

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»**

КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И ПРОГРАММИРОВАНИЯ

О.Д. МЕРДИНА

БАЗЫ ДАННЫХ

Учебное пособие

**ИЗДАТЕЛЬСТВО
САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО
ЭКОНОМИЧЕСКОГО УНИВЕРСИТЕТА
2019**

ББК 32.81
М52

Мердина О.Д.

М52 Базы данных : учебное пособие / О.Д. Мердина. – СПб. : Изд-во СПбГЭУ, 2019. – 99 с.

ISBN 978-5-7310-4597-7

Цель учебного пособия – дать студентам необходимые знания по теоретическим основам баз данных, методам проектирования концептуальной модели базы данных, средствам разработки в среде современной СУБД Microsoft SQL Server. В пособии рассматривается пример проектирования и разработки учебной базы данных. Большое внимание уделено изучению основ структурированного языка запросов Transact-SQL, команд управления объектами базы данных и механизмов манипулирования данными. Для успешного освоения языка Transact-SQL в пособии для учебной базы данных приведено большое количество примеров.

Учебное пособие подготовлено в соответствии с программой обучения по дисциплине «Базы данных» и предназначено для направления подготовки бакалавров 10.03.01 «Информационная безопасность», может представлять интерес для бакалавров других направлений при изучении смежных дисциплин.

The purpose of the tutorial – to give students the necessary knowledge on the theoretical foundations of databases, methods of designing a conceptual database model, development tools in a modern database Microsoft SQL Server. The manual describes an example of the design and development of a training database. Much attention is paid to the study of the basics of structured Transact-SQL query language, database object management commands and data manipulation mechanisms. For the successful development of the Transact-SQL language in the manual for the training database, a large number of examples are given.

ББК 32.81

Рецензенты: канд. техн. наук, доцент **Г.М. Чернокнижный**
канд. экон. наук, доцент **И.Г. Гниденко**

ISBN 978-5-7310-4597-7

© СПбГЭУ, 2019

ОГЛАВЛЕНИЕ

Введение	4
1. ТЕОРИЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ	5
1.1. История развития, назначение и роль баз данных	5
1.2. Основы реляционной модели данных	14
1.3. Языки манипулирования данными в реляционной модели	19
2. СИСТЕМА УПРАВЛЕНИЯ РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ	22
2.1. Общие принципы построения СУБД	22
2.2. Архитектура СУБД	28
2.3. Компоненты СУБД Microsoft SQL Server	33
2.4. Архитектура базы данных SQL Server	37
3. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ	47
3.1. Этапы и методы проектирование реляционных баз данных	47
3.2. Семантическая модель данных	58
4. ЯЗЫК TRANSACT-SQL	71
4.1. Введение в Transact -SQL	71
4.2. Язык описания данных (DDL)	77
4.3. Язык манипулирования данными (DML)	82
4.4. Средства разработки процедур в Transact-SQL	88
5. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РЕЛЯЦИОННЫХ СУБД	91
5.1. Системы, ориентированные на анализ данных	91
Библиографический список	99

ВВЕДЕНИЕ

Целью освоения дисциплины «Базы данных» является получение студентами необходимых знаний базового уровня в области теоретических основ баз данных, проектирования, создания и администрирования баз данных. Учебное пособие предназначено для студентов, обучающихся по направлению 10.03.01 – «Информационная безопасность». Понимание логической и физической организации базы данных является очень важным для выявления возможных уязвимостей в системах баз данных.

Особое внимание в пособии уделено вопросу проектирования базы данных, построению концептуальной модели. Владея знаниями о методах проектирования, правилах организации структур данных, правилах ограничений целостности, которым должны соответствовать элементы данных или связанные между собой данные, специалист в области безопасности базы данных сможет противодействовать угрозам, связанным с нарушением конфиденциальности данных.

Значительное внимание уделено в пособии общим принципам организации и структуре системы управления базами данных Microsoft SQL Server, которая на сегодняшний день является одной из наиболее популярной среди СУБД, используемых для создания и поддержки базы данных в информационных системах.

В четвертом разделе учебного пособия изложены основы структурированного языка запросов Transact-SQL, команды определения и манипулирования данными. Владение этим языком, умение разрабатывать и отлаживать запросы к SQL-серверу является обязательным требованием к профессиональной подготовки специалистов информационной безопасности в области баз данных.

Материал учебного пособия подготовлен в соответствии с рабочей программой дисциплины «Базы данных» и достаточно полно раскрывает темы дисциплины. В теоретико-методологическом направлении материал учебного пособия связан с темами дисциплины «Защита баз данных» учебного плана студентов, обучающихся по направлению 10.03.01 – «Информационная безопасность».

1. ТЕОРИЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

1.1. История развития, назначение и роль баз данных

Базы данных нашли применение в информационных системах практически всех типов и уровней. Информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса.

Первые информационные системы представляли собой системы обработки данных, хранящихся в файлах. Особенностью таких систем являлось хранение данных в файлах, имеющих определенную структуру.

В файловых системах обработки данных использовались файлы трех типов, которые отличались между собой по способу организации и доступу к данным:

Файлы с последовательным доступом. Создание систем обработки данных, использующих для хранения данных файлы последовательного доступа, было связано с имеющимися на тот момент устройствами хранения данных, которые могли обеспечить только последовательный доступ к данным. Структурно файл состоял из записей разной длины, содержащих данные.

Появление внешних запоминающих устройств произвольного доступа, таких как магнитные диски, позволило создавать файлы с произвольным доступом и индексно последовательные файлы.

Файлы с произвольным доступом. Организации данных в файлах с произвольным доступом имела следующие особенности:

- данные в файле располагались в предварительно определенных записях с полями заданной длины и типа;
- все записи файла имели одинаковую длину;
- каждая следующая запись начиналась через каждые n байтов (n – длина записи) после предыдущей.

Такая организация позволяла обеспечивать к данным как последовательный, так и прямой доступ по номеру записи.

Индексно-последовательные файлы. В индексно-последовательных файлах появилось новое понятие – ключ. Ключ – это поле записи, значение которого идентифицирует запись. Таким образом в индексно-последовательных файлах стал возможен прямой доступ к записи файла по значению ключа.

В системах обработки данных решение конкретной задачи выполнялось отдельной прикладной программой. Прикладные программы разрабатывались с использованием методов процедурного программирования.

Недостатки файловых систем:

- *Большие трудозатраты программистов.* Для решения новой задачи в системе требовалось написание новой прикладной программы.

- *Избыточность данных.* Использование прикладной программой большого числа файлов с данными снижало надежность работы, приводило к замедлению расчетов, поэтому проще было продублировать нужное поле в файле, чем подключать файл, его содержащий. Наличие избыточного дублирование данных приводит к появлению несогласованных данных, к снижению качества информации, хранящейся в файловой системе.

- *Управление данными осуществлялось с помощью прикладных программ.* Для каждого файла требовалась собственная система управления, состоящая из программ создания файла, добавления данных в файл, удаления данных из файла и их изменения. А при изменении структуры данных файла требовалось вносить изменения во все прикладные программы, использующие этот файл.

Процесс поддержание данных в файловых системах был очень сложным. Разработчики программного обеспечения пришли к необходимости использовать другой подход к управлению данными. Этот подход был реализован в рамках новых программных систем, названных впоследствии системами управления базами данных.

Система управления базами данных (СУБД) – это комплекс программных и языковых средств, необходимых для создания баз данных, поддержки их в актуальном состоянии и организации поиска информации в них.

Первой промышленной СУБД стала система IMS (Information Management System — система управления информацией), которая была введена в эксплуатацию в 1968 году фирмой IBM для больших электронно-вычислительных машин (ЭВМ).

Использование персональных компьютеров в информационных системах привело к появлению целого семейства СУБД, представителями которых являлись Dbase (DbaseIII+, DbaseIV), FoxPro, Paradox, система программирования Clipper.

Современные СУБД позволяют реализовать все современные технологии работы с данными, в информационных системах любой архитектуры. В пятерку самых популярных по данным за 2018 год вошли СУБД Oracle, MySQL, MS SQL SERVER, PostgreSQL, mongoDB. [12]

Объемы информации, которые приходится хранить в базах данных информационных систем достаточно велики, а сама информация имеет как правило достаточно сложную структуру.

База данных – это поименованная совокупность структурированных данных, относящихся к определенной предметной области.

В широком смысле база данных – совокупность сведений об объектах предметной области. Если говорить о системах обработки данных, то предметная область определяется тем множеством объектов, информация о которых необходима для реализации связанных между собой функций, задач управления, с помощью которых достигается выполнение поставленной цели.

Чтобы организовать быстрый поиск информации в базах данных ее нужно хранить в упорядоченном, структурированном виде. **Структурирование** – это введение соглашений о способах представления данных.

Примером неструктурированных данных может служить текстовый файл, созданный, например, приложением Windows Блокнот и содержащий в каждой строке следующие сведения о студенте: ФИО, адрес проживания, телефон, дата рождения, записанные в произвольном порядке их следования. Если оговорить единый порядок следования сведений для каждого студента и формат представления данных, сохранить их, например, в таблице, то полученные данные будут структурированными.

Эффективность работы СУБД с базой данных во многом определяется тем, что из себя представляют структуры данных и как они взаимосвязаны между собой.

Модель базы данных представляет собой совокупность логических конструкций для представления структур данных и отношений между ними. Различают концептуальные модели базы данных и модели реализации баз данных. [2]

Концептуальная модель базы данных отражает логическую природу данных, предназначенных для хранения в базе данных.

Модель реализации базы данных определяет логическую структуру данных, то есть способ представления данных в базе данных.

База данных должна хранить данные, предназначенных для решения прикладных задач разных пользователей. Поэтому **концептуальная модель** должна представлять обобщенное представление всех пользователей о данных предметной области.

Однако при создании базы данных необходимо также определить какие устройства будут использоваться для хранения данных, какие способы будут использоваться для размещения данных в памяти и т.д. Все эти данные не имеют отношения к концептуальной модели, но требуют описания.

В 1975 г. Комитетом планирования стандартов и норм Национального института стандартизации США ANSI (American National Standards Institute) предложена трехуровневая архитектура базы данных, определяющая три различных уровня описания данных (внешний, концептуальный, внутренний). Основная цель этой архитектуры состоит в отделении пользовательского представления о данных в базе данных от их физического

представления. Структура трехуровневой модели базы данных представлена на рисунке 1.

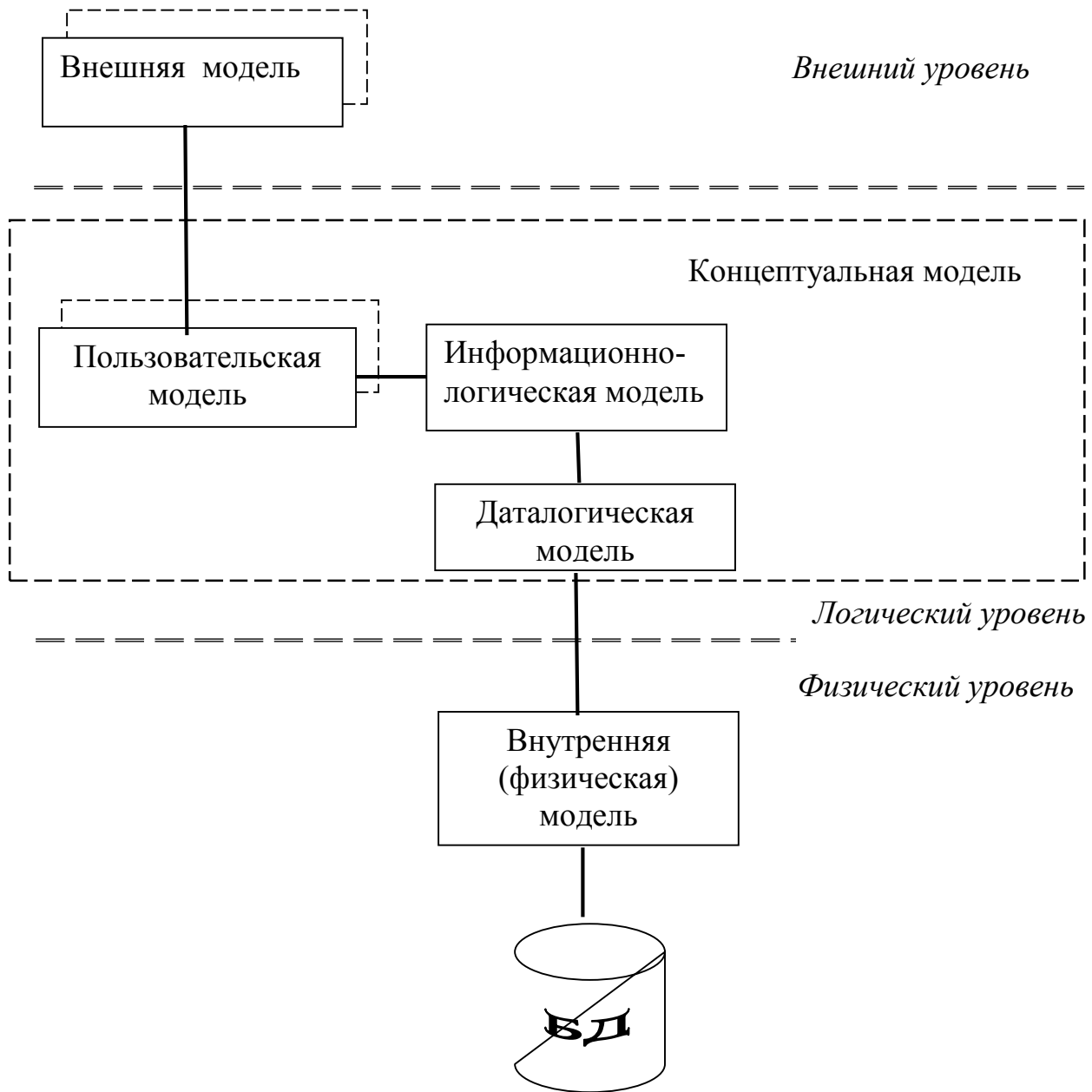


Рис. 1. Трехуровневая архитектура базы данных

На логическом уровне **концептуальная модель** представлена моделями.

Информационно-логическая модель (инфологическая модель) — это модель отображения предметной области в виде информационных объектов и связей между ними. Под информационным объектом понимается абстрактный объект, информацию о свойствах которого требуется хранить в БД. Такими информационными объектами, например, могут

быть: СТУДЕНТ, Дисциплина, Оценка. Инфологическая модель определяет схему базы данных.

Пользовательская модель – это модель, отражающая интересы конкретного пользователя с точки зрения информации о предметной области. В определенном смысле, пользовательская модель – это вся инфологическая модель или некоторая ее часть, которая называется также подсхемой базы данных.

Даталогическая модель – это модель, ориентированная на реализацию БД в конкретной СУБД, т.е. это инфологическая модель, трансформированная с учетом требований и ограничений конкретной СУБД.

На внешнем уровне – уровне взаимодействия конечного пользователя с базой данных определена **внешняя модель**. Это модель, соответствует пользовательской модели с конкретным отображением информационных потребностей пользователя (какие данные и как должны быть представлены, каковы процедуры обработки данных и т.д.).

Внутренняя модель – это модель, которая отражает используемые запоминающие устройства, способы расположения элементов данных в памяти и физической реализации логических связей между ними.

Такое представление данных позволяет обеспечить логическую и физическую независимость при работе с данными. Изменение одной клиентской программы может привести к изменению внешнего представления данных, однако логическая модель всей базы данных остается неизменной и, соответственно, не потребуются изменения остальных клиентских программ. А изменения в физической модели не влияют на логическую модель и соответственно не требуют изменения клиентских программ.

Каким образом данные объединяются в структуры определяются моделью представления (реализации) данных. С момента появления в 70-х годах прошлого века первой СУБД и до сегодняшнего дня модели данных прошли свой путь развития. Некоторые модели данных утратили свою актуальность и в настоящее время не используются. Однако понимание того, каким образом определялись структуры данных, какие механизмы использовались для манипулирования данными позволяют лучше понять методы проектирования баз данных. На сегодняшний день известны следующие модели данных: иерархическая; сетевая; объектно-ориентированная; реляционная; семантическая.

Первые СУБД использовали иерархическую модель данных. Иерархическая модель базы данных основана на структуре, которая по сути является графом, организованным по принципу дерева. Приведем основные понятия иерархической модели.

Элемент данных - характеристика информационного объекта;

Сегмент – это структура, которая размещается в узле графа и представляет собой совокупность элементов данных, описывающих конкретный информативный объект;

Связь между сегментами (ребро графа) описывает логическую связь между двумя информационными объектами.

Отличительные признаки иерархической структуры:

- на верхнем уровне иерархии существует только один сегмент, корневой;
- каждый сегмент в иерархической структуре является либо главным, либо подчиненным, либо и тем и другим одновременно;
- любой подчиненный сегмент непосредственно взаимодействует с одним и только одним главным для него сегментом (расположенным на предшествующем уровне иерархии);
- связи между сегментами одного уровня отсутствуют;
- к каждому сегменту существует только один путь от корневого сегмента.

С каждым сегментом связано два понятия:

- тип сегмента (состоит из элементов данных);
- экземпляр сегмента (состоит из значений элементов данных).

Например, для информационного объекта Студент может быть определен сегмент с типом, состоящим из Номер зачетной книжки (НЗК), ФИО, Дата рождения.

Тогда экземплярами сегмента будут являться следующие значения:

12345, Иванов И.И., 20.01.1999

12346, Кузмин А.Л., 30.04.2001

Физическая запись базы данных (дерево) представляет собой совокупность всех сегментов, подчиненных экземпляру одного корневого сегмента.

Иерархическая база данных – это упорядоченный набор всех экземпляров одного типа дерева.

На рисунке 2 приведен пример иерархической структуры и физической записи, соответствующей этой структуре.

Иерархические связи между сегментами определялись самой моделью базы данных, поэтому в базе данных не могло существовать экземпляра сегмента Студент, не принадлежащего никакому экземпляру сегмента Группа, также не могло существовать экземпляра сегмента Группа без его отношения к какому-либо экземпляру сегмента Специальность и т.д. Такая организация данных являлась преимуществом модели. Однако, с увеличением количества сегментов модель становилась громоздкой,

сложной с точки зрения работы с данными. Помимо обычных операций манипулирования иерархически организованными данными, такими как изменение, добавление и удаление данных необходимы были и навигационные операции: найти указанное дерево БД, перейти от одного сегмента к другому внутри дерева (например, от группы к первому студенту), перейти от одного дерева к другому и т.д.



Рис. 2. Пример иерархической структуры базы данных и соответствующей ей физической записи

Кроме Information Management System (IMS) на иерархической модели данных основано было сравнительно ограниченное количество СУБД, в числе которых можно назвать зарубежные системы PC / Focus, Team - Up и Data Edge, а также отечественные системы Ока, ИНЭС и МИРИС.

Практическая работа с иерархическими базами данных выявила главный недостаток - невозможность организации связи между сегментами одного уровня, между сегментом и несколькими сегментами предшествующего уровня. Эти ограничения были сняты в сетевой модели данных.

Сетевая модель базы данных также основывается на понятиях узел, уровень, связь (рис. 3), однако в этой модели любой элемент любого уровня может быть связан с любым другим элементом.

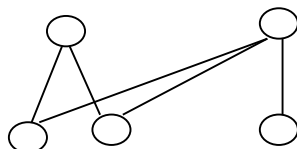
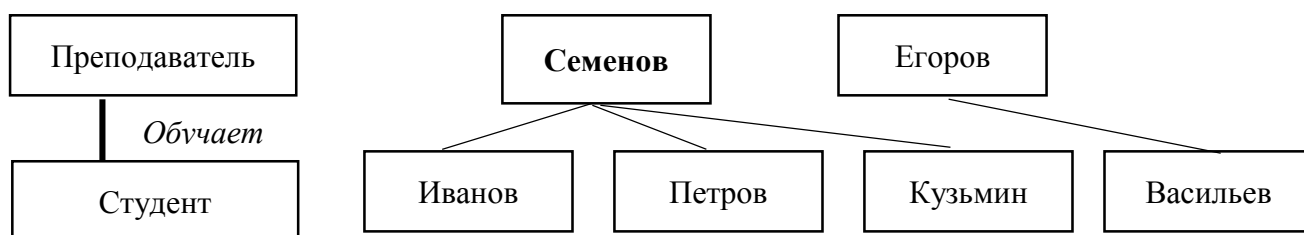
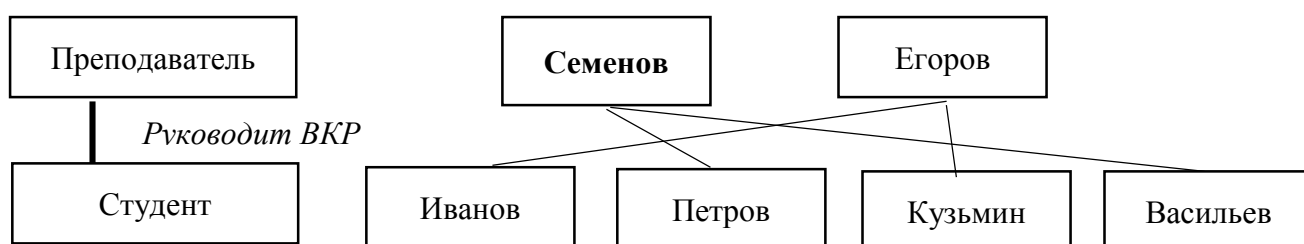


Рис. 3. Графическое представление сетевой модели данных

Сетевая база данных представляет собой множество наборов записей. Каждый набор из этого множества представляет собой двухуровневый граф, связывающий два типа записей. Запись, соответствует понятию сегмент иерархической модели и представляет информационный объект, например, Студент. Одна из записей является основной (главной) и является владельцем набора. Запись второго типа является зависимой (подчиненной) и называется членом набора. Между двумя типами записей может быть определено любое количество наборов (рис. 4).



а) Связь *Обучает* между типами записей *Преподаватель* и *Студент*



б) Связь *Руководит ВКР* между типами записей *Преподаватель* и *Студент*

Рис. 4. Наборы записей семантической базы данных.

Из приведенного на рисунке 4 примера видно, что каждый экземпляр владельца набора связан только с теми экземплярами подчиненной записи, которые соответствуют логике связи. Из примера также видно, что экземпляр записи Студент со значением Кузьмин в разных наборах связан с разными

экземплярами записи Преподаватель (Семенов и Егоров), т.е. имеет несколько записей-владельцев.

Сетевая модель данных позволила реализовать связи между записями расположенными в любом месте структурной схемы базы данных, что в конечном счете привело к высокой сложности и жесткости схемы базы данных и явилась ее недостатком.

Типичным представителем систем, основанных на сетевой модели данных, являлась СУБД IDMS (Integrated Database Management System), разработанная компанией Cullinet Software. Архитектура системы основана на предложениях Data Base Task Group (DBTG) организации CODASYL (COncference on DAta SYstems Languages). Наиболее известными сетевыми СУБД являлись также db_VistaIII, СЕТЬ, СЕТОР и КОМПАС.

Поиски новых подходов в организации структур данных привели к появлению новой модели, которая получила название «реляционная модель данных». Основные положения этой модели будут изложены ниже.

Развитие и широкое применение объектно-ориентированного подхода в программировании (ООП) привело к появлению объектно-ориентированной модели.

Основные понятия объектно-ориентированной модели:

объект – информационный объект описывающий некоторую сущность реального мира (например, студента Иванова);

свойства объекта - элементы – данные, являющиеся характеристиками объекта;

методы объекта- действия над свойствами объекта;

класс - описывает объект. Схожие по описанию объекты группируются в класс, который содержит описание всех данных (свойств объекта) и действий (методов). Например, класс Студент описывает все характеристики (свойств), значения которых должны храниться для каждого студента. Набор значений свойств конкретного студента является экземпляром класса (это и есть объект класса).

Классы организуются в иерархию и образуют модель базы данных. Каждый объект класса наследует свойства своего класса и всех классов, которые в иерархии классов расположены выше.

Использование методов наследование, инкапсуляция и полиморфизм ООП позволило в объектно-ориентированной модели данных упростить механизмы манипулирования данными.

Основным достоинством объектно-ориентированной модели данных является способность отображать информацию о сложных объектах с исчерпывающим описанием взаимосвязей между ними. Обычно модель применяется для сложных предметных областей, для моделирования которых не хватает функциональности реляционной модели.

Гибридом реляционной и объектно-ориентированной моделей является объектно-реляционная модель данных. Основная идея объектно-реляционного подхода - это разрешение использовать атрибуты не только простых типов данных, но и составные, и абстрактные типы данных, что противоречит классической концепции реляционной теории.

Объектно-реляционные СУБД сочетают в себе преимущества современных объектно-ориентированных языков программирования с функциями реляционных баз данных.

1.2. Основы реляционной модели данных

В 1970 году вышла статья Кодда Е.Ф., в которой был изложен новый подход к организации баз данных, что послужило началом создания новой модели данных – реляционной. Свое название модель получила от английского термина *relation* (отношение). Понятие *отношение* используется как для представления сущностей, так и для представления связей между ними.

Согласно теории реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: структурной, манипуляционной и целостной части.

В **структурной части модели** утверждается, что единственной структурой данных, используемой в реляционных базах данных, является нормализованное *n-арное отношение*. Для того, чтобы понять, что собой представляет отношение нужно дать определение понятиям атрибут, домен, кортеж, заголовок и тело отношения.

Атрибут – это любая характеристика информационного объекта. Например, атрибутами информационного объекта *Студент* являются такие характеристики как *Номер зачетной книжки*, *Фамилия студента*.

Тип данных является характеристикой атрибутов и определяет какими могут быть значения атрибутов и сколько места следует отводить для их хранения.

Современные реляционные базы данных позволяют хранить данные стандартного набора простых типов: текстовые, числовые, даты, время и др. Если определить атрибут *Возраст* информационного объекта *Студент* на типе, предназначенном для хранения натурального числа в одном байте, то СУБД позволит присвоить ему любое из тех значений, которые можно сохранить в байте (от 0 до 255). Однако возраст студента определяется меньшим диапазоном допустимых значений, например, от 17 до 40.

В реляционной модели множество, значения которого могут быть значениями атрибута называется **доменом атрибута**.

Тогда домен атрибута определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат «истина», то элемент данных является элементом домена.

В нашем примере доменом атрибута *Возраст* будут значения определенные на типе данных - натуральное число, удовлетворяющие условию: $17 \leq \text{Возраст} \leq 40$

Рассмотрим определение отношения на примере. Пусть определены домен D1 атрибута *Фамилия* (A1), домен D2 атрибута *Дисциплина* (A2) и домен D3 атрибута *Оценка* (A3), тогда **n-арное отношение R** есть **подмножество декартова произведения множеств D1, D2, D3**.

Полное декартово произведение множеств дает набор всевозможных сочетаний из n (количество значений атрибута) элементов, где каждый элемент берется из своего домена.

Если исходные домены содержат значения:

$D1 = \{\text{Иванов; Петров}\}$

$D2 = \{\text{Философия; Математика}\}$

$D3 = \{3; 4; 5\}$

Тогда декартово произведение доменов дает:

$D1 \times D2 \times D3 = \{\text{Иванов, Философия, 3; Иванов, Философия, 4; Иванов, Философия, 5; Иванов, Математика, 3; Иванов, Математика, 4; Иванов, Математика, 5; Петров, Философия, 3; Петров, Философия, 4; Петров, Философия, 5; Петров, Математика, 3; Петров, Математика, 4; Петров, Математика, 5}\}$

Отношение R есть подмножеством этого множества, то есть может содержать любые из его элементов. Все элементы полученного отношения R однотипны, их структура определяется схемой отношения.

Схема отношения - это именованное множество пар:

$\{\text{имя атрибута, имя домена}\} (\{A_i, D_i\})$.

Множество пар $\{\text{имя атрибута, значение}\} (\{A_i, a_i\})$, соответствующих данной схеме отношения образует **кортеж** отношения. То есть кортеж содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения.

Можно дать другое определение отношения. **Отношение** – это множество кортежей, соответствующих одной схеме отношения.

Схема отношения является **заголовком отношения**. Мощность множества (количество) атрибутов отношения определяет «**арность**» (или степень) схемы отношения.

Множество кортежей также называют **телом отношения**. Тело отношения характеризуется **кардинальным числом**, которое равно количеству содержащихся в нем кортежей.

Тело отношения отражает состояние сущности, поэтому во времени оно постоянно меняется. Так если схема отношения описывает атрибуты информационного объекта **Студент**, то тело – множество студентов.

Схема БД (в структурном смысле) представляет собой набор именованных схем отношений.

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

Отношение имеет простую графическую интерпретацию – **таблица** (см. рис. 5).

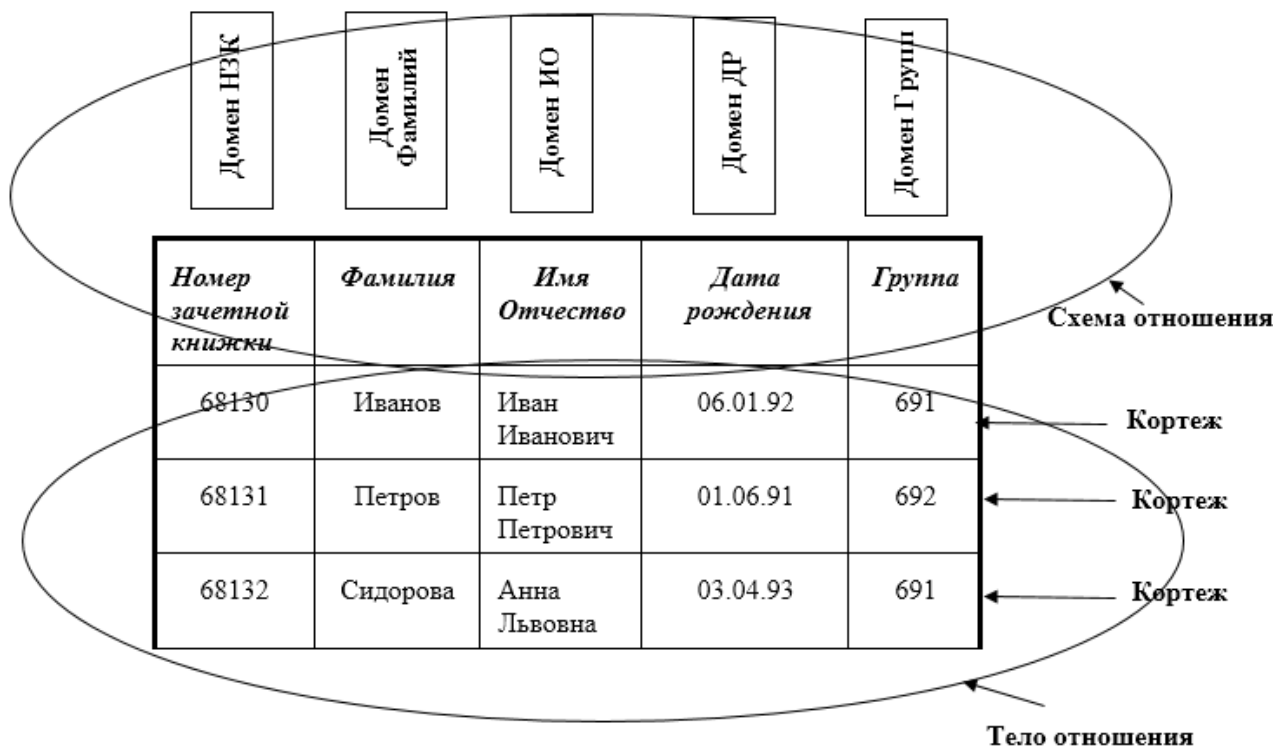


Рис. 5. Отношение *Студент*

Основываясь на приведенных выше определениях сформулируем свойства отношений.

Атомарность значений атрибутов. Любой атрибут должен быть определен на простом типе данных, поэтому значения всех атрибутов являются атомарными (неделимыми).

Отсутствие упорядоченности атрибутов. Атрибуты отношений не упорядочены, поскольку по определению схема отношения есть множество пар {имя атрибута, имя домена}. Для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута.

Отсутствие упорядоченности кортежей. Свойство отсутствия упорядоченности кортежей отношения также является следствием определения отношения-экземпляра как элемента множества кортежей.

Эквивалентность кортежей. Любой кортеж отношения определен на схеме отношения, поэтому кортежи одного отношения эквивалентны.

Отсутствие кортежей-дубликатов. То свойство, что отношения не содержат кортежей-дубликатов, следует из определения отношения как множества кортежей.

Кортежи отношения не имеют идентификаторов, для того чтобы отличить один кортеж от другого используют ключ.

Ключ - это один атрибут или несколько атрибутов, которые однозначно идентифицируют кортеж отношения (имеют уникальные значения в каждом кортеже).

Ключ, содержащий два и более атрибута, называется **составным ключом**.

В отношении может быть несколько одиночных атрибутов или наборов атрибутов, значения которых уникальны, тогда их называют **потенциальными (возможными) ключами**.

Один из потенциальных ключей назначается в качестве главного и его называют **первичный ключ**. Обычно первичным ключом назначается тот возможный ключ, которым проще всего пользоваться при повседневной работе.

Для индикации связи между отношениями используются внешние ключи. **Внешний ключ** — это один или набор атрибутов одного отношения, являющийся потенциальным ключом другого отношения.

Понятия первичный и внешний ключ являются важными при определении ограничений целостности.

В **манипуляционной части модели** утверждаются два механизма манипулирования данными, которые называются: реляционная алгебра и реляционное исчисление.

Язык реляционной алгебры базируется, в основном, на классической теории множеств, а язык реляционного исчисления - на математической логике (классическом логическом аппарате исчисления предикатов первого порядка). Оба языка обладают одним важным свойством: они замкнуты относительно понятия отношения. Это означает, что выражения реляционной алгебры и формулы реляционного исчисления определяются над отношениями реляционных БД и результатом вычисления также являются отношения.

Наконец, в **целостной части реляционной модели данных** фиксируются требования целостности данных, которые должны поддерживаться в любой реляционной СУБД.

Целостность данных – это однозначность и непротиворечивость данных. Целостность данных обеспечивается набором правил, которые также

называют ограничения целостности данных. Определены три категории ограничений целостности.

Целостность доменов. Каждый атрибут принимает лишь допустимые значения, принадлежащие тому домену, в котором атрибут определен. Ограничения целостности доменов обеспечивают проверку значения не только на принадлежность допустимому диапазону или списку допустимых значений, но и на смысл значений.

Целостность сущностей. Состоит в том, что любой кортеж любого отношения должен быть отличим от любого другого кортежа этого же отношения. Это означает, что в отношении не должно быть двух одинаковых кортежей (с одинаковыми значениями атрибутов). Целость сущностей достигается определением первичного ключа.

Целостность по ссылкам (ссылочная целостность). Набор правил, в соответствии с которыми логически связанные между собой кортежи разных отношений физически и однозначно связываются через общие значения атрибутов. Атрибут значения, которого однозначно характеризуют сущности, представленные кортежами некоторого другого отношения, называют **внешним ключом**.

Требование целостности по ссылкам, или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся (дочернем) отношении, в отношении (родительском), на которое ведет ссылка, должен найтись кортеж с таким же значением первичного (или возможного) ключа.

Целостность по ссылкам обеспечивает логическую согласованность первичных и внешних ключей при вставке, обновлении и удалении записей как в дочернем, так и в родительском отношении.

Существуют три подхода, каждый из которых поддерживает целостность по ссылкам. Для примера возьмем операцию удаления записей. Первый подход (Restrict) заключается в том, что запрещается производить удаление кортежа из родительского отношения, если на него существуют ссылки из дочернего отношения. При втором подходе (Set Null) при удалении кортежа из родительского отношения, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным (или заранее определенным). Наконец, третий подход (Cascade) состоит в том, что при удалении кортежа из родительского отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

Целость по ссылкам достигается определением внешнего ключа в дочернем отношении.

1.3. Языки манипулирования данными в реляционной модели

Как уже говорилось выше в реляционной теории утверждается, что доступ к реляционным данным осуществляется при помощи языка реляционной алгебры или эквивалентного ему языка реляционного исчисления. Реляционная алгебра представляет собой набор операторов, использующих отношения в качестве аргументов, и возвращающие отношения в качестве результата. Таким образом, реляционный оператор f выглядит как функция с отношениями в качестве аргументов:

$$R = f(R_1, R_2, \dots, R_n)$$

В качестве аргументов в реляционные операторы можно подставлять другие реляционные операторы, подходящие по типу:

$$R = f(f_1(R_{11}, R_{12}, \dots, R_{1n}), R_2, \dots, R_n)$$

Каждое отношение и исходное, и результирующее обязано иметь имя уникальное в пределах базы данных. Отношения, полученные в результате реляционных выражений, могут не иметь имен, так как время существования этих отношений – время выполнения операции.

Набор основных алгебраических операций состоит из восьми операций, которые делятся на два класса - теоретико-множественные операции и специальные реляционные операции.

К теоретико-множественным операциям относятся:

- объединения отношений;
- пересечения отношений;
- взятия разности отношений;
- прямого произведения отношений.

Специальные реляционные операции включают:

- ограничение отношения;
- проекцию отношения;
- соединение отношений;
- деление отношений.

Некоторые реляционные операторы требуют, чтобы отношения имели одинаковые заголовки (схемы отношения). Такие отношения называют **совместимыми по типу**. Отношения совместимы по типу:

- если имеют одно и то же множество имен атрибутов, т.е. для любого атрибута в одном отношении найдется атрибут с таким же наименованием в другом отношении,
- если атрибуты с одинаковыми именами определены на одних и тех же доменах.

Некоторые отношения не являются совместимыми по типу, так как могут иметь атрибуты с разными именами, определенными на одном и том же домене. Для того чтобы можно было изменить имя введен вспомогательный **оператор переименования атрибутов**

$$\text{rename } A_{old} \text{ as } A_{new}$$

Приведем правила получения отношений в результате выполнения реляционных операторов.

Объединением двух совместимых по типу отношений R_1 и R_2 называется отношение с тем же заголовком, что и у отношений R_1 и R_2 , и телом, состоящим из кортежей, принадлежащих или R_1 , или R_2 , или обоим отношениям.

Пересечением двух совместимых по типу отношений R_1 и R_2 называется отношение с тем же заголовком, что и у отношений R_1 и R_2 , и телом, состоящим из кортежей, принадлежащих одновременно обоим отношениям R_1 и R_2 .

Разностью двух совместимых по типу отношений R_1 и R_2 называется отношение с тем же заголовком, что и у отношений R_1 и R_2 , и телом, состоящим из кортежей, принадлежащих отношению R_1 и не принадлежащих отношению R_2 .

Декартовым произведением двух отношений $R_1(A_1, A_2, \dots, A_n)$ и $R_2(B_1, B_2, \dots, B_m)$ называется отношение, заголовок которого является **сцеплением заголовков** отношений R_1 и R_2 :

$$(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m),$$

а тело состоит из кортежей, являющихся **сцеплением кортежей** отношений R_1 и R_2 :

$$(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m),$$

таких, что атрибуты (a_1, a_2, \dots, a_n) принадлежат отношению R_1 , а атрибуты (b_1, b_2, \dots, b_m) принадлежат отношению R_2 .

Выборкой (ограничением) на отношении R с условием s называется отношение с тем же заголовком, что и у отношения R , и телом, состоящем из кортежей, значения атрибутов которых при подстановке в условие s дают значение ИСТИНА. s - логическое выражение, в которое могут входить атрибуты отношения R и (или) скалярные выражения.

Проекцией отношения R по атрибутам X, Y, Z , где каждый из атрибутов принадлежит отношению R , называется отношение с заголовком (X, Y, \dots, Z) и телом, содержащим множество кортежей вида (x, y, \dots, z) , таких, для которых в отношении R найдутся кортежи со значением атрибута X равным x , значением атрибута Y равным y , ..., значением атрибута Z равным z .

Соединением отношений R_1 и R_2 по условию s называется отношение, полученное выборкой из результата декартового произведения отношений R_1 и R_2 :

$(R_1 \text{ TIMES } R_2) \text{ WHERE } c$

c представляет собой логическое выражение, в которое могут входить атрибуты отношений R_1 и R_2 и (или) скалярные выражения.

Частным случаем общего соединения является **экви-соединение** (или внутреннее соединение) при выполнении которого условием выборки кортежей отношения полученного декартовым произведением исходных отношений является равенство атрибутов исходных отношений определенных на одном и том же домене:

$(R_1 \text{ TIMES } R_2) \text{ WHERE } A=B,$

где атрибут A принадлежит отношению R_1 , а атрибут B - R_2

Операция деления отношений

Пусть даны отношения $R_1(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$ и $R_2(B_1, B_2, \dots, B_m)$, причем атрибуты B_1, B_2, \dots, B_m - общие для двух отношений. **Делением отношений** R_1 на R_2 называется отношение с заголовком (A_1, A_2, \dots, A_n) и телом, содержащим множество кортежей (a_1, a_2, \dots, a_n) , таких, что для *всех* кортежей (b_1, b_2, \dots, b_m) принадлежащих R_2 в отношении R_1 найдется кортеж $(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m)$.

Отношение R_1 выступает в роли **делимого**, отношение R_2 выступает в роли **делителя**. Деление отношений выполняется аналогично делению чисел с остатком.

Реляционное исчисление является прикладной ветвью формального механизма исчисления предикатов первого порядка.

Исчисления выражают только то, каким должен быть результат вычисления, а не то, каким образом проводить вычисления. В основе исчисления лежит понятие **переменной** с определенной для нее областью допустимых значений и понятие **правильно построенной формулы**, опирающейся на переменные, предикаты и кванторы.

В зависимости от того, что является областью определения переменной, различают:

- исчисление кортежей,
- исчисление доменов.

В исчислении кортежей областями определения переменных являются тела отношений базы данных, т. е. допустимым значением каждой переменной является кортеж тела некоторого отношения.

В исчислении доменов областями определения переменных являются домены, на которых определены атрибуты отношений базы данных, т. е. допустимым значением каждой переменной является значение некоторого домена.

Правильно построенная формула языка логики предикатов соответствуют предложениям естественного языка, выражающим суждения, или же соответствуют словосочетаниям естественного языка, выражающим предикаты.

2. СИСТЕМА УПРАВЛЕНИЯ РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ

2.1. Общие принципы построения СУБД

Языково-программным средством управления БД является система управления базами данных (СУБД).

Системой управления базами данных называют программную систему, предназначенную для создания базы данных на ЭВМ, поддержания этой БД в актуальном состоянии и обеспечения доступа пользователей к содержащимся в ней данным.

По степени универсальности различают два больших класса СУБД:

- системы общего назначения;
- специализированные системы.

СУБД общего назначения не ориентированы на какую-либо предметную область или информационные потребности какой-либо группы пользователей. Каждая система такого класса реализуется как программное средство, способное функционировать на некоторой модели ЭВМ в определенной операционной системе и поставляемое как программный продукт. Использование СУБД общего назначения в качестве инструментального средства для создания информационных систем, основанных на технологии автоматизированной обработки данных, существенно сокращает сроки разработки и экономит трудовые ресурсы. Этим СУБД присущи развитые функциональные возможности по организации, хранению и управлению данными.

Специализированные СУБД создаются в редких случаях при невозможности или нецелесообразности использования СУБД общего назначения (как, например, в военном деле).

СУБД общего назначения – это сложные программные комплексы, предназначенные для выполнения всей совокупности функций по созданию и эксплуатации БД информационных систем.

К числу основных функций СУБД принято относить следующие:

- управление данными во внешней памяти;
- управление буферами оперативной памяти;
- управление транзакциями;
- журнализация и восстановление БД после сбоев;
- поддержание языков БД.

Управление данными во внешней памяти. Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для ускорения доступа к данным в некоторых случаях (обычно для

этого используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. Однако при работе в развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то, как организованы файлы.

Управление буферами оперативной памяти. СУБД обычно работают с БД значительного размера; по крайней мере этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. Управление буферами состоит из двух механизмов:

- диспетчер буферов для доступа и обновления страниц базы данных;
- буферный кэш (известный как буферный пул) для сокращения операций ввода-вывода файла базы данных. Буферный кэш делится на 8-килобайтовые страницы, размер которых соответствует размеру страниц данных.

Диспетчер буферов управляет обменом данных с внешней памятью:

- чтением страниц данных из файлов базы данных на диске в буферный кэш;
- записью измененной страницы данных обратно на диск.

Если какая-либо из страниц в буферном кэше изменилась, она не записывается сразу на диск, а помечается как грязная. Страница остается в буферном кэше до тех пор, пока диспетчеру буферов не понадобится область буфера для считывания дополнительных данных. Данные записываются обратно на диск, только если они были изменены. Данные в буферном кэше могут измениться несколько раз, прежде чем будут сохранены обратно на диске.

Управление транзакциями

Транзакция - последовательность операций, производимых над базой данных и переводящих базу данных из одного согласованного состояния в другое согласованное состояние. Несмотря на то, что транзакция может представлять собой последовательность операций над БД, она рассматривается как единое целое. ***Поэтому транзакция - это логическая единица работы системы.***

Чтобы называться транзакцией логическая единица работы должна обладать четырьмя свойствами атомарностью, согласованностью, изоляцией и длительностью.

Свойство атомарности (Atomicity) выражается в том, что транзакция должна быть выполнена в целом или не выполнена вовсе.

Свойство согласованности (Consistency) гарантирует, что по мере выполнения транзакций данные переходят из одного согласованного состояния в другое (при котором применены все правила для обеспечения целостности всех данных).

Свойство изолированности (Isolation) означает, что конкурирующие за доступ к базе данных транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.

Свойство долговечности (Durability) трактуется следующим образом: если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах (даже в случае последующих ошибок).

Транзакцию обладающие указанными свойствами называют ACID транзакцией.

Если все операции транзакции выполнены успешно и в процессе выполнения транзакции не произошло никаких сбоев программного или аппаратного обеспечения, то транзакция фиксируется.

Фиксация транзакции - это действие, обеспечивающее запись на диск изменений в БД, которые были сделаны в процессе выполнения транзакций. Фиксация транзакций означает, что все результаты ее выполнения становятся постоянными, и станут видимыми другим транзакциям.

Если в процессе выполнения транзакций случилось нечто такое, что делает невозможным ее нормальное завершение, БД должна быть возвращена в исходное состояние.

Откат транзакции - это действие, обеспечивающее аннулирование всех изменений данных, которые были сделаны операторами SQL в теле текущей незавершенной транзакции.

Понятие транзакции необходимо для поддержания логической целостности БД. Поддержание механизма транзакций является обязательным условием в многопользовательских СУБД. Управление параллельно выполняющихся транзакций со стороны СУБД должно быть организовано таким образом, чтобы каждый из пользователей мог в принципе ощущать себя единственным пользователем СУБД.

С управлением транзакциями в многопользовательской СУБД связаны важные понятия *сериализации транзакций* и *сериальный план выполнения смеси транзакций*.

Под **сериализацией параллельно выполняющихся транзакций** понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения.

Сериальный план выполнения смеси транзакций - это такой план, который приводит к сериализации транзакций. Понятно, что если удастся добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы по сравнению с однопользовательским режимом).

Наиболее распространенным механизмом, который используется СУБД для реализации на практике сериализации транзакций является **механизм блокировок**.

С помощью блокировок ядро БД синхронизирует одновременный доступ нескольких пользователей к одному фрагменту данных. Прежде чем транзакция сможет распоряжаться текущим состоянием фрагмента данных, например, для чтения или изменения данных, она запрашивает блокировку фрагмента данных. Блокировка имеет несколько аспектов:

- длительность блокировки;
- гранулярность блокировки;
- режим блокировки.

Длительность блокировки определяет период времени, в течение которого заблокированы ресурсы. Длительность блокировки зависит от режима блокировки и выбранного уровня изоляции.

Гранулярность блокировки указывает какой ресурс блокируется. Блокировки могут быть установлены на строку, таблицу, страницу, экстенд или на всю базу данных.

В СУБД поддерживаются следующие основные режимы блокировок.

Совмещаемая (разделяемая) блокировка (S-Shared) используется для операций считывания, которые не меняют и не обновляют данные, такие как инструкция SELECT. Пока для ресурса существуют совмещаемые (S) блокировки, другие транзакции не могут изменять данные.

Блокировка обновления (U - Update) используется для операции обновления, которая требует предварительного чтения данных. Другие транзакции могут считывать данные в то время, когда эти данные считываются для инструкции UPDATE, но другие инструкции UPDATE будут ждать, пока не будет освобождена блокировка обновлений.

Монопольная блокировка (X-Exclusive) используется для операций модификации данных, таких как инструкции INSERT, UPDATE или DELETE. Монопольные блокировки предотвращают доступ к данным со стороны других транзакций.

Все блокировки, удерживаемые транзакцией, освобождаются после ее завершения (при фиксации или откате).

Управления параллельно выполняющимися транзакциями осуществляется посредством выбора **уровней изоляции транзакций** для соединений. Благодаря различным уровням изоляции СУБД дает разработчикам возможность определить для каждой отдельной транзакции строгость изоляции от другой транзакции. Уровни изоляции транзакций определяют:

- будут ли блокировки использоваться при чтении данных;
- как долго удерживаются блокировки;
- какие типы блокировок используются для чтения данных;
- что произойдет, если операции чтения потребуется считать данные,

на которые распространяется монополярная блокировка другой транзакции:

- ✓ подождать до снятия блокировки другой транзакцией;
- ✓ прочесть незафиксированные данные;
- ✓ прочесть последнюю зафиксированную версию данных.

Журнализация и восстановление БД после сбоев

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под **надежностью хранения** понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя.

Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризующиеся потерей информации на носителях внешней памяти.

Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия незавершенной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Наиболее распространенным методом поддержания такой информации является ведение журнала транзакций БД.

Журнал транзакций - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью, в которую поступают записи обо всех изменениях основной части БД.

При ведении журнала придерживаются стратегии «упреждающей» записи в журнал (так называемого протокола *Write Ahead Log* - WAL). Эта стратегия заключается в том, что запись об изменении любого объекта БД должна

попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол *WAL*, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Самая простая ситуация восстановления - индивидуальный откат транзакции. Все записи в журнале имеющие один и тот же идентификатор транзакции связываются в обратный список. Началом списка для не закончившихся транзакций является запись о последнем изменении базы данных, произведенном данной транзакцией. Концом списка всегда служит первая запись об изменении базы данных, произведенном данной транзакцией. Для отката транзакции достаточно выполнить обратные операции, следуя от конца списка записей журнала к его началу.

В этом случае мягкого сбоя возможна потеря той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти, но не была зафиксирована во внешней памяти. Страницы буферного кэша могут содержать изменения, выполненные транзакцией, которая либо уже успешно завершена, либо нет. В этом случае восстановление после мягкого сбоя с использованием журнала будет выполняется следующим образом:

- для всех завершенных транзакций выполняется накат - повторное выполнение операции завершенных транзакций, результаты которых не отражены во внешней памяти;
- для всех незавершенных транзакций - выполняется откат.

Восстановление БД после жесткого сбоя возможно только при наличии полной резервной копии БД, созданной на определенный момент времени и резервных копий журнала транзакций. К сохранности резервных копий базы данных и копий журнала во внешней памяти в СУБД предъявляются особо повышенные требования. Никогда не следует выполнять резервное копирование на дисковое устройство, которое размещается на том же физическом устройстве хранения, что и сама база данных.

Поддержка языков БД

Для работы с базами данных используются специальные языки, в целом называемые *языками баз данных*. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - *язык определения схемы БД (SDL - Schema Definition Language)* и *язык манипулирования данными (DML - Data Manipulation Language)*. *SDL* служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. *DML* содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык *SQL*. Он содержит все необходимые механизмы для ведения реляционных баз данных.

Также поддерживается и язык структуры данных XML (Extensible Markup Language) – это стандарт, используемый в Интернете для обмена данными между различными системами с использованием самоопределяемых наборов данных. XML представляет собой набор меток (тэгов), которые служат для представления логической структуры таблицы, как XML-документа.

2.2. Архитектура СУБД

Логически в современной реляционной СУБД можно выделить:

- ядро СУБД (часто его называют *DataBase Engine*);
- компилятор языка БД (обычно *SQL*);
- подсистему поддержки времени выполнения;
- набор утилит.

В некоторых системах эти части выделяются явно, в других - нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно, можно выделить такие компоненты ядра (по крайней мере, логически, хотя в некоторых системах эти компоненты выделяются явно), как менеджер данных, менеджер буферов, менеджер транзакций и менеджер журнала. Функции этих компонентов взаимосвязаны, и для обеспечения корректной работы СУБД все эти компоненты должны взаимодействовать по тщательно продуманным и проверенным протоколам.

Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором *SQL* (или в подсистеме поддержки выполнения таких программ) и утилитах БД. Ядро СУБД является основной резидентной частью СУБД. При использовании архитектуры «клиент-сервер» ядро является основной составляющей серверной части системы.

Компилятор языка БД. Основной функцией компилятора языка БД является компиляция операторов языка БД в некоторую выполняемую программу. Основной проблемой реляционных СУБД является то, что языки

этих систем (а это, как правило, *SQL*) являются непроцедурными, т.е. в операторе такого языка специфицируется некоторое действие над БД, но эта спецификация не является процедурой, а лишь описывает в некоторой форме условия совершения желаемого действия. Поэтому компилятор должен решить, каким образом выполнять оператор языка прежде, чем произвести программу. Существуют достаточно сложные методы оптимизации операторов. Однако результатом компиляции всегда является выполняемая программа, представляемая в некоторых системах в машинных кодах, но более часто в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением **подсистемы поддержки времени выполнения**, которая представляет собой, по сути дела, интерпретатор этого внутреннего языка.

Наконец, в отдельные утилиты БД обычно выделяют такие процедуры, которые не выполняются с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

Как уже было сказано выше ядро СУБД является основной резидентной частью СУБД и представляет собой основную компоненту обеспечивающую хранение, обработки и безопасность данных.

Местоположение ядра СУБД в значительной степени влияет на процесс разработки приложений, обрабатывающего содержащиеся в этой базе данные. В зависимости от относительного расположения приложения и БД различают:

- локальные БД;
- удаленные БД.

Локальные БД располагаются на том же компьютере, что и использующие их приложения. В этом случае информационная система имеет локальную архитектуру (рис.6).

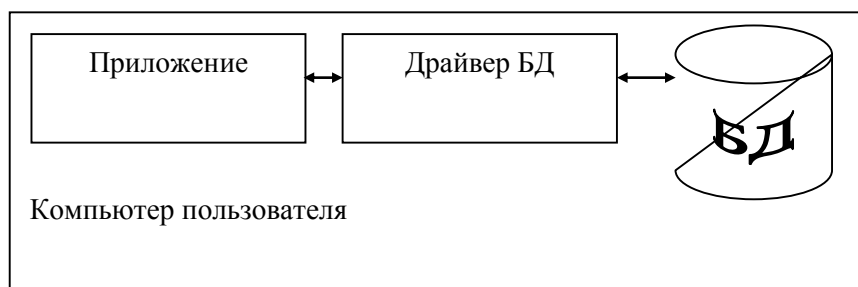


Рис. 6. Локальная архитектура информационной системы

Работа с локальной БД происходит, как правило, в однопользовательском режиме.

При использовании БД в сети возможна организация многопользовательского доступа к ней. В этом случае файлы БД и приложение, предназначенное для работы с БД, размещаются на сервере сети. Каждый пользователь запускает со своего компьютера приложение, установленное на сервере, при этом на его собственном компьютере загружается копия приложения. Такой сетевой вариант применения БД соответствует архитектуре «файл-сервер» (рис.7). Приложение в архитектуре «файл-сервер» также может быть установлено и на каждый компьютер сети. В данной реализации приложению должно быть известно местонахождение общей БД.

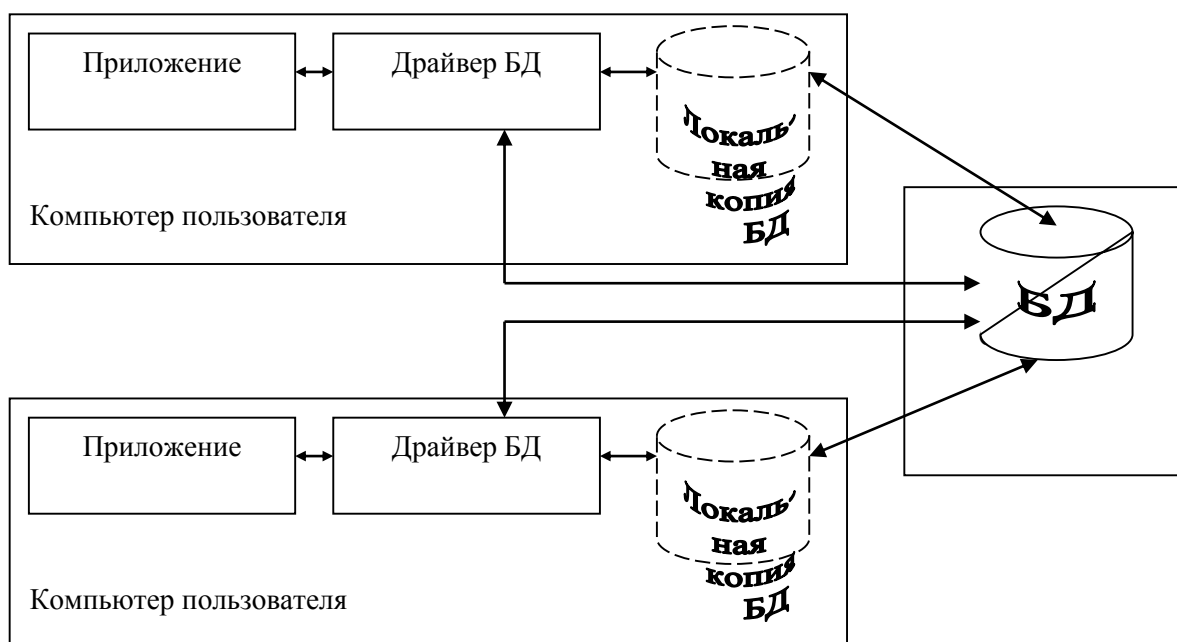


Рис. 7. Файл-серверная архитектура информационной системы

При работе с данными на каждом пользовательском компьютере сети используется локальная копия БД. Эта копия периодически обновляется данными, содержащимися в БД на сервере.

Архитектура «файл-сервер» обычно применяется в сетях с небольшим количеством пользователей. Достоинством данной архитектуры являются простота реализации, а также то, что приложение фактически разрабатывается в расчете на одного пользователя и не зависит от компьютера сети, на который оно устанавливается.

Архитектуре «файл-сервер» присущие следующие недостатки.

Пользователь работает со своей локальной копией БД, информация в которой обновляется при каждом запросе к какой-либо из таблиц. Если запрашивается несколько записей, с сервера по сети пересылается вся таблица. В результате циркуляции больших объемов избыточного трафика

резко возрастает нагрузка на сеть, что приводит к снижению ее быстродействия и производительности информационной системы в целом.

В связи с тем, что на каждом компьютере пользователя сети имеется своя локальная копия БД, изменения, внесенные в нее одним пользователем, в течение некоторого времени являются неизвестными другим пользователям. Вследствие этого необходимо постоянно обновлять БД. Кроме того, возникает необходимость синхронизации работы отдельных пользователей, связанная с блокировкой записей таблицы, которые редактирует другой пользователь.

Управление БД в «файл-серверной» архитектуре осуществляется с разных компьютеров, поэтому в значительной степени затруднена организация контроля доступа, соблюдения конфиденциальности и поддержания целостности БД.

Архитектуре «клиент-сервер», в которой информационная система делится на неоднородные части – сервер и клиент БД. Удаленная БД размещается на компьютере-сервере сети, а приложение, обрабатывающее данные этой БД, находится на компьютере пользователя (рис. 8).

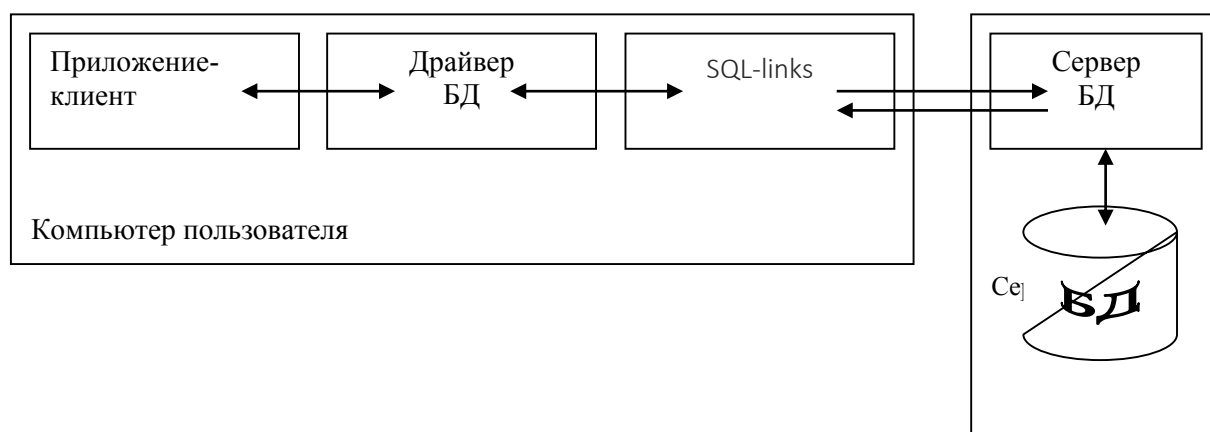


Рис. 8. Двухуровневая клиент-серверная архитектура информационной системы («толстый» клиент)

В связи с тем, что компьютер-сервер расположен отдельно от клиента, его также называют **удаленным сервером**. Существенно, что сервером называется, как специальная программа, которая управляет БД, так и компьютер, на котором установлены эти БД и программа. Поскольку в основе организации запросов к БД лежит язык SQL, то программу, управляющей БД, называют **SQL-сервером**, а БД – **базой данных SQL**.

Клиентом является приложение пользователя, которое также называют **приложением-клиентом**. Для обращения к данным приложение-клиент формулирует и отправляет запрос удаленному серверу, содержащему БД. После получения запроса удаленный сервер направляет его **SQL-серверу**.

(серверу баз данных). *SQL*-сервер обеспечивает выполнение запроса и выдачу клиенту его результата – требуемых данных. Вся обработка запроса происходит на удаленном сервере.

В роли *SQL*-сервера выступают СУБД, такие как *Oracle*, *MySQL*, *MS SQL Server* и др.

Данная архитектура позволила получить следующие преимущества:

- снижение нагрузки на сеть;
- повышение уровня защищенности информации, обусловленное тем, что обработка запросов всех клиентов осуществляется единой программой, расположенной на сервере;
- снижение сложности клиентских приложений, в которых отсутствует код, связанный с контролем БД и разграничением доступа к ней.

Рассмотренная архитектура является двухуровневой, в которой уровни соотносятся с приложением-клиентом и сервером БД. В этой архитектуре клиентское приложение называют сильным или «толстым» клиентом. Развитие «клиент-серверных» технологий привело к появлению *трехуровневой архитектуры* (рис. 9), в которой помимо сервера БД и приложений-клиентов дополнительно присутствует сервер приложений.

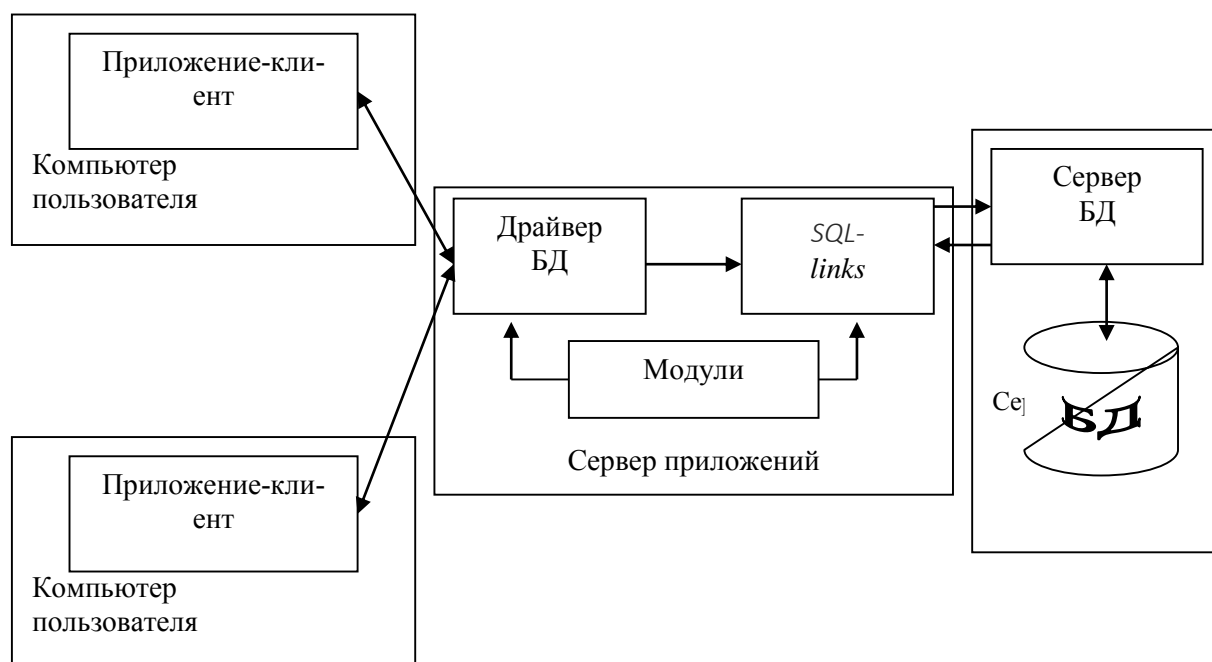


Рис. 9. Трехуровневая клиент-серверная архитектура информационной системы («тонкий» клиент)

В трехуровневой архитектуре появляется третий участник - **сервер приложений**, который являясь промежуточным уровнем обеспечивает организацию взаимодействия клиентов («тонких» клиентов) и сервера БД.

К функциям сервера приложений могут, например, относиться: выполнение соединения с сервером, разграничение доступа к данным, реализация бизнес-правил и т.д. Кроме того, сервер приложений может организовывать взаимодействие с клиентами, установленными на различных аппаратных платформах, функционирующих на компьютерах различного типа под управлением разных информационных систем. Сервер приложений называют также *брокером данных*.

Основные достоинства трехуровневой архитектуры «клиент-сервер» состоят в следующем:

- разгрузка сервера БД от выполнения части операций, перенесенных на сервер приложений;
- уменьшение размера клиентских приложений за счет сокращения объема лишнего кода;
- единая линия поведения всех клиентов;
- упрощение настройки клиентов – при изменении общего кода сервера приложений автоматически меняется поведение приложений клиентов.

2.3. Компоненты СУБД Microsoft SQL Server

Программный комплекс СУБД SQL Server представляет собой комплекс служб, графических клиентских приложений и утилит. В составе СУБД SQL Server имеется несколько различных видов компонентов:

- Серверные компоненты. Они составляют основу SQL Server и работают как службы операционной системы.
- Графические клиентские приложения и утилиты командной строки для администрирования сервера. Они, как и другие клиентские приложения, используют средства обмена данными между клиентом и сервером, предоставляемые SQL Server 2000.
- Компоненты SQL Server 2000, обеспечивающие обмен данными между клиентом и сервером, предоставляют клиентским приложениям множество способов доступа к данным сервера. Эти компоненты реализованы в виде провайдеров, драйверов, интерфейсов баз данных и сетевых библиотек Net Libraries.

Серверные компоненты

Database Engine (ядро СУБД) представляет собой основную службу для хранения, обработки и обеспечения безопасности данных. Используется для создания реляционных баз данных с целью оперативной обработки транзакций или оперативной аналитической обработки данных. Выполняет создание таблиц для хранения данных и объектов баз данных, таких как индексы, представления и хранимые процедуры для просмотра и защиты данных и позволяет управлять ими.

Реляционное ядро БД обеспечивает достоверность и защиту хранимых данных, отказоустойчивость, динамически оптимизирует производительность, а также налагает блокировки для реализации параллелизма.

Службы Analysis Services представляют собой решение для построения и развертывания аналитических баз данных, используемых для поддержки принятия решений в приложениях бизнес-аналитики. Базой любого решения Analysis Services являются семантическая модель данных бизнес-аналитики и экземпляр сервера, на котором создаются, обрабатываются, отправляются запросы и производится управление объектами в этой модели. Модели создаются на данных, которые накапливаются в транзакционных базах данных. После создания модель развертывается на сервере служб Analysis Services в качестве базы данных и становится доступной авторизованным пользователям, которые соединяются с ней через клиентские приложения или другие средства. Службы Analysis Services позволяют создавать многомерные и табличные модели данных.

Многомерная модель состоит из кубов и измерений, которые могут быть аннотированы и расширены для поддержки сложных конструкций запросов. Такое представление итоговых данных, полученных по разным измерениям, позволяет в несколько раз ускорить время выполнения запросов, по сравнению с выполнением аналогичных запросов в транзакционных базах данных.

Табличные модели являются базами данных в памяти служб Analysis Services. С помощью современных алгоритмов сжатия и многопоточного обработчика запросов подсистема аналитики в памяти xVelocity (VertiPaq) обеспечивает быстрый доступ к объектам табличной модели и данным. Подход табличного моделирования более понятен бизнес-аналитику, привыкшему работать с реляционными данными. Табличное моделирование отличается от реляционного, поскольку в нем построение семантической модели бизнес-аналитики выполняется на основе таблиц и связей, а не кубов и измерений.

Службы SQL Server Integration Services представляют собой платформу для извлечения данных из нескольких источников, их преобразования, их объединения, перемещения в один или несколько целевых источников данных. Службы Integration Services могут извлекать и преобразовывать данные из ряда таких источников, как файлы XML-данных, неструктурированные файлы и источники реляционных данных, и затем загружать эти данные в один или несколько реляционных объектов, хранилищ данных.

В состав служб Службы Integration Services входит широкий набор встроенных задач и преобразований, средства для построения пакетов, а также служба Службы Integration Services для выполнения пакетов и управления ими.

Службы Reporting Services представляют собой средства для создания корпоративных отчетов с поддержкой веб-интерфейса, которые могут представлять данные из различных источников. С помощью служб Reporting Services можно создать интерактивные, табличные, графические отчеты и отчеты свободной формы из реляционных, многомерных и XML-источников данных. Отчеты могут включать визуализацию сложных данных, в том числе диаграммы, карты и спарклайны. Можно публиковать отчеты, планировать их обработку, осуществлять доступ к отчетам по требованию и экспортировать отчеты в другие приложения.

Клиентские средства администрирования

Утилиты и средства администрирования SQL Server реализованы в виде клиентских приложений, которые устанавливают соединение с SQL Server по локальной сети, используя клиентские компоненты обмена данными. Дадим характеристику некоторым из них.

SQL Server Management Studio — утилита Microsoft SQL Server для конфигурирования, управления и администрирования всех компонентов Microsoft SQL Server.

SQL Server Management Studio — представляет собой интегрированную среду для доступа, управления, настройки, администрирования и разработки всех компонентов SQL Server. Включает графические средства управления, которые позволяют работать с объектами и настройками сервера. Кроме Database Engine среда SQL Server Management Studio позволяет работать со всеми компонентами SQL Server, например, со службами Reporting Services и Integration Services. Компонент среды редактор кода содержит встроенные редакторы скриптов для пользовательской разработки скриптов Transact-SQL, многомерных выражений, расширений интеллектуального анализа данных и XML для аналитики.

Диспетчер конфигурации SQL Server. Эта утилита администрирования предназначена для управления службами SQL Server. С ее помощью можно запускать, останавливать, приостанавливать, перезапускать и настраивать конфигурацию служб SQL Server.

SQL Server Profiler — графический пользовательский интерфейс для трассировки SQL, с помощью которого можно наблюдать за экземпляром компонента Database Engine или службами Analysis Services. Контролируемые события сохраняются в файле трассировки, который затем может быть проанализирован или использован для воспроизведения определенных последовательностей шагов для выявления возникших проблем. Приложение позволяет:

- создать трассировку на основе шаблона;
- наблюдать результаты трассировки во время ее работы;
- сохранять результаты трассировки в таблицу;

- запускать, останавливать, приостанавливать и изменять результаты трассировки по мере необходимости;
- воспроизводить результаты трассировки.

Клиентские компоненты, обеспечивающие обмен данными между клиентом и сервером.

Пользователи обращаются к данным SQL сервера при помощи клиентских приложений. SQL Server поддерживает два основных вида клиентских приложений. Во-первых, это приложения для работы с реляционными БД — наиболее распространенный тип клиентских приложений в двухуровневой или трехуровневой клиент-серверной среде. Они передают серверу БД запрос на выполнение процедуры и получают результаты их выполнения в виде реляционных наборов данных. Во-вторых, это интернет-приложения, они передают серверу СУБД операторы Transact-SQL или запросы XPath, и получают обратно документы в формате XML.

Клиентские приложения для работы с реляционными БД обращаются к SQL серверу при помощи интерфейса прикладного программирования БД, который определяет на уровне кода приложения, каким образом это приложение будет подключаться к экземпляру SQL сервера и передавать команды в базы данных. В SQL Server поддерживаются встроенные интерфейсы прикладного программирования БД — OLE DB и ODBC.

OLE DB — это интерфейс прикладного программирования, позволяющий приложениям, использующим технологию COM, использовать данные из источников данных OLE DB. В SQL Server имеется встроенный OLE DB-провайдер, который представляет собой COM-компонент получающий вызовы, адресованные интерфейсу прикладного программирования OLE DB, и выполняет все необходимые действия по обработке запроса к источнику данных. Этот провайдер поддерживает приложения, написанные с использованием технологии OLE DB или других интерфейсов прикладного программирования, использующих OLE DB, например, ADO (ActiveX Data Objects — «объекты данных ActiveX»).

ODBC представляет собой интерфейс уровня вызовов (Call-Level Interface, CLI), позволяющий приложениям осуществлять доступ к данным из источников данных ODBC. В SQL Server имеется встроенный ODBC-драйвер — DLL-библиотека, принимающая вызовы, адресованные функциям интерфейса прикладного программирования ODBC API и выполняющая все необходимые действия по обработке запроса к источнику данных. Этот драйвер поддерживает приложения и компоненты, написанные с использованием ODBC или других интерфейсов прикладного программирования, использующих ODBC, например, Data Access Objects (DAO).

Сетевые библиотеки. OLE DB-провайдер и ODBC-драйвер используют клиентскую сетевую библиотеку (Net-Library) для обмена данными с серверной сетевой библиотекой из состава SQL Server. Сетевые библиотеки инкапсулируют запросы, которыми обмениваются клиентские компьютеры и серверы, для последующей передачи этих запросов в нижележащий сетевой протокол. Клиент и SQL сервер можно сконфигурировать так, чтобы использовалась любая из сетевых библиотек:

TCP/IP Sockets - используется для соединения с SQL сервером по протоколу TCP/IP.

Named pipes - используется для соединения с SQL сервером по именованным каналам. Канал представляет собой механизм файловой системы, обеспечивающий взаимодействие процессов.

Обмен данными может осуществляться с шифрованием по протоколу Secure Sockets Layer (SSL).

Серверные сетевые библиотеки взаимодействуют с уровнем реляционной БД, называемым Open Data Services. Open Data Services — это интерфейс между ядром БД и серверными сетевыми библиотеками Net-Libraries. На уровне Open Data Services выполняется преобразование пакетов, полученных от серверных сетевых библиотек, в события, которые затем отправляются ядру БД. Ядро БД использует Open Data Services для отправки клиентам ответов SQL сервера через серверные библиотеки Net-Libraries.

2.4. Архитектура базы данных SQL Server

Архитектура базы данных на логическом уровне

Логическая архитектура базы данных SQL сервера представляет собой объектную модель, на верхнем уровне которой находится ядро базы данных (компонент Database Engine). Database Engine отвечает за хранение, обработку поступающих запросов и передачу соответствующих результатов клиентским компьютерам.

После установки СУБД создается экземпляр SQL сервера. Он представляет собой копию исполняемого файла sqlservr.exe, который работает как служба операционной системы. Каждый экземпляр сервера имеет уникальное имя, как правило, совпадающее с именем компьютера, на котором он исполняется. Однако на одном компьютере можно установить несколько экземпляров SQL сервера, в этом случае каждый из них должен иметь уникальное имя.

Экземпляр сервера управляет системными и пользовательскими базами данных. Системными являются следующие базы данных:

- **master** - хранит информацию о конфигурации сервера, сведения обо всех учетных записях пользователя, сведения обо всех остальных базах данных;

- **model** - является шаблоном для вновь создаваемой базы данных и для базы данных tempdb. База данных model содержит системные таблицы, которые используются каждой базой данных и контролирует параметры базы данных, включая параметры, устанавливаемые по умолчанию, полномочия пользователей и правила базы данных;

- **resource** - содержит все системные объекты SQL сервера. В каждой базе данных можно через схему sys отобразить эти объекты.

- **msdb** – содержит для агента SQL Server информацию о планировании заданий и событий, и информацию об организации работы операторов;

- **tempdb** - база данных для хранения временных объектов пользователей, системы и промежуточных результатов выполнения запросов. Она является общим (глобальным) ресурсом, доступным всем пользователям. Эта база данных создается заново каждый раз при запуске SQL Server. При отключении пользователя от SQL Server все его временные таблицы в tempdb автоматически уничтожаются.

Каждый экземпляр сервера позволяет управлять множеством пользовательских баз данных. Каждая база данных, с данными которой можно работать в режиме записи, должна иметь ассоциированный с ней журнал транзакций.

База данных состоит из разных объектов, таких как таблицы, индексы, ограничения, представления, хранимые представления.

Таблица (table) – объект, предназначенный для хранения данных. В SQL Server имеются таблицы двух основных типов: системные и пользовательские.

Системные таблицы содержат информацию об SQL сервере и его объектах. Данные системных таблиц доступны через представления схемы sys. Например, таблица syslogins содержит информацию обо всех учетных записях сервера и данные этой таблицы можно получить запросом к системному представлению sys.server_principals.

Таблица представляет собой совокупность связанных столбцов (полей). Каждый столбец (поле) таблицы имеет имя, тип данных и размер.

SQL Server имеет набор встроенных типов данных, называемых системными. На основе некоторых системных типов данных пользователи могут создавать собственные типы, так называемые пользовательские типы данных (user – defined data type).

Индексы (indexes) используются для уменьшения времени выполнения операций поиска и выборки данных. Индексы создаются на основе од-

ного или нескольких столбцов таблицы. Индекс содержит отсортированные значения индексируемого столбца (столбцов) со ссылкой на место физического разделения каждой записи. Индекс представляет собой отдельный объект базы данных.

Индексы Database Engine основаны на структуре данных в виде В-дерева (сбалансированного двоичного дерева).

В-дерево имеет древовидную структуру, в которой все самые нижние узлы (узлы- листья) имеют одинаковый номер уровня по отношению к верхнему(корневому) узлу дерева. При добавлении или удалении новых данных это свойство поддерживается.

Построение индекса осуществляется по следующему правилу. Рассмотрим его на примере. Пусть имеется таблица, записи которой содержат столбец *Номер студента*, по значениям которого требуется построить индекс.

Построение индекса осуществляется с нижнего уровня. Последовательность записей, соответствующая записям исходной таблицы, упорядочивается по значениям первичного ключа. Логические записи объединяются в блоки (по k записей в блоках). Значением ключа блока является минимальное значение ключа у записей, входящих в блок.

Пусть в таблице ключ Номер студента имеет значения 2, 7, 8, 12, 15, 27, 28, 40, 43, 50. Для определенности возьмем $k=2$ (в блок объединяем по 2 экземпляра записей). Тогда на нижнем уровне будет определено 4 блока (рис. 10). Так как блоков больше одного строится индекс предыдущего уровня. Записи этого уровня содержат значение ключа блока следующего уровня и указатель-адрес связи соответствующего блока; записи этого уровня также объединяются в блоки (по k записей). Затем аналогично строится индекс более высокого уровня и т.д., пока количество записей индекса на определенном уровне будет не более k .

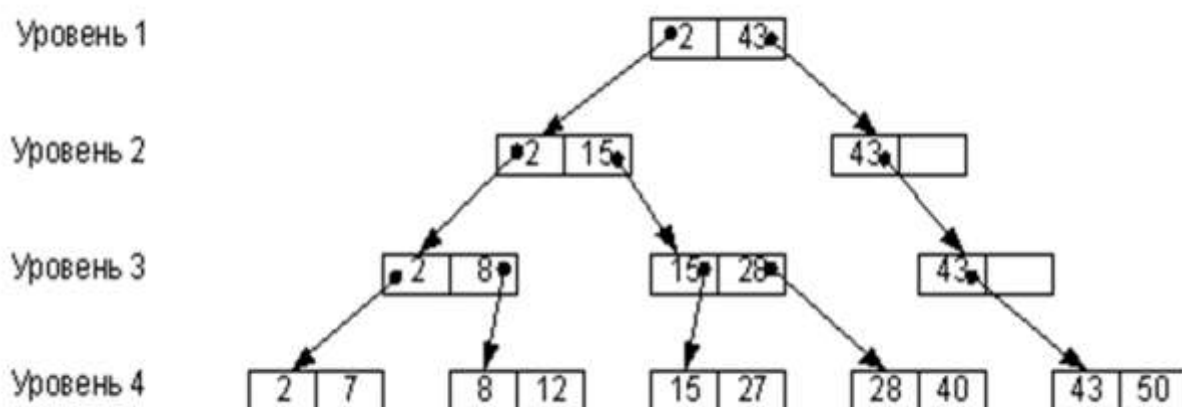


Рис.10. Структура В-дерева

Индекс, имеющий структуру В-дерева позволяет быстро осуществлять поиск нужных строк в таблице. Поиск выполняется по следующему алгоритму. Читается верхний индекс и сравнивается заданное значение ключа со значением ключа последней записи индекса. Если заданное значение ключа больше или равно значению ключа очередной записи индекса (если такая запись имеется), то по адресу связи, указанному в текущей записи, читается блок записей индекса следующего уровня. Далее процесс повторяется. На рисунке 11 показано как будет осуществляться поиск ключа со значением равным 12.

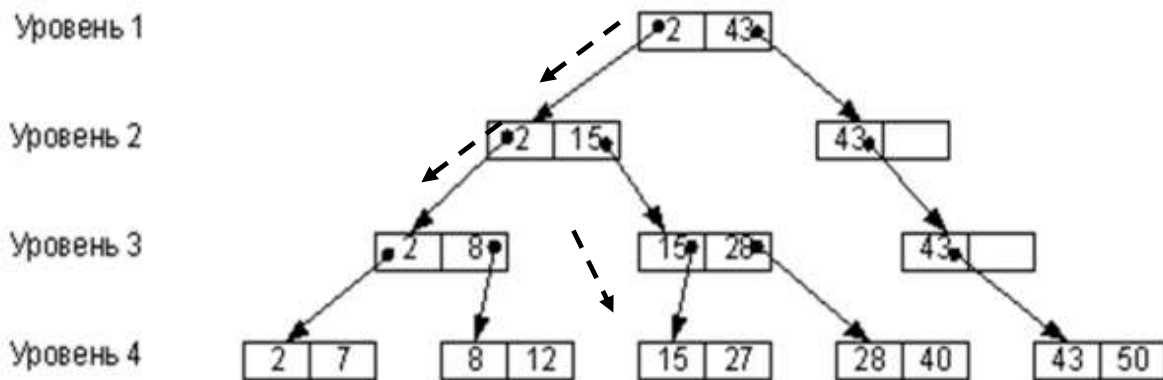


Рис. 11. Поиск данных в В-дереве

SQL Server использует индексы 2-х типов: кластерные и некластерные. В кластерных индексах блоки нижнего уровня (листья) содержат строки таблицы. При создании и изменении кластерного индекса требуется изменение местоположения записей в таблице в соответствии со структурой индекса. Для таблицы может быть определен только один кластерный индекс.

В некластерных индексах блоки нижнего уровня (листья) содержат блоки индекса, поэтому при их создании или изменении физическое расположение записей основной таблицы не меняется. Для одной таблицы можно определить несколько некластерных индексов, однако их количество не должно быть большим. Использование индексов позволяет повысить производительность базы данных, особенно при работе с большими таблицами.

Ограничения целостности - это правила, соблюдение которых гарантируют целостность данных для таблиц и столбцов.

СУБД позволяют поддерживать разные ограничения целостности, однако наиболее важными среди них являются:

1. Ограничение по первичному ключу (PRIMARY KEY). Используется для обеспечения целостности сущности и гарантирует, что для всех строк

таблицы будет существовать уникальный ключ, не равный NULL. Применение ограничения первичного ключа создает также уникальный индекс по таблице.

2. Ограничение по внешнему ключу (FOREIGN KEY). Используется для обеспечения ссылочной целостности. Внешний ключ связывает один или несколько столбцов в подчиненной таблице с первичным ключом главной таблицы. SQL Server требует, чтобы главная и подчиненная таблицы находились в одной и той же базе данных и каждый столбец внешнего ключа был того же типа и размера, что и соответствующий ему столбец главной таблицы.

3. Ограничение на диапазоне (CHECK). Обеспечивает контроль значений, которые могут быть введены в столбец. Ограничение на значения определяются на уровне столбца.

4. Ограничение NOT NULL (ограничение на неопределенное значение). Используется в том случае, если значение столбца должно быть обязательно определено. Действует только на уровне столбца.

Представление (views) – это виртуальная таблица, содержимое которой определяется запросом. Физически представление реализуется как инструкция, на основе которой производится выборка данных из одной или нескольких таблиц или представлений.

Представления создают для:

- группировки столбцов разных таблиц в виде одного объекта;
- для ограниченного доступа пользователей к определенным строкам или определенным столбцам таблицы;
- для просмотра информации, полученной в результате преобразования данных столбца. Поле представления может содержать агрегированную информацию (sum, max, min и т.д.)

Хранимая процедура (stored procedure) – это именованный набор команд Transact SQL, хранящейся на сервере и являющийся самостоятельным объектом базы данных. Вызов хранимой процедуры может осуществляться клиентской программой, другой хранимой процедурой или триггером.

Триггер – это хранимая процедура, которая выполняется автоматически при возникновении события для объектов базы данных или сервера.

Архитектура базы данных на физическом уровне

При работе с базой данных пользователь оперирует ее логическими компонентами – таблицами, представлениями, процедурами и т.д. Тем не менее, для профессиональной работы с базами данных, необходимо понимать, как устроена база данных и на физическом уровне.

Базы данных в SQL Server хранятся в файлах. Способ хранения каждой базы данных в одном или нескольких файлах позволяет динамически наращивать и уменьшать размеры базы данных. Для каждой базы данных создается два файла:

- файл базы данных (например, master.mdf);
- файл журнала транзакций (например, mastlog.ldf).

Файлы базы данных имеют два имени.

– *логическое* — имя, используемое для ссылки на физический файл во всех инструкциях Transact-SQL. Логическое имя файла должно соответствовать правилам для идентификаторов SQL Server и быть уникальным среди логических имен файлов в соответствующей базе данных;

– *физическое* — это имя физического файла, включая путь к каталогу. Оно должно соответствовать правилам для имен файлов операционной системы. Местоположение файлов базы данных указывается при ее создании.

База данных может располагаться в нескольких файлах: первичном и вторичных. Вторичные файлы базы данных включают все файлы данных, которые не являются первичными файлами данных. Базы данных могут не иметь ни одного вторичного файла данных, или иметь несколько вторичных файлов данных.

Основной единицей хранения данных на уровне файла базы данных является страница (рис.12). **Страница** представляет собой блок фиксированной длины, обрабатываемый при дисковых операциях ввода/вывода как единое целое. Таким образом, если пользователь обращается к одной строке данных, с диска будет считана целиком вся страница, в которой эта строка находится.

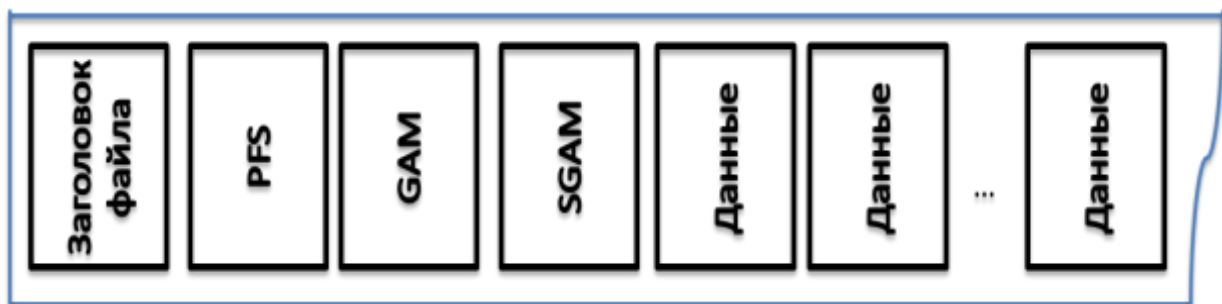


Рис. 12. Страничная структура файла базы данных

Каждая страница в файле идентифицирована. Идентификатор страницы содержит идентификатор файла и номер страницы в файле.

Первые страницы каждого файла являются системными и одна из них представляет собой загрузочную страницу базы данных, которая содержит сведения об атрибутах этой базы данных. Далее следуют страницы разных типов. В таблице 1 приведена характеристика информации, хранящейся на страницах разного типа.

Таблица 1. Типы страниц

Типы страниц	Тип информации
Данные (Data)	Строки данных, содержащиеся в таблицах, за исключением данных больших объемов (text, image и т.д.)
Индекс (Index)	Индексные записи
Текст / изображение	Текстовые и графические данные больших объемов (text, image и т.д.)
Глобальная карта распределения (GAM-Global Allocation Map) Общая глобальная карта распределения(SGAM)	Информация о выделенных (используемых) экстентах
Свободное пространство на страницах (PFS-Page Free Space)	Информация о наличии свободного пространства на страницах
Карта распределения индексов (IAM-Index Allocation Map)	Информация об экстентах, используемых таблицей либо индексом для единицы распределения

Страницы всех типов имеют одинаковую размер 8Кбайт и одинаковую структуру. Первые 96 байт отводятся под заголовок страницы, оставшаяся часть представляет собой тело страницы. Страницы разных типов содержат в области тела разные данные.

На рисунке 13 показана базовая структура страницы данных. Заголовок страницы содержит системную информацию – тип страницы, объем свободного пространства на странице, идентификационный номер объекта базы данных (таблицы), которому принадлежит страница, идентификаторы следующей и предыдущей страницы в цепи страниц и т.д.

В теле страницы данных размещаются строки таблицы. Каждая страница может хранить строки только одной таблицы. Строки данных заносятся на страницу последовательно, сразу же после заголовка. Для определения положения строки с конца страницы размещается таблица смещения строк, которая содержит одну запись для каждой строки на странице. Каждая запись таблицы смещения имеет размер 2 байта и состоит из номера строки (первый байт) и байта смещения адреса этой строки на странице.

Индексная информация храниться на индексных страницах. Структура индексной страницы является аналогичной структуре страницы данных. Разница заключается в типе информации, хранящейся на странице. В общем случае строка в индексной странице содержит ключевой индекс и указатель на страницу на следующем (нижнем) уровне. Фактическая

информация, хранящаяся на индексной странице, зависит от типа индексной страницы, а также от того, имеем ли мы дело со страницей уровня листьев либо нет.

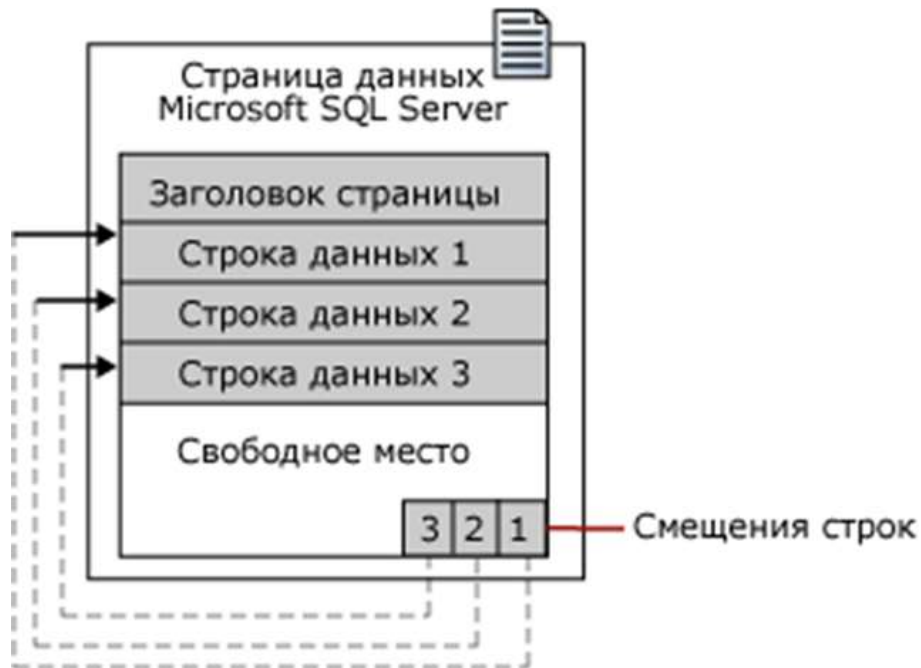


Рис. 13. Структура страницы данных

Данные из разных строк таблицы, которые требуется читать могут находиться на разных страницах данных и в разных местах диска. С целью экономного распределения места и более эффективного управления памятью страницы объединяются в экстененты.

Экстент – это коллекция, состоящая из восьми физически непрерывных страниц.

Определены экстененты двух типов: однородные (uniform) и смешанные (mixed).

В однородном экстененте все восемь страниц могут быть использованы только одним владеющим объектом (принадлежат определенной таблице или индексу).

Смешанные экстененты могут находиться в общем пользовании, у них может быть не более восьми объектов. Каждая из восьми принадлежащих экстененту страниц может находиться во владении разных объектов.

Первоначально новой таблице или индексу выделяется смешанный экстент. При увеличении размера таблицы или индекса до восьми страниц эти таблица или индекс переходят на использование однородных экстенентов для последовательных единиц распределения.

Управление размещением экстенента осуществляется с использованием карт двух типов:

- глобальная карта распределения (GAM);
- общая глобальная карта распределения (SGAM).

На **GAM-страницах** записано, какие экстененты были размещены. В карте GAM приходится по одному биту на каждый экстенент, т.е. карта GAM содержит информацию о 64 000 экстенентах (почти 4 ГБ данных). Если бит карты равен 1, то экстенент свободен; если бит равен 0, то экстенент размещен.

На **SGAM-страницах** записано, какие экстененты в текущий момент используются в качестве смешанных экстенентов и имеют как минимум одну неиспользуемую страницу. В карте SGAM приходится по одному биту на каждый экстенент, то есть также содержится информация о 64 000 экстенентах. Если бит равен 1, то экстенент используется как смешанный экстенент и имеет свободную страницу. Если бит равен 0, то экстенент однородный, или он является смешанным экстенентом, но все его страницы используются.

Отслеживания свободного места в системе осуществляется с помощью страницы Page Free Space. На **страницы PFS** записывается состояние размещения каждой страницы, информация о том, была ли отдельная страница использована или нет, а также количество свободного места на каждой странице. В PFS на каждую страницу приходится по одному байту, хранящему информацию о том, была ли страница использована или нет, а если была — то пустая она, или ее заполнение находится в промежутке от 1 до 50%, от 51% до 80%, от 81% до 95% или от 96% до 100%.

Страница PFS является страницей номер один в каждом файле и для каждых 8 тыс. страниц в файле должна существовать отдельная страница PFS (рис. 12). В файле за страницей PFS следует пара страниц GAM и SGAM, для каждых 64000 тыс. экстенентов в файл будет добавляться пара новых страниц GAM и SGAM.

На страницах данных строки таблицы могут храниться по порядку ключа кластеризованного индекса, или без определенного порядка, в той последовательности, в которой строки добавлялись в таблицу. Таблицы, строки данных которых хранятся без определенного порядка называются **кучи**. Страницы **IAM** (Index Allocation Map) выполняют отслеживание экстенентов, используемых определенной кучей или индексом.

Каждый бит карты IAM-страницы представляет экстенент, первый бит схемы представляет первый экстенент диапазона, второй бит — второй экстенент и т. д. Если бит равен 0, то соответствующий ему экстенент не привязан к единице распределения, которой принадлежит IAM-страница. Если он равен 1, то соответствующий ему экстенент привязан к единице распределения, которой принадлежит IAM-страница. Страницы IAM выделяются при необходимости, а затем распределяются случайным образом среди файлов базы данных.

Размеры файлов базы данных могут увеличиваться автоматически, превышая изначально заданную величину. При определении файла можно задать степень приращения. Каждый раз при добавлении данных в файл происходит увеличение его размера на величину, заданную степенью приращения. Кроме этого для каждого файла может быть задан максимально допустимый размер. Если этот параметр не задан, увеличение размеров файла будет происходить до тех пор, пока на диске существует свободное пространство.

Файлы базы данных могут группироваться в файловые группы, что облегчает работу по распределению данных и администрированию. Для файлов и файловых групп действуют следующие правила:

- файл или файловая группа не могут использоваться несколькими базами данных;
- файл может быть элементом только одной файловой группы;
- файлы журнала транзакций не могут входить ни в какие файловые группы.

3. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

3.1. Этапы и методы проектирование реляционных баз данных

При проектировании базы данных необходимо решение двух основных проблем:

- проблемы логического проектирования – каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области и было по возможности наилучшим с точки зрения возможности получения данных;

- проблемы физического проектирования – каким образом расположить данные во внешней памяти, чтобы обеспечить эффективность выполнения запросов к базе данных.

Очевидно, что физическому и логическому проектированию должен предшествовать анализ собственно целей и задач создания БД. В проектировании базы данных выделяют три этапа:

- системный анализ;
- логическое проектирование;
- физическое проектирование.

Системный анализ. На этом этапе осуществляется:

- определение потребностей и целей создания БД;
- выделение предметной области из окружающей среды;
- формирование возможных информационных объектов;
- исследование информационных потоков;
- выбор типа модели данных.

Системный анализ осуществляется с использованием *предметного* подхода, суть которого заключается в том, что в состав описания предметной области включаются те объекты и взаимосвязи между ними, которые наиболее характерны и существенны для предметной области.

Этап **логического проектирования** включает в себя инфологическое и даталогическое проектирование.

В результате инфологического проектирования:

- выявляются характеристики (свойства) информационных объектов;
- осуществляется окончательное формирование информационных объектов;
- выполняется построение инфологической модели.

Далее полученная в результате логического проектирования модель трансформируется в даталогическую модель. Перед описанием этой модели осуществляется выбор целевой СУБД. Даталогическая модель включает:

- Описание логической структуры информационных объектов;
- Описание механизмов обеспечения целостности данных.

Физическое проектирование заключается в реализации даталогической модели и решений по физическому размещению данных во внешней памяти средствами выбранной СУБД. Этот этап завершается после отладки проекта и проверке базы данных на работоспособность.

Современные программные средства автоматизированного проектирования CASE-средства (Computer-Aided Software/System Engineering) позволяют при проектировании базы данных совместить первые два этапа, выполняя с использованием графических средств моделирование данных и генерацию схем баз данных (как правило, на языке SQL) для наиболее распространенных СУБД. Однако, даже выполняя автоматизированное проектирование реляционной базы данных требуется понимать суть методов, которые положены в основу моделирования. Поэтому более подробно рассмотрим методы логического проектирования.

Инфологическое проектирование методом нормализации данных

В реляционной модели данных утверждается, что единственной структурой является нормализованное n -арное отношение. Нормализованном является отношение, схема которого соответствует определенным требованиям. Процесс приведения схемы отношения к нормализованной называется нормализацией данных.

Нормализация данных – это формальный аппарат ограничений на формирование отношений. Инфологическое проектирование методом нормализации данных проводится, как правило, *методом последовательных приближений к удовлетворительному набору схем отношений*, описывающим разбиение отношения на две и более части.

Целью нормализации является получение такой схемы базы данных, в которой каждый элемент данных хранится в базе в одном и только одном экземпляре, т. е. исключены избыточность и противоречивость в хранимых данных. Избыточное дублирование может привести к проблемам при модификации данных (аномалии).

Аномалиями называют такую ситуацию в базе данных, которая приводит к противоречивости данных и усложняет обработку данных.

Различают три вида аномалий:

Аномалии модификации – проявляются в том, что изменение значения одного данного может повлечь за собой необходимость просмотра всех кортежей отношения и изменения некоторых из них, содержащих изменяемое значение.

Аномалии модификации – проявляются в том, что изменение значения одного данного может повлечь за собой необходимость просмотра всех

кортежей отношения и изменения некоторых из них, содержащих изменяемое значение.

Аномалии добавления возникают в том случае, когда добавляются не полные сведения.

Рассмотрим метод нормализации данных на примере. Отправной точкой в проектировании является представление предметной области в виде одного или нескольких отношений.

Условие задания на проектирование. Требуется разработать инфологическую модель базы данных *Университет* для хранения данных, позволяющих выполнять учет и анализ данных успеваемости студентов. Анализ успеваемости должен проводиться для каждого студента, по каждой дисциплине, для группы, по направлению или специальности и для факультета в целом. Очевидно, что база данных должна хранить информацию о студентах и результатах сдачи экзаменов. Для того, чтобы иметь возможность получения обобщенных данных по успеваемости должны быть определены характеристики, указывающие на принадлежность студента к группе, специальности и факультету.

Определим два отношения Студент и Общая ведомость:

Студент (Номер группы, Фамилия, Номер зачетной книжки, Дата рождения, Номер факультета, Наименование факультета, Код специальности, Наименование специальности, Стоимость обучения, Коммерческий)

Общая ведомость (Номер зачетной книжки, Код дисциплины, Наименование дисциплины, Оценка)

Метод последовательных приближений к удовлетворительному набору схем отношений, заключается в разбиение отношения на две и более части в том случае, если отношение не соответствует так называемой нормальной форме.

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

В теории реляционных баз данных утверждается следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Нормальные формы обладают свойствами:

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

Процесс проектирования заключается в определении того, находится ли отношение в указанной нормальной форме. Если требуется, чтобы отношение как минимум находилось в 3NF, то необходимо сначала привести его к 1NF, затем проверить соответствует ли оно ограничениям 2NF и, если оно находится в 2NF, приводить к 3NF.

Ограничения нормальных форм отношений основываются на фундаментальном в теории реляционных баз данных понятии **функциональной зависимости**. Поясним определение функциональных зависимостей.

В отношении R атрибут Y **функционально** зависит от атрибута X (X и Y могут быть составными) в том и только в том случае, если каждому значению X соответствует в точности одно значение Y: $R.X \rightarrow R.Y$.

Например, в отношении студент

Студент (Номер группы, Фамилия, Номер зачетной книжки, Дата рождения, Номер факультета, Наименование факультета, Код специальности, Наименование специальности, Стоимость обучения, Коммерческий)

Существуют функциональные зависимости между:

Номер зачетной книжки \rightarrow Фамилия

Номер зачетной книжки \rightarrow Дата рождения

Каждый студент имеет только одну фамилию, и одну дату рождения.

Не существует функциональных зависимостей между

Номер группы \nrightarrow Номер Зачетной книжки

Номер факультета \nrightarrow Фамилия

Номер зачетной книжки функционально не зависит от номера группы, так как в каждой группе несколько студентов, поэтому одному номеру группы может соответствовать несколько номеров зачетных книжек разных студентов. Аналогично одному номеру факультета может соответствовать несколько фамилий, так как на одном факультете обучается много студентов.

Функциональная зависимость $R.X \rightarrow R.Y$ называется **полной**, если атрибут Y не зависит функционально от любого подмножества X.

Например, в отношении

Общая ведомость (Номер зачетной книжки, Код дисциплины, Наименование дисциплины, Оценка)

атрибут *Оценка* функционально полно зависит от пары значений, потому что это результат сдачи экзамена по конкретной дисциплине конкретным студентом.

$\{\text{Номер зачетной книжки}, \text{Код дисциплины}\} \rightarrow \text{Оценка}$

Атрибут *Наименование дисциплины* функционально полно не зависит от пары значений (*Номер зачетной книжки*, *Код дисциплины*), а зависит только от кода дисциплины

$\{\text{Номер зачетной книжки}, \text{Код дисциплины}\} \nrightarrow \text{Наименование дисциплины}$

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

В отношении *Студент* существует транзитивная зависимость между атрибутами *Номер зачетной книжки* и *Наименование факультета*, поскольку существует такой атрибут *Номер факультета*, что имеются функциональные зависимости

$\text{Номер зачетной книжки} \rightarrow \text{Номер факультета}$

$\text{Номер факультета} \rightarrow \text{Наименование факультета}$

и отсутствует функциональная зависимость

$\text{Номер факультета} \nrightarrow \text{Номер зачетной книжки}$

Два или более атрибутов **взаимно независимы**, если ни один из этих атрибутов не является функционально зависимым от других.

Правило первой нормальной формы (1NF)

Отношение R находится в первой нормальной форме (1NF) в том и только в том случае, если значения его атрибутов атомарны (неделимы).

В рассматриваемом примере все атрибуты отношений атомарные, то есть нет таких значений, которые представляли бы собой совокупность из нескольких значений. Поэтому можно утверждать, что отношения *Студент* и общая ведомость находятся в 1NF.

Правило второй нормальной формы (2NF)

Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда находится в 1NF, и каждый неключевой атрибут функционально полно зависит от первичного ключа.

Во второй форме может находиться отношение в котором определен первичный ключ.

В отношении *Студент* первичным ключом может быть выбран *Номер зачетной книжки*, так как у каждого студента он имеет уникальное значение и не может быть студента, для которого номер зачетной книжки не имеет значения. Все остальные атрибуты функционально зависят от ключа, так как для конкретного студента имеется только по одному значению каждого из них.

Нужно определить первичный ключ в отношении *Общая ведомость*. *Номер зачетной книжки* не может быть первичным ключом, поскольку один студент сдает много экзаменов и отношение может иметь несколько кортежей с одинаковым значением номера зачетной книжки. Точно также и *Код дисциплины* не может быть первичным ключом, так как экзамен по одной дисциплине может сдавать несколько студентов. А может ли повторяться в совокупности пара значений *Номер зачетной книжки* и *Код дисциплины*? Нет, потому что не может быть двух кортежей, содержащих результат сдачи экзамена по одному предмету одним студентом. Таким образом мы пришли к выводу, что первичный ключ отношения *Общая ведомость* составной и представляется парой значений, значением атрибута *Номер зачетной книжки* и значением атрибута *Код дисциплины*.

Чтобы отношение *Общая ведомость* находилось в 2NF необходимо, чтобы не ключевые атрибуты находились в полной функциональной зависимости от первичного ключа.

Разбирая определение полной функциональной зависимости на примере отношения *Общая ведомость* мы выявили, что функционально полно от составного ключа зависит только атрибут *Оценка*, так как атрибут *Наименование дисциплины* зависит от атрибута *Код дисциплины*, который является частью составного ключа.

Необходимо изменить схему отношения *Общая ведомость* для того, чтобы она находилась в 2NF.

Используемый метод выделения новых схем называется «**декомпозиция без потерь**». Суть метода: каждая новая схема является проекцией по отношению к схеме исходного отношения, и выполненная декомпозиция должна обеспечить то, что запросы к отношениям полученным в результате декомпозиции и к исходному отношению дадут одинаковый результат.

Выполнив проекцию на *Код дисциплины* и *Наименование дисциплины* в отношении *Общая ведомость* получаем новое отношение:

Дисциплина (Код дисциплины, Наименование дисциплины)

Первичный ключ полученного отношения *Код дисциплины*, отношение в 2NF.

Выполняем проекцию на все атрибуты, кроме *Наименования дисциплины* в *Общей ведомости* и получает отношение, которое находится в 2NF:

Общая ведомость (Номер зачетной книжки, Код дисциплины, Оценка)

Правило третьей нормальной формы (3NF)

Отношение **R** находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 2NF и каждый не ключевой атрибут не транзитивно зависит от первичного ключа.

Отношения *Дисциплина* и *Общая ведомость* находятся в 3NF, так как у них по одному не ключевому атрибуту, и транзитивная зависимость не возможна. Транзитивная зависимость возможна только в том случае, когда существует функциональная зависимость между не ключевыми атрибутами.

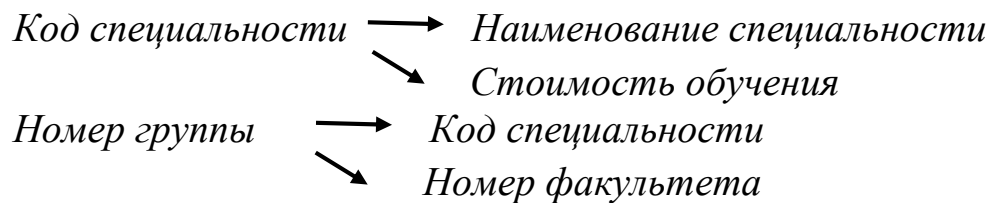
Посмотрим еще раз на отношение *Студент*:

Студент (Номер группы, Фамилия, Номер зачетной книжки, Дата рождения, Номер факультета, Наименование факультета, Код специальности, Наименование специальности, Стоимость обучения, Коммерческий)

Разбирая определение транзитивной зависимости на примере отношения *Студент* мы выявили функциональную зависимость между не ключевыми атрибутами:

Номер факультета \longrightarrow *Наименование факультета*

Кроме этого существуют функциональные зависимости между следующими не ключевыми атрибутами



После декомпозиции отношения студент в результате выполнения четырех проекций будут получены следующие отношения:

Студент (Номер группы, Фамилия, Номер зачетной книжки, Дата рождения, Коммерческий)

Факультет (Номер факультета, Наименование факультета)

Специальность (Код специальности, Наименование специальности, Стоимость обучения)

Группа (Номер группы, Код специальности, Номер факультета)

На практике третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

Нормальная форма Бойса-Кодда (усиленная 3NF)

Отношение R находится в нормальной форме Бойса-Кодда (BCNF) в том и только в том случае, если каждый детерминант является возможным ключом.

Детерминант - любой атрибут, от которого полностью функционально зависит некоторый другой атрибут. Если в отношении нет возможных ключей, то 3NF и BCNF эквивалентны.

Рассмотрим для примера отношение:

Консультация (Номер зачетной книжки, Код дисциплины, Преподаватель).

Кортежи отношения содержат информацию о том, у кого студент может консультироваться и по каким дисциплинам. Следует также учесть, что:

- студент может консультироваться по нескольким дисциплинам,
- по одной и той же дисциплине может консультировать несколько преподавателей,
- но один и тот же преподаватель может консультировать только по одной дисциплине.

Поищем возможные ключи:

Номер зачетной книжки не может быть ключом, т.к. студент может консультироваться по нескольким дисциплинам

Код дисциплины + Преподаватель не может быть ключом, так как тогда преподаватель сможет консультировать по предмету только одного студента

Номер зачетной книжки + Код дисциплины может, так как студент по одной дисциплине консультируется у одного преподавателя

Номер зачетной книжки + Преподаватель может, так как студент у одного и того же преподавателя консультируется только по одной дисциплине.

По условию существует также зависимость - преподаватель консультирует только по одной дисциплине. Атрибут преподаватель – детерминант.

Посмотрим какие аномалии возможны на примере данных таблицы 2.

Таблица 2. Отношение *Консультация*

Номер зачетной книжки	Код дисциплины	Преподаватель
1001/01	11	Иванов
1001/01	12	Петров
1002/01	11	Кузьмин
1003/01	11	Иванов

Возможна аномалия удаления: если студент 1001/01 перестает консультироваться по предмету 12, то теряются сведения о консультациях преподавателя Петров по дисциплине 12.

Возможно аномалии изменения: при изменении кода дисциплины 11, требуется обновлять несколько строк, так как консультируют по этой дисциплине несколько преподавателей.

Для приведения к BCNF требуется выполнить декомпозицию отношения *Консультация* следующим образом (рис. 14). Первичным ключом первого отношения являются атрибуты *Номер зачетной книжки* и *Преподаватель*, первичным ключом второго – атрибут *Преподаватель*.

Номер зачетной книжки	Преподаватель	Преподаватель	Код дисциплины
1001/01	Иванов	Иванов	11
1001/10	Петров	Петров	12
1002/01	Кузьмин	Кузьмин	11
1003/01	Иванов		

Рис. 14. Результаты декомпозиции отношения *Консультация*

Правило четвертой нормальной формы (4NF)

Введем понятие многозначной зависимости.

Многозначная зависимость $R.A \twoheadrightarrow R.B$ в отношении $R(A,B,C)$ существует в том и только в том случае, если множество значений B соответствующее A и C зависит только от A и не зависит от C .

Для примера рассмотрим отношение, хранящее сведения об экзаменах, которые сдают студенты:

Сессия (Номер зачетной книжки, Номер группы, Код дисциплины)

В отношении две многозначные зависимости:

Номер группы \twoheadrightarrow Номер зачетной книжки

Номер группы \twoheadrightarrow Код дисциплины

Пусть отношение *Сессия* хранит следующие данные (таблица 3).

Таблица 3. Отношение *Сессия*

Номер зачетной книжки (B)	Номер группы (A)	Код дисциплины (C)
1001/01	ИБ1601	11
1002/01	ИБ1601	11
1003/01	ИБ1601	11
1001/01	ИБ1601	12
1002/01	ИБ1601	12
1003/01	ИБ1601	12
1004/01	ИБ1602	13
1004/01	ИБ1602	14

Анализируя данные таблицы 3 можно сделать вывод о возможных аномалиях.

Аномалия изменения: при изменении учебного плана группы, следует изменять все кортежи студентов этой группы

Аномалии добавления: при добавлении нового студента в группу, нужно добавлять столько кортежей, сколько экзаменов сдает группа

Отношение R находится в четвертой нормальной форме (4NF) в том и только в том случае, если в случае существования многозначной зависимости $A \twoheadrightarrow B$ все остальные атрибуты отношения функционально зависят от A.

Декомпозиция отношения, имеющего многозначную зависимость основывается на теореме Фэйджина, которая утверждает:

Отношение $R(A,B,C)$ можно спроецировать без потерь в отношения $R_1(A,B)$ и $R_2(A,C)$ в том и только том случае, если существуют многозначные зависимости $A \twoheadrightarrow B$ и $A \twoheadrightarrow C$.

Выполним декомпозицию исходного отношения по теореме Фэйджина и получим отношения, приведенные на рисунке 15.

R(A,C)		R(A,B)	
Номер группы	Код дисциплины	Номер зачетной книжки	Номер группы
ИБ1601	11	Иванов	11
1001/10	12	Петров	12
1002/01	13	Кузьмин	11
1003/01	14		

Рис. 15. Результаты декомпозиции отношения *Сессия*

Правило пятой нормальной формы (PJ/NF)

Под проецированием без потерь понимается такой способ декомпозиции отношения, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений. Во всех рассмотренных до этого момента нормализациях производилась декомпозиция одного отношения в два.

Отношение R находится в пятой нормальной форме (нормальной форме проекции-соединения - PJNF) в том и только в том случае, когда в каждой его полной декомпозиции все проекции содержат возможный ключ.

В качестве примера приведем отношение, содержащее сведения о том какие дисциплины ведет преподаватель:

(Кафедра, Дисциплина, Преподаватель)

Если преподаватель может работать на нескольких кафедрах и вести несколько дисциплин, то отношение находится в 4NF (ключ – полный набор атрибутов), но не находится в 5NF.

Чтобы привести отношение к 5NF его следует разбить на 3 отношения:

(Преподаватель, Кафедра)

(Преподаватель, Предмет)

(Кафедра, Предмет)

Пятая нормальная форма - это последняя нормальная форма, которую можно получить путем декомпозиции.

Если в процессе нормализации данных осуществляется декомпозиция исходных отношений, то их число растет. Большое число отношений в схеме базы данных приведет к:

- увеличению операций обмена данными с диском;
- увеличению времени на обработку запросов.

Это в конечном счете приведет к снижению скорости работы всей системы в целом. С целью повышения производительности работы системы возможна денормализация данных.

Денормализация – это процесс проектирования данных, при котором уровень требований нормализации данных снижается (до 2NF или 3NF).

В результате приведения исходных отношений рассматриваемого примера к 3NF была выполнена их декомпозиция в отношения:

Студент (Номер группы, Фамилия, Номер зачетной книжки, Дата рождения, Коммерческий);

Группа (Номер группы, Код специальности, Номер факультета);

Специальность (Код специальности, Наименование специальности, Стоимость обучения);

Факультет (Номер факультета, Наименование факультета);

Дисциплина (Код дисциплины, Наименование дисциплины);

Общая ведомость (Номер зачетной книжки, Код дисциплины, Оценка).

На рисунке 16 представлена инфологическая модель, которая отражает взаимосвязи между полученными в результате проектирования отношениями.

В модели отражены связи между:

- отношениями *Факультет* и *Группа* по их общему атрибуту *Номер факультета*;
- отношениями *Специальность* и *Группа* и по их общему атрибуту *Код специальности*;
- отношениями *Группа* и *Студент* по их общему атрибуту *Номер группы*;

- отношениями Студент и Общая ведомость по их общему атрибуту Номер зачетной книжки;
- отношениями Дисциплина и Общая ведомость по их общему атрибуту Код дисциплины.

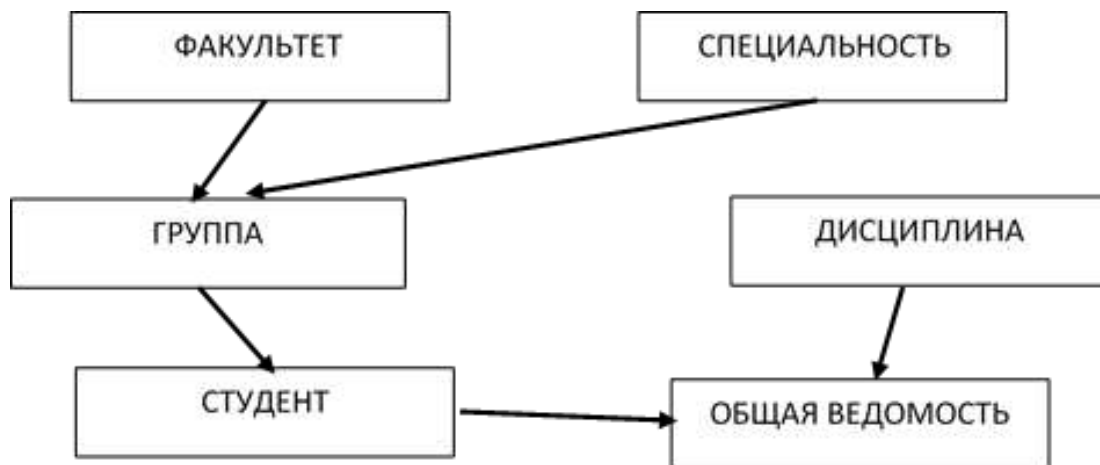


Рис. 16. Инфологическая модель предметной области

Тип всех установленных связей один – ко многим. Возможные типы связей будут рассмотрены при изложении семантической модели данных.

Стоит, однако, обратить внимание на то, что полученная инфологическая модель, в целом удовлетворяющая требованиям нормализации, с точки зрения логики может некорректно описывать предметную область. Модель, показанная на рис. 16, допускает существование некоторой специальности, которая не преподается ни на одном факультете! Естественным желанием при этом является добавление связи между объектами ФАКУЛЬТЕТ и СПЕЦИАЛЬНОСТЬ. Альтернативой этому может быть определение домена для атрибута Код специальности, который будет содержать только «существующие» номера специальностей.

3.2. Семантическая модель данных

Потребности проектировщиков баз данных в более удобных и мощных средствах моделирования предметной области привели к появлению семантических моделей данных. При этом любая развитая семантическая модель данных, как и реляционная модель, включает структурную, манипуляционную и целостную части.

Главным назначением семантических моделей является обеспечение возможности выражения семантики (смысла) данных и их взаимосвязей.

Семантическое моделирование используется на первых стадиях проектирования базы данных для представления концептуальной модели базы

данных в терминах семантической модели. Затем полученная модель вручную или автоматизировано преобразуется к реляционной (или какой-либо другой) схеме.

Одной из наиболее популярных семантических моделей данных является модель *Сущность-Связь* (часто ее называют кратко *ER-моделью*). Модель была предложена Питером Ченом в 1976 г. Моделирование предметной области в этом случае базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. В связи с наглядностью представления концептуальных схем баз данных, *ER-модели* получили широкое распространение в системах *CASE*, поддерживающих автоматизированное проектирование реляционных баз данных.

Основными понятиями *ER-модели* являются *сущность*, *связь* и *атрибут*.

Сущность - это реальный или абстрактный объект, информация о котором должна сохраняться и быть доступна.

Атрибутом сущности является любая значимая деталь, которая служит для идентификации, классификации, числовой характеристики или выражения состояния сущности (свойства сущности).

Множество сущностей одного типа (обладающих одинаковыми атрибутами) представляют **набор сущностей**. При определении типа сущности необходимо гарантировать, что каждый экземпляр сущности является отличным от любого другого экземпляра той же сущности.

Связь - это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь).

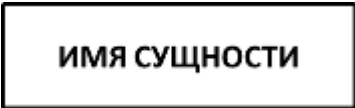
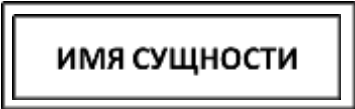





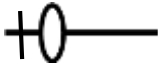

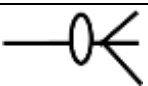

В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается **имя конца связи**, **степень конца связи** (сколько экземпляров данной сущности связывается), **обязательность связи** (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи).

Для представления модели *Сущность-Связь* используется *ER-диаграмма*. В процессе построения диаграммы можно выделить несколько этапов:

- Идентификация сущностей и связей.
- Идентификация семантической информации в наборах связей (например, является ли некоторый набор связей отображением один-ко-многим).
- Определение кардинальностей связей.
- Определение атрибутов и наборов их значений (доменов).
- Организация данных в виде отношений "сущность-связь".

Символы, применяемые для графического представления модели, весьма различны. Существует несколько различных нотаций (языков) изображения ER-диаграмм. В таблице 4 приведены обозначения сущностей, атрибутов и связей в нотации Чена и обозначения, степеней и кардинальности связей в нотации Мартина. Как видно из таблицы в ER-диаграммах сущность представляется прямоугольником, внутри которого указывается имя сущности. Имя сущности – это имя типа сущности, а не некоторого конкретного экземпляра этого типа. Имя сущности должно быть существительным в единственном числе.

Таблица 4. Элементы ER-диаграммы

Обозначение	Пояснение
	Набор независимых сущностей
	Набор зависимых сущностей
	Атрибут
	Ключевой атрибут
	Набор связей
	Кардинальность – нет
	Кардинальность 1,1
	Кардинальность 0,1
	Кардинальность N,M
	Кардинальность 0,N
	Кардинальность 1,N

Связь представляется в виде линии, связывающей две сущности или ведущей от сущности к ней же самой.

То число сущностей, которое может быть ассоциировано через набор связей с другой сущностью, называют **степенью связи**.

Могут существовать следующие степени бинарных связей:

- один к одному (1 : 1);
- один ко многим (1 : N);
- много к одному (N : 1);
- многие ко многим (N : N).

Значение 1 указывает, что один экземпляр сущности ассоциирован через набор с другой сущностью, а значение n - на наличие нескольких таких экземпляров.

Максимальное количество сущностей на каждой стороне связи определяет кардинальность связи (класс принадлежности).



Рис. 17. Пример кардинальности бинарных связей степени 1

В примере на рисунке 17 сущность **СТУДЕНТ** имеет обязательный класс принадлежности, так как хотя бы один студент проживает в общежитии. Обозначается $\text{+} \text{---}$ и указывает интервал числа возможных вхождений сущности в связь (1,1).

Сущность **ОБЩЕЖИТИЕ** имеет необязательный класс принадлежности связи, так как есть студенты, не проживающие в общежитии. Обозначается $\text{---} \text{O}$ и указывает интервал (0,1).

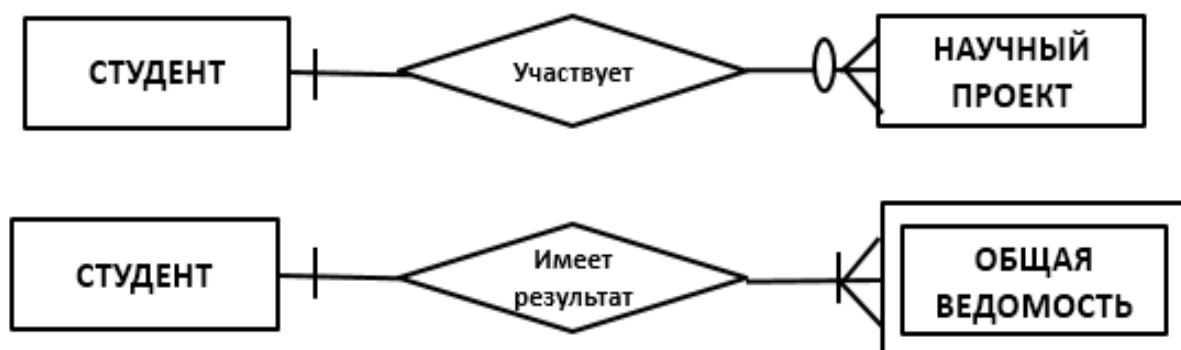


Рис. 18. Пример кардинальности бинарных связей степени N

В первом примере, приведенных на рисунке 18 кардинальность связи $(0,N)$, так как есть студенты, которые не участвуют в научных проектах, или участвуют в одном или нескольких проектах. В тоже время нет таких научных проектов, в которых не участвовали бы студенты. Во втором примере, если утверждать что нет такого студента, который бы не сдавал экзамены, то кардинальность такой связи будет $(1,N)$.

Если существование сущности зависит от существования другой сущности, то она называется зависимой сущностью. В примере на рисунке 18 сущность *Общая ведомость* зависимая, так как не может существовать оценки студента, которого нет.

Сущность может быть расщеплена на два или большее число взаимно исключающих подтипов, каждый из которых включает общие атрибуты. В этом случае супертип будет содержать совместно используемые атрибуты, а подтип – уникальные атрибуты. На рисунке 19 приведен пример супер-типа *Сотрудник* и подтипы сущностей *Специалист* и *Преподаватель*.



Рис. 19. Сущности супертипы и подтипы

Между супертипом *Сотрудник* и его подтипами *Специалист* и *Преподаватель* поддерживается кардинальность которой $(1,1)$.

Рассмотренные нотации представления сущностей, связей в ER-диаграммах использовались при ручном проектировании ER-модели базы данных. Появление средств автоматизированного проектирования привело к необходимости разработки стандарта языка представления данных ER-

модели. Таким стандартом для разработки реляционных баз данных является метод *IDEFIX*, который описывает условный синтаксис, специально разработанный для удобного построения семантической модели. Использование метода *IDEFIX* наиболее целесообразно для построения логической структуры базы данных после того, как все информационные ресурсы исследованы и решение о внедрении базы данных, как части корпоративной информационной системы, принято.

Сущность в *IDEFIX* описывает собой совокупность или набор экземпляров похожих по свойствам, но однозначно отличаемых друг от друга по одному или нескольким признакам. Каждый экземпляр является реализацией сущности. Таким образом, сущность в *IDEFIX* описывает конкретный набор экземпляров предметной области. Примером сущности *IDEFIX* может быть сущность *СТУДЕНТ*.

Сущность описывается в диаграмме *IDEFIX* графическим объектом в виде прямоугольника (рис. 20). Каждый прямоугольник, отображающий собой сущность, разделяется горизонтальной линией на часть, в которой расположены ключевые атрибуты (*ключевая область*) и часть, где расположены неключевые атрибуты (*областью данных*).



Рис. 20. Сущность *СТУДЕНТ*

Ключевая область содержит первичный ключ для сущности. *Первичный ключ* - это набор атрибутов, выбранных для идентификации уникальных экземпляров сущности.

Каждой сущности присваивается уникальное имя и номер, разделяемые косой чертой «/» и помещаемые над блоком.

Сущность в стандарте *IDEFIX* может быть независимой, или зависимой. Сущность является *независимой*, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями.

Сущность называется зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности (рис. 21).



Рис. 21. Изображение независимой и зависимой сущности

Если в рассмотренном при изучении метода нормализации данных примере факультеты принадлежат самому университету, то можно утверждать, что в ER-модели базы данных *Университет* сущность *Факультет* будет независимой сущностью. Сущность *Общая ведомость* является зависимой, так как экземпляр этой сущности не может быть определен без его отношения к экземпляру сущности *Студент* (рис.20).

Сущности могут иметь *внешние ключи (Foreign Key)*, которые могут использоваться в качестве части или целого первичного ключа или неключевого атрибута. Внешний ключ изображается с помощью помещения внутрь блока сущности имен атрибутов, после которых следуют буквы *FK* в скобках. Если внешний ключ входит в состав первичного ключа, то такая сущность является зависимой от той сущности, на которую ссылается внешний ключ.

В таблице 5 приведено обозначение и название связей стандарта *IDEFIX*.

Идентифицирующая связь между сущностью-родителем и сущностью-потомком изображается сплошной линией. Сущность-потомок в идентифицирующей связи является зависимой от идентификатора сущностью. Сущность-родитель в идентифицирующей связи может быть, как независимой, так и зависимой от идентификатора сущностью (это определяется ее связями с другими сущностями).

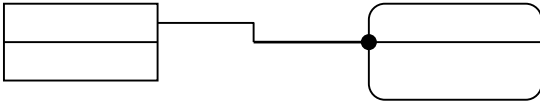
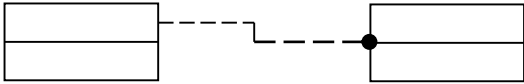
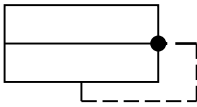
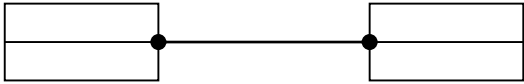
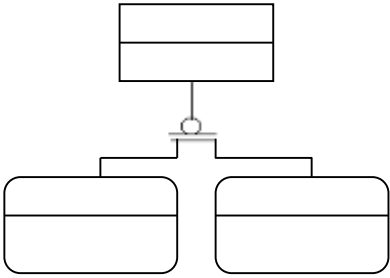
Пунктирная линия изображает *неидентифицирующую связь*. Сущность-потомок в неидентифицирующей связи будет независимой от идентификатора, если она не является также сущностью-потомком в какой-либо идентифицирующей связи.

Рекурсивная связь это неидентифицирующая связь между сущностью и ею же самой.

Идентифицирующая и неидентифицирующая связи имеют тип *один-ко-многим*. Тип связи **один-ко-многим** означает, что одному экземпляру





первой сущности соответствует 0, 1 или несколько экземпляров второй сущности. Первая сущность называется родительской, вторая – дочерней. Каждый экземпляр дочерней сущности связан с одним экземпляром родительской сущности. Примером связи типа один-ко-многим может служить связь между сущностями *Дисциплина* и *Общая ведомость*.

Таблица 5. Типы связей стандарта *IDEF1X*

Обозначение	Наименование
	Идентифицирующая связь
	Неидентифицирующая связь
	Рекурсивная связь
	Связь типа «многие-ко-многим»
	Связь супертипа с подтипами

Дисциплина является родительской сущностью, а *Общая ведомость* – дочерней. Для экземпляра сущности *Дисциплина* может не существовать экземпляров сущности *Общая ведомость* (никто еще не сдавал экзамен по этой дисциплине) или существует один или более экземпляров (экзамен по этой дисциплине сдавали). Однако для каждого экземпляра сущности *Общая ведомость* (результат сдачи экзамена студентом) родительская сущность обязательно содержит экземпляр сущности *Дисциплина*.

В стандарте IDEF1X приняты следующие обозначения кардинальности связи:

	Ноль, один или более
	Ноль или один
	Один или более
	Ноль, один или N (целое положительное число)

На рисунке 22 представлена связь между сущностями Дисциплина и Общая ведомость.

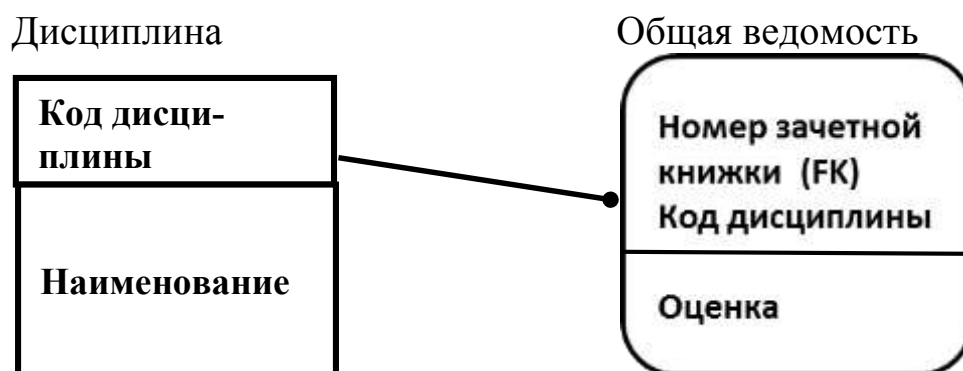


Рис. 22. Связь типа один-ко-многим

Пример связи типа «многие-ко-многим» между сущностями *Дисциплина* и *Группа* показан на рисунке 23. Экземпляру сущности *Дисциплина* соответствует несколько экземпляров сущности *Группа*, так как одну дисциплину могут изучать студенты разных групп. В то же время экземпляру сущности *Группа* может соответствовать несколько экземпляров сущности *Дисциплина*, поскольку студенты одной группы изучают несколько дисциплин.

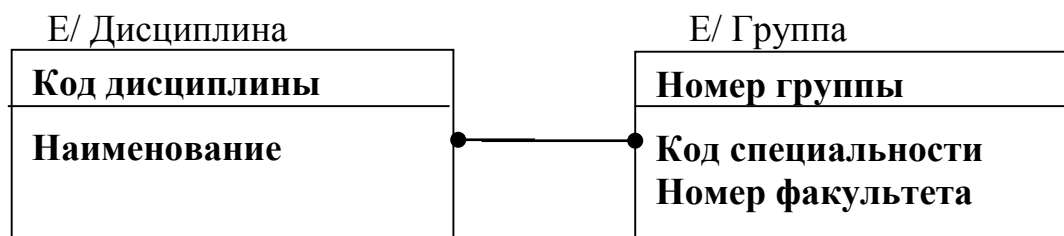


Рис. 23. Связь типа «многие-ко-многим»

Необходимо отметить, что связей типа «многие-ко-многим» следует избегать при создании БД. Эта связь порождает чрезмерное дублирование

данных, а, самое главное, существенно затрудняет обеспечение целостности по ссылкам. Стандартом определен подход к разрешению связи данного типа (рис.24).



Рис. 24. Разрешение связи типа «многие-ко-многим»

Разрешение связи типа многие–ко-многим осуществляется посредством введения ассоциативной сущности *Учебный план*, в ключевую область которой добавляют атрибуты исходных сущностей. Таким образом связь многие ко многим реализуется связями один-ко-многим между сущностями *Дисциплина* и *Учебный план* и между сущностями *Группа* и *Учебный план*.

Полностью ER-диаграмма модели базы данных Университет приведена на рисунке 25.

Средства метода *IDEFIX* в полном объеме реализуют семантическую модель данных. Между объектами *IDEFIX*-модели и объектами реляционной модели данных существует взаимосвязь, которая позволяет преобразовать *ER*-модель в реляционную.

Средства метода *IDEFIX* в полном объеме реализуют семантическую модель данных. Между объектами *IDEFIX*-модели и объектами реляционной модели данных существует взаимосвязь, которая позволяет преобразовать *ER*-модель в реляционную. Процесс преобразования осуществляется следующим образом:

- каждая сущность, не являющаяся подтипом и не имеющая подтипов становится таблицей. Имя сущности становится именем таблицы;
- каждый атрибут становится столбцом с тем же именем таблицы;
- атрибуты первичного ключа становятся первичным ключом таблицы;

- связи «многие-ко-многим» разрешаются посредством введения ассоциативной таблицы;
- индексы создаются для первичного ключа (уникальный индекс) и внешних ключей.



Рис. 25. ER-диаграмма модели базы данных Университет

Если в ER-модели присутствовали подтипы, то они либо сводятся в одну таблицу, либо для каждого подтипа создается отдельная таблица.

Ручное преобразование ER-модели в реляционную осуществляется на этапе *даталогического проектирования*. После выбора СУБД осуществляется описание логической структуры сформированных реляционных таблиц и правила ссылочной целостности для каждой связи которые должны быть установлены между таблицами. Правила ссылочной целостности (referential

integrity) – это логические конструкции, которые выражают правила использования взаимосвязанных данных при их добавлении, обновлении и удалении. При реализации базы данных для каждой связи будут приписаны триггеры ссылочной целостности (RI-триггеры), которые и обеспечат установленное правило целостности.

Для примера приведем даталогическую модель, описывающую часть ER-модели, приведенной на рис. 25. Таблицы 6-8 содержат описание логической структуры таблиц ФАКУЛЬТЕТ, ГРУППА и СТУДЕНТ, предполагаемых к реализации в базе данных Университет.

Таблица 6. Логическая структура таблицы ФАКУЛЬТЕТ

Поле		Признак ключа	Формат поля		
Имя	Наименование		Тип	Размер	Точность
Номер факультета	Номер факультета, в который входит группы студентов	*	Числовой	Целое	-
Наименование	Наименование факультета		Текстовый	50	-

Таблица 7. Логическая структура таблицы ГРУППА

Поле		Признак ключа	Формат поля		
Имя	Наименование		Тип	Размер	Точность
Номер группы	Номер группы, в которой обучается студент	*	Числовой	Целое	-
Код специальности	Номер специальности, на которой обучаются в группе		Числовой	Целое	-
Номер факультета	Номер факультета, в который входит группа		Числовой	Целое	-
Курс	Номер курса группы		Числовой	Целое	-

В соответствии с ER-моделью (рис. 25) для каждой связи между приведенными выше таблицами определим должна ли поддерживаться ссылочная целостность и, если она должна поддерживаться в базе данных, то сформулируем правила модификации связанных данных.

Таблица 8. Логическая структура таблицы СТУДЕНТ

Поле		Признак ключа	Формат поля		
Имя	Наименование		Тип	Размер	Точность
Номер зачетной книжки	Номер зачетной книжки студента	*	Текстовый	8	-
Номер группы	Номер группы, в которой обучается студент		Числовой	Целое	-
Фамилия	Фамилия студента		Текстовый	50	-
Дата рождения	Дата рождения студента		Дата/время	Кратк. формат даты	-
Коммерческий	Форма обучения		Логический	Да/Нет	-

Для связи ФАКУЛЬТЕТ-ГРУППА:

- необходима поддержка целостности связанных данных, которая будет гарантировать, что таблица ГРУППА может содержать данные только о тех группах, которые принадлежат факультетам, номера которых содержатся в столбце *Номер факультета* таблицы ФАКУЛЬТЕТ;

- из таблицы ФАКУЛЬТЕТ нельзя удалить данные по тому факультету которому принадлежат группы студентов (есть связанные строки в таблице ГРУППА);

- при изменении значения столбца *Номер факультета* в таблице ФАКУЛЬТЕТ выполнить каскадное изменение этого значения во всех строках таблицы ГРУППА.

Для связи ГРУППА-СТУДЕНТ:

- необходимо установить ссылочную целостность, обеспечивающую следующий контроль вводимых данных - в таблицу СТУДЕНТ можно будет вводить данные только о том студенте, номер группы которого уже содержится в значениях столбца *Номер группы* таблицы ГРУППА;

- запретить удаление из таблицы Группа той группы, сведения о студентах которой уже содержатся в таблице СТУДЕНТ;

- при изменении значения столбца *Номер группы* в таблице ГРУППА выполнять изменение этого значения во всех строках таблицы СТУДЕНТ.

4. ЯЗЫК TRANSACT-SQL

4.1. Введение в Transact -SQL

Transact-SQL структурированный язык запросов, являющийся инструментальным средством для описания, управления и манипулирования реляционными данными в СУБД Microsoft SQL Server. Первый структурированный язык запросов SEQUEL (Structured English Query Language) для доступа к реляционным данным был разработан фирмой IBM в 1974 году. В 1992 году был разработан первый международный стандарт SQL-92. Стандарт, действующий в настоящее время описывает все современные технологии работы с базами данных.

Основные понятия языка Transact-SQL

Идентификатор. Любой объект в базе данных должен быть идентифицирован. Для этого ему присваивают имя.

Существуют правила задания идентификатора:

1. Длина идентификатора от 1 до 128 символов.
2. Идентификатор может включать любые символы, определенные стандартом Unicode Standard 2.0 кроме запрещенных символов.
3. В идентификатор не должны входить слова, являющиеся зарезервированными.
4. Первым символом в идентификаторе может быть: буква алфавита или символ подчеркивания «_».
5. Для обозначения временных объектов (переменных и параметров) разрешается использовать в качестве первого символа «@».
6. Transact-SQL не различает регистр. Например, идентификатор *Фамилия* соответствует идентификатору *фамилия*.

Идентификатор, заданный в соответствии с этими правилами называется **обычным идентификатором**. Кроме обычных идентификаторов можно задавать идентификаторы с разделителем.

Примеры обычного идентификатора: Студент, Группа, Курс, Оценка.

Идентификаторы с разделителем заключаются в квадратные скобки ([]) или в двойные кавычки (" ") и может при необходимости содержать недопустимый символ или зарезервированные слова.

Примеры идентификатора с разделителем: [Общая ведомость], [Номер группы], [Код специальности].

Константы и переменные

Константами называются постоянные величины, значения которых не могут быть изменены. Можно использовать константы разных типов:

- числовые – 10, 105.25
- символьные (строковые) – 'Иванов', 'Базы данных'
- десятично-шестнадцатеричные – 0x23657, 0xFF0A12.

Переменная – именованная область памяти определенного объема, которая может изменяться. Переменная используется для хранения и передачи данных и физически является последовательностью нескольких байт. То, как сервер будет воспринимать последовательность байт, принадлежащих переменной зависит от типа данных переменной. Тип переменной определяется при ее объявлении. Объявление переменной выполняется с помощью команды DECLARE, имеющей следующий синтаксис:

```
DECLARE @имя_переменной тип_данных [, ...]
```

С помощью одной команды может быть объявлено несколько переменных. После объявления значение переменной не определено.

Для присвоения значение переменной используются команды SET и SELECT, имеющие следующий синтаксис:

```
SET @имя_переменной=значение
```

```
SELECT @имя_переменной=значение
```

Командой SET можно присвоить значение только одной переменной:

```
SET @количество=28
```

```
SER @группа='1011'
```

Командой SELECT можно присвоить значения нескольким переменным:

```
SELECT @количество=28, @группа='1011'
```

С помощью команды SELECT можно присвоить значение переменной в теле запроса. Например:

```
DECLARE @количество_студентов int
```

```
SELECT @количество_студентов=COUNT (фамилия)
```

```
FROM студент WHERE номер_группы='1701'
```

Выражение - это совокупность операндов и операторов. Операндами в выражении могут быть:

- константы и переменные;
- функции - заранее созданные именованные программы, которые выполняют обработку данных и возвращают определенные значения;
- имена полей таблиц (столбцов). Пользователь указывает в выражении имя интересующего столбца, а сервер автоматически подставляет соответствующее значение;
- подзапрос – запрос, данные которого используются другим запросом. В результате работы подзапроса сервер создает временную таблицу с необходимой структурой, копирует в нее данные и использует полученный набор в выражении.

Выполняемые над операндами действия задаются с помощью операторов. Операторы делятся на следующие типы:

1. Арифметические операции (в порядке убывания приоритета выполнения): унарные (+) и (-), умножение (*) и деление (/), остаток от целочисленного деления (%), сложение (+) и вычитание (-).

2. Строковые операторы - +.

3. Операторы сравнения: равно (=), больше (>), меньше (<), меньше или равно (<= или !>), больше или равно (>= или !<), не равно (!= или <>).

4. Логические операторы NOT, AND и OR.

Логическими являются также операторы:

- BETWEEN - проверяет, лежит ли значение в указанном диапазоне;
- LIKE - проверяет, соответствует ли значение указанному шаблону;
- IN - проверяет, является ли значение в указанном списке;
- ALL - выполняет сравнение для набора данных. Если условие выполнено для всего набора данных, возвращает значение TRUE;

- ANY - выполняет сравнение для набора данных. Если условие выполнено хотя бы для одного элемента из набора данных, возвращает значение TRUE;

- EXIST - проверяет существование данных;

- SOME - выполняет сравнение для набора данных. Если условие выполнено хотя бы для одного элемента из набора данных, возвращает значение TRUE (аналог ANY).

Типы данных

Тип данных определяет диапазон значений, которые можно сохранить в переменной или в поле таблицы. SQL Server поддерживает встроенные (системные) типы данных и позволяет на их основе создавать множество пользовательских типов. Ниже рассмотрены некоторые из наиболее часто используемых встроенных типов.

Числовые типы:

1) *Целочисленные типы данных* (общее название integer):

- bigint - восьмибайтовое двоичное целое число со знаком. Старший бит числа отводится под знак, если он равен единице, то число отрицательное. Диапазон значений: от -2^{63} до $2^{63}-1$

- int - четырехбайтовое двоичное целое число со знаком. Диапазон значений: от -2^{31} до $2^{31}-1$

- smallint - двухбайтовое двоичное целое число со знаком. Диапазон значений: от -2^{15} до $2^{15}-1$

- tinyint - однобайтовое двоичное целое число без знака. Диапазон значений: только положительные числа в диапазоне от 0 до 255

2) *Нецелочисленные типы данных*

Подразделяются на два типа: *десятичные* (decimal) и *приблизительные* (approximate).

Десятичные данные хранятся в виде последовательности цифр. Для представления каждой цифры используется 4 бита. К десятичным относятся типы Decimal (p[,s]) и Numeric(p[,s]) - десятичные типы с фиксированным количеством знаков до и после запятой. p – общее количество цифр числа, а s – количество десятичных знаков после запятой ($s \leq p$).

Если s не указан, то дробная часть равна нулю. В обычном режиме сервер ограничивает общее количество цифр в числе ($p=28$).

Приблизительные типы используются для работы с данными, имеющими значения от очень малых величин до предельно больших. К ним относятся:

- Float [(n)] – представляет тип с плавающей запятой. Диапазон значений: $-1,79 \cdot 10^{308} / 1,79 \cdot 10^{308}$. n – определяет количество разрядов мантиисы. Если n имеет значение 1 / 24, то поддерживается точность 7 цифр (4б). Если $n \leq 53$, то поддерживается точность 15 (8б)

- Real – представляет тип с плавающей запятой. Для хранения типа real – 4б ($n=24$). Диапазон значений: $-3,40 \cdot 10^{38} / 3,40 \cdot 10^{38}$

Денежные типы данных

Используются для хранения данных о денежных суммах. Позволяет хранить после запятой четыре знака. К денежным типам относятся:

- Money - для данных этого типа отводится 8 байт. Диапазон значений: -922 337 203 685 477.5808 / +922 337 203 685 477.5808;

- Smallmoney - для данных этого типа отводится 4 байт. Диапазон значений: -214 748.3648 / +214 748.3648.

Тип даты и времени

Существуют встроенные типы, позволяющие хранить сведения о дате и о времени одновременно, или отдельно. С данными этого типа можно выполнять арифметические операции сложения и вычитания, а также сравнивать их. К временным типам относятся:

- Datetime - для представления данного типа используется 8 байт. Первые четыре байта хранят информацию о дате, а последние – о времени. Значение даты представляет собой смещение относительно базовой даты в днях. Диапазон дат – 1.01.1753 – 31.12.9999 года. Значение времени – это информация о количестве миллисекунд, прошедших после полуночи данного дня;

- Smalldatetime - для представления данных используется 4 байта: 2- дата, 2 – время. Диапазон значений: 1.01.1900 – 6.06.2079 г. с точностью до минуты;

- Date - для представления данных даты используется 3 байта. Диапазон значений: 1.01.001 – 31.12.9999 г;

- Time - для представления данных времени используется 3 байта.

Символьные и текстовые типы данных

Для хранения текстовой информации используются символьные и текстовые типы данных.

К *символьным типам* данных относятся:

- тип Char(n) – используется для хранения набора символов длиной n ($n_{\max} = 8000$). Строка данного типа является строкой фиксированной длины. Если строка содержит символов меньше, чем n, то в конец строки добавляются пробелы. Если символов больше n, то часть конечных символов будет потеряна.

- тип Varchar(n) – данный символьный тип является строкой переменной длины. Для строки будет выделяться память в соответствии с реальным размером строки. (Например, n=50, длина строки=20, тогда выделяется память для 20 символов).

- типы Nchar(n) и Nvarchar(n) аналогичны типам Char и Varchar, отличаются тем, что предназначены для хранения символов Unicode. То есть каждый символ занимает 2 байта, поэтому $n_{\max} = 4000$.

Если необходимо указать тип строки, как Unicode, то перед строкой следует добавить символ N. Например, N'Stroka in Unicode Standart'

К *текстовым типам* относятся:

- Text – предназначен для хранения очень большого количества символов (до $2^{31}-1$). Для данных типа text память выделяется страницами (8 Кбайт).

- Ntext – предназначен для хранения текста большого объема в формате Unicode.

Бинарные типы данных

Бинарные типы данных используются для хранения последовательности двоичных значений большой длины. В бинарных типах данных пользователь может хранить любые значения, начиная от текста и заканчивая исполняемым кодом программы. К бинарным типам относятся:

- тип Binary(n). Он позволяет хранить до 8000 байт.

Под хранимое значение будет выделяться n+4 байта (четыре байта используются для хранения значения n- описание длины).

- тип Varbinary(n). Этот тип аналогичен binary, отличается тем, что для хранения значения выделяется ровно столько байт, сколько ввел пользователь плюс 4 байта на описание длины.

- тип Image. Позволяет хранить битовые значения длиной до $2^{31}-1$ (2 147 483 647). Память выделяется целыми страницами.

Кроме рассмотренных типов данных используемых для представления каких-то определенных значений (дата, текст, числа и т.д.), в SQL-Server имеются специальные типы данных, используемые в основном для внутренних нужд.

Преобразование типов данных. В SQL-Server имеется две функции, позволяющие преобразовывать значения одного типа в значения любого другого типа (если такое преобразование возможно).

Формат функций:

- CAST (*значение AS тип_данных*)
- CONVERT(*тип_данных[(длина)], значение*)

Аргумент *значение* задает величину, которую необходимо преобразовать.

Аргумент *тип_данных* определяет новый тип данных.

Управляющие конструкции Transact-SQL. Управляющие конструкции языка предназначены для группировки команд, организации ветвлений и циклов.

BEGIN ... END

Выполняет объединение команд в единый блок. Объединенные в блок команды воспринимаются интерпретатором как одна команда.

Оператор IF

Условный оператор позволяет выполнять указанную команду или блок команд только при соблюдении указанного логического условия.

Синтаксис оператора:

```
IF логическое_условие
    блок_команд_1
[ ELSE
    блок_команд_2 ]
```

Если логическое условие возвращает значение TRUE, выполняется первый блок команд, в противном случае (значение FALSE) – второй блок команд. Если при несоблюдении условия не требуется выполнения действий, то ключевое слово ELSE можно не указывать.

Оператор CASE

Оператор выбора для реализации разветвления по многим направлениям. Синтаксис оператора:

```
CASE
    WHEN логическое_условие
        THEN блок_команд
    [... n]
    [ ELSE блок_команд ]
END
```

В операторе может быть указано множество блоков WHEN...THEN, о чем свидетельствует параметр [...n]. Если логическое условие блока WHEN...THEN возвращает значение TRUE, то выполняется указанный блок команд и оператор CASE завершает свою работу. Если логическое

условие возвращает значение FALSE, то проверяется логическое условие следующего блока. Если ни одно из указанных условий не выполняется то после ключевого слова ELSE можно указать выполняемый в этом случае блок команд.

Операторы WHILE, BREAK, CONTINUE

Оператора WHILE позволяет организовать цикл. Формат оператора:

WHILE *логическое_условие*

блок_команд

[BREAK]

блок_команд

[CONTINUE]

блок_команд

Оператор WHILE осуществляет циклическое выполнение блока команд, если указанное логическое условие возвращает значение TRUE. С помощью оператора BREAK можно принудительно остановить работу и выйти из цикла. Оператор CONTINUE позволяет начать цикл заново, не дожидаясь выполнения всех команд цикла

Стандартом SQL все команды языка определены в разделы:

Data Definition Language (DDL) - язык определения данных содержит команды создания, изменения и удаления объектов базы данных;

Data Manipulation Language (DML) - язык манипулирования данными содержит команды извлечения и изменения данных, хранящихся в таблицах;

Data Control Language (DCL) - язык управления данными включает в себя команды предоставления пользователю привилегий доступа или их отмены.

4.2. Язык описания данных (DDL)

Язык описания данных содержит команды:

- CREATE объект - создание объекта;
- ALTER объект - изменение объекта;
- DROP объект - удаление объекта;

Синтаксис команд определяется объектом, для которого выполняется действие. Объектами являются база данных, таблица, представление, индекс, представление, хранимая процедура, пользовательская функция, триггер.

Создание базы данных выполняется командой CREATE DATABASE, имеющей следующий синтаксис:

CREATE DATABASE *имя_БД*

ON [PRIMARY] *имя_файла* [, ...n]]

```
[ , имя_файловой_группы [ , ...n ] ]
[ LOG ON имя_файла_журнала_транзакций [ , ...n ] ]
[ COLLATE имя_сопоставления ]
```

При создании базы данных обязательным является лишь указание ее имени, все остальные параметры могут отсутствовать¹. Параметры базы данных имеют следующее назначение:

Имя_БД - имя, которое будет присвоено создаваемой базе данных.

PRIMARY - это ключевое слово свидетельствует, что описываемый далее файл является первичным файлом базы данных. Только один файл в базе данных может быть первичным. Если первичный файл не определен явно, то в этом качестве будет использоваться первый файл, указанный в конструкции имя_файла.

LOG ON означает, что файлы журнала транзакций будут определены явно. После слов LOG ON должно следовать имя файла журнала транзакций. Если параметр отсутствует, то сервер автоматически создает единственный файл размером 25 % от суммарного размера файлов данных, но не менее 512 Кбайт. Имя файла генерируется на основе имени базы данных, но в конце к нему добавляются символы «_Log».

COLLATE - параметр указывается сопоставление, которое будет использоваться в качестве сопоставления по умолчанию для всех объектов, создаваемых в базе данных.

Для файла базы можно явно задать значения следующим параметрам:

NAME - определяет логическое имя файла, под которым он будет опознаваться при выполнении различных команд Transact-SQL;

FILENAME предназначен для задания полного пути и названия соответствующего физического файла, который будет создан на жестком диске;

SIZE предназначен для указания первоначального размера, который будет иметь соответствующий файл. Минимальный размер файла составляет 512 Кбайт;

FILEGROWTH определяет шаг прироста файла базы данных. Шаг прироста может задаваться в виде конкретного количества мегабайт (Mb) или килобайт (Kb) либо в виде процента (%) от первоначального размера файла. Если параметр опущен, то файл будет увеличиваться на 10 % от первоначального объема. Минимальный размер, на который может быть увеличен файл, составляет 64 Кбайт;

MAXSIZE = { max_size | UNLIMITED }². - Параметр позволяет ограничивать размер файла или указать UNLIMITED - размер файла не

¹ Необязательные параметры при описании синтаксиса команды заключаются в квадратные скобки.

² Параметры команды перечисленные через разделитель «|» и заключенные в фигурные скобки указывают на выбор одного из них в команде

ограничивается, он будет расти, пока не заполнит все доступное пространство на диске.

Пример создания базы данных *univer*:

```
CREATE DATABASE [univer]
ON (NAME = N'univer_Data',
    FILENAME = N'C:\Program Files\Microsoft SQL
    Server\MSSQL\Data\univer_Data.MDF', SIZE = 2,
    FILEGROWTH = 10%)
LOG ON (NAME = N'univer_Log',
    FILENAME = N'C:\Program Files\Microsoft SQL
    Server\MSSQL\Data\univer_Log.LDF', SIZE = 1, FI-
    LEGROWTH = 10%)
```

К управлению базой данных на физическом уровне относится вся работа по изменению имен, размеров, количества, положения файлов базы данных, усечение базы данных и журнала транзакций, создание групп файлов, изменение группы файлов по умолчанию, изменение имени и владельца базы данных, присоединение и отсоединение баз данных, изменение параметров базы данных.

Подключение к базе данных

В общем случае при доступе к объекту базы данных на зарегистрированном SQL сервере следует задавать параметр команды доступа в формате `имя_сервера. [имя_базы_данных]. [имя_схемы]. имя_объекта`

Для смены контекста базы данных используется следующая команда:

```
USE database_name
```

После переключения на контекст указанной базы данных к ее объектам можно обращаться, не указывая имя базы данных.

Создание таблицы

Создание новой таблицы выполняется командой `CREATE TABLE`.

Синтаксис команда:

```
CREATE TABLE имя_таблицы
({ описание_столбца
  | имя_вычисляемого_столбца AS выражение [PERSISTED]
  | ограничения_целостности_уровня_таблицы} [, ...])
```

где:

имя_таблицы - имя создаваемой таблицы;

имя_вычисляемого_столбца. - представляет собой имя столбца, физически не хранящийся в таблице если для него не установлен параметр `PERSISTED`. Значение столбца вычисляется на основе выражения, использующего другие столбцы той же таблицы.

Описание столбца таблицы выполняется следующим образом:

```
имя_столбца тип_данных [ (размер) ]
```

```
[ {DEFAULT значение_по_умолчанию |
  IDENTITY [ (значение, шаг) ] } ] [ {NULL|NOT NULL} ]
[ {PRIMARY KEY | UNIQUE } [ {CLUSTERED|NOCLUSTERED} ] ]
[ FOREIGN KEY REFERENCES имя_таблицы(имя_столбца)
  [ON UPDATE {NO ACTION|CASCADE|SET NULL|SET DEFAULT} ]
  [ON DELETE {NO ACTION|CASCADE|SET NULL|SET DEFAULT} ] ]
[ CHECK (логическое выражение) ] ]
```

Параметры базы данных имеют следующее назначение:

IDENTITY указывает, что новый столбец является столбцом для которого автоматически формируется уникальное последовательное значение. Если значение и шаг приращения не заданы, то по умолчанию они принимаются равными 1. Для каждой таблицы можно создать только один такой столбец.

NULL|NOT NULL определяет, допустимы ли для столбца значения **NULL**. Константа **NULL** в базах данных обозначает неопределенное значение. Ограничение **NOT NULL** для столбца делает его обязательным для заполнения.

PRIMARY KEY — указывает на то, что столбец является первичным ключом таблицы.

UNIQUE - ограничение, которое обеспечивает сущностную целостность для указанного столбца. Таблица может содержать несколько ограничений уникальности.

CLUSTERED | NONCLUSTERED - указывает, что для ограничения **PRIMARY KEY** или **UNIQUE** создается кластеризованный или некластеризованный индекс.

FOREIGN KEY - ограничение, которое обеспечивает ссылочную целостность данных в этом столбце. Ограничения **FOREIGN KEY** требуют, чтобы каждое значение в столбце существовало в соответствующем связанном столбце в связанной таблице. Ограничения **FOREIGN KEY** могут ссылаться только на столбцы, являющиеся ограничениями **PRIMARY KEY** или **UNIQUE** в связанной таблице. Этот столбец связанной таблице должен быть определен на типе данных столбца, являющегося внешним ключом. Если столбец определен внешним ключом, то для него должны быть заданы правила изменения связанных данных:

ON UPDATE указывает, какое действие совершается над строками в изменяемой таблице, когда у этих строк есть ссылочная связь и строка родительской таблицы, на которую указывает ссылка, обновляется:

— **NO ACTION** - обновление строки родительской таблицы не выполняется, если есть строки, ссылающиеся на обновляемую строку.

– CASCADE - строки обновляются в ссылающейся таблице, если эта строка обновляется в родительской таблице.

– SET NULL – означает, что всем значениям, составляющим внешний ключ, присваивается значение NULL, когда обновляется соответствующая строка в родительской таблице.

– SET DEFAULT означает, что всем значениям, составляющим внешний ключ, присваивается их значение по умолчанию, когда обновляется соответствующая строка в родительской таблице.

ON DELETE определяет операцию, которая производится над строками создаваемой таблицы, если эти строки имеют ссылочную связь, а строка, на которую имеются ссылки, удаляется из родительской таблицы:

– NO ACTION - выполняется откат операции удаления строки из родительской таблицы.

– CASCADE - если из родительской таблицы удаляется строка, соответствующие ей строки удаляются из ссылающейся таблицы.

– SET NULL - все значения, составляющие внешний ключ, при удалении соответствующей строки родительской таблицы устанавливаются в NULL.

– SET DEFAULT - все значения, составляющие внешний ключ, при удалении соответствующей строки родительской таблицы устанавливаются в значение по умолчанию.

CHECK - ограничение, обеспечивающее доменную целостность путем ограничения возможных значений, которые могут быть введены в столбец или столбцы.

В качестве примера приведем запросы для создания таблиц, составленные на основе рассмотренной даталогической модели (см. стр.).

Первой должна создаваться таблица ФАКУЛЬТЕТ, так как она не имеет внешних ключей и, следовательно, не ссылается на строки других таблиц. Запрос на создание таблицы ФАКУЛЬТЕТ:

```
CREATE TABLE [факультет]
([номер факультета] tinyint PRIMARY KEY,
наименование] char(50))
```

Таблица ГРУППА ссылается только на строки таблицы ФАКУЛЬТЕТ по значению столбца *Номер факультета*, поэтому ее можно создавать, когда таблица ФАКУЛЬТЕТ уже существует. Запрос на создание имеет вид:

```
CREATE TABLE [группа]
( [номер группы] smallint PRIMARY KEY,
[код специальности] int,
[номер факультета] tinyint FOREIGN KEY
REFERENCES факультет([номер факультета])
```

```
ON DELETE NO ACTION ON UPDATE CASCADE,
[курс] tinyint
check([курс] between 1 and 5))
```

Обратим внимание на то, что столбец *Номер факультета*, являющийся внешним ключом, определен также, как и соответствующий ему столбец в таблице ФАКУЛЬТЕТ. В запросе при описания внешнего ключа учтены правила изменения связанных данных, описанные в даталогической модели.

В запросе приведен пример использования ограничения check для столбца *курс*.

После выполнения запроса таблица ГРУППА будет создана. Теперь можно создавать таблицу СТУДЕНТ, в которой столбец *Номер группы* является внешним ключом, ссылающимся на ГРУППУ:

```
CREATE TABLE [студент]
( [номер зачетной книжки] char(8) PRIMARY KEY ,
  [номер группы] smallint NOT NULL FOREIGN KEY
  REFERENCES группа([номер группы])
  ON DELETE NO ACTION ON UPDATE CASCADE,
  [фамилия] char(50) NOT NULL ,
  [дата рождения] date NOT NULL ,
  [коммерческий] bit NOT NULL)
```

Столбец *номер группы* является внешним ключом, ссылающимся на соответствующий столбец таблицы ГРУППА, поэтому они определены одинаково. Учтены также и определенные в даталогической модели правила изменения связанных данных.

С помощью команды ALTER TABLE можно выполнять изменение свойств существующих столбцов, удалять их или добавлять в таблицу новые столбцы, управлять ограничениями целостности, как на уровне столбцов, так и на уровне таблицы. Удаление таблицы выполняется при помощи команды DROP TABLE. При удалении таблицы следует учитывать связи, установленные в базе данных между таблицами. Если на удаляемую таблицу с помощью ограничения целостности FOREIGN KEY ссылается другая таблица, то SQL сервер не разрешит ее удаление.

4.3. Язык манипулирования данными (DML)

Команды языка манипулирования данными позволяют из базы извлекать и изменять данные.

Извлечение данных выполняет команда SELECT. При выполнении этой команды осуществляется поиск указанной таблицы или таблиц, извлекаются заданные столбцы, выделяются строки, соответствующие условию отбора, строки сортируются или группируются и формируется результирующий набор данных. Синтаксис команды:

```

SELECT [предикат] список_столбцов
FROM имена_таблиц
    [WHERE критерий_отбора]
    [GROUP BY критерий_группировки ]
    [HAVING критерий_отбора]
    [ORDER BY критерий_сортировки]

```

Команда содержит шесть разделов, из которых только два являются обязательными:

- раздел SELECT определяет структуру результирующего набора данных;
- раздел FROM указывает откуда извлекать данные.

Параметр *предикат* в разделе SELECT определяет какие строки из результирующего набора данных выводить. Если указан предикат ALL, то все строки полученные по запросу, DISTINCT – строки, содержащие неповторяющиеся значения в столбце.

В списке столбцов можно указать:

- 1) * - что означает выбрать данные из всех столбцов таблицы;

Пример: Вывести полные сведения о всех студентах:

```
SELECT * FROM студент
```

- 2) [таблица.]столбец или [представление.]столбец или [псевдоним.]столбец – выбрать значение из указанного столбца таблицы или представления, или объекта, имеющего указанный псевдоним. Псевдонимом является другое имя таблицы или представления, которое может быть использовано в дальнейшем для ссылки на них.

- 3) выражение [AS псевдоним] – вычисляемый столбец.

Пример: Вывести сведения о всех группах университета.

```
SELECT [номер факультета],[номер группы],
        [курса] as [номер курса]
```

```
FROM группа
```

В примере для столбца задан псевдоним и теперь он будет использоваться в качестве заголовка столбца результирующей таблицы или подписи к значению столбца.

Раздел FROM определяет имена одной или нескольких таблиц или представлений, которые содержат отбираемые данные. Синтаксис раздела: FROM таблица [AS псевдоним] | представление [AS псевдоним] [, ...]

Имена таблиц в разделе разделяются запятой.

Пример: Вывести следующие сведения о студентах: номер факультета, номер группы, фамилия.

```
SELECT [номер факультета],
студент.[номер группы],фамилия
FROM студент, группа
```

Если в разделе FROM имена таблиц перечислены через запятую, то результатом команды будет таблица, строки которой получены прямым произведением исходных таблиц. В нашем примере для каждого студента будет добавлено столько строк, сколько строк в таблице ГРУППА.

Команда SELECT должна содержать раздел WHERE, если требуется извлечь данные, значение которых соответствует некоторому правилу отбора. Критерий отбора определяет логическое условие включения строки в результат запроса.

В стандарте ANSY/ISO определены следующие условия отбора (предикаты):

Предикат сравнения – проверяемое значение сравнивается со значением выражения.

Пример: Вывести сведения о студентах группы с номером 1701.

```
SELECT * FROM Студент
WHERE [Номер группы]=3591
```

Критерий отбора может содержать несколько условий, связанных логическими операторами, такими как NOT, AND и OR.

Пример: Вывести сведения о тех, кто не сдал экзамены по дисциплинам с кодами 11 и 12.

```
SELECT * FROM [общая ведомость]
WHERE ([код дисциплины]=11
OR [код дисциплины]=12) and оценка=2
```

Предикат «между» - проверяется, попадает ли значение в указанный диапазон.

Пример: Вывести сведения о факультетах, имеющих номера с 1 до 5.

```
SELECT * FROM Факультет
WHERE [Номер факультета] BETWEEN 1 AND 5
```

Предикат «в» - проверяется, совпадает ли значение с одним из значений заданного множества

Пример: Вывести сведения о студентах, родившихся зимой.

```
SELECT [номер группы], фамилия, [дата рождения]
FROM студент
WHERE MONTH([дата рождения]) IN (12, 1, 2)
```

Предикат «как» - проверяется, соответствует ли строковое значение заданному шаблону.

Пример: Вывести сведения о студентах, фамилия которых начинается с буквы «А».

```
SELECT * FROM студент
WHERE фамилия LIKE 'А%'
```

Знак % в шаблоне означает что с этой позиции можно подставить любое количество произвольных символов.

«**NULL – предикат**» - проверяется, не содержится ли в столбце значение NULL. При проверке на NULL возвращается значение TRUE, если проверяемое значение не определено, в противном случае - FALSE.

Пример: Вывести сведения о группах, для которых курс не определен.

```
SELECT * FROM Группа
WHERE курс IS NULL
```

«**Предикат существования**» - проверяется, получены ли в результате работы подзапроса одна или несколько строк. Предикат реализуется использованием оператора EXISTS, который проверяет существование данных.

Извлекаемые с помощью SELECT данные можно обрабатывать с целью получения итоговых значений. Раздел GROUP BY позволяет выполнять группировку строк таблиц по определенным критериям для выполнения вычислений над значениями строк таблицы попавшими в группу. Синтаксис раздела:

```
GROUP BY критерий_группировки
[HAVING критерий_отбора]
```

Критерий_группировки – это имя одного, или нескольких столбцов, которые называются группирующими столбцами. Они определяют, по какому признаку строки делятся на группы. Столбцы, значения которых участвуют в получении итогов называются группируемыми. Если в команде SELECT указан раздел GROUP BY, то в списке столбцов для одного или нескольких столбцов должна быть указана функция агрегирования.

Функции агрегирования позволяют выполнять статистическую обработку данных, к ним относятся:

AVG ({выражение | [DISTINCT] имя_столбца}) – вычисляет среднее значение для набора данных.

COUNT ({[DISTINCT] имя_столбца | *}) – подсчитывает количество непустых значений для указанного столбца в строках группы.

SUM([DISTINCT] имя_столбца) – выполняет суммирование всех значений группы в указанном столбце.

MAX (выражение) - возвращает максимальное значение в указанном столбце (выражении).

MIN (выражение) - возвращает минимальное значение в указанном столбце (выражении).

Пример: Определить количество студентов в каждой группе.

```
SELECT [номер группы] , COUNT (фамилия)
FROM студент
GROUP BY [номер группы]
```

Пример: Определить сколько пятерок получили студенты в сессию по каждой дисциплине.

```
SELECT [код дисциплины], COUNT(оценка)
FROM [общая ведомость]
WHERE оценка=5
GROUP BY [код дисциплины]
```

Совместно с разделом GROUP BY команда SELECT может содержать раздел HAVING. Критерий отбора этого раздела позволяет ограничивать количество возвращаемых запросом агрегаций. Критерий отбора задается логическим условием, определяющим какие итоговые строки, полученные в результате группировки данных включать в результирующий набор.

Пример: Определить группы, в которых меньше 15 человек.

```
SELECT [номер группы],
COUNT(фамилия) as [количество]
FROM студент
GROUP BY [номер группы]
HAVING COUNT(фамилия) < 15
```

Раздел ORDER BY команды SELECT используется, если необходимо упорядочить данные, полученные по запросу. Синтаксис раздела:

```
ORDER BY критерий_столбца [{ASC|DESC}] [, ...]
```

Критерий столбца задается именем столбца, по значениям которого будет упорядочен результирующий набор данных. С помощью ключевых слов ASC и DESC можно указать порядок сортировки (ASC – по возрастанию значений, DESC – по убыванию значений). По умолчанию выполняется сортировка по возрастанию значений.

Пример: Вывести сведения о студентах родившихся в 1999 - 2002 годах, отсортировать полученные данные по возрастанию номеров групп, а внутри групп по убыванию даты рождения студентов.

```
SELECT * FROM студент
WHERE YEAR([дата рождения]) BETWEEN 1999 AND 2002
ORDER BY [номер группы], [дата рождения] DESC
```

Извлекать данные можно сразу же из нескольких связанных таблиц, в этом случае раздел From имеет следующий синтаксис:

```
FROM таблица_1 INNER JOIN таблица_2
ON таблица_1.поле_1 оператор таблица_2.поле_2
```

INNER JOIN указывает выполнять внутреннее соединение таблиц (экви-соединение).

Пример. Для каждого наименования факультета подсчитать количество студентов. Для того, чтобы посчитать сколько студентов на каждом факультете нужно получить таблицу соединением таблиц СТУДЕНТ и ГРУППА и затем то что получилось соединить с таблицей ФАКУЛЬТЕТ, так как в результирующей таблице нужно иметь столбец наименование факультета. В примере запроса таблицам присвоены псевдонимы.

```
SELECT [наименование], COUNT(фамилия)
FROM студент s INNER JOIN группа g ON
```

```
s.[номер группы]=g.[номер группы])
INNER JOIN факультет f ON
g.[номер факультета]=f.[номер факультета]
GROUP BY [наименование]
```

Язык DML кроме команды извлечения данных SELECT содержит команды добавления данных INSERT, обновления данных UPDATE и удаления данных DELETE.

Команда INSERT INTO позволяет вставить одну или несколько строк в уже существующую таблицу. Синтаксис команды:

```
INSERT INTO таблица [(список_столбцов)]
{VALUES (список_значений) | запрос }
```

В списке столбцов указываются имена тех столбцов, в которые команда будет добавлять значения. Если список столбцов отсутствует, то вставка значений выполняется во все столбцы таблицы. Вставляемые в таблицу значения могут быть переданы следующими способами:

1) Списком значений VALUES, количество значений в котором будет определяется списком столбцов. Если список столбцов отсутствует, то список значений должен содержать значения для всех столбцов таблицы.

2) Значениями, полученными в результате выполнения запроса.

В любом случае типы данных передаваемых значений должны соответствовать типам данных тех столбцов, в которые они добавляются.

Пример: Добавить строку в таблицу ФАКУЛЬТЕТ.

```
INSERT INTO факультет ([номер факультета],
[наименование])
VALUES (14, 'Факультет управления')
```

Команды UPDATE позволяет обновлять данные одного или нескольких столбцов таблицы в одной или нескольких строках. Синтаксис команды:

```
UPDATE таблица
SET имя_столбца=новое_значение[,...]
WHERE критерий_поиска
```

SET определяет изменения, которые необходимо выполнить для одного, нескольких, или всех столбцов. Новое значение может быть задано выражением. Раздел WHERE задает условие отбора тех строк, которые нужно изменить.

Пример: Создать запрос на перевод студента Иванова из группы 1701 в группу 1702.

```
UPDATE студент SET [номер группы]=1702
WHERE [номер группы]=1701 AND [фамилия]='Иванов'
```

Удаление данных из таблицы выполняется командой DELETE. Синтаксис команды:

```
DELETE таблица
WHERE критерий_поиска
```

Удаление данных выполняется построчно, за одну операцию можно удалить как одну, так и нескольких строк. Если раздел WHERE отсутствует, команда удалит все строки таблицы.

Пример: Удалить сведения о студенте с фамилией Иванов.

```
DELETE студент WHERE фамилия='Иванов'
```

Использование подзапросов в командах языка DML

Подзапрос представляет собой запрос, содержащийся в предложении WHERE, HAVING или FROM другого (основного) запроса. Основной запрос формирует результат в соответствии с возвращаемым подзапросом значением.

Пример: Сформировать список групп, объединяющих студентов факультета управления.

```
SELECT [Номер группы] FROM Группа
WHERE [Номер факультета] =
    (SELECT [Номер факультета] FROM Факультет
     WHERE Наименование = 'Факультет управления')
```

Подзапрос определяет номер нужного факультета, и это значение используется предикатом сравнения для выбора нужных строк таблицы ГРУППА.

Пример: Вывести сведения о тех студентах, которые в сессию не сдавали экзамены.

```
SELECT * FROM студент
WHERE not [номер зачетной книжки] IN
    (SELECT DISTINCT [номер зачетной книжки]
     FROM [общая ведомость])
```

В этом примере подзапрос возвращает несколько значений одного столбца, которые используются в предикате «в» для выборки нужных строк из таблицы СТУДЕНТ.

Пример: Удалить из таблицы студентов пятого курса.

```
DELETE студент
WHERE [номер группы] IN
    (SELECT [номер группы] FROM группа
     WHERE курс=5)
```

Подзапрос возвращает значения столбца номер группы тех групп, студенты которых числятся на пятом курсе. Список возвращаемых подзапросом значений определяет какие строки таблицы СТУДЕНТ нужно удалить.

4.4. Средства разработки процедур в Transact-SQL

Для многократного использования одной или нескольких команд целесообразно сохранить их на SQL сервере в виде именованных объектов: представлений, процедур и функций.

Представление (views) – это виртуальная таблица, содержимое которой определяется запросом на извлечение данных. Команда создания представления имеет следующий синтаксис:

```
CREATE VIEW имя_представления [(имя_столбца [, ...]) ]
AS команда_select
```

Таблица, создаваемая представлением наследуют столбцы исходных таблиц, если они явно не были заданы при создании представления.

Пример. Создать запрос на создание представления для получения списка студентов с указанием группы и курса.

```
CREATE VIEW [список студентов]
AS
SELECT s.[номер группы], [фамилия],
[номер зачетной книжки], курс
FROM студент s INNER JOIN группа g
ON s.[номер группы] = g.[номер группы]
```

После выполнения запроса в базе данных появляется представление с именем [список студентов] и теперь, вместо повторного соединения таблиц СТУДЕНТ и ГРУППА, можно для получения нужных данных использовать представление. Если требуется сформировать список студентов первого курса достаточно выполнить запрос:

```
SELECT * FROM [список студентов]
WHERE курс = 1
```

Пользовательская хранимая процедура (stored procedure) — это именованный набор команд Transact-SQL, хранящийся непосредственно на сервере и представляющий собой самостоятельный объект базы данных. Использование хранимых процедур позволяет реализовать бизнес-логику на стороне сервера. Клиент осуществляет вызов хранимой процедуры по ее имени, SQL сервер выполняет команды процедуры и возвращает клиенту результат. Команда создания процедуры имеет следующий синтаксис:

```
CREATE PROCEDURE имя_процедуры
@переменная тип_данных [, ...]
AS
Команды Transact-SQL
```

Хранимая процедура может иметь параметры, с помощью которых в процедуру передаются нужные для ее работы данные. В этом случае в команде создания процедуры нужно определить переменные, являющиеся параметрами процедуры.

Вызов процедуры осуществляется командой:

```
EXEC имя_процедуры [параметры]
```

Пример. Создать запрос на создание процедуры для получения результатов сдачи экзамена студентами группы с параметрами наименование дисциплины и номер группы.

```
CREATE PROCEDURE [результаты экзамена группы]
```

```

        @disname varchar(40),
        @groupnumber smallint
AS
SELECT s.[номер группы], [фамилия],
[наименование дисциплины], [оценка]
FROM [дисциплина] d INNER JOIN [общая ведомость] v
ON d.[код дисциплины] = v.[код дисциплины]
INNER JOIN [студент] s ON
v.[номер зачетной книжки] = s.[номер зачетной книжки]
WHERE [наименование дисциплины] = @disname AND
s.[номер группы] = @groupnumber

```

Пользовательская хранимая процедура с параметрами может быть вызвана следующими способами:

- с указанием имен параметров и значений, передаваемых процедуре:

```

EXEC [результаты экзамена группы]
    @disname='Базы данных', @groupnumber='1701'

```

- указанием только передаваемых процедуре значений:

```

EXEC [результаты экзамена группы] 'Базы данных',
    '1701'

```

Триггер – это специальный тип хранимой процедуры, которая запускается автоматически при изменении какой-либо таблицы одной из команд языка DML: UPDATE, INSERT или DELETE. Эти операции по отношению к триггеру называют событиями модификации данных. В отличие от других типов хранимых процедур триггеры запускаются автоматически при возникновении указанных событий модификации.

Триггер создается с привязкой к одной таблице, но команды триггера могут выполнять доступ и к другим таблицам и объектам.

Когда пользователь пытается выполнить изменение данных таблицы триггера, то возникает событие модификации данных и сервер автоматически запускает триггер. Команда модификации и триггер выполняются одной транзакцией. В том случае, если триггер завершает свою работу успешно, изменение данных фиксируется в таблице. Если триггер не выполняется, то не выполнится и команда изменения данных.

5. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РЕЛЯЦИОННЫХ СУБД

5.1. Системы, ориентированные на анализ данных

Операционная (транзакционная) обработка в базах данных направлена на ввод новых данных, модификацию уже хранящихся и получение нужных данных. Имеющаяся в OLTP (Online Transactions Processing) системах возможность группирования и обобщения данных позволяет получать агрегаты, на основе которых можно выполнять оперативный анализ.

Система, обладающие средствами ввода, хранения и анализа данных с целью поиска решений называется СППР – *системой поддержки принятия решений* (DSS-Dicision Support System).

В ССПР выделяют три класса задач анализа:

- информационно-поисковый, анализ основывается на результатах поиска нужных данных с помощью заранее разработанных запросов;
- оперативно-аналитический – на основе аналитических данных, полученных в результате группирования и обобщения данных. Правила группирования данных могут меняться, поэтому не всегда возможно подготовить нужные запросы заранее;
- интеллектуальный анализ направлен на поиск функциональных и логических закономерностей в накопленных данных, построение моделей и правил, которые объясняют найденные закономерности и прогнозируют развитие анализируемых процессов.

OLTP–технология предназначена для обработки текущей оперативной информации определенного временного периода. Она характеризуется следующими свойствами:

- база данных хранит минимально необходимый объем данных;
- хранящиеся данные характеризуются большим количеством изменений;
- большое количество пользователей одновременно работают с базой данных;
- в результате при обработке параллельных транзакций выполняется большое количество операций чтения и записи, основанных на небольшом количестве строк.

Хранение в операционных базах данных ретроспективных данных, необходимых для решения аналитических задач, увеличивает объемы данных, требует дополнительной их обработки, что в конечном счете влияет на производительность системы в целом.

Технология оперативной аналитической обработки данных, использующая методы и средства для сбора, хранения и анализа многомерных

данных в целях поддержки принятия решения называется OLAP (online analytical processing - аналитическая обработка в реальном времени). OLAP-технология обработки данных заключается в подготовке суммарной (агрегированной) информации на основе больших массивов данных, структурированных по многомерному принципу. Отличительными свойствами OLAP-систем являются:

- периодически выполняемые операции чтения в результате запросов, основанных на большом количестве строк;
- небольшое количество пользователей;
- большой объем хранящихся данных;
- данные в OLAP системе статичны (как правило получены из OLTP систем).

В таблице 9 приведены данные сравнительного анализа OLTP- и OLAP –систем по ряду показателей.

Таблица 9. Сравнение OLTP и OLAP систем

Свойство	OLTP	OLAP
Назначение данных	Оперативный поиск и обработка	Прогнозирование, моделирование, анализ и выявление связей, анализ и выявление закономерностей
Уровень агрегации данных	Детальные данные	Агрегированные данные
Период хранения данных	Год	Десятки лет
Упорядоченность данных	По любому столбцу	По хронологии
Критерий эффективности работы	Количество транзакций в единицу времени	Скорость выполнения сложных запросов

В СППР для реализации подсистемы хранения используется концепция хранилищ данных.

Хранилище данных (ХД) представляет собой предметно-ориентированный, интегрированный, поддерживающий хронологию набор данных, организованный для целей поддержки принятия решений. Назначение ХД - информационное обеспечение компьютерной системы поддержки принятия решений по всем или основным видам деятельности организации. Свойства ХД:

Предметная ориентация позволяет хранить в ХД только те данные, которые необходимы для анализа. Т.е. данные должны представлять объекты (студент, дисциплина), а не процессы.

Интеграция - гарантирует, что данные отражающие один и тот же объект и полученные из разных источников должны быть приведены к одному формату.

Поддержка хронологии – все данные, хранящиеся в ХД должны соответствовать последовательным интервалам времени и быть ориентированы на многолетний и многомерный анализ данных, результаты которого могут быть использованы для принятия решений.

Неизменяемость – данные не обновляются, а пополняются за счет баз данных. После загрузки из БД данные в ХД только читаются

ХД содержит данные следующих категорий:

- **детальные данные** - данные, переносимые непосредственно из операционных баз данных. Соответствуют элементарным событиям, фиксируемым в OLTP-системах. Подразделяются на:

- измерения - наборы данных, необходимые для описания событий (например, дисциплина, группа, студент, семестр, курс);

- факты - данные, отражающие сущность события (например, оценка, количество баллов по дисциплине);

- **агрегированные (обобщенные) данные** - данные, получаемые на основании детальных путем суммирования по определенным измерениям (сумма баллов, количество студентов, средний балл);

- **метаданные** - данные о данных, содержащихся в ХД. Могут описывать: объекты предметной области, информация о которых содержится в ХД; категории пользователей, использующих данные в ХД; действия, выполняемые над данными и т.п.

Методом логического моделирования и визуализации хранилища данных является *многомерное моделирование*. Метод многомерного моделирования базируется на понятиях: факты, атрибуты, измерения, параметры (метрики), иерархия, гранулированность.

Факт - это набор связанных элементов данных, содержащих метрики и описательные данные. Каждый факт обычно представляет элемент данных, численно описывающий деятельность организации, бизнес-операцию или событие, которое может быть использовано для анализа деятельности организации или бизнес-процессов.

Атрибут - это описание характеристики реального объекта предметной области. Как правило, атрибут содержит заранее известное значение, характеризующее факт. Обычно атрибуты представляются текстовыми полями с дискретными значениями. Например, фамилия студента, группа.

Измерение - это интерпретация факта с некоторой точки зрения в реальном мире. Измерения, подобно атрибутам, содержат текстовые значения, которые сильно связаны по смыслу между собой. Измерения обычно представляют нечисловые, лингвистические переменные, такие как Студент, Семестр т.д.

Параметр, метрика или показатель - это числовая характеристика факта, который определяет эффективность деятельности или бизнес-действия организации с точки зрения измерения. Как правило, метрика содержит заранее не известное значение характеристики факта. Конкретные значения метрики описываются с помощью переменных. Например, метрикой является численное выражение количества баллов студента по дисциплине, средний балл студента в сессию и т.д. Метрика определяется с помощью комбинации элементов измерения и, таким образом, представляет факт.

Гранулированность - это уровень детализации данных, сохраняемых в ХД. Например, Количество баллов студента, Количество баллов студентов группы.

В OLAP-системах многомерная модель ХД представляется с помощью *гиперкуба* (см. рис.26), в ячейках которого находятся анализируемые данные.

Оси куба представляют собой измерения, по которым откладывают параметры, относящиеся к анализируемой предметной области. Измерения задаются перечислением своих элементов.

Например, измерение – параметр «семестр» может содержать следующие элементы: «1», «2», и т.д.

Элементы измерения могут находиться в отношении «родитель-потомок», что позволяет ввести на измерении одну или несколько иерархий. Например, иерархия на измерении «Студенты»: Группа и Студент.

На пересечении осей измерений располагаются данные, количественно характеризующие анализируемые факты – меры. Например, количество баллов, полученных студентом (группой) по дисциплине в семестр. Ячейка гиперкуба также может содержать и несколько значений. Над гиперкубом можно выполнять следующие операции:

Срез - выделение из гиперкуба тех данных, которые соответствуют фиксированному значению одного или нескольких элементов измерений. Из одного куба можно создать множество срезов. Например, срез – результаты баллов по всем дисциплинам всех студентов заданной группы в третьем семестре.

Вращение - изменение расположения измерений в пространстве для представления данных в удобном для анализа виде.

Консолидация или детализация - предназначены либо для агрегирования (обобщения) данных, либо для их детализации. Осуществление

этих операций возможно благодаря иерархии, установленной среди измерителей.

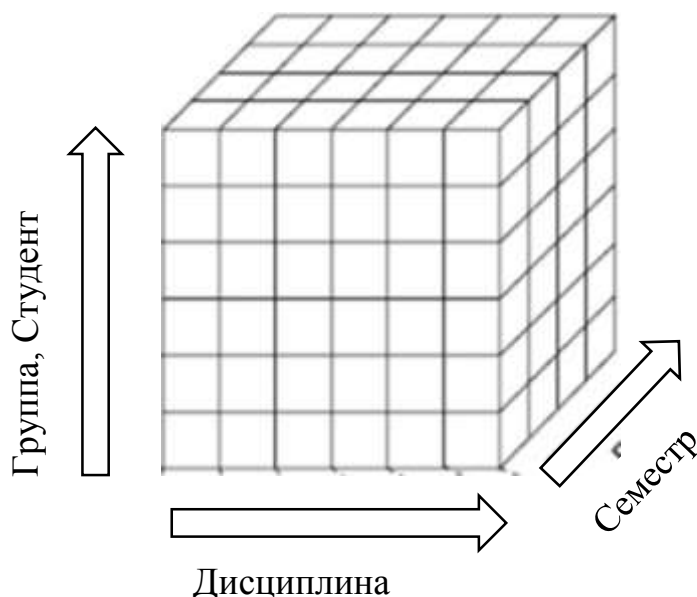


Рис. 26. Гиперкуб

Выбор способа хранения данных зависит от объема и структуры детальных данных, требований к скорости выполнения запросов и частоты обновления OLAP-кубов. Выделяют три основных способа хранения данных:

- MOLAP - для реализации многомерной модели используют многомерные БД;
- ROLAP - для реализации многомерной модели используют реляционные БД;
- HOLAP - для реализации многомерной модели используются многомерные и реляционные БД.

В случае MOLAP, исходные и агрегатные данные хранятся в многомерной БД или в многомерном кубе. Физически данные хранятся в "плоских" файлах, при этом куб представляется в виде одной плоской таблицы, в которую построчно вписываются все комбинации элементов всех измерений с соответствующими им значениями (см. таблицу 10).

Преимущества использования многомерных БД. Поиск и выборка данных осуществляется значительно быстрее, так как многомерная БД денормализована и содержит заранее агрегированные показатели, и не требуется дополнительных преобразований при переходе от множества связанных таблиц к многомерной модели.

Таблица 10. Представление куба в виде плоской таблицы данных

Измерения				Меры	
Специальность	Группа	Студент	Дисциплина	Баллы	Оценка
Инф. без.	ИБ-1601	Петров	Базы данных	85	5
Инф. без.	ИБ-1601	Кузьмин	Базы данных	78	4
Инф. без.	ИБ-1602	Лазарев	Базы данных	60	3
Инф. без.	ИБ-1601	Петров	Криптография	76	4

Недостатки MOLAP:

- за счет денормализации и предварительно выполненной агрегации объем данных в многомерной БД, как правило, увеличивается в десятки раз по отношению к объему исходных детализированных данных;

- многомерные базы данных содержат большой объем метаданных, что связано с организацией многомерного представления;

- многомерные БД имеют как правило сложную структуру, их сложно модифицировать. Например, при добавлении нового измерения приходится изменять структуру всей БД, что влечет за собой большие затраты времени.

MOLAP целесообразно использовать при относительно небольших объемах данных и стабильном наборе измерений.

В *ROLAP*-модели измерения и факты хранятся в отдельных таблицах реляционной базы данных. Многомерная модель данных реализуется средствами реляционных СУБД. Существует две основные схемы реализации многомерного представления данных с помощью реляционных таблиц: «звезда» и «снежинка».

Особенностью схемы «звезда» является расположение в центре схемы денормализованной таблицы фактов, которая может состоять из множества строк и содержит фактические или агрегатные данные необходимые при решении аналитических задач. С таблицей фактов связаны денормализованные таблицы измерений. Таблица фактов и таблицы измерений связаны идентифицирующими связями и первичные ключи таблиц измерений входят в таблицу фактов внешними ключами, которые в свою очередь являются частями составного первичного ключа таблицы фактов. Следовательно, каждая таблица измерения находится в отношении один-ко-многим с таблицей фактов.

При этом агрегированные данные хранятся совместно с исходными. Каждое измерение содержится в одной таблице.

На рисунке 27 приведен пример схемы «Звезда».

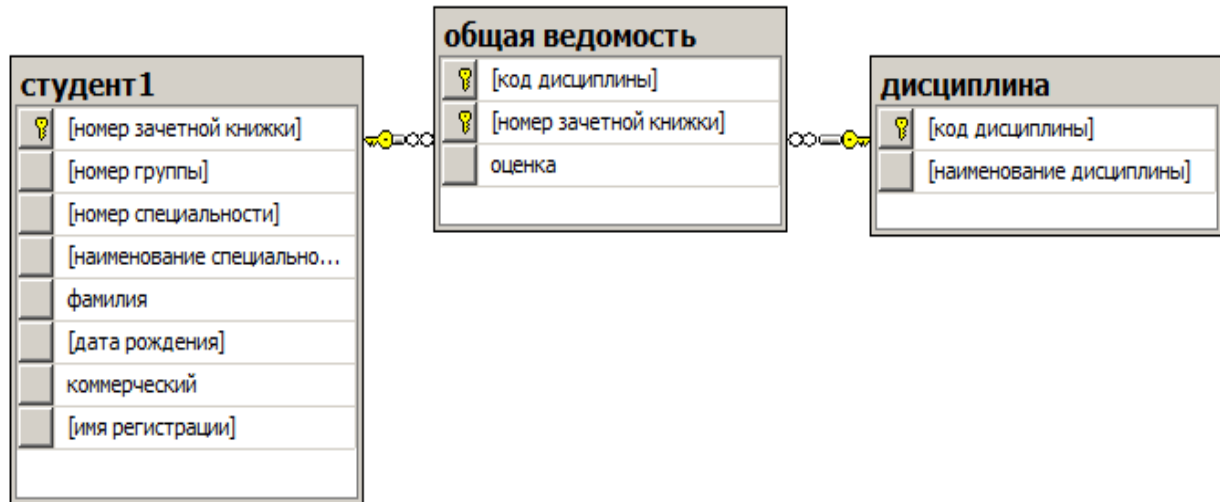


Рис. 27. Схема «звезда»

В схеме типа «снежинка» в центре схемы также располагается денормализованная таблица фактов. Она связана с нормализованными таблицами измерений. Между таблицей фактов и таблицами измерений установлены идентифицирующие связи, имеющие тип один-ко-многим. Первичный ключ таблицы фактов целиком состоит из первичных ключей всех таблиц измерений. Первичные ключи в таблицах измерений состоят из единственного атрибута (соответствуют единственному элементу измерения).

Несколько таблиц измерений, которые нормализованы в отличие от схемы «звезда» имеют меньшее количество строк, чем таблицы фактов, и содержат описательную информацию. Эти таблицы позволяют пользователю быстро переходить от таблицы фактов к дополнительной информации.

Измерение содержится в нескольких иерархически связанных между собой таблицах измерений, тем самым образуя иерархию измерения (см. рис. 28).

В схеме "снежинка" агрегированные данные могут храниться отдельно от исходных.

Преимущество ROLAP. Обеспечивает обработку больших объемов данных за счет более экономичного хранения.

Недостаток. При использовании высокого уровня обобщения и иерархии измерений растет число реляционных таблиц, соединение которых для получения агрегируемых данных — фактов, может снижать скорость обработки.

ROLAP- системы целесообразно использовать при значительных объемах, данных, несложной иерархии измерений и частых изменениях размерности данных.

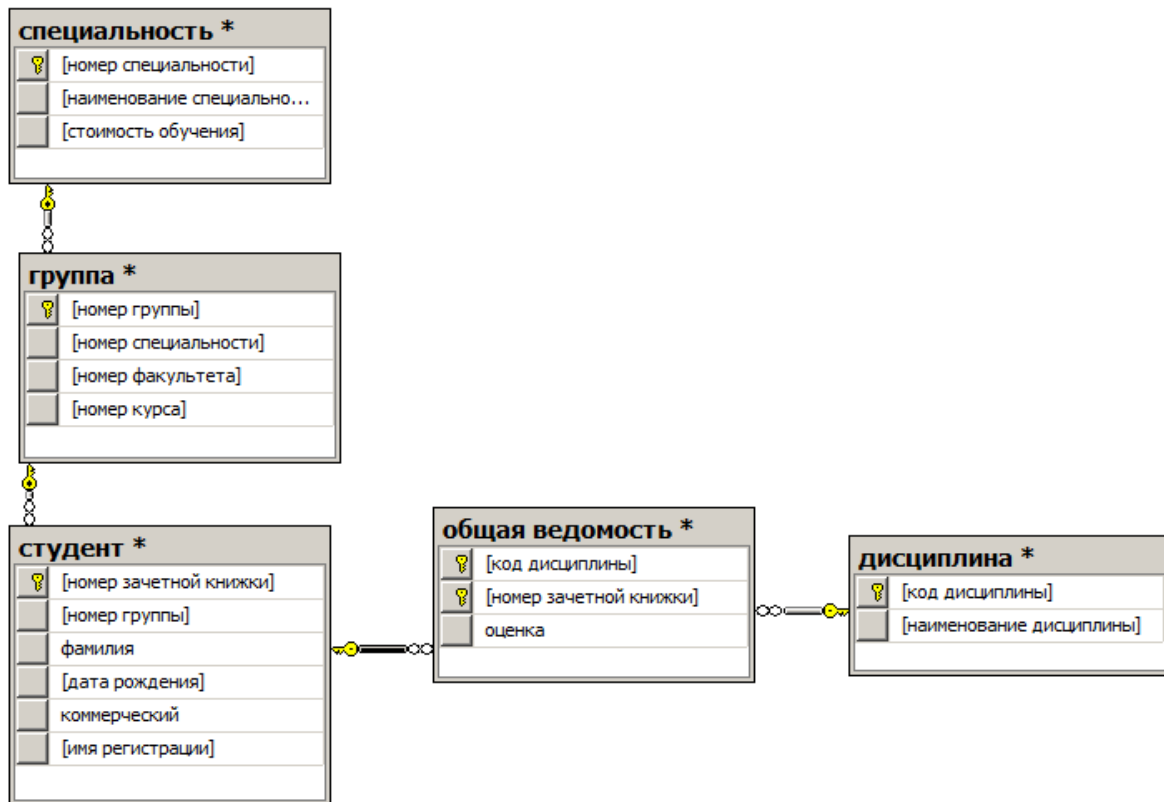


Рис. 28. Схема «Снежинка»

HOLAP представляет собой гибридный OLAP и позволяет сочетать многомерный и реляционный подход. При этом детализированные данные хранятся в реляционной структуре ROLAP, а агрегированные данные хранятся в многомерной структуре MOLAP. HOLAP сочетает высокую производительность многомерной модели и возможности реляционной модели по хранению больших массивов данных.

Современные СУБД поддерживают OLAP технологию. Серверная компонента СУБД Microsoft SQL Server служба Analysis Services предоставляет несколько подходов для создания многомерной семантической модели бизнес-аналитики: табличный, многомерный и Power Pivot для SharePoint. Все созданные модели разворачиваются как базы данных, которые работают в экземпляре служб Analysis Services и доступны клиентским средствам для представления данных в отчетах.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Федеральный закон Российской Федерации от 27 июля 2006 г. N 149-ФЗ «Об информации, информационных технологиях и о защите информации», 2006.
2. Роб П., Коронел К. Системы баз данных: проектирование, реализация и управление. – СПб.: БХВ-Петербург, 2004. – 1040 с
3. Кириллов В.В., Громов Г.Ю. Введение в реляционные базы данных. – СПб.: БХВ-Петербург, 2009. – 464 с.
4. Голицына О.Л., Патрыка Т.Л., Попов И.И. Система управления базами данных. – М.: Форум, Инфра-М, 2010.
5. Душан Петкович. Microsoft SQL Server 2012. Руководство для начинающих. – СПб.: БХВ-Петербург, 2013, – 816 с.
6. Ицик Бен-Ган. Microsoft SQL Server 2012. T - SQL: учебный курс. – СПб.: БХВ-Петербург, 2015, – 400 с.
7. Мердина О.Д., Стельмашенок Е.В., Быкова Н.Н. Информационное обеспечение, базы данных: учебное пособие. – СПб.: Изд-во СПбГЭУ, 2014. – 82 с
8. Мердина О.Д., Тарзанов В.В. Информационные технологии управления: учебное пособие. – СПб.: 2012. – 237 с.
9. Мердина О.Д., Шленов В.В. Microsoft SQL Server в клиент-серверных технологиях: учебное пособие. – СПб.: СПбГИЭУ, 2003. – 148 с.
10. Фуфаев Э.В., Фуфаев Д.Э. Базы данных: учебное пособие. – М.: Академия, 2011. – 320 с.
11. Microsoft. Электронная документация по SQL Server 2012 - [https://docs.microsoft.com/ru-ru/previous-versions/sql/sql-server-2012/ms130214\(v=sql.110\)](https://docs.microsoft.com/ru-ru/previous-versions/sql/sql-server-2012/ms130214(v=sql.110)) (20.02.2019г.)
12. DB-Engines Knowledge Base of Relational and NoSQL Database Management Systems - <https://db-engines.com/en/ranking> (27/02/2019г.)

Учебное издание

Мердина Ольга Дмитриевна

БАЗЫ ДАННЫХ

Учебное пособие

Издано в авторской редакции

Подписано в печать 17.05.19. Формат 60×84 1/16.
Усл. печ. л. 6,25. Тираж 60 экз. Заказ 658.

Издательство СПбГЭУ. 191023, Санкт-Петербург, Садовая ул., д. 21.

Отпечатано на полиграфической базе СПбГЭУ