**Электронный образовательный ресурс дисциплины «Базы данных»**

Оглавление

[1. Модуль 1. Логическое проектирование 3](#_Toc534481844)

[1.1. Теоретический материал 3](#_Toc534481845)

[1.1.1. Основные понятия, лежащие в основе концепции баз данных 3](#_Toc534481846)

[1.1.2. Модели данных 4](#_Toc534481847)

[1.1.3. Операции реляционной алгебры 8](#_Toc534481848)

[1.1.4. Типы связей между сущностями 15](#_Toc534481849)

[1.1.5. Нормализация данных 17](#_Toc534481850)

[1.1.6. Целостность данных 21](#_Toc534481851)

[1.2. Индивидуальная практическая работа 1 (ИПР1) 23](#_Toc534481852)

[1.2.1. Методические указания по выполнению ИПР1 23](#_Toc534481853)

[1.2.2. Варианты заданий ИПР1 27](#_Toc534481854)

[1.3. Промежуточный контрольный тест 28](#_Toc534481855)

[1.4. Правильные ответы на промежуточный контрольный тест 29](#_Toc534481856)

[2. Модуль 2. Физическое проектирование 31](#_Toc534481857)

[2.1. Теоретический материал 31](#_Toc534481858)

[2.1.1. Назначение и функции СУБД 31](#_Toc534481859)

[2.1.2. Типы баз данных 35](#_Toc534481860)

[2.2. Промежуточный контрольный тест 35](#_Toc534481861)

[2.3. Правильные ответы на промежуточный контрольный тест 36](#_Toc534481862)

[3. Модуль 3. Язык структурированных запросов SQL 38](#_Toc534481863)

[3.1. Теоретический материал 38](#_Toc534481864)

[3.1.1. Введение 38](#_Toc534481865)

[3.1.2. Команды манипулирования данными 38](#_Toc534481866)

[3.1.3. Основные команды определения данных 49](#_Toc534481867)

[3.2. Индивидуальная практическая работа 2 (ИПР2) 52](#_Toc534481868)

[3.2.1. Методические указания по выполнению ИПР2 53](#_Toc534481869)

[3.2.2. Варианты заданий ИПР2 54](#_Toc534481870)

[3.3. Промежуточный контрольный тест 55](#_Toc534481871)

[3.4. Правильные ответы на промежуточный контрольный тест 55](#_Toc534481872)

[4. Модуль 4. Распределенные базы данных и хранилища данных 57](#_Toc534481873)

[4.1. Теоретический материал 57](#_Toc534481874)

[4.1.1. Распределенные базы данных 57](#_Toc534481875)

[4.1.2. Системы «клиент/сервер» 58](#_Toc534481876)

[4.1.3. Поддержка принятия решений 58](#_Toc534481877)

[4.2. Промежуточный контрольный тест 62](#_Toc534481878)

[4.3. Правильные ответы на промежуточный контрольный тест 62](#_Toc534481879)

[5. Итоговый тест по дисциплине 63](#_Toc534481880)

[6. Правильные ответы на итоговый тест 64](#_Toc534481881)

[7. Сценарий изучения дисциплины 65](#_Toc534481882)

# 1. Модуль 1. Логическое проектирование

## 1.1. Теоретический материал

### 1.1.1. Основные понятия, лежащие в основе концепции баз данных

Понятие системы

В системном анализе системой называется совокупность взаимосвязанных элементов, действующих как единое целое в интересах достижения определенной цели. Необходимо подчеркнуть единство трех компонент: элементы – связи – цель. При этом система обладает такими качествами, которые не присущи ни одному из образующих ее элементов.

Каждая система обладает свойствами делимости и целостности.

*Делимость* системы означает, что ее можно представить состоящей из относитель­но самостоятельных частей — подсистем, каждая из которых рассматри­вается как система. Такой подход упрощает анализ системы, ее разработку и внедрение в эксплуатацию, однако процедура разбиения на подсистемы представляет собой нетривиальную задачу.

*Целостность* системы означает согласованность ее цели с целями образующих систему подсистем и элементов.

Понятие информационной системы

*Информационная система (ИС)* – это коммуникационная и вычислительная система по сбору, хранению, обработке и передаче информации, снаб­жающая работников различного ранга той информацией, которая необходима им для реализа­ции функций управления.

В зависимости от уровня автоматизации информационные системы подразделяются на автоматизированные и автоматические.

*В автоматизированных ИС* – часть функций по управлению и обработке информации выполняется автоматически, а часть – человеком.

*В автоматических ИС* – все функции управления и обработки информации выполняются техническими средствами без участия человека

В зависимости от сферы применения ИС подразделяются следующим образом

* системы обеспечения научных исследований;
* системы автоматизированного проектирования (САПР);
* автоматизированные системы управления производством (АСУП);
* автоматизированные системы управления технологическими процессами (АСУТП);
* системы управления процессами в реальном масштабе времени;
* системы поддержки принятия решений;
* системы обеспечения банковской деятельности;
* системы автоматизации офиса;
* бухгалтерские системы;
* системы автоматизации торговли;
* системы контроля качества;
* системы управления графиком выполнения работ;
* технологии виртуальной реальности;
* системы машинного перевода;
* технологии формирования и распространения информации для СМИ

Понятие системы баз данных

Система баз данных – это, по сути, не что иное, как компьютеризированная система хранения и управления массивами упорядоченных или, как еще говорят, структурированных данных. Саму же базу данных можно рассматривать как подобие очень хорошо организованной электронной картотеки, т.е. как хранилище или контейнер для некоторого набора файлов данных.

Пользователям этой системы предоставляется возможность извлечения и модификации информации, хранящейся в этих файлах. К информации может относиться все, что заслуживает внимания отдельного пользователя или организации, использующей систему, иначе говоря, все необходимое для работы данного пользователя или предприятия.

Роль БД в качестве экономических активов непрерывно возрастает, и они все шире используются в различных сферах науки и техники, производства, образования, бизнеса и экономики. Новые сферы применения БД связаны с системами поддержки принятия решений, экспертными системами, автоматизированным проектированием, разработкой программного обеспечения, национальными программами создания цифровых библиотек и др.

Одна из важнейших целей создания систем баз данных - обеспечение независимости от данных и обеспечение целостности данных.

Независимость от данных можно определить как невосприимчивость приложений к изменениям в физическом представлении данных и в методах доступа к ним. Например, в базу данных могут быть добавлены новые виды данных, могут появиться новые типы запоминающих устройств и т.д., однако это не повлечет за собой необходимость модификации существующих приложений.

Целостность данных означает их правильность в любой момент времени.

**Жизненный цикл БД**

Жизненный цикл БД – это процесс ее проектирования, реализации и поддержки. Он состоит из следующих этапов:

1) предварительное планирование;

2) проверка осуществимости;

3) выработка требований;

4) концептуальное проектирование;

5) логическое проектирование;

6) физическое проектирование;

7) оценка работы, поддержка и сопровождение.

На этапе *концептуального проектирования* анализируются подробные модели пользовательских представлений о данных предметной области. Затем они интегрируются в концептуальную модель, фиксирующую все элементы корпоративных данных, которые будут загружаться и хранится в БД.

На этапе *логического проектирования* производится выбор типа логической модели данных, в результате чего определяются конкретные структуры, в которых будут храниться данные. Затем концептуальная модель отображается в логическую модель, основанную уже на структурах, характерных для выбранной модели.

На этапе *физического проектирования* логическая модель расширяется характеристиками, необходимыми для определения: способов физического хранения данных, типов запоминающих устройств, методов доступа к данным, требуемого объема памяти, правил сопровождения БД и т.п.

### 1.1.2. Модели данных

Понятие модели

Сущность моделирования заключается в установлении отношения эквивалентности между некоторой реальной или проектируемой системой с одной стороны, и абстрактной системой, описанной на каком-либо языке, – с другой. Если вторая из этих систем оказывается проще для исследования чем первая, то о свойствах первой можно судить, наблюдая поведение второй. В этом случае используемую для исследования систему называют моделью.

Различают модели математические, логические, физические, функциональные, структурные и др.

Модели могут описываться на любых языках: на языке математических понятий и уравнений, на языке графических схем, на языке биологии, химии и т.д.

Модели, описанные на языке математических понятий и уравнений, называются математическими. Создание математических моделей для тех или иных предметных областей составляет суть всей прикладной математики.

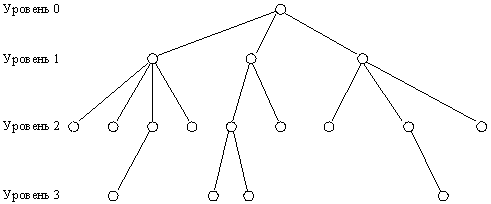
Логические модели данных

Как и концептуальные модели, основанные на пользовательских представлениях о данных предметной области, логические модели данных отражают абстрактный взгляд на данные. Это означает, что данные представляются так, как они выглядят в реальном мире, например, «постоянный клиент», «отдел», «фамилия сотрудника» и т. д. Однако, в отличие от концептуальных моделей, логические модели данных определяют конкретные структуры, в которых будут храниться данные.

Разработка логических моделей данных началась в 60-ых годах прошлого столетия в связи с бурным развитием средств вычислительной техники. Можно выделить 4 основные разновидности логических моделей данных: иерархические, сетевые, реляционные и объектно-ориентированные (перечислены с учетом хронологии их появления).

**Иерархическая модель данных**

Эта модель графически представляется совокупностью элементов (вершин) и связей (ребер), образующих структуру типа «дерево»:



Каждая вершина связана с одной вершиной предыдущего более высокого уровня и с любым (в том числе нулевым) числом вершин следующего более низкого уровня. Исключением является вершина самого высокого уровня, которая связана только с вершинами более низкого уровня. Эта вершина называется корнем дерева и располагается на нулевом уровне. Подчиненные вершины располагаются на последующих уровнях. Вершины, не имеющие подчиненных вершин, называются концевыми.

В этой модели каждому элементу данных соответствует единственный путь, ведущий к нему из корневой вершины.

Данные чаще всего представляются концевыми вершинами, но это не обязательно. Важнейшими операциями для работы с иерархическими структурами являются операции перемещения (навигации) по иерархическим указателям, позволяющие переходить вверх и вниз по ребрам дерева.

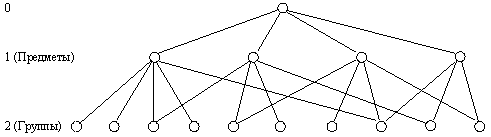
Достоинством данной модели является возможность выполнить очень быстрый поиск нужных данных. Недостаток заключается в том, что, если данные плохо поддаются иерархической организации, – а это часто бывает на практике, – то модель чрезмерно усложняется и соответственно снижается производительность при технической реализации.

**Сетевая модель данных**

В ее основе лежат те же понятия, что и в иерархической модели, т.е. вершина, уровень, связь, однако существенным отличием является то, что любая вершина может быть связана с любой другой вершиной независимо от уровня иерархии последней.

Если использовать язык теории графов, то иерархическая структура отображается графом типа дерево, а сетевая структура отображается произвольным графом, т.е. графом, допускающим существование контуров (или циклов – для ориентированных графов).

Примером сетевой модели данных может служить модель, отражающая распределение учебных дисциплин по группам студентов: каждая группа изучает ряд предметов, а каждый предмет изучается в каких-то группах.



Достоинством сетевой модели является высокая скорость поиска нужных данных и возможность адекватно представлять данные для решения задач в самых различных предметных областях. Недостаток заключается в том, что сетевые модели при реализации образуют сложные структуры данных с многочисленными связями.

Общий недостаток и сетевой и иерархической моделей заключается в том, что конфигурация структуры данных (перечень вершин и связей между ними) задается на этапе моделирования и после перехода к практической реализации уже не может быть изменена. Другими словами, независимость по данным в этих моделях не сопровождается структурной независимостью.

**Реляционная модель данных**

Одним из самых естественных способов представления данных является представление их в виде двумерных таблиц. Это могут быть различные отчеты и сводки, реестры, ведомости, расписание движения транспорта и т.д.

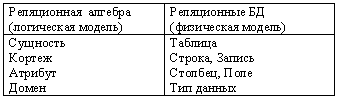
Если имеется несколько таблиц, то связи между данными в разных таблицах также можно представить в табличной форме, используя дополнительные, связующие таблицы.

Существенным отличием реляционной модели от графовых моделей (иерархическая, сетевая) является переход от работы с элементами данных к работе с макрообъектами табличного типа. При этом реляционные операции применяются сразу ко всему множеству строк таблицы, а не к отдельной ее строке, и, таким образом, операндами и результатами являются наборы данных, содержащие множество строк.

Еще одно важное отличие реляционной модели состоит в том, что независимость по данным здесь сопровождается и структурной независимостью, т.к. нет никаких указателей, связывающих таблицы друг с другом, и соответственно, отсутствует навигационная схема доступа к данным.

Было установлено, что любая сетевая структура данных может быть представлена (с некоторой избыточностью) совокупностью древовидных структур, а те, в свою очередь, (тоже с некоторой избыточностью) – в виде взаимосвязанных двумерных таблиц. Поэтому тот факт, что любую структуру данных при желании можно представить в виде взаимосвязанных двумерных таблиц и привел к созданию реляционной модели данных. Эта модель была предложена в 1970 г. сотрудником фирмы IBM Эдгаром Коддом и вскоре получила самое широкое распространение из-за простоты и наглядности модели для пользователей-непрограммистов, с одной стороны, и серьезного теоретического обоснования, лежащего в основе модели, – с другой стороны.

Математический аппарат, используемый для моделирования табличного представления данных, получил название реляционной алгебры. В ней используются следующие основные понятия.



*Сущность* – это объект предметной области, которая исследуется и моделируется. Другими словами сущность – это любой различимый объект, о котором необходимо хранить информацию в базе данных.

*Кортеж* – это экземпляр сущности. Например для сущности «Студент» кортеж – это конкретный студент, например,: студент Иванов, студент Петров и т.д. Для сущности «Предмет» кортеж – это конкретный предмет: высшая математика, физика, история и т.д.

*Атрибут* отражает определенное свойство, качество, признак сущности. Например для сущности студент атрибутами могут быть: № зачетной книжки, № группы, фамилия, имя, отчество, год рождения, адрес, телефон и т.д.

*Домен* задает множество допустимых значений атрибута. Например для атрибута «Экзаменационная оценка» это будет интервал [0, 10]; для атрибута «Предмет» – перечень все учебных дисциплин, изучаемых в ВУЗе и т.д.

Смысл доменов состоит в том, что они ограничивают сравнения и позволяют правильно моделировать предметную область. Действительно, если атрибуты связаны с одним и тем же доменом, то сравнение значений этих атрибутов имеет смысл (например, минимальные, средние и максимальные температуры можно сравнивать между собой; аналогично, можно сравнивать даты и т д.). Если же атрибуты связаны с различными доменами, то сравнение значений таких атрибутов чаще всего лишено смысла, даже если эти значения выбираются из одного и того же множества значений (стоит ли сравнивать, например, № группы с № телефона?).

**Определение сущности**

Сущность S на доменах D1,D2,…,Dn (не обязательно, чтобы все они были различны) состоит из заголовка и тела.

Заголовок состоит из такого множества атрибутов {A1,A2,…,An} что существует взаимно однозначное соответствие между атрибутом Ai и определяющим его доменом Di для всех i = 1,2,...,n.

Тело сущности состоит из меняющегося во времени множества кортежей, где каждый кортеж, в свою очередь, состоит из множества пар вида «атрибут-значение» {(A1,V1),(A2,V2),…,(An,Vn)}. Для любой такой пары (Ai,Vi) Vi является некоторым значением из домена Di, связанного с атрибутом Ai.

Степень сущности – это число ее атрибутов. Мощность или кардинальное число сущности – это число ее кортежей. Мощность сущности изменяется во времени в отличие от ее степени.

*Замечания*.

1. Поскольку множество в математике не содержит повторяющихся элементов, то сущность также не содержит повторяющихся кортежей и повторяющихся атрибутов. Однако среди доменов D1,D2,…,Dn повтор возможен.

2. Сущности и таблицы – не одно и то же. Строки и столбцы таблицы упорядочены, а кортежи и атрибуты сущности – нет, т.к. это множества, а множества в математике не обладают свойством упорядоченности.

3. Все атрибуты сущности являются простыми (атомарными). Это означает, что их нельзя разложить на составные части без потери смысла. Например, если фамилию сотрудника разложить в массив символов, то смысл фамилии как единого понятия теряется.

4. Заголовок сущности статичен, и описывает декартово произведение доменов, на которых задана сущность. Тело сущности представляет собой изменяемый во времени набор кортежей, т.е. подмножество декартового произведения доменов. Таким образом, если использовать терминологию теории множеств, тело сущности является отношением. Поскольку «отношение» на английском есть “relation”, то отсюда и произошло название «Реляционная модель данных».

**Понятие первичного ключа**

Подмножество K = {Ai,Aj,…,Ak} атрибутов cущности S является ее *возможным или потенциальным ключом,* если выполняются следующие два условия:

1. *Уникальность –* никакие два различных кортежа из S не имеют одного и того же набора значений атрибутов Ai,Aj,…,Ak.
2. *Минимальность –* ни один из атрибутов Ai,Aj,…,Ak не может быть исключен из K без нарушения условия уникальности.

*Утверждение*. Каждая сущность обладает по меньшей мере одним возможным ключом.

Действительно, если других ключей нет, то им будет ключ K = {A1,A2,…An}, поскольку, как следует из определения сущности, все ее кортежи различны.

Возможный ключ, состоящий из одного атрибута, называется *простым*, а состоящий из нескольких атрибутов – *составным*.

Один из возможных ключей выбирается в качестве первичного ключа. Остальные возможные ключи, если они имеются, получают статус альтернативных или уникальных ключей.

Роль ключа (первичного или альтернативного) заключается в том, что зная для какого-либо кортежа значения лишь ключевых атрибутов, можно всегда определить значения и всех остальных атрибутов. Например, по номеру зачетной книжки можно определить фамилию и группу студента при условии, что номер зачетной книжки является ключом сущности «Студент».

При поиске возможных ключей необходимо проводить смысловой анализ понятий конкретной предметной области. Формальный алгоритм поиска ключей отсутствует.

### 1.1.3. Операции реляционной алгебры

Реляционная алгебра базируется на понятиях теории множеств и математической логики. Каждая операция реляционной алгебры использует одну или несколько сущностей в качестве операндов и продуцирует в результате некоторую новую сущность.

Существует 8 базовых реляционных операций, которые разбиваются на две группы: теоретико-множественные операции и специальные операции.

К теоретико-множественным реляционным операциям относятся операции объединения, пересечения, разности и декартова произведения.

К специальным реляционным операциям относятся операции селекции, проекции, соединения и деления.

*Определение*. Сущности S1 и S2 совместимы по типу, если они одной и той же степени, например n, и их i-е атрибуты связаны с одним и тем же доменом Di для всех i = 1,2,…,n. Другими словами, совместимые по типу сущности имеют одинаковые заголовки.

*Операция объединения.* Объединением двух совместимых по типу сущностей S1 и S2 называется сущность с тем же заголовком, что и у сущностей S1 и S2, и телом, содержащим все кортежи сущности S1 и все кортежи сущности S2.

Синтаксис операции объединения: S1 UNION S2

*Замечание*. Объединение, как и любая сущность, не может содержать одинаковых кортежей, поэтому, если какой-то кортеж принадлежит и сущности S1 и сущности S2, то в объединение он включается только один раз.

*Операция пересечения*. Пересечением двух совместимых по типу сущностей S1 и S2 называется сущность с тем же заголовком, что и у сущностей S1 и S2, и телом, состоящим из всех кортежей, принадлежащих одновременно и сущности S1 и сущности S2.

Синтаксис операции пересечения: S1 INTERSECT S2

*Операция разности*. Разностьюдвух совместимых по типу сущностей S1 и S2 называется сущность с тем же заголовком, что и у сущностей S1 и S2, и телом, состоящим из всех кортежей, принадлежащих сущности S1 и не принадлежащих сущности S2.

Синтаксис операции разности: S1 MINUS S2

*Замечание*. Операция разности в отличие от операций объединения и пересечения не является коммутативной операцией и не является ассоциативной операцией.

*Операция декартова произведения.* В математике декартово произведение двух множеств X и Y записывается следующим образом:

X × Y = {(xi, yj) | xi ∈ X, yj ∈ Y}

x1 y1

x1 y2

x2 y1

x2 y2

x3 y1

x3 y2

y1

y2

x1

x2

x3

× =

*Декартовым произведением* двух сущностей S и R с заголовками {A1, …, An} и {B1, …, Bm} соответственно называется сущность с заголовком {A1, …, An, B1, …, Bm} и телом, состоящим из множества кортежей вида (a1, …, an, b1, … bm), таких, что (a1, …, an) ∈ S и (b1, … bm) ∈ R.

Синтаксис операции декартова произведения: S TIMES R

*Замечания*.

1. Степень декартова произведения S TIMES R равна сумме степеней сущностей S и R, а мощность – произведению их мощностей, т.к. каждый кортеж сущности S соединяется с каждым кортежем сущности R.

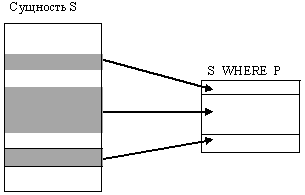
2. Если в сущностях S и R имеются атрибуты с одинаковыми именами, то перед выполнением операции декартова произведения их необходимо переименовать так, чтобы каждое имя встречалось только один раз.

3. Перемножать можно любые две сущности, т.к. совместимости по типу не требуется.

**Специальные реляционные операции**

*Операция селекции*. Селекциейсущности S по условию P называется сущность с тем же заголовком, что и у сущности S, и телом, состоящим из тех кортежей сущности S, которые удовлетворяют условию P. Условие P – это предикат, т.е. выражение логического типа, которое может принимать одно из двух значений: истина или ложь. Предикат задает условие отбора кортежей сущности S: те кортежи, для которых значение предиката есть истина, будут включены в результирующую сущность.

Синтаксис операции селекции: S WHERE P

****

Обычно предикаты строятся с использованием имен атрибутов, констант, арифметических операций, операций сравнения (=, ≠, <, ≤, >, ≥) и логических связок AND, OR, NOT (соответственно И, ИЛИ, НЕ).

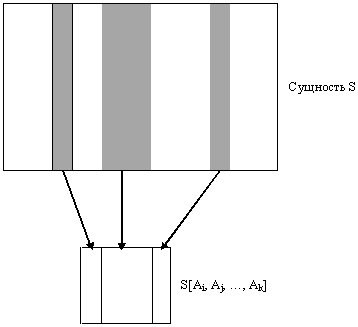
Принято говорить, что операция селекции позволяет получить "*горизонтальный срез*" сущности, т.е. подмножество ее кортежей, удовлетворяющих некоторому заданному условию.

*Операция проекции*. Проекциейсущности S по подмножеству ее атрибутов {Ai, Aj, …, Ak}, называется сущность с заголовком {Ai, Aj, …, Ak} и телом, содержащим множество кортежей вида (ai, aj, …, ak) таких, для каждого из которых в сущности S найдется кортеж со значением атрибута Ai равным ai, значением атрибута Aj равным aj, и т.д., значением атрибута Ak равным ak.

Синтаксис операции проекции: S[Ai, Aj, …, Ak].

*Замечание*. Т.к. степень результирующей сущности меньше, чем степень сущности S, то выполнение операции проекции сопровождается удалением появляющихся при этом дубликатов кортежей.

Принято говорить, что операция проекции позволяет получить "*вертикальный срез*" сущности, т.е. сущность без тех атрибутов, имена которых не указаны в заданном списке имен атрибутов.

****

*Операция соединения*. Эта операция является одной из важнейших реляционных операций и имеет четыре разновидности:

1. общая операция соединения;
2. тэта-соединение;
3. экви-соединение;
4. естественное соединение.

Последние три разновидности являются частными случаями общей операции соединения.

Общая операция соединения. Общая операция соединения сущностей S и R по условию P позволяет получить сущность

(S TIMES R) WHERE P,

где предикат P может включать в себя имена атрибутов и сущности S и сущности R.

Таким образом, общая операция соединения есть результат последовательного применения операций декартова произведения и селекции.

Тэта-соединение. Эта операция предназначена для тех случаев, когда необходимо соединить две сущности по одиночным атрибутам на обнове некоторой операции сравнения.

Пусть заголовок сущности S включает в себя атрибут A, а заголовок сущности R включает в себя атрибут B. Тогда *тэта-соединением* сущностей S и R по атрибутам A и B называется сущность

(S TIMES R) WHERE A θ B,

где «θ» – это одна из операций сравнения: =, ≠, <, ≤, >, ≥.

Обычно для операции тэта-соединения применяют следующий, более короткий синтаксис: S[A θ B]R.

Экви-соединение. Если θ представляет собой операцию проверки на равенство, то тэта-соединение называется экви-соединением или соединением по равенству: S[A = B]R.

Недостатком экви-соединения является то, что если соединение происходит по совпадающим атрибутам, т.е. имеющим одинаковые имена и связанные с одним и тем же доменом (а так чаще всего и происходит на практике), то в результирующей сущности появляются два атрибута с одинаковыми значениями. Избавиться от этого недостатка можно, взяв проекцию по всем атрибутам, кроме одного из дублирующих. Именно так действует естественное соединение.

Естественное соединение. Пусть даны сущность S с заголовком {A1, …, An,B1, …, Bm} и сущность R с заголовком {B1, …, Bm, C1, …, Cp}, при этом заголовки содержат совпадающие атрибуты B1, …, Bm. Тогда *естественным соединением* сущностей S и R называется сущность с заголовком {A1,…, An,B1, …, Bm, C1, …, Cp} и телом, содержащим множество кортежей вида (a1,…, an,b1, …, bm, c1, …, cp), таких, для каждого из которых имеет место:

(a1,…, an,b1, …, bm) ∈ S и (b1, …, bm, c1, …, cp) ∈ R.

Естественное соединение является настолько важным, что для него используется специальный синтаксис: S JOIN R

*Замечания*.

1. В синтаксисе естественного соединения не указывается, по каким атрибутам производится соединение. Это объясняется тем, что соединение производится *по всем* одинаковым атрибутам.

2. Естественное соединение эквивалентно следующей последовательности операций:

а) переименовать в сущностях атрибуты с одинаковыми именами;

б) выполнить декартово произведение сущностей;

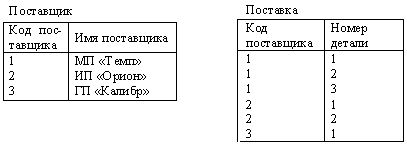
в) выполнить операцию селекции по совпадающим значениям атрибутов, имевших вначале одинаковые имена;

г) выполнить операцию проекции, удалив повторяющиеся атрибуты (с учетом того, что было выполнено переименование);

д) вернуть первоначальные имена тем атрибутам, которые были переименованы.

3. Естественное соединение (как, впрочем, и соединение общего вида) обладает свойством *ассоциативности*, т.е. (S JOIN R) JOIN T ≡ S JOIN (R JOIN T), поэтому такие соединения можно записывать, опуская скобки: S JOIN R JOIN T.

*Пример*. Необходимо получить естественное соединение сущностей «Поставщик» и «Поставка», которые для простоты заданы в виде следующих таблиц:

****

Результат:

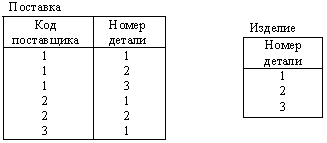
****

Операция деления. Пусть даны сущность S с заголовком {A1,…, An,B1, …, Bm} и сущность R с заголовком {B1, …, Bm}, при этом заголовки содержат совпадающие атрибуты B1, …, Bm. Тогда *делением сущностей S на R* называется сущность с заголовком {A1,…, An} и телом, содержащим множество кортежей вида (a1,…, an), таких, что для любого кортежа (b1, …, bm) ∈ R существует кортеж (a1, …, an, b1, …, bm) ∈ S.

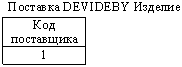
Сущность S выступает в роли *делимого*, сущность R выступает в роли *делителя*. Деление сущностей похоже на деление целых чисел с отбрасыванием остатка.

Синтаксис операции деления: S DEVIDEBY R

*Замечание*. Типичные задачи, решаемые с помощью операции деления, обычно содержат в своей формулировке слово "все". Например: какие поставщики поставляют *все* детали для изготовления некоторого изделия?

****

Результат:

****

Оказалось, что только поставщик с кодом 1 поставляет все детали, необходимые для изготовления изделия.

Общие замечания к базовым операциям реляционной алгебры.

*Замечание 1*. Не все операции реляционной алгебры являются примитивными, т.е. независимыми от других операций. Некоторые из них могут быть определены в терминах других операций. Так операция пересечения выражается через операцию разность:

S INTERSECT R ≡ S MINUS (S MINUS R) ≡ R MINUS (R MINUS S)

Операция деления выражается через операции разность, декартово произведение и проекцию:

S DEVIDEBY R ≡ S[A] MINUS ((S[A] TIMES R) MINUS S)[A],

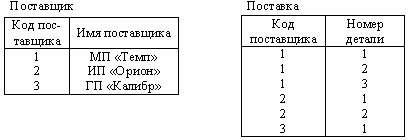
где A = {A1,…, An}.

Операция соединения, как рассматривалось выше, определяется через операции декартово произведение и селекцию. Для операции естественного соединения добавляется еще операция проекции.

Остальные реляционные операции (*объединение*, *разность*, *декартово произведение*, *селекция*, *проекция*) являются примитивнымиоперациями и не определяются друг через друга.

*Замечание 2*. *Ни одна из реляционных операций не передает результирующей сущности никаких сведений о возможных ключах*. Это объясняется тем, что возможный ключ – понятие семантическое, отражающее различимость экземпляров объектов предметной области. Поэтому возможные ключи *не выводятся* из структуры сущности, а определяются на основе анализа ее смысла.

*Заключительный пример*. Необходимо определить имена поставщиков, которые поставляют деталь номер 2.

****

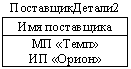
Решение:

((Поставщик JOIN Поставка) WHERE Номер детали = 2)[Имя поставщика]

или, что то же самое,

((Поставка WHERE Номер детали = 2) JOIN Поставщик)[Имя поставщика]

Можно поставить вопрос, какой вариант лучше?

****

Подводя итог можно сказать, что модель данных является реляционной, если:

1) данные воспринимаются как сущности и только как сущности;

2) любые операции над сущностями приводят к генерации новых сущностей из старых.

Достоинствами реляционной модели являются наглядность, простота и гибкость структуры данных, а также относительная простота практической реализации. Недостатком является некоторая ограниченность и предопределенность набора атрибутов и доменов, что можно устранить при переходе к объектно-реляционной модели.

**Объектно-ориентированные модели данных.**

Объектно-ориентированные логические модели данных основаны на использовании базовых идей объектно-ориентированного программирования и предназначены для моделирования сложных структур данных.

Существует два направления развития объектно-ориентированных моделей данных. Первое направление связано с созданием моделей, принципиально отличных от традиционных моделей (иерархической, сетевой, реляционной). Суть второго направления заключается в расширении реляционной модели данных объектно-ориентированными средствами. Это так называемые объектно-реляционные модели. Их преимущество – в использовании огромного пространства уже существующих информационных технологий реляционного типа, для которых созданы тысячи прикладных программ и систем. Если интенсивно развивающиеся в настоящее время объектно-ориентированные базы данных пока представляют собой уникальные разработки, то объектно-реляционные базы данных уже широко внедряются в различные сферы научной и практической деятельности человека.

### 1.1.4. Типы связей между сущностями

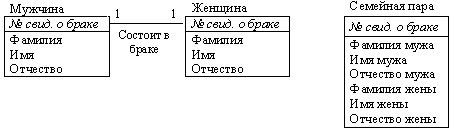
В реляционной модели данных сущности обычно связаны друг с другом, поскольку моделируемые ими объекты предметной области, как правило, не являются изолированными друг от друга.

Очень важно то, что связи являются такой же неотъемлемой составляющей данных, как и основные сущности. Поэтому связи должны быть представлены в базе данных наравне с основными сущностями предметной области.

Каждая связь между сущностями идентифицируется содержательным названием, обычно глаголом (в активной или пассивной форме) или глагольной формой. Например, «студент» занимается в «группе», «отделом» руководит «сотрудник», «самолет» управляется «экипажем» и т.д.

Существует 3 основных типа связей между сущностями.

*Связь «один к одному» (1 : 1)*. Эта связь каждому кортежу одной сущности ставит в соответствие не более одного кортежа другой сущности и наоборот. Например:

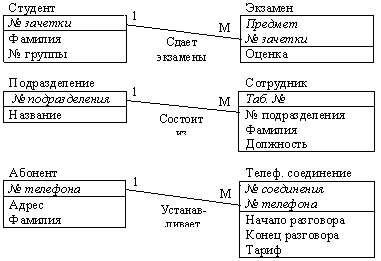
****

Другие примеры – связь между сущностями «декан» и «факультет» или между сущностями «человек» и «национальность» и др.

Связь «один к одному» на практике встречается довольно редко, поскольку сущности с такой связью обычно объединяют в одну сущность. Связь «один к одному» используют тогда, когда не хотят, чтобы сущность «разросталась» от включения в нее второстепенной информации, которую и оформляют в виде новой сущности и снабжают тем же ключом, что и у основной сущности. Так сущности «Мужчина» и «Женщина» можно совместить в сущность «Семейная пара».

*Связь «один ко многим» (1 : М)*. В этом случае одна из связываемых сущностей выступает в роли главной или родительской, а другая – в роли подчиненной или дочерней. Эта связь каждому кортежу родительской сущности ставит в соответствие любое (в том числе нулевое) число кортежей дочерней сущности, однако каждый кортеж дочерней сущности может быть связан только с одним кортежем родительской сущности.

Механизм реализации связи «один ко многим» состоит в том, что в дочернюю сущность добавляются атрибуты, дублирующие ключевые атрибуты родительской сущности. Эти атрибуты получают название *внешнего ключа* и с их помощью становится возможной связь между кортежами родительской сущности – с одной стороны – и подмножествами кортежей дочерней сущности – с другой. Еще такие атрибуты называют *мигрирующими* из родительской сущности. Если дочерняя сущность является зависимой от родительской сущности, то мигрирующие атрибуты включаются в состав первичного ключа дочерней сущности, в противном случае они включаются в состав ее неключевых атрибутов. Примеры:

****

Связь «один ко многим» является самой распространенной и позволяет, в частности, моделировать иерархические структуры данных.

Замечания.

1. Внешний ключ, также как и первичный ключ, может быть простым и составным.

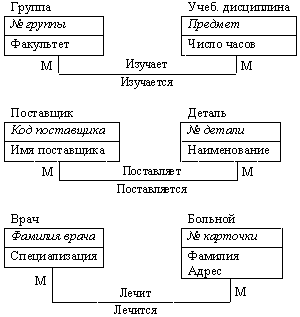
2. Внешний ключ определен на тех же доменах, что и соответствующий ему первичный ключ родительской сущности, однако имена атрибутов, составляющих внешний и первичный ключи, не обязательно должны совпадать.

3. Внешний ключ, как правило, *не обладает свойством уникальности,*  т.е. в дочерней сущности может быть несколько кортежей, ссылающихся на один и тот же кортеж родительской сущности. В противном случае связь между сущностями будет иметь тип "*один к одному*".

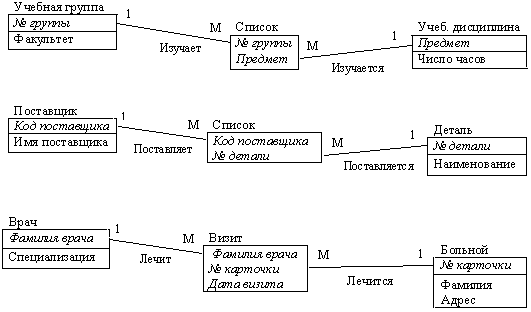
4. Каждое значение внешнего ключа обязано совпадать со значением первичного ключа в некотором кортеже родительской сущности, однако обратное, вообще говоря, неверно. Например, может существовать подразделение, в которое еще не зачислен ни один сотрудник.

5. Для внешнего ключа допускается, чтобы он был частью некоторого возможного ключа дочерней сущности.

*Связь «многие ко многим» (М : М)*. Эта связь каждому кортежу одной сущности ставится в соответствие любое (в том числе нулевое) число кортежей другой сущности и наоборот. Примеры:

****

В отличие о связей типа «1:1» и «1:М» связь «М:М» трудно реализовать в физической модели данных, поэтому эту связь при переходе к физической модели преобразуют в две связи типа «один ко многим» путем создания дополнительной, связующей сущности. Последняя снабжается первичным ключом, составленным из ключей исходных сущностей.

****

### 1.1.5. Нормализация данных

Обычно первоначальная логическая модель данных в виде набора взаимосвязанных сущностей не является оптимальной с точки зрения избыточности данных и наличия в них различных аномалий.

Нормализация – это процесс проверки и реорганизации сущностей в результате которых устраняются избыточные копии данных и тем самым предотвращается возможность появления противоречивых данных, поскольку информация о каждом факте будет хранится только в одном месте.

Процесс нормализации заключается в последовательном приведении сущностей к нормальным формам. Известны 5 нормальных форм.

Предварительно введем необходимые понятия и определения.

*Определение 1*. Атрибут B сущности функционально зависит от атрибута A этой же сущности, если каждому значению атрибута А соответствует точно одно значение атрибута B (другими словами А однозначно определяет B). Например: атрибут «Оклад по должности» функционально зависит от атрибута «Должность.

*Определение 2*. Если между атрибутами А и В сущности отсутствует функциональная зависимость, однако атрибут А ограничивает диапазон изменения атрибута В некоторым числом определенных значений, то говорят, что между такими атрибутами существует многозначная зависимость. Например: многозначная зависимость существует между атрибутами «№ группы» и «Предмет», или между атрибутами «Наименование детали» и «Имя поставщика» и т.д.

**Первая нормальная форма**

Сущность находится в первой нормальной форме, если все атрибуты являются простыми или атомарными (их нельзя раделить на составные части без потери смысла) и среди атрибутов отсутствуют повторяющиеся группы. Также, не допускается хранить в одном атрибуте разные по смыслу значения.

Для приведения сущности к первой нормальной форме необходимо:

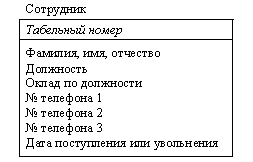
- разделить сложные атрибуты на атомарные;

- для групп повторяющихся атрибутов создать новые сущности;

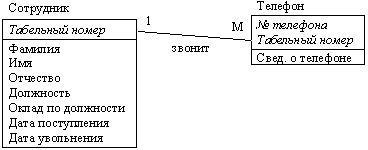
- установить с новыми сущностями связи типа «один ко многим»;

- атрибуты, хранящие разносмысловую информацию, разделить на односмысловые.

Пример:



После приведения сущности «Сотрудник» к первой нормальной форме получим:



**Вторая нормальная форма**

Сущность находится во второй нормальной форме, если она находится в первой нормальной форме и в ней отсутствуют неключевые атрибуты, функционально зависящие от части первичного ключа. Если сущность имеет простой первичный ключ (т.е. состоящий из одного атрибута) и находится в первой нормальной форме, то она автоматически находится и во второй нормальной форме.

Для приведения сущности ко второй нормальной форме необходимо:

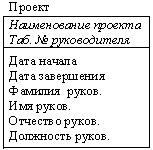
- выделить атрибуты, которые функционально зависят от одной и той же части первичного ключа;

- поместить эти атрибуты в новую сущность;

- установить с новой сущностью связь типа «один ко многим»;

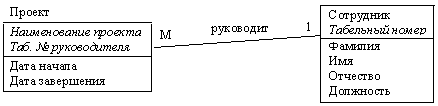
- повторить указанные выше действия, если это возможно.

Пример.



Сущность «Проект» находится в первой нормальной форме. После ее приведения ко второй нормальной форме получим:

**Сотрудник**



Приведение ко второй нормальной форме позволяет избежать следующих аномалий.

*Аномалия вставки*. Невозможно ввести данные о сотруднике, если он в данный момент не руководит ни одним проектом.

*Аномалия обновления*. Если сотрудник руководит несколькими проектами и данные о сотруднике меняются, то необходимо корректировать несколько кортежей сущности «Проект». Если же выполнить нормализацию, то достаточно внести изменения лишь в один кортеж сущности «Сотрудник».

*Аномалия удаления.* Если сотрудник временно прекращает руководство проектами, данные о нем теряются.

*Замечание*. Причиной этих аномалий является избыточность данных, обусловленная тем, что в одной сущности хранится разнородная информация (о проектах и о сотрудниках).

**Третья нормальная форма**

Сущность находится в третьей нормальной форме, если она находится во второй нормальной форме и между ее неключевыми атрибутами отсутствуют функциональные зависимости.

Таким образом, можно сказать, что в третьей нормальной форме каждый неключевой атрибут сущности функционально зависит только от всего первичного ключа и ни от чего другого.

Для приведения сущности к третьей нормальной форме необходимо:

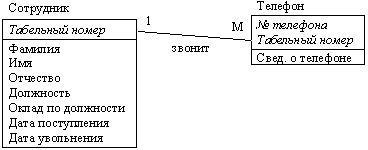
- выделить атрибуты, которые функционально зависят от одного и того же неключевого атрибута;

- поместить эти атрибуты в новую сущность;

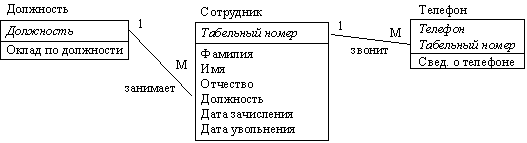
- установить с новой сущностью связь типа «один ко многим»;

- повторить указанные выше операции, если это возможно.

Вернемся к примеру, рассмотренному ранее:



Выполнив приведение к третьей нормальной форме получим:



Приведение к третьей нормальной форме позволяет избежать следующих аномалий.

*Аномалия вставки*. Невозможно ввести данные об окладе, если в данный момент соответствующую должность не занимает ни один сотрудник.

*Аномалия обновления*. Если должность занимают несколько сотрудников и оклад по должности меняется, то необходимо корректировать несколько кортежей сущности «Сотрудник». Если же выполнить нормализацию, то достаточно внести изменения лишь в один кортеж сущности «Должность».

*Аномалия удаления.* Если удалить сотрудника, занимающего уникальную должность, то информация об окладе теряется.

*Замечание*. Причиной этих аномалий опять является избыточность данных, обусловленная тем, что в одной и той же сущности «Сотрудник» хранится информация и о сотрудниках и о должностных окладах.

**Усовершенствованная третья нормальная форма или нормальная форма Кодда-Бойса**

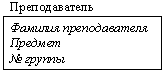
Нормальная форма Кодда-Бойса требует, чтобы любой неключевой атрибут функционально зависел только от возможного ключа. Таким образом, нормальная форма Кодда-Бойса запрещает функциональные зависимости между неключевыми атрибутами, но разрешает их между неключевыми атрибутами и альтернативными ключами.

Если сущность имеет только один возможный ключ (он же является первичным ключем), то нормальная форма Кодда-Бойса совпадает с третьей нормальной формой.

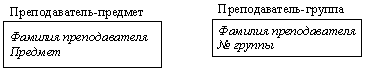
**Четвертая нормальная форма**

Сущность находится в четвертой нормальной форме, если она находится в третьей нормальной форме и между ее атрибутами отсутствуют многозначные зависимости.

Для приведения сущности к четвертой нормальной форме нужно создать новую сущность и поместить атрибуты с многозначной зависимостью в разные сущности. При этом связь между сущностями не устанавливается, поскольку после установления связи атрибуты с многозначной зависимостью путем миграции вновь окажутся в одной сущности. Например:



После приведения к четвертой нормальной форме получим:



**Пятая нормальная форма**

*Определение*. Декомпозицией без потери данных сущности S называется такая совокупность некоторого числа ее проекций, естественное соединение которых полностью восстанавливает исходную сущность S.

Если сущность находится в четвертой нормальной форме, имеет три или более возможных ключа и можно провести ее декомпозицию без потери данных, то приведение такой сущности к пятой нормальной форме заключается в разделении ее на отдельные сущности-проекции для каждого возможного ключа.

Приведение сущности к пятой нормальной форме выполняется достаточно редко, т. к. редко можно найти сущность с тремя или более возможными ключами.

*Общее замечание*. Нетрудно видеть, что нормализация данных сопровождается «размножением» сущностей и связей. При больших объемах данных это приводит к уменьшению производительности, поэтому на практике обычно ограничиваются приведением сущностей к третьей нормальной форме.

### 1.1.6. Целостность данных

Для пользователей информационной системы недостаточно, чтобы база данных просто отражала объекты и процессы предметной области. Важно, чтобы при выполнении любых операций с данными была обеспечена их целостность и непротиворечивость.

Под целостностью данных понимается их правильность в любой момент времени, поэтому поддержание целостности данных может рассматриваться как защита данных от неверного их изменения или разрушения (не путать с незаконным изменением или разрушением, которые предотвращаются системой обеспечения безопасности данных).

Целостность данных определяется:

* целостностью таблиц;
* целостностью внешних ключей;
* целостностью типов данных.

**Целостность таблиц**

Она обеспечивается тем, что все таблицы должны быть снабжены первичными ключами, а поля, входящие в состав первичных ключей, не должны принимать неопределенных или, как их еще называют, Null‑значений.

**Целостность внешних ключей**

Целостность внешних ключей, называемая также *ссылочной целостностью*, обеспечивается тем, что внешние ключи не могут ссылаться на строки, отсутствующие в родительских таблицах. Ссылочную целостность могут нарушить следующие операции над данными:

* обновление первичного ключа в строке родительской таблицы;
* удаление строки в родительской таблице;
* вставка новой строки в дочернюю таблицу;
* обновление внешнего ключа в строке дочерней таблицы.

Для поддержания ссылочной целостности чаще всего используются следующие два *стандартных правила*:

* *ОГРАНИЧИТЬ (RESTRICT) –* не разрешать выполнение операции, приводящей к нарушению ссылочной целостности.
* *КАСКАДИРОВАТЬ (CASCADE) –* разрешить выполнение требуемой операции, но внести при этом необходимые изменения в связанные таблицы так, чтобы не допустить нарушения ссылочной целостности между данными. Изменение начинается в главной (родительской) таблице и каскадно, т.е. поочередно, выполняется в подчиненных таблицах.

*Дополнительными правилами* поддержания ссылочной целостности являются:

* *УСТАНОВИТЬ В NULL (SET NULL)* – разрешить выполнение требуемой операции, однако все некорректные значения внешнего ключа изменить на null-значение.
* *УСТАНОВИТЬ ПО УМОЛЧАНИЮ (SET DEFAULT)* – разрешить выполнение требуемой операции, однако все некорректные значения внешнего ключа изменить на некоторое значение, принятое по умолчанию.

*Замечание*. Дополнительные правила ссылочной целостности могут использоваться только тогда, когда дочерняя таблица является независимой от родительской.

Можно вообще отказаться от поддержания ссылочной целостности с помощью правила

* *ИГНОРИРОВАТЬ (IGNORE)* – разрешить выполнение любых операций, не обращая внимания на нарушения ссылочной целостности.

**Целостность типов данных**

Она обеспечивается тем, что создаются и интегрируются в СУБД новые типы данных для тех доменов логической модели, для которых в физической модели отсутствуют эквивалентные типы данных.

Например, в логической модели атрибут «Экзаменационная оценка» был определен на домене со значениями от 0 до 10, однако практически во всех СУБД наименьшим целым типом является тип «байт» с допустимыми значениями от 0 до 255 или от ‑128 до 127. Чтобы исключить появление в поле «Экзаменационная оценка» значений, меньших 0 или больших 10, нужно создать новый тип данных с диапазоном значений от 0 до 10.

## 1.2. Индивидуальная практическая работа 1 (ИПР1)

### 1.2.1. Методические указания по выполнению ИПР1

**Тема: Логические и физические модели данных**

**Моделирование структур данных в рамках реляционной модели** осложняется невозможностью сразу определить полный список сущностей, связей, атрибутов и определяющих их доменов, а также нужное распределение атрибутов по сущностям, поскольку эти аспекты в процессе проектирования могут многократно уточняться и изменяться. Поэтому различают три уровня логической модели, отличающиеся глубиной представления информации о структуре данных:

* Модель уровня сущностей
* Модель данных, основанная на ключах
* Полная атрибутивная модель.

*Модель уровня сущностей* представляет собой модель данных верхнего уровня, которая отражает основные бизнес-правила предметной области. Модель этого уровня не слишком подробна и включает в себя лишь основные сущности и связи между ними.

*Модель данных, основанная на ключах*, является более подробной и включает в себя все сущности, их первичные ключи, а также связи между сущностями.

*Полная атрибутивная модель* дает наиболее детальное представление о структуре данных и включает все сущности, атрибуты, домены и связи. Сущности, как правило, приведены к третьей нормальной форме.

Эти разновидности моделей, представленные в графической форме, называются диаграммами «сущность-связь» или ER‑диаграммами (Entity – сущность, Relation – связь). Существуют различные варианты ER‑диаграмм, отличающиеся способами графического изображения сущностей и связей. Первый вариант был предложен Питером Ченом в 1976 г. (нотация Чена). Затем появились другие варианты (нотация Мартина, нотация IDEF1X, нотация Баркера и др.).

Впоследствии был создан целый ряд программных средств для автоматизированного построения ER-диаграмм, например: ERwin, SilverRun, Design/IDEF, Power Designer и др. Все они относятся к классу CASE-технологий (Computer-Assisted Systems Engineering – компьютеризированное проектирование систем).

Эти CASE-средства удобны тем, что в них процесс выделения сущностей, связей, а затем и атрибутов, является итерационным (повторяющимся). Разработав первый приближенный вариант ER-диаграммы, далее его уточняют, опрашивая экспертов предметной области, после чего все повторяется. При этом документацией, в которой фиксируются результаты опросов, являются сами ER-диаграммы. Попутно выполняется нормализация данных. Процесс заканчивается получением ER-диаграммы, соответствующей полной атрибутивной модели.

Рассмотрим работу с ER-диаграммами в нотации IDEFIX, используемой в CASE-средстве ERwin. Изучив конкретную предметную область, строим исходную ER‑диаграмму, соответствующую модели уровня сущностей (рис. 1.1).

Клиент размещает Заказ выполняется Сотрудник

Отдел состоит из

Рис. 1.1. ER‑диаграмма модели уровня сущностей

Связи между сущностями на ER‑диаграммах обозначаются следующим образом:

сильная или идентифицирующая связь типа 1 : М

слабая или неидентифицирующая связь типа 1 : М

связь типа М : М

В случае связи 1 : М (один ко многим) одна из связываемых сущностей выступает в роли родительской или главной, а другая – в роли дочерней или подчиненной. Эта связь каждому кортежу родительской сущности ставит в соответствие любое (в том числе нулевое) число кортежей дочерней сущности, однако каждый кортеж дочерней сущности может быть связан только с одним кортежем родительской сущности.

Механизм реализации связи «один ко многим» состоит в том, что в дочернюю сущность добавляются атрибуты, дублирующие ключевые атрибуты родительской сущности. Эти атрибуты получают название *внешнего ключа* (Foreign Key, сокращенно FK) и с их помощью устанавливается связь между кортежами родительской сущности – с одной стороны – и подмножествами кортежей дочерней сущности – с другой. Еще такие атрибуты называют *мигрирующими*из родительской сущности. Если дочерняя сущность является зависимой от родительской сущности, то мигрирующие атрибуты включаются в состав первичного ключа дочерней сущности, в противном случае – в состав ее неключевых атрибутов.

На уровне логической модели возможны также связи типа М : М (многие ко многим), которые используются тогда, когда между атрибутами сущностей существуют многозначные зависимости. Связь такого типа каждому кортежу одной сущности ставит в соответствие любое (в том числе нулевое) число кортежей другой сущности и наоборот.

Возвращаясь к рассмотрению примера, строим на следующем итерационном шаге более полную ER‑диаграмму, соответствующую модели данных, основанной на ключах (рис. 1.2).

Клиент размещает Заказ выполняется Сотрудник

Код клиента

Таб. №

№ заказа

Код клиента (FK)

Таб. № (FK)

Код товара (FK)

№ отдела (FK)

Товар заказывается Отдел состоит из

Код товара

№ отдела

Рис. 1.2. ER‑диаграмма модели, основанной на ключах

Далее, на очередном итерационном шаге, получаем ER‑диаграмму, соответствующую полной атрибутивной модели данных (рис. 1.3).

Клиент Заказ Сотрудник

Таб. №

Код клиента

№ заказа

Код клиента (FK)

Таб. № (FK)

Код товара (FK)

Фамилия

Адрес

Телефон

№ отдела (FK)

Фамилия

Имя

Отчество

размещает выполняется

Дата заказа

Количество

Описание заказа

Товар

Отдел

Код товара

Наименование

Ед. измерения

Цена

ГОСТ

№ отдела

заказывается состоит из

Название

Рис. 1.3. ER‑диаграмма полной атрибутивной модели

Переход от логической модели данных к физической заключается в том, что сущности преобразуются в таблицы, атрибуты и кортежи становятся соответственно столбцами и строками таблиц, домены отображаются в типы данных, принятые в конкретной СУБД. Таблицы, как и сущности, снабжаются первичными и внешними ключами и связываются между собой с помощью связей типа 1:1 и 1:М. Связи типа М:М при переходе к физической модели преобразуются в пары связей типа 1:М и связующие таблицы.

Если логическая модель представлена в виде ER-диаграммы, то переход к физической модели значительно упрощается. В этом случае с помощью CASE-средства, например ERwin, можно выбрать нужную СУБД и автоматически создать соответствующую физическую модель данных. Затем на ее основе ERwin может сгенерировать системный каталог базы данных или соответствующий SQL-скрипт (описание базы данных на языке SQL). Этот процесс называется прямым проектированием.

С другой стороны, ERwin способен по содержимому системного каталога базы данных или SQL-скрипту воссоздать физическую и логическую модели данных. Этот процесс называется обратным проектированием. На основе логической модели, полученной в процессе обратного проектирования, можно сгенерировать физическую модель и системный каталог базы данных для другой СУБД.

**Порядок выполнения работы**

1. Создайте в Erwin новую модель данных (тип модели – Логическая/Физическая, целевая база данных – Access).

2. Создайте сущности логической модели согласно выбранному варианту задания. Каждую сущность снабдите первичным ключом и кратким описанием.

3. Создайте между сущностями связи типа 1 : М (один ко многим).

Связь 1 : М между парой сущностей, одна из которых рассматривается как родительская, а другая – как дочерняя, создается следующим образом: на панели инструментов нужно нажать одну из двух кнопок создания связи 1:М (идентифицирующей – Identifying relationship или неидентифицирующей – Non-identifying relationship), после чего щелкнуть мышью на родительской сущности, а затем – на дочерней. При этом между сущностями появляется связь и в дочернюю сущность будет автоматически добавлены атрибуты, дублирующие ключевые атрибуты родительской сущности. Эти атрибуты получают название внешнего ключа (FK) и будут включены в состав ключевых (идентифицирующая связь) или неключевых (неидентифицируюшая связь) атрибутов дочерней сущности.

4. Если требуется, создайте между сущностями связи типа М : М (многие ко многим).

Эта связь создается следующим образом: на панели инструментов нужно нажать кнопку создания связи М : М (Many-to-many relationship), после чего щелкнуть мышью на одной сущности, а затем – на другой.

5. Дайте имена связям. Для связей 1:М достаточно указать имя, характеризующее отношение или от родительской сущности к дочерней (Parent-to-Child) или от дочерней сущности к родительской (Child-to-Parent). Для связи М:М следует указывать имена как Parent-to-Child так и Child-to-Parent.

6. С помощью команды меню **Model→Subject Areas** откройте окно **Subject Areas** и создайте помимо существующего основного множества **Main Subject Area** еще одно подмножество модели: **Subject Area 1**, включив (вкладка **Members**) в него некоторую сущность со всеми непосредственно связанными с ней (Level = 1) сущностями. После этого просмотрите поочередно обе модели, используя для переключения между ними кнопку с выпадающим списком **Create Subject Area**, расположенную на панели инструментов.

7. С помощью команды меню **Format→Stored Display Settings** откройте окно **Stored Displays** и создайте для подмножества модели **Subject Area 1** два новых хранимых отображения (**Display2** и **Display3**), отличающиеся друг от друга и от исходного отображения **Display1** расположением частей логической модели на экране, масштабом, уровнями просмотра этой модели, видимостью отдельных компонентов сущностей и связей, а также другими характеристиками.

8. Опробуйте варианты показа на диаграмме различных уровней логической модели (команды меню: **Format→Display Level→…**) и различных способов отображения атрибутов в сущностях (команды меню: **Format→Entity Display→…**).

9. С помощью команды меню **Model→Key Groups** откройте окно **Key Groups** и, используя кнопку **New**, создайте для сущностей дополнительно к имеющемуся первичному ключу еще и возможные альтернативные ключи (Alternate Key) и возможные инверсионные входы (Inversion Entry). Включите режим их отображения на диаграмме.

10. Переключившись в режим показа физической модели, проверьте и, в случае необходимости, скорректируйте типы и длины полей, ориентируясь на специфику типов данных СУБД Access. Например, поле **Цена** должно иметь тип **Currency**, а поле **Количество** - тип **Single**.

11. В физической модели с помощью команды меню **Model→Columns** откройте окно **Columns**, перейдите на вкладку **Constraint** и задайте для тех полей, где это возможно, правила валидации (Validation Constraint) и/или значения по умолчанию (Default). Например, для полей **Цена** и **Количество** правило валидации заключается в выборе только неотрицательных значений, что обеспечивается вводом выражения: **>= 0.** Значением по умолчанию для поля **ЕдиницаИзм** будет "**штука**"; для поля **ДатаЗаказа** – **Date()**, т.е. текущая дата и т.д.

12. Если между таблицами имеется связь типа **М:М**, то в окне **Model Properties** (вызывается командой меню **Model→Model Properties**) на вкладке **General** установите флажок **Auto apply Supertype-Subtype Identity transform**. В физической модели выделите мышью связь М:М, после чего нажмите кнопку **Many to Many Transform** на панели инструментов и запустите мастер преобразования связи М:М. В результате эта связь будет преобразована в две связи 1:М путем создания дополнительной связующей таблицы. При этом в логической модели преобразований не будет и останется прежняя связь М:М.

*Примечание 1*. Если в окне **Model Properties** на вкладке **General** не устанавливать флажок **Auto apply Supertype-Subtype Identity transform**, то указанное выше преобразование связи будет выполнено также и в логической модели.

*Примечание 2*. Если в окне **Model Properties** на вкладке **General** установить флажок **Auto apply Many-to-Many transform**, то Erwin будет автоматически выполнять упомянутую выше трансформу связи М:М при переключении с логической модели на физическую.

13. Сохраните текущую модель данных на жестком диске.

14. Запустите СУБД Access и создайте новую пустую базу данных, после чего закройте Access.

15. В ERwin из режима показа физической модели с помощью команды меню **Tools→Forward Engineer/Schema Generation** (или соответствующей кнопки на панели инструментов) откройте окно **Forward Engineer Schema Generation** и нажмите кнопку **Generate**. В появившемся окне **Access Connection** задайте имя пользователя (User Name) равным **Admin**, а также с помощью кнопки **Browse** (первой сверху) задайте полное имя созданной базы данных. Далее нажмите кнопку **Connect** и выполните процесс прямого проектирования (Forward Engineer) с наполнением файла базы данных метаданными согласно созданной физической модели данных. После завершения процесса прямого проектирования с помощью команды меню **Database→Database Connection** откройте окно **Access Connection** и разорвите соединение с базой данных путем нажатия кнопки **Disconnect**.

16. Запустите СУБД Access, откройте базу данных и просмотрите структуры полученных таблиц и наличие в них первичных ключей. Убедитесь, что для конкретных полей установлены свойства «Условие на значение» и «Значение по умолчанию», которые соответствуют правилам валидации (Valid) и значениям по умолчанию (Default) в физической модели данных.

17. Путем ввода данных в таблицы убедитесь, что они снабжены уникальными индексами, соответствующими альтернативным ключам в логической модели данных.

### 1.2.2. Варианты заданий ИПР1

Логическую и физическую модели данных необходимо разработать применительно к одному из следующих направлений деятельности:

1. Основные средства предприятия.

2. Банковские операции на предприятии.

3. Авансовые отчеты и бухгалтерские справки на предприятии.

4. Работа с доверенностями на предприятии.

5. Снабжение предприятия.

6. Отгрузка продукции на предприятии.

7. Отпуск в производство и выпуск продукции на предприятии.

8. Кассовые операции на предприятии.

9. Перемещение и списание материальных ценностей на предприятии.

10. Управление договорной деятельностью предприятия.

11. Управление работой нефтебазы.

12. Банки Беларуси.

13. Управление телефонной станцией.

14. Управление аптекой.

15. Управление поликлиникой.

16. Работа с недвижимостью.

17. Работа банков с расчетными счетами.

18. Работа банков с электронными карточками.

19. Управление работой АЗС.

20. Управление железнодорожными перевозками.

21. Управление автобусным движением.

22. Управление библиотечным фондом.

23. Животный и растительный мир Беларуси.

24. Климатическая карта Беларуси.

25. Отдел кадров предприятия.

26. Работа деканата в вузе.

27. Сеть магазинов определенного профиля.

28. Управление таксопарком.

29. Продажа и бронирование авиабилетов.

30. Учет товаров на складе.

**Указания по выбору варианта**

ИПР1 содержит 30 вариантов заданий. Выбор варианта, в соответствии с которым необходимо выполнить ИПР1, производится следующим образом.

Если две последние цифры номера вашей зачетки образуют число N, не превышающее 30, то это число и будет номером вашего варианта. Если же число N будет больше тридцати, то необходимо из N последовательно вычитать число 30 до тех пор, пока полученный остаток не окажется меньшим или равным числу 30. Этот остаток и будет представлять собой ваш номер варианта, по которому нужно выполнить ИПР1.

Например, две последние цифры номера вашей зачетки представляют собой число N = 72. В этом случае N > 30 и, поэтому, придется дважды вычитать из числа N число 30 для получения остатка, не превосходящего число 30. Этим остатком будет число 12, которое и будет вашим номером варианта.

**Порядок оформление итогового отчета**

Итоговый отчет формируется с помощью текстового редактора Word, и, помимо титульного листа и листа содержания, должен включать в себя:

1) описание конкретного варианта задания, подлежащего решению;

2) перечень конкретных действий по конструированию логической и физической моделей данных в соответствии с выполняемым вариантом задания;

3) скриншоты (копии экрана монитора), иллюстрирующие результат основных действий по конструированию моделей данных (логической и физической);

4) описание процесса прямого проектирования со скриншотами, отображающими: окно конструктора одной из таблиц, вид открытой таблицы с данными, окно схемы данных;

5) выводы по работе;

6) список используемой литературы.

## 1.3. Промежуточный контрольный тест

1. Сколько этапов составляют жизненный цикл базы данных?

2. В каких моделях данных обеспечивается независимость по данным (иерархической, сетевой, реляционной, объектно-реляционной)?

3. Сопоставьте, какие термины логической и физической моделей соответствуют друг другу.

4. Можно ли значения атрибутов, связанных с разными доменами, сравнивать друг с другом?

5. Каким двум условиям должен удовлетворять возможный ключ?

6. Сколько первичных ключей может иметь сущность?

7. Может ли альтернативный ключ быть частью первичного ключа этой же сущности?

8. Можно ли разработать алгоритм нахождения возможного ключа сущности?

9. Сколько всего имеется базовых реляционных операций?

10. Как выражаются степень и мощность декартова произведения сущностей через степени и мощности сущностей-операндов?

11. Сколько имеется разновидностей у реляционной операции соединения?

12. Какое слово обычно содержат в своей формулировке задачи, решаемые с помощью операции деления?

13. Сколько операций реляционной алгебры являются примитивными, т.е. не выражаются через другие операции?

14. Может ли кортеж родительской сущности не ссылаться ни на один кортеж дочерней сущности после установления связи типа «один ко многим»?

15. Может ли внешний ключ быть частью некоторого возможного ключа дочерней сущности?

16. Приведение к какой нормальной форме устраняет в сущности повторяющиеся группы атрибутов?

17. Что является причиной наличия в сущности аномалий вставки, обновления и удаления?

18. Какими способами обеспечивается непротиворечивость данных, хранящихся в базе данных?

19. При разработке базы данных за счет чего обеспечивается целостность: а)таблиц; б) внешних ключей; в) типов данных?

## 1.4. Правильные ответы на промежуточный контрольный тест

1. Сколько этапов составляют жизненный цикл базы данных?

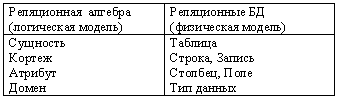
Ответ: 7.

2. В каких моделях данных обеспечивается независимость по данным (иерархической, сетевой, реляционной, объектно-реляционной)?

Ответ: реляционной, объектно-реляционной.

3. Сопоставьте, какие термины логической и физической моделей соответствуют друг другу.

Ответ:



4. Можно ли значения атрибутов, связанных с разными доменами, сравнивать друг с другом?

Ответ: нет.

5. Каким двум условиям должен удовлетворять возможный ключ?

Ответ: 1) уникальность;

2) минимальность.

6. Сколько первичных ключей может иметь сущность?

Ответ: 1.

7. Может ли альтернативный ключ быть частью первичного ключа этой же сущности?

Ответ: нет.

8. Можно ли разработать алгоритм нахождения возможного ключа сущности?

Ответ: нет.

9. Сколько всего имеется базовых реляционных операций?

Ответ: 8.

10. Как выражаются степень и мощность декартова произведения сущностей через степени и мощности сущностей-операндов?

Ответ: степень = сумме степеней; мощность = произведению мощностей.

11. Сколько имеется разновидностей у реляционной операции соединения?

Ответ: 4.

12. Какое слово обычно содержат в своей формулировке задачи, решаемые с помощью операции деления?

Ответ: «все».

13. Сколько операций реляционной алгебры являются примитивными, т.е. не выражаются через другие операции?

Ответ: 5.

14. Может ли кортеж родительской сущности не ссылаться ни на один кортеж дочерней сущности после установления связи типа «один ко многим»?

Ответ: да.

15. Может ли внешний ключ быть частью некоторого возможного ключа дочерней сущности?

Ответ: да.

16. Приведение к какой нормальной форме устраняет в сущности повторяющиеся группы атрибутов?

Ответ: к первой.

17. Что является причиной наличия в сущности аномалий вставки, обновления и удаления?

Ответ: в ней хранится разнородная информация.

18. Какими способами обеспечивается непротиворечивость данных, хранящихся в базе данных?

Ответ: выполнением процедур нормализации данных.

19. При разработке базы данных за счет чего обеспечивается целостность: а)таблиц; б) внешних ключей; в) типов данных?

Ответ:

а) все таблицы должны быть снабжены первичными ключами, а поля, входящие в их состав, не должны принимать Null-значений;

б) внешние ключи не должны ссылаться на строки, отсутствующие в родительских таблицах;

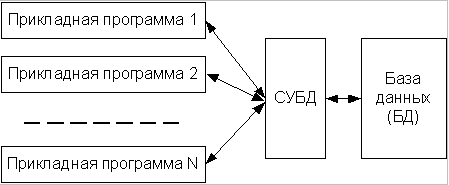
в) создаются новые типы данных для тех доменов логической модели, для которых в физической модели отсутствуют эквивалентные типы данных.

# 2. Модуль 2. Физическое проектирование

## 2.1. Теоретический материал

### 2.1.1. Назначение и функции СУБД

Упрощенная система баз данных, предполагающая, что вся информация хранятся в одной базе данных, выглядит следующим образом.



Между собственно физической базой данных и пользователями системы располагается уровень программного обеспечения, который называется системой управления базами данных (СУБД). Все запросы пользователей на получение доступа к базе данных обрабатываются СУБД.

Основная задача СУБД – дать пользователю базы данных возможность работать с ней, не вникая во все подробности работы на уровне аппаратного обеспечения. Поэтому СУБД предоставляет в распоряжение конечному пользователю набор операций, выражаемых в терминах языка запросов высокого уровня, примером которого является язык SQL.

Необходимо отметить, что СУБД – это наиболее важный, но не единственный программный компонент системы. В числе других компонентов можно назвать утилиты, средства разработки приложений, средства проектирования, генераторы отчетов, диспетчер обработки транзакций и др.

Конечный пользователь может получать доступ к базе данных, применяя одно из приложений, или же интерфейс, интегрированный в программное обеспечение самой СУБД. Большинство СУБД включают в себя по крайней мере одно такое встроенное приложение, а именно – процессор языка запросов, позволяющий пользователю в диалоговом режиме вводить запросы (команды) к базе данных. Кроме языка запросов, в большинстве СУБД дополнительно предоставляются специализированные встроенные интерфейсы, используя которые пользователь работает с базой данных за счет выбора необходимых элементов меню или заполнения требуемых полей в представленных формах.

СУБД выполняет целый ряд функций, большинство из которых для конечного пользователя незаметны. Важнейшими функциями СУБД являются следующие.+

**1. Управление словарем данных**

Словарь данных (другие названия: репозитарий данных, энциклопедия данных) можно считать самостоятельной базой данных, но не пользовательской, а системной. Словарь содержит «данные о данных», называемые также метаданными или дескрипторами, т.е. в нем содержатся определения других объектов системы, а не просто обычные данные. В частности в словаре данных должны содержаться описания структур и свойств данных и их взаимосвязей, установленные ограничения целостности данных, список авторизованных пользователей и их полномочия доступа и т.д.

Так как любые программы получают доступ к базе данных через посредство СУБД, то использование словаря данных упрощает разработку прикладных программ за счет использования уже имеющихся стандартов организации и именования данных.

**2. Управление хранением, преобразованием и представлением данных**

СУБД создает сложные структуры, необходимые для хранения данных, освобождая пользователя от знания деталей их физического представления в памяти компьютера. Кроме того СУБД оптимизирует и преобразует запросы пользователей в команды, обеспечивающие определение физического местоположения нужной информации, ее извлечение, обработку и форматирование для придания вида, удобного для чтения.

Современные СУБД обеспечивают хранение не только данных, но и связанных с ними экранных форм, схем отчетов, правил проверки данных, кода хранимых процедур и функций, систем обработки видео, форматов изображений и многое другое.

**3. Обеспечение безопасности данных**

СУБД создает систему безопасности, которая обеспечивает защиту и конфиденциальность пользовательских данных, содержащихся в базе данных.

Правила безопасности устанавливают, какие пользователи могут получить доступ к базе данных, а также какие операции разрешено выполнять тому или иному пользователю над конкретными объектами базы данных. Такие разрешенные операции называются привилегиями. Установка, контроль и снятие привилегий – это прерогатива администратора базы данных. Дополнительные возможности связаны с назначением пользователям определенных ролей, по аналогии с включением пользователей в группы пользователей операционной системы Windows.

**4. Обеспечение целостности данных**

В СУБД предусмотрены средства обеспечения целостности данных, что позволяет постоянно поддерживать базу данных в актуальном состоянии. Обеспечение целостности данных с помощью СУБД основано на использовании механизма транзакций при выполнении операций с данными.

Транзакция – это последовательность операций с базой данных, которая рассматривается как некоторое неделимое действие, осмысленное с точки зрения пользователя. Можно сказать, что транзакция – это логическая единица работы системы, которая реализует некоторую прикладную функцию.

Если все операции, составляющие транзакцию, успешно выполняются, то СУБД фиксирует в базе данных изменения, произведенные этой транзакцией. В этом случае говорят, что транзакция подтверждается или фиксируется. Если хотя бы одна из операций не будет выполнена, то база данных останется в том же состоянии, что и перед началом выполнения транзакции, т.е. все операции, составляющие транзакцию, отменяются и транзакция, соответственно, не подтверждается или, как еще говорят, откатывается.

Таким образом, все операции с базой данных выполняются только в рамках транзакций, каждая из которых переводит базу данных из одного целостного состояния в другое.

**5. Управление многопользовательским доступом к данным**

СУБД включает в себя средства, обеспечивающие доступ к данным нескольким пользователям одновременно без нарушения целостности данных. В основе этого лежат механизмы транзакций и блокировок.

Упрощенно механизм блокировок заключается в том, что если несколько пользователей хотят одновременно изменить одни и те же данные, то только транзакция одного пользователя получит доступ к общему ресурсу, а остальные транзакции будут ждать пока она не завершится и не разблокирует этот ресурс, после чего начнет выполняться другая транзакция и т.д.

Транзакции и блокировки тесно связаны друг с другом. Блокировки являются тем механизмом, который обеспечивает независимость транзакций друг от друга. Однако использование блокировок существенно замедляет процесс обработки транзакций из-за необходимости ожидания, когда освободятся данные, захваченные конкурирующей транзакцией. Эти задержки можно уменьшить, уменьшая размеры фрагментов данных, захватываемых транзакциями. В частности, можно блокировать сразу всю базу данных (очевидно, что это неприемлемый вариант), таблицу базы данных, часть таблицы (страницу), отдельную строку таблицы. Все эти варианты известны под названием *уровни блокировки*.

Большинство СУБД используют блокировки на уровне страниц и/или на уровне строк. Так как размер страниц сравнительно невелик (2,4,8, реже 16 Кб), то время ожидания транзакций, конкурирующих за доступ к страницам, оказывается вполне приемлемым. Наивысшую производительность обеспечивают блокировки на уровне отдельных строк.

**6. Управление резервным копированием и восстановлением данных**

СУБД содержит специальные утилиты, с помощью которых администратор базы данных может выполнять регулярные и экстренные процедуры резервного копирования и восстановления данных. Восстановление данных производится в следующих случаях:

* после аварийного отказа аппаратуры (жесткий сбой системы), например, в случае появления сбойного сектора на жестком диске или поломки магнитных головок дисковых накопителей. В этом случае данные повреждаются физически;
* после аварийного отказа программного обеспечения (мягкий сбой системы), например, в результате аварийного отключения электрического питания или в результате появления неустранимого сбоя процессора. Мягкий сбой характеризуется утратой оперативной памяти системы, но данные, хранящиеся на диске, остаются неповрежденными.

Для восстановления данных необходимо, чтобы они сохранялись в базе данных с некоторой избыточностью, позволяющей восстановить то состояние базы данных, которое было непосредственно перед сбоем. Такую избыточность обеспечивают полная архивная копия базы данных и журнал транзакций.

Журнал транзакций содержит перечень всех транзакций, выполненных после создания последней архивной копии базы данных, с описанием всех деталей каждой транзакции.

Полная архивная копия базы данных создается периодически с учетом скорости наполнения журнала транзакций и, как правило, размещается на другом сетевом диске.

Для восстановления данных, поврежденных в результате жесткого сбоя системы, используют последнюю по времени полную архивную копию базы данных и журнал транзакций. Восстановление данных состоит в том, что исходя из полной архивной копии по журналу транзакций последовательно воспроизводится работа всех подтвержденных транзакций. В результате база данных восстанавливается в том состоянии, в котором она была непосредственно перед повреждением.

Восстановление данных, поврежденных в результате мягкого сбоя системы, осуществляется как *часть процедуры перезагрузки* системы. При перезагрузке системы транзакции подвергаются анализу для выявления завершившихся транзакций и транзакций, прерванных из-за возникновения сбоя. Транзакции, подтвержденные до наступления сбоя, но результат выполнения которых не записан на диск, выполняются заново. Транзакции, не завершившиеся из-за сбоя, и транзакции, результат выполнения которых записан на диск, откатываются. В результате данные оказываются восстановленными.

Самым плохим случаем является ситуация, когда разрушены физически и база данных, и журнал транзакций. Единственное, что можно сделать в этом случае - это восстановить состояние базы данных на момент последнего резервного копирования. Чтобы не допустить возникновение такой ситуации, базу данных и журнал транзакций необходимо размещать на *физически* разных дисках, управляемых физически разными контроллерами.

Аппаратные способы повышения надежности системы:

* Резервный сервер. Эта технология предполагает наличие с сети дополнительного сервера, на котором дублируется вся информация и который можно использовать в случае выхода из строя основного сервера.
* Кластер. Эта технология предполагает создание т.н. виртуального сервера, к которому могут обращаться пользователи. На самом деле виртуальный сервер состоит из нескольких компьютеров (чаще всего от 2 до 4) и называется кластером. При выходе из строя одного компьютера кластер будет продолжать принимать и обрабатывать пользовательские запросы. Кластер остается работоспособным, пока в нем есть хотя бы один исправный компьютер.
* Дисковый массив RAID (Redundant Array of Independent Disks). Эта технология обеспечивает безаварийную работу системы в случае выхода из строя любого из жестких дисков, входящих в массив RAID.

**7. Наличие механизмов тиражирования (репликации) данных**

Работа распределенной базы данных базируется на синхронной фиксации изменений сразу на нескольких узлах системы, что предъявляет жесткие требования к производительности и надежности каналов связи. Реальной альтернативой этой технологии является технология тиражирования данных, не требующая синхронной фиксации изменений. Действительно, далеко не всегда требуется поддерживать идентичность баз данных на различных узлах в любое время – достаточно это делать лишь в определенные, критичные, моменты времени. Следовательно, можно накапливать изменения на одном узле в виде последовательности транзакций и периодически копировать эти изменения на другие узлы системы. Эти функции выполняет специальный модуль СУБД, называемый репликатором. Его задача — автоматически синхронизировать базы данных распределенной системы и, тем самым, обеспечивать актуальность их копий, или, как их еще называют, реплик.

**8. Наличие возможностей экспорта и импорта данных**

При экспорте данных СУБД выполняет их преобразование в формат, понятный внешней системе. Например, при передаче данных в MS Exel будет получен файл с расширением **xls**. При импорте данных СУБД создает соответствующие структуры данных, предназначенные для размещения в них импортируемой информации.

Таким образом, наличие в СУБД возможностей экспорта и импорта данных позволяет организовать обмен данными между множеством разнородных (гетерогенных) хранилищ информации.

**9. Наличие языков доступа к данным и интерфейсов прикладного программирования**

Современные СУБД обеспечивают доступ к данным с помощью языка структурированных запросов SQL из различных сред программирования, таких как Visual C++, Visual Basic, Java, Delphi, C++ Builder и др., с использованием различных механизмов доступа к данным, таких как ODBC, ADO, OLE DB, JDBC, BDE и др.

В составе СУБД имеются также специальные утилиты, предназначенные для администраторов и проектировщиков базы данных, для целей внедрения, текущего контроля и обслуживания баз данных.

**10. Наличие интерфейсов взаимодействия с базой данных**

Современные СУБД обеспечивают разнообразные возможности для взаимодействия конечных пользователей с базой данных. Например, для доступа к данным в качестве внешнего интерфейса можно использовать Web-браузеры или системы электронной почты.

### 2.1.2. Типы баз данных

В зависимости от количества пользователей базы данных подразделяются на однопользовательские, т.е. обслуживающие в любой момент времени только одного человека, и многопользовательские, которые могут обслуживать сразу многих пользователей.

Если однопользовательская база данных развернута на персональном компьютере, то ее называют настольной базой данных.

Если многопользовательская база данных обслуживает относительно небольшое число пользователей, то она называется базой данных рабочей группы. Если же база данных используется в рамках всего предприятия, то она называется базой данных предприятия или корпоративной базой данных.

Если база данных размещается на одном сервере, то она называется централизованной, а если на нескольких серверах – то распределенной.

В зависимости от способа применения базы данных можно грубо разбить на операционные и базы данных для систем поддержки принятия решений.

Операционные базы данных предназначены для оперативного выполнения важнейших каждодневных операций, таких как продажа товаров и услуг, различные платежи, закупка материальных ценностей, резервирование авиабилетов и мест в гостиницах и т.д. Результат таких операций фиксируется и вступает в силу немедленно.

Базы данных для систем поддержки принятия решений предназначены в основном для получения необходимой информации при выработке стратегических или тактических решений на уровне высшего и среднего руководства предприятия. При этом выполняется широкомасштабная обработка данных, накопленных за достаточно продолжительное время, для извлечения из них нужной информации.

## 2.2. Промежуточный контрольный тест

1. Что такое словарь данных?

2. Что дает использование пользовательских ролей в системе обеспечения безопасности данных?

3. С помощью какого механизма в СУБД обеспечивается целостность данных?

4. С помощью каких механизмов в СУБД обеспечивается управление многопользовательским доступом к данным?

5. Укажите, какие единицы хранения данных соответствуют используемым в СУБД уровням блокировки?

6. Что нужно использовать для восстановления данных, разрушенных в результате жесткого сбоя системы?

7. Что будет утрачено в результате возникновения мягкого сбоя системы?

8. Какие имеются аппаратные способы повышения надежности системы?

9. С помощью какого механизма можно синхронизировать данные, размещенные на узлах распределенной базы данных (т.е. устранить в них различия)?

10. С помощью какого механизма можно организовать обмен данными между множеством разнородных (гетерогенных) хранилищ информации?

## 2.3. Правильные ответы на промежуточный контрольный тест

1. Что такое словарь данных?

Ответ: Словарь данных – это системная база данных, которая содержит метаданные («данные о данных»), а не просто обычные данные.

2. Что дает использование пользовательских ролей в системе обеспечения безопасности данных?

Ответ: на уровне роли можно легко менять привилегии пользователей-членов роли.

3. С помощью какого механизма в СУБД обеспечивается целостность данных?

Ответ: с помощью механизма транзакций.

4. С помощью каких механизмов в СУБД обеспечивается управление многопользовательским доступом к данным?

Ответ: с помощью механизма транзакций и механизма блокировок.

5. Укажите, какие единицы хранения данных соответствуют используемым в СУБД уровням блокировки?

Ответ: база данных, **страница**, сегмент, столбец, **строка**, ячейка.

(правильные ответы выделены полужирным шрифтом с подчеркиванием)

6. Что нужно использовать для восстановления данных, разрушенных в результате жесткого сбоя системы?

Ответ: полную архивную копию базы данных и журнал транзакций.

7. Что будет утрачено в результате возникновения мягкого сбоя системы?

Ответ: данные в оперативной памяти системы.

8. Какие имеются аппаратные способы повышения надежности системы?

Ответ: резервный сервер, кластер, дисковый массив RAID.

9. С помощью какого механизма можно синхронизировать данные, размещенные на узлах распределенной базы данных (т.е. устранить в них различия)?

Ответ: с помощью механизма тиражирования (репликации) данных.

10. С помощью какого механизма можно организовать обмен данными между множеством разнородных (гетерогенных) хранилищ информации?

Ответ: с помощью механизма экспорта и импорта данных.

# 3. Модуль 3. Язык структурированных запросов SQL

## 3.1. Теоретический материал

### 3.1.1. Введение

Язык SQL (Structured Query Language) был разработан фирмой IBM в конце 70-х годов прошлого столетия и вскоре стал общепризнанным средством выполнения любых операций с реляционными базами данных независимо от применяемых аппаратных и программных платформ. Первый международный стандарт языка был принят Международной организацией по стандартизации ISO (**I**nternational **S**tandards **O**rganization) в 1989 г., а затем более полные – в 1992, 1999, 2003 годах.

В настоящее время все производители реляционных СУБД поддерживают стандарт SQL92, расширяя его, однако, новыми возможностями, снимающими ряд ограничений, присущих стандарту. Это привело к появлению диалектов SQL, например, фирма Microsoft предлагает в своих программных продуктах языки Jet/SQL и Transact/SQL, фирма Oracle – PL/SQL, фирма Informix –Informix/SQL и т.д.

Мы ограничимся рассмотрением стандарта SQL92, причем его простейшего варианта – интерактивного SQL, где команда вводится, выполняется и сразу же выдается результат ее выполнения, если он есть.

Все команды языка SQL можно грубо разбить на две категории: команды определения данных или, по-другому, команды языка DDL (Data Definition Language) и команды манипулирования данными или команды языка DML (Data Manipulation Language).

Команды определения данных работают с метаданными и предназначены для создания и модификации объектов базы данных, а также для предоставления и отмены привилегий доступа к базе данных и ее объектам. Команды манипулирования данными позволяют работать с хранящейся в базе данных информацией, т.е. выполнять операции: вставки, обновления, удаления данных, а также а также выборку их из базы данных по различным критериям.

Команды SQL мы будем изучать применительно к базе данных «Сессия», которая задается следующей схемой данных.

|  |
| --- |
| Группа |
| N\_группы  состо-  ит из |
| факультет |

Экзамен

|  |
| --- |
| Студент |
| N\_зачетки |
| фамилия  сдает  год\_рождения  год\_поступления N\_группы (FK) |

оценка

семестр

предмет

N\_зачетки (FK) (FK)

### 3.1.2. Команды манипулирования данными

**1. Вставка новой записи в таблицу.**

INSERT INTO <имя таблицы> [(<имя поля> [, <имя поля> …])]

VALUES (<значение> [, <значение> …]);

Эта команда состоит из двух предложений и завершается символом «;». Прописными буквами записаны ключевые (зарезервированные) слова языка, хотя это и не обязательно. Ключевые слова запрещается использовать в качестве имен объектов базы данных. Команда записана в общем виде с применением элементов системы обозначений, известной как нотация Бэкуса-Наура. Например:

INSERT INTO Группа

VALUES (472301, ‘ИЭ’);

INSERT INTO Студент (N\_зачетки, фамилия, N\_группы)

VALUES (12345, ‘Иванов’, 472301);

Последнюю команду можно записать другим способом:

INSERT INTO Студент

VALUES (12345, ‘Иванов’, NULL, NULL, 472301);

В языке SQL зарезервированное слово NULL используется для указания того, что значение поля неизвестно.

**2. Удаление записей из таблицы.**

DELETE FROM <имя таблицы>

[WHERE <предикат>];

Предикат – это выражение логического типа, которое может принимать одно из двух значений: TRUE или FALSE (соответственно «истина» или «ложь»). Предикат задает условие отбора данных: те записи, для которых значение предиката есть истина, будут удалены. Например:

DELETE FROM Группа;

(из таблицы «Группа» будут удалены все записи; сама таблица остается, но она будет пустой)

Теперь предположим, что какой-то студент отчислен из ВУЗа и нужно удалить относящиеся к нему данные из таблиц «Студент» и «Экзамен». Если для поддержания ссылочной целостности используется правило ОГРАНИЧИТЬ (RESTRICT), то данные сначала надо удалить из дочерней таблицы «Экзамен» и только затем – из родительской таблицы «Студент» (наоборот не получится):

.

DELETE FROM Экзамен

WHERE N\_зачетки = 12345;

(из дочерней таблицы удаляются все записи, относящиеся к данному студенту)

DELETE FROM Студент

WHERE N\_зачетки = 12345;

(из родительской таблицы удаляется запись, относящаяся к данному студенту)

Если же для поддержания ссылочной целостности используется правило КАСКАДИРОВАТЬ(CASCADE), то можно сразу удалить запись из родительской таблицы «Студент», а все зависимые от нее записи в дочерней таблице «Экзамен» удалятся автоматически.

Предикаты, помимо имен полей и констант, обычно включают в себя операции сравнения: =, <>, <, <=, >, >= и логические связки NOT, AND, OR (соответственно НЕ, И, ИЛИ), которые перечислены в порядке убывания их старшинства. В свою очередь операции сравнения имеют более высокий приоритет, чем логические связки, что часто позволяет обходиться без использования круглых скобок. В остальном, построение сложных предикатов подчиняется стандартным правилам, принятым в большинстве языков программирования.

Рассмотрим предикат, в котором скобки необходимы:

WHERE (фамилия = ‘Иванов’ OR фамилия = ‘Петров’) AND (N\_группы = 472301 OR N\_группы = 472302 OR N\_группы = 472303)

(выбираются все Ивановы и Петровы, которые учатся в группах 472301, 472302, 472303)

Если здесь скобки опустить, то предикат, с учетом старшинства логических связок, станет эквивалентным следующему предикату:

WHERE фамилия = ‘Иванов’ OR (фамилия = ‘Петров’ AND N\_группы = 472301) OR N\_группы = 472302 OR N\_группы = 472303

(а этот предикат, в сравнении с исходным, задает совсем другое условие отбора данных)

Предикаты могут включать в себя операторы IN и BETWEEN. Первый из них задает множество значений, второй – интервал значений, которые используются в качестве критериев отбора данных.

С использованием этих операторов рассмотренный выше предикат можно переписать так:

WHERE фамилия IN (‘Иванов’, ‘Петров’) AND N\_группы BETWEEN 472301 AND 472303

(оператор BETWEEN всегда используется со связкой AND, разделяющей граничные значения интервала)

Для текстовых полей в предикатах может использоваться оператор LIKE, который задает строку символов в качестве критерия отбора данных. Эта строка может включать в себя метасимволы «%» и «\_». Первый из них обозначает любое (в том числе нулевое) число любых символов, а второй – один любой символ. Например:

WHERE фамилия LIKE ‘А%’

WHERE фамилия LIKE ‘Я\_\_о%ий’ AND N\_группы = 472301

WHERE фамилия LIKE ‘И%ов’ OR фамилия LIKE ‘П%ов’

Зарезервированное слово NULL используется в предикатах всегда со связкой IS, например:

DELETE FROM Студент

WHERE N\_группы IS NULL AND фамилия IS NOT NULL;

(нельзя ставить знак «=» вместо связки IS)

**3. Обновление записей в таблице.**

UPDATE <имя таблицы>

SET <имя поля> = <выражение> [, <имя поля> = < выражение> …]

[WHERE <предикат>];

Эта команда включает в себя три предложения, первые два из которых – обязательны. Конструкция «выражение» может включать в себя константы, имена полей и знаки арифметических операций. Однако чаще всего это просто константа. Примеры:

UPDATE Экзамен

SET оценка = 10;

(обновление всех записей в таблице «Экзамен»)

UPDATE Экзамен

SET оценка = 10

WHERE семестр = 3 AND предмет = 'Теория систем' AND N\_зачетки = 12345;

(обновление конкретной оценки конкретному студенту)

UPDATE Экзамен

SET оценка = оценка + 1

WHERE предмет = 'Теория систем' AND оценка < 10;

**4. Выборка данных из таблиц.**

В отличии от предыдущих команд, которые не возвращали результатов, команда на выборку возвращает результирующий набор данных.

SELECT [DISTINCT] { \* | <имя поля> [, <имя поля> ...]}

FROM <имя таблицы> [, <имя таблицы> …]

[WHERE <предикат>]

[GROUP BY <имя поля> [, <имя поля> ...]]

[HAVING <предикат>]

[ORDER BY <имя поля> [, <имя поля> ...]];

Простейшая команда на выборку данных будет, например, такой:

SELECT \*

FROM Студент;

(результирующий набор данных будет совпадать с таблицей «Студент»)

SELECT фамилия, год\_ рождения

FROM Студент;

(к таблице «Студент» применили вертикальный фильтр: выводятся все строки, но только 2 столбца)

SELECT \*

FROM Студент

WHERE N\_группы = 472301;

(к таблице «Студент» применили горизонтальный фильтр: выводится часть строк и все столбцы)

SELECT фамилия, год\_ рождения

FROM Студент

WHERE N\_группы = 472301;

(к таблице «Студент» применили и вертикальный и горизонтальный фильтры)

С помощью зарезервированного слова DISTINCT устраняются повторы строк в результирующем наборе данных. Например:

SELECT DISTINCT год\_ рождения

FROM Студент

WHERE N\_группы = 472301;

(выводятся без повтора годы рождения студентов в группе 472301)

В предложении ORDER BY указываются поля, по которым выполняется сортировка строк результирующего набора данных. Поля указываются в порядке убывания их приоритета. С помощью зарезервированных слов ASC и DESC можно указать соответственно возрастающий или убывающий порядок сортировки для каждого поля, входящего в предложение ORDER BY. По умолчанию выполняется сортировка по возрастанию. Например:

SELECT \*

FROM Студент

ORDER BY год\_рождения, фамилия;

(выводятся все строки таблицы «Студент», упорядоченные по году рождения и фамилии)

*Использование агрегатных функций.*

Агрегатные функции предназначены для подсчета итоговых значений применительно ко всему набору данных, определяемому командой выборки данных. Имеются следующие функции:

COUNT – подсчитывается число строк или значений поля, отличных от NULL

SUM – подсчитывается суммарное значение

AVG – подсчитывается среднее значение

MAX – находится максимальное значение

MIN – находится минимальное значение

SELECT COUNT(\*), COUNT(год\_рождения), COUNT(DISTINCT год\_рождения)

FROM Студент

WHERE N\_группы = 472301;

(результирующая строка содержит одну строку и 3 столбца: число студентов в группе; число студентов, у которых поле «год\_рождения» заполнено; число различных годов рождения в группе)

SELECT MIN(оценка), AVG(оценка), MAX(оценка), SUM(оценка)

FROM Экзамен

WHERE семестр = 3 AND N\_зачетки = 12345;

(результирующая строка содержит итоги экзаменов за 3-й семестр для конкретного студента)

*Использование вычисляемых полей.*

Вычисляемые поля – это поля, содержащиеся в результирующем наборе данных, но отсутствующие в таблицах и представлениях базы данных. Например:

SELECT фамилия, год\_поступления - год\_рождения

FROM Студент

WHERE N\_группы = 472301;

(выдается фамилия и подсчитывается возраст поступления в ВУЗ для каждого студента группы 472301)

С помощью зарезервированного слова AS можно задать имя вычисляемого поля.

SELECT фамилия, год\_поступления - год\_рождения AS возраст\_поступления

FROM Студент

WHERE N\_группы = 472301;

SELECT AVG(год\_поступления - год\_рождения) AS средний\_возраст

FROM Студент;

(результирующая строка содержит одну строку: средний возраст поступления в ВУЗ)

Выражения для вычисляемых полей могут включать в себя имена полей, константы, агрегатные функции и знаки арифметических операций (+, -, \*, /). При этом знак «+» используется еще и для конкатенации (соединения, слияния) текстовых значений.

Например:

(пусть в таблице «Студент» имеются еще поля «Фамилия», «Имя», «Отчество»)

SELECT фамилия + ' ' + имя + ' ' + отчество AS ФИО

FROM Студент;

(результирующий набор данных будет содержать столько же строк, что и таблица «Студент», и один столбец «ФИО»)

*Группировка строк.*

Команда выборки данных с предложением GROUP BY позволяет сгруппировать строки набора данных на основе значений одного или нескольких полей и для каждой группы подсчитать итоговые данные с помощью агрегатных функций. Например:

SELECT факультет, COUNT(\*)

FROM Группа

GROUP BY факультет;

(результирующий набор данных содержит число строк, равное числу факультетов в ВУЗе, причем в каждой строке кроме факультета указано также количество групп на этом факультете)

SELECT год\_рождения, COUNT(\*)

FROM Студент

WHERE N\_группы = 472301

GROUP BY год\_рождения;

(результирующий набор данных содержит число строк, равное числу различных годов рождения, причем в каждой строке кроме года рождения указано также количество студентов, родившихся в этом году. Подсчет ведется применительно к группе 472301)

SELECT предмет, семестр, MIN(оценка), AVG(оценка), MAX(оценка)

FROM Экзамен

GROUP BY предмет, семестр;

(результирующий набор данных содержит итоги экзаменов по каждому предмету)

Предложение SELECT как правило включает в себя все поля, содержащиеся в предложении GROUP BY, а также поля, используемые для подсчета итоговых данных.

Команда выборки данных с группировкой может включать в себя предложение HAVING, которое, как и предложение WHERE задает условие отбора данных. Однако, если предложение WHERE задает условие отбора для отдельных строк, то предложение HAVING задает условие отбора для групп строк, определяемых предложением GROUP BY. Поэтому в предложении HAVING обязательно должна быть указана хотя бы одна из агрегатных функций, фигурирующих в предложении SELECT, в то время как в предложении WHERE это недопустимо.

Таким образом, результирующий набор данных формируется в два этапа: сначала выполняется обычная группировка и подсчет итоговых данных, а потом к полученному набору данных применяется условие отбора, задаваемое предложением HAVING.

SELECT факультет, COUNT(\*)

FROM Группа

GROUP BY факультет

HAVING COUNT(\*) <= 15;

(результирующий набор данных содержит сведения о количестве групп на тех факультетах, на которых это количество не превышает 15)

SELECT семестр, предмет, MIN(оценка), AVG(оценка), MAX(оценка)

FROM Экзамен

GROUP BY семестр, предмет

HAVING AVG(оценка) < 6;

(результирующий набор данных содержит итоговые данные по тем предметам, по которым средняя оценка ниже 6)

*Соединение таблиц.*

Команда выборки данных позволяет извлечь данные сразу из нескольких таблиц. Для этого их имена нужно перечислить в предложении FROM, а в предложении WHERE необходимо задать условия соединения, указывая через знак равенства поля, по которым производится соединение таблиц. Чаще всего такими полями являются первичные и внешние ключи таблиц, между которыми установлены связи типа «1 : М».

SELECT Группа.\*, Студент.фамилия

FROM Группа, Студент

WHERE Группа.N\_группы = Студент.N\_группы;

(поле «N\_группы», по которому производится соединение таблиц, является первичным ключем в таблице «Группа» и внешним ключем в таблице «Студент»)

Чтобы упростить запись таких команд, имена таблиц в предложении FROM снабжаются псевдонимами, которые можно использовать лишь в контексте данной команды. Поэтому предыдущую команду можно записать следующим образом:

SELECT Г.\*, С.фамилия

FROM Группа Г, Студент С

WHERE Г.N\_группы = С.N\_группы;

Пример соединения трех таблиц:

SELECT Г.\*, С.фамилия, Э.\*

FROM Группа Г, Студент С, Экзамен Э

WHERE Г.N\_группы = С.N\_группы

AND С.N\_зачетки = Э.N\_зачетки

(результат получается в 2 этапа: сначала соединяются «Группа» и «Студент», а затем полученный набор данных соединяется с «Экзамен»)

Если условия соединения не указывать, то результирующий набор данных будет представлять собой декартово произведение таблиц, перечисленных в предложении FROM. Например, если в предыдущем примере число строк в таблицах будет 10, 100 и 500 и будет опущено предложение WHERE с условиями соединения таблиц, то результирующий набор данных будет содержать 10×100×500 = 500000 строк.

Как правило, базы данных не содержат изолированных таблиц, поэтому можно соединять любые таблицы, добавляя при необходимости промежуточные, связующие таблицы. Еще пример:

SELECT С.фамилия, С.N\_зачетки, Э.оценка

FROM Студент С, Экзамен Э

WHERE С.N\_зачетки = Э.N\_зачетки

AND С.N\_группы = 472301 AND Э.предмет = ‘Теория систем’

ORDER BY С.фамилия;

(для студентов группы 472301, упорядоченных по фамилии, выдаются оценки по предмету ‘Теория систем’)

Рассмотренный способ соединение таблиц известен как внутреннее соединение, и он характерен тем, что результирующий набор данных будет содержать только те строки, которые удовлетворяют условиям соединения, указанным в предложении WHERE.

Рассмотрим снова команду:

SELECT Г.\*, С.фамилия

FROM Группа Г, Студент С

WHERE Г.N\_группы = С.N\_группы;

Предположим, что таблица «Студенты» содержит 100 строк, в половине из которых поле «N\_группы» не заполнено, т.е. имеет значение NULL. Тогда рассмотренная выше команда вернет результирующий набор данных, содержащий не 100, а 50 строк, т.к. поле «N\_группы» фигурирует в условии соединения.

Если же мы хотим получить фамилии всех 100 студентов и увидеть, у кого из них отсутствует информация о группе, то нужно использовать другой тип соединения, называемый внешним соединением.

Внешнее соединение может быть левым или правым. Левое внешнее соединение включает в результирующий набор данных все строки таблицы, расположенной слева в предложении FROM, а правое внешнее соединение – все строки таблицы, расположенной справа. При этом условие соединения записывается в предложении FROM после зарезервированного слова ON. Например:

SELECT Г.\*, С.фамилия

FROM Студент С LEFT OUTER JOIN Группа Г

ON Г.N\_группы = С.N\_группы;

(левое внешнее соединение, позволяющее получить фамилии всех 100 студентов)

SELECT Г.\*, С.фамилия

FROM Группа Г RIGHT OUTER JOIN Студент С

ON Г.N\_группы = С.N\_группы;

(правое внешнее соединение, позволяющее получить фамилии всех 100 студентов)

Левое внешнее соединение трех таблиц:

SELECT Г.\*, С.фамилия, Э.оценка

FROM Студент С LEFT OUTER JOIN Группа Г

ON Г.N\_группы = С.N\_группы

LEFT OUTER JOIN Экзамен Э

ON С.N\_зачетки = Э.N\_зачетки

WHERE Г.факультет = ‘ИЭ’ AND Э.предмет = ‘Теория систем’;

(для всех студентов инженерно-экономического факультета, сдавших экзамен по предмету «Теория систем», выдаются оценки по этому предмету не зависимо от того, проставлено ли поле N\_группы в таблице «Студент» или нет)

Внешнее соединение используются не очень часто и обычно в тех случаях, когда поля, по которым производится соединение, содержат NULL-значения. Если же указанные поля не содержат NULL-значений, то внутреннее и внешнее соединения порождают один и тот же результирующий набор данных.

Имеется другой, альтернативный способ записи внутренних соединений, аналогичный способу записи внешних соединений. С использованием этого способа последнюю команду можно переписать так:

SELECT С.фамилия, Э.\*

FROM Студент С INNER JOIN Экзамен Э

ON С.N\_зачетки = Э.N\_зачетки

WHERE Э.оценка = 10;

(здесь условие внутреннего соединения записывается в предложении FROM после зарезервированного слова ON)

*Использование подзапросов.*

Язык SQL позволяет вкладывать запросы на выборку друг в друга. Обычно внутренний запрос, называемый подзапросом, генерирует набор данных, с помощью которого выполняется оценка истинности предиката внешнего запроса. Например:

SELECT \*

FROM Студент

WHERE N\_группы IN (SELECT N\_группы

FROM Группа

WHERE факультет = 'ИЭ');

(перечисляются все студенты инженерно-экономического факультета)

SELECT С.\*

FROM Группа Г INNER JOIN Студент С

ON Г.N\_группы = С.N\_группы

WHERE Г.факультет = ‘ИЭ’;

(аналогичная команда с использованием соединения таблиц)

Пример с подзапросом и соединением таблиц:

SELECT Э.предмет, С.фамилия, Э.оценка

FROM Студент С INNER JOIN Экзамен Э

ON С.N\_зачетки = Э.N\_зачетки

WHERE Э.оценка > (SELECT AVG(оценка)

FROM Экзамен)

ORDER BY Э.предмет, С.фамилия;

(выбираются только те оценки, которые больше средней оценки по ВУЗу)

*Оператор EXISTS в командах с подзапросом.*

Оператор EXISTS использует подзапрос в качестве аргумента и принимает значение истина только в том случае, если подзапрос возвращает набор данных, содержащий хотя бы одну строку. Например:

SELECT \*

FROM Студент С

WHERE EXISTS (SELECT \*

FROM Экзамен Э

WHERE Э.N\_зачетки = С.N\_зачетки

AND оценка <= 3);

(выдается список всех студентов, имеющих неудовлетворительные оценки)

*Оператор ANY (SOME) в командах с подзапросом.*

Оператор ANY (SOME) перебирает все значения, выдаваемые подзапросом, и оценивает результат как истину, если хотя бы одно из этих значений обеспечивает истинность предиката внешнего запроса.

SELECT \*

FROM Группа Г

WHERE 1995 = ANY(SELECT год\_рождения

FROM Студент С

WHERE С.N\_группы = Г.N\_группы);

(выдается список тех групп, в которых хотя бы один студент с 1995 года рождения)

*Оператор ALL в командах с подзапросом.*

Оператор ALL перебирает все значения, выдаваемые подзапросом, и оценивает результат как истину, если каждое из этих значений обеспечивает истинность предиката внешнего запроса.

SELECT \*

FROM Группа Г

WHERE 1995 <= ALL(SELECT год\_рождения

FROM Студент С

WHERE С.N\_группы = Г.N\_группы);

(выдается список тех групп, в которых каждый из студентов с 1995 или последующего года рождения)

**5. Объединение данных.**

< запрос на выборку данных>

UNION

< запрос на выборку данных>

[UNION

< запрос на выборку данных> . . . ]

[ORDER BY <имя поля> [, <имя поля> ...]];

В объединении все запросы выполняются независимо, а затем их выходные данные объединяются в один результирующий набор данных с удалением повторяющихся строк.

Для того, чтобы два или более запроса можно было объединить, они должны содержать одинаковое число совместимых по объединению столбцов.

### 3.1.3. Основные команды определения данных

**1. Создание базы данных.**

CREATE DATABASE <имя базы данных>;

Например:

CREATE DATABASE Сессия;

(создается новая пустая база данных «Сессия»)

**2. Создание таблицы.**

CREATE TABLE <имя таблицы> (

<имя поля> <тип данных>[(<размер>)] [NOT NULL] [PRIMARY KEY] [UNIQUE]

[CHECK (<ограничение на столбец>)] [DEFAULT = <значение по

умолчанию>]

[<имя поля> <тип данных>[(<размер>)] [NOT NULL] [UNIQUE]

[CHECK (<ограничение на столбец>)] [DEFAULT = <значение по

умолчанию>] … ]

[UNIQUE (<имя поля> [, <имя поля> …])]

[CHECK (<ограничение на таблицу>)]

[PRIMARY KEY (<имя поля> [, <имя поля> …])]

);

В стандарте SQL92 определены следующие типы данных.

Текстовый: CHAR(n), где n – максимальная длина строки в символах.

Целое: INT.

Короткое целое: SMALLINT.

Вещественное с фиксированной точкой: DECIMAL(n,m), где n – число разрядов в целой части числа, m – число разрядов в дробной части.

Вещественное с плавающей точкой: FLOAT, DOUBLE.

Типы для даты и времени: DATE, TIME.

Например:

CREATE TABLE Группа (

N\_группы INT NOT NULL PRIMARY KEY

факультет CHAR(10) DEFAULT = ‘ИЭ’

);

CREATE TABLE Студент (

N\_зачетки INT NOT NULL PRIMARY KEY

фамилия CHAR(20) NOT NULL

год\_рождения SMALLINT CHECK (год\_рождения > 1970)

год\_поступления SMALLINT CHECK (год\_поступления > 2010)

N\_группы INT

);

CREATE TABLE Экзамен (

семестр SMALLINT NOT NULL

предмет CHAR(50) NOT NULL

N\_зачетки INT NOT NULL

оценка SMALLINT CHECK (оценка >= 0 AND оценка <= 10)

PRIMARY KEY (семестр, предмет, N\_зачетки)

);

**3. Добавление в таблицу нового поля.**

ALTER TABLE <имя таблицы> ADD <имя поля> <тип данных> [(<размер>)]

[NOT NULL] [UNIQUE]

[CHECK (<ограничение на столбец>)]

[DEFAULT = <значение по умолчанию>];

Например:

ALTER TABLE Студент ADD имя CHAR(20);

ALTER TABLE Студент ADD отчество CHAR(20);

**4. Удаление поля из таблицы.**

ALTER TABLE <имя таблицы> DROP <имя поля>;

Например:

ALTER TABLE Студент DROP имя;

ALTER TABLE Студент DROP отчество;

**5. Удаление таблицы.**

DROP TABLE <имя таблицы>;

Например:

DROP TABLE Экзамен;

**6. Создание представления.**

Представление – это объект базы данных, который не содержит собственных данных. Это своего рода таблица, содержимое которой берется из других таблиц в результате выполнения запроса на выборку данных. Поскольку данные в таблицах меняются, это приводит к соответствующему изменению выходных данных представления.

CREATE VIEW <имя представления> AS < запрос на выборку данных>;

Пример:

CREATE VIEW Экзамен\_472301 AS

SELECT С.фамилия, Э.\*

FROM Студент С INNER JOIN Экзамен Э

ON С.N\_зачетки = Э.N\_зачетки

WHERE С.N\_группы = 472301;

Представление можно использовать так же как и таблицу: применять команды выборки, вставки, обновления и удаления данных, соединять с другими таблицами и представлениями и др. Например:

SELECT фамилия, оценка

FROM Экзамен\_472301

WHERE предмет = 'Теория систем'

ORDER BY фамилия;

Любое представление можно использовать в командах выборки данных, однако не каждое из них может быть использовано в командах добавления, удаления и обновления записей.

**7. Удаление представления.**

DROP VIEW <имя представления>;

Например:

DROP VIEW Экзамен\_472301;

**8. Создание индекса.**

Индекс ускоряет выполнение команд выборки данных, в условиях которых фигурируют индексированные поля, а также операций сортировки и фильтрации строк по таким полям.

Индекс представляет собой значения индексированного столбца, упорядоченные по возрастанию или убыванию, которые дополнены указателями на соответствующие им строки таблицы. При поиске данных СУБД сначала выполняет поиск в индексе, а затем по соответствующим ссылкам выбирает нужные данные уже непосредственно из таблицы. Поиск в упорядоченных данных (индексах) выполняется очень быстро, например, с помощью метода дихотомии. Индекс можно создавать как для одного, так и сразу для нескольких столбцов.

CREATE [UNIQUE] INDEX <имя индекса> ON <имя таблицы> (<имя поля> [, <имя поля> …]);

Например:

CREATE UNIQUE INDEX Гр\_Факультет ON Группа (факультет);

CREATE INDEX Ст\_ГодПостФамилия ON Студент (год\_поступления, фамилия);

**9. Удаление индекса.**

DROP INDEX <имя индекса>;

Например:

DROP INDEX Гр\_Факультет;

DROP INDEX Ст\_ГодПостФамилия;

Необходимо отметить, что выше были рассмотрены самые распространенные команды языка SQL, однако имеется еще ряд команд, связанных, например, с предоставлением и отменой привилегий доступа к данным, стартом, финишем и отменой транзакций и др.

## 3.2. Индивидуальная практическая работа 2 (ИПР2)

### 3.2.1. Методические указания по выполнению ИПР2

**Тема: Создание базы данных и ее объектов с помощью команд языка Transact-SQL**

При помощи пользовательского меню Windows запустите утилиту **SQL Server Management Studio**, после чего на панели **Object Explorer** в древовидной структуре раскройте папку **Databases**.

Далее на панели инструментов нажмите кнопку **New Query**, что вызовет появление на экране окна создания запросов к базе данных. Это окно, называемое окном **Query**, используется для последовательного формирования в нем сценария (скрипта), который далее можно сохранить в файле на диске. Сценарий представляет из себя последовательность команд языка Transact-SQL, с помощью которых можно будет впоследствии автоматически сгенерировать базу данных со всеми ее объектами, а также наполнить ее таблицы данными и выполнить ряд других действий.

Создание сценария заключается в том, что каждый раз нужно добавлять в окно **Query** очередную команду языка Transact-SQL (или группу команд, образующих пакет), проверять ее синтаксис и затем выполнять, нажимая на панели инструментов кнопку **Execute** (или клавишу F5). При этом необходимо, чтобы данная команда (или пакет) была выделена подсветкой, иначе будут выполнены все без исключения команды, содержащиеся в окне **Query**. Если очередная команда (или пакет) будет успешно выполнена, то в окно **Query** можно дописать новую команду (или пакет), выделить подсветкой и далее повторить процесс, описанный выше. В конечном итоге, путем поочередного выполнения команд, будет создана база данных со всеми ее объектами и параллельно в окне **Query** будет сформирован сценарий, который можно сохранить в файле на диске. Теперь с помощью этого сценария можно на любом компьютере, где установлен MS SQL Server, выполнить процедуру создания базы данных.

**Порядок выполнения работы**

1. Согласно выбранному варианту задания, создайте SQL-сценарий, который позволяет сгенерировать базу данных в формате **MS SQL Server**, а также наполнить созданную базу данных конкретными данными.

2. Дополните сценарий текстами перечисленных ниже хранимых процедур:

* Создайте хранимую процедуру, которая для указанной таблицы подсчитывает число строк, содержащихся в этой таблице. Эта процедура должна иметь один входной параметр (с помощью которого задается имя таблицы) и один выходной параметр.
* Создайте хранимую процедуру, которая изменяет все даты, имеющиеся в таблицах базы данных, добавляя к ним или отнимая от них некоторое число дней. Эта процедура должна иметь два входных параметра. Первый параметр определяет, нужно ли увеличивать или уменьшать все даты. Второй параметр указывает количество дней, которые должны добавляться или отниматься. Если же в таблицах отсутствуют даты, то аналогичную операцию необходимо проделать применительно ко всем имеющимся ценам и стоимостям. При этом второй параметр будет задавать процент, на который должны измениться все цены и стоимости.
* Создайте хранимую процедуру, которая для конкретной таблицы подсчитывает минимальное, среднее, максимальное и суммарное значения в столбце числового типа. Эта процедура должна иметь один входной параметр (с помощью которого указывается имя или номер столбца) и четыре выходных параметра.
* Создайте хранимую процедуру, которая из конкретной таблицы отбирает строки по условию, налагаемому на значения конкретных двух столбцов. Эта процедура должна иметь два входных параметра, задающих значения для отбора данных по каждому из этих столбцов. При этом значение NULL должно трактоваться как отсутствие какого-либо критерия отбора строк по данному столбцу (например, если оба параметра будут иметь значение NULL, то должны быть выбраны все строки таблицы).
* Создайте хранимую процедуру, которая на основе текстовой строки, содержащей фамилию, имя и отчество, формирует текстовую строку, содержащую фамилию и инициалы, и при этом все строчные буквы должны быть заменены прописными. Эта процедура должна иметь один входной параметр (с помощью которого задается исходная текстовая строка) и один выходной параметр. При этом необходимо расширить возможности процедуры таким образом, чтобы была допустима исходная строка (задаваемая входным параметром), содержащая не один, а несколько пробелов между фамилией и именем или между именем и отчеством, а также допускающая наличие лидирующих пробелов перед фамилией.

*Примечание*. Некоторые их указанный выше хранимых процедур можно, по желанию, оформить в виде соответствующих пользовательских функций.

3. Создайте в базе данных таблицу **Протокол** со структурой, приведенной ниже.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер | ДатаВремя | Пользователь | Действие | ЧислоСтрок |
| **. . .** | **. . .** | **. . .** | **. . .** | **. . .** |

Здесь столбец **Номер** является автоинкрементным первичным ключом. В столбце **Действие** будет указываться одна из трех возможных операций с данными: «Вставка», «Обновление», «Удаление». Столбец **ЧислоСтрок** будет содержать данные о числе вставленных, либо обновленных, либо удаленных строк.

4. Создайте триггер, связанный с конкретной таблицей базы данных, который предназначен для автоматической фиксации в таблице **Протокол** всех действий, вызванные вставкой, обновлением или удалением данных в таблице, снабженной триггером. Каждая SQL-команда, изменяющая содержимое этой таблицы, должна быть отражена отдельной строкой в таблице **Протокол**.

5. Разработайте SQL-код для проверки работы созданных хранимых процедур и триггера.

### 3.2.2. Варианты заданий ИПР2

SQL-сценарий в формате MS SQL Server для генерации базы данных и ее таблиц, а также хранимых процедур и триггера, необходимо разработать применительно к одному из следующих направлений деятельности:

1. Основные средства предприятия.

2. Банковские операции на предприятии.

3. Авансовые отчеты и бухгалтерские справки на предприятии.

4. Работа с доверенностями на предприятии.

5. Снабжение предприятия.

6. Отгрузка продукции на предприятии.

7. Отпуск в производство и выпуск продукции на предприятии.

8. Кассовые операции на предприятии.

9. Перемещение и списание материальных ценностей на предприятии.

10. Управление договорной деятельностью предприятия.

11. Управление работой нефтебазы.

12. Банки Беларуси.

13. Управление телефонной станцией.

14. Управление аптекой.

15. Управление поликлиникой.

16. Работа с недвижимостью.

17. Работа банков с расчетными счетами.

18. Работа банков с электронными карточками.

19. Управление работой АЗС.

20. Управление железнодорожными перевозками.

21. Управление автобусным движением.

22. Управление библиотечным фондом.

23. Животный и растительный мир Беларуси.

24. Климатическая карта Беларуси.

25. Отдел кадров предприятия.

26. Работа деканата в вузе.

27. Сеть магазинов определенного профиля.

28. Управление таксопарком.

29. Продажа и бронирование авиабилетов.

30. Учет товаров на складе.

**Указания по выбору варианта**

ИПР2 содержит 30 вариантов заданий. Выбор варианта для выполнения работы производится так же, как и при выполнении ИПР1.

**Порядок оформление итогового отчета**

Итоговый отчет формируется с помощью текстового редактора Word, и, помимо титульного листа и листа содержания, должен включать в себя:

1) описание конкретного варианта задания, подлежащего решению;

2) листинг SQL-сценария для генерации базы данных, таблиц и др., а также хранимых процедур и триггера;

3) скриншоты, отображающие: окно конструктора одной из таблиц; вид этой же открытой таблицы, наполненной данными;

4) тексты SQL-кода для проверки работы созданных хранимых процедур и триггера, а также скриншоты, демонстрирующие работу их работу;

5) выводы по работе;

6) список используемой литературы.

## 3.3. Промежуточный контрольный тест

1. Если в команде удаления записей из таблицы не указывать условие отбора данных, то какие записи будут удалены?

2. Как удалить строку родительской таблицы, если для нее существуют зависимые строки в дочерней таблице?

3. Необходимо упорядочить по убыванию приоритета логические связки И, ИЛИ, НЕ, используемые при построении условий отбора данных.

4. Что задают операторы IN, BETWEEN, LIKE, используемые при построении условий отбора данных?

5. Приведенное ниже предложение языка SQL состоит из восьми лексем, разделяемых пробелами:

WHERE фамилия LIKE ‘И%’ OR N\_группы = NULL

Укажите номер (считая слева направо) той лексемы, которая содержит ошибку.

6. Какие ключевые слова всегда присутствуют в любой команде выборки данных?

7. Вместе с каким ключевым словом используются зарезервированные слова ASC и DESC?

8. Вместе с каким ключевым словом используется ключевое слово HAVING?

9. Можно ли использовать агрегатные функции в предложении WHERE в команде выборки данных с группировкой и отсутствием подзапросов?

10. Можно ли в командах выборки данных вместо таблиц использовать представления?

11. В каком виде данные индексированных полей хранятся в индексе?

## 3.4. Правильные ответы на промежуточный контрольный тест

1. Если в команде удаления записей из таблицы не указывать условие отбора данных, то какие записи будут удалены?

Ответ: никакая; первая; первые 10; последняя; **все**.

(правильный ответ выделен полужирным шрифтом с подчеркиванием)

2. Как удалить строку родительской таблицы, если для нее существуют зависимые строки в дочерней таблице и каскадные удаления данных запрещены?

Ответ: сначала удалить зависимые строки в дочерней таблице, а затем удалить строку родительской таблицы.

3. Необходимо упорядочить по убыванию приоритета логические связки И, ИЛИ, НЕ, используемые при построении условий отбора данных.

Ответ: НЕ, И, ИЛИ.

4. Что задают операторы IN, BETWEEN, LIKE, используемые при построении условий отбора данных?

Ответ: IN – множество значений; BETWEEN – диапазон значений; LIKE – текстовые строки по образцу.

5. Приведенное ниже предложение языка SQL состоит из восьми лексем, разделяемых пробелами:

WHERE фамилия LIKE ‘И%’ OR N\_группы = NULL

Укажите номер (считая слева направо) той лексемы, которая содержит ошибку.

Ответ: 7.

6. Какие ключевые слова всегда присутствуют в любой команде выборки данных?

Ответ: **SELECT**; DISTINCT; **FROM**; WHERE; GROUP BY; HAVING; ORDER BY.

(правильный ответ выделен полужирным шрифтом с подчеркиванием)

7. Вместе с каким ключевым словом используются зарезервированные слова ASC и DESC?

Ответ: ORDER BY.

8. Вместе с каким ключевым словом используется ключевое слово HAVING?

Ответ: GROUP BY.

9. Можно ли использовать агрегатные функции в предложении WHERE в команде выборки данных с группировкой и отсутствием подзапросов?

Ответ: нет.

10. Можно ли в командах выборки данных вместо таблиц использовать представления?

Ответ: да.

11. В каком виде данные индексированных полей хранятся в индексе?

Ответ: в отсортированном виде.

# **4. Модуль 4. Распределенные базы данных и хранилища** данных

## 4.1. Теоретический материал

### 4.1.1. Распределенные базы данных

Система распределенных баз данных состоит из набора узлов, связанных коммуникационной сетью, в которой:

а) каждый узел – это полноценная СУБД;

б) узлы взаимодействуют между собой таким образом, что пользователь любого из них может получить доступ к любым данным в сети так, как будто они находятся на его собственном узле.

Отсюда следует, что распределенная база данных в действительности представляет собой виртуальную базу данных, компоненты которой физически хранятся в нескольких различных реальных базах данных на нескольких различных узлах.

Фундаментальный принцип создания распределенных баз данных заключается в том, что для пользователя распределенная система должна выглядеть так же, как нераспределенная система.

Следствием фундаментального принципа являются следующие правила, называемые еще двенадцатью правилами Дейта:

1. Локальная независимость. Она означает, что все операции на узле контролируются этим узлом, а также, что все узлы в распределенной системе должны рассматриваться как равные.

2. Независимость от центрального узла. Это означает, что не должно быть никаких обращений к центральному, или главному узлу для получения некоторой централизованной услуги.

3. Непрерывное функционирование. Это означает, что достигается более высокая степень надежности и доступности, т.к. функционирование системы не зависит от сбоя на каком-либо узле.

4. Независимость от расположения. Пользователи не должны знать, где именно данные хранятся физически и поступать так, как если бы все данные хранились на их собственном локальном узле.

5. Независимость от фрагментации. Для повышения производительности системы данные могут быть разбиты на части, или фрагменты, которые могут храниться на различных узлах в тех местах, где они чаще всего используются. Пользователям нет необходимости знать имена фрагментов для получения доступа к ним.

6. Независимость от репликации. Для пользователей должна быть создана такая среда, чтобы они могли считать, что, отсутствует дублирование данных на других узлах в виде отдельных копий или реплик.

7. Обработка распределенных запросов. Означает возможность с помощью одного запроса выбрать данные сразу из нескольких фрагментов данных, хранящихся на различных узлах.

8. Управление распределенными транзакциями. Такие транзакции могут обновлять данные сразу на нескольких узлах. Чтобы обеспечить неразрывность транзакции в распределенной среде, система должна гарантировать, что все множество относящихся к данной транзакции операций или зафиксировало свои результаты, или выполнило откат. Такого результата можно достичь с помощью протокола так называемой двухфазной фиксации транзакций.

9. Аппаратная независимость. Предполагает возможность использования компьютеров различных типов.

10. Независимость от операционной системы. Предполагает возможность использования любых операционных платформ.

11. Независимость от сети. Предполагает возможность работы на любой сетевой платформе.

12. Независимость от вида СУБД. Чаще всего обеспечивается за счет использования шлюзов. Однако реализация полной независимости от СУБД связана со значительными техническими сложностями. Если же на каждом узле работает некоторая копия одной и той же СУБД, то такая система называется однородной и чаще всего и применяется на практике.

Необходимо отметить, что не все указанные выше цели независимы одна от другой. Кроме того, они не исчерпывающие и не все одинаково важны.

### 4.1.2. Системы «клиент/сервер»

Системы «клиент/сервер» могут рассматриваться как частный случай распределенных систем, в которых одни узлы – *клиенты*, а другие – *серверы*. Все данные размещены на узлах, которые являются серверами. Все приложения выполняются на узлах-клиентах. При этом полная локальная независимость (первое правило Дейта) не предоставляется.

Для корректного взаимодействия компонентов клиент/серверной архитектуры между собой требуется их соответствие некоторым дополнительным правилам:

1. Открытый доступ к сервисам. Это означает, что все клиенты в системе должны иметь неограниченный доступ ко всем сервисам, предоставляемым внутри сети.

2. Возможность распределения процессов. Главное свойство клиент/серверных систем состоит в том, что обработка информации распределяется между клиентами и серверами. В зависимости от степени разделения процессов обработки данных между клиентом и сервером каждый из них может считаться либо сильным, либо слабым. Слабый (тонкий) клиент выполняет минимум обработки на стороне клиента, в то время как сильный (толстый) клиент берет на себя значительную часть функций обработки данных.

3. Стандартизация. Все правила взаимодействия в клиент/серверной архитектуре должны быть основаны на стандартах. Например, стандартными должны быть пользовательский интерфейс, сетевые протоколы, взаимный обмен между процессами и т.д. Стандарты гарантируют, что для достижения желаемого результата все компоненты будут действовать упорядоченно.

### 4.1.3. Поддержка принятия решений

Системы поддержки принятия решений – это системы, которые служат для анализа деловой информации. Их назначение – помочь руководителям «выявить тенденции, определить проблемы и далее предложить разумное решение». Подобные системы создаются на основе таких теорий, как исследование операций, теория поведения, теория управления, а также с помощью методов статистической обработки.

Основной идеей является накопление в операционных базах данных производственных оперативных данных и приведение их к виду, в котором они могли бы использоваться для анализа хода деловых процессов и разумной корректировки деловой активности. Однако в настоящее время требования к информации стали настолько сложными, что даже с помощью систем поддержки принятия решений стало трудно извлекать необходимую информацию из структур данных, которые хранятся в операционных базах данных. Поэтому были разработаны новые технологии хранения информации, ориентированные на поддержку принятия решений.

К таким технологиям относятся:

***хранилища данных*** (data warehouse);

***магазины*** или ***витрины* *данных*** (data mart);

***банки оперативных данных*** (operational data store);

***оперативная аналитическая обработка*** (online analytical processing – OLAP);

***многомерные базы данных*** (multidimentional database);

***разработка***или ***извлечение данных*** (data mining).

**Хранилища данных**

Если данные для систем поддержки принятия решений извлекаются из различных оперативных систем (часто в корне отличных по своей организации), образуя тем самым обширный фонд данных, и помещаются в собственное хранилище, реализованное на отдельной платформе, то такое отдельное хранилище называют базой данных поддержки принятия решений или хранилищем данных.

Хранилища данных обладают особенными характеристиками, к которым относятся следующие:

1. Данные в них в основном предназначены только для чтения. Модификация содержимого базы данных обычно ограничивается периодическими операциями загрузки (вставки новых данных). Операция удаления данных выполняется крайне редко, а операция обновления (команда UPDATE) не выполняется никогда. Поэтому часто не имеет смысла объявлять ограничения целостности данных, т.к. если база данных предназначена только для чтения, то такие ограничения на самом деле никогда не будут нарушены.

2. Столбцы чаще всего используются в сочетаниях. Эти сочетания столбцов часто рассматриваются как единое целое, т.е. к столбцам, входящим в состав таких сочетаний, никогда не приходится обращаться по отдельности. Такие сочетания столбцов называются составными столбцами. В этом случае зависимости, включающие элементы составного столбца и не включающие никакие другие столбцы, не имеют смысла и могут просто игнорироваться.

3. Поддержкой целостности данных в общем случае не занимаются. Подразумевается, что данные будут правильными, т.к. после загрузки они не корректируются.

4. Ключи части содержат временной компонент. Операционные базы данных обычно содержат только текущие данные. Хранилища данных, наоборот, обычно содержат архивы исторически накопленных данных и, поэтому, большая часть данных или даже все данные включают временной штамп или временную отметку.

5. Хранилища данных обычно содержат большой объем информации, особенно в том случае, когда данные деловых транзакций извлекаются из нескольких операционных баз данных и накапливаются в течение достаточно продолжительного времени.

6. В хранилище данных, как правило, широко применяется индексация. Наряду с традиционными индексами, известными под названием B-индексов, получили распространение также битовые индексы, хэшированные индексы, мультитабличные индексы, логические индексы, функциональные индексы, а также различные виды смешанных или гибридных индексов. Кроме того, предлагается множество специализированных видов индексов, таких, например, как R-деревья, которые предназначены для обработки пространственных данных.

7. Хранилища данных часто содержат различного рода контролируемую избыточность. Избыточность называется контролируемой, если она управляется СУБД и скрыта от пользователей. Контролируемая избыточность позволяет сократить количество операций ввода-вывода и минимизировать конкурентный доступ пользователей к данным в базе. Существует два общих вида такой избыточности. Первый вид включает обслуживание в базе данных точных копий или реплик. Второй вид включает обслуживание производных данных в дополнение к базовым данным. Производными данными чаще всего являются итоговые таблицы и вычисляемые столбцы.

Итоговые таблицы и вычисляемые столбцы чаще всего реализуются с помощью триггеров, управляемых системой. В этом случае автоматически поддерживается согласованность между базовыми и производными данными.

**Подготовка данных**

Многие из вопросов, связанных с хранилищами данных, в первую очередь касаются задач получения и подготовки данных. Эти данные следует:

***- извлечь*** из разных источников;

***- очистить***;

***- преобразовать*** и ***консолидировать***;

***- загрузить*** в хранилище.

Впоследствии загруженные данные должны периодически пополняться.

***Извлечение данных*** – это процесс выборки данных из операционных баз данных и других источников.

***Очистка данных*.** Она необходима, т.к. удовлетворительный контроль информации обеспечивают лишь немногие источники данных. Вследствие этого, прежде чем данные будут введены в хранилище данных, обычно требуется выполнить их очистку. Обычно очистка предусматривает заполнение отсутствующих значений, корректировку опечаток и других ошибок, допущенных при вводе данных, определение стандартных сокращений и форматов, замену синонимов стандартными идентификаторами и т.д. Данные, которые определяются как ошибочные и не могут быть исправлены, отбрасываются.

***Преобразование и консолидация данных***. После очистки данных полученная информация, скорее всего, еще не будет отвечать требованиям системы поддержки принятия решений и, следовательно, будет нуждаться в соответствующем преобразовании, которое чаще всего предусматривает выполнение операций построчного разбиения или объединения исходных записей. Процедура преобразования приобретает особую важность, когда необходимо выполнить слияние данных, поступивших из нескольких источников. Этот процесс называется консолидацией. В таком случае любая неявная связь между данными из отдельных источников должна быть преобразована в явную путем введения явных значений данных.

***Загрузка данных***. Операция загрузки данных разбивается на следующие этапы:

а) пересылка преобразованных и консолидированных данных в хранилище данных;

б) проверка согласованности данных (т.е. проверка их целостности);

в) построение всех необходимых индексов.

В итоге можно сказать, что хранилище данных определяется как «предметно-ориентированное, интегрированное, постоянное, изменяющееся во времени хранилище информации для поддержки управленческих решений». Здесь термин «постоянное» означает, что, будучи введенными, данные впоследствии не изменяются, хотя и могут быть удалены.

Как правило, такие хранилища очень велики, имеют много индексов и предназначены преимущественно для чтения. В них обычно присутствует контролируемая избыточность, чаще всего в форме репликации и предварительно вычисленных итоговых таблиц. Ключи обычно содержат временной компонент, а запросы, как правило, очень сложны. Исходя из этих соображений, при проектировании первостепенное значение уделяется обеспечению производительности систем.

**Магазины (витрины) данных**

Хранилища данных в общем случае представляют собой единый источник информации для любой обработки, связанной с поддержкой принятия решений. Однако часто пользователи выполняют различные операции анализа данных на относительно небольшом подмножестве всего множества информации в хранилище данных. Это привело к появлению концепции магазинов (витрин) данных. Магазин данных можно определить как «специализированное, предметно-ориентированное, интегрированное, непостоянное, изменяющееся во времени хранилище информации для поддержки конкретного подмножества управленческих решений». Различие между магазинами данных и хранилищами данных заключается в том, что магазины данных – специализированные и непостоянные. Последнее означает, что пользователи могут обновлять данные и, возможно, даже создавать в каких-то целях новые объекты данных, например, новые таблицы.

**Банки оперативных данных**

Банки оперативных данных представляют собой «предметно-ориентированные, интегрированные, обновляемые, текущие (или почти текущие) коллекции данных». Можно сказать, что это специализированные базы данных, которые могут использоваться для разных целей: как область накопления для реорганизации извлеченных оперативных данных, как средство формирования оперативных отчетов, как средство поддержки оперативных решений, а также местом консолидации данных, если оперативные данные поступают из нескольких источников.

**Многомерные базы данных**

Многомерными называются такие базы данных, в которых данные концептуально хранятся в ячейках многомерного массива. Поддерживающая СУБД называется многомерной.

В качестве простого примера можно привести трехмерный массив, представляющий, соответственно, товары, заказчиков и периоды времени. В каждой из ячеек хранится общий объем указанного товара, проданного конкретному заказчику за указанный период времени. С учетом связей между данными товар, заказчик и период времени можно считать независимыми переменными (по аналогии с полями возможного ключа реляционной таблицы), а количество – единственной зависимой переменной. Независимые переменные задают размерность массива данных, а также образуют схему адресации. Значения зависимых переменных, представляющих фактические данные, сохраняются в ячейках массива.

**Разработка (извлечение) данных**

Разработку данных (data mining) можно охарактеризовать как исследовательский анализ данных. Цель такой разработки – поиск интересных зависимостей среди данных, которые впоследствии могут использоваться для выработки стратегии деловой активности или для выявления необычного поведения. Например, внезапное возрастание интенсивности использования какой-то кредитной карточки может означать, что она украдена. В инструментах разработки данных используются статистические методы, применяемые к большим объемам хранимых данных, что и позволяет найти интересующие пользователя закономерности.

## 4.2. Промежуточный контрольный тест

1. В чем заключается фундаментальный принцип создания распределенных баз данных?

2. Сколько имеется правил, сформулированных Дейтом, для распределенных баз данных?

3. Какое из правил Дейта не выполняется в системах «клиент/сервер»?

4. Как часто выполняются в хранилищах данных следующие операции: чтение данных, вставка новых данных, обновление данных, удаление данных?

5. Используются ли в хранилищах данных средства поддержки целостности данных?

6. В чем основное отличие ключей в таблицах хранилищ данных от ключей в обычных операционных базах данных?

7. Как широко используется индексация в таблицах хранилищ данных?

8. На основе чего реализуется контролируемая избыточность в хранилищах данных?

9. С помощью чего автоматически поддерживается согласованность между базовыми и производными данными в хранилищах данных?

10. В каком порядке выполняются перечисленные ниже этапы процесса подготовки данных в хранилищах данных?

11. Как называется технология, в которой используются статистические методы, применяемые к большим объемам хранимых данных, для поиска интересующих пользователя закономерностей?

## 4.3. Правильные ответы на промежуточный контрольный тест

1. В чем заключается фундаментальный принцип создания распределенных баз данных?

Ответ: для пользователя распределенная база данных должна выглядеть так же, как нераспределенная.

2. Сколько имеется правил, сформулированных Дейтом, для распределенных баз данных?

Ответ: 12.

3. Какое из правил Дейта не выполняется в системах «клиент/сервер»?

Ответ: правило 1 (Локальная независимость).

4. Как часто выполняются в хранилищах данных следующие операции: чтение данных, вставка новых данных, обновление данных, удаление данных?

Ответ:

чтение данных – **часто**; периодически; крайне редко; никогда;

вставка новых данных – часто; **периодически**; крайне редко; никогда;

обновление данных – часто; периодически; крайне редко; **никогда**;

удаление данных – часто; периодически; **крайне редко**; никогда.

5. Используются ли в хранилищах данных средства поддержки целостности данных?

Ответ: нет.

6. В чем основное отличие ключей в таблицах хранилищ данных от ключей в обычных операционных базах данных?

Ответ: ключи таблиц в хранилищах данных обычно содержат временной штамп (отметку).

7. Как широко используется индексация в таблицах хранилищ данных?

Ответ: **широко, с использованием различных видов индексов**; также, как и в операционных БД; слабо; индексация не используется.

(правильный ответ выделен полужирным шрифтом с подчеркиванием)

8. На основе чего реализуется контролируемая избыточность в хранилищах данных?

Ответ: механизм транзакций; **механизм репликации**; **итоговые таблицы**; индексы; **вычисляемые столбцы**; резервное копирование.

(правильные ответы выделены полужирным шрифтом с подчеркиванием)

9. С помощью чего автоматически поддерживается согласованность между базовыми и производными данными в хранилищах данных?

Ответ: с помощью триггеров.

10. Этапами процесса подготовки данных в хранилищах данных являются: загрузка, преобразование, извлечение, консолидация, очистка. В каком порядке выполняются эти этапы?

Ответ: 1- извлечение; 2- очистка; 3- преобразование; 4- консолидация; 5- загрузка.

11. Как называется технология, в которой используются статистические методы, применяемые к большим объемам хранимых данных, для поиска интересующих пользователя закономерностей?

Ответ: Разработка или извлечение данных.

# 5. Итоговый тест по дисциплине

1. Какое главное достоинство реляционной модели данных?

2. Может ли заголовок сущности содержать несколько атрибутов, связанных с одним и тем же доменом?

3. Может ли сущность не иметь ни одного возможного ключа?

4. Сколько существует базовых реляционных операций, операнды которых должны быть совместимы по типу?

5. Можно ли при разработке моделей данных обойтись без использования связи типа «один к одному»?

6. Приведение к какой нормальной форме устраняет в сущности многозначные зависимости?

7. Когда не требуется выполнять операцию приведения сущности ко второй нормальной форме?

8. Если внешний ключ таблицы является обязательным полем (не допускается значение NULL), то какие правила ссылочной целостности могут быть использованы?

9. Если в команде вставки новой записи в таблицу перечисляются имена полей, то для каких полей имена можно не указывать?

10. Сколько строк и столбцов содержит результирующий набор данных, получаемый с помощью команды:

SELECT COUNT(\*), COUNT(год\_рождения), COUNT(N\_группы)

FROM Студент;

11. Когда внутреннее и внешнее соединение двух таблиц дает один и тот же результирующий набор данных?

12. Какие операторы используются только в командах выборки данных с подзапросом?

# 6. Правильные ответы на итоговый тест

1. Какое главное достоинство реляционной модели данных?

Ответ: очень быстрый поиск данных.

2. Может ли заголовок сущности содержать несколько атрибутов, связанных с одним и тем же доменом?

Ответ: да.

3. Может ли сущность не иметь ни одного возможного ключа?

Ответ: нет.

4. Сколько существует базовых реляционных операций, операнды которых должны быть совместимы по типу?

Ответ: 3.

5. Можно ли при разработке моделей данных обойтись без использования связи типа «один к одному»?

Ответ: да.

6. Приведение к какой нормальной форме устраняет в сущности многозначные зависимости?

Ответ: к четвертой нормальной форме.

7. Когда не требуется выполнять операцию приведения сущности ко второй нормальной форме?

Ответ: когда сущность снабжена простым первичным ключом.

8. Если внешний ключ таблицы является частью первичного ключа дочерней таблицы, то какие правила ссылочной целостности могут быть использованы?

Ответ: **RESTRICT**, **CASCADE**, SET NULL, SET DEFAULT, IGNORE.

(правильные ответы выделены полужирным шрифтом с подчеркиванием)

9. Если в команде вставки новой записи в таблицу перечисляются имена полей, то для каких полей имена можно не указывать?

Ответ: 1) для полей, допускающих значение NULL; 2) для полей, снабженных значением по умолчанию; 3) для автоинкрементных полей.

10. Сколько строк и столбцов содержит результирующий набор данных, получаемый с помощью команды:

SELECT COUNT(\*), COUNT(год\_рождения), COUNT(N\_группы)

FROM Студент;

Ответ: одну строку и три столбца.

11. Когда внутреннее и внешнее соединение двух таблиц дает один и тот же результирующий набор данных?

12. Какие операторы используются только в командах выборки данных с подзапросом?

Ответ: когда внешний ключ дочерней таблицы не содержит значений NULL.

# 7. Сценарий изучения дисциплины

|  |
| --- |
| Теория по модулю 1 / презентация  Задание по модулю 1 (ИПР1)  Тест по модулю 1  Теория по модулю 2 / презентация  Тест по модулю 2  Теория по модулю 3 / презентация  Задание по модулю 3 (ИПР2)  Тест по модулю 3  Теория по модулю 4 / презентация  Тест по модулю 4  Итоговый тест по дисциплине |