана по направлению подготовки 09.03.02 «Информационные системы и технологии».

Федеральным государственным образовательным стандартом высшего профессионального образования рекомендуется изучение дисциплины «Управление данными» в рамках базовой части профессионального цикла.

Материал курса делится на 3 основных раздела: основные положения теории баз данных, проектирование баз данных и язык SQL.

При разработке данного пособия были поставлены следующие задачи:

- оптимизировать содержание и трудоемкость дисциплины «Управление данными» для выполнения обязательных требований стандарта;
- обеспечить обучающихся всей необходимой информацией по проектированию баз данных;
- рассмотреть основные понятия языка баз данных SQL с примерами его использования.

Учебное пособие содержит следующие темы:

1. ***Введение в управление данными.*** Основные определения. Архитектура системы баз данных. Система управления базой данных. Система управления передачей данных. Архитектура «клиент-сервер». Распределённая обработка.

2. ***Реляционные базы данных.*** Реляционная модель данных. Типы данных. Условия целостности данных. Основные свойства отношений. Индексы.

3. ***Проектирование баз данных.*** Цели проектирование. Нормализация данных. Универсальное отношение. Проблемы, вызываемые использованием единственного отношения. Приведение модели к требуемому уровню нормальной формы.

4. ***Введение в язык баз данных SQL.*** Типы команд SQL. Типы данных SQL. Управление объектами базы данных. Манипулирование данными. Изменение данных, хранящихся в таблице. Создание SQL-запросов.

В конце каждой темы приведены вопросы и задания для самопроверки, список рекомендуемой литературы.

1. Основные понятия теории баз данных

1.1. Основные определения

Информация – данные, которым придается некоторый смысл (интерпретация) в конкретной ситуации в рамках некоторой системы понятий. Информация представляется посредством кодирования данных и извлекается путем их декодирования и интерпретации.

Данные – способ представления информации в определенной, фиксированной форме, пригодной для обработки, хранения и передачи.

Банк данных – это система специальным образом организованных данных: баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного хранения и коллективного многоцелевого использования данных.

База данных (БД) – именованная совокупность данных, отражающая состояние объектов и их отношений в конкретной предметной области.

База знаний – это особого рода база данных, разработанная для оперирования знаниями (метаданными). Полноценные базы знаний содержат в себе не только фактическую информацию, но и правила вывода, допускающие автоматические умозаключения о вновь вводимых фактах и, как следствие, осмысленную обработку информации.

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

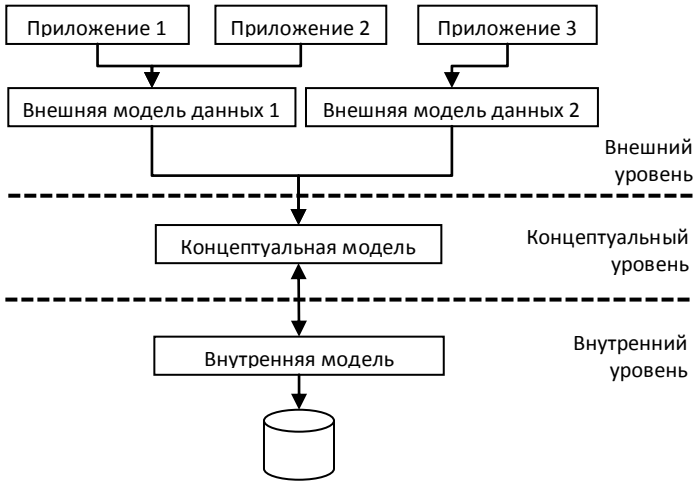
Программы, с помощью которых пользователи работают с базой данных, называются приложениями.

Информационная система (ИС) – организационная совокупность технических и обеспечивающих средств, технологических процессов и кадров, реализующих функции сбора, обработки, хранения, поиска, выдачи и передачи информации

Основной целью создания ИС является удовлетворение информационных потребностей пользователей путем предоставления необходимой им информации на основе хранимых данных.

1.2. Архитектура системы баз данных

Архитектура ANSI/SPARC, предложена в 1975 г. Комитетом планирования стандартов и норм SPARC (Standards Planning and Requirements Committee) Национального института стандартизации США (American National Standards Institute – ANSI) включает три уровня и соответствующие модели данных: внутренний, концептуальный и внешний (Рис. 1).



Три уровня архитектуры ANSI/SPARC

Основная цель этой архитектуры состоит в отделении пользовательского представления о данных в БД от их физического представления. Использование таких представлений о данных позволяет обеспечить выполнение основного требования к БД – независимости программ и данных. При изменении прикладных программ может измениться соответствующее внешнее представление, но концептуальная модель данных не изменяется, и, соответственно, не будут изменяться другие прикладные программы. При изменении внутреннего представления (структур хранения) концептуальная модель не изменяется, соответственно, не изменяются прикладные программы.

Уровень внешних моделей данных – самый верхний уровень, где каждая модель имеет свое отражение данных. Данный уровень определяет точку зрения отдельных приложений на БД. Каждому приложению доступны для обработки только те данные, которые ему необходимы, т.е. включены в его внешнюю модель. С помощью внешних моделей поддерживается санкционированный доступ к данным БД приложений (ограничен состав и структура данных концептуальной модели, доступных в приложении, а также заданы допустимые режимы обработки этих данных: добавление, изменение, удаление, поиск).

Концептуальный уровень – центральное связующее звено. Здесь база данных представлена в наиболее общем виде, который объединяет информацию, используемую всеми приложениями, работающими с этой базой данных. Концептуальный уровень отражает обобщенную модель предметной области (объектов реального мира), для которой создавалась база данных. Концептуальная модель отражает только существенные с точки зрения обработки особенности объектов реального мира.

Внутренний уровень (физический) – это уровень, наиболее близкий к физическому хранению информации, т.е. связанный со способами сохранения информации на физических устройствах.

Если внешний уровень связан с индивидуальными представлениями пользователей, то концептуальный уровень связан с обобщённым представлением пользователей. Т.е. может существовать несколько внешних представлений, каждое из которых состоит из более или менее абстрактного представления определённой части базы данных, и только одно концептуальное представление, состоящее из абстрактного представления базы данных в целом. Также есть единственное внутреннее представление, отражающее способ физического хранения всей базы данных.

Появление новых или изменение информационных потребностей существующих приложений требуют определения для них корректных внешних моделей, при этом на уровне концептуальной и внутренней модели данных изменений не происходит.

Изменения в концептуальной модели, вызванные появлением новых видов данных или изменением их структур, могут затрагивать не все приложения, т.е. обеспечивается определенная независимость программ от данных.

Изменения в концептуальной модели должны отражаться на внутренней модели, и при неизменной концептуальной модели возможна самостоятельная модификация внутренней модели БД с целью улучшения ее характеристик (время доступа к данным, расхода памяти внешних устройств и др.).

1.3. Система управления базой данных

СУБД представляет собой программное обеспечение, которое управляет всем доступом к базе данных. Данный процесс происходит следующим образом:

1. Пользователь выдаёт запрос на доступ к хранимым данным, применяя определённый язык данных (например, SQL).
2. СУБД получает этот запрос и анализирует его.
3. СУБД просматривает внешнюю схему для этого пользователя, соответствующую связку внешний-концептуальный, концептуальную схему, связку концептуальный-внутренний и определение структуры хранения.
4. СУБД выполняет необходимые операции в хранимой базе данных.

Функции СУБД:

1. Определение данных. СУБД должна предоставлять средства определения данных (внешние схемы, концептуальную схему, внутреннюю схему, а также все связки между ними), в исходной форме и преобразовывать эти определения в форму соответствующих объектов. Т.е. СУБД должна включать в себя компонент языкового процессора для различных языков определений данных. СУБД должна также понимать синтаксис языка определений данных.
2. Обработка данных. СУБД должна обрабатывать запросы пользователя на выборку, изменение или удаление информации в базе данных или на добавление новых данных в базу данных, т.е. СУБД должна включать в себя компонент процессора языка обработки данных.
3. Оптимизация и выполнение. Запросы должны быть обработаны оптимизатором, назначение которого состоит в поиске достаточно эффективного способа выполнения каждого из запросов.

4. Защита и сохранение целостности данных. СУБД должна контролировать пользовательские запросы и пресекать любые попытки нарушения ограничений защиты и сохранения целостности данных, определенные администратором базы данных.
5. Восстановление данных и поддержка параллельности. СУБД или другой связанный с ней программный компонент, обычно называемый менеджером транзакций, должен обеспечивать необходимый контроль над восстановлением данных и управлением параллельностью обработки.
6. Словарь данных. СУБД должна поддерживать функцию ведения словаря данных. Сам словарь данных вполне можно считать базой данных, но не пользовательской, а системной. Словарь содержит информацию о данных (метаданные), т.е. определения других объектов системы, а не просто «сырые сведения». В словаре будут сохранены исходная и объектная формы всех существующих схем (внешней, концептуальной и т.д.) и связок.
7. Производительность. СУБД должна выполнять все указанные функции с максимально возможной эффективностью.

Назначением СУБД является предоставление пользовательского интерфейса с системой баз данных. Пользовательский интерфейс может быть определён как существующий в системе ограничительный уровень, ниже которого всё невидимо для пользователя.

1.4. Система управления передачей данных

Запросы к БД от конечных пользователей передаются от рабочей станции пользователя (может быть физически удалена от самой системы баз данных) к некоторому интерактивному приложению, а от него к СУБД в форме коммуникационных сообщений. Ответы пользователю также передаются в форме подобных сообщений. Передача таких сообщений происходит под

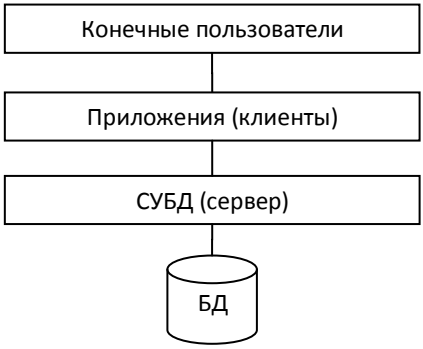
управлением другого программного компонента – менеджера передачи данных, который не является частью СУБД, а представляет собой автономную систему со своими правами.

От менеджера передачи данных и СУБД требуется согласованная совместная работа, они иногда рассматриваются как равноправные партнёры более высокого уровня, называемого системой базы данных и передачи данных.

СУБД отвечает за работу с базой данных, а менеджер передачи данных обрабатывает все сообщения, поступающие в СУБД и из неё (в то приложение, которое использует СУБД).

1.5. Архитектура «клиент-сервер»

Назначение систем баз данных – поддержка разработки и выполнения приложений баз данных. Поэтому на высоком уровне систему баз данных можно рассматривать как систему с очень простой структурой, состоящей из двух частей: сервера (внутреннего компонента или машины базы данных) и набора клиентов (внешнего компонента, или внешнего интерфейса) (Рис. 2).



Архитектура «клиент-сервер»

Сервер – это и есть СУБД. Сервер поддерживает все основные функции СУБД: определение данных, обработку данных, защиту данных, поддержание целостности данных и т.д. Он предоставляет полную поддержку внешнего, концептуального и внутреннего уровней архитектуры ANSI/SPARC.

Клиенты – это различные приложения, выполняющиеся поверх СУБД: приложения, написанные пользователями; встроенные приложения, предоставляемые поставщиками СУБД или некоторыми сторонними поставщиками программного обеспечения. Для сервера нет разницы между встроенными приложениями и приложениями, написанными пользователем, – все они используют один и тот же интерфейс сервера (интерфейс внешнего уровня).

Приложения делятся на две чётко определённые категории:

- Приложения, написанные пользователями, – обычные прикладные программы.
- Приложения, предоставляемые поставщиками (инструментальные средства). Их задача – содействовать процессу создания и выполнения других приложений (для решения некоторых специфических задач). С их помощью конечные пользователи могут создавать приложения без написания традиционных программ (кодирования).

Поставляемые инструментальные средства, в свою очередь, делятся на несколько самостоятельных классов:

- процессоры языков запросов;
- генераторы отчётов;
- графические бизнес-подсистемы;
- электронные таблицы;
- процессоры обычных языков;
- статистические пакеты;

- средства управления копированием или средства извлечения данных.
- генераторы приложений;
- другие средства разработки приложений, включая CASE-инструменты (Computer-Aided Software Engineering – автоматизация разработки программного обеспечения).

Поддержка создания и выполнения приложений – это главная задача системы баз данных. Качество имеющихся клиентских инструментальных средств должно быть главным фактором при выборе СУБД.

Разделение системы на две части – сервер и клиенты – позволяет работать им на разных компьютерах. Иначе говоря, существует возможность организации распределённой обработки данных. Распределённая обработка предполагает, что отдельные компьютеры можно соединить коммуникационной сетью таким способом, что выполняемая задача обработки данных будет распределяться между ними.

1.6. Распределённая обработка

Распределённая обработка может быть самой разнообразной и осуществляться на разных уровнях.

Архитектура «клиент-сервер» стала синонимом структуры, в которой клиент и сервер запускаются на разных компьютерах.

Существует множество аргументов в пользу этой схемы:

- Параллельная обработка. В этом случае для решения общей задачи применяется сразу несколько процессоров, так как работа сервера (базы данных) и клиента (приложения) осуществляется параллельно. Показатели времени реакции системы на запросы и производительность системы должны улуч-

шиться.

- Компьютер сервера специализирован для работы с СУБД и, соответственно, может обеспечить лучшую производительность СУБД.
- Компьютер клиента также может быть адаптированным к потребностям конечного пользователя и поэтому способен обеспечивать наиболее удобный интерфейс, гарантировать высокий уровень готовности, быструю реакцию системы и другие дополнительные удобства при использовании.
- Одна база данных может совместно использоваться несколькими различными клиентскими системами.

Работа сервера и клиента на отдельных компьютерах отвечает принципам работы многих предприятий.

Отдельный компьютер клиента может иметь доступ к нескольким разным серверам. Это полезная возможность, так как предприятие обычно выполняет обработку данных таким образом, что полный набор всех данных не сохраняется на одном компьютере, а распределяется на отдельных компьютерах, причем в приложениях иногда необходим доступ к данным сразу нескольких компьютеров.

Такой доступ предоставляется двумя способами.

1. Клиент может получать доступ к любому количеству серверов, но лишь к одному из них в каждый момент времени. В подобной системе невозможно за один запрос получить комбинированные данные от двух или более серверов. Также пользователь в такой системе должен знать, на каком именно сервере какая часть данных содержится.

2. Клиент может получать доступ к любому количеству серверов одновременно (т.е. за один запрос можно получить комбинированные данные двух или более серверов). В этом случае

серверы рассматриваются клиентом с логической точки зрения как единый сервер, и пользователь может не знать, на каком именно компьютере какая часть данных содержится. Подобную систему обычно и называют распределённой системой баз данных.

Полная поддержка распределённых баз данных означает, что отдельное приложение может «прозрачно» обрабатывать данные, распределённые на множестве различных баз данных, управление которыми осуществляют разные СУБД, работающие на соединенных коммуникационными сетями компьютерах разных типов с различными операционными системами.

Вопросы и задания для самопроверки

1. Дайте определения понятиям информации, данных, базы данных, базы знаний, СУБД, ИС.
2. Что такое архитектура ANSI/SPARC?
3. Охарактеризуйте внешний уровень архитектуры ANSI/SPARC.
4. Охарактеризуйте внутренний уровень архитектуры ANSI/ SPARC.
5. Охарактеризуйте концептуальный уровень архитектуры ANSI/ SPARC.
6. Перечислите функции СУБД.
7. Охарактеризуйте взаимодействие в архитектуре «клиент-сервер».
8. Охарактеризуйте распределенную обработку данных.

Рекомендуемая литература

1. Громов, Ю.Ю. Управление данными [Текст]: учеб. пособие / Ю.Ю. Громов, О.Г. Иванова, В.Н. Точка. – Тамбов: Изд-во Тамб. гос. техн. ун-та, 2009. – 80 с.
2. Дейт, К.Дж. Введение в системы баз данных [Текст] / К.Дж. Дейт. – М.: Издательский дом «Вильямс», 2006. – 1 328 с.
3. Королева, О.Н. Базы данных [Электронный ресурс]: курс лекций / О.Н. Королева, А.В. Мажукин, Т.В. Королева. – Электрон. текстовые данные. – М.: Московский гуманитарный университет, 2012. – 66 с. – Режим доступа: http://www.iprbookshop.ru/14515. – ЭБС «IPRbooks», по паролю.
4. Кузнецов, С.Д. Базы данных. Вводный курс [Электронный ресурс] / С.Д. Кузнецов. – Режим доступа: http://citforum.ru/database/advanced_intro/ (01.12.2014).
5. Кузнецов, С.Д. Основы современных баз данных [Электронный ресурс] / С.Д. Кузнецов. – Режим доступа: http://citforum.ru/database/osbd/contents.shtml (01.12.2014).
6. Кузовкин, А.В. Управление данными [Текст]: учебник для студ. высших учеб. заведений / А.В. Кузовкин, А.А. Цыганов, Б.А. Щукин. – М.: Издательский центр «Академия», 2010. – 256 с.
7. Ульман, Дж. Введение в системы баз данных [Текст] / Дж. Ульман, Дж. Уидом. – М.: Лори, 2006. – 379 с.
8. Хомоненко, А.Д. Базы данных [Текст]: учебник для высш. учеб. заведений / А.Д. Хомоненко. – СПб.: КОРОНА – Век, 2009. – 736 с.
9. Швецов, В.И. Базы данных [Электронный ресурс] / В.И. Швецов. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2009. – 155 с. – Режим доступа: http://www.iprbookshop.ru/16688. – ЭБС «IPRbooks», по паролю.

2. Реляционные базы данных

2.1. Реляционная модель данных

Понятие **реляционный** (от англ. relation – отношение) связано с разработками известного американского специалиста в области систем баз данных Эдгара Франка Кодда.

Реляционная модель данных (РМД) характеризуется простотой структуры данных, удобным для пользователя табличным представлением и возможностью использования формального аппарата алгебры отношений и реляционного исчисления для обработки данных.

РМД некоторой предметной области представляет собой набор отношений, изменяющихся во времени. При создании информационной системы совокупность отношений позволяет хранить данные об объектах предметной области и моделировать связи между ними.

Элементы РМД и формы их представления приведены в табл. 1.

Таблица 1

Элементы РМД

Элемент реляционной модели	Форма представления
Отношение	Таблица
Схема отношения	Строка заголовков таблицы
Кортеж	Строка таблицы
Сущность	Набор свойств объекта
Атрибут	Заголовок столбца таблицы
Первичный ключ	Один или несколько атрибутов
Тип данных	Тип значений элементов таблицы

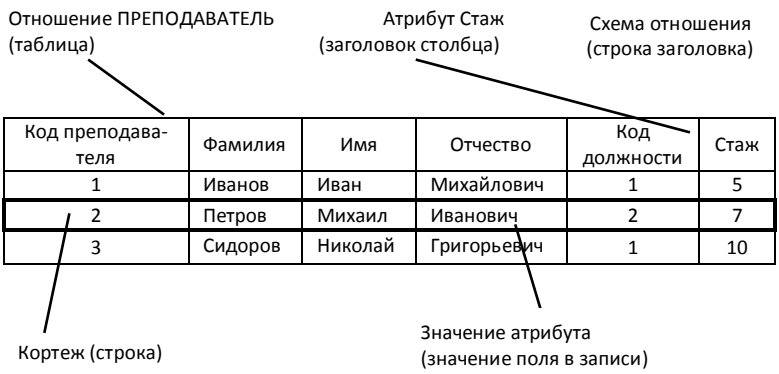
Важнейшим является понятие **отношения**, которое представляет собой двумерную таблицу, содержащую некоторые данные.

Сущность есть объект любой природы, данные о котором хранятся в базе данных. Данные о сущности хранятся в отношении.

Атрибуты представляют собой свойства, характеризующие сущность.

Математически **отношение** можно описать следующим образом. Пусть даны N множеств D1, D2, ... DN, тогда R есть отношение над этими множествами, если R есть множество упорядоченных n-кортежей вида <d1, d2, ... dn>, где d1 – элемент из D1, d2 – элемент из D2, … и dn – элемент из DN. D1, D2, ... DN называются доменами отношения R.

На рис. 3. приведен пример представления отношения ПРЕПОДАВАТЕЛЬ.



Представление отношения ПРЕПОДАВАТЕЛЬ

Множество всех допустимых значений каждого атрибута отношения образует домен. Отношение ПРЕПОДАВАТЕЛЬ включает 6 доменов. Домен 1 содержит коды всех преподавателей, домен 2 – фамилии всех преподавателей, домен 3 – имена всех преподавателей, домен 4 – отчества всех преподавателей, домен 5 – коды всех должностей, домен 6 – стаж работы преподавателей. Каждый домен образует значения одного типа, например, числовые или символьные данные.

Отношение ПРЕПОДАВАТЕЛЬ содержит 3 кортежа. Кортеж рассматриваемого отношения состоит из 6-и элементов, каждый из которых выбирается из соответствующего домена. Каждому кортежу соответствует строка таблицы.

Схема отношения представляет собой список имен атрибутов. Например, для приведенного примера схема отношения имеет вид ПРЕПОДАВАТЕЛЬ (Код преподавателя, Фамилия, Имя, Отчество, Код должности, Стаж).

Число столбцов в отношении называется **степенью** (кардинальным числом). Текущее число кортежей в отношении называется **мощностью**. Степень отношения обычно не изменяется после создания отношения, но мощность будет колебаться по мере добавления новых и удаления старых кортежей.

Ключом отношения, или **первичным ключом**, называется атрибут отношения (набор атрибутов), однозначно идентифицирующий каждый из его кортежей. Например, в отношении ПРЕПОДАВАТЕЛЬ (Код преподавателя, Фамилия, Имя, Отчество, Код должности, Стаж) ключевым является атрибут **Код преподавателя**. В дальнейшем первичный ключ в отношении будем подчеркивать – Код преподавателя, а при записи брать в угловые скобки – <Код преподавателя>.

Первичный ключ должен обладать следующими свойствами:

- иметь уникальные значения;
- не содержать пустых (неопределенных) значений;

- быть компактным, т.е. содержать только такие атрибуты, удаление любого из которых может привести к утрате уникальности.

В качестве первичного ключа может использоваться:

- естественный ключ – набор атрибутов сущности, уникально её идентифицирующий (например, при отсутствии в отношении ПРЕПОДАВАТЕЛЬ атрибута *Код преподавателя*, роль первичного ключа мог сыграть набор естественных атрибутов: *Фамилия, Имя, Отчество*);
- суррогатный ключ – автоматически сгенерированное поле, никак не связанное с информационным содержанием записи. Обычно в роли суррогатного ключа выступает автоинкрементное поле целого типа.

Существует два мнения:

- Суррогатные ключи должны использоваться, только если естественных ключей не существует. Если же естественный ключ существует, то идентификация записи внутри БД осуществляется по имеющемуся естественному ключу.
- Суррогатные ключи должны добавляться в любую таблицу, на которую существуют ссылки из других таблиц, и связи между таблицами должны организовываться только при помощи суррогатных ключей. Поиск записи и представление её пользователю по-прежнему производятся на основании естественного ключа.

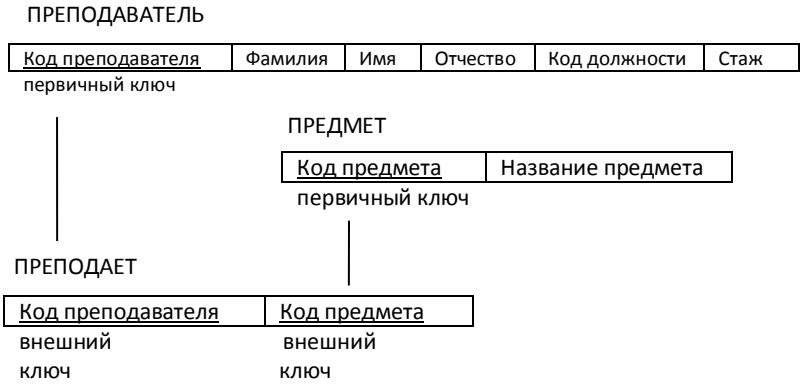
Достоинства суррогатных ключей:

- Упрощение сопровождения. Поскольку операции связи между таблицами отделены от логики «внутри таблиц», и то и другое можно менять независимо и не затрагивая остального.
- Уменьшение размера БД.
- Увеличение скорости выборки данных.
- Увеличение скорости обновления данных.

При организации связи между двумя отношениями одно из них будет являться родительским (главным), а другое – дочерним (подчиненным). Таким образом, в родительском отношении описана родительская сущность, в дочернем – дочерняя.

Внешний ключ существует только для дочерней сущности и является ссылкой на значение ключа родительской сущности. При создании связей (отношений) между сущностями в дочернюю сущность передаются атрибуты, составляющие первичный ключ родительской сущности. Эти атрибуты и составляют внешний ключ.

С помощью внешних ключей устанавливаются связи между отношениями. Например, имеются два отношения ПРЕПОДАВАТЕЛЬ (Код преподавателя, Фамилия, Имя, Отчество, Код должности, Стаж) и ПРЕДМЕТ (Код предмета, Название предмета), которые связываются отношением ПРЕПОДАЕТ (Код преподавателя, Код предмета) (Рис. 4). В связующем отношении атрибуты *Код преподавателя* и *Код предмета* образуют составной первичный ключ. Эти же атрибуты представляют собой внешние ключи, являющиеся первичными ключами других отношений.



Связь отношений

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.

Условия и ограничения, накладываемые на отношения, выполнение которых позволяет таблицу считать отношением:

1. Все строки таблицы должны быть уникальны, т.е. не может быть строк с одинаковыми первичными ключами.
2. В таблице не должно быть столбцов с повторяющимися именами.
3. Все строки одной таблицы должны иметь одну структуру, соответствующую именам и типам столбцов.
4. Имена столбцов таблицы должны быть различны, а значения их простыми, т.е. не допустима группа значений в одном столбце одной строки.
5. Порядок размещения строк в таблице может быть произвольным.

К отношениям можно применять систему операций, позволяющую получать одни отношения из других. Например, результатом запроса к реляционной БД может быть новое отношение, вычисленное на основе хранящихся в базе отношений. Отсюда появляется возможность разделить обрабатываемые данные на хранимую и вычисляемую части.

Основной единицей обработки данных в реляционных БД является отношение, а не отдельные его кортежи (записи), как это принято в традиционных языках программирования.

Операции, выполняемые над отношениями, можно разделить на две группы.

Первую группу составляют операции над множествами: объединение, пересечение, разность, деление и декартово произведение.

Вторую группу составляют специальные операции над отношениями: проекция, соединение, выбор.

В реляционных СУБД для выполнения операций над отношениями используют две группы языков, имеющие в качестве своей математической основы реляционную алгебру и реляционное исчисление соответственно.

В реляционной алгебре операнды и результаты всех действий являются отношениями. Языки реляционной алгебры являются процедурными, так как отношение, являющееся результатом запроса к реляционной БД, вычисляется при выполнении последовательности операций над хранимыми в ней отношениями. В основном языки СУБД являются процедурными.

Языки реляционного исчисления являются непроцедурными. Запрос к БД, выполненный с использованием подобного языка, содержит лишь информацию о желаемом результате. Для этих языков характерно наличие наборов правил для записи запросов. В частности, к языкам этой группы относится SQL.

Между реляционной алгеброй и реляционным исчислением существует связь с помощью так называемой процедуры редукции, которая сводит любое выражение реляционного исчисления к набору стандартных операций реляционной алгебры и наоборот.

Сейчас основным предметом критики РМД является некоторая ограниченность при использовании в нетрадиционных областях (системы автоматизации проектирования), в которых требуются предельно сложные структуры данных.

Выделяют ещё один недостаток: невозможность адекватного отражения семантики предметной области (возможности представления знаний о семантической специфике предметной области в реляционных системах очень ограничены).

Их устранение привело к созданию объектно-ориентированных баз данных.

2.2. Типы данных

Практически все СУБД поддерживают следующие типы данных:

- целочисленные;
- вещественные;
- строковые;
- специализированные типы данных для денежных величин;
- специальные типы данных для временных величин (дата и/или время);
- типы двоичных объектов.

Наименьшая единица данных реляционной модели – это отдельное атомарное (неразложимое) для данной модели значение данных.

Пустое значение (NULL) – это не ноль и не пустая строка, а неизвестное значение атрибута, которое не определено в данный момент времени и в принципе может быть определено позднее.

2.3. Условия целостности данных

Чтобы информация, хранящаяся в базе данных, была однозначной и непротиворечивой, в реляционной модели устанавливаются некоторые ограничительные условия.

Ограничительные условия – это правила, определяющие возможные значения данных.

Они обеспечивают логическую основу для поддержания корректных значений данных в базе. Ограничения целостности позволяют свести к минимуму ошибки, возникающие при обновлении и обработке данных.

Категорийная целостность. Кортежи представляют в базе данных элементы определенных объектов реального мира (категорий). Первичный ключ таблицы однозначно определяет каждый кортеж (каждый элемент категории).

Для извлечения данных, содержащихся в строке таблицы или для манипулирования этими данными необходимо знать значение ключа для этой строки.

Поэтому строка не может быть занесена в базу данных до тех пор, пока не будут определены все атрибуты ее первичного ключа.

Никакой атрибут первичного ключа строки не может быть пустым.

Ссылочная целостность. Если две таблицы связаны между собой, то внешний ключ таблицы должен содержать только те значения, которые уже имеются среди значений ключа, по которому осуществляется связь.

Если корректность значений внешних ключей не контролируется СУБД, то может нарушиться ссылочная целостность данных.

Для соблюдения целостности сущности достаточно гарантировать отсутствие в любом отношении кортежей с одним и тем же значением первичного ключа.

При обновлении ссылающегося отношения достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа.

При удалении кортежа из отношения, на которое ведет ссылка, возможно использовать один из трех подходов, каждый из которых поддерживает целостность по ссылкам:

1. Запрещается производить удаление кортежа, на который существуют ссылки (то есть сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа).

2. При удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным.
3. При удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи (каскадное удаление).

2.4. Основные свойства отношений

Отсутствие упорядоченности кортежей. В таблицах реляционной базы данных информация хранится в неупорядоченном виде. Упорядочивание в принципе не поддерживается СУБД, и такое понятие, как порядковый номер кортежа, не имеет никакого смысла.

Отсутствие упорядоченности атрибутов. Атрибуты отношений также не упорядочены. Для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута.

Атомарность значений атрибутов. Значения всех атрибутов являются атомарными. Это следует из определения домена как потенциального множества значений простого типа данных, то есть среди значений домена не могут содержаться множества значений (отношения).

Реляционная база данных – это совокупность отношений, содержащих всю информацию, которая должна храниться в базе данных.

Набор средств для управления подобным хранилищем называется **реляционной системой управления базами данных**.

Свойства таблиц реляционной базы данных:

- каждая таблица состоит из однотипных строк и имеет уникальное имя;

- строки имеют фиксированное число полей (столбцов) и значений (множественные поля и повторяющиеся группы недопустимы);
- строки таблицы обязательно отличаются друг от друга хотя бы единственным значением, что позволяет однозначно идентифицировать любую строку;
- столбцам таблицы присваиваются уникальные имена, и в каждом из них размещаются однородные значения данных;
- полное информационное содержание базы данных представляется в виде явных значений данных, и такой метод представления является единственным;
- при выполнении операций с таблицей ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их информационному содержанию.

2.5. Индексы

Индекс представляет собой указатель на данные, размещенные в реляционной таблице. Индексы используются для ускорения поиска информации. При добавлении в таблицу новых записей или удалении существующих индекс модифицируется.

Ускорение поиска информации при использовании индекса достигается за счет того, что:

- обращение к индексу выполняется быстрее, чем к таблице;
- в индексе записи хранятся в упорядоченном виде.

Простой индекс строится на основе только одного столбца реляционной таблицы.

Составные индексы строятся по двум и более столбцам реляционной таблицы.

Условия оптимальности следования столбцов в составном индексе:

- первым следует помещать столбец, содержащий наиболее ограничивающее значение (то есть содержащий меньшее количество повторов);
- первым следует помещать столбец, содержащий данные, которые наиболее часто задаются в условиях поиска.

Эти условия противоречивые, необходим компромисс.

Обычно не следует индексировать:

- столбцы, данные в которых подвержены частому изменению;
- столбцы, содержащие большое количество пустых значений;
- столбцы, содержащие небольшое количество уникальных значений;
- небольшие таблицы;
- поля большого размера.
- **Уникальные индексы** не допускают введения в таблицу дублирующих значений.

Вопросы и задания для самопроверки

1. Охарактеризуйте элементы РМД.
2. Что такое первичный ключ?
3. Охарактеризуйте естественные и суррогатные ключи.
4. Что такое внешний ключ?
5. В каком случае таблица считается отношением?
6. Назовите основные типы данных СУБД.
7. Что понимается под целостностью данных.
8. Перечислите основные свойства отношений.
9. Для чего применяются индексы?

Рекомендуемая литература

1. Громов, Ю.Ю. Управление данными [Текст]: учеб. пособие / Ю.Ю. Громов, О.Г. Иванова, В.Н. Точка. – Тамбов: Изд-во Тамб. гос. техн. ун-та, 2009. – 80 с.
2. Дейт, К.Дж. Введение в системы баз данных [Текст] / К.Дж. Дейт. – М.: Издательский дом «Вильямс», 2006. – 1 328 с.
3. Королева, О.Н. Базы данных [Электронный ресурс]: курс лекций / О.Н. Королева, А.В. Мажукин, Т.В. Королева. – Электрон. текстовые данные. – М.: Московский гуманитарный университет, 2012. – 66 с. – Режим доступа: http://www.iprbookshop.ru/14515. – ЭБС «IPRbooks», по паролю.
4. Кузнецов, С.Д. Основы современных баз данных [Электронный ресурс] / С.Д. Кузнецов. – Режим доступа: http://citforum.ru/database/osbd/contents.shtml (01.12.2014).
5. Кузнецов, С.Д. Базы данных. Вводный курс [Электронный ресурс] / С.Д. Кузнецов. – Режим доступа: http://citforum.ru/database/advanced_intro/ (01.12.2014).
6. Кузовкин, А.В. Управление данными [Текст]: учебник для студ. высших учеб. заведений / А.В. Кузовкин, А.А. Цыганов, Б.А. Щукин. – М.: Издательский центр «Академия», 2010. – 256 с.
7. Пушников, А.Ю. Введение в системы управления базами данных. [Электронный ресурс] / А.Ю. Пушников. – Режим доступа: http://citforum.ru/database/dblearn/index.shtml (01.12.2014).
8. Ульман, Дж. Введение в системы баз данных [Текст] / Дж. Ульман, Дж. Уидом. – М.: Лори, 2006. – 379 с.
9. Хомоненко, А.Д. Базы данных [Текст]: учебник для высш. учеб. заведений / А.Д. Хомоненко. – СПб.: КОРОНА – Век, 2009. – 736 с.
10. Швецов, В.И. Базы данных [Электронный ресурс] / В.И. Швецов. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2009. – 155 с. – Режим доступа: http://www.iprbookshop.ru/16688. – ЭБС «IPRbooks», по паролю.

3. Проектирование баз данных

Перед проектировщиком базы данных возникают две основные проблемы:

1. Проблема логического проектирования баз данных.

Каким образом перевести объекты предметной области в абстрактные объекты модели данных, чтобы этот перевод не противоречил семантике предметной области и был, по возможности, лучшим вариантом?

2. Проблема физического проектирования баз данных.

Как обеспечить эффективность выполнения запросов к базе данных, т.е. каким образом, учитывая специфику конкретной СУБД, расположить данные во внешней памяти, какие дополнительные структуры (например, индексы) потребовать создать и т.д.?

При физическом проектировании слишком много зависит от используемой СУБД. В ряде случаев пользователю предоставляется возможность настройки отдельных параметров системы, что не составляет большой проблемы.

Логическое проектирование заключается в определении числа и структуры таблиц, формировании запросов к БД, определении типов отчетных документов, разработке алгоритмов обработки информации, создании форм для ввода и редактирования данных в базе и решении ряда аналогичных задач.

Решение задач логического проектирования БД в основном определяется спецификой задачи предметной области. Наиболее важной здесь является проблема структуризации данных. Классическим и исторически первым является подход, связанный со сбором информации об объектах решаемой задачи

в рамках одного отношения и последующей декомпозиции его на несколько взаимосвязанных отношений на основе процедуры нормализации.

3.1. Цели проектирования

В чем заключается желаемый конечный результат процесса проектирования РБД? Среди множества целей, стоящих перед проектированием, следующие представляются наиболее важными.

1. Возможность хранения всех необходимых данных в БД

БД должна содержать все данные, представляющие интерес для организации, так что при проектировании следует предусмотреть возможность размещения в БД всех необходимых данных. Первым шагом в процессе проектирования является определение всех атрибутов, которые впоследствии будут помещены в БД.

2. Исключение избыточности данных

Необходимо уяснить четкое различие между дублированием данных и избыточным дублированием данных. Рассмотрим поясняющий пример. В отношение ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ (табл. 2) содержатся данные, указывающие должность каждого преподавателя. Конкретные значения должностей могут неоднократно появляться в отношении. В действительности же они появляются один раз для каждого преподавателя. Поэтому наличие повторяющихся значений должностей не является избыточным. Действительно, при удалении одного из значений должностей будет утеряна информация о том, какую должность занимает преподаватель. В табл. 3 показано отношение ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ после удаления дублированных значений должностей (вместо значений должностей Егорова Е.Е. и Иванова И.И. поставлены «прочерки» – неопределенные значения).

Таблица 2

Отношение ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ

Фамилия Имя Отчество	Должность
Васильев В.В.	Доцент
Егоров Е.Е.	Доцент
Иванов И.И.	Доцент
Петров П.П.	Ст. преп.
Сидоров С.С.	Ассистент

Таблица 3

Отношение ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ
после удаления дублированных значений должностей

Фамилия Имя Отчество	Должность
Васильев В.В.	Доцент
Егоров Е.Е.	----------
Иванов И.И.	----------
Петров П.П.	Ст. преп.
Сидоров С.С.	Ассистент

Пример избыточного дублирования представляет приведенное в табл. 4 отношение ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ–ОКЛАД, которое в отличие от отношения ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ дополнено атрибутом *Оклад*. Естественно предположить, что все преподаватели с одним и тем же значением должности имеют один и тот же размер оклада. Следовательно, в рассматриваемом отношении имеется избыточное дублирование данных. Так, в связи с тем что Егоров Е.Е. и Иванов И.И. занимают должность доцента, ту же, что и Васильев В.В., то их оклад можно узнать из кортежа со сведениями о Васильеве В.В.

В табл. 5 приведен пример неудачного отношения ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ–ОКЛАД, в котором вместо окладов Егорова Е.Е. и Иванова И.И. поставлены «прочерки» (неопределенные значения).

Таблица 4

Отношение ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ–ОКЛАД

Фамилия Имя Отчество	Должность	Оклад
Васильев В.В.	Доцент	160
Егоров Е.Е.	Доцент	160
Иванов И.И.	Доцент	160
Петров П.П.	Ст. преп.	120
Сидоров С.С.	Ассистент	100

Таблица 5

Отношение ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ–ОКЛАД после удаления дублированных значений окладов

Фамилия Имя Отчество	Должность	Оклад
Васильев В.В.	Доцент	160
Егоров Е.Е.	Доцент	---
Иванов И.И.	Доцент	---
Петров П.П.	Ст. преп.	120
Сидоров С.С.	Ассистент	100

Неудачность подобного способа исключения избыточности заключается в следующем. Во-первых, при программировании придется потратить дополнительные усилия на создание механизма поиска информации для прочерков в таблицы. Во-вторых, память все равно выделяется под атрибуты с прочерками, хотя дублирование данных и исключено. В-третьих, что особенно важно, при исключении из коллектива Васильева В.В.

кортеж со сведениями о нем будет исключен из отношения, а значит, уничтожена информация об окладе для должности доцент, что недопустимо.

Возможный способ выхода из данной ситуации заключается в декомпозиции исходного отношения ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ–ОКЛАД на два отношения ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ (табл. 6) и ДОЛЖНОСТЬ–ОКЛАД (табл. 7). Первое из них содержит информацию о должностях преподавателей, а второе – информацию об окладах должностей. Теперь, если Васильева В.В. и уволят из организации и, как следствие этого, удалят всякую информацию о нем из базы данных организации, то это не приведет к утере информации об окладе должности доцент.

Процедура декомпозиции отношения ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ–ОКЛАД на два отношения ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ и ДОЛЖНОСТЬ–ОКЛАД является основной процедурой нормализации отношений.

Таблица 6

Отношение ПРЕПОДАВАТЕЛЬ–ДОЛЖНОСТЬ

Фамилия Имя Отчество	**Должность**
Васильев В.В.	Доцент
Егоров Е.Е.	Доцент
Иванов И.И.	Доцент
Петров П.П.	Ст. преп.
Сидоров С.С.	Ассистент

Таблица 7

Отношение ДОЛЖНОСТЬ–ОКЛАД

Должность	Оклад
Доцент	160
Ст. преп.	120
Ассистент	100

3. Сведение числа хранимых в БД отношений к минимуму.

Эта цель обусловлена тем, что разбиение одного отношения на два или более меньших отношений желательно с точки зрения исключения определенных проблем, но это неудобно для пользователя. Таким образом, нельзя допускать неограниченный рост числа отношений.

4. Нормализация отношений.

Для некоторых отношений очень важна проблема удаления и обновления (например, рассмотренная в цели 2 потеря наименования должности преподавателя). Проектировщик должен уметь обнаруживать эти потенциально опасные отношения и «нормализовать их» посредством разбиения предписанным образом.

Цели 3 и 4 противоречат друг другу, поэтому здесь требуется взаимный компромисс. Это будет частью завершающего этапа проектирования.

3.2. Нормализация данных

Нормализация представляет собой процесс реорганизации данных путем ликвидации повторяющихся групп и иных противоречий с целью приведения таблиц к виду, позволяющему осуществлять непротиворечивое и корректное редактирование данных.

Цель нормализации – каждый факт должен храниться только в одном месте.

Избыточность информации устраняется не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных и упрощения управления ими.

Использование ненормализованных таблиц может привести к нарушению целостности данных (противоречивости информации) в базе данных.

Проблемы, возникающие при использовании ненормализованных таблиц:

- **избыточность данных** – проявляется в том, что в нескольких записях таблицы базы данных повторяется одна и та же информация;
- **аномалии обновления** – обнаруживается, когда обновление одного значения требует обновления нескольких строк;
- **аномалии удаления** – возникают при удалении записей из ненормализованной таблицы;
- **аномалии ввода** – возникают при добавлении в таблицу новых записей и обычно возникают, когда для некоторых полей таблицы заданы ограничения NOT NULL (не пустое значение).

Чтобы свести к минимуму возможность появления такого рода аномалий, и используется нормализация.

Теория нормализации основана на концепции **нормальных форм**.

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

Выделяют следующую последовательность нормальных форм:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (3НФ);

- усиленная третья нормальная форма или нормальная форма Бойса-Кодда (НФБК);
- четвертая нормальная форма (4НФ);
- пятая нормальная форма, или нормальная форма проекции-соединения (5НФ или НФПС).

Впоследствии были введены в пользование:

- доменно-ключевая нормальная форма (ДКНФ);
- шестая нормальная форма (6НФ).

Основные **свойства** нормальных форм:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

В основе процесса проектирования лежит метод нормализации – декомпозиция отношения, находящегося в предыдущей нормальной форме, на два или более отношения, удовлетворяющих требованиям следующей нормальной формы.

Декомпозиция отношения должна обеспечить то, что запросы (выборка данных по условию) к исходному отношению и к отношениям, получаемым в результате декомпозиции, дадут одинаковый результат.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии **функциональной зависимости**.

Функционально зависимым считается такой атрибут, значение которого однозначно определяется значением другого атрибута.

Функционально зависимые атрибуты обозначаются следующим образом: $X \rightarrow Y$.

Эта запись означает, что если два кортежа в таблице имеют одно и то же значение атрибута *X*, то они имеют одно и то же значение атрибута *Y*.

Атрибут, указываемый в левой части функциональной зависимости, называется **детерминантом**.

Первичный ключ таблицы является детерминантом, так как его значение однозначно определяет значение любого атрибута таблицы.

Основной операцией метода является **операция проекции**. Предположим, что в отношении R (A, B, C, D, E, ...) устранение функциональной зависимости $C \rightarrow D$ позволит перевести его в следующую нормальную форму. Для решения этой задачи выполним декомпозицию отношения R на два новых отношения R1 (A, B, C, E, ...) и R2 (C, D). Отношение R2 является проекцией отношения R на атрибуты *C* и *D*.

3.2.1. Первая нормальная форма

Ограничение первой нормальной формы – значения всех атрибутов отношения должны быть атомарными. Данное требование является базовым требованием классической реляционной модели данных, поэтому любая реляционная таблица по определению уже находится в первой нормальной форме.

3.2.2. Вторая нормальная форма

Отношение находится во второй нормальной форме в том и только в том случае, когда это отношение находится в первой нормальной форме и каждый неключевой атрибут полностью зависит от первичного ключа.

Неключевым называется любой атрибут отношения, не входящий в состав первичного ключа.

Чтобы перейти от первой нормальной формы ко второй, нужно выполнить следующие шаги:

1. Определить, на какие части можно разбить первичный ключ, так чтобы некоторые из неключевых полей зависели от одной из этих частей (причем эти части могут содержать несколько атрибутов).
2. Создать новые таблицы для каждой такой части ключа и группы зависящих от нее полей. Часть бывшего первичного ключа станет при этом первичным ключом новой таблицы.
3. Удалить из исходной таблицы поля, перемещенные в другие таблицы, кроме тех из них, которые станут внешними ключами.

3.2.3. Третья нормальная форма

Функциональная зависимость атрибутов *X* и *Y* отношения R называется **транзитивной**, если существует такой атрибут *Z*, что имеются функциональные зависимости $X \rightarrow Z$ и $Z \rightarrow Y$, но отсутствует функциональная зависимость $Z \rightarrow X$.

Отношение R находится в третьей нормальной форме в том и только в том случае, если оно находится во второй нормальной форме и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Чтобы перейти от второй нормальной формы к третьей, нужно выполнить следующие шаги:
1. Определить все поля (или группы полей), от которых зависят другие поля.
2. Создать новые таблицы для каждого такого поля (или группы полей) и группы, зависящих от него полей. Поле (или группа полей), от которого зависят все остальные перемещенные поля, станет при этом первичным ключом новой таблицы.
3. Удалить перемещенные поля из исходной таблицы, оставив лишь те из них, которые станут внешними ключами.

3.2.4. Нормальная форма Бойса-Кодда

На практике построение третьей нормальной формы схем отношений в большинстве случаев является достаточным. Процесс проектирования реляционной БД заканчивается после приведения отношений к третьей нормальной форме. Если же в отношении имеется зависимость атрибутов составного ключа от неключевых атрибутов, то необходимо перейти к усиленной 3НФ или нормальной форме Бойса-Кодда.

Отношение находится в НФБК, если каждый детерминант отношения является возможным ключом.

Эдгар Франк Кодд доказал утверждение о том, что большинство потенциальных аномалий в БД будет устранено в случае декомпозиции каждого отношения в **нормальную форму Бойса-Кодда**. Также им предложен перевод отношения из 1НФ в НФБК (алгоритм декомпозиции).

Алгоритм декомпозиции:

1. Построение универсального отношения для БД.
2. Определение всех функциональных зависимостей, существующих между атрибутами универсального отношения.
3. Проверка отношения на НФБК. Если отношение находится в НФБК, то процесс декомпозиции заканчивается, иначе необходимо выделить самую крайнюю функциональную зависимость в новое отношение и повторить пункт 3 для получившегося нового отношения и для оставшихся.

3.3. Универсальное отношение

Классический подход к проектированию БД начинается с определения всех объектов, сведения о которых будут включены в базу, и определения их атрибутов. Затем атрибуты сводятся в одну таблицу – универсальное отношение.

Пример

Предположим, что для учебной части факультета создается БД о преподавателях. На первом этапе проектирования БД в результате общения с заказчиком должны быть определены содержащиеся в базе сведения о том, как она должна использоваться и какую информацию заказчик хочет получать в процессе ее эксплуатации. В результате устанавливаются атрибуты, которые должны содержаться в отношениях БД, связи между ними. Перечислим имена выделенных атрибутов и их краткие характеристики.

ФИО – фамилия, имя, отчество преподавателя.

Должность – должность, занимаемая преподавателем.

Оклад – оклад преподавателя.

Кафедра – наименование кафедры, на которой числится преподаватель.

Предмет – наименование предмета (дисциплины), читаемого преподавателем.

Группа – наименование учебной группы, в которой преподаватель проводит занятия.

Вид занятий – вид занятий, проводимых преподавателем в учебной группе.

Часы – количество часов, проводимых преподавателем указанного вида занятий по данному предмету в данной учебной группе.

В табл. 8 представлен образец данных, концептуализированных учебной частью для их хранения в БД, но указанная таблица отношением не является.

Для иллюстрации того, почему табл. 8 не является отношением, выделим одну строку из неё в табл. 9. В табл. 9 значения полей *ФИО, Должность, Оклад, Кафедра* – атомарные,

в то время как значения в полях *Предмет*, *Группа*, *Вид занятий* и *Часы* – множественные. Данная строка очевидным образом отличается по форме от кортежей, представленных в простых отношениях, рассмотренных выше. Отличие в том, что не все поля строки содержат атрибуты, значения которых атомарные.

Указанное отношение имеет следующую схему: ПРЕПОДАВАТЕЛЬ (ФИО, Должность, Оклад, Кафедра, Предмет, Группа, Вид занятий, Часы).

Табл. 10 представляет собой экземпляр корректного отношения. Его называют **универсальным отношением** проектируемой БД. В одно универсальное отношение включаются все представляющие интерес атрибуты, и оно может содержать все данные, которые предполагается размещать в БД в будущем. Для малых БД универсальное отношение может использоваться в качестве отправной точки при проектировании БД.

3.4. Проблемы, вызываемые использованием единственного отношения

Начинающий проектировщик будет использовать отношение ПРЕПОДАВАТЕЛЬ (табл. 10) в качестве завершенной БД. Это выглядит достаточно последовательным. Зачем разбивать отношение ПРЕПОДАВАТЕЛЬ на несколько более мелких отношений, если оно способно заключать в себе все данные? Существует несколько причин, почему не следует использовать данное отношение в качестве единственного в БД. Это обусловлено тем, как будет использоваться БД и какое воздействие на данные в отношении ПРЕПОДАВАТЕЛЬ будут оказывать определенные операции. Различаются три специфичные проблемы: про-

блема, связанная с обновлением (модификацией) данных в БД; проблема, обусловленная необходимостью удаления кортежей; проблема, обусловленная необходимостью включения (добавления) новых кортежей. Выделенные проблемы обычно называют аномалиями вставки, удаления и обновления, понимая под аномалией отклонение от нормы.

3.4.1. Проблема добавления

Если у учебной части появляется новый преподаватель, еще не преподающий никакой предмет, для него необходимо включить в БД кортеж с нулевыми (пустыми) значениями атрибутов Предмет, Группа и Вид занятий. Нулевых значений следует избегать. Следовательно, включение в БД нового преподавателя невозможно вплоть до назначения ему предмета.

В табл. 11 показан пример того, как будет выглядеть отношение ПРЕПОДАВАТЕЛЬ в случае принудительного включения в него информации о преподавателе, не преподающем ни одного предмета.

Пусть требуется найти список преподавателей, проводящих занятия в группах (на языке SQL текст запроса выглядел бы следующим образом: *SELECT DISTINCT ФИО, Группа FROM ПРЕПОДАВАТЕЛЬ*). В число таких преподавателей попадет Топоров В.В., хотя он не преподает ни в одной группе (Примечание. Конечно, этого можно избежать, применив в запросе, условие на выборку: *SELECT DISTINCT ФИО, Группа FROM ПРЕПОДАВАТЕЛЬ WHERE Группа <> 0.* Но это требует дополнительных усилий со стороны пользователя).

Таблица 8

Образец концептуализированных данных

ФИО	Должность	Оклад	Кафедра	Предмет	Группа	Вид занятий	Часы
Егоров Е.Е.	Доцент	160	Кафедра экономики	Экономика организации	191	Лекции	28
				Экономика организации	191	Практики	44
Иванов И.И.	Доцент	160	Кафедра информатики	Управление данными	191	Лекции	30
				Управление данными	191	Практики	50
				Операционные системы	192	Лекции	24
Петров П.П.	Ст. преп.	120	Кафедра информатики	Управление данными	191	Практики	50
				Операционные системы	192	Практики	36
Сидоров С.С.	Ассистент	100	Кафедра экономики	Экономика организации	191	Практики	44

Таблица 9

Данные о преподавателе Иванов И.И.

ФИО	Должность	Оклад	Кафедра	Предмет	Группа	Вид занятий	Часы
Иванов И.И.	Доцент	160	Кафедра информатики	Управление данными	191	Лекции	30
				Управление данными	191	Практики	50
				Операционные системы	192	Лекции	24

Таблица 10

Универсальное отношение ПРЕПОДАВАТЕЛЬ

ФИО	Должность	Оклад	Кафедра	Предмет	Группа	Вид занятий	Часы
Егоров Е.Е.	Доцент	160	Кафедра экономики	Экономика организации	191	Лекции	28
Егоров Е.Е.	Доцент	160	Кафедра экономики	Экономика организации	191	Практики	44
Иванов И.И.	Доцент	160	Кафедра информатики	Управление данными	191	Лекции	30
Иванов И.И.	Доцент	160	Кафедра информатики	Управление данными	191	Практики	50
Иванов И.И.	Доцент	160	Кафедра информатики	Операционные системы	192	Лекции	24
Петров П.П.	Ст. преп.	120	Кафедра информатики	Управление данными	191	Практики	50
Петров П.П.	Ст. преп.	120	Кафедра информатики	Операционные системы	192	Практики	36
Сидоров С.С.	Ассистент	100	Кафедра экономики	Экономика организации	191	Практики	44

Таблица 11

Добавление нового преподавателя

ФИО	Должность	Оклад	Кафедра	Предмет	Группа	Вид занятий	Часы
Егоров Е.Е.	Доцент	160	Кафедра экономики	Экономика организации	191	Лекции	28
Егоров Е.Е.	Доцент	160	Кафедра экономики	Экономика организации	191	Практики	44
Иванов И.И.	Доцент	160	Кафедра информатики	Управление данными	191	Лекции	30
Иванов И.И.	Доцент	160	Кафедра информатики	Управление данными	191	Практики	50
Иванов И.И.	Доцент	160	Кафедра информатики	Операционные системы	192	Лекции	24
Петров П.П.	Ст. преп.	120	Кафедра информатики	Управление данными	191	Практики	50
Петров П.П.	Ст. преп.	120	Кафедра информатики	Операционные системы	192	Практики	36
Сидоров С.С.	Ассистент	100	Кафедра экономики	Экономика организации	191	Практики	44
Топоров В.В.	Ассистент	100	Кафедра экономики	----------	---	----------	--

3.4.2. Проблема обновления

В отношении ПРЕПОДАВАТЕЛЬ содержится большое число избыточных данных. Избыточность данных всегда свидетельствует о возможности модификации только части требуемых данных с помощью операции обновления.

Отношение ПРЕПОДАВАТЕЛЬ характеризуется как явной, так и неявной избыточностью.

Явная избыточность заключается в том, что строки с данными о преподавателе, проводящих предметы в нескольких группах, повторяются соответствующее число раз. Например, в отношении ПРЕПОДАВАТЕЛЬ все данные по Иванову И.И. повторяются трижды. Поэтому если Иванов И.И. станет старшим преподавателем, то этот факт должен быть отражен в трех строках. В противном случае будет иметь место противоречие в данных, обусловленное явной избыточностью данных в отношении.

Неявная избыточность проявляется в одинаковых окладах у всех преподавателей, имеющих одинаковые должности. Поэтому если при изменении окладов за должность с 160 на 180 это значение изменят у всех преподавателей, кроме, например, Егорова Е.Е., то база станет противоречивой. Это пример аномалии для варианта с неявной избыточностью.

3.4.3. Проблема удаления

В экземпляре отношения ПРЕПОДАВАТЕЛЬ (табл. 10) присутствует только один кортеж, в котором значение атрибутов *ФИО* равно Сидоров С.С. Предположим, что учебная часть узнает, что Сидоров С.С. не преподает предмет «Экономика организации», как это отмечено, и удаляет этот кортеж из отношения. Поскольку это единственный кортеж с информацией об этом

преподавателе, его удаление приведет к исключению преподавателя из БД. Если учебная часть вслед за данной операцией удаления запросит список содержащихся в отношении ПРЕПОДАВАТЕЛЬ имен всех преподавателей, то Сидорова С.С. в этом списке уже не будет.

Средством исключения избыточности в отношениях и, как следствие, аномалий является нормализация отношений.

3.5. Приведение модели к требуемому уровню нормальной формы

Приведем наше отношение ПРЕПОДАВАТЕЛЬ к НФБК, согласно алгоритму декомпозиции.

1 шаг: Построение универсального отношения для БД

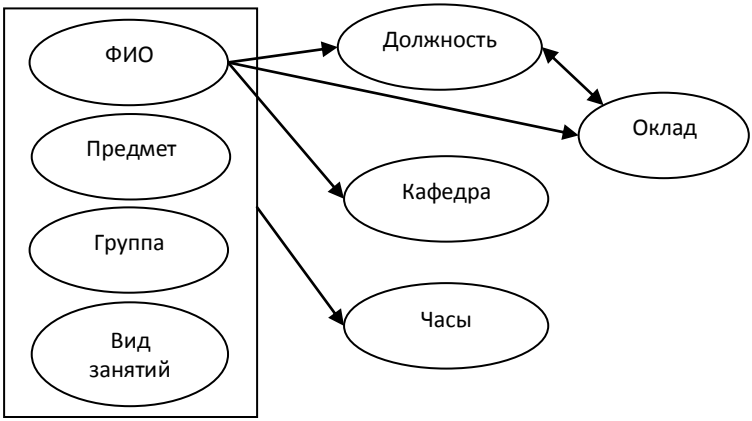
Универсальное отношение ПРЕПОДАВАТЕЛЬ показано в табл. 10.

2 шаг: Определение всех ФЗ, существующих между атрибутами универсального отношения.

В универсальном отношении ПРЕПОДАВАТЕЛЬ (ФИО, Должность, Оклад, Кафедра, Предмет, Группа, Вид занятий, Часы) определены следующие ФЗ:

- ФИО → Должность;
- ФИО → Оклад;
- ФИО → Кафедра;
- Должность → Оклад;
- Оклад → Должность;
- ФИО, Предмет, Группа, Вид занятий → Часы.

На рис. 5 показано графическое отображение ФЗ между атрибутами отношения ПРЕПОДАВАТЕЛЬ.



Графическое отображение ФЗ между атрибутами отношения ПРЕПОДАВАТЕЛЬ

К выделению этих функциональных зависимостей ФЗ для рассматриваемого примера приводят следующие соображения.

ФИО у преподавателей факультета уникальны. Каждый преподаватель имеет определенную должность (ассистент, ст. преп., доцент, профессор), но одну и ту же должность могут занимать несколько преподавателей, т.е. имеет место ФЗ: *ФИО → Должность*, а обратная ФЗ отсутствует. Каждый преподаватель является сотрудником одной и только одной кафедры. Поэтому ФЗ: *ФИО → Кафедра* имеет место. С другой стороны, на каждой кафедре много преподавателей, обратной ФЗ нет. Каждому преподавателю соответствует конкретный оклад, который одинаков для всех преподавателей с одинаковыми должностями, что учитывается зависимостями *ФИО → Оклад и Должность → Оклад*. Нет одинаковых окладов для разных должностей, поэтому имеет место ФЗ: *Оклад → Должность*.

Один и тот же преподаватель в одной группе по разным предметам и разным видам занятий может проводить разные

виды часов. Определение количества часов, которые проводит преподаватель, невозможно без указания предмета, группы и вида занятий, поэтому имеет место ФЗ: *ФИО, Предмет, Группа, Вид занятий → Часы*. Действительно, Иванов И.И. по предмету «Управление данными» в 191 группе проводит лекции в количестве 30 часов, а практики по данному же предмету и в данной же группе проводит уже в количестве 50 часов.

Нами не были выделены зависимости между атрибутами *ФИО*, *Предмет*, *Группа* и *Вид занятий*, поскольку они образуют составной ключ и не учитываются в процессе нормализации исходного отношения.

После того как выделены все функциональные зависимости, следует проверить их согласованность с данными исходного отношения ПРЕПОДАВАТЕЛЬ (табл. 10). Например, Должность – Доцент и Оклад – 160 всегда соответствуют друг другу во всех кортежах, т.е. подтверждается функциональная зависимость *Должность* ↔ *Оклад*. Так же следует проверить и остальные функциональные зависимости, не забывая об ограниченности имеющихся в отношении данных.

В нашем отношение присутствует транзитивная зависимость: *ФИО → Оклад*, т.к. существуют следующие зависимости *ФИО → Должность → Оклад*. В дальнейшем эту ФЗ мы исключим. Возможен и второй вариант: в качестве транзитивной зависимости выбрать *ФИО → Должность*, т.к. существуют следующие зависимости *ФИО → Оклад → Должность*. Но первый вариант более подходит для организации, т.к. чаще всего выполняется поиск для преподавателя должности, а уже затем для должности оклада.

3 шаг: Проверка отношения на НФБК

В отношении в табл. 10 первичный ключ является составным и состоит из атрибутов <ФИО, Предмет, Группа, Вид занятий>. Других возможных ключей, которые могут играть роль первичного ключа, нет.

В качестве детерминантов выступаю атрибуты, находящиеся в левых частях ФЗ (Табл. 12).

Таблица 12

Возможные ключи и детерминанты отношения ПРЕПОДАВАТЕЛЬ

Возможные ключи	Детерминанты
1. ФИО, Предмет, Группа, Вид занятий	1. ФИО 2. Должность 3. Оклад 4. ФИО, Предмет, Группа, Вид занятий

Т.к. не выполняется условие, чтобы каждый детерминант отношения являлся возможным ключом, то отношение ПРЕПОДАВАТЕЛЬ **не находится в НФБК** и его необходимо подвергнуть процедуре декомпозиции. Поэтому и были выявлены аномалии при использовании данного единственного отношения в БД.

Выделим самую крайнюю функциональную зависимость (*Должность → Оклад*) в отношение R1 (Рис. 6).

R1 (Должность, Оклад)

Должность → Оклад

Оклад → Должность

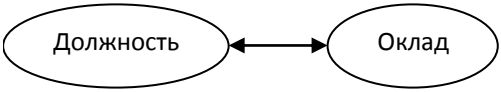


Рис. 6. Отношение R1

Найдем возможные ключи и детерминанты отношения R1 (Табл. 13).

Таблица 13

Возможные ключи и детерминанты отношения R1

Возможные ключи	Детерминанты
1. Должность 2. Оклад	1. Должность 2. Оклад

Обратите внимание, что в данном случае условия принадлежности отношения R1 к НФБК выполняются. Дадим отношению R1 осмысленное имя ДОЛЖНОСТЬ и выберем в качестве первичного ключа атрибут <Должность> (т.к. в дальнейшем мы будем выполнять поиск оклада по должности) (Табл. 14).

Таблица 14

Отношение ДОЛЖНОСТЬ (Должность, Оклад)

Должность	Оклад
Ассистент	100
Доцент	160
Ст. преп.	120

Таким образом, мы смогли устранить избыточное дублирование информации об окладах должностей.

Оставшиеся атрибуты отношения ПРЕПОДАВАТЕЛЬ войдут в отношение R2, проверим его на НФБК (Рис. 7).

R2 (ФИО, Должность, Кафедра, Предмет, Группа, Вид занятий, Часы)

ФИО → Должность

ФИО → Кафедра

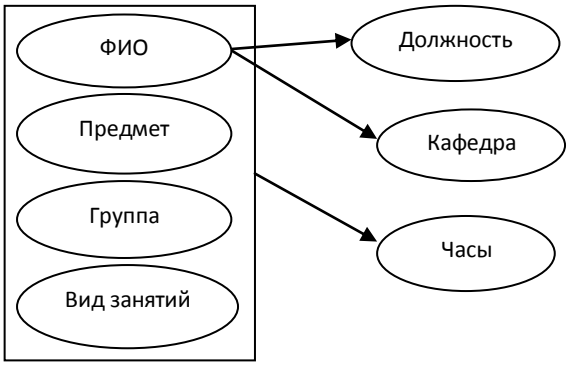


Рис. 7. Отношение R2

Найдем возможные ключи и детерминанты отношения R2 (Табл. 15).

Таблица 15

Возможные ключи и детерминанты отношения R2

Возможные ключи	**Детерминанты**
1. ФИО, Предмет, Группа, Вид занятий	1. ФИО 2. ФИО, Предмет, Группа, Вид занятий

Условия нахождения отношения R2 в НФБК не выполняются, следовательно, его необходимо подвергнуть процедуре декомпозиции. Выделим функциональные зависимости *ФИО → Должность* и *ФИО → Кафедра* в отношение R3 (Рис. 8).

R3 (ФИО, Должность, Кафедра)

ФИО → Должность

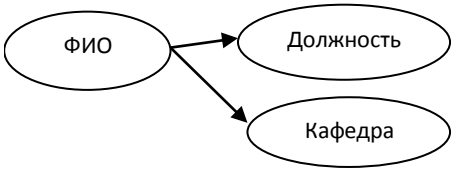


Рис. 8. Отношение R3

Найдем возможные ключи и детерминанты отношения R3 (Табл. 16).

Таблица 16

Возможные ключи и детерминанты отношения R3

Возможные ключи	Детерминанты
1. ФИО	1. ФИО

Условия нахождения отношения R3 в НФБК выполняются, дадим осмысленное имя отношению – ПРЕПОДАВАТЕЛЬ, первичным ключом будет являться атрибут <ФИО> (Табл. 17).

Таблица 17

Отношение ПРЕПОДАВАТЕЛЬ
(ФИО, Должность, Кафедра)

ФИО	Должность	Кафедра
Иванов И.И.	Доцент	Кафедра информатики
Петров П.П.	Ст. преп.	Кафедра информатики
Егоров Е.Е.	Доцент	Кафедра экономики
Сидоров С.С.	Ассистент	Кафедра экономики

Оставшиеся атрибуты отношения R2 войдут в отношение R4, проверим его на НФБК (Рис. 9).

Рис. 9. Отношение R4

Найдем возможные ключи и детерминанты отношения R4 (Табл. 18).

Таблица 18

Возможные ключи и детерминанты отношения R4

Возможные ключи	**Детерминанты**
1. ФИО, Предмет, Группа, Вид занятий	1. ФИО, Предмет, Группа, Вид занятий

Условия нахождения отношения R4 в НФБК выполняются, дадим осмысленное имя отношению – ПРОВОДИТ, первичным ключом будет составной включающий в себя атрибуты *ФИО*, *Предмет*, *Группа* и *Вид занятий* (Табл. 19).

Таблица 19

Отношение ПРОВОДИТ (ФИО, Предмет, Группа, Вид занятий, Часы)

ФИО	Предмет	Группа	Вид занятий	Часы
Иванов И.И.	Управление данными	191	Лекции	30
Иванов И.И.	Управление данными	191	Практики	50
Иванов И.И.	Операционные системы	192	Лекции	24
Петров П.П.	Управление данными	191	Практики	50
Петров П.П.	Операционные системы	192	Практики	36
Егоров Е.Е.	Экономика организации	191	Лекции	28
Егоров Е.Е.	Экономика организации	191	Практики	44
Сидоров С.С.	Экономика организации	191	Практики	44

Таким образом, мы получили отношения в НБФК:

ДОЛЖНОСТЬ (Должность, Оклад);

ПРЕПОДАВАТЕЛЬ (ФИО, Должность, Кафедра);

ПРОВОДИТ (ФИО, Предмет, Группа, Вид занятий, Часы).

Проверим получившиеся отношения на возможные аномалии.

При добавлении в базу данных нового преподавателя, ещё не преподающего ни одного предмета, необходимо добавить кортеж только в отношение ПРЕПОДАВАТЕЛЬ; соответственно, запрос, выдающий список преподавателей проводящих занятий, сформирует перечень фамилий преподавателей без данных о новом преподавателе.

Так как уникальные данные храниться только в одном месте, то проблем, связанных с обновлением данных, не возникнет.

И проблема удаления также корректно обрабатывается: удаление информации о проведении занятий не затронет основной информации о преподавателе.

Попробуем решить нашу исходную задачу, максимально приблизив её к реальной практике создания и использования баз данных.

Во-первых, потребуется использовать для идентификации текстовых значений суррогатных ключей (будут добавлены поля – *Код*). Во-вторых, вместо поля *ФИО* будем использовать три отдельных поля: *Фамилия*, *Имя* и *Отчество*.

Соответственно, исходный перечень атрибутов и их краткие характеристики будут модифицированы (Табл. 20).

Таблица 20

Перечень атрибутов и их краткие характеристики

Наименование атрибута	**Краткое наименование**	**Характеристика**
Код преподавателя	КодПреп	Уникальный номер для каждого преподавателя
Фамилия	Фамилия	Фамилия преподавателя
Имя	Имя	Имя преподавателя
Отчество	Отчество	Отчество преподавателя
Код должности	КодДолж	Уникальный номер, соответствующий должности, занимаемой преподавателем
Наименование должности	НаимДолж	Должность, занимаемая преподавателем
Оклад	Оклад	Оклад преподавателя
Код кафедры	КодКаф	Уникальный номер кафедры, на которой числится преподаватель

Окончание табл. 20

Наименование атрибута	Краткое наименование	Характеристика
Наименование кафедры	НаимКаф	Наименование кафедры, на которой числится преподаватель
Код предмета	КодПред	Уникальный номер предмета (дисциплины), читаемого преподавателем
Наименование предмета	НаимПред	Наименование предмета (дисциплины), читаемого преподавателем
Код группы	КодГр	Уникальный номер учебной группы, в которой преподаватель проводит занятия
Наименование группы	НаимГр	Наименование учебной группы, в которой преподаватель проводит занятия
Код вида занятий	КодВЗ	Уникальный код вида занятий, проводимого преподавателем в учебной группе
Наименование вида занятий	НаимВЗ	Наименование вида занятий, проводимого преподавателем в учебной группе
Часы	Часы	Количество часов, проводимых преподавателем указанного вида занятий по данному предмету в данной учебной группе

В табл. 21 представлено отношение ПРЕПОДАВАТЕЛЬ, на рис. 10 – функциональные зависимости (а – математически, б – графически).

Отношение

Код Преп	Фамилия	Имя	Отчество	Код Долж	Наим Долж	Оклад	Код Каф	Наим Каф
1	Иванов	Иван	Иванович	1	Доцент	160	1	Кафедра информатики
1	Иванов	Иван	Иванович	1	Доцент	160	1	Кафедра информатики
1	Иванов	Иван	Иванович	1	Доцент	160	1	Кафедра информатики
2	Петров	Петр	Петрович	2	Ст. преп.	120	1	Кафедра информатики
2	Петров	Петр	Петрович	2	Ст. преп.	120	1	Кафедра информатики
3	Егоров	Егор	Егорович	1	Доцент	160	2	Кафедра экономики
3	Егоров	Егор	Егорович	1	Доцент	160	2	Кафедра экономики
4	Сидоров	Сергей	Сергеевич	3	Ассистент	100	2	Кафедра экономики

Таблица 21

ПРЕПОДАВАТЕЛЬ

Код Пред	Наим Пред	КодГр	НаимГр	КодВЗ	НаимВЗ	Часы
1	Управление данными	1	191	1	Лекции	30
1	Управление данными	1	191	2	Практики	50
2	Операционные системы	2	192	1	Лекции	24
1	Управление данными	1	191	2	Практики	50
2	Операционные системы	2	192	2	Практики	36
3	Экономика организации	1	191	1	Лекции	28
3	Экономика организации	1	191	2	Практики	44
3	Экономика организации	1	191	2	Практики	44

ПРЕПОДАВАТЕЛЬ (КодПреп, Фамилия, Имя, Отчество, КодДолж, НаимДолж, Оклад, КодКаф, НаимКаф, КодПред, НаимПред, КодГр, НаимГр, КодВЗ, НаимВЗ, Часы)

КодПреп → Фамилия
КодПреп → Имя
КодПреп → Отчество
КодПреп → КодДолж
КодПреп → КодКаф
КодДолж → НаимДолж
КодДолж → Оклад
КодКаф → НаимКаф
КодПред → НаимПред
КодГр→ НаимГр
КодВЗ → НаимВЗ

а)

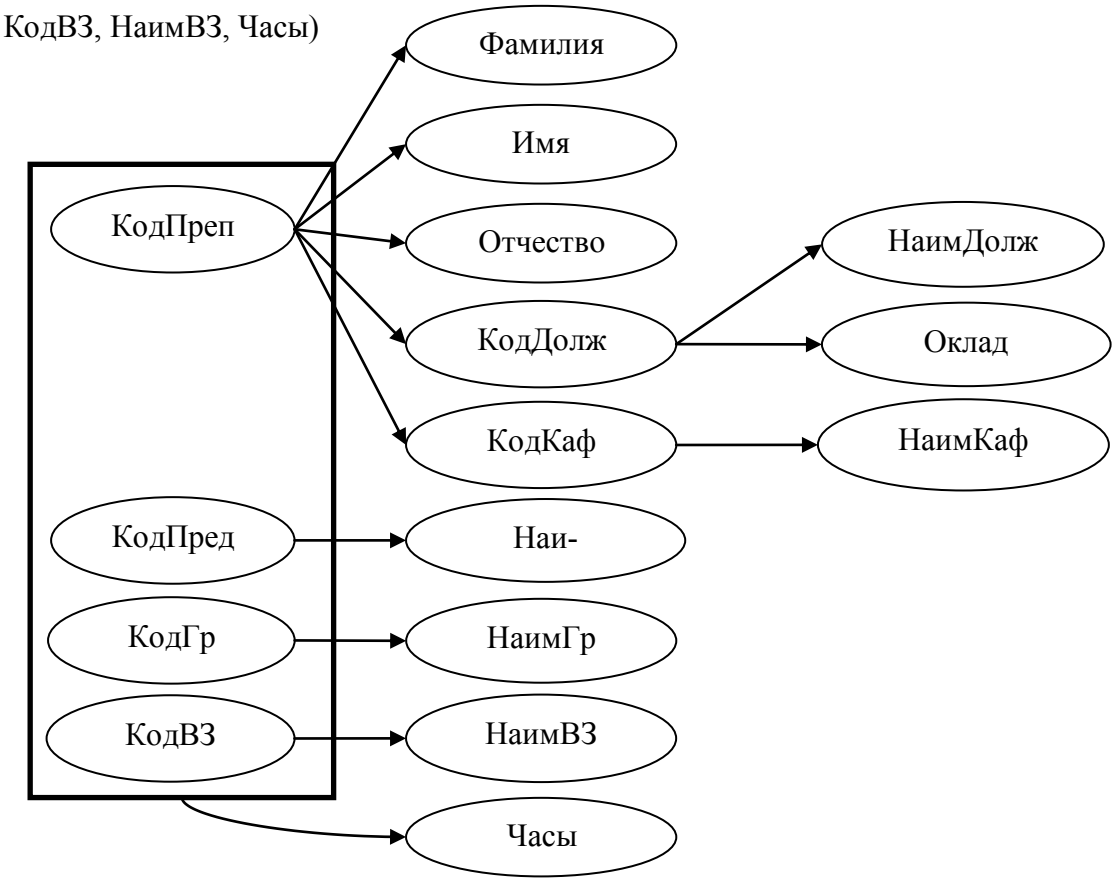


Рис. 10. Функциональные зависимости отношения ПРЕПОДАВАТЕЛЬ

Приведем отношение ПРЕПОДАВАТЕЛЬ (Табл. 21) к НФБК. Первичный ключ нашего отношения является составным и состоит из атрибутов <КодПреп, КодПред, КодГр, КодВЗ>. Других возможных ключей, которые могут играть роль первичного ключа, нет.

В качестве детерминантов выступаю атрибуты, находящиеся в левых частях ФЗ (Табл. 22).

Таблица 22

Возможные ключи и детерминанты отношения ПРЕПОДАВАТЕЛЬ

Возможные ключи	Детерминанты
1. КодПреп, КодПред, КодГр, КодВЗ	1. КодПреп 2. КодДолж 3. КодКаф 4. КодПред 5. КодГр 6. КодВЗ 7. КодПреп, КодПред, КодГр, КодВЗ

Т.к. не выполняется условие, чтобы каждый детерминант отношения являлся возможным ключом, то отношение ПРЕПОДАВАТЕЛЬ **не находится в НФБК** и его необходимо подвергнуть процедуре декомпозиции.

Выделим самую крайнюю функциональную зависимость (*КодДолж → НаимДолж и КодДолж → Оклад*) в отношение R1 (Рис. 11).

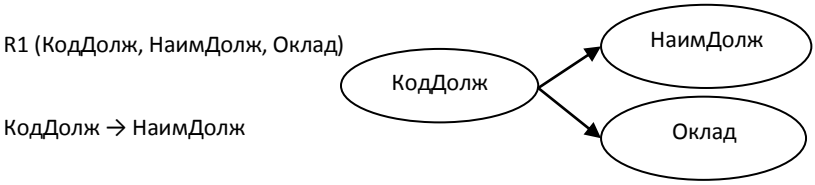


Рис. 11. Отношение R1

Найдем возможные ключи и детерминанты отношения R1 (Табл. 23).

Таблица 23

Возможные ключи и детерминанты отношения R1

Возможные ключи	Детерминанты
1. КодДолж	1. КодДолж

Отношение R1 находится в НФБК. Дадим отношению R1 осмысленное имя – ДОЛЖНОСТЬ, первичным ключом будет являться атрибут – <КодДолж> (Табл. 24).

Таблица 24

Отношение ДОЛЖНОСТЬ (КодДолж, НаимДолж, Оклад)

КодДолж	НаимДолж	Оклад
1	Доцент	160
2	Ст. преп.	120
3	Ассистент	100

Оставшиеся атрибуты отношения ПРЕПОДАВАТЕЛЬ войдут в отношение R2, проверим его на НФБК (Рис. 12).

R2 (КодПреп, Фамилия, Имя, Отчество, КодДолж, КодКаф, НаимКаф, КодПред, НаимПред, КодГр, НаимГр, КодВЗ, НаимВЗ, Часы)

КодПреп → Фамилия

КодПреп → Имя

КодПреп → Отчество

КодПреп → КодДолж

КодПреп → КодКаф

КодКаф → НаимКаф

КодПред → НаимПред

КодГр→ НаимГр

КодВЗ → НаимВЗ

КодПреп, КодПред, КодГр, КодВЗ → Часы

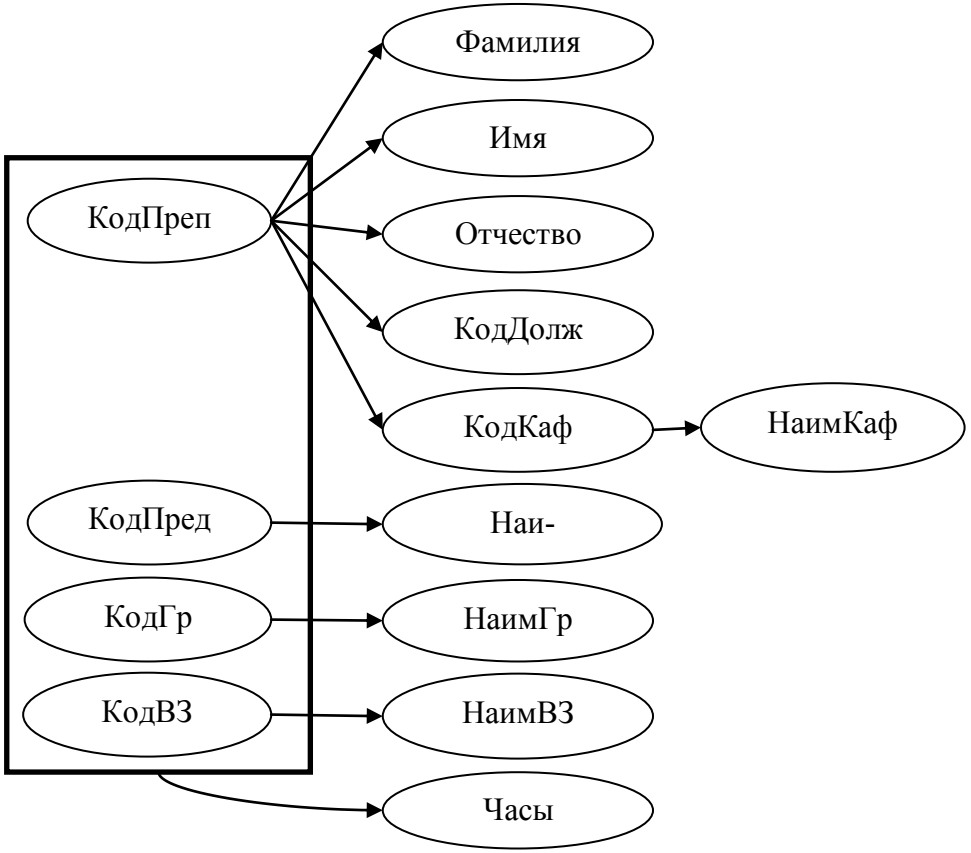


Рис. 12. Отношение R2

Найдем возможные ключи и детерминанты отношения R2 (Табл. 25).

Таблица 25

Возможные ключи и детерминанты отношения R2

Возможные ключи	Детерминанты
1. КодПреп, КодПред, КодГр, КодВЗ	1. КодПреп 2. КодКаф 3. КодПред 4. КодГр 5. КодВЗ 6. КодПреп, КодПред, КодГр, КодВЗ

Отношение R2 не находится в НФБК. Выделим самую крайнюю функциональную зависимость (*КодКаф* → *НаимКаф*) в отношение R3 (Рис. 13).

Рис. 13. Отношение R3

Найдем возможные ключи и детерминанты отношения R3 (Табл. 26).

Таблица 26

Возможные ключи и детерминанты отношения R3

Возможные ключи	Детерминанты
1. КодКаф	1. КодКаф

Отношение R3 находится в НФБК. Дадим отношению R3 осмысленное имя – КАФЕДРА, первичным ключом будет являться атрибут – <КодКаф> (Табл. 27).

Таблица 27

Отношение КАФЕДРА (КодКаф, НаимКаф)

КодКаф	НаимКаф
1	Кафедра информатики
2	Кафедра экономики

Оставшиеся атрибуты отношения R2 войдут в отношение R4, проверим его на НФБК (Рис. 14).

R4 (КодПреп, Фамилия, Имя, Отчество, КодДолж, КодКаф, КодПред, НаимПред, КодГр, НаимГр, КодВЗ, НаимВЗ, Часы)

КодПреп → Фамилия
КодПреп → Имя
КодПреп → Отчество
КодПреп → КодДолж
КодПреп → КодКаф
КодКаф → НаимКаф
КодПред → НаимПред
КодГр→ НаимГр
КодВЗ → НаимВЗ
КодПреп, КодПред, КодГр,
КодВЗ → Часы

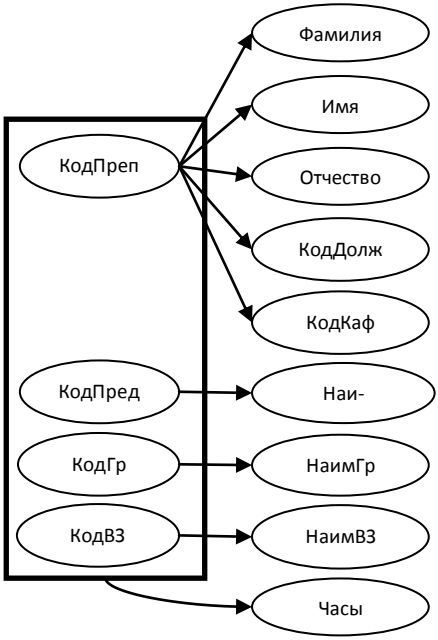


Рис. 14. Отношение R4

Найдем возможные ключи и детерминанты отношения R4 (Табл. 28).

Таблица 28

Возможные ключи и детерминанты отношения R4

Возможные ключи	Детерминанты
1. КодПреп, КодПред, КодГр, КодВЗ	1. КодПреп 2. КодПред 3. КодГр 4. КодВЗ 5. КодПреп, КодПред, КодГр, КодВЗ

Отношение R4 не находится в НФБК. Выделим самую крайнюю функциональную зависимость (*КодПреп → Фамилия, КодПреп → Имя, КодПреп → Отчество, КодПреп → КодДолж, КодПреп → КодКаф*) в отношение R5 (Рис. 15).

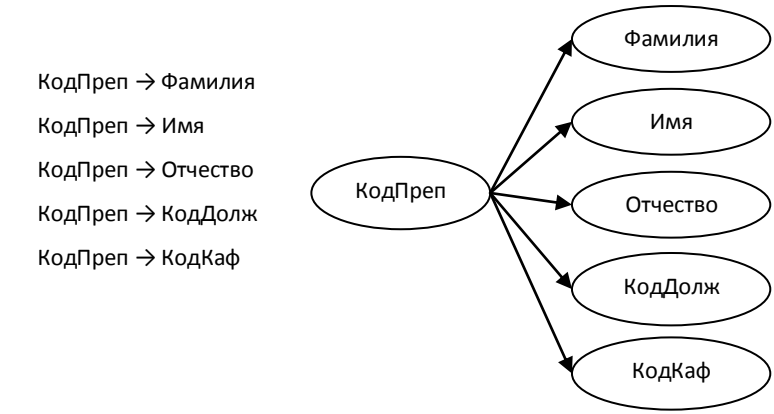


Рис. 15. Отношение R5

Найдем возможные ключи и детерминанты отношения R5 (Табл. 29).

Таблица 29

Возможные ключи и детерминанты отношения R5

Возможные ключи	Детерминанты
1. КодПреп	1. КодПреп

Отношение R5 находится в НФБК. Дадим отношению R5 осмысленное имя – ПРЕПОДАВАТЕЛЬ, первичным ключом будет являться атрибут – <КодПреп> (Табл. 30).

Таблица 30

Отношение ПРЕПОДАВАТЕЛЬ (КодПреп, Фамилия, Имя, Отчество, КодДолж, КодКаф)

КодПреп	Фамилия	Имя	Отчество	КодДолж	КодКаф
1	Иванов	Иван	Иванович	1	1
2	Петров	Петр	Петрович	2	1
3	Егоров	Егор	Егорович	1	2
4	Сидоров	Сергей	Сергеевич	3	2

Оставшиеся атрибуты отношения R4 войдут в отношение R6, проверим его на НФБК (Рис. 16).

R6 (КодПреп, КодПред, НаимПред, КодГр, НаимГр, КодВЗ, НаимВЗ, Часы)

КодПред → НаимПред

КодГр→ НаимГр

КодВЗ → НаимВЗ

КодПреп, КодПред, КодГр,

КодВЗ → Часы

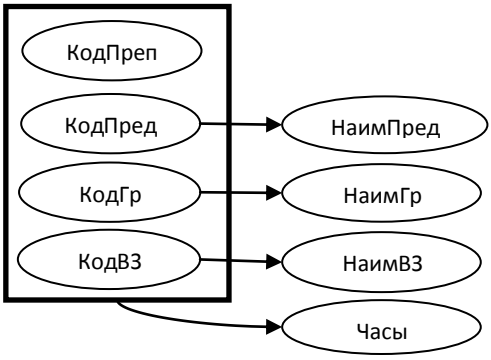


Рис. 16. Отношение R6

Найдем возможные ключи и детерминанты отношения R6 (Табл. 31).

Таблица 31

Возможные ключи и детерминанты отношения R6

Возможные ключи	Детерминанты
1. КодПреп, КодПред, КодГр, КодВЗ	1. КодПред 2. КодГр 3. КодВЗ 4. КодПреп, КодПред, КодГр, КодВЗ

Отношение R6 не находится в НФБК. Выделим самую крайнюю функциональную зависимость (*КодПреп* → *НаимПред*) в отношение R7 (Рис. 17).

Рис. 17. Отношение R7

Найдем возможные ключи и детерминанты отношения R7 (Табл. 32).

Таблица 32

Возможные ключи и детерминанты отношения R7

Возможные ключи	Детерминанты
1. КодПред	1. КодПред

Отношение R7 находится в НФБК. Дадим отношению R7 осмысленное имя – ПРЕДМЕТ, первичным ключом будет являться атрибут – <КодПред> (Табл. 33).

Таблица 33

Отношение ПРЕДМЕТ (КодПред, НаимПред)

КодПред	НаимПред
1	Управление данными
2	Операционные системы
3	Экономика организации

Оставшиеся атрибуты отношения R6 войдут в отношение R8, проверим его на НФБК (Рис. 18).

R8 (КодПреп, КодПред, КодГр, НаимГр, КодВЗ, НаимВЗ, Часы)

КодГр→ НаимГр

КодВЗ → НаимВЗ

КодПреп, КодПред, КодГр,

КодВЗ → Часы

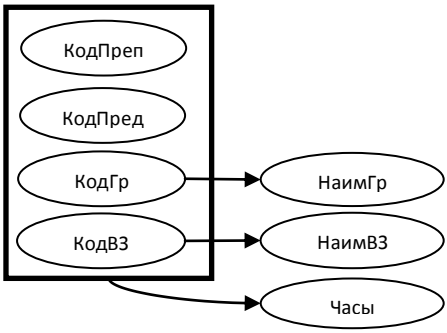


Рис. 18. Отношение R8

Найдем возможные ключи и детерминанты отношения R8 (Табл. 34).

Таблица 34

Возможные ключи и детерминанты отношения R8

Возможные ключи	Детерминанты
1. КодПреп, КодПред, КодГр, КодВЗ	1. КодГр 2. КодВЗ 3. КодПреп, КодПред, КодГр, КодВЗ

Отношение R8 не находится в НФБК. Выделим самую крайнюю функциональную зависимость (*КодГр → НаимГр*) в отношение R9 (Рис. 19).

Рис. 19. Отношение R9

Найдем возможные ключи и детерминанты отношения R9 (Табл. 35).

Таблица 35

Возможные ключи и детерминанты отношения R9

Возможные ключи	Детерминанты
1. КодГр	1. КодГр

Отношение R9 находится в НФБК. Дадим отношению R9 осмысленное имя – ГРУППА, первичным ключом будет являться атрибут – <КодГр> (Табл. 36).

Таблица 36

Отношение ГРУППА (КодГр, НаимГр)

КодГр	НаимГр
1	191
2	192

Оставшиеся атрибуты отношения R8 войдут в отношение R10, проверим его на НФБК (Рис. 20).

R10 (КодПреп, КодПред, КодГр, КодВЗ, НаимВЗ, Часы)

КодВЗ → НаимВЗ

КодПреп, КодПред, КодГр, КодВЗ → Часы

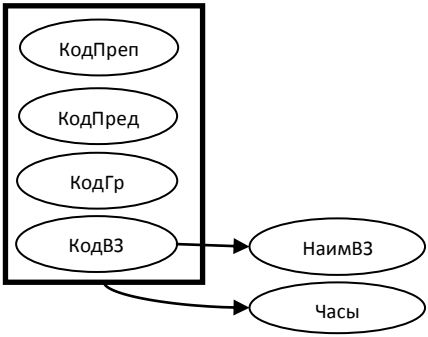


Рис. 20. Отношение R10

Найдем возможные ключи и детерминанты отношения R10 (Табл. 37).

Таблица 37

Возможные ключи и детерминанты отношения R10

Возможные ключи	Детерминанты
1. КодПреп, КодПред, КодГр, КодВЗ	1. КодВЗ 2. КодПреп, КодПред, КодГр, КодВЗ

Отношение R10 не находится в НФБК. Выделим самую крайнюю функциональную зависимость (*КодВЗ → НаимВЗ*) в отношение R11 (Рис. 21).

R11 (КодВЗ, НаимВЗ)

КодВЗ → НаимВЗ

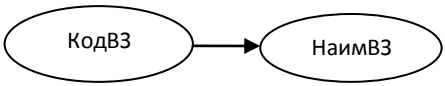


Рис. 21. Отношение R11

Найдем возможные ключи и детерминанты отношения R11 (Табл. 38).

Таблица 38

Возможные ключи и детерминанты отношения R11

Возможные ключи	Детерминанты
1. КодВЗ	1. КодВЗ

Отношение R11 находится в НФБК. Дадим отношению R11 осмысленное имя – ВИД ЗАНЯТИЙ, первичным ключом будет являться атрибут – <КодВЗ> (Табл. 39).

Таблица 39

Отношение ВИД ЗАНЯТИЙ (КодВЗ, НаимВЗ)

КодВЗ	НаимВЗ
1	Лекции
2	Практики

Оставшиеся атрибуты отношения R10 войдут в отношение R12, проверим его на НФБК (Рис. 22).

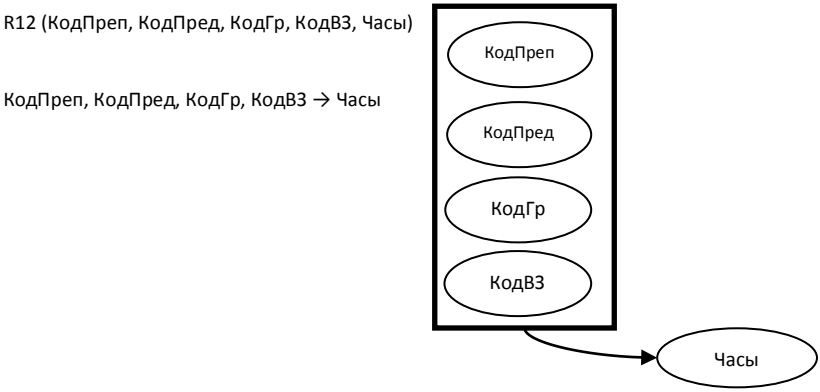


Рис. 22. Отношение R12

Найдем возможные ключи и детерминанты отношения R12 (Табл. 40).

Таблица 40

Возможные ключи и детерминанты отношения R12

Возможные ключи	Детерминанты
1. КодПреп, КодПред, КодГр, КодВЗ	1. КодПреп, КодПред, КодГр, КодВЗ

Отношение R12 находится в НФБК. Дадим отношению R12 осмысленное имя – ПРОВОДИТ, первичный ключ будет составным – <КодПреп, КодПред, КодГр, КодВЗ> (Табл. 41).

Таблица 41

Отношение ПРОВОДИТ (<u>КодПреп</u>, <u>КодПред</u>, <u>КодГр</u>, <u>КодВЗ</u>, Часы)

КодПреп	КодПред	КодГр	КодВЗ	Часы
1	1	1	1	30
1	1	1	2	50
1	2	2	1	24
2	1	1	2	50
2	2	2	2	36
3	3	1	1	28
3	3	1	2	44
4	3	1	2	44

Таким образом, мы получили отношения в НБФК:

ДОЛЖНОСТЬ (КодДолж, НаимДолж, Оклад);

КАФЕДРА (КодКаф, НаимКаф);

ПРЕПОДАВАТЕЛЬ (КодПреп, Фамилия, Имя, Отчество, КодДолж, КодКаф);

ПРЕДМЕТ (КодПред, НаимПред);

ГРУППА (КодГр, НаимГр);

ВИД ЗАНЯТИЙ (КодВЗ, НаимВЗ);

ПРОВОДИТ (КодПреп, КодПред, КодГр КодВЗ, Часы).

Вопросы и задания для самопроверки

1. Перечислите наиболее важные цели проектирования.
2. Дайте определение понятия нормализации данных.
3. Приведите пример функциональной зависимости.
4. Дайте определения первой, второй и третьей нормальной формы.
5. При выполнении каких условий, отношение будет находиться в нормальной форме Бойса-Кодда.
6. Перечислите шаги алгоритма декомпозиции.
7. Что такое универсальное отношение?
8. В чем заключаются проблемы добавления, обновления и удаления данных?

Рекомендуемая литература

1. Дейт, К.Дж. Введение в системы баз данных [Текст] / К.Дж. Дейт. – М.: Издательский дом «Вильямс», 2006. – 1328 с.
2. Джексон, Г. Проектирование реляционных баз данных для использования с микроЭВМ [Текст] / Г. Джексон. – М.: Мир, 1991. – 252 с.
3. Кириллов, В.В. Основы проектирования реляционных баз данных [Электронный ресурс] / В.В. Кириллов. – Режим доступа: http://citforum.ru/database/dbguide/index.shtml (01.12.2014).
4. Королева, О.Н. Базы данных [Электронный ресурс]: курс лекций / О.Н. Королева, А.В. Мажукин, Т.В. Королева. – Электрон. текстовые данные. – М.: Московский гуманитарный университет, 2012. – 66 с. – Режим доступа: http://www.iprbookshop.ru/14515. – ЭБС «IPRbooks», по паролю.
5. Кузнецов, С.Д. Основы современных баз данных [Электронный ресурс] / С.Д. Кузнецов. – Режим доступа: http://citforum.ru/database/osbd/contents.shtml (01.12.2014).
6. Кузнецов, С.Д. Базы данных. Вводный курс [Электронный ресурс] / С.Д. Кузнецов. – Режим доступа: http://citforum.ru/database/advanced_intro/ (01.12.2014).
7. Кузовкин, А.В. Управление данными [Текст]: учебник для студ. высших учеб. заведений / А.В. Кузовкин, А.А. Цыганов, Б.А. Щукин. – М.: Издательский центр «Академия», 2010. – 256 с.
8. Основы современных баз данных [Электронный ресурс]: метод. разработка к выполнению лабораторных работ. – Электрон. текстовые данные. – № 1–3. – Липецк: Липецкий

государственный технический университет, ЭБС АСВ, 2013. – 37 с. – Режим доступа: http://www.iprbookshop.ru/22906. – ЭБС «IPRbooks», по паролю.
9. Пушников, А.Ю. Введение в системы управления базами данных [Электронный ресурс] / А.Ю. Пушников. – Режим доступа: http://citforum.ru/database/dblearn/index.shtml (01.12.2014).
10. Туманов, В.Е. Основы проектирования реляционных баз данных [Электронный ресурс]: учебное пособие / В.Е. Туманов. – Электрон. текстовые данные. – М.: БИНОМ. Лаборатория знаний, Интернет-Университет Информационных Технологий (ИНТУИТ), 2007. – 420 с. – Режим доступа: http://www.iprbookshop.ru/22431. – ЭБС «IPRbooks», по паролю.
11. Ульман, Дж. Введение в системы баз данных [Текст] / Дж. Ульман, Дж. Уидом. – М.: Лори, 2006. – 379 с.
12. Хомоненко, А.Д. Базы данных [Текст]: учебник для высш. учеб. заведений / А.Д. Хомоненко. – СПб.: КОРОНА – Век, 2009. – 736 с.
13. Швецов, В.И. Базы данных [Электронный ресурс] / В.И. Швецов. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2009. – 155 с. – Режим доступа: http://www.iprbookshop.ru/16688. – ЭБС «IPRbooks», по паролю.

4. Введение в язык баз данных SQL

SQL (англ. Structured Query Language – структурированный язык запросов) – стандартный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных.

Структурированный язык запросов SQL основан на реляционном исчислении с переменными кортежами. Имеет несколько стандартов (SQL:1986, SQL:1989, SQL:1992, SQL:1999, SQL:2003, SQL:2006, SQL:2008).

Язык SQL предназначен для выполнения операций с таблицами (создание, удаление и изменение структуры), с данными таблиц (выборка, изменение, добавление и удаление) и для выполнения некоторых сопутствующих операций. Язык SQL автономно не используется, он погружен в среду встроенного языка программирования СУБД.

4.1. Типы команд SQL

Команды языка SQL обычно подразделяются на несколько групп. Основные группы команд следующие:

- DDL (Data Definition Language) – язык определения данных. Команды данной группы используются для создания и изменения структуры объектов базы данных (например, для создания и удаления таблиц).
- DML (Data Manipulation Language) – язык манипулирования данными. Команды DML используются для манипулирования информацией, содержащейся в объектах базы данных.
- DCL (Data Control Language) – язык управления данными.

Соответствующие команды предназначены для управления доступом к информации, хранящейся в базе данных.

- DQL (Data Query Language) – язык запросов. Это наиболее часто используемые команды, предназначенные для формирования запросов к базе данных (запрос – это обращение к базе данных для получения соответствующей информации).
- Команды администрирования базы данных предназначены для осуществления контроля за выполняемыми действиями и анализа производимых операций.
- Команды управления транзакциями.

4.2. Типы данных SQL

Типы данных, используемые в стандартном SQL, можно подразделить на следующие группы:

- строковые типы;
- числовые типы;
- типы для представления даты и времени.

Строковые типы

В SQL определены два строковых типа:

- символьные строки фиксированной длины;
- символьные строки переменной длины.

Символьные строки фиксированной длины

Данные, хранящиеся в виде символьных строк фиксированной длины, всегда занимают один и тот же, определяемый при объявлении поля, объем памяти, независимо от реального размера строки, занесенной в поле. Объявление строки фиксированной длины имеет вид: **CHARACTER(n)** (где n – длина строки, определяющая размер поля).

При использовании строк фиксированной длины пустые места обычно заполняются пробелами. Например, если размер

поля задан равным 10, а в него введена строка, состоящая из 3 символов, то оставшиеся 7 символов заполняются пробелами.

Символьные строки переменной длины

Длина строк переменной длины не является постоянной для всех данных, а зависит от реального размера строки, хранящейся в поле таблицы базы данных. Объявление строки переменной длины имеет вид: **VARCHAR(n)** (где n – число, определяющее максимально возможную длину строки).

В отличие от типа CHARACTER использование VARCHAR обеспечивает более экономное расходование дискового пространства. Независимо от того какой размер строки указан в объявлении, поле будет занимать столько места, сколько необходимо для хранения занесенной в него информации. Например, если объявлено поле VARCHAR(10) и в него занесена строка длиной 3 символа, то для хранения этой строки будет использовано только три байта, а не 10, как в случае строки фиксированной длины.

Числовые типы

Числовые типы подразделяются на:

- целочисленные типы;
- вещественные типы с фиксированной точкой;
- вещественные типы с плавающей точкой;
- двоичные строки фиксированной и переменной длины.

Целочисленные типы

В стандартном SQL устанавливаются два целочисленных типа:

- INTEGER – целое число со знаком, использующее 4 байта. Может представлять числа в диапазоне от -2 147 483 648 до 2 147 483 647;
- SMALLINT – короткое целое число со знаком, использующее 2 байта. Может представлять целые числа в диапазоне от -32 768 до 32 767.

Вещественные типы с фиксированной точкой

Вещественные типы с фиксированной точкой предназначены для точного представления дробных чисел. Наиболее часто эти типы используются в том случае, когда недопустимы погрешности, неизбежные при представлении вещественных чисел с плавающей запятой в двоичной форме (например, при хранении значений денежных величин). Вещественные типы с фиксированной запятой, по сути, являются целочисленными типами, в которых отображается десятичная точка.

Синтаксис объявления типа с фиксированной запятой следующий: **DECIMAL(n,m)** (где n – точность; m – масштаб).

Точность – это общая длина числового значения.

Масштаб – количество знаков, расположенных справа от десятичной точки.

Вещественные типы с плавающей точкой

Типы с плавающей точкой обычно используются в научных и инженерных расчетах. При использовании этих типов следует учитывать, что в процессе занесения в базу данных некоторого числа при его преобразовании в двоичную форму с плавающей точкой всегда вносится некоторая погрешность. И хотя эта погрешность очень мала, в некоторых случаях она является недопустимой и может внести серьезную ошибку, например, при суммировании большого количества значений. Поэтому типы с плавающей точкой неприменимы для хранения значений денежных величин.

Наиболее часто используются два вещественных типа с плавающей точкой:

- **FLOAT** – числа с одинарной точностью;
- **DOUBLE** – числа с двойной точностью.

Двоичные строки

Двоичные строки используются сравнительно редко. Обычно поля такого типа применяются в качестве флагов или

двоичных масок. Двоичные строки бывают фиксированной и переменной длины.

Двоичные строки фиксированной длины объявляются следующим образом: **BIT(n)** (где n – длина строки в байтах).

Объявление строк переменной длины выглядит так: **BIT VARYING(n)** (где n – максимальная длина строки в байтах).

Типы для представления даты и времени

В стандарте SQL определены следующие типы данных для хранения информации о дате и времени:

- **DATE** – используется для хранения даты;
- **TIME** – используется для хранения времени;
- **TIMESTAMP** – хранит дату и время;
- **INTERVAL** – хранит промежуток времени между двумя датами или между двумя моментами времени.

4.3. Управление объектами базы данных

Объект базы данных – это любой объект, определенный в базе данных и используемый для хранения информации или для обращения к информации. Примерами объектов базы данных могут служить таблицы, представления и индексы.

Для управления объектами базы данных используется подмножество команд DDL языка SQL.

Создание, модификация и удаление таблиц

Таблица является основным объектом для хранения информации в реляционной базе данных. При создании таблицы обязательно указываются имена полей, содержащихся в таблице, и типы данных, соответствующие полям. Кроме того, при создании таблицы для полей могут оговариваться ограничительные условия и значения, задаваемые по умолчанию.

Ограничительные условия – это правила, ограничивающие значения величин в поле таблицы базы данных.

Значение по умолчанию – значение, которое автоматически вводится в поле таблицы базы данных при добавлении новой записи, если пользователь не указал значение этого поля.

Оператор CREATE TABLE

Для создания таблицы используется оператор CREATE TABLE. Синтаксис этого оператора имеет следующий вид:

CREATE TABLE имя_таблицы
(имя_поля_1 тип_данных [(размер)]
[, имя_поля_2 тип_данных [(размер)]]
…
[, имя_поля_N тип_данных [(размер)]]).

Здесь **имя_таблицы** – имя создаваемой таблицы; **имя_поля_1**, **имя_поля_2**, **имя_поля_N** – имена полей таблицы; **тип_данных** – тип данных поля; **размер** – размер текстового поля. Конструкции, указанные в квадратных скобках, являются необязательными.

Для примера рассмотрим оператор, создающий таблицу ПРЕПОДАВАТЕЛЬ рассмотренную в главе 3 (этот и последующие операторы были проверены в СУБД Microsoft Office Access 2013):

CREATE TABLE ПРЕПОДАВАТЕЛЬ
(КодПреп INTEGER,
Фамилия VARCHAR(30),
Имя VARCHAR(30),
Отчество VARCHAR(30),
КодДолж INTEGER).

Оператор ALTER TABLE

Созданная таблица может быть модифицирована с использованием оператора ALTER TABLE. С помощью этого оператора можно добавлять и удалять поля таблицы, изменять тип данных полей, добавлять и удалять ограничения.

В общем виде синтаксис оператора ALTER TABLE выглядит следующим образом:

ALTER TABLE имя_таблицы

[ALTER COLUMN] [имя_поля тип_данных [(размер)]]

[ADD] [COLUMN] [имя_поля тип_данных [(размер)]]

[DROP] [COLUMN] [имя_поля].

Действие, выполняемое оператором ALTER TABLE, определяется ключевым словом, указываемым после имени таблицы:

- ALTER COLUMN – изменяет определение поля;
- ADD – добавляет новое поле в таблицу;
- DROP – удаляет поле из таблицы.

Для добавления поля используется следующий синтаксис оператора ALTER TABLE: ALTER TABLE имя_таблицы ADD COLUMN имя_поля тип_данных [(размер)].

Например, для того чтобы добавить в таблицу ПРЕПОДАВАТЕЛЬ поле, в котором будет содержаться код кафедры, следует использовать следующий оператор:

ALTER TABLE ПРЕПОДАВАТЕЛЬ ADD COLUMN КодКаф INTEGER.

Если же требуется изменить тип данных существующего поля, то следует использовать оператор ALTER TABLE в паре с ключевым словом ALTER COLUMN: ALTER TABLE имя таблицы ALTER COLUMN (имя поля тип данных).

Пусть, например, после того как мы добавили в таблицу ПРЕПОДАВАТЕЛЬ поле *КодКаф*, выяснилось, что использование типа INTEGER для этого поля неэффективно. Целесообразнее применить для этого поля тип данных SMALLINT. Для изменения типа данных вызовем оператор ALTER TABLE: *ALTER TABLE ПРЕПОДАВАТЕЛЬ ALTER COLUMN КодКаф SMALLINT.*

Удаление существующего поля выполняется вызовом оператора ALTER TABLE с ключевым словом DROP: ALTER TABLE имя_таблицы DROP имя_поля.

Оператор DROP TABLE

Для удаления таблиц используется оператор DROP TABLE. Синтаксис этого оператора имеет следующий вид: DROP TABLE имя_таблицы [RESTRICT | CASCADE].

Если при вызове оператора DROP TABLE используется ключевое слово RESTRICT и на удаляемую таблицу ссылается какое-либо представление или ограничение, то при выполнении оператора удаления таблицы будет сгенерировано сообщение об ошибке. Если же использовать ключевое слово CASCADE, то удаление таблицы будет выполнено и вместе с таблицей будут удалены все ссылающиеся на нее представления и ограничения.

Задание ограничений

Ограничения используются для того, чтобы обеспечить достоверность и непротиворечивость информации в базе данных. Существует достаточно большое количество различного рода ограничений, из которых мы рассмотрим лишь основные:

- ограничение NOT NULL;
- ограничение первичного ключа;
- ограничение UNIQUE;
- ограничение внешнего ключа;
- ограничение CHECK.

Ограничение NOT NULL

Ограничение NOT NULL может быть установлено для любого поля реляционной таблицы. При наличии этого ограничения запрещается ввод значений NULL в поле, для которого это ограничение установлено (NULL означает, что поле содержит неопределенное значение (поле пустое), то есть в него не была занесена никакая информация).

Ограничение NOT NULL устанавливается при создании таблицы с помощью оператора CREATE TABLE. Чтобы задать

ограничение NOT NULL для некоторого поля, следует указать NOT NULL после указания типа поля:

CREATE TABLE имя_таблицы
(имя поля_1 тип_данных NOT NULL,
имя поля_2 тип_данных NULL,
…
имя поля_N тип_данных NOT NULL).

Если же после задания типа данных поля следует слово NULL, то данное поле может содержать пустые значения. Однако атрибут NULL обычно устанавливается по умолчанию, поэтому указывать его явно нет необходимости.

Ограничение NOT NULL устанавливается для тех полей, в которые при занесении данных в таблицу обязательно должна быть введена какая-либо информация. Например, в таблице, содержащей данные о преподавателях, можно задать ограничение NOT NULL для полей, в которых будут содержаться фамилия, имя и отчество преподавателя. Поэтому оператор создания таблицы ПРЕПОДАВАТЕЛЬ следует видоизменить следующим образом:

CREATE TABLE ПРЕПОДАВАТЕЛЬ
(КодПреп INTEGER,
Фамилия VARCHAR(30) NOT NULL,
Имя VARCHAR(30) NOT NULL,
Отчество VARCHAR(30) NOT NULL,
КодДолж INTEGER).

Ограничение первичного ключа

Первичные ключи указываются при создании таблицы. Так как поля, входящие в состав первичного ключа, не могут принимать значение NULL, то для них обязательным является ограничение NOT NULL, Ограничение первичного ключа может быть задано двумя путями.

• В том случае, когда первичный ключ состоит только из одного поля, то он может быть задан с помощью ключевых слов PRIMARY KEY, указываемых при описании поля в операторе CREATE TABLE:

```
CREATE TABLE имя таблицы
(имя_поля_1 тип_данных NOT NULL PRIMARY KEY,
имя_поля_2 тип_данных,
…
имя_поля N тип_данных).
```

Обратите внимание на то, что указание ограничения NOT NULL для поля, являющегося первичным ключом, является обязательным.

• Первичный ключ может быть также задан в конце описания таблицы, после определений всех полей. Для этого также используется ключевая фраза PRIMARY KEY, после которой в круглых скобках указывается имя поля, составляющего первичный ключ:

```
CREATE TABLE имя_таблицы
(имя_поля_1 тип_данных NOT NULL,
имя_поля_2 тип данных,
…
имя_поля_N тип_данных,
PRIMARY KEY (имя_поля_1)).
```

Второй способ особенно удобен для задания составных первичных ключей. В этом случае в скобках следует указать через запятую все поля, составляющие первичный ключ:

```
CREATE TABLE имя_таблицы
(имя_поля_1 тип_данных NOT NULL,
имя_поля_2 тип_данных,
имя_поля_3 тип_данных NOT NULL,
…
имя_поля_N тип_данных,
PRIMARY KEY (имя_поля_1, имя_поля_3)).
```

Ограничение UNIQUE

Ограничение UNIQUE похоже на ограничение первичного ключа, так как при наличии этого ограничения для некоторого поля все значения, содержащиеся в этом поле, должны быть уникальными. Однако, в отличие от первичного ключа, ограничение UNIQUE допускает наличие пустых значений поля (если, конечно, для этого поля не установлено ограничение NOT NULL).

Ограничение UNIQUE задается при создании таблицы с помощью ключевого слова UNIQUE, указываемого при описании поля:

```
CREATE TABLE имя_таблицы
(имя_поля_1 тип_данных NOT NULL PRIMARY KEY,
имя_поля_2 тип_данных UNIQUE,
имя_поля_3 тип_данных,
…
имя_поля_N тип_данных NOT NULL UNIQUE).
```

Можно также задать ограничение UNIQUE не для одного поля, а для группы полей.

Объявление группы полей уникальной отличается от объявления уникальными индивидуальных полей, так как именно комбинация значений, а не просто индивидуальные значения, обязана быть уникальной. То есть значение каждого поля, входящего в группу, не обязательно должно быть уникальным, а комбинация знаний полей всегда должна быть уникальной.

Ограничение UNIQUE для группы полей так же, как и составной первичный ключ, задается после описания всех полей таблицы:

```
CREATE TABLE имя_таблицы
(имя_поля_1 тип_данных NOT NULL PRIMARY KEY,
имя_поля_2 тип_данных,
имя_поля_3 тип_данных NOT NULL,
…
```

имя_поля_N тип_данных NOT NULL UNIQUE,
UNIQUE (имя_поля_2, имя_поля_3)).

Ограничение внешнего ключа

Ограничение внешнего ключа является основным механизмом для поддержания ссылочной целостности базы данных. Поле, определяемое в качестве внешнего ключа, используется для ссылки на поле другой таблицы обычно называют родительским ключом, а таблица, на которую внешний ключ ссылается, называется – родительской таблицей (родительский ключ часто является первичным ключом родительской таблицы).

Типы полей внешнего и родительского ключа обязательно должны быть идентичны. А вот имена полей могут быть разными. Однако во избежание путаницы желательно и имена полей для внешнего и родительского ключей задавать одинаковыми.

Внешний ключ не обязательно должен состоять только из одного поля. Подобно первичному ключу внешний ключ может состоять из любого числа полей, которые обрабатываются как единый объект. Поля родительского ключа, на который ссылается составной внешний ключ, должны следовать в том же порядке, что и во внешнем ключе.

Когда поле таблицы является внешним ключом, оно определенным образом связано с таблицей, на которую этот ключ ссылается. Это фактически означает, что значение внешнего ключа непосредственно привязано к значению в родительском ключе.

Ограничение внешнего ключа (FOREIGN KEY) может быть задано либо в операторе CREATE TABLE, либо с помощью оператора ALTER TABLE. Синтаксис ограничения FOREIGN KEY имеет следующий вид: FOREIGN KEY имя_внешнего_ключа (список полей внешнего ключа) REFERENCES имя_родительской_таблицы (список полей родительского ключа).

Первый список полей – это список из одного или нескольких полей таблицы, разделенных запятыми. Второй список полей – это список полей, которые будут составлять родительский ключ. Списки полей, указываемые в качестве внешнего и родительского ключей, должны быть совместимы:

- они должны иметь одинаковое число полей;
- порядок следования полей в списках должен совпадать.

Причем совпадение определяется не именами полей, которые могут быть различны, а типами данных и размером полей.

Рассмотрим пример создания базы данных со связанными таблицами ДОЛЖНОСТЬ и ПРЕПОДАВАТЕЛЬ (рис. 23):

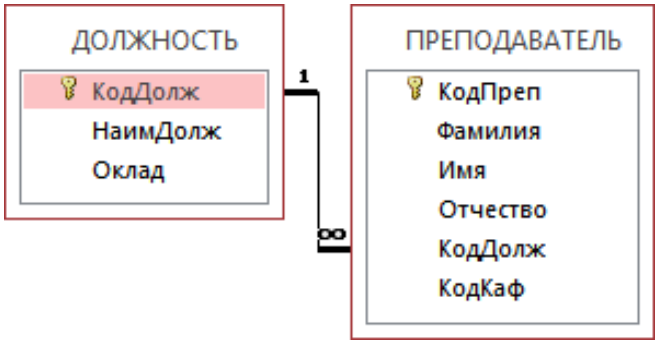


Рис. 23. Пример связанных таблиц

```
CREATE TABLE ДОЛЖНОСТЬ
 (КодДолж        INTEGER NOT NULL PRIMARY KEY,
  НаимДолж       VARCHAR(30) NOT NULL UNIQUE,
  Оклад          INTEGER NOT NULL)

CREATE TABLE ПРЕПОДАВАТЕЛЬ
 (КодПреп        INTEGER NOT NULL PRIMARY KEY,
  Фамилия        VARCHAR(30) NOT NULL,
  Имя            VARCHAR(30) NOT NULL,
```

Отчество VARCHAR(30) NOT NULL,
КодДолж INTEGER NOT NULL,
КодКаф SMALLINT NOT NULL,
UNIQUE (Фамилия, Имя, Отчество),
CONSTRAINT FK_PD FOREIGN KEY (КодДолж) REFERENCES ДОЛЖНОСТЬ (КодДолж)).

Внешний ключ может быть добавлен и после создания таблицы, с помощью оператора ALTER TABLE. Синтаксис оператора ALTER TABLE, используемый для создания внешнего ключа, имеет следующий вид: ALTER TABLE имя_таблицы ADD CONSTRAINT имя_внешнего_ключа FOREIGN KEY (список полей внешнего ключа) REFERENCE имя_родительской_таблицы (список полей родительского ключа).

В нашем случае это будет следующая команда: *ALTER TABLE ПРЕПОДАВАТЕЛЬ ADD CONSTRAINT FK_PD FOREIGN KEY (КодДолж) REFERENCES ДОЛЖНОСТЬ (КодДолж).*

Внешний ключ ограничивает значения, которые можно ввести в таблицу. Чтобы в поля, составляющие внешний ключ, можно было ввести некоторое значение, необходимо, чтобы это значение уже было введено в родительской таблице. Например, чтобы занести в таблицу ПРЕПОДАВАТЕЛЬ фамилию, имя и отчество нового преподавателя, необходимо, чтобы в таблице ДОЛЖНОСТЬ уже существовала запись о его должности, иначе невозможно будет заполнить обязательное поле *КодДолж.*

Для внешнего ключа может быть задано ограничение NOT NULL, но это необязательно, а в некоторых случаях даже нежелательно. Например, предположим, что принимается на работу новый преподаватель, но еще не определена однозначно должность, которую он займет. В этом случае можно занести все необходимые данные о нем в таблицу ПРЕПОДАВАТЕЛЬ,

ничего не указывая в поле *КодДолж*, которое будет заполнено позже.

Ограничение внешнего ключа также оказывает влияние на удаление и модификацию записей родительской таблицы. Никакое значение родительского ключа, на которое ссылается какой-либо внешний ключ, не может быть удалено или изменено. Это означает, например, что нельзя удалить из таблицы ДОЛЖНОСТЬ запись о должности, если она связана с записью в таблице ПРЕПОДАВАТЕЛЬ.

Аналогично, нельзя изменять значение родительского ключа, на который ссылается какой-либо внешний ключ, – это также приведет к потере информации и нарушению ссылочной целостности базы данных.

В некоторых реализациях SQL имеется возможность задавать для внешних ключей каскадное удаление и каскадное обновление. Это означает, что при попытке удалить или модифицировать значение родительского ключа, на которое ссылается внешний ключ, соответствующие записи внешнего ключа также будут удалены (каскадное удаление) или изменены (каскадное обновление).

Одним из синтаксических вариантов задания каскадного обновления и удаления является следующий:

UPDATE OF имя_родительской_таблицы CASCADES;

DELETE OF имя_родительской_таблицы CASCADES.

Ключевые фразы UPDATE OF и DELETE OF указываются в операторе CREATE TABLE. Вместо ключевого слова CASCADES можно указать слово RESTRICTED. В этом случае изменение и удаление значений родительского ключа, на которые ссылается внешний ключ из данной таблицы, будет запрещено.

Ограничение CHECK

Ограничение CHECK используется для проверки допустимости данных, вводимых в поле таблицы.

Ограничение CHECK состоит из ключевого слова CHECK, сопровождаемого предложением предиката, который использует указанное поле. Любая попытка модифицировать или вставить значение поля, которое могло бы сделать этот предикат неверным, будет отклонена.

Проверка корректности значений, заносимых в базу данных, может также выполняться в пользовательских приложениях. Однако использование ограничения CHECK обеспечивает дополнительный уровень защиты от ошибок.

Задание ограничения CHECK производится при создании таблицы. Для этого после описания полей таблицы указывается ключевая фраза: CONSTRAINT имя_ограничения CHECK (ограничение).

В рассматриваемом нами примере ограничение может быть задано, например, для поля *Оклад* таблицы ДОЛЖНОСТЬ, оклад не может быть меньше 0. Тогда оператор создания таблицы ДОЛЖНОСТЬ, в котором задано это ограничение, будет иметь следующий вид:

```
CREATE TABLE ДОЛЖНОСТЬ
 (КодДолж      INTEGER NOT NULL PRIMARY KEY,
  НаимДолж     VARCHAR(30) NOT NULL UNIQUE,
  Оклад        INTEGER NOT NULL,
  CHECK (Оклад>=0)).
```

Можно задавать ограничение и для нескольких полей. Для этого следует просто включить их в ограничительное условие. Для формирования сложного ограничения, включающего несколько условий, используются логические операторы AND и OR.

Задание значений по умолчанию

Для полей таблицы можно задавать значения по умолчанию, которые будут заноситься в поля при добавлении новой записи в таблицу, если значения этих полей не определены.

Значение NULL фактически является значением по умолчанию, принятым для каждого поля таблицы, для которого не задано ограничение NOT NULL и которое не имеет и другого значения по умолчанию.

Для задания значения по умолчанию используется директива DEFAULT, указываемая в команде CREATE TABLE при описании поля, для которого устанавливается значение по умолчанию:

```
CREATE TABLE
(…
имя_поля_N тип_данных DEFAULT = значение_по_умолчанию
…).
```

Создание и удаление индексов

Синтаксис оператора создания индекса может быть различным в зависимости от используемой реализации SQL. Наиболее часто встречается следующая синтаксическая форма команды создания индекса: CREATE INDEX имя_индекса ON имя_таблицы (имя_поля_1, [имя_поля_2, ...]).

Приведенная форма оператора CREATE INDEX может быть дополнена рядом многочисленных параметров, которые сильно различаются в разных реализациях SQL. Эти параметры используются, например, для упорядочивания информации по возрастанию или убыванию (параметры ASC и DESC).

Создание простого индекса

Простой индекс является простейшей и вместе с тем распространенной разновидностью индексов. Простой индекс состоит только из одного поля (столбца) таблицы, поэтому он часто также называется одностолбцовым индексом. Наиболее типичный синтаксис команды создания простого индекса имеет вид: CREATE INDEX имя_индекса ON имя_таблицы (имя_столбца).

Например, для таблицы ДОЛЖНОСТЬ можно было бы создать индекс по полю, содержащему названия должностей, с помощью следующего оператора: *CREATE INDEX IDX_DOLG ON ДОЛЖНОСТЬ (НаимДолж).*

Уникальные индексы

Уникальный индекс не допускает введения в таблицу дублирующихся значений. Таким образом, уникальные индексы используются не только с целью повышения производительности, но и для поддержания целостности данных.

Типичный синтаксис оператора создания уникального индекса имеет следующий вид: CREATE UNIQUE INDEX имя_индекса ON имя_таблицы (имя_поля).

Например, для таблицы ДОЛЖНОСТЬ можно создать уникальный индекс по полю название должности с помощью следующей команды:

CREATE UNIQUE INDEX IDX_DOLGU ON ДОЛЖНОСТЬ (НаимДолж).

Составные индексы

Составными называются индексы, построенные по двум и более полям. При создании составного индекса необходимо учитывать, что порядок следования полей в составном индексе оказывает существенное влияние на скорость поиска данных. В общем случае поля в индексе следует располагать в порядке уменьшения ограничивающих значений. Синтаксис задания составного индекса имеет следующий вид: CREATE INDEX имя_индекса ON имя_таблицы (имя_поля_1, имя_поля_2, ...).

В нашем примере имеет смысл создать составной индекс для полей *Фамилия*, *Имя*, *Отчество* таблицы ПРЕПОДАВАТЕЛЬ. Оператор создания такого индекса имеет следующий вид:

CREATE INDEX FIO_IDX ON ПРЕПОДАВАТЕЛЬ (Фамилия, Имя, Отчество).

Удаление индексов

Удаление индексов не вызывает никаких проблем. Для удаления необходимо знать только имя индекса (и обладать соответствующими правами). Синтаксис оператора удаления индекса имеет следующий вид: DROP INDEX имя_индекса.

Удаление индекса никак не влияет на информацию, содержащуюся в индексированных полях. После удаления индекс может быть создан вновь.

4.4. Манипулирование данными

Для манипулирования данными, хранящимися в базе данных, используется группа операторов SQL, выделяемая в качестве отдельного типа команд, называемых языком манипулирования данными (DML – Data Manipulation Language). С помощью операторов DML пользователь может загружать в таблицы новые данные, модифицировать и удалять существующие данные.

В языке SQL определены только три основных оператора DML:

- INSERT;
- UPDATE;
- DELETE.

Добавление в таблицу новой информации

Процесс ввода в таблицу базы данных новой информации обычно называется загрузкой данных. Для загрузки данных используется оператор INSERT.

Добавление к таблице новой записи

Для добавления к таблице новой записи используется следующая синтаксическая форма оператора INSERT: INSERT INTO имя_таблицы VALUES (значение_1, значение_2, …, значение_N).

При использовании данной формы оператора INSERT список VALUES должен содержать количество значений, равное количеству полей таблицы. Причем тип данных каждого из значений, указываемых в списке VALUES, должен совпадать с типом данных поля, соответствующего этому значению.

Значения, относящиеся к символьным типам и датам, должны быть заключены в апострофы. В списке значений может также использоваться значение NULL.

Для добавления новой записи в таблицу ДОЛЖНОСТЬ следует использовать следующий оператор INSERT:

INSERT INTO ДОЛЖНОСТЬ VALUES (4, 'Преподаватель', 500).

Ввод данных в отдельные поля таблицы

При добавлении данных в таблицу можно заполнять не все поля, а лишь некоторые из них. В этом случае используется следующая синтаксическая форма оператора INSERT: INSERT INTO имя_таблицы (имя_поля_1, имя_поля_2, …, имя_поля_N) VALUES (значение_1, значение_2, …, значение_N).

Список полей в операторе INSERT может иметь произвольный порядок, не зависящий от порядка, по которому задаются поля при создании таблицы. Однако список значений должен соответствовать порядку, в котором указаны поля, связанные с этими значениями.

При выполнении данного оператора во все остальные поля будет занесено значение NULL. Естественно, что поля, которые не указываются в круглых скобках после имени таблицы, не должны иметь ограничения NOT NULL, иначе попытка выполнения оператора INSERT окажется неудачной.

Занесение в таблицу данных, содержащихся в другой таблице

Иногда требуется перенести часть информации из одной таблицы в другую. Такого рода операцию можно выполнить с помощью комбинации оператора INSERT с оператором выборки данных SELECT.

Объединяя операторы INSERT и SELECT, можно добавить в таблицу данные, полученные в результате выполнения запроса из другой таблицы (таблиц). Синтаксис оператора INSERT в этом случае будет иметь следующий вид: INSERT INTO имя_таблицы (имя_поля_1, имя_поля_2, …, имя_поля_N) SELECT [* | имя_поля_1, имя_поля_2, …, имя_поля_N] FROM имя_таблицы [WHERE условие].

В данном операторе вместо предложения VALUES используется оператор SELECT. Кратко поясним синтаксис этого оператора. После слова SELECT указывается список полей, значения которых включаются в выборку (если после SELECT указать символ *****, то в выборку будут включены все поля). Предложение FROM используется для указания имени таблицы, из которой производится выборка данных. Предложение WHERE является необязательным и используется для наложения ограничений на данные, включаемые в выборку.

Количество полей, указываемых в круглых скобках после имени таблицы в операторе INSERT, должно быть равно количеству полей, включаемых в выборку. Соответствие полей определяется порядком их следования: первому полю в списке оператора INSERT соответствует первое поле в списке оператора SELECT и т.д.

4.5. Изменение данных, хранящихся в таблице

Для изменения данных, уже занесенных в таблицу, используется оператор UPDATE. Данный оператор не добавляет новых записей в таблицу, а заменяет существующие данные на новые. Оператор UPDATE может быть применен как к одному полю таблицы (наиболее часто используемый случай), так и к нескольким полям. Количество изменяемых записей зависит

от потребностей пользователя – с помощью UPDATE можно изменить как одну, так и несколько записей (вплоть до изменения значения всех записей, содержащихся в таблице).

Модификация данных в одном поле таблицы

Для изменения данных только в одном из полей таблицы используется наиболее простая форма оператора UPDATE, имеющая следующий вид: UPDATE имя_таблицы SET имя_поля = значение [WHERE условие].

После ключевого слова UPDATE указывается имя таблицы, в которой модифицируются данные, после ключевого слова SET выполняется присвоение полю с заданным именем нового значения. Условие, задаваемое с помощью необязательного предложения WHERE, определяет количество записей, которые будут модифицированы.

При использовании оператора UPDATE необходимо быть очень внимательным и правильно формулировать ограничительные условия. В противном случае выполнение оператора UPDATE может привести к потере информации, хранящейся в базе данных.

Изменение значений в нескольких полях таблицы

С помощью оператора UPDATE можно одновременно изменять значения в нескольких полях таблицы. Для этого следует указать после ключевого слова SET не одно, а несколько полей:

```
UPDATE имя_таблицы
SET иия_поля_1 = значение_1,
имя_поля_2 = значение_2,
…
имя_поля_N = значение_N
[WHERE условие].
```

Использование оператора в данной форме ничем не отличается от рассмотренного ранее. Здесь точно так же нужно быть очень осторожным при формировании условия.

Пример установки нового значения оклада для должности с идентификатором 4:

UPDATE ДОЛЖНОСТЬ SET Оклад=550 WHERE КодДолж=4.

Удаление данных из таблицы

Удаление данных из таблицы выполняется с помощью оператора DELETE. Данный оператор полностью удаляет всю запись, а не данные из отдельных полей. Синтаксис оператора DELETE имеет следующий вид: DELETE FROM имя_таблицы [WHERE условие].

Удаляемые записи определяются в соответствии с условием, заданным с помощью необязательного предложения WHERE. При отсутствии предложения WHERE в операторе DELETE данные будут удалены из всей таблицы.

Пример удаления должности с идентификатором 4:

DELETE FROM ДОЛЖНОСТЬ WHERE КодДолж = 4.

4.6. Создание SQL-запросов

Одним из наиболее эффективных и универсальных способов выборки данных из таблиц базы данных является использование запросов языка SQL. Команды SQL подразделяются на несколько категорий. Для выборки данных используются команды, относящиеся к так называемому языку запросов DQL (Data Query Language), SQL-запросы можно использовать как при работе с локальными базами данных, так и с SQL-серверами баз данных (Oracle, Informix, Sybase, InterBase, Microsoft SQL Server). Причем при формировании SQL-запросов не имеет особого значения, какая система управления базами данных используется, так как команды языке SQL стандартизованы. Однако следует учитывать, что производители СУБД обычно предлагают свои реализации SQL, которые могу включать расширения команд стандарта и даже отклонения от него. Тем не менее большинство

команд SQL имеют одинаковый или очень похожий синтаксис в различных реализациях. Поэтому, изучив одну из реализаций SQL, впоследствии можно легко перейти на другую.

Язык запросов, являющийся одной из категорий языка SQL состоит всего из одной команды SELECT. Эта команда вместе с множеством опций и предложений используется для формирования запросов к базе данных.

Запросы формируются для извлечения из таблиц базы данных информации, соответствующей некоторым требованиям, задаваемым пользователем.

Оператор SELECT не используется автономно, вместе с ним обязательно должны задаваться уточняющие предложения. Предложения, используемые совместно с командой SELECT, могут быть обязательными и дополнительными. Обязательным считается только одно предложение – FROM, без которого оператор SELECT не может использоваться.

Простейшая форма оператора SELECT

Оператор SELECT вместе с предложением FROM используется для получения информации из базы данных. Синтаксис простейшей формы оператора SELECT: SELECT {* | ALL | DISTINCT поле1, поле2, …, полеN} FROM таблица1 {, таблица2, …, таблицаN}.

Здесь за ключевым словом SELECT следует список полей, которые возвращаются в результате выполнения запроса:

- имена полей в списке разделяются запятой;
- для выборки всех полей таблицы (таблиц) используется символ подстановки «*»;
- опция ALL (задана по умолчанию) означает, что результат выборки будет содержать все записи, включая дублирующие друг друга;
- при использовании опции DISTINCT результат запроса не будет содержать дублирующихся строк.

Совместно с командой SELECT всегда используется предложение FROM, с помощью которого указывается имя таблицы (таблиц), откуда производится выборка. Если в предложении FROM указывается несколько таблиц, то их имена разделяются запятыми.

Пример. Выберем фамилии, имена и отчества преподавателей из таблицы ПРЕПОДАВАТЕЛЬ (Рис. 24):

SELECT Фамилия, Имя, Отчество FROM ПРЕПОДАВАТЕЛЬ.

Фамилия	Имя	Отчество
Иванов	Иван	Иванович
Петров	Петр	Петрович
Егоров	Егор	Егорович
Сидоров	Сергей	Сергеевич
*		

Рис. 24. Результат

Если мы хотим выбрать только одни имена (в таблице возможны повторы), то должны указать опцию DISTINCT (Рис. 25):

SELECT DISTINCT Имя FROM ПРЕПОДАВАТЕЛЬ.

Имя
Егор
Иван
Петр
Сергей

Рис. 25. Результат

Задание условий при выборке данных

Для ограничения отбираемой из базы данных информации оператор SELECT позволяет использовать условие, которое задается с помощью предложения WHERE. В случае реализации условной выборки оператор SELECT имеет следующий вид: SELECT {* | ALL | DISTINCT поле1, поле2, …, полеN} FROM таблциаl {, таблица2, …, таблицаN} WHERE условие.

Специальные операторы языка SQL, применяемые для задания условия, можно разделить на следующие группы:

- операторы сравнения;
- логические операторы;
- операторы объединения;
- операторы отрицания.

Результатом выполнения каждого из этих операторов является логическое значение (TRUE или FALSE). Если для некоторой записи оператор возвращает значение TRUE, то запись включается в результат выборки, если FALSE – не включается.

Операторы сравнения

Операторы сравнения используются в запросах SQL для наложения ограничений на информацию, возвращаемую в результате выполнения запроса. Это типичные операторы, существующие во всех алгоритмических языках:

оператор равенства (=) используется для отбора записей, в которых значение определенного поля точно соответствует заданному;

оператор неравенства (<>) возвращает значение TRUE, если значение поля не совпадает с заданным значением;

операторы «меньше» и «больше» (соответственно, < и >) позволяют отбирать записи, в которых значение определенного поля меньше или больше некоторой заданной величины;

операторы «меньше или равно» и «больше или равно» (соответственно, «<=» и «>=») представляют собой объединение

операторов «меньше» и «равно», «больше» и «равно». В отличие от операторов «<» и «>» операторы «<=» и «>=» возвращают значение TRUE, если значение поля совпадает с заданным значением.

В качестве примера рассмотрим запрос, выбирающий из таблицы ПРЕПОДАВАТЕЛЬ данные только тех преподавателей, код кафедры которых имеет значение 2 (Рис. 26):

SELECT Фамилия, Имя, Отчество FROM ПРЕПОДАВАТЕЛЬ WHERE КодКаф=2.

Фамилия	Имя	Отчество
Егоров	Егор	Егорович
Сидоров	Сергей	Сергеевич
*		

Рис. 26. Результат

Логические операторы

К логическим относятся операторы, в которых для задания ограничений на отбор данных используются специальные ключевые слова. В SQL определены следующие логические операторы: IS NULL, BETWEEN...AND, IN, LIKE, EXISTS, UNIQUE, ALL, ANY.

Оператор IS NULL

Оператор IS NULL предназначен для сравнения текущего значения поля со значением NULL. Он используется для отбора записей, в некоторое поле которых не занесено никакое значение.

Оператор BETWEEN...AND

Оператор BETWEEN…AND применяется для отбора записей, в которых значения поля находится внутри заданного диапазона. Границы диапазона включаются в условие отбора.

Выберем названия должностей, которым соответствует значение оклада в диапазоне 120 – 160 (Рис. 27):

*SELECT * FROM ДОЛЖНОСТЬ WHERE Оклад between 120 And 160.*

	КодДолж	НаимДолж	Оклад
	2	Ст. преп.	120,00 р.
	3	Доцент	160,00 р.
*	(№)		0,00 р.

Рис. 27. Результат

Оператор IN

Оператор IN используется для выборки записей, в которых значение некоторого поля соответствует хотя бы одному из значений заданного списка.

Выберем из таблицы ПРЕПОДАВАТЕЛЬ данные преподавателей, которых зовут Иван или Петр (Рис. 28):

SELECT Фамилия, Имя, Отчество FROM ПРЕПОДАВАТЕЛЬ WHERE Имя IN ('Иван','Петр').

	Фамилия	Имя	Отчество
	Иванов	Иван	Иванович
	Петров	Петр	Петрович
*			

Рис. 28. Результат

Оператор LIKE

Оператор LIKE применяется для сравнения значения поля со значением, заданным при помощи шаблонов. Для задания шаблонов используются два символа:

- знак звездочка «*» заменяет последовательность символов любой (в том числе и нулевой) длины (для некоторых систем это может быть символ процента – «%»);
- символ вопроса «?» заменяет любой единичный символ (для некоторых систем это может быть символ подчеркивания «_»).

Найдем в таблице ГРУППА записи о группах 1 курса (при условии, что в наименовании группы первый символ обозначает номер курса) (Рис. 29):

SELECT НаимГр FROM ГРУППА WHERE НаимГр Like '1'.*

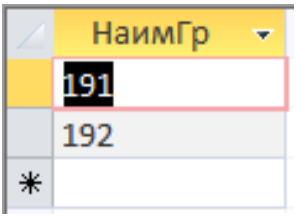


Рис. 29 Результат

Найдем в таблице ГРУППА записи о группах, заканчивающихся на 1 (Рис. 30):

*SELECT НаимГр FROM ГРУППА WHERE НаимГр Like '*1'.*

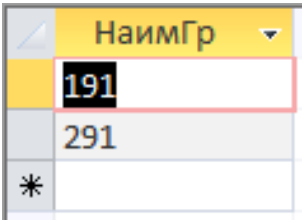


Рис. 30. Результат

Оператор EXISTS

Оператор EXISTS используется для отбора записей, соответствующих заданному критерию.

Для иллюстрации его работы рассмотрим следующий пример. Из таблицы ПРЕПОДАВАТЕЛЬ требуется отобрать фамилии, имена и отчества преподавателей, для которых код проводимых видов занятий равен 1 (Рис. 31):

SELECT Фамилия, Имя, Отчество FROM ПРЕПОДАВАТЕЛЬ WHERE EXISTS (SELECT КодПреп FROM ПРОВОДИТ WHERE КодВЗ=1 AND ПРЕПОДАВАТЕЛЬ.КодПреп=ПРОВОДИТ.КодПреп).

Фамилия	Имя	Отчество
Иванов	Иван	Иванович
Егоров	Егор	Егорович
*		

Рис. 31. Результат

В этом запросе после ключевого слова EXISTS следует оператор SELECT, отбирающий из таблицы ПРОВОДИТ записи, для которых код вида занятий равен 1. Оператор EXISTS отбирает из таблицы ПРЕПОДАВАТЕЛЬ записи, в которых значение поля *КодПреп* соответствует отобранным из таблицы ПРОВОДИТ.

Оператор UNIQUE

Оператор UNIQUE используется для проверки записи таблицы на уникальность. По своему действию он аналогичен оператору EXISTS. Единственное отличие заключается в том, что подзапрос, задаваемый после ключевого слова UNIQUE, не должен возвращать более одной записи.

Оператор ALL

Оператор ALL используется для сравнения исходного значения со всеми другими значениями, входящими в некоторый набор данных.

Оператор ANY

Оператор ANY применяется для сравнения заданного значения с каждым из значений некоторого набора данных.

Операторы объединения

Часто при написании запроса на выборку данных требуется задать сложное условие, для которого недостаточно использовать только один оператор. В этом случае используется объединение нескольких условий с помощью специальных операторов. В SQL определены два таких оператора:

- Оператор AND используется в тех случаях, когда необходимо отобрать записи, соответствующие нескольким условиям. Причем для каждой записи, включаемой в результат выборки, должны выполняться все заданные ограничения. Оператор AND объединяет несколько условий путем выполнения операции логического умножения результатов всех заданных ограничений. Результат TRUE, соответственно, будет получен только в том случае, если все объединяемые условия принимают значение TRUE.
- Оператор OR выполняет операцию логического сложения результатов всех заданных условий. При использовании данного оператора запись включается в результирующую выборку в случае выполнения хотя бы одного из заданных ограничений.

Исправим рассмотренный ранее пример на оператор Between…And:

*SELECT * FROM ДОЛЖНОСТЬ WHERE (Оклад >= 120) And (Оклад <= 160).*

Оператор отрицания

Для каждого из рассматриваемых операторов может быть выполнена операция отрицания, меняющая результат выполнения оператора на противоположный. Для реализации этой операции используется оператор NOT. Ниже приведены примеры использования этого оператора с логическими операторами:

IS NOT NULL;

NOT BETWEEN;
NOT IN;
NOT LIKE;
NOT EXISTS;
NOT UNIQUE.

Упорядочение данных

Для упорядочения данных в выборке, полученной в результате выполнения запроса, используется предложение ORDER BY. Синтаксис оператора SELECT в этом случае будет следующим: SELECT {* | ALL | DISTINCT поле1, поле2, …, полеN} FROM таблица1 {, таблица2, …, таблицаN} WHERE условие ORDER BY поле {ASC | DESC}.

После ключевых слов ORDER BY указывается имя поля (полей), по которому производится сортировка. После имени каждого поля может быть указан режим сортировки:

- ASC – режим, используемый по умолчанию. При этом информация располагается в порядке возрастания значения указанного поля (для текстовых полей – в алфавитном порядке).
- DESC используется для вывода информации в порядке убывания значений указанного поля (для текстовых полей – в порядке, обратном алфавитному).

Например, чтобы отсортировать список преподавателей по алфавиту, следует использовать следующий запрос (Рис. 32):

SELECT Фамилия, Имя, Отчество FROM ПРЕПОДАВАТЕЛЬ ORDER BY Фамилия, Имя, Отчество.

Фамилия	Имя	Отчество
Егоров	Егор	Егорович
Иванов	Иван	Иванович
Петров	Петр	Петрович
Сидоров	Сергей	Сергеевич
*		

Рис. 32. Результат

Вместо имени поля в предложении ORDER BY можно использовать целое число, определяющее порядковый номер поля в списке после ключевого слова SELECT (если производится выборка всех полей таблицы с помощью символа *, то число указывает порядковый номер поля в таблице базы данных).

Использование вычисляемых полей

Язык SQL позволяет создавать вычисляемые поля в тексте запроса. Для реализации этой функции в запросе просто приводится выражение, в котором используются арифметические и математические операторы, а также имена полей в качестве переменных. В результате выполнения запроса с вычисляемыми полями выборки будет содержать не только ту информацию, которая содержится в таблицах базы данных, но и дополнительную информацию, полученную в результате вычисления заданного выражения.

Кроме математических операций в SQL поддерживается ряд строковых функций, выполняющий такие операции, как конкатенация строк, выделение подстроки, поиск подстроки внутри строки и ряд других. В запросах также могут применяться функции преобразования символьного типа в числовой, числового типа в символьный, символьного типа в дату и т. п.

При создании вычисляемого поля можно использовать следующие арифметические операторы:

- оператор сложения «**+**»;
- оператор вычитания «**–**»;
- оператор умножения «*****»;
- оператор деления «**/**».

Приоритет перечисленных операторов соответствует общепринятому: умножение и деление, затем сложение и вычитание. Порядком выполнения операторов можно управлять с помощью круглых скобок.

Рассмотрим пример использования вычисляемых полей. Пусть каждому преподавателю назначается премия в размере 20% от оклада (Рис. 33):

*SELECT Фамилия, Имя, Отчество, Оклад, Оклад*0.2 AS Премия FROM ПРЕПОДАВАТЕЛЬ, ДОЛЖНОСТЬ WHERE ПРЕПОДАВАТЕЛЬ.КодДолж = ДОЛЖНОСТЬ.КодДолж.*

Фамилия	Имя	Отчество	Оклад	Премия
Сидоров	Сергей	Сергеевич	100,00 р.	20,00 ₽
Петров	Петр	Петрович	120,00 р.	24,00 ₽
Иванов	Иван	Иванович	160,00 р.	32,00 ₽
Егоров	Егор	Егорович	160,00 р.	32,00 ₽

Рис. 33. Результат

Для отображения данных в денежном формате использовалась функция: Format (Оклад*0.2 ,'Currency').

Кроме арифметических операторов допускается использование ряда математических функций:

- ABS – вычисление абсолютного значения;
- ROUND – округление;
- SQR – извлечение квадратного корня;
- EXP – экспонента;
- LOG – натуральный логарифм;
- SIN, COS, TAN – тригонометрические функции.

Арифметические операторы и математические функции можно использовать как в списке полей после ключевого слова SELECT, так и в предложение, задающем условие выборки (WHERE).

Набор математических функций зависит от конкретной реализации языка SQL. Синтаксис одинаковых функций в разных реализациях также может различаться (например, функция вычисления квадратного корня может обозначаться либо SQR, либо SQRT).

Псевдонимы полей

В запросах SQL можно изменять имена полей. Задаваемые при этом новые имена называются псевдонимами (aliases). Их удобно применять при задании в запросе вычисляемых полей. С помощью псевдонимов этим полям можно присваивать осмысленные имена. Псевдоним помещается после имени поля или после вычисляемого выражения через ключевое слово AS.

Переименование поля с помощью псевдонима действительно только в пределах конкретного запроса.

В ранее представленном примере дан псевдоним вычисляемого поля «Премия».

Функции агрегирования

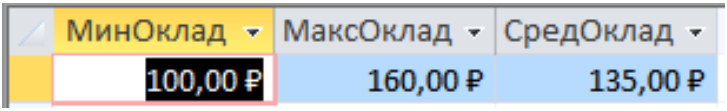
Функциями агрегирования называются функции, которые позволяют определить количество записей в таблице или количество значений в столбце таблицы, находят минимальное, максимальное и среднее значение для столбца таблицы, а также вычисляют сумму данных для столбца. Таким образом, агрегирующие функции обеспечивают получение некоторой обобщенной информации.

В SQL определены следующие стандартные функции агрегирования:

- COUNT – выполняет подсчет записей в таблице или подсчет ненулевых значений в столбце таблицы;
- SUM – возвращает сумму содержащихся в столбце значений;
- MIN – возвращает минимальное значение в столбце;
- MAX – возвращает максимальное значение в столбце;
- AVG – вычисляет среднее значение для содержащихся в столбце значений.

Найдем минимальный, максимальный и средний оклад для преподавателей (Рис. 34):

SELECT Min (Оклад) AS МинОклад, Max (Оклад) AS МаксОклад, Avg (Оклад) AS СредОклад FROM ПРЕПОДАВАТЕЛЬ, ДОЛЖНОСТЬ WHERE ПРЕПОДАВАТЕЛЬ.КодДолж = ДОЛЖНОСТЬ.КодДолж.



МинОклад	МаксОклад	СредОклад
100,00 ₽	160,00 ₽	135,00 ₽

Рис.34. Результат

Группировка данных

Группировка данных – это объединение записей в соответствии со значениями некоторого заданного поля. Для группировки результатов выборки совместно с оператором SELECT используется предложение GROUP BY. Данное предложение должно следовать после предложения WHERE, но перед предложением ORDER BY. После ключевых слов GROUP BY указывается список полей, включенных в выборку с помощью оператора SELECT. Причем нужно обязательно указывать все отбираемые поля (за исключением полей, относящихся к агрегирующим функциям), хотя порядок их перечисления после предложения GROUP BY может не соответствовать порядку списка полей после слова SELECT.

Синтаксис оператора SELECT с предложением GROUP BY следующий: SELECT поле1, поле2, …, полеN FROM таблицаI {, таблица2, … таблицаN} WHERE условие GROUP BY поле1, поле2, …, полеN ORDER BY полеI {ASC | DESC}.

Применение предложения GROUP BY без дополнительных функций дает такой же результат, как и применение предложения упорядочения ORDER BY.

Подсчитаем количество предметов, проводимых каждым преподавателем (Рис. 35):

SELECT Фамилия, Имя, Отчество, Count (ПРОВОДИТ.КодПреп) AS Количество FROM ПРЕПОДАВАТЕЛЬ, ПРОВОДИТ WHERE ПРЕПОДАВАТЕЛЬ.КодПреп = ПРОВОДИТ.КодПреп GROUP BY *Фамилия, Имя, Отчество.*

Фамилия	Имя	Отчество	Количество
Егоров	Егор	Егорович	2
Иванов	Иван	Иванович	3
Петров	Петр	Петрович	2
Сидоров	Сергей	Сергеевич	1

Рис. 35. Результат

Для задания ограничений на создаваемые группы совместно с ключевым словом GROUP BY может использоваться предложение HAVING. Оно должно следовать после GROUP BY, но до предложения ORDER BY (если оно присутствует в запросе).

Если мы хотим установить ограничение на общее количество предметов, то необходимо применить предложение HAVING (Рис. 36):

SELECT Фамилия, Имя, Отчество, Count (ПРОВОДИТ.КодПреп) AS Количество FROM ПРЕПОДАВАТЕЛЬ, ПРОВОДИТ WHERE ПРЕПОДАВАТЕЛЬ.КодПреп = ПРОВОДИТ.КодПреп GROUP BY *Фамилия, Имя, Отчество HAVING Count (ПРОВОДИТ.КодПреп)>1.*

Фамилия	Имя	Отчество	Количество
Егоров	Егор	Егорович	2
Иванов	Иван	Иванович	3
Петров	Петр	Петрович	2

Рис. 36. Результат

Выборка данных из нескольких таблиц

Как правило, информация, хранящаяся в базе данных, содержится в нескольких связанных между собой таблицах. Язык SQL позволяет создавать запросы, извлекающие данные из нескольких таблиц. При этом выполняется операция соединения,

состоящая в объединении нескольких таблиц с целью поиска в них запрошенных данных.

Существует несколько способов соединения таблиц. Наиболее часто встречаются следующие:

- соединение равенства;
- соединение неравенства;
- внешние соединения.

Для задания вида соединения используется предложение WHERE, в котором вид соединения указывается с помощью операторов сравнения или логических операторов.

Соединение равенства

Данное соединение является наиболее часто используемым. Соединение равенства обычно производится по общему для нескольких таблиц полю (которое, как правило, является первичным ключом).

Синтаксис оператора выборки для этого способа соединения таблиц будет следующим: SELECT таблица1.поле1, таблица2.поле2 {, ..., таблицаN.полеN} FROM таблица1, таблица2 {, …, таблицаN} WHERE таблица1.общее_поле1 = таблица2.общее_поле1 {AND таблица1.общее_поле2 = таблица2.общее_поле2}.

При формировании запроса на выборку из нескольких таблиц в списке полей после слова SELECT перед именем поля обычно указывается имя таблицы, к которой это поле относится. Такое действие называется квалификацией полей запроса. Квалификация обязательна только для полей, имеющих одинаковые имена в разных таблицах, из которых производится выборка.

Пример выборки из двух таблиц с использованием соединения равенства был показан при использовании вычисляемых полей.

При связывании таблиц можно использовать предложение группировки. Пример был показан при группировке данных.

Соединение неравенства

В случае применения соединения неравенства информация из двух таблиц объединяется таким образом, чтобы значения в заданном поле одной таблицы не совпадали со значениями соответствующего ему поля в другой таблице.

Синтаксис запроса при соединении неравенства аналогичен предыдущему случаю, только вместо оператора «=» в предложении WHERE используются операторы «<>», «<», «>» и т.п.: SELECT таблица1.поле1, таблица2.поле2 {, ..., таблицаN.полеN} FROM таблица1, таблица2 {, …, таблицаN} WHERE таблица1.общее_поле1 <> таблица2.общее_поле1 {AND таблица1. общее_поле2 > таблица2.общее_поле2}.

Соединения неравенства используются довольно редко.

Внешние соединения

При использовании внешнего соединения результат запроса будет содержать все записи одной из таблиц, даже в том случае, если в связанной с ней таблице отсутствуют совпадающие значения. Этот тип соединения реализуется с помощью оператора OUTER JOIN.

Внешнее соединения подразделяются на три группы:

▪ левое внешнее соединение (LEFT OUTER JOIN) – выборка будет содержать все записи таблицы, имя которой указано слева от оператора OUTER JOIN;

▪ правое внешнее соединение (RIGHT OUTER JOIN) – выборка будет содержать все записи таблицы, имя которой указано справа от оператора OUTER JOIN;

▪ полное внешнее соединение (FULL OUTER JOIN) – в выборку включаются все записи из правой и левой таблицы.

Для внешнего соединения условие соединения указывается не с помощью предложения WHERE, а входит в оператор OUTER JOIN после ключевого слова ON: SELECT таблица1.поле1,

таблица2.поле2 {, ... , таблицаN.полеN} FROM таблица1 LEFT | RIGHT | FULL {OUTER} JOIN таблица2 ON условие {LEFT | RIGHT | FULL {OUTER} JOIN таблица3 ON условие}.

Внутренние соединения

Внутреннее соединение двух таблиц INNER JOIN производит выборку только тех записей, которые есть в обеих таблицах. Порядок таблиц для оператора неважен, поскольку оператор является симметричным.

Заголовок таблицы-результата является объединением заголовков соединяемых таблиц, а тело результата логически формируется следующим образом: каждая строка одной таблицы сопоставляется с каждой строкой второй таблицы, после чего для полученной «соединённой» строки проверяется условие соединения. Если условие истинно, в таблицу-результат добавляется соответствующая «соединённая» строка.

Рассмотрим следующий пример. На основе наших таблиц, получим итоговую информацию (Рис. 37):

SELECT ПРЕПОДАВАТЕЛЬ.Фамилия, ПРЕПОДАВАТЕЛЬ.Имя, ПРЕПОДАВАТЕЛЬ.Отчество, ДОЛЖНОСТЬ.НаимДолж, ДОЛЖНОСТЬ.Оклад, КАФЕДРА.НаимКаф, ПРЕДМЕТ.НаимПред, ГРУППА.НаимГр, ВИДЗАНЯТИЙ.НаимВЗ FROM (КАФЕДРА INNER JOIN (ДОЛЖНОСТЬ INNER JOIN ПРЕПОДАВАТЕЛЬ ON ДОЛЖНОСТЬ.КодДолж = ПРЕПОДАВАТЕЛЬ.КодДолж) ON КАФЕДРА.КодКаф = ПРЕПОДАВАТЕЛЬ.КодКаф) INNER JOIN (ПРЕДМЕТ INNER JOIN (ГРУППА INNER JOIN (ВИДЗАНЯТИЙ INNER JOIN ПРОВОДИТ ON ВИДЗАНЯТИЙ.КодВЗ = ПРОВОДИТ.КодВЗ) ON ГРУППА.КодГр = ПРОВОДИТ.КодГр) ON ПРЕДМЕТ.КодПред = ПРОВОДИТ.КодПред) ON ПРЕПОДАВАТЕЛЬ.КодПреп = ПРОВОДИТ.КодПреп.

Фамилия	Имя	Отчество	НаимДолж	Оклад	НаимКаф	НаимПред	НаимГр	НаимВЗ
Иванов	Иван	Иванович	Доцент	160,00 р.	Кафедра информатики	Управление данными	191	Лекции
Иванов	Иван	Иванович	Доцент	160,00 р.	Кафедра информатики	Управление данными	191	Практики
Иванов	Иван	Иванович	Доцент	160,00 р.	Кафедра информатики	Операционные системы	192	Лекции
Петров	Петр	Петрович	Ст. преп.	120,00 р.	Кафедра информатики	Управление данными	191	Практики
Петров	Петр	Петрович	Ст. преп.	120,00 р.	Кафедра информатики	Операционные системы	192	Практики
Егоров	Егор	Егорович	Доцент	160,00 р.	Кафедра экономики	Экономика организации	191	Лекции
Егоров	Егор	Егорович	Доцент	160,00 р.	Кафедра экономики	Экономика организации	191	Практики
Сидоров	Сергей	Сергеевич	Ассистент	100,00 р.	Кафедра экономики	Экономика организации	191	Практики

Рис. 37. Результат

Подзапросы

Подзапрос представляет собой запрос, помещенный внутри другого запроса. Подзапросы применяются для получения данных, которые затем используются другим запросом. Результаты подзапроса можно использовать в предложение WHERE, для наложения ограничений на данные, включаемые в выборку; в предложение FROM, для указания источника, из которого производится выборка; в предложение SELECT, при указании списка полей, значения которых будут включаются в выборку.

Запрос, содержащий подзапрос, называется сложным. В процессе его выполнения сначала выполняется подзапрос, а затем – основной запрос. При создании сложного запроса необходимо следовать следующему набору правил:

- подзапросы должны заключаться в круглые скобки;
- предложение ORDER BY может быть использовано только в основном запросе;
- подзапросы, возвращающие более одной записи, могут использоваться только с многозначными операторами;
- в основном запросе нельзя использовать оператор BETWEEN.

Ниже приведен синтаксис оператора SELECT с подзапросом: SELECT {* | ALL | DISTINCT поле1, поле2, …, полеN} FROM таблица1 {, таблица2, …, таблицаN} WHERE условие (SELECT поле1 {, поле2, ..., полеN} FROM таблица1 {, таблица2, …, таблицаN} [WHERE условие]).

В подзапросе можно использовать более частные подзапросы. Максимальный уровень вложенности подзапросов определяется конкретной реализацией SQL.

Объединение запросов

Язык SQL позволяет объединять несколько запросов с помощью специальных операторов. Запросы, включающие в себя несколько операторов SELECT, принято называть составными.

Составные запросы формируют один набор данных на основе результатов, полученных при выполнении каждого отдельного запроса, входящего в объединение. Во многих случаях составные запросы целесообразно использовать вместо простых запросов со сложным условием выборки. Это связано с тем, что разбиение сложного условия на несколько более простых запросов делает текст запроса более понятным. Как правило, проще написать составной запрос, чем аналогичный простой запрос со сложным условием отбора данных.

Для объединения запросов наиболее часто используются операторы UNION и UNION ALL (предусмотренные стандартом ANSI).

В стандарте ANSI определены также и другие операторы объединения: EXCEPT и INTERSECT, которые расширяют возможности составных запросов.

При объединении запросов, независимо от типа используемых операторов объединения, необходимо следовать следующим правилам:

- каждый из запросов, входящих в объединение, должен возвращать одинаковое количество полей (в том числе и вычисляемых);
- типы полей, возвращаемых в результате выполнения каждого запроса, должны совпадать.

Оператор UNION

При использовании оператора UNION результаты выполнения отдельных запросов объединяются. При этом дублирующие друг друга записи исключаются из результирующего набора данных.

Например, запрос:

*SELECT * FROM ДОЛЖНОСТЬ WHERE (Оклад <=110) OR (Оклад >=150)*

преобразуем в запрос, с использованием оператора UNION (Рис. 38):

*SELECT * FROM ДОЛЖНОСТЬ WHERE Оклад <=110*

UNION

*SELECT * FROM ДОЛЖНОСТЬ WHERE Оклад >=150.*

КодДолж	НаимДолж	Оклад
1	Ассистент	100,00 р.
3	Доцент	160,00 р.

Рис. 38. Результат

Оператор UNION ALL

Данный оператор аналогичен оператору UNION, за исключением того что в результирующую выборку включаются дублирующие записи.

Упорядочение и группировка данных в составных запросах

В составном запросе для упорядочения данных допускается использование предложения ORDER BY. Независимо от того сколько запросов входит в объединение, можно использовать только одно предложение ORDER BY. Для указания полей, по которым производится сортировка, в этом предложении допускается использование как имен полей, так и их порядковых номеров в списке оператора SELECT.

В отличие от ORDER BY предложение GROUP BY можно применять в каждом из запросов, входящих в объединение. Вместе с GROUP BY допускается применение оператора HAVING. Предложение GROUP BY можно применять и для группировки результатов исполнения составного запроса.

Вопросы и задания для самопроверки

1. На какие группы команд подразделяются команды языка SQL.
2. Перечислите типы данных, используемые в стандартном SQL.
3. Составьте оператор, создающий таблицу ДОЛЖНОСТЬ (Код-Долж, НаимДолж, Оклад).
4. Каким образом можно изменить тип данных поля *Оклад* в таблице ДОЛЖНОСТЬ.
5. Укажите, для чего используются ограничение NOT NULL.
6. Каким образом можно указать первичный ключ для таблицы.
7. Составьте ограничения внешнего ключа для примера из раздела 3.
8. Какие уникальные индексы можно предложить для примера из раздела 3.
9. Приведите примеры запросов на SQL.

Рекомендуемая литература

1. Microsoft Access – программа для работы с базами данных [Электронный ресурс]. – Режим доступа: http://office.microsoft.com/ru-ru/access/ (01.12.2014).
2. Борзунова, Т.Л. Базы данных освоение работы в MS Access 2007 [Электронный ресурс]: электронное пособие / Т.Л. Борзунова, Т.Н. Горбунова, Н.Г. Дементьева – Электрон. текстовые данные. – Саратов: Вузовское образование, 2014. – 148 c. – Режим доступа: http://www.iprbookshop.ru/20700. – ЭБС «IPRbooks», по паролю.
3. Дейт, К.Дж. Введение в системы баз данных [Текст] / К.Дж. Дейт. – М.: Издательский дом «Вильямс», 2006. – 1 328 с.
4. Кириллов, В.В. Основы проектирования реляционных баз данных [Электронный ресурс] / В.В. Кириллов. – Режим доступа: http://citforum.ru/database/dbguide/index.shtml (01.12.2014).

5. Королева, О.Н. Базы данных [Электронный ресурс]: курс лекций / О.Н. Королева, А.В. Мажукин, Т.В. Королева. – Электрон. текстовые данные. – М.: Московский гуманитарный университет, 2012. – 66 с. – Режим доступа: http://www.iprbookshop.ru/14515. – ЭБС «IPRbooks», по паролю.
6. Крис Фиайли SQL [Электронный ресурс] / Крис Фиайли. – Электрон. текстовые данные. – М.: ДМК Пресс, 2007. – 451 с. – Режим доступа: http://www.iprbookshop.ru/6918. – ЭБС «IPRbooks», по паролю.
7. Кузнецов, С.Д. Базы данных. Вводный курс [Электронный ресурс] / С.Д. Кузнецов. – Режим доступа: http://citforum.ru/database/advanced_intro/ (01.12.2014).
8. Кузнецов, С.Д. Основы современных баз данных [Электронный ресурс] / С.Д. Кузнецов. – Режим доступа: http://citforum.ru/database/osbd/contents.shtml (01.12.2014).
9. Кузнецов, С.Д. Ландшафт области управления данными [Электронный ресурс]: аналитический обзор / С.Д. Кузнецов, М.Н. Гринев. – Режим доступа: http://citforum.ru/database/data_management_overview/ (01.12.2014).
10. Кузовкин, А.В. Управление данными [Текст]: учебник для студ. высших учеб. заведений / А.В. Кузовкин, А.А. Цыганов, Б.А. Щукин. – М.: Издательский центр «Академия», 2010. – 256 с.
11. Основы современных баз данных [Электронный ресурс]: метод. разработка к выполнению лабораторных работ. – Электрон. текстовые данные. – № 1–3. – Липецк: Липецкий государственный технический университет, ЭБС АСВ, 2013. – 37 с. – Режим доступа: http://www.iprbookshop.ru/22906. – ЭБС «IPRbooks», по паролю.
12. Полякова, Л.Н. Основы SQL [Электронный ресурс] / Л.Н. Полякова. – Электрон. текстовые данные. – М.: Интернет-Университет

Информационных Технологий (ИНТУИТ), 2007. – 187 с. – Режим доступа: http://www.iprbookshop.ru/22421. – ЭБС «IPRbooks», по паролю.
13. Пушников, А.Ю. Введение в системы управления базами данных [Электронный ресурс] / А.Ю. Пушников. – Режим доступа: http://citforum.ru/database/dblearn/index.shtml (01.12.2014).
14. СУБД. Язык SQL в примерах и задачах [Электронный ресурс]: учеб. пособие / И.Ф. Астахова [и др.]. – Электрон. текстовые данные. – М.: ФИЗМАТЛИТ, 2009. – 168 с. – Режим доступа: http://www.iprbookshop.ru/12971. – ЭБС «IPRbooks», по паролю.
15. Ульман, Дж. Введение в системы баз данных [Текст] / Дж. Ульман, Дж. Уидом. – М.: Лори, 2006. – 379 с.
16. Хомоненко, А.Д. Базы данных [Текст]: учебник для высш. учеб. заведений / А.Д. Хомоненко. – СПб.: КОРОНА – Век, 2009. – 736 с.
17. Швецов, В.И. Базы данных [Электронный ресурс] / В.И. Швецов. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2009. – 155 с. – Режим доступа: http://www.iprbookshop.ru/16688. – ЭБС «IPRbooks», по паролю.

Заключение

В процессе изучения курса «Управление данными» студенты вначале осваивают основные положения теории баз данных, затем изучают реляционную модель данных и процесс проектирования баз данных, используя теорию нормализации. Как инструмент управления данными студенты изучают стандартизированный язык баз данных SQL. Все основные команды языка баз данных SQL демонстрируются примерами в СУБД Microsoft Office Access 2013.

Вопросы для самопроверки и задания позволяют оценить степень усвоения материала соответствующего раздела и почерпнуть дополнительный материал.

Таким образом, все поставленные перед автором задачи были успешно решены, а студенты освоят необходимые знания и навыки, выделенные в Федеральном государственном образовательном стандарте высшего профессионального образования.

Материал учебного пособия может быть полезен и студентам других направлений подготовки, изучающих дисциплины «Базы данных» и ей подобные, а также учителям и педагогам образовательных организаций.

Библиографический список

1. Microsoft Access – программа для работы с базами данных [Электронный ресурс]. – Режим доступа: http://office.microsoft.com/ru-ru/access/ (01.12.2014).
2. Борзунова, Т.Л. Базы данных освоение работы в MS Access 2007 [Электронный ресурс]: электронное пособие / Т.Л. Борзунова, Т.Н. Горбунова, Н.Г. Дементьева – Электрон. текстовые данные. – Саратов: Вузовское образование, 2014. – 148 с. – Режим доступа: http://www.iprbookshop.ru/20700. – ЭБС «IPRbooks», по паролю.
3. Громов, Ю.Ю. Управление данными [Текст]: учеб. пособие / Ю.Ю. Громов, О.Г. Иванова, В.Н. Точка. – Тамбов: Изд-во Тамб. гос. техн. ун-та, 2009. – 80 с.
4. Дейт, К.Дж. Введение в системы баз данных [Текст] / К.Дж. Дейт. – М.: Издательский дом «Вильямс», 2006. – 1 328 с.
5. Джексон, Г. Проектирование реляционных баз данных для использования с микроЭВМ [Текст] / Г. Джексон. – М.: Мир, 1991. – 252 с.
6. Кириллов, В.В. Основы проектирования реляционных баз данных [Электронный ресурс] / В.В. Кириллов. – Режим доступа: http://citforum.ru/database/dbguide/index.shtml (01.12.2014).
7. Королева, О.Н. Базы данных [Электронный ресурс]: курс лекций / О.Н. Королева, А.В. Мажукин, Т.В. Королева. – Электрон. текстовые данные. – М.: Московский гуманитарный университет, 2012. – 66 с. – Режим доступа: http://www.iprbookshop.ru/14515. – ЭБС «IPRbooks», по паролю.
8. Крис Фиайли SQL [Электронный ресурс] / Крис Фиайли. –

Электрон. текстовые данные. – М.: ДМК Пресс, 2007. – 451 с. – Режим доступа: http://www.iprbookshop.ru/6918. – ЭБС «IPRbooks», по паролю.
9. Кузнецов, С.Д. Основы современных баз данных [Электронный ресурс] / С.Д. Кузнецов. – Режим доступа: http://citforum.ru/database/osbd/contents.shtml (01.12.2014).
10. Кузнецов, С.Д. Базы данных. Вводный курс [Электронный ресурс] / С.Д. Кузнецов. – Режим доступа: http://citforum.ru/database/advanced_intro/ (01.12.2014).
11. Кузнецов, С.Д. Ландшафт области управления данными [Электронный ресурс]: аналитический обзор / С.Д. Кузнецов, М.Н. Гринев. – Режим доступа: http://citforum.ru/database/data_management_overview/ (01.12.2014).
12. Кузовкин, А.В. Управление данными [Текст]: учебник для студ. высших учеб. заведений / А.В. Кузовкин, А.А. Цыганов, Б.А. Щукин. – М.: Издательский центр «Академия», 2010. – 256 с.
13. Основы современных баз данных [Электронный ресурс]: метод. разработка к выполнению лабораторных работ. – Электрон. текстовые данные. – № 1–3. – Липецк: Липецкий государственный технический университет, ЭБС АСВ, 2013. – 37 с. – Режим доступа: http://www.iprbookshop.ru/22906. – ЭБС «IPRbooks», по паролю.
14. Полякова, Л.Н. Основы SQL [Электронный ресурс] / Л.Н. Полякова – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2007. – 187 с. – Режим доступа: http://www.iprbookshop.ru/22421. – ЭБС «IPRbooks», по паролю
15. Пушников, А.Ю. Введение в системы управления базами данных [Электронный ресурс] / А.Ю. Пушников. – Режим доступа: http://citforum.ru/database/dblearn/index.shtml (01.12.2014).
16. СУБД. Язык SQL в примерах и задачах [Электронный ресурс]: учеб. пособие / И.Ф. Астахова [и др.]. – Электрон. текстовые

данные. – М.: ФИЗМАТЛИТ, 2009. – 168 с. – Режим доступа: http://www.iprbookshop.ru/12971. – ЭБС «IPRbooks», по паролю
17. Туманов, В.Е. Основы проектирования реляционных баз данных [Электронный ресурс]: учебное пособие / В.Е. Туманов. – Электрон. текстовые данные. – М.: БИНОМ. Лаборатория знаний, Интернет-Университет Информационных Технологий (ИНТУИТ), 2007. – 420 с. – Режим доступа: http://www.iprbookshop.ru/22431. – ЭБС «IPRbooks», по паролю.
18. Ульман, Дж. Введение в системы баз данных [Текст] / Дж. Ульман, Дж. Уидом. – М.: Лори, 2006. – 379 с.
19. Хомоненко, А.Д. Базы данных [Текст]: учебник для высш. учеб. заведений / А.Д. Хомоненко. – СПб.: КОРОНА – Век, 2009. – 736 с.
20. Швецов, В.И. Базы данных [Электронный ресурс] / В.И. Швецов. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2009. – 155 с. – Режим доступа: http://www.iprbookshop.ru/16688. – ЭБС «IPRbooks», по паролю.

Учебное издание

Рузаков Андрей Александрович

Управление данными

Учебное пособие

ISBN 978-5-906777-40-9

Работа рекомендована РИСом ЧГПУ
Протокол № 7 (пункт 25) от 25.12.2014

Издательство ЧГПУ
454080, г. Челябинск, пр. Ленина, 69

Редактор Л.Н. Корнилова
Компьютерный набор А.А. Рузаков

Объем 6 уч.-изд. л. Подписано в печать 25.05.2015
Формат 60х84/16. Бумага офсетная
Тираж 100 экз. Заказ №

Отпечатано с готового оригинал-макета в типографии ЧГПУ
454080, г. Челябинск, пр. Ленина, 69