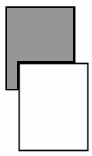
И.А. ДЬЯКОВ

БАЗЫ ДАННЫХ
ЯЗЫК SQL

Обозначение	Определение	LEAP
$R_1 - R_2$	$\{r : r \in R_1 \wedge r \notin R_2\}$	r = (R1) difference (R2) r = (R1) minus (R2)

Пример:



$$R[Q,T] - S = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \end{bmatrix} \cap \begin{bmatrix} 5 & a \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} 3 & a \\ 9 & a \\ 2 & b \\ 4 & b \end{bmatrix}$$

• **Издательство ТГТУ** •

```
one_ext      VARCHAR(4),
        hire_date      DATE DEFAULT 'NOW' NOT NULL,
        dept_no        DEPTNO NOT NULL,
        job_code       JOBCODE NOT NULL,
        job_grade      JOBGRADE NOT NULL,
        job_country    COUNTRYNAME NOT NULL,
        salary         SALARY NOT NULL,
```

Министерство образования Российской Федерации
Тамбовский государственный технический университет

И.А. ДЬЯКОВ

БАЗЫ ДАННЫХ
ЯЗЫК SQL

Утверждено Ученым советом университета
в качестве учебного пособия

Тамбов
Издательство ТГТУ
2004

УДК 681(075)
ББК *з*973-018.3я73
Д931

Рецензент
Кандидат технических наук, доцент
А.Е. Бояринов

Дьяков И.А.
Д931 Базы данных. Язык SQL: Учеб. пособие. Тамбов: Изд-во Тамб. гос. техн. ун-та, 2004. 80 с.
ISBN

Учебное пособие состоит из двух частей, где рассматриваются теоретические основы и практические приемы разработки баз данных. В первой части основное внимание уделено теории баз данных, их моделям и математическим основам. Во второй части рассматривается применение языка SQL для машинной реализации баз данных.

Предназначено для студентов 3 курса дневного отделения специальности 220300 и может быть использовано для студентов других специальностей, где изучается дисциплина «Базы данных».

УДК 681(075)
ББК $_{З}$973-018.3я73

ISBN

Учебное издание

ДЬЯКОВ Игорь Алексеевич

БАЗЫ ДАННЫХ
ЯЗЫК SQL

Учебное пособие

Редактор Т.М. Глинкина
Компьютерное макетирование Е.В. Кораблевой

Подписано в печать 10.02.04
Формат 60 × 84 / 16. Бумага офсетная. Печать офсетная
Гарнитура Times New Roman. Объем: 4,65 усл. печ. л.; 4,5 уч.-изд. л.
Тираж 100 экз. С. 101

Издательско-полиграфический центр
Тамбовского государственного технического университета,
392000, Тамбов, Советская, 106, к. 14

Часть 1
БАЗЫ ДАННЫХ

1 КЛАССИФИКАЦИЯ ДАННЫХ

1.1 Информация и данные

Рассмотрим понятия "*информация*" и "*данные*". В узком смысле информация – это приращение знаний. Понятие "информация" связано с семантикой, т.е. с содержательной интерпретацией данных. В более широком смысле информация отождествляется с некоторыми сведениями [1]. Таким образом, данные рассматриваются как носитель информации. Автоматизированные системы поддерживают модель некоторой части реального мира – предметной области. В зависимости от уровня семантической интерпретируемости, обеспечиваемой используемой моделью, выделяют модели данных (даталогические) и информационные (инфологические) модели.

Структурно данные могут быть представлены тремя уровнями: *концептуальным*, *внешним* и *внутренним*. *Концептуальный уровень* отражает объективные свойства данных, описывающих предметную область. *Внешний уровень*, напротив, отражает субъективные взгляды приложений на данные. В практических случаях внешний уровень является подмножеством концептуального представления. *Внутренний уровень* представления данных определяет машинно-ориентированное, физическое представление данных. В структуре концептуальный уровень находится между внешним и внутренним. Рассмотрим в качестве примера уровни представления списка или списков деталей, входящих в состав изделия. Как видно из схемы (рис. 1.1), на концептуальном уровне могут быть представлены не обязательно данные из списка деталей – это может быть и бухгалтерский документ. Одновременно внешний уровень отражает наше восприятие. Здесь решающим является квалификация, например инженера-конструктора. Внутренний уровень не играет в решении данной задачи главной роли и может меняться в зависимости от типа и уровня развития компьютерной техники.

Современные системы обработки информации выполняют преобразования больших массивов данных. Здесь и далее будем для простоты синонимом понятия "данные" считать термин "информация", так как это ближе для технических систем. САПР также оперирует большим числом данных различного типа и назначения.

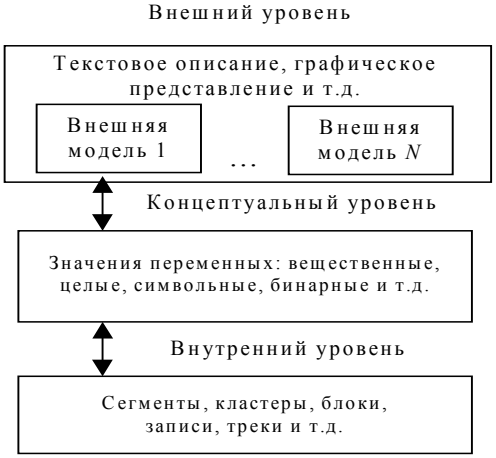


Рис. 1.1 Уровни представления данных

1.2 Основные понятия

Рассмотрим основные определения понятий, часто используемых в информационном обеспечении (ИО) и являющихся основополагающими [2].

Термин *база данных* начал применяться с 1963 г. и записывался на английском языке как data base. По мере развития вычислительной техники, эти два слова были объединены в одно (database). Основной смысл, вкладываемый в термин "база данных" – это база информационной системы. Инструмент в системе обработки данных – это ЭВМ. Информационная база или база данных представляет собой совокупность данных, предназначенных для совместного применения.

Одним из разработчиков теории баз данных, Инглисом (R. Engles), в 1972 г. дано следующее рабочее определение: база данных представляет собой совокупность хранимых операционных данных, используемых прикладными системами некоторого предприятия.

Другой классик теории баз данных К. Дейт в своих работах дает более предметное определение базы данных, как совокупности данных, хранящихся во вторичной памяти (на дисках).

Одновременно российские разработчики теории баз данных предложили понимать под термином "база данных" даталогическое представление информационной модели предметной области. Это наиболее абстрактное и емкое определение.

Государственным комитетом по науке и технике (ГКНТ) в 1982 г. был принят ряд документов, определяющих базу данных как именованную совокупность данных, отражающую состояние объектов и их отношений в рассматриваемой предметной области.

Таким образом, единого мнения по поводу определения термина "база данных" пока не существует. На основе анализа существующих определений и истории развития данной науки в дальнейшем будем пользоваться следующими определениями.

База данных (БД) – структурированная совокупность данных.

Наименьшая единица описания данных называется элементом описания. Совокупность элементов описания, объединенных отношением принадлежности к одному описываемому объекту, называется записью. Например, код типа микросхемы, логическая функция, мощность потребления, коэффициент разветвления в совокупности составляют запись и описывают свойства конкретного объекта − микросхемы.

Система управления базами данных (СУБД) состоит из языковых и программных средств, предназначенных для создания, ведения и эксплуатации баз данных.

Банк данных (БнД) – совокупность баз данных и системы управления базами данных.

Модель данных − формализованное описание, отражающее состав и типы данных, а также взаимосвязь между ними.

На каждом из рассмотренных на рис. 1.1 уровней присутствует своя модель данных. Это так называемый логический уровень моделей данных. По способам отражений связей между данными на логическом уровне различают модели: иерархическую, сетевую и реляционную. Модель называется сетевой, если данные и их связи имеют структуру графа. Если структура отражаемых связей представляется в виде дерева, то модель называют иерархической. Представление данных в форме таблиц соответствует реляционной модели данных. Задание модели данных в БД осуществляется на специальном языке описания данных.

2 ОРГАНИЗАЦИЯ ИНФОРМАЦИОННОГО ОБЕСПЕЧЕНИЯ САПР

САПР – сложная и многокомпонентная система, процессы преобразования данных в которой разнообразны. При этом данные, являющиеся результатом одного процесса преобразования, могут быть исходными для другого процесса. Вследствие этого термин "данные" может иметь различную трактовку в САПР. Так, например, для управляющего монитора в состав данных входит совокупность программных модулей, обеспечивающих процесс проектирования. Для подсистемы математического моделирования и оптимизации к данным относится совокупность исходных и результирующих чисел. Для правильной организации проектных работ, пользователю САПР в качестве данных требуется иметь исходную проектную документацию, справочные данные, типовые проектные решения и т.д. Форма представления данных также может быть различной: в виде текстов, графических изображений, звука, видеороликов. Совокупность данных, используемых всеми компонентами САПР, представляет *информационный фонд САПР*.

Назначение информационного обеспечения (ИО) САПР – реализация информационных потребностей всех составных компонентов САПР. *Основная функция* ИО САПР – ведение информационного фонда. Таким образом, информационное обеспечение САПР имеет две составляющих – это информационный фонд и средства его ведения.

Состав информационного фонда САПР можно определить следующим образом.

1 *Программные модули*, участвующие в процессе проектирования, начиная от операционных систем и заканчивая пакетами прикладных программ. Часть этих данных меняется довольно редко, другая может динамично изменяться, например, при разработке новых методик или нового математического обеспечения САПР.

2 *Исходные и результирующие данные* (цифровые, текстовые, графические, видео, звуковые) для обработки программными модулями. Эти данные меняются часто в процессе проектирования, однако их тип постоянен.

3 *Нормативно-справочная проектная документация* (НСПД) включает справочные данные об элементах проектируемых изделий, технологиях их изготовления и испытаний, унифицированных узлах и конструкциях. Государственные и отраслевые стандарты, руководства и указания, типовые проектные решения, регламентирующие документы, также относятся к НСПД.

4 *Проектная документация* отражает состояние и ход выполнения проекта. Изменяется в процессе проектирования и представляется в виде текстового и графического материала. Возможно звуковое сопровождение и динамическая смена графики (видеоролики). Сюда можно отнести также и готовые проектные решения.

Рассмотрим пример. Пусть надо разработать программу для микропроцессорной системы, управляющей роботом. Тогда в проекте должны быть следующие документы: руководство оператора, руковод-ство программиста, описание языка, описание программ; порядок и методика испытаний. В каждом документе может быть ссылка на другие документы (справочные данные, ГОСТы и пр.). Таким образом, документы могут иметь иерархическую структуру.

Проблема организации и ведения информационного фонда решается в двух направлениях: методологическом и организационном. Первое направление полностью определено методикой автоматизированного проектирования и алгоритмами решения частных задач. Во втором направлении различают следующие способы: использование файловой системы, построение библиотек, использование банков данных, создание информационных программ-адаптеров (для организации межмодульного интерфейса).

3 БАНКИ ДАННЫХ. ОБЩИЕ ТРЕБОВАНИЯ К НИМ, ИХ ТРАДИЦИОННАЯ АРХИТЕКТУРА

Рассмотрим более полное определение [3] банков данных. *Банк данных* – это система специальным образом организованных данных (баз данных), программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

Термин "банк данных" не является общепризнанным. Наиболее близким к нему в англоязычной литературе является термин "система баз данных" (*data base system*). Система баз данных включает базу данных, СУБД, соответствующее оборудование и персонал. Понятие "система баз данных" уже, чем БнД, так как "банк" обозначает то, что хранится в нем и всю инфраструктуру, но по сути они одинаковы.

БнД является сложной человеко-машинной системой, включающей в свой состав различные взаимосвязанные и взаимозависимые компоненты (рис. 3.1). Ядром БнД является база данных.

Информационный компонент состоит из БД, схем БД, словарей данных. Последние играют в САПР особенно важное значение.

Программные средства БнД представляют собой сложный комплекс, обеспечивающий взаимодействие всех частей информационной системы при ее функционировании (рис. 3.2).

Основу программных средств представляет СУБД. В ней можно выделить ядро СУБД, обеспечивающее организацию ввода, обработки и хранения данных и средства настройки, тестирование и утилиты вспомогательных функций для восстановления БД, сборы статистики о функционировании БД и др.

Компиляторы и трансляторы являются важной компонентой языковых средств СУБД. Все СУБД работают под управлением операционной системы (ОС).

Рис. 3.1 Компоненты банков данных

Рис. 3.2 Программные средства банков данных

Для обработки запросов к БД разрабатывается соответствующее прикладное программное обеспечение.

Языковые средства БнД обеспечивают интерфейс пользователей разных категорий с банком данных и базируются на языковых средствах СУБД. Их спектр достаточно широк (рис. 3.3).

Категории языковых средств различаются по функциональным возможностям:

– языки ввода данных по запросу (устарели);

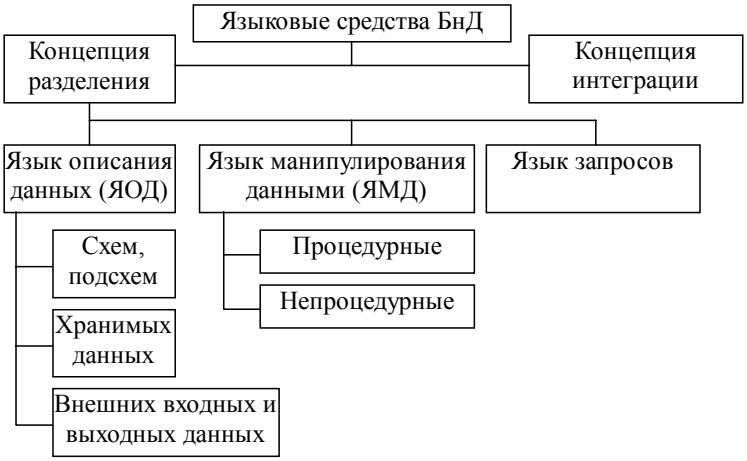


Рис. 3.3 Языковые средства банков данных

данных

–языки запросов – обновлений (сложные запросы по нескольким взаимосвязанным записям);

–генератор отчетов для выбора данных и формирования в виде формы требуемого документа;

– графические языки, аналогичны генератору отчетов, данные отображаются в виде диаграмм, графиков и т.п.;

– языки принятия решений (пролог);

– генераторы приложений для автоматизированной генерации программ;

– параметризированные пакеты прикладных программ (ППП), для генерирования собственных отчетов и запросов;

– языки приложений.

По форме представления различают аналитические, табличные и графические языковые средства. Такая классификация справедлива и для ЯОД и для ЯМД.

Современные СУБД включают в свой состав несколько языковых средств разного уровня.

В качестве *технических средств* для БнД чаще всего используются универсальные ЭВМ, стандартный набор периферийных устройств и сетевого оборудования. Для создания и эксплуатации систем баз данных используются специальные технические средства, например серверы, накопители на магнитных лентах (стримеры), накопители на оптических носителях (CD R/W).

К *организационно-методическим* средствам БнД относятся различные инструкции, методические и регламентирующие документы, для пользователей различных категорий.

Группа специалистов, обеспечивающих разработку, функционирование и развитие систем баз данных, называется администратором банка данных (АБД).

3.1 Классификация систем баз данных

Банки или системы баз данных являются сложными системами и их классификация может быть проведена по различным признакам, относящимся как в целом к БнД, так и к его компонентам. Большинство классификационных признаков относится к центральной компоненте БнД – базе данных. Рассмотрим некоторые из них.

По *форме* представления данных системы разделены на видео- и аудиосистемы, также мультимедиа, символьные. Пока наибольшее применение находят базы данных, содержащие обычные символьные данные. Они, в свою очередь разделены на неструктурированные, частично структурированные и структурированные (семантические сети, обычный текст и построение по модели). По *типу хранимой информации* БД можно разделить на документальные, фактографические и лексикографические. По характеру *организации* и хранения данных и *обращению* к ним различают локальные (однопользовательские), общие (интегрированные), распределенные и объектно-ориентированные. На рис. 3.4 приведена структурная схема классификации БнД по различным признакам

3.2 Свойства систем баз данных

Скорость. Время реакции, т.е. получение ответа на запрос.

Доступность. Какие данные, содержащиеся в БД, доступны данной категории пользователей.

Гибкость. Возможность получить ответ на сложные запросы.

Целостность. Снижение избыточности данных, согласованность данных при упорядочении обновления.

3.3 Функции СУБД

Система управления базой данных представляет собой программное обеспечение, которое управляет доступом к базе данных. Это происходит следующим образом [4].

1 Пользователь выдает запрос на доступ, применяя определенный подъязык данных (обычно *SQL*).

2 СУБД перехватывает этот запрос и анализирует его.

3 Затем СУБД просматривает внешнюю схему для этого пользователя, концептуальную, внутреннюю схему и определяет структуру хранения.

4 На последнем шаге СУБД выполняет необходимые операции над хранимой базой данных.

Описанные действия выполняются благодаря функциям СУБД. Рассмотрим их более подробно.

– Определение данных. СУБД должна допускать определение данных в исходной форме и преобразовывать эти определения в форму соответствующих объектов.

– Обработка данных. СУБД должна уметь обрабатывать запросы пользователя по выбору, изменение или удаление существующих данных в базе данных или добавление новых данных. Программное обеспечение должно быть построено таким образом, чтобы реализовать *планируемые* и *непланируемые* запросы. К планируемым относятся запросы, необходимость которых предусмотрена заранее, непланируемым (специальные) – наоборот. Планируемые запросы обычно осуществляются из написанных заранее приложений, а непланируемые запросы по определению производятся интерактивно.

– Безопасность и целостность данных. Достигается контролем пользовательских запросов и пресечением попыток нарушения правил безопасности и целостности данных, определяемых администратором БД.

– Восстановление и дублирование данных. Выполняется СУБД или другим программным компонентом, называемым администратором транзакций.

– Словарь данных. СУБД должна обеспечивать функцию словаря данных. Сам словарь данных является системной БД, содержащей данные о данных, например, исходные и объектные схемы внешнего и концептуального уровня, перекрестные ссылки программ или частей БД, отчеты для различных пользователей и т.д.

– Производительность. Все перечисленные функции должны выполняться с максимально возможной эффективностью.

Следует отличать СУБД от *системы управления файлами*, не учитывающей внутреннюю структуру хранимых данных (записей), имеющих особую поддержку безопасности и целостности данных. Запросы, основанные на знании структуры данных, здесь не обрабатываются.

Равноправным с СУБД программным компонентом можно считать *систему управления передачей данных*. Применяется такая система при обработке запросов пользователей, физически удаленных от СУБД. Запросы и результат работы СУБД оформляются в виде коммуникационных сообщений. Наличие такой системы может быть отражено в определении БнД как системы баз данных и передачи данных.

Основные функции СУБД, рассмотренные выше, можно дополнить еще несколькими входящими в уже известные [5].

Для функции *производительности* характерно применение функции непосредственного управления данными во внешней памяти и управление буферами оперативной памяти. Первая функция включает обеспечение необходимых структур внешней памяти как для хранения непосредственных данных, так и служебных целей, например, для хранения индексов. Еще одним реальным способом увеличения скорости является буферизация данных в оперативной памяти, независимо от буферов ОС. Развитие СУБД поддерживают собственный набор буферов ОЗУ с собственной дисциплиной их замены.

С функцией *обработки данных* связано управление транзакциями. *Транзакция* – это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется и СУБД фиксирует (*commit*) изменения БД, произведенные ею во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. При помощи транзакций поддерживается логическая целостность БД за счет объединения элементарных операций над разными файлами в одну транзакцию. Например, при приеме нового сотрудника информация должна быть внесена в разные файлы "сотрудники" и "отделы" одной БД. Таким образом, поддержание механизма транзакций – обязательное условие однопользовательских, а тем более многопользовательских СУБД.

Функции *восстановления и дублирования* данных в СУБД принято называть журнализацией. В результате аварийного выключения питания компьютера, сбоя в программе, выхода из строя носителя информации происходит потеря данных в БД. В первых двух случаях (мягкий сбой) восстановить данные можно ликвидацией последствий одной транзакции, в третьем – (жесткий сбой) только копированием из архива. Соблюдение надежности системы достигается избыточностью хранимых данных или ведением журнала изменений БД. *Журнал* – это особая часть БД, недоступная пользователям СУБД и поддерживаемая особенно тщательно, в которую поступают записи обо всех изменениях основной части БД.

Изменения БД *журнализируются* на разных уровнях, например, операции удаления строки из таблицы реляционной БД или операции модификации страницы внешней памяти.

Самая простая ситуация восстановления – индивидуальный откат транзакции. В более сложных случаях применяют одновременно журнал и архивную копию. Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу, воспроизводится работа всех транзакций, которые закончились к моменту сбоя.

Для *функции определения данных* характерно наличие языкового процессора. В ранних СУБД поддерживалось несколько специализированных по функциям языков. Чаще всего выделялись два – язык описания данных (ЯОД) или схемы БД (*DDL* (*SDL*) *Data* (*Schema*) *Definition Language*) и язык манипулирования данными (ЯМД) (*DML – Data Manipulation Language*). В современных СУБД обычно поддерживается единый интегрированный язык, например *SQL* (*Structured Query Language*).

3.4 Структура СУБД

Организация типичной СУБД и состав ее компонентов соответствуют рассмотренному нами набору функций.

Логически, например в реляционной СУБДб можно выделить наиболее внутреннюю часть – ядро СУБД, компилятор языка БД, подсистему поддержки времени выполнения и набор утилит (рис. 3.5). В некоторых системах эти части выполняются явно, в других нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором *SQL* (или в подсистеме поддержки выполнения

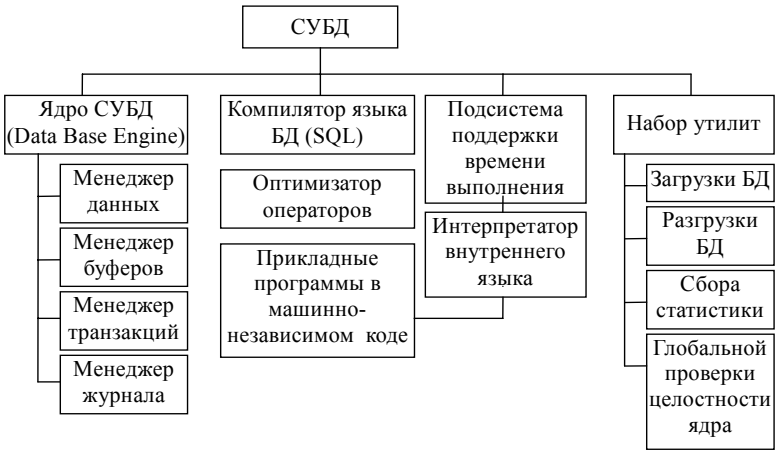


Рис. 3.5 Логическая структура СУБД

таких программ), и утилитах БД. Ядро СУБД является основной резидентной частью СУБД. *Компилятор* непроцедурного языка *SQL* решает проблему выполнения оператора, его оптимизации и генерирования программного кода. Реальное выполнение оператора осуществляется подсистемой поддержки времени выполнения, представляющей собой интерпретатор внутреннего языка. Утилиты программируются с использованием интерфейса ядра СУБД или с проникновением внутрь ядра.

4 МОДЕЛИ ДАННЫХ

Традиционно СУБД делятся по типу модели данных на *иерархические, сетевые и реляционные*. Такое деление моделей и СУБД основывается на характере связей между записями. При всей разнице в терминологии можно считать, что основными компонентами любой из этих моделей являются файлы, которые состоят из записей. Различают *внутризаписную* и *межзаписную* структуры.

В отличие от моделей внутризаписная структура может быть линейной либо иерархической. При *линейной структуре* запись состоит из простых элементов, следующих один за другим. Такая структура считается нормализованной. *Иерархическая* структура включает простые и составные компоненты, например векторы, повторяющиеся и

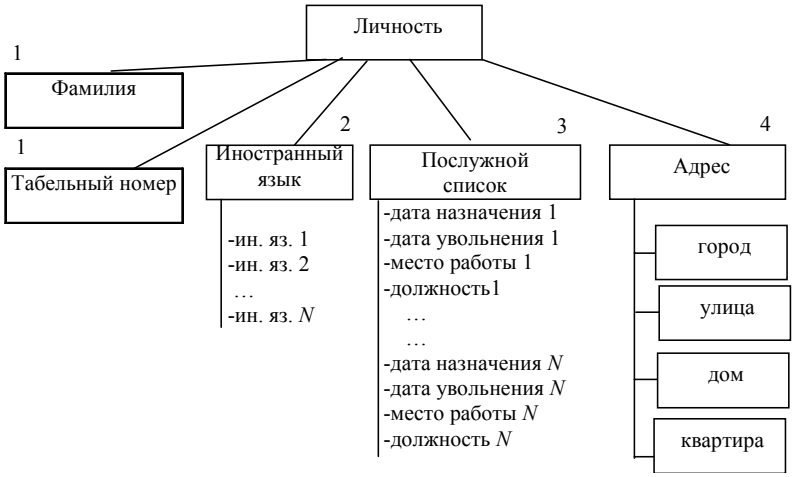


Рис. 4.1 Иерархическая структура записи:
1 – простой элемент; *2* – вектор (набор) однотипных элементов;
3 – повторяющаяся группа (набор разноплановых элементов);
4 – неповторяющаяся группа (набор разнотипных элементов)

допускает многоуровневость (рис. 4.1).

Состав записей в структуре может быть постоянным или переменным. Например, если один из сотрудников окончил университет и имеет ученую степень и ученое звание с данными их присвоения, то другой сотрудник может их вообще не иметь. Это значит, что поля в соответствующих записях просто отсутствуют. Основными характеристиками записи являются ее тип (символьный, числовой, дата, логический и т.д.) и длина (фиксированная, переменная и неопределенная).

Межзаписная структура или модель данных, как было уже отмечено, бывает иерархической, сетевой и реляционной. Рассмотрим их более подробно.

4.1 Иерархическая модель

В классических *иерархических моделях* имеется один файл, который является входом в структуру (корень дерева). Остальные файлы связаны друг с другом таким образом, что каждый из них за исключением корневой вершины имеет ровно одну исходную вершину ("предок") и любое число подчиненных вершин ("потомков"). Между записью файла-"предка" и записями порожденного файла имеется отношение "один ко многим" (1:M). Как частный случай может быть отношение "один к одному". Различают также тип связи "многие ко многим" (M:M). Типичным представителем иерархических СУБД можно считать систему IMS (Information Management System).

Иерархическая БД состоит из упорядоченного набора деревьев, а точнее из упорядоченного наборов нескольких экземпляров одного типа дерева.

Пример схемы иерархической БД "Деталь" показан на рис. 4.2. Здесь отдел является предком ("родителем") для начальника и сотрудников, а начальник и сотрудники – потомки отдела.

Состав информации базы данных "Деталь":

1 Для каждой детали: шифр детали (уникальный), название и краткое описание назначения детали, информация о технических характеристиках и наличии на складе;

2 Характеристики детали включают необходимые технические данные, в примере это вес детали, материал и ее габаритные размеры;

3 Получение деталей на склад характеризуется датой получения, количеством деталей и уникальным номером накладной;

4 Для каждой детали существует предприятие-изготовитель, имеющее почтовый адрес, название и шифр;

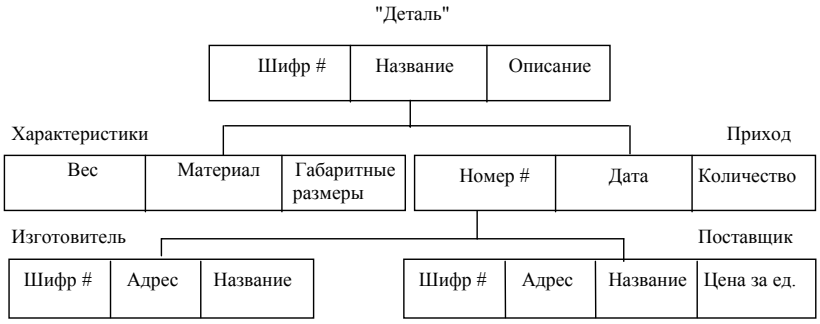


Рис. 4.2 Один экземпляр дерева иерархической БД

5 Деталь может быть куплена через посредников, поэтому отдельно записывается название поставщика, его почтовый адрес, цена за одну деталь и шифр поставщика.

Таким образом, в базе данных есть 5 типов сегментов: Деталь, Характеристики, Приход, Изготовитель, Поставщик. Деталь – корневой сегмент. Остальные – подчиненные типы сегментов. Каждому подчиненному типу сегмента соответствует исходный тип сегмента. Каждому исходному типу сегмента соответствует по крайней мере один порожденный тип сегмента.

Для одного экземпляра любого заданного типа сегмента может существовать любое количество экземпляров, в том числе и нуль, каждого из его порожденных типов сегментов.

Например, имеется один экземпляр корневого типа сегмента Деталь, один экземпляр Характеристики, два экземпляра Приход, один экземпляр Изготовитель и два экземпляра Поставщик (рис. 4.3). Первому экземпляру Приход подчинен один экземпляр Изготовитель и два экземпляра Поставщик. Для второго экземпляра Приход пока неизвестны Изготовитель и Поставщик, например собственное экспериментальное производство.

Иерархический порядок в БД считается очень важным, т.к. определяет доступ к информации. Поиск в БД осуществляется по составному ключу, т.е. для выполнения запроса надо указывать значение ключа на каждом уровне иерархии.

Значение ключа будет состоять из значения поля упорядочения данного сегмента с кодом типа сегмента в качестве префикса, которому предшествует значение ключа его исходного сегмента.

Например, для экземпляра сегмента Поставщик, соответствующего адресу Тамбов, значение ключа будет иметь вид: 1 D1 1 P500 2 M10.

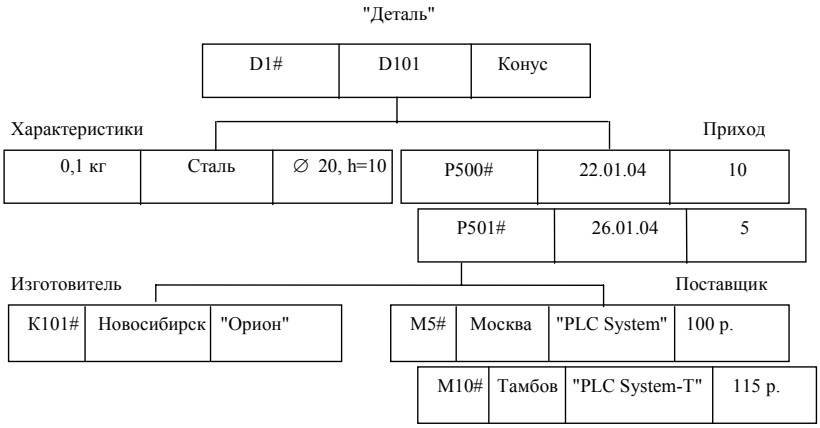


Рис. 4.3 Экземпляры дерева иерархической БД "Деталь"

Иерархический порядок определяется здесь возрастанием значений ключа иерархического упорядочения.

Примерами типичных операторов манипулирования иерархически организованными данными могут быть следующие:

- найти указанное дерево БД (например, деталь D101);
- перейти от одного дерева к другому;
- перейти от одной записи к другой внутри дерева (например, от D101 к описанию Конус);

– перейти от одной записи к другой в порядке обхода иерархии;
– вставить новую запись в указанную позицию;
– удалить текущую запись.

Целостность ссылок между предками и потомками поддерживается автоматически. Основное правило здесь: *никакой потомок не может существовать без своего родителя*.

Заметим, что ссылки между записями различных деревьев не поддерживаются.

4.2 Сетевая модель

В *сетевых* моделях, если на нее не накладывается никаких ограничений, в принципе любой файл может быть точкой входа в систему, каждый из файлов может быть связан с произвольными числами других файлов, и между записями связанных файлов могут быть любые отношения 1 : 1, 1 : M, M : M. Однако в реальных СУБД на модель накладываются различные ограничения.

Во многих сетевых СУБД не поддерживается отношение M : M. В таких моделях каждая связь между парой файлов определяется отдельно, и для каждой из них один файл в этой паре объявляется "владельцем", а другой "членом". Отношение между записями 1 : M.

Связи между файлами в иерархических и сетевых моделях определяются при описании структуры БД и физически передаются при помощи различных указателей.

Типичным представителем является *Integrated Database Manegement System* (*IDMS*) компании *Cullinet software, inc*. Архитектура системы основана на предложениях *Data Base Task Group* (*DBTG*) комитета по языкам программирования *Counference of Data System Languages* (*CODASYL*).

Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре потомок может иметь любое число предков.

Сетевая БД состоит из набора записей и набора связей между ними, а точнее из набора экземпляров каждого типа из заданного в схеме БД набора типов записи и набора экземпляров каждого типа из заданного набора типов связи.

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка (рис. 4.4).

Для данного типа связи *L* с типом записи предка *P* и типом записи потомка *C* должны выполняться два условия: каждый экземпляр типа P является предком только в одном экземпляре *L*; каждый экземпляр *C* является потомком не более чем в одном экземпляре *L*.

На формирование типов связи не накладываются особые ограничения, возможны, например следующие ситуации.

– Тип записи потомка в одном типе связи *L*1 может быть типом записи предка в другом типе связи *L*2 (как в иерархии) (рис.4.5, *а*).
– Данный тип записи *P* может быть типом записи предка в любом числе типов связи (рис. 4.5, *б*).
– Данный тип записи *P* может быть типом записи потомка в любом числе типов связи (рис. 4.5, *в*).
– Может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; если *L*1 и *L*2 – два типа связи с одним и тем же типом записи предка *P* и одним и тем же типом записи потомка *C*, то правила, по которым образуется родство, в разных связях могут различаться (рис. 4.5, *г*).

Рис. 4.6 Пример сетевой БД

– Типы записи *X* и *Y* могут быть предком и потомком в одной связи и потомком и предком в другой (рис. 4.5, *д*).

– Предок и потомок могут быть одного типа записи (рис. 4.5, *е*).

На рис. 4.6 показан простой пример сетевой схемы БД.

Примерный набор операций может быть таковым:

– найти конкретную запись в наборе однотипных записей, например, программиста Сидорова;
– перейти от предка к первому потомку (к первому сотруднику отдела САПР);
– перейти к следующему потомку в некоторой связи (от Сидорова к Иванову);
– перейти от потомка к предку по некоторой связи (найти отдел Сидорова);
– создать новую запись;
– удалить запись;

– модифицировать запись;
– включить связь;
– исключить из связи;
– переставить в другую связь и т.д.

4.3 Достоинства и недостатки иерархических и сетевых СУБД

Достоинства:
– развитие средства управления данными во внешней памяти на низком уровне;
– возможность построения вручную эффективных прикладных систем;
– возможность экономии памяти за счет разделения подобъектов (в сетевых системах);

Недостатки:
– слишком сложно пользоваться;
– фактически необходимы знания о физической организации;
– прикладные системы зависят от этой организации;
– их логика перегружена деталями организации доступа.

4.4 Реляционная модель

В *реляционной* модели используется своеобразная терминология, но это не меняет сущности модели. Так, на логическом уровне элемент чаще всего называют атрибутом; кроме того, для него используются термины "колонки", "столбец", "поле". Совокупность атрибутов образует кортеж (ряд, запись, строку). Совокупность кортежей образует отношение (таблицу или файл БД).

Связи между файлами в реляционной модели в явном виде могут не описываться. Они устанавливаются динамически в момент обработки данных по равенству значений соответствующих полей. Структуры записей в реляционных БД – линейные.

Каждое отношение по определению имеет ключ, т.е. атрибут (простой ключ) или совокупность атрибутов (составной ключ), однозначно идентифицирующий кортеж.

Атрибут или группа атрибутов, которая в рассматриваемом отношении не является ключом, а в другом отношении ключом является, называется *внешним* ключом.

Если какая-то таблица содержит внешний ключ, то она: а) логически связана с таблицей, содержащей соответствующий первичный ключ; б) эта связь имеет характер один ко многим.

Реляционная модель была разработана Коддом в 1969 – 70 гг. на основе математической теории отношений и опирается на систему понятий, важнейшими из которых являются таблица, отношение, строка, столбец, первичный ключ, внешний ключ.

Реляционной считается такая модель данных, в которой все данные представлены для пользователя в виде прямоугольных таблиц значений данных, и все операции над базой данных сводятся к манипулированию таблицами. Таблица состоит из строк и столбцов и имеет имя, уникальное внутри базы данных. Таблица отражает тип объекта реального мира (сущность), а каждая ее строка – конкретный объект. Рассмотрим основные понятия реляционных моделей на примере таблицы "Деталь" (рис. 4.7).

Пусть в таблице содержатся сведения о всех деталях, хранящихся на складе, а ее строки содержат набор значений атрибутов каждой конкретной детали.

Каждый столбец имеет имя, которое обычно записывается в верхней части таблицы. Оно должно быть уникальным в таблице, однако различные таблицы могут иметь столбцы с одинаковыми именами. Любая таблица должна иметь по крайней мере один столбец; столбцы расположены в таблице в соответствии с порядком следования их имен при ее создании. В отличие от столбцов (атрибутов), строки не имеют имен, порядок их следования не определен, а количество не ограничено.

Любая таблица имеет один или несколько столбцов, значения которых однозначно идентифицирует каждую ее строку.

Первичный ключ в примере (рис. 4.7) – это столбец "Номер детали".

Значения атрибутов выбираются из наименьшей информационной единицы – домена. Другими словами, домен – это множество всех возможных значений атрибута объекта. Рассмотрим еще два понятия "Степень" и "Кардинальное число". Под кардинальным числом отношения понимают количество кортежей, а степень отношения – это количество атрибутов данного отношения.

Взаимосвязь таблиц является важнейшим элементом реляционной модели данных. Она поддерживается внешними ключами. Рассмотрим пример, в котором БД хранит информацию о сотрудниках (таблица "Сотрудник") и руководителях (таблица "Руководитель") в некоторой организации (рис. 4.8).

Первичный ключ таблицы "Руководитель" – столбец "Номер". Столбец "Фамилия" не является уникальным, поэтому не применяется в качестве первичного ключа. Столбец "Номер Руководителя" является внешним ключом в таблице "Сотрудник".

В БД дополнительно к самим данным должен храниться словарь данных и другие объекты, например, экранные формы, отчеты, просмотры (*views*) и прикладные программы.

Ограничения целостности в реляционных БД требуют, чтобы, например, значения атрибутов выбирались только из соответствующего домена, или внешний ключ не может быть указателем на несуществующую строку в таблице (целостность по ссылке).

Рассмотрим более подробно понятие "Отношение".

В реляционных моделях следует различать переменные отношений и значения отношений. Переменная отношения – это обычная переменная, такая же, как и в языках программирования, т.е. именованный объект, значение которого может изменяться со временем. А значение этой переменной в любой момент времени и будет *значением отношения*.

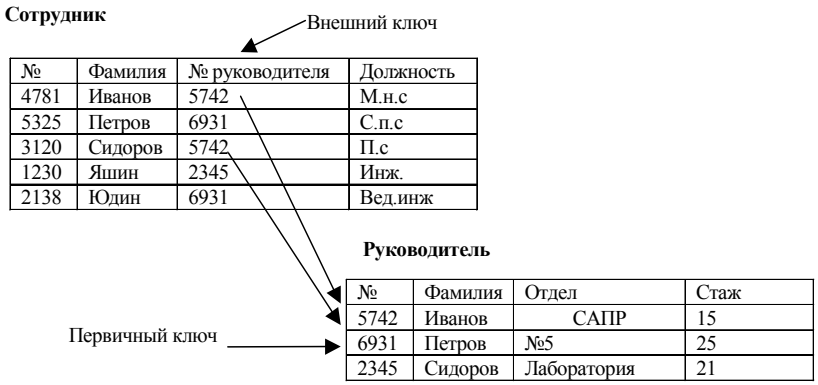


Рис. 4.8 Взаимосвязь таблиц

Если говорить о таблице, то точнее можно сказать "Переменная базового отношения" (базовая таблица). Переменная отношения в разное время представляет разные таблицы. В них будут разные строки, а столбцы будут одинаковыми.

Введем понятие "Значение отношения".

Отношение R на множестве доменов D_1, D_2,…, D_n (не обязательно различных) содержит две части: *заголовок* и *тело*. В терминах таблиц заголовок – это строка заголовков столбцов, а тело – это множество строк данных.

1 Заголовок содержит фиксированное множество атрибутов, а точнее пар <имя_атрибута, имя_домена> $\{<A_1:D_1>, <A_2:D_2>,\ldots, <A_n:D_n>\}$, причем каждый атрибут $A_j \in D_j \; \forall \; j = 1,\ldots,n$. Имена A_j разные.

2 Тело содержит множество кортежей. Каждый кортеж содержит множество пар <имя атрибута; значение атрибута> $\{<A_1:Vi_1>, <A_2:Vi_2>,\ldots,<A_n:Vi_n>\} \; \forall \; i = 1,\ldots,m$; m – количество кортежей. В каждом кортеже есть пара $<A_j:Vi_j> \; \forall \; A_j$ в заголовке. Для любой пары $<A_j:Vi_j>$ Vi_j является значением из уникального домена D_j, связанного с A_j.

Пусть R – переменная отношения. Переменная R будет иметь разные значения в разное время. Однако все возможные значения переменной R будут иметь одинаковые заголовки.

4.5 Свойства отношений

Можно выделить следующие свойства отношений:

- нет одинаковых кортежей;
- картежи не упорядочены сверху вниз;
- атрибуты не упорядочены слева направо;
- все значения атрибутов атомарные.

Первое свойство следует из того факта, что тело отношения – это математическое множество (кортежей), а множество не содержит одинаковых элементов. Это свойство показывает отличие отношения и таблицы. Следует заметить, что стандарт языка *SQL* допускает, чтобы таблицы содержали одинаковые строки.

Второе свойство следует из того, что тело отношения – это математическое множество, а простые множества не упорядочены. В каком бы порядке не расположены кортежи, отношение останется тем же самым. Здесь тоже отличие от таблицы.

Третье свойство основано на понятии множества (атрибутов) и, следовательно, упорядочение не обязательно. С точки зрения модели данных невозможно различить первый, следующий или последний атрибуты.

Последнее свойство можно сформулировать еще и так: отношение не содержит групп повторения. В терминах таблицы это значит, что в каждой позиции пересечения строки и столбца расположено только одно значение, а не набор значений. Такие отношения считают нормализованными, или представленными в первой нормальной форме (1НФ).

В различных системах встречаются некоторые из видов отношений.

1 Именованное отношение – это переменная отношения, определенная в СУБД посредством операторов открытия или создания отношения.

2 Базовое отношение – это наиболее важное, автономное именованное отношение, являющееся частью БД.

3 Произвольное отношение определяется через другие именованные отношения, и в конечном счете, через базовые отношения.

4 Выражаемое отношение получается из набора именованных отношений через некоторые реляционные выражения (результат отчетов).

5 Представлением (просмотром) называется именованное производное отношение. Представления виртуальны и представлены в системе через определения в терминах других именованных отношений.

6 Снимки (*shnapshot*) – это именованные производные отношения, как и представления, но реальны в отличие от последних. Создание снимки похоже на выполнения запроса, результат которого сохраняется в БД.

7 Результат запроса – именованное производное отношение, полученное в результате некоторого определенного запроса.

8 Промежуточным результатом называется именованное производное отношение, являющееся результатом некоторого реляционного выражения, вложенного в другое, большее выражение.

9 Хранимым называется отношение, которое поддерживается в физической памяти. Хранимое отношение не всегда совпадает с базовым.

Каждое отношение имеет некоторую интерпретацию, причем пользователи должны знать ее для эффективного использования БД. Например, интерпретация отношения Деталь может быть следующей.

Деталь с определенным номером (Номер_детали) имеет определенное имя (Название_детали), имеется на складе в количестве (Кол-во_детали) весом (Вес) килограмм каждая и выполнена из (Материал); кроме того нет двух деталей с одинаковыми номерами.

Это утверждение называется *предикатом*, или функцией значения истинности, в нашем примере – функцией пяти аргументов. Подстановка значений аргументов приводит к получению выражения, имеющего истинное либо ложное утверждение. Операции вставки новых кортежей, обновления существующих выполняются в случае истинного предиката для данного кортежа, т.е. при соблюдении правил целостности.

4.6 Потенциальные ключи

К реляционным базам данных применяются два общих правила целостности, и относятся они к потенциальным (первичным) ключам и ко внешним ключам.

Если говорить нестрого, то первичный ключ – это уникальный идентификатор для некоторого отношения. Однако первичный ключ является частным случаем общего понятия – *потенциального ключа.* Рассмотрим это понятие. Пусть R – некоторое отношение. Тогда потенциальный ключ K для R – это подмножество множества атрибутов R, обладающих следующими свойствами:

– свойством уникальности; нет двух различных кортежей в отношении R с одинаковым значением K;

– свойство неизбыточности; никакое из подмножеств K не обладает свойством уникальности.

Данное определение относится к значениям отношения, а не к переменным отношения. Для переменных отношения определение потенциального ключа дополняется следующим образом. Пусть R – некоторая *переменная* отношения. Тогда потенциальный ключ K для R – это подмножество множества

атрибутов R, *всегда* обладающее свойствами уникальности и неизбыточности. Свойство уникальности рассматривается для различных кортежей *в текущем значении* переменной R.

На практике отношения чаще всего имеют только один потенциальный ключ, хотя их может быть несколько. Например, в периодической системе элементов химические элементы имеют уникальное имя, обозначение (Cu, Pb, Au,…) и атомное число. Это уже три различных потенциальных ключа, или составной потенциальный ключ, состоящий более чем из одного атрибута. Потенциальные ключи не должны включать лишних атрибутов для идентификации уникальности. Это и есть свойство неизбыточности.

Если в отношении "Деталь" определить потенциальный ключ, как комбинацию {номер_детали, материал} вместо "номер_детали", тогда система не сможет соблюдать ограничение, обеспечивающее уникальность в "локальном смысле", т.е. для одного типа материала.

На практике физическое понятие индекса часто играет роль потенциального ключа.

Причина важности потенциальных ключей состоит в том, что они обеспечивают *основной механизм адресации на уровне кортежей.* Другими словами, единственный гарантируемый системой способ точно указать кортеж – это указать значение некоторого потенциального ключа.

Таким образом, базовое отношение может иметь больше одного потенциального ключа. В реляционной модели один из потенциальных ключей выбирают в качестве *первичного ключа.* Если есть еще потенциальные ключи в этом базовом отношении, то их считают альтернативными (обозначение элемента и название элемента, атомное число для примера периодической системы элементов).

Реляционная модель традиционно требует, чтобы внешние ключи в точности соответствовали первичным ключам, а не просто потенциальным ключам. Уточним определение внешнего ключа.

Пусть R_2 – базовое отношение. Тогда *внешний ключ FK* в отношении R_2 – это подмножество множества атрибутов R_2, такое, что:

– существует базовое отношение R_1 с потенциальным ключом *CK*,

– каждое значение *FK* в текущем значении R_2 всегда совпадает со значением *CK* некоторого кортежа в текущем значении R_1.

Внешний ключ может быть составным, тогда и только тогда, когда соответствующий потенциальный ключ также составной. Аналогично определяется соответствие для простого ключа. Значение внешнего ключа представлено ссылкой к кортежу с соответствующим потенциальным ключом. Для баз данных иногда строят ссылочные (целевые) диаграммы. Например, если объединить отношения "Сотрудник" (С) и "Руководитель" (Р) в базу Предприятие (П), то можно записать диаграмму С←П→Р, где стрелка обозначает внешний ключ. Иногда указывают над ссылкой имя атрибута $\text{С} \xleftarrow{\text{ном. сотр.}} \text{П} \xrightarrow{\text{ном. рук.}} \text{Р}$.

Отношение может быть одновременно ссылочным и ссылающимся, например для R_2: $R_3 \to R_2 \to R_1$.

Пусть в отношении R_n, $R_{n\text{-}1}$,…, R_2, R_1 имеется ссылочное ограничение из R_n в $R_{n\text{-}1}$ и $R_{n\text{-}1}$ в $R_{n\text{-}2}$ и … и R_2 в R_1 или $R_n \to R_{n\text{-}1} \to R_{n\text{-}2} \to \ldots \to R_2 \to R_1$.

Тогда цепочки стрелок из R_n в R_1 представляют *ссылочный путь* из R_n в R_1.

Отношения R_1 и R_2 в определении внешних ключей не обязательно различны. Такие самоссылающиеся отношения представляют собой отдельный случай. Отношения R_n, $R_{n\text{-}1}$, …, R_2, R_1 образуют *ссылочный цикл* ($R_n \to \ldots R_1 \to R_n$).

Вместе с понятием внешнего ключа реляционная модель включает правило *ссылочной целостности*, т.е. БД не должна иметь несогласованных ключей. Если $R_i \to R_j$ (ссылается), то R_j должно существовать.

Для корректного применения внешних ключей существуют некоторые правила. Что будет происходить при удалении или обновлении объекта ссылки внешнего ключа? Если использовать правила *ограничения* и *каскадирования,* то целостность БД не нарушается.

Свойство ограничения заключается в том, что операция (удаление или добавление) выполняется до момента, когда не будет существовать соответствующих ссылок кортежей. Свойство каскадирования состоит в том, что операцию выполняют столько раз, сколько кортежей будет обнаружено.

Пусть R_1 и R_2 имеют ссылочное отношение $R_2 \to R_1$.

Тогда удаление кортежа из R_1 влечет удаление определенных кортежей в R_2.

Пусть имеем ссылочное отношение $R_3 \to R_2 \to R_1$. Тогда, если операция удаления из R_2 выполняются только для R_2, то нарушается ссылка $R_3 \to ? \to R_1$. Следовательно, такая операция не должна быть выполнена.

В реляционных моделях существует еще один фактор, связанный с потенциальными ключами – это нуль значения (*NULL*). Когда говорят о *null*-значении, то в основном подразумевают базис, используемый при решении проблемы отсутствующей информации. Эта проблема почти не имеет математической проработки.

5 РЕЛЯЦИОННАЯ АЛГЕБРА

5.1 Введение в реляционную алгебру

Реляционная алгебра, определенная Коддом, состоит из восьми операторов, составляющих две группы.

В первую группу входят традиционные операции над множествами: объединение ($\cup$), пересечение ($\cap$), вычитание (–) и декартово произведение (*). Все операции модифицированы с учетом того, что их операндами являются отношения, а не произвольные множества.

Вторую группу образуют специальные реляционные операции: выборка, проекция, соединение и деление.

Рассмотрим подробнее результаты этих операций над отношениями.

Объединение $\cup$. Возвращает отношение, содержащее все кортежи, которые принадлежат одному из двух определенных отношений или обоим (рис. 5.1, *а*).

Пересечение $\cap$. Возвращает отношение, содержащее все кортежи, которые принадлежат одновременно двум определенным отношениям (рис. 5.1, *б*).

Вычитание –. Возвращает отношение, содержащее все кортежи, которые принадлежат первому из двух определенных отношений и не принадлежат второму (рис. 5.1, *в*).

Декартово произведение *. Возвращает отношение, содержащее всевозможные кортежи, которые являются сочетанием двух кортежей, принадлежащих соответственно двум определенным отношениям (рис. 5.1, *г*).

Выборка – возвращает отношение, содержащее все кортежи из определенного отношения, которое удовлетворяет определенным условиям. С точки зрения алгебраических операций это *ограничение* (рис. 5.2, *а*).

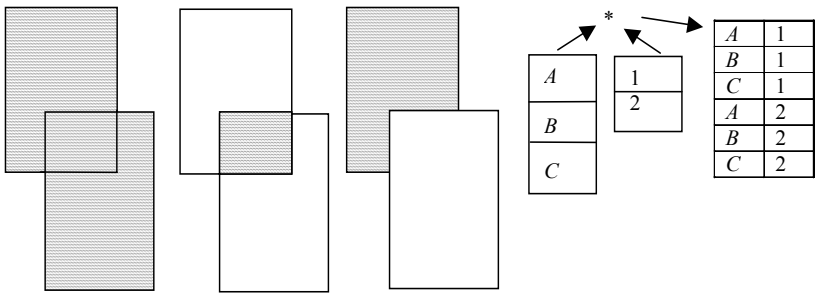
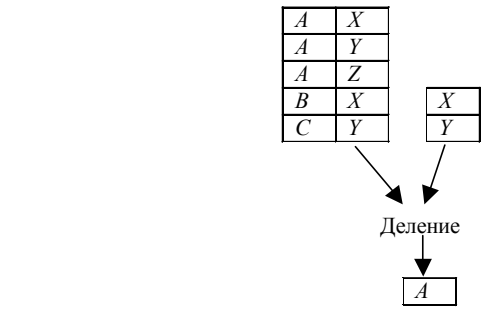


Рис. 5.1 Операции над множествами:
а – объединение; *б* – пересечение;
в – вычитание; *г* – декартовое произведение

кортежи (подкортежи) определенного отношения после исключения из него некоторых атрибутов (рис. 5.2, *б*).

Соединение – возвращает отношение, кортежи которого – это сочетания двух кортежей (принадлежащих соответственно двум определенным), имеющих общее значение для одного или нескольких общих атрибутов этих двух отношений. Общие значения в результирующем кортеже появляются только один раз, а не дважды. Такое соединение называют естественным соединением (рис. 5.2, *в*).

Деление – для двух отношений бинарного и унарного, возвращает отношение, содержащее все значения одного атрибута бинарного отношения, которые соответствуют всем значениям в унарном отношении (рис. 5.3).



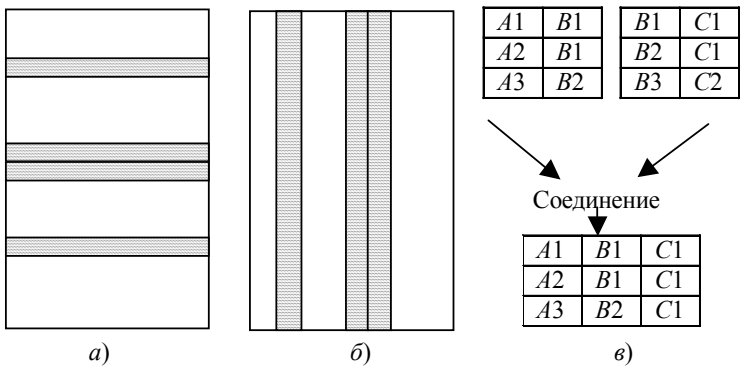


Рис. 5.2 Специальное реляционное отношение:
а – выборка; *б* – проекция; *в* – соединение

нные отношения:
оединение

Рис. 5.3 Деление

Результат каждой операции над отношением также является отношением. Это реляционное свойство называется свойством *замкну- тости*.

Результат одной операции может использоваться в качестве исходных данных для другой. Следовательно, существует возможность, например, взять проекцию от объединения, или соединение от двух выборок и т.д. Такие выражения считаются вложенными.

Каждое отношение включает заголовок, тело, множество потенциальных ключей. При выполнении реляционных операций необходимо пред правил наследования имен атрибутов и потенциальных ключей.

5.2 Стандартные реляционные операции

Рассмотрим выполнение операций над отношениями подробнее.

Для операций объединения (*union*), пересечения (*intersect*) и вычитания (*minus*) должны выполняться два свойства:

- операнды должны иметь одну и ту же степень;
- соответствующие атрибуты должны быть определены на одном и том же домене.

Операция умножения не требует выполнения этих условий.

Традиционные операции

Объединением двух совместимых по типу отношений *A* и *B* (*A union B*) называется отношение *C* с тем же заголовком и телом, состоящим из множества кортежей *t*, принадлежащих *A* или *B* или обоим отношениям.

$$C = (A \text{ union } B) \mid t_i \in C \ \forall \ t_j \in A \ \& \ t_i \in C \ \forall \ t_j \in B.$$

Пример: пусть отношения *A* и *B* будут такими, как они отражены ниже: *A* – детали изготовленные из стали; *B* – детали весом больше 0,5 кг.

Тогда *A union B* представляет детали, которые *или* изготовлены из стали, *или* имеют вес больше 0,5 кг.

A

K	Название детали	**Вес**	**Материал**
*K*1	*D*1	0.8	**Сталь**
*K*2	*D*2	1.0	Сталь
*K*3	*D*3	0.5	Сталь

B

K	Название детали	Вес	**Материал**
*K*1	*D*1	0.8	Сталь
*K*2	*D*2	1.0	Сталь
*K*4	*D*4	0.7	Алюминий

В результате получим 4 кортежа, а не 6 – повторяющиеся значения удаляются.

C

K	Название детали	Вес	Материал
*K*1	*D*1	0.8	Сталь
*K*2	*D*2	1.0	Сталь
*K*3	*D*2	0.5	Сталь
*K*4	*D*4	0.7	Алюминий

Пересечением двух совместимых по типу отношений *A* и *B* (*A intersect B*) называется отношение с тем же заголовком и телом, состоящим из множества кортежей *t*, принадлежащих одновременно обоим отношениям *A* и *B*.

$$C = (A \text{ intersect } B) \mid \forall \ t_i \in C \mid t_i \in A \ \& \ t_i \in B.$$

Пример: A intersect *B* для нашего примера представляет детали, изготовленные из стали и весом более 0,5 кг.

C

K	Название детали	Вес	Материал
*K*1	*D*1	0.8	Сталь

*K*2	*D*2	1.0	Сталь

Вычитанием двух совместимых по типу отношений *A* и *B* (*A minus B*) называется отношение с тем же заголовком и телом, состоящим из множества кортежей *t*, принадлежащих отношению *A* и не принадлежащих отношению *B*.

$$C = (A \ minus \ B) \mid \forall \ t_i \in C \mid t_i \in A \ \& \ t_i \notin B.$$

Пример: выражение (*A* minus *B*) представляет детали, которые изготовлены из стали и не весят больше 0,5 кг.

C = (*A minus B*)

K	Название детали	**Вес**	**Материал**
*K*3	*D*3	0.8	Сталь

C = (*B minus A*)

K	Название детали	**Вес**	**Материал**
*K*4	*D*4	0.7	Алюминий

Выражение (*B minus A*) представляет детали, которые не изготовлены из стали и весят больше 0,5 кг.

Следует заметить, что операция вычитания учитывает порядок основания операндов

C = (*A minus B*) не одно и то же, что *C* = (*B minus A*).

Произведение. Декартово произведение двух отношений есть множество упорядоченных пар кортежей, сохраняющих свойство замкнутости.

Декартово произведение двух отношений *A* и *B* (*A times B*), где *A* и *B* не имеют общих имен атрибутов, определяется как отношение с заголовком, который представляет собой сцепление двух заголовков исходных отношений *A* и *B* и телом, состоящим из множества всех кортежей *t*, таких, что *t* представляет собой сцепление кортежа *a*, принадлежащего отношению *A*, и кортежа *b*, принадлежащего отношению *B*.

$$C = (A \ times \ B) \mid \forall \ a_i \in A \ \& \ \forall \ b_i \in B \ \& \ a_i \neq b_i \ \exists \ t \mid t = a \cup b.$$

Кардинальное число результата равняется произведению кардинальных чисел исходных отношений *A* и *B*, а степень – сумме из степеней.

Пример. Пусть отношения *A* и *B* будут такими, как показано в таблице, т.е. отношения *A* представляет детали изделия , а отношение *B* – предприятий-изготовителей.

B

Изготовитель	Город	Телефон	Адрес
*P*1	…	…	…
*P*2	…	…	…
*P*3	…	…	…

Сокращенно представим таблицу *A*, как столбец *K* = *K*1, *K*2, *K*3, *K*4, *K*5, а таблицу *B* как *P* = *P*1, *P*2, *P*3.

Тогда *A times B* – это все пары деталь-изготовитель, изготовитель-деталь. Таблица *C* будет иметь пары *C* = {*K*1, *P*1; *K*1, *P*2; *K*1, *P*3; *K*2, *P*1; *K*2, *P*2; *K*2, *P*3; *K*3, *P*1; *K*3, *P*2; *K*3, *P*3; *K*4, *P*1; *K*4, *P*2; *K*4, *P*3; *K*5, *P*1; *K*5, *P*2; *K*5, *P*3}.

Результат не говорит нам ничего нового, он просто подтверждает, что существуют номера деталей и предприятий-изготовителей.

Операция декартова произведения не очень важна на практике, за исключением операций Θ-соединений.

5.3 Свойства стандартных операций

Операции объединения, пересечения и декартова произведения обладают свойствами:

- ассоциативность:

(*A union B*) *union C* ~ *A union* (*B union C*) ⇒ *A union B union C*;

- коммуникативность:

A union B ~ *B union A*;

A intersect B ~ *B intersect A*;

A times B ~ *B times A* (но не в теории множеств, 1 и 2 свойства не выполняются).

Указанные свойства не выполняются для операции вычитания.

5.4 Специальные реляционные операции

Выборка (ограничение). Это сокращенное название так называемой Θ-выборки, где Θ обозначают любой скалярный оператор сравнения (=, ≠, ≥, и т.д.). Θ-выборкой из отношения *A* по атрибутам *X* и *Y*: *A where X*Θ*Y* называется отношение, имеющее тот же заголовок, что и отношение *A* и тело, содержащее множество всех кортежей *t* отношения *A*, для которых условие *X*Θ*Y* имеет значение "истина". Атрибуты *X* и *Y* должны быть определены на одном и том же домене, а оператор должен иметь смысл для этого домена.

Пример: получить список деталей весом от 1 кг и выше для отношения *A*.

A where Вес >=1.0

A

K	Назва-ние детали	Вес	Материал
*K*2	*D*2	1.0	Сталь

Операция *сравнения* может проводиться для символьных и строковых переменных (=, ≠). В качестве действия над атрибутами используют и логические операции AND, OR, NOT.

Пусть есть отношение *P* (таблица поставщиков деталей).

P

N_пост	Назва-ние завода	Город	Улица	Номер дома	Теле-фон
*P*1	Протон	Москва	–	–	–
*P*2	–	Санкт-Петербург	–	–	–
*P*3	–	Зеленоград	–	–	–
*P*4	–	Москва	–	–	–
*P*5	–	Тамбов	–	–	–

Тогда очередную выборку можно провести для поиска, например, всех поставщиков из города Москва:

P where Город = "Москва" или

P where Город = "Тамбов" and улица = "Ленинградская".

На основании свойства замкнутости можно расширить условие в выражении *where* до произвольного числа логических сочетаний или простых сравнений, применяя следующие тождества:

1 *A where X and Y* ≡ (*A where X*) *intersect* (*A where Y*)

2 *A where X or Y* ≡ (*A where X*) *union* (*A where Y*)

3 *A where not X* ≡ *A minus* (*A where X*)

Проекция. Проекцией отношения *A* по атрибутам *X*, *Y*, …, *Z*, где каждый из атрибутов принадлежит отношению *A* (*A*[*X*, *Y*, …, *Z*]), называется отношение с заголовками {*X, Y, ..., Z*} и телом, содержащим множество всех кортежей {*X* : *x*, *Y* : *y*,…, *Z* : *z*} таких, что в отношении *A* значение атрибута *X* равно *x*, атрибута *Y* = *y*, атрибута *Z* = *z*. Результат операции проекции – подмножество указанных в списке атрибутов из множества имеющихся атрибутов с последующим исключением дублирующих кортежей.

Операция проекции допускает тождественную *R* и нулевую *R*[ ] проекцию. В первом случае результат – то же отношение, во втором нет ни одного кортежа.

Пример: найти проекцию отношения *P* по атрибуту "Город":

Город
Москва
Санкт-Петербург
Зеленоград
Тамбов

или (*P where* Город = "Москва")

*N*_пост
*P*1
*P*4

Соединение. Операция соединения имеет несколько вариантов: это наиболее важное *естественное* соединение и Θ-соединение. Для обозначения естественного соединения применим термин *join*. Пусть отношения *A* и *B* имеют заголовки {X_1, X_2, …, X_m, Y_1, Y_2,…, Y_n} и {Y_1, Y_2, …, Y_n, Z_1, Z_2,…, Z_p} соответственно. Предположим, что соответствующие атрибуты (с одинаковыми именами) определены на одном и том же домене. Рассмотрим выражения {X_1, X_2, …, X_m}, {Y_1, Y_2, …, Y_n} и {Z_1, Z_2, …, Z_p} как три составных атрибута *X*, *Y*, *Z*. Тогда естественным соединением отношений *A* и *B* (*A join B*) называется отношение с заголовком {*X*, *Y*, *Z*} и телом, содержащим множество всех кортежей {*X* : *x*, *Y* : *y*, *Z* : *z*}, таких, что в отношении *A* значение атрибута *X* равно *x*, а атрибута *Y* равно *y*, и в отношении *B* значение атрибута *Y* равно *y*, а атрибута *Z* равно *z*.

Пример. Пусть имеем таблицу деталей *C* и таблицу поставщиков *P*.

C

K	Название детали	**Вес**	Материал	**Город**
*K*1	D1	0.8	Сталь	Москва
*K*2	D2	1.0	Сталь	Москва
*K*3	D3	0.5	Сталь	Пенза
*K*4	D3	0.7	Алюминий	Липецк

P

*N*_пост	Название завода	Город	**Улица**	Дом	**Телефон**
*P*1	…	Москва	…	…	…

*P*2	…	Санкт-Петербург	…	…	…
*P*3	…	Зеленоград	…	…	…
*P*4	…	Москва	…	…	…
*P*5	…	Тамбов	…	…	…

C JOIN P

K	Название детали	Вес	Материал	Город	*N*_пост	Название завода	Улица	Дом	Телефон
*K*1	*D*1	0.8	Сталь	Москва	*P*1	…	…	…	…
*K*1	*D*1	0.8	Сталь	Москва	*P*4	…	…	…	…
*K*2	*D*2	1.0	Сталь	Москва	*P*1	…	…	…	…
*K*2	*D*2	1.0	Сталь	Москва	*P*4	…	…	…	…

Естественное соединение обладает свойствами ассоциативности и коммутативности:

(*A join B*) *join C* ~ *A join* (*B join C*) ~ *A join B join C* и *A join B* ~ *B join A*.

Если *A* и *B* не имеют общих имен атрибутов, то *A* join *B* ≡ *A* times *B* (соед. $\xrightarrow{в}$ произвед.).

Θ-*соединение*. Эта операция используется в более редких случаях, когда надо соединить два отношения на основе некоторых условий, отличных от эквивалентности. Θ-соединение эквивалентно двум операциям: нахождению декартова произведения от двух отношений и последующему выполнению указанной выборки из полученного результата. Тогда Θ-соединением отношения *A* по атрибуту *X* с отношением *B* по атрибуту *Y* называется результат вычисления выражения

(*A times B*) *where X*Θ*Y*.

Результат – отношение с тем же заголовком, что при декартовом произведении отношений *A* и *B* и с телом, содержащим множество кортежей *t* таких, что *t* принадлежит этому декартову произведению и выполнение условия *X*Θ*Y* дает значение истина для этого кортежа.

Деление. Пусть отношения *A* и *B* имеют заголовки $\{X_1, X_2, \ldots, X_m, Y_1, Y_2, \ldots, Y_n\}$ и $\{ Y_1, Y_2, \ldots, Y_n\}$ соответственно. Пусть соответствующие атрибуты определены на одном и том же домене. Пусть *X* и *Y* – два составных атрибута, где $X = \{X_1, X_2, \ldots, X_m\}$, $Y = \{Y_1, Y_2, \ldots, Y_n\}$. Тогда делением отношений *A* и *B* (*A divide by B*) называется отношение с заголовком *X* и телом, содержащим множество всех кортежей $\{X : x\}$ таких, что существует кортеж $\{X : x, Y : y\}$, который принадлежит отношению *A* для всех кортежей $\{Y : y\}$, принадлежащих отношению *B*.

Нестрого это можно сформулировать так: результат содержит такие *X*-значения из отношения *A*, для которых соответствующие *Y*-значения из *A* включают все *Y*-значения из отношения *B*.

Пользоваться этой операцией следует осторожно, так как не исключено возникновение пустых отношений.

Пример. Пусть есть отношение исходное *AP* и делители B_i для $i = 1, 2, 3$.

AP

K	*P*	*K*	*P*
*K*1	*P*1	K2	*P*1
*K*1	*P*2	K2	*P*2
*K*1	*P*3	K3	*P*2
*K*1	*P*4	K4	*P*3
*K*1	*P*5	K4	*P*4
*K*1	*P*6	K4	*P*5

B_1

P
*P*1

B_2

P
*P*2
*P*4

B_3

P
*P*1
*P*2
*P*3
*P*4
*P*5
*P*6

*AP divide by B*1

K
*K*1
*K*2

*AP divide by B*2

K
*K*1
*K*4

Получить поставщиков, поставляющих "все" детали *AP divide by B*3

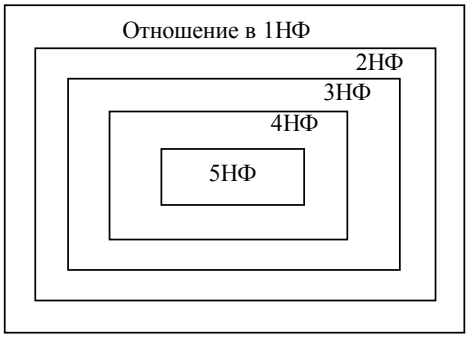
K
*K*1

6 ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

6.1 Нормальные формы

Отношение находится в некоторой нормальной форме, если удовлетворяет заданному условию. Отношение находится в первой нормальной форме (1НФ) тогда и только тогда, когда оно содержит только скалярные значения.

Коддом были определены три нормальных формы (1НФ, 2НФ, 3НФ). Все нормализованные отношения входят в 1НФ, в 2НФ, в 3НФ и т.д. Вторая НФ более желательна, чем первая, третья, чем вторая и т.д. При проектировании баз данных следует использовать отношения не только в 1НФ, но и в 2НФ, 3НФ. При помощи определенной последовательности действий отношения преобразуются из одной НФ в другую.



1 - Нормальные формы

Рис. 6.1 Нормальные формы

Бойсом и Коддом переработана 3НФ и в более строгом смысле называется нормальной формой Бойса-Кодда (НФБК). В ней устранены некоторые неадекватности, возможные в 3НФ.

Фейгином одновременно с введением НФБК определена 4НФ, они практически одинаковы. Далее Фейгин предложил 5НФ или проективно-соединительную НФ.

6.2 Декомпозиция без потерь функциональной зависимости

Процедура нормализации включает декомпозицию данного отношения на другие отношения. Декомпозиция должна быть обратимой.

Пример декомпозиции отношения типа {код, имя_детали, город}.

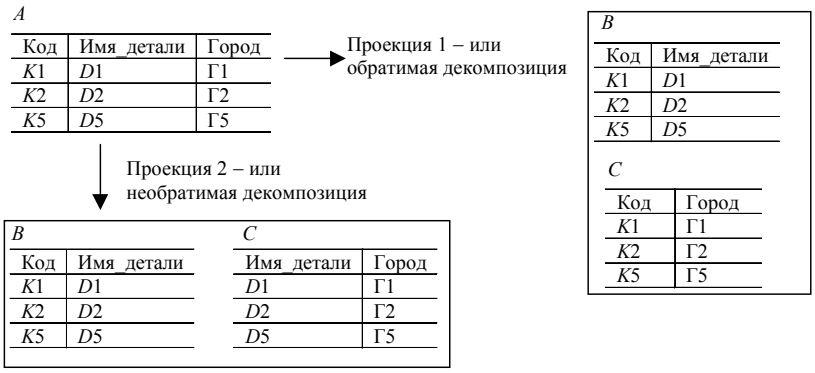


Рис. 6.2 Пример декомпозиции

Декомпозиция проводилась с использованием *теоремы Хеза*:

Пусть $R\{A, B, C\}$ есть отношение, где A, B, C – атрибуты этого отношения. Если R удовлетворяет зависимости $A \rightarrow B$, то R равно соединению его проекций $\{A, B\}$ и $\{A, C\}$.

Стрелкой ($\rightarrow$) показана некоторая функциональная зависимость. Важную роль в понятии функциональных зависимостей (ФЗ) играет *неприводимая слева ФЗ*, т.е. левая часть зависимости не должна быть избыточной. Например, ФЗ {код_детали, код_города, город} может быть записана без атрибута код_города, {код_детали} $\rightarrow$ город. Последняя ФЗ является неприводимой слева.

ФЗ могут изображаться в виде *диаграмм* (*схем*).

Диаграмма функциональной зависимости строится для отношения и некоторого неприводимого множества зависимостей для этого отношения. Например, отношение "Деталь" имеет следующую схему.

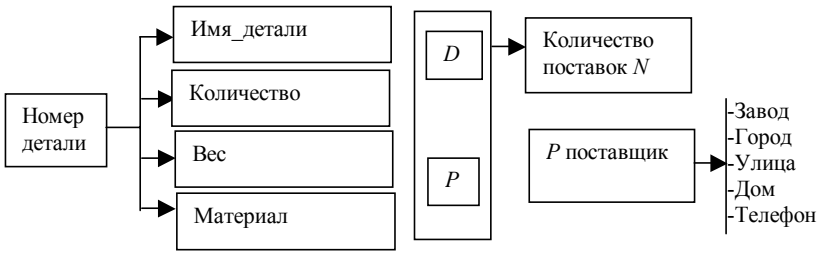


Рис. 6.3 Схема функциональной зависимости

ФЗ – это особый вид ограничений целостности и, несомненно, *семантическое* понятие. Распознавание ФЗ является частью процесса выяснения *смысла* тех или иных данных. Например, Ном_дет $\rightarrow$ {название детали, количество, вес, материал} означает, что каждая деталь имеет свой код, который точно определяет ее название, количество, вес и материал.

6.3 Нормализация отношений

Одна из целей проектирования баз данных состоит в получении НФБК и форм более высокого порядка. Формы 1НФ, 2НФ и 3НФ представляют собой промежуточные результаты.

На примере вышеприведенной схемы введено добавочное отношение количества поставок N детали D_i поставщиком P_i; первичный ключ – комбинация $\{D, P\}$.

Рассмотрим формы отношений.

• Отношение находится в первой нормальной форме (1НФ) тогда и только тогда, когда все используемые домены содержат только скалярные значения.

Пусть имеем объединенное отношение деталей, поставщиков и количества поставок.

DP (*D*, имя_детали, количество, вес, материал, *P*, количество_поставок, завод, город, улица, дом, телефон).

Дополнительное ограничение имя_детали $\xrightarrow{\text{ФЗ}}$ количество. Первичный ключ для *DP* это (*D*, *P*). Диаграмма ФЗ имеет вид.

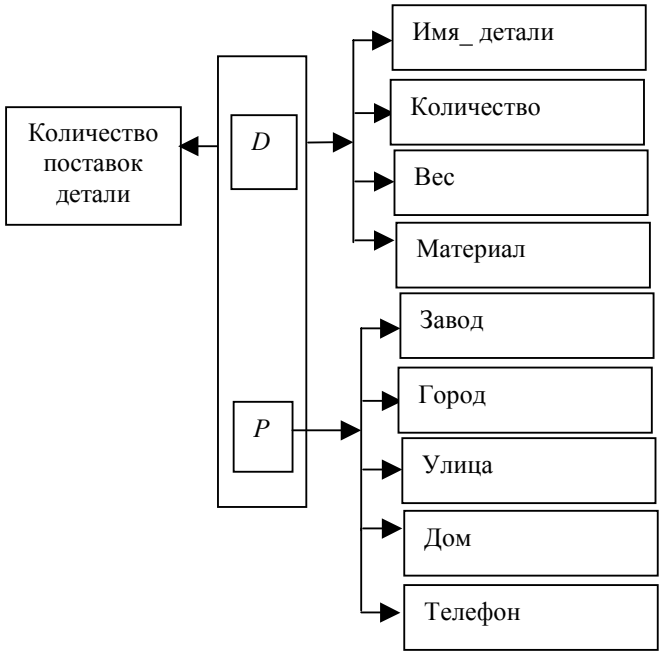


Рис. 6.4 Диаграмма функциональной зависимости в 1 альной зависимости в 1НФ

Отношение избыточно. Например, в нем не может быть детали или поставщика, который был ранее известен, а в последнее время не сделал ни одной поставки и т.д., или количество деталей 0.

• Отношение находится во второй нормальной форме (2НФ) тогда и только тогда, когда оно находится в 1НФ и каждый неключевой атрибут неприводимо зависит от первичного ключа.

В примере имеем диаграмму ФЗ ранее известную:

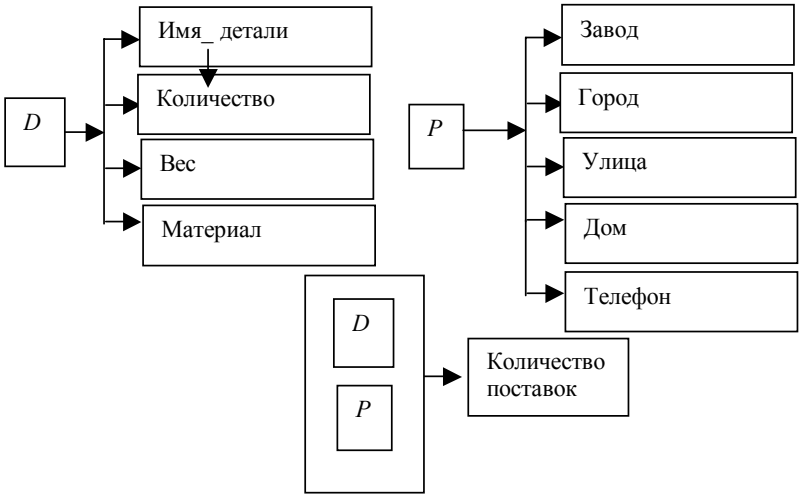


Рис. 6.5 Диаграмма функциональной зависимости в

Первичные ключи *D*, *P*, {*D*, *P*} и три отношения.

• Отношение находится в третьей нормальной форме (3НФ) тогда и только тогда, когда оно находится во второй нормальной форме и каждый неключевой атрибут нетранзитивно (отсутствие какой-либо зависимости) зависит от первичного ключа.

ФЗ для отношения "Детали":

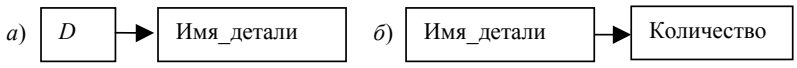


Рис. 6.6 Диаграмма функциональной зависимости в

Отношения *А* и *Б* в 3НФ, первичные ключи *D* и Имя_детали.

Следовательно, этапы нормализации следующие:

1. Создание проекций для исключения "приводимых" ФЗ.
2. Создание проекций для исключения транзитивных ФЗ.

Отношение находится в нормальной форме Бойса-Кодда тогда и только тогда, когда каждая нетривиальная и неприводимая слева ФЗ обладает потенциальным ключом в качестве детерминанта (левая часть ФЗ).

На диаграмме стрелки ФЗ начинаются только с потенциальных ключей.

Если убрать связь имя_детали → количество и ввести дополнительный независимый атрибут, например *DD*, в качестве потенциального ключа, то схема, показанная на диаграмме, будет находиться в НФБК.

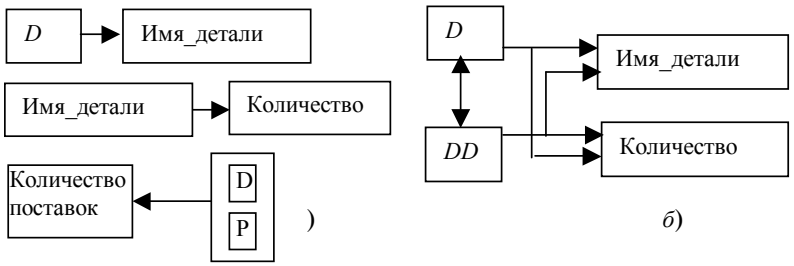


Рис. 6.7 Диаграмма функциональной зависимости в НФБК

Для веса и материала можно также ввести потенциальные ключи, ФЗ усложнится.

6.4 Этапы проектирования баз данных

Число этапов проектирования БД напрямую зависит от количества уровней представления данных, или моделей данных. Известно четыре основных модели данных: даталогическая (ДЛМ), физическая (ФМ), внешняя (ВМ) и инфологическая (ИЛМ). Следовательно, можно говорить о четырех этапах проектирования БД (рис. 6.8).

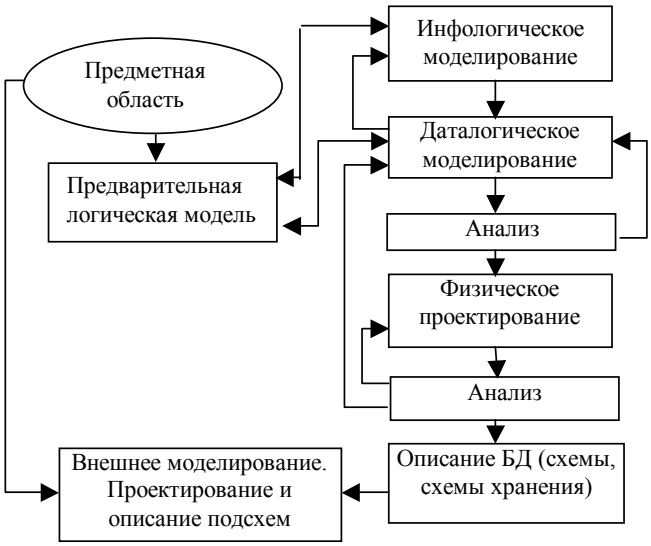


Рис. 6.8 Взаимосвязь этапов проектирования этапов проектирования

1 Даталогическое проектирование основано на модели логического уровня и представляет собой описание и построение схем связей между элементами данных безотносительно к их содержанию и среде хранения.

2 Физическое проектирование состоит в описании и построении схем хранения данных для определенной среды хранения. На этом этапе осуществляется выбор типа носителя, способ организации данных, методов доступа, определение параметров физического блока, управление работой памяти, считывание данных и т.д.

3 Внешнее моделирование состоит в описании и построении схем или логических структур с точки зрения конкретного пользователя. На этом этапе формализуются допустимые режимы обработки данных в рамках данной схемы или подсхемы. Для реляционных моделей это описание процедуры View конкретного приложения.

4 Инфологическое проектирование состоит в описании и построении схем отражений предметной области, выполненном без ориентации на используемые в дальнейшем программные и технические средства.

6.5 Инфологическое моделирование и проектирование

Инфологическая модель выполняется с использованием специальных искусственно формализованных языковых средств. Основное требование к ИЛМ – это адекватное отражение предметной области. Дополнительные требования связаны с обеспечением возможности композиции и декомпозиции модели.

ИЛМ включает ряд компонентов (рис. 6.9). Центральной компонентой ИЛМ является ER-модель, описывающая объекты предметной области и связи между ними.

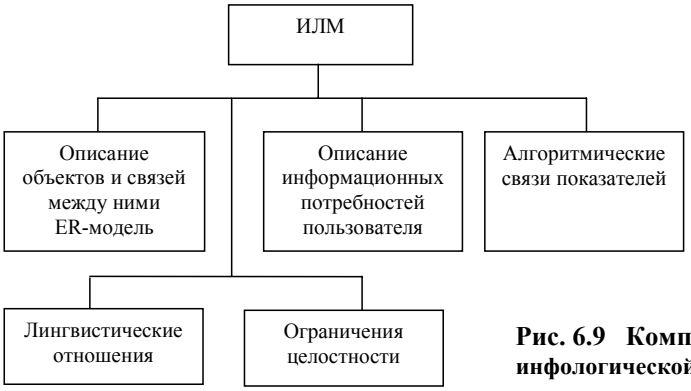


Рис. 6.9 Компоненты инфологической модели

Для описания ER-модели (объект – свойство – отношение) используют как языковые, так и графические средства (последние наиболее часто).

Объекты, имеющие одинаковый набор свойств, группируются в классы объектов со своими идентификаторами.

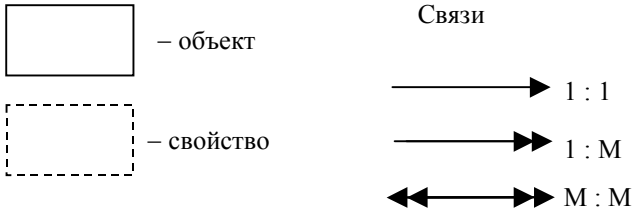


Рис. 6.10 Элементы ER-диаграмм

Свойства, не изменяющиеся во времени – статические (*S*), изменяющиеся – динамические (*D*).

Класс принадлежности показывает, может ли отсутствовать связь объекта одного класса с объектом другого класса или она обязательна. В последнем случае в обозначение объекта добавляется разделитель с точкой.

В приведенных ниже примерах показаны диаграммы ER-экземпляров и ER-типов для базы данных, имеющей два объекта "Изделие" и "Деталь".

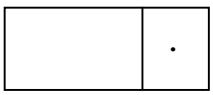


Рис. 6.11 Обозначение класса принадлежности

1 Изделие имеет в своем составе детали, но ни одно изделие не имеет более одной детали.

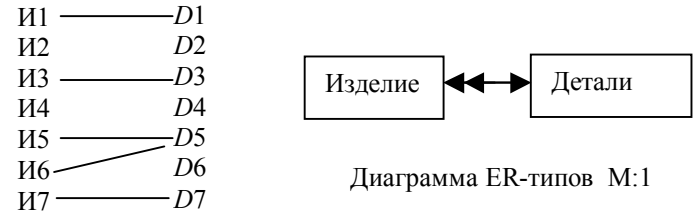


Диаграмма ER-типов M:1

Диаграмма ER-э

Рис. 6.12

2 Изделие имеет в своем составе детали. Каждое изделие должно иметь хотя бы одну деталь, но не более чем одну.

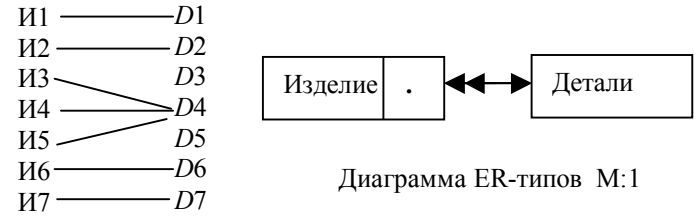


Диаграмма ER-типов M:1

Диаграмма ER-экземпляров

Рис. 6.13

3 Изделие имеет в своем составе детали. Но некоторые изделия состоят из нескольких деталей.

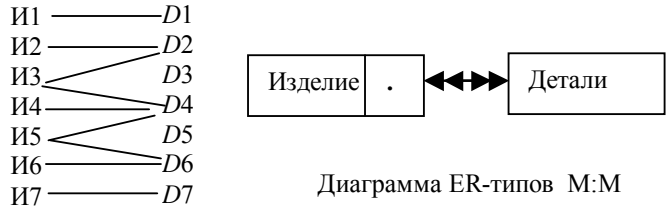


Диаграмма ER-типов M:M

Диаграмма ER-экземпляров

Рис. 6.14

4 Изделие имеет в своем составе детали. Каждое изделие обязательно состоит из нескольких деталей. Каждая деталь обязательно применяется в изделии.

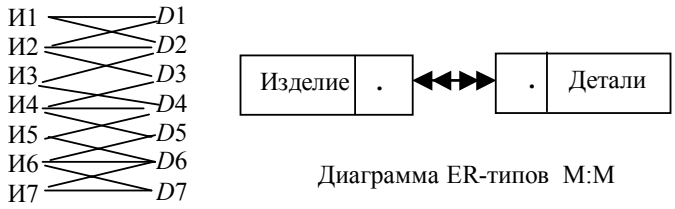


Диаграмма ER-типов M:M

Диаграмма ER-экземпляров

Рис. 6.15

Объекты могут быть простыми и сложными. Простой это неделимый на составляющие объект. Сложные – это составные, обобщенные и агрегированные объекты.

Составные соответствуют отображению отношения "целое-часть". Например, изделие-детали или группа-студенты и т.д. Специальных условных обозначений на схемах нет.

Обобщенный объект отражает наличие связи "род-вид". Например, объекты "студент", "аспирант", "школьник" образуют обобщенный объект "учащиеся" с наследованием некоторых свойств. В инфологической модели подклассы выделяются в явном или неявном виде и обозначаются треугольником.

Пример: обобщенный объект "личность" имеет несколько категорий "сотрудник", "студент", "аспирант". Объект "личность" разбит на два подкласса "сотрудник" и "учащийся". Объект "сотрудник" может быть классифицирован далее, например "преподаватели" и "администрация".

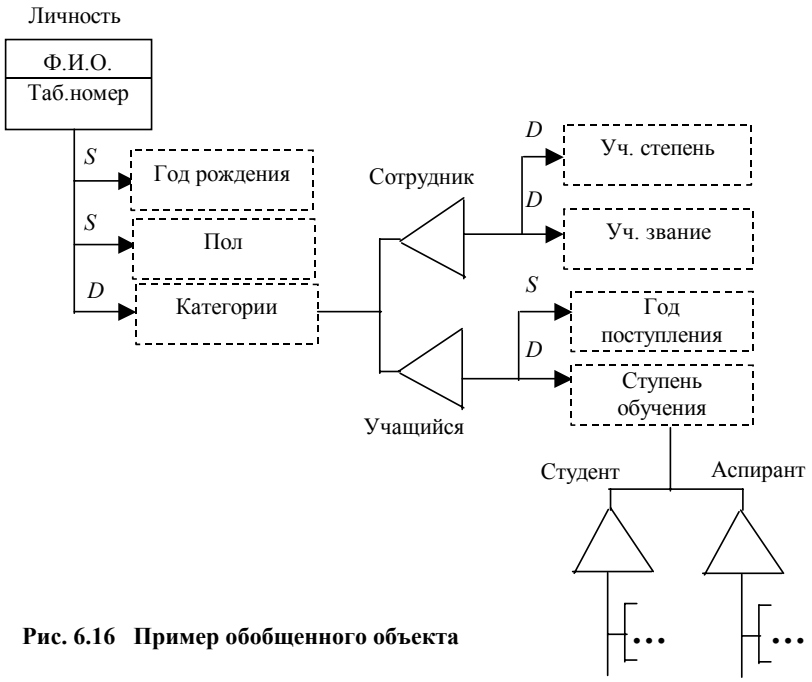


Рис. 6.16 Пример обобщенного объекта

Агрегированные объекты соответствуют обычно какому-либо процессу, в который вовлечены другие объекты. Например, рассмотрим процесс поставки деталей заводом-изготовителем заказчику.

Агрегированный объект в инфологической модели обозначается ромбом. Объект и свойства обозначаются как и прежде, или могут иметь и другие схематические изображения в различных стандартах.

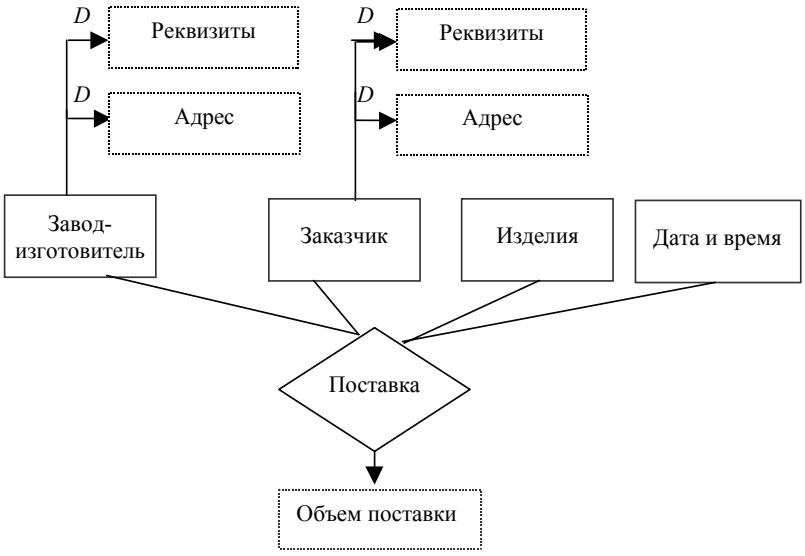
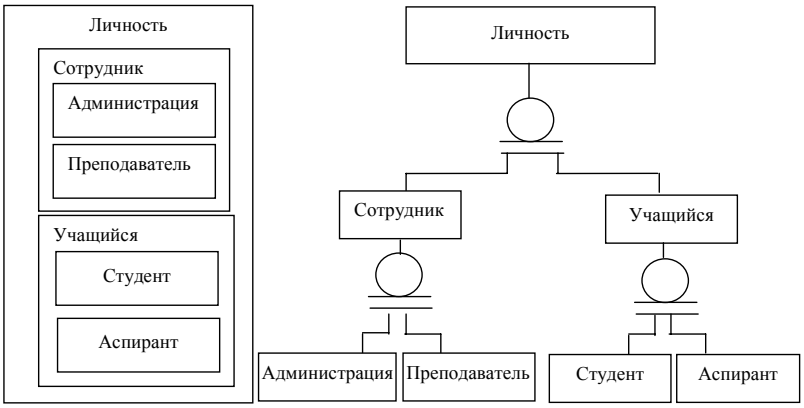


Рис. 6.17 Пример агрегированного объекта ированного объекта



а) *б*)

Рис. 6.17 Элементы ER-диаграммы в различных стандартах:
а – CASE ORACLE; *б* – IDEF

В последнее время широко используются способы изображения, нотации, Чена, Мартина и IDEF1X.

6.6 Даталогическое проектирование

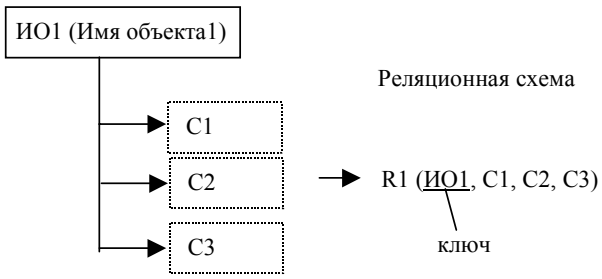
Конечным результатом даталогического проектирования является описание логической структуры БД на языке описания. В логической структуре определяются все информационные единицы и связи между ними, типы данных и количественные характеристики. Однако не все виды связей могут отображаться в ДЛМ, например те, которые не поддерживает конкретная СУБД. На этапе разработки ДЛМ определяется состав БД, например, хранить только исходные данные, а все производные могут быть получены расчетным путем в результате запроса. При отображении объекта в файл исключаются одинаковые идентификаторы различных объектов, даются новые имена, определяются количество и структура файлов.

Важную роль на этом этапе играет внутризаписная структура данных (векторы, группы и пр.) и межзаписная структура (реляционная, иерархическая, сетевая).

Для перехода от ИЛМ к реляционной ДЛМ надо выполнить следующие операции по замене *ER*-типа на описание атрибутов отношений.

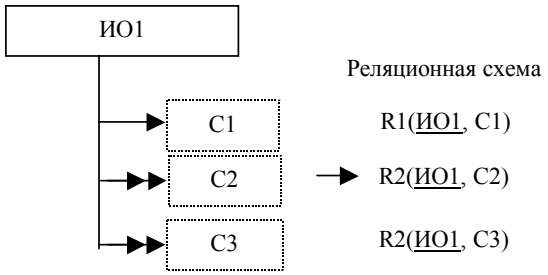
1 Простой объект с единичными свойствами.

ИЛМ

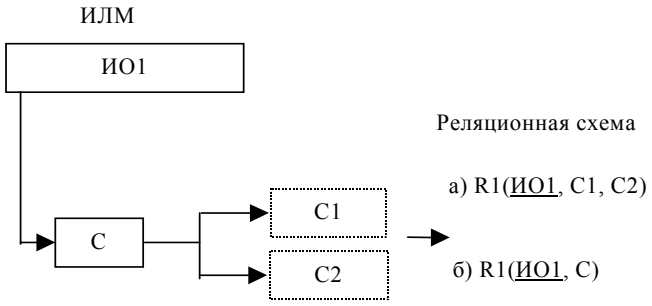


2 Множественные свойства объекта. Им в соответствие ставится отдельное отношение.

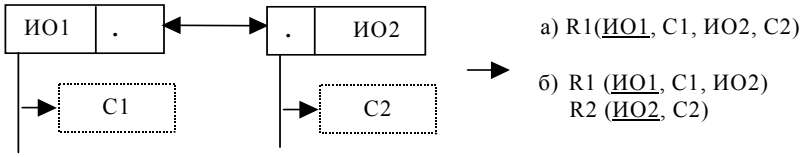
ИЛМ



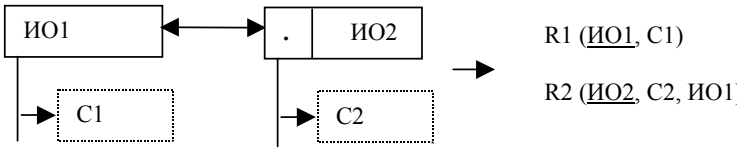
3 Объект с составным свойством. Если многие объекты обладают свойством, то их можно считать единичными (*а*). В противном случае отдельное отношение с обобщенным свойством (*б*).



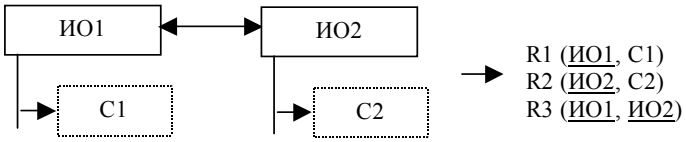
4 Связь 1:1 и обязательный класс принадлежностей сущностей.



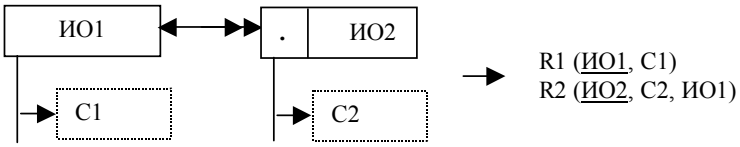
5 Связь 1:1 и обязательный класс принадлежностей одной из сущностей.



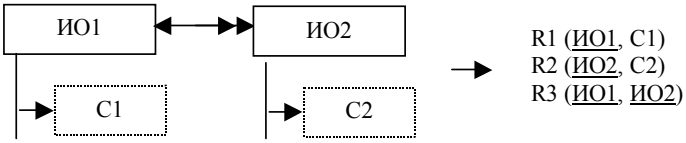
6 Связь 1:1 и необязательный класс принадлежности сущностей.



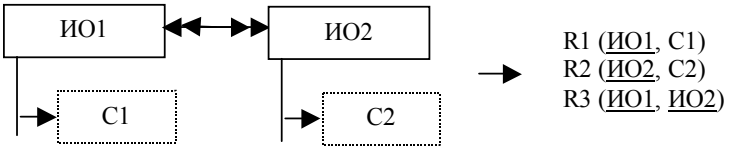
7 Связь 1 : М и обязательный класс принадлежностей *n*-связной сущности.



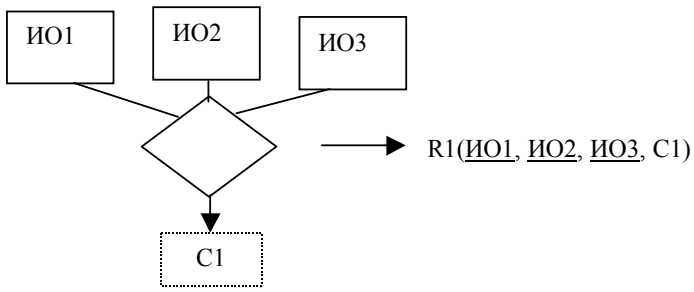
8 Связь 1 : М и необязательный класс принадлежности *n*-связной сущности.



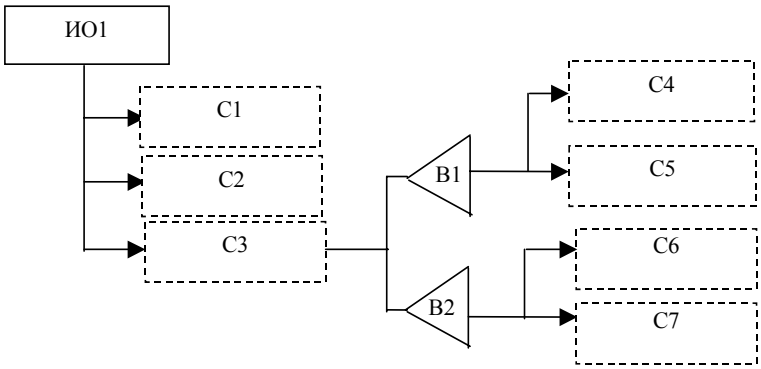
9 Связь М : М и необязательный класс принадлежности *n*-связной сущности.



10 Агрегированный объект.



11 Обобщенный объект.



R1 (ИО1, C1, C2, C3, C4, C5, C6, C7) или R1 (ИО1, C1, C2, C4, C5)
R2 (ИО1, C1, C2, C6, C7)

Полученные реляционные отношения будут находиться в 4 нормальной форме. Преобразования в пунктах 5, 7 и в 6, 8, 9 имеют одинаковые реляционные схемы.

На рисунках показан только вид заголовков таблиц. Полная даталогическая модель БД будет включать набор таких таблиц с указанием типов данных, длинны переменной, принадлежность к первичному или внешнему ключу и т.д., и соответственно будет иметь более сложный вид. Одновременно надо указать связи между ключевыми атрибутами.

Часть 2
СТРУКТУРИРОВАННЫЙ ЯЗЫК ЗАПРОСОВ SQL

1 ВВЕДЕНИЕ В SQL

SQL символизирует собой *структурированный язык запросов.* Это язык, который дает вам возможность создавать и работать в реляционных базах данных, которые являются наборами связанной информации, сохраняемой в таблицах. Прежде, чем вы сможете использовать SQL, вы должны понять, что такое реляционные базы данных.

Реляционная база данных – это тело связанной информации, сохраняемой в двумерных таблицах. Напоминает адресную или телефонную книгу. В книге имеется большое количество входов, каждый из которых соответствует определенной особенности. Для каждой такой особенности может быть несколько независимых фрагментов данных, например имя, телефонный номер и адрес. Предположим, что вы должны сформатировать эту адресную книгу в виде таблицы со строками и столбцами. Каждая строка (называемая также записью) будет соответствовать определенной особенности; каждый столбец будет содержать значение для каждого типа данных – имени, телефонного номера и адреса, представляемого в каждой строке. Адресная книга могла бы выглядеть следующим образом:

Имя	Телефон	Адрес
Иванов Иван	(237) 333-10	Кирсанов, ул. Советская 5, 15
Петров Петр	(22) 476-438	Тамбов, ул. Мичуринская 10, 3
Сидоров Олег	(203) 233-70	Котовск, ул. Интернациональная 23

То что вы получили, является основой реляционной базы данных, как и было определено в начале этого обсуждения – а именно, двумерной (строка и столбец) таблицей информации. Однако реляционные базы данных редко состоят из одной таблицы. Такая таблица меньше, чем файловая система. Создав несколько таблиц взаимосвязанной информации, вы сможете выполнить более сложные и мощные операции с вашими данными. Мощность базы данных зависит от связи, которую вы можете создать между фрагментами информации, а не от самого фрагмента информации.

1.1 Интерактивный и вложенный SQL

Имеются два SQL: *интерактивный* и *вложенный*.

В основном, обе формы работают одинаково, но используются различно.

Интерактивный SQL используется для функционирования непосредственно в базе данных, чтобы производить вывод для использования его заказчиком. В этой форме SQL, когда вы введете команду, она сейчас же выполнится, и вы сможете увидеть вывод (если он вообще получится) – немедленно.

Вложенный SQL состоит из команд SQL, помещенных внутри программ, которые обычно написаны на некотором другом языке (типа Си или Паскаля). Это делает программы более мощными и эффективным. Однако, допуская эти языки, приходится иметь дело со структурой SQL и стилем управления данных, который требует некоторых расширений к интерактивному SQL. Передача SQL команд во вложенный SQL является выдаваемой ("*passed off*") для переменных или параметров используемых программой, в которую они были вложены.

2 СОЗДАНИЕ БАЗЫ ДАННЫХ

2.1 Оператор *create database*

Для создания БД используется оператор *SQL*, имеющий следующий формат:

Create {*database* | *schema*} "< имя файла>"
[*user* "имя пользователя" [*password* "пароль"]]
[*page_size* [=] целое]
[*length* [=] целое [*page* [*s*]]]
[*Default character set* набор_символов]
[<вторичный файл>];

[ ] – необязательный элемент; { } – возможные элементы.

<вторичный файл> = *file* "имя файла" [<файлов_информ>]
[вторичный файл]
<файлов_информ> = *length* [=] целое [*page* [*s*]] | *starting* [*at* [*page*]]
целое [файлов_информ]

где "<имя файла>" – спецификация файла, в котором будет храниться БД;

user "имя пользователя" – проверяется при соединении пользователя с сервером;

password "пароль" – проверяется совместно с именем пользователя;

page_size [=] целое – размер страницы БД пользователя в байтах 1024 (по умолчанию), 2048, 4096 или 8192;

default character set – определяет набор символов применяемых в БД, по умолчанию *None*;

file "<имя файла>" – имя одного или нескольких файлов, в которых будет располагаться БД;

starting [*at* [*page*]] – если БД занимает несколько файлов, то это предложение позволяет определить с какой страницы располагается БД в указанным файле;

length [=] целое [*page* [*s*]] – длина файла в страницах. По умолчанию 75 страниц, минимум 50, а максимум ограничен дисковым пространством.

В многофайловой БД самый первый файл называется *первичным*, остальные – *вторичными.*

Например:

Create database "*D*:*BD**base.gdb*"
file "*D*:*BD**base.gd*1" *starting at page* 1001
length 500
file "*D*:*BD**base.gd*2".

Здесь определяется БД *d*:*bd**base.gdb* состоящая из 3-х файлов: первичного *base.gdb* длиной 1000 страниц, *base.gd*1 длиной 500 страниц и *base.gd*2 неопределенной длины.

Если для вторичного файла не указана длина, следует указать, с какой страницы он должен начинаться.

Размер страницы указывается в байтах, например:

Create database "*base.gdb*" *page_size* 4096.

Увеличение размера страницы может привести к ускорению работы с БД за счет уменьшения глубины индексов, уменьшения операций считывания длинных записей. Но не оправдано когда запросы

возвращают небольшое количество записей, так как считывается страница целиком и в ней много лишних записей.

2.2 Типы записей на примере СУБД Interbase

НЕ ВСЕ ТИПЫ ЗНАЧЕНИЙ, КОТОРЫЕ МОГУТ ЗАНИМАТЬ ПОЛЯ ТАБЛИЦЫ – ЛОГИЧЕСКИ ОДИНАКОВЫЕ. НАИБОЛЕЕ ОЧЕВИДНОЕ РАЗЛИЧИЕ – МЕЖДУ ЧИСЛАМИ И ТЕКСТОМ. ВЫ НЕ МОЖЕТЕ ПОМЕЩАТЬ ЧИСЛА В АЛФАВИТНОМ ПОРЯДКЕ ИЛИ ВЫЧИТАТЬ ОДНО ИМЯ ИЗ ДРУГОГО. ТАК КАК СИСТЕМЫ С РЕЛЯЦИОННОЙ БАЗОЙ ДАННЫХ БАЗИРУЮТСЯ НА СВЯЗЯХ МЕЖДУ ФРАГМЕНТАМИ ИНФОРМАЦИИ, РАЗЛИЧНЫЕ ТИПЫ ДАННЫХ ДОЛЖНЫ ПОНЯТНО ОТЛИЧАТЬСЯ ДРУГА ОТ ДРУГА, ТАК ЧТОБЫ СООТВЕТСТВУЮЩИЕ ПРОЦЕССЫ И СРАВНЕНИЯ МОГЛИ БЫТЬ В НИХ ВЫПОЛНЕНЫ.

В SQL, это делается с помощью назначения каждому полю – типа данных, который указывает на тип значения, которое это поле может содержать. Все значения в данном поле должны иметь одинаковый тип. Вы часто будете сравнивать некоторые или все значения в данном поле, поэтому вы можете выполнять действие только на определенных строках, а не на всех. Вы не могли бы сделать этого, если бы значения полей имели смешанный тип данных.

Smallint	2 байта	– 32768 ÷ + 32767

Два типа чисел INTEGER (ЦЕЛОЕ ЧИСЛО) и DECIMAL (ДЕСЯТИЧНОЕ ЧИСЛО) (которые можно сокращать как INT и DEC, соответственно), будут адекватны для наших целей, также как и для целей большинства практических деловых прикладных программ. Естественно, что тип ЦЕЛОЕ можно представить как ДЕСЯТИЧНОЕ ЧИСЛО, которое не содержит никаких цифр справа от десятичной точки.

Integer	4 байта	– 2147483648 ÷ + 2147483647

Тип для текста – CHAR (или СИМВОЛ), который относится к строке текста.

Поле типа CHAR имеет определенную длину, которая определяется максимальным числом символов, которые могут быть введены в это поле. Больше всего реализаций также имеют нестандартный тип называемый VARCHAR (ПЕРЕМЕННОЕ ЧИСЛО СИМВОЛОВ), который является текстовой строкой, которая может иметь любую длину до определенного реализацией максимума (обычно 254 символа). CHARACTER и VARCHAR значения включаются в одиночные кавычки как "текст". Различие между CHAR и VARCHAR в том, что CHAR должен резервировать достаточное количество памяти для максимальной длины строки, а VARCHAR распределяет память, так как это необходимо

char (*n*) или		
character (*n*)	0 ÷ 32768 байт	*n* – символов
varchar (*n*)	0 ÷ 32768 байт	до *n* – символов
float	4 байта	$3.4 \cdot 10^{-38} \div 3.4 \cdot 10^{+38}$
double precision	8 байт	$1.7 \cdot 10^{-308} \div 1.7 \cdot 10^{+308}$

Такие как, DATE (ДАТА) и TIME (ВРЕМЯ) – фактически почти стандартные типы (хотя точный формат их меняется). Некоторые пакеты также поддерживают такие типы, как например MONEY (ДЕНЬГИ) и BINARY (ДВОИЧНЫЕ). MONEY – это специальная система исчисления, используемая компьютерами.

date	8 байт	01.01.0100 до 11.12.5941
blob	переменный любой тип двоичных данных, например файл *BMP*.	

3 ВВОД ЗНАЧЕНИЙ

3.1 Оператор *insert*

Все строки в SQL вводятся с использованием команды модификации INSERT. В самой простой форме, INSERT использует следующий синтаксис:

```
insert into <table name>
values ( <value>, <value> . . .);
```

Так, например, чтобы ввести строку в таблицу Students , вы можете использовать следующее условие:

```
insert into Students
values (11, "Комсомоленко", "В-31");
```

Команды не производят никакого вывода, но ваша программа должна дать вам некоторое подтверждение того, что данные были использованы. Имя таблицы (в нашем случае – Students), должно быть предварительно определено, в команде CREATE TABLE, а каждое значение пронумерованное в предложении значений, должно совпадать с типом данных столбца, в который оно вставляется. Значения, конечно же, вводятся в таблицу в поименном порядке, поэтому первое значение с именем, автоматически попадает в столбец 1, второе в столбец 2 и так далее.

3.2 Вставка пустых указателей (*null*)

Если вам нужно ввести *пустое* значение(NULL), вы вводите его точно также как и обычное значение.

Так как значение NULL – это специальный маркер, а не просто символьное значение, он не включается в одиночные кавычки.

3.3 Именование столбца для вставки (*insert*)

Вы можете также указывать столбцы, куда вы хотите вставить значение имени. Это позволяет вам вставлять имена в любом порядке. Предположим, что вы берете значения для таблицы студентов из отчета выводимого на принтер, который помещает их в таком порядке: группа, фамилия и для упрощения, вы хотите ввести значения в том же порядке:

```
insert into Students (группа, фио)
values ("В-31", "Репин");
```

4 СОЗДАНИЕ ДОМЕНА

4.1 Оператор *create domain*

Обратите внимание, что столбец номер_в_жур
тически установлена в значение – по умолчанию. П
или другое значение определяемое как – по умолча
ния NULL в данном столбце, и этот столбец не уст
обеспечен значением для любой команды INSERT

Если в таблице присутствуют столбцы с одинаковыми характеристиками, то можно предварительно описать их тип и поведение с помощью домена. Например, создать тип *Pol_type* для таблицы *Sotr*.

```
Create domain Pol_type as char (3) collate PXW_CYRL;
Create table Sotr (
Fio char (20) not null,
Pol Pol_type,
otdel char (10),
dolj char (20),
primary key (Fio)
);
```

где *Not null* – столбцы ассоциированные с доменом обязательно должны содержать значение.

4.2 Ограничения на значения домена

Записываются в определении столбца и начинаются со слова *check*.

<огр_домена>={
value <оператор> <значение>
| *value* [*not*] *between* <значение 1> *and* <*значение* 2>
| *value* [*not*] *like* <значение> [*escape* <*значение*>]
| *value* [*not*] *in* (<значение 1> [, <значение 2>…])
| *value is* [*not*] *null*
| *value* [*not*] *containing* <*значение*>
| *not* <огр_домена>
| <огр_домена> *or* <огр_домена> | (<огр_домена>)
| <огр_домена> *and* <огр_домена>
}

где:

<оператор> = {= |<|>|<=|=>|!<|>!|<>|!=}

!< – не меньше, !> – не больше, != – не равно.

Value означает, что элементы считаются правильными.

Create domain ID_Type as integer check (*value* >=100); *id_Type* >= 100.

Between <значение 1> *and* <значение 2> – значение домена в интервале от значения 1 до значения 2, включая их.

Like <значение 1> [<значение 2>] – значение домена должно "походить" на значение 1.

Like "%*USD*" – вводимое значение должно оканчиваться на *USD*, все предыдущие значения не имеют.

Like "__04" ("_"– единичный символ) вводится значение четырьмя символами, два последних – 04.

Если "%" и "_" нужны как символы в *Like*, то указываются значения в *Escape* и заменяются другими символами.

Например, *Summa* должна заканчиваться %⇒

Create domain summa as char (10) *check* (*like* "%!%" *escape* "!");

После символа ! служебные символы (%) теряют свою силу и становятся обычными символами.

In (<значение 1>[,<значение 2>…]) – значение домена должно совпадать с одним из приведенных параметров списка.

Create domain Pol_type as
char (3) *check* (*value in* ("муж","жен"));

Containing <значение> – значение домена должно иметь вхождения параметра <значение> в любом месте.

Например, в наименовании отдела вхождения 041 где угодно "отдел – 041002" или "00304192" и т. д.

Create domain otdel_type as
varchar (10) *check* (*value containing* "041" *collate PXW_CYRL*;

Starting [*with*] <значение> – значение домена должно начинаться с <значение>, например "041".

Большинство условий могут комбинироваться *AND* или *OR* или указывать *NOT*.

Check (*value not between* 1 *and* 100);

и т.д.

4.3 Изменения домена

Alter domain имя {
[*set default* {литерал | *null* | *user*}]
| *drop default*]
| [*add* [*constraint*] *check* (<огр_домена>)]
| [*drop constraint*]
};

Для столбца *not null* определение уже нельзя изменить и тип данных тоже.

Set default – по умолчанию;

Drop default – отменяет значение по умолчанию;

Add [*constraint*] *check* (<огр_домена…>) добавляет условие на значения столбцов;

Drop constraint – удаляет условия.
Пример

Create domain id_type as
integer check (*value* >=100);

Пусть определен домен:

в таблице *A*

create table A (
id id_type not null,
fio varchar (20),
primary key (*id*));

Изменить ограничение на $100 \le x \le 500$

1) удалить старое условие *alter domain id_type drop constraint*;
2) добавить новое условие *alter domain id_type check* (*value* >= 100 *and value* <=500).

5 СОЗДАНИЕ ТАБЛИЦ

5.1 Оператор *create table*

Перед созданием таблиц БД необходимо продумать определения всех столбцов таблицы и характеристик каждого столбца. При определении таблицы применяются домены. БД, в которую добавляется таблица, должна быть открыта.

Создание таблицы БД осуществляется оператором
Create table имя_табл [*external* [*file*] "<имя файла>"]
(<опр_ст>[, < опр_ст > | <ограничение>…]);
[*external* [*file*] "<имя файла>" – относится к внешним таблицам БД.
<опр_ст> – определение столбца БД имеет формат:
<опр_ст>= имя_ст {тип_данных | *computed* [*BY*]
(<выражение>) | домен}
[*default* {литерал | *null* | *user*}]
[*not_null*] [<огранич_столбца>]
[*collate collation*]
computed [*by*] (<выражение>) – определение столбца вычисляемых значений.
Default – значение столбца по умолчанию, ассоциировано с доменом.
<огранич_столбца> – ограничение на значение столбца.
Collate collation – порядок сортировки символов.

5.2 Столбцы вычисляемых значений

Значение таких столбцов не вводится, а вычисляется согласно выражению, например в таблице *A* есть столбцы номера квартала *N_Q*, количество продаж в данном квартале в прошлом году *kol_s* и текущем году *kol_n* и прирост продаж за квартал *prirost*:

Create table A (
N_Q integer not null,
Kol_s integer,
Kol_n integer,
prirost computed by (*kol_n – kol_s*),
primary key (*N_Q*));

5.3 ОГРАНИЧЕНИЯ ЦЕЛОСТНОСТИ

Бывают двух уровней, на уровне столбца или на уровне всей таблицы. Наложение ограничения целостности на отдельный столбец следует за его именем и типом:

Tovar varchar (20) *not null primary key*,
create table… (
tovar varchar (20) *not null*,
…,
primary key (*tovar*));

во втором случае ограничения указываются по

primary key – это первичный ключ построений по столбцу или столбцам.

Столбцы должны иметь значение *not null*. Первичный ключ служит для установления связи в внешним ключом (*foreign key*) дочерней таблицы и определяет ссылочную целостность между родительской и дочерней таблицами.

5.4 Уникальный ключ

Строится по столбцу (столбцам) когда столбец не входит в первичный ключ и имеет уникальное значение /нет одинаковых значений/

Create table klient (
Imja_klienta varchar (20) *not null primary key*,
Nom_scheta varchar (50) *not null*,
Unique (*nom_scheta*));

Или *nom_scheta varchar* (50) *not null unique*.

5.5 Внешний ключ

Строится в дочерней таблице для соединения с родительской.

Форматы:

Foreign key (<список _ст_внешнего_ключа>)
references <имя_род_табл>
[<список_ст_род_табл>]
[*on delete* {*no action* | *cascade* | *set default* | *set null*}]
[*on update* {*no action* | *cascade* | *set default* | *set null*}]

список_ст_внешнего_ключа – столбцы дочерней таблицы;

имя_род_табл – таблица, в которой описан первичный ключ (или столбец с атрибутом *unique*);

список_ст_род_табл – не обязателен при ссылке на первичный ключ родительской таблицы, в других случаях необходим.

On delete, *on update* – способ изменения записей дочерней таблицы при удалении или изменении поля связи в родительской таблице.

No action – запрет;

Cascade – название;

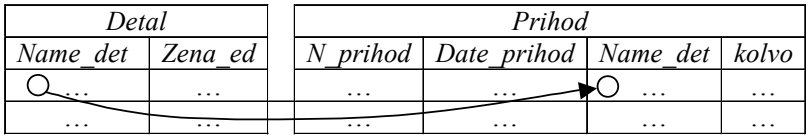
Set default – в поле дочерней таблицы заносится значение, определенное ранее по умолчанию;

Set null – заносится значение *null*.

Пример: определим две таблицы.

Родительская *detal* с полями *name_det* (имя детали) и *zena_ed* (цена за единицу), первичный ключ по полю *name_det*.

Дочерняя *prihod* (приход со склада) с полями *n_prihod* (номер прихода), *date_prihod* (дата прихода), *name_det* (имя детали), *kolvo* (количество деталей в приходе). Первичный ключ по *n_prihod*, внешний – *name_det*.



Create table detal (
Name_det varchar (20) *not null*[1],
Zena_ed integer not null,
primary key (*name_det*));
Create table prihod (
N_prihod integer not null primary key,
Date_prihod date not null,
Name_det varchar (20) *not null*[1],

[1] Должны быть описаны одинаково.

Kolvo integer not null
Foreign key (*name_det*) *references detal*).

5.6 Именование ссылочной целостности

Ссылочная целостность может именоваться следующим образом:

[*constraint* <имя ссылочной целостности>]
foreign key (<список столбцов внешнего ключа>)
references <имя родительской таблицы>
[<список столбцов родительской таблицы>]

Необязательное имя ссылочной целостности присутсвует в смешанных сообщениях и может использоваться при анализе БД.

Для конкретного примера в таблицу *prihod* добавим:

Kolvo integer not null,
Constraint po_detaly
foreign key (*name_det*) *references detal*;

5.7 Требования к значениям столбцов

Определяются как на уровне отдельного столбца так и на уровне всей таблицы.

Например, для таблицы параметров человека (*parametr*) рост (*rost*) должен быть больше веса (*ves*). Ограничения можно записать:

Create table parametr (
Id integer not null primary key,
Rost integer not null,
Ves integer not null check (*rost* > *ves*));

или

Create table parametr (
Id integer not null,
Rost integer not null,
Ves integer not null,
primary key (*id*),
check (*rost* > *ves*));

Ограничения, накладываемые на столбец, определяются предложением *check*, имеющим формы:

Check (<условия_поиска>)

где

<условия_поиска> =
{<значение><оператор>{<значение1 > | (<выбор_одного>)}
| <значение> [*not*] *between* <значение 1> *and* <значение 2>
| <значение> [*not*] *like* <значение> [*escape* <значение>]
| <значение> [*not*] *in* <значение 1>[, <значение 2>…] | <выбор_многих>)
| <значение> *is* [*not*] *null*
| <значение> {[*not*] {=| < | > } >= | <= } {*all* | *some* | *any* } (<выбор многих>)
| *exists* (<выражение_выбора>)
| *singular* (<выражение_выбора>)
| <значение> [*not*] *containing* <значение 1>
| <значение> [*not*] *starting* [*with*] <значение 1>
| (<условие_поиска>)
| *not* (<условие_поиска>)
| <условие_поиска> *or* <условие_поиска>
| <условие_поиска> *and* <условие_поиска>

где

<значение>={столбец | <константа> | <выражение> | <функция> | *null* | *user* | *RDB*$*DB_key*
константа = число | "строка"
функция = {
count (* | [*all*] <значение> | *distinct* <значение>)

| *sum* ([*all*] <значение> | *distinct* <значение>)
| *avg* ([*all*] <значение> | *distinct* <значение>)
max([*all*] <значение> | *distinct* <значение>)
min([*all*] <значение> | *distinct* <значение>)
cast (<значение> *as* <тип_данных>)
upper (<значение>)
| *gen_id* (генератор, <значение>)
}
выбор_одного – *select* возвращает одно значение;
выбор_многих – *select* возвращает список или ни одного выражения выбора *select;*
singular – true для списка из одной строки;
exists – true если вписок не пустой.

6 ИЗМЕНЕНИЕ ТАБЛИЦЫ

6.1 Оператор *alter table*

Команда **ALTER TABLE** не часть стандарта ANSI; но это – широко доступная, и довольно содержательная форма, хотя ее возможности несколько ограничены. Она используется, чтобы изменить определение существующей таблицы. Обычно, она добавляет столбцы к таблице. Иногда она может удалять столбцы или изменять их размеры, а также в некоторых программах добавлять или удалять ограничения. Типичный синтаксис чтобы добавить столбец к таблице:

alter table <имя_табл> ADD <имя_столбца>
<data type> <size>;

Столбец будет добавлен со значением NULL для всех строк таблицы. Новый столбец станет последним по порядку столбцом таблицы. Вообще то, можно добавить сразу несколько новых столбцов, отделив их запятыми, в одной команде. Имеется возможность удалять или изменять столбцы. Наиболее часто, изменением столбца может быть просто увеличение его размера, или добавление (удаление) ограничения. Ваша система должна убедиться, что любые изменения не противоречат существующим данным – например, при попытке добавить ограничение к столбцу, который уже имел значение, при нарушении которого ограничение будет отклонено. Лучше всего дважды проверить это.

Оператор *alter table* позволяет:

- добавить определение нового столбца;
- удалить столбец из таблицы;
- удалить атрибуты целостности таблицы или отдельного столбца;
- добавить новые атрибуты целостности.

Пусть имеем таблицу:

Create table sotr (
Id_sotr integer not null primary key,
Fio char (10),
Otdel varchar (10),
Doljnost char (10));

Пусть необходимо изменить характеристики столбца *fio c char* (10) на *varchar*(25):

1 добавим в таблицу новый временный столбец *fio_tmp*:

alter table sotr
add fio_tmp char (10);

2 копируем данные из *fio* в *fio_tmp*:

update sotr

По крайней мере, посмотрите документацию
что именно это было причиной. Из-за нестандартн
необходимо посмотреть тот раздел вашей системн

ALTER TABLE – не действует, когда таблица
тывать вашу базу данных по возможности так что
структуры таблицы, когда она уже в использован
торые, являясь вторичными таблицами с извлечен
но работают, а программы, использующие вложе
правильно. Кроме того, изменение может стереть
к таблице. По этим причинам, вы должны разраба
TABLE только в крайнем случае.

set fio_tmp = fio;
3 удаляем столбец *fio*:
alter table sotr
drop fio;
4 создаем новый стобец *fio*:
alter table sotr
add fio varchar (25);
5 переписываем данные:
update sotr
set fio = fio_tmp;
6 удаляем временный столбец:
alter table sotr
drop fio_tmp.

6.2 Изменение атрибутов столбца

Добавление нового столбца в таблицу БД:
Alter table <имя таблицы> *add* <определение столбца>;
Добавление новых ограничений целостности:
Alter table <имя таблицы> *add* [*constraint* <имя ограничения>] <определение целостности>;
Удаление столбца (столбцов) из таблицы:
Alter table <имя таблицы> *drop* <имя столбца1> [, <имя столбца 2>…];
Удаление ограничений целостности (уровень таблицы):
Alter table <имя таблицы> *drop* <ограничения целостности>;
Пример. Для таблицы *prihod*:
Create table prihod (
N_prihod integer not null primary key,
Date_prihod date not null,
Name_det varchar (20) *not null*,
Kolvo integer not null
Constraint po_tovaru
foreign key (*name_det*) *references detal*);
Удалить целостность *po_tovaru*:
Alter table prihod drop po_tovaru.

6.3 Удаление таблицы

Удаление таблицы целиком:
Drop table <имя таблицы>;
Удаление может быть блокировано для родительских таблиц, если есть дочерние. Удаление таблицы разрушит ссылочную целостность.

7 РАБОТА С ИНДЕКСАМИ

Первичный и внешний ключи строятся для обеспечения ссылочной целостности реляционно-связанных таблиц. Кроме этого, первичный ключ, выполняет функции уникальности своих значений. Для этих же целей используется и просто уникальный ключ.

Индексы, в отличие от ключей, создаваемые оператором *create index*, служат для сортировок и оптимизации доступа к данным. В конечном счете, ключи и индексы преобразуются в *физические индексы* – специальный механизм быстрого доступа к данным.

7.1 Необходимость создания индексов

Создаются в случаях, когда:

– часто производится поиск в БД (столбец (столбцы)) часто перечисляется в предложении *where* оператора *select*);
– часто строится объединение таблиц;
– часто производится сортировка (*order by* в операторе *select*).

Не рекомендуется строить индексы по столбцам или группами столбцов, которые:
– редко используются для поиска;
– часто меняют значение (надо часто обновлять индекс);
– содержит небольшое число вариантов значения.

7.2 Создание индекса, оператор *create index*

Оператор
Create [*unique*] [*asc*[*ending*] | *desc* [*ending*]]
Index <имя индекса> *on* <имя таблицы> (столбец 1,…);
Unique – уникальный индекс, не допускает одинаковых значений
Asc[*ending*] – сортировка полей индекса по возрастанию (по умолчанию)
desc[*ending*] – сортировка полей индекса по убыванию.
Пример. Для таблицы *prihod*:

```
Create table prihod (
N_prihod integer not null primary key,
Date_prihod date not null,
Name_det varchar (20) not null,
Kolvo integer not null);
```

Создать индекс в порядке убывания значений *date_prihod* и *name_det*:

```
Create desc index D_P
On prihod (date_prihod, name-det);
```

После многократного внесения изменений в таблицу индексы могут быть разбалансированы, "глубина" индекса возрастает.
Это приводит к увеличению времени поиска.
Необходимо время от времени:
– выполнять балансировку индекса оператором *alter index*;
– переписывать выбираемость индекса оператором *set statistics*;
– уничтожать и вновь создавать индекс операторами *drop index* и *create index*.

7.3 Пересоздание и балансировка индекса

1. *alter index* <имя индекса> *deactivate*;
2. *alter index* <имя индекса> *activate*;

Нельзя:
– перестроить индекс, находящийся в запросах,
– перестроить индекс, построенный по первичному, внешнему ключу.

7.4 Удаление индекса

Drop index <имя индекса>;

8 ЗАПРОСЫ В SQL

8.1 Оператор *select*

Позволяет производить выборки данных, преобразовывать полученные результаты, реализует сложные условия выбора.
Формат оператора:
Select [*distinct* | *all* {* | <значение 1> [, <значение 2>…]}

from <таблица1 > [, <таблица2 >…]
[*where* <условие_поиска>]
[*group by* столбец [*collate collation*]
[, столбец 1 [*collate collation*]…]
[*having* <условия_поиска>]
[*union* <оператор_*select*>]
[*plan* <план_выполнения_запроса>]
[*order by* <список_столбцов>];

В простейшем случае, когда требуется просмотреть все записи одной или нескольких таблиц, оператор имеет вид:

Select {* | <значение 1> [, <значение 2>…]}
from <таблица 1> [, <таблица 2>…];

<значение 1>, <значение 2>… – имя столбца возвращаемого оператором, * – все столбцы
<таблица 1>, <таблица 2>… – имя таблицы, из которой происходит выборка данных.

Например, создать набор данных, состоящий из всех столбцов:

Select *
from prihod;

Такой же набор данных можно получить:

Select n_prihod, *date_prihod*, *name_det*, *kolvo*,
from prihod;

Предложение *where* используется для включения в БД лишь нужных записей, удовлетворяющих условию:

Select { * | <значение 1> [, <значение 2>…]}
from <таблица 1> [, <таблица 2>…]
where <условия поиска>;

8.2 Сравнение с константой

При сравнении значения столбца с константой условие поиска имеет вид:

<имя_столбца> < оператор> < константа>
<оператор> – =, <, >, <= (!>), >=(!<), <>(!=).

Например, выбрать из таблицы *prihod* все операции приема товара объемом 20 единиц:

Select *
from prihod
where kolvo=20;

8.3 Внутреннее соединение таблиц

При сравнении значения столбца одной таблицы со значением столбца другой таблицы условие поиска имеет вид:

<имя_столбца_табл1> <оператор> <имя_столбца_табл2>.

Например, выбрать все записи о приходе деталей из таблицы *prihod* и для каждой детали указать его цену из таблицы *detal*:

Select prihod.*, *detal.zena_ed*
from prihod, *detal*
where prihod.name_det = detal.name_det;

Для каждой записи из таблицы *prihod* ищется запись в таблице *detal*, у которой значение в поле *name_det* совпадает со значением *name_det* текущей записи таблицы *prihod*. Порядок перечисления в условии поиска значения не имеет:

Prihod.name-det = detal.name_det

или

Detal.name_det = prihod.name_det

N_prihod	*Date_prihod*	*Name_det*	*Kolvo*	*Name_det*	*Zena_ed*

1	20.09.00	*D*1	100	*D*1	1.00
2	29.09.00	*D*1	100	*D*2	50.00
3	1.10.00	D*2*	200	*D*3	30.00
4	1.10.00	*D*1	200	*D*4	100.00
5	2.10.00	*D*2	200	*D*5	35.00
6	4.10.00	*D*3	100	*D*6	50.00
7	4.10.00	*D*3	110	*D*7	55.00

8.4 Использование псевдонимов таблиц

Идентификация столбцов через имя таблицы неудобно из-за громоздкости обозначений. Лучше присвоить каждой таблице краткое имя.

Такие имена называются *псевдонимами* таблиц. Они отделяются пробелом от фактического имени таблицы в списке *from*:

Select …
from <таблица 1 псевдоним 1> [, <таблица 2 псевдоним 2> …]
where … ;

Например:

Select prihod.*, *detal.zena_ed*
from prihod P, *detal D*
where P.name-det = D.name_det;

8.5 Определение сортировки *order by*

Результирующий НД можно упорядочить с помощью предложения:

Order by <список_столбцов>.

Если в списке столбцов указано больше одного столбца, то первый будет использоваться для глобальной сортировки, второй – для сортировки внутри группы, определенной единым значением первого столбца, и т.д.

Например, показать все записи приема деталей, отранжировать по имени детали:

Select name_det, *zena_ed*
from detal
where zena-ed >= 35;
order by name_det;

Name_det	*Zena_ed*
*D*2	50.00
*D*4	100.06
*D*5	35.00
*D*6	50.00
*D*7	55

или

Select n_prihod, *date_prihod*, *name_det*, *kolvo*
from prihod
order by name_det, *kolvo*, *date_prihod*;

N_prihod	*Date_prihod*	*Name_det*	*Kolvo*
1	20.09.00	*D*1	100
2	29.09.00	*D*1	100
3	4.10.00	*D*1	100
4	1.10.00	*D*1	200
5	1.10.00	*D*2	200
6	2.10.00	*D*2	200
7	4.10.00	*D*3	110

8.6 Устранение повторяющихся записей

Ключевое слово *Distinct*. Повторяющимися считаются записи, содержащие идентичные значения во всех столбцах результирующего НД. Если в результирующем НД нужно вносить все записи, то указывают ключевое слово *All* (по умолчанию *All*).

Например, получить наименование всех деталей, полученных на склад:

```
Select distinct name_det
from prihod;
```

Name_det
*D*1
*D*2
*D*3

8.7 Расчет вычисляемых столбцов

Для расчета вычисляемых столбцов результирующего НД используются арифметические выражения:

Select [*distinct* | *all*] { * | <столбец 1> [, <выражение 1>…]}
from <таблица 1> [, <таблица 2>…];

Если столбцу надо присвоить нестандартное имя, то оно может быть указано за выражением при помощи ключевого слова *As*.

Например: рассчитать общую стоимость полученных деталей для каждого факта получения:

```
Select p.*, d.zena_ed, p.kolvo * d.zena_ed
as stoim
from prihod p, detal d
where p.name_det = d.name_det;
```

N_prihod	*Date_prihod*	*Name_det*	*Kolvo*	*Zena_ed*	*Stoim*
1	20.09.00	*D*1	100	1.00	100.00
2	29.09.00	*D*1	100	1.00	100.00
3	1.10.00	*D*2	200	50.00	10000.00
4	1.10.00	*D*1	200	1.00	200.00
5	2.10.00	*D*2	200	50.00	10000.00
6	4.10.00	*D*1	100	1.00	100.00
7	4.10.00	*D*3	110	30.00	3300.00

8.8 Агрегатные функции

– *count* (<выражение>) – подсчитывает число вхождений значения выражения во все записи результирующего НД;
– *sum* (<выражение>) – суммирует значение выражения;
– *avg* (<выражение>) – находит среднее значение;
– *max* (<выражение>) – определяет максимальное значение;
– *min* (<выражение>) – определяет минимальное значение.

а) количество наименований деталей, оприходованных на список:

```
select count (distinct name_det) as count_nam
from prihod;
```

Count_name
3

б) вычислить общую стоимость оприходованных деталей за 4.10.00:

```
select sum (p.kolvo * d.zena_ed) as itogo
from prihod p, detal d
where (p.name_det = d.name_det) and
```

Itogo
3400.00

(*p.date_prihod* = "04.10.00");

8.9 Группировка записей

Для группы записей столбца, характеризующие одинаковые значения можно получить агрегированные значения (*min*, *max*, *avg*). При этом один из столбцов представляется агрегирующей функцией, и предложение *group by* столбец [, столбец…] перед предложением *where*.

Например, получить общее количество прихода деталей по каждой из них:

Select p.name_det sum (*p.kolvo*) *as pr*
from prihod p
group by p.name_det;

Name_det	*Priem*
*D*1	500
*D*2	400
*D*3	110

Или общая цена на каждую деталь на каждую дату:

Select p.name_det, *p.date_prihod*, *sum* (*p.kolvo * d.zena_ed*) *as sum*
from prihod p, *detal d*
where p.name_det = d.name_det
group by p.name_det, *p.date_pri*

Name_det	*Date_prihod*	*Sum*
*D*1	20.09.00	
*D*1	29.09.00	
*D*2	1.10.00	
*D*1	1.10.00	
*D*2	2.10.00	
*D*1	4.10.00	
*D*3	4.10.00	

8.10 Предложение *having*

Наложение ограничений на группировку записей.

Агрегация выдается только по группам удовлетворяющим условию.

Формат: после *group by*

Having <агрег_функц> <отношение> <значение>

Агрег_функц – *min*, *max*, *avg*, *sum*;

Отношение – =, <>, <, <=, >=;

где:

Значение – константа, результат вычисления или единичное значение, возвращенное *select*.

Найти минимальный приход товара (деталей) не меньше 100 единиц.

Select name_det, *min* (*kolvo*)
from prihod
group by name_det
having min (*kolvo*) >=100;

Name_det	*Min*
*D*1	100
*D*2	200
*D*3	110

Можно задавать разные функции для столбца и условия *having*.

В условие *where* нельзя вносить агрегированную функцию.

Where – исключает не удовлетворяющие условию.

Having – исключает группы с агрегированными значениями.

8.11 Использование подзапросов

Часто невозможно решить поставленную задачу путем использования единственного запроса. Например, в тех случаях, когда при использовании условия поиска в предложении *where* значение с которым надо сравнить, далее не определено, а вычисляется оператором *select*.

В таких случаях применяют вложенные запросы или подзапросы.

Оператор *select* имеет вид:

Select …
from …
where <сравниваемое значение> <оператор> (*select*);

Пусть надо найти дату, на которую приходится максимальный приход деталей. Тогда запрос может быть записан так:

Select kolvo, *date_prihod*

Kolvo	*Date_prihod*
200	1.10.00
200	2.10.00

from prihod
where kolvo = (*select max*
(*kolvo*) *from prihod*);

Оператор *select* возвращает не одно значение, а список. Поэтому может возникнуть ошибка. Чтобы ее избежать надо заменить оператор = на оператор выбора из нескольких возможных значений (*in*).

Синтаксис вложенного запроса ничем не отличается от синтаксиса основного запроса. Это значит, что в подзапрос может быть вложен другой подзапрос и т.д.

Например, составим список получения деталей от поставщика, который в свое время поставил максимальную партию любой детали:

*Select p*0.*
*from prihod p*0
*where p*0.*post in*
(*select p*1.*post*
*from prihod p*1
where kolvo in
(*select max* (*p*2.*kolvo*)
*from prihod p*2));

Сначала определим *max*, далее имя поставщика, а затем все записи, связанные с данным поставщиком.

Таблица *prihod* имеет дополнительно столбцы поставщика:

Post – имя поставщика; *Gorod* – город поставщика; *Addr* – адрес поставщика.

8.12 Внешние соединения

Определяются в предложении *from* согласно спецификации:

Select { * | <значение 1> [, <значение 2>…]}
from <таблица 1> <вид соединения > *join* < таблица 2>
on <условие поиска>;

Внешнее соединение отличается от внутреннего тем, что в результирующий НД включаются записи ведущей таблицы соединения, которые объединяются с пустым множеством записей другой таблицы. Какая из таблиц будет ведущей, определяет вид соединения (*left* – левое внешнее соединение, ведущая таблица 1; *right* – правое внешнее соединение, ведущая таблица 2; *full* – полное внешнее соединение).

В случае полного внешнего соединения ведущими являются обе таблицы. В результирующий НД вкладываются все записи обеих таблиц согласно алгоритму:

1 если для записи таблицы 1 имеются записи таблицы 2, удовлетворяющие условию соединения, то в результирующий НД включаются все комбинации записей таблиц 1 и 2;

2 иначе в НД включается запись таблиц 1, соединенная с пустой записью таблицы 2;

3 п. 1 и 2 повторяются для таблицы 2 и таблицы 1.

Пример. Пусть имеем таблицы *A* и *B*.

A

*P*1	*P*2	*P*3
a	*x*	400
b	*x*	200
c	*y*	500
d	–	–

B

*P*1	*P*2
x	1
y	2
z	2

Тогда выполнение оператора

Select A.*P*1, *A*.*P*2, *B*.*P*2
from A left join B
on A.*P*2 = *B*.*P*1;

A.*P*1	*A*.*P*2	*A*.*P*2
a	*x*	1
b	*x*	1
c	*y*	2
d	–	–

A.*P*1	*A*.*P*2	*A*.*P*2
a	*x*	1
b	*x*	1
c	*y*	2
–	–	2

Запрос внешнего правого соединения:

*Select A.P*1, *A.P*2, *B.P*2
from A right join B
*on A.P*2 = *B.P*1;

дает другой результат:

Выполнение оператора полного внешнего соединения таблиц *A* и *B*:

*Select A.P*1, *A.P*2, *B.P*2
from A full join B
*on A.P*2 = *B.P*1;

приведет к результату:

*A.P*1	*A.P*2	*B.P*1	*B.P*2
a	*x*	*x*	1
b	*x*	*x*	1
c	*y*	*y*	2
d			
		z	2

В запросах к БД эта операция может быть использована, например, когда требуется найти поставщика, соответствующего каждой поставке детали или найти все поставки по каждому поставщику и т.п.

Select p.date_prihod, *p.name_det*, *p.kolvo*, *p*1.*post*, *p*1.*gorod*
*from prihod p left join postavshik p*1
*on p.name_det = p*1.*name_det*;

или

Select p.date_prihod, *p.name_det*, *p.kolvo*, *p*1.*post*, *p*1.*gorod*
*from prihod p right join postavshik p*1
*on p.name_det = p*1.*name_det*;

Объединение результатов нескольких операторов *select*.

Объединение производится оператором *union*. Результирующие *HD* должны иметь одиноковую структуру. Одинаковые записи не дублируются.

Пример: соединим результаты трех запросов:

1 *select p.**
from prihod p
where p.name_det containing ‘3’;

7	4.10.00	*D*3	110	*Samsung*

2 *select p.**
from prihod p
where p.kolvo > 100;

3	1.10.00	*D*2	200	*Intel*
4	1.10.00	*D*1	200	*Microsoft*
5	2.10.00	*D*2	200	*Intel*
7	4.10.00	*D*3	110	*Samsung*

3 *select p.**
from prihod p
where p.post = ‘*AMD*’;

2	29.09.00	*D*1	100	**AMD**

Произведем объединение трех результирующих наборов данных:

*select p.**
from prihod p
where p.name_det containing ‘3’;
union
*select p.**
from prihod p
where p.kolvo > 100;
union
*select p.**
from prihod p
where p.post = ‘*AMD*’;

результирующий НД:

N_prihod	*Date_prihod*	*Name_det*	*Kolvo*	*Post*
2	29.09.00	*D*1	100	*AMD*
3	1.10.00	*D*2	200	*Intel*
4	1.10.00	*D*1	200	*Microsoft*
5	2.10.00	*D*2	200	*Intel*
7	4.10.00	*D*3	110	*Samsung*

9 ДОБАВЛЕНИЕ, ИЗМЕНЕНИЕ, УДАЛЕНИЕ ЗАПИСЕЙ

Формат оператора:

Insert into <объект>[(столбец 1 [, столбец 2 …])]
{*values* (<значение 1> [, <значение 2>…]) |
< оператор *select*> };

Если значения присваиваются всем столбцам по порядку, то имена столбцов можно не писать, в противоположном случае миена столбцов надо указать.

Insert into prihod
values (8, "12-*oct*-2000", "*D*4", 50, "*AMD*");

Вместо явного указания номера прихода, можно воспользоваться генератором уникального ключа:

Insert into prihod
values (*gen_ID* (*prihod_n_prihod*, 2), "12-*oct*-2000", "*D*4", 50, "*AMD*");

Применение оператора *select* не нарушает порядка присваивания значений столбцам.

Пусть определена таблица *prihod_data* аналогичная *prihod*:

Create table prihod_data (
N_prihod integer not null,
Date_prihod date not null,
Name_det varchar (20) *not null*,
Kolvo intreger not null,
Post varchar (20),
Primary key (*n_prihod*));

Пусть в эту таблицу надо ежедневно копировать все записи о полученных деталях, например для передачи в бухгалтерию. Тогда выгрузка записей из *prihod* в таблицу *prihod_data* будет реализована оператором:

Insert into prihod_data
Select *
from prihod
where date_prihod = "13-*oct*-2000";

Оператор *update* применяется для изменения значения в группе записей или одной записи.

Формат:

Update <объект>
set столбец 1 = <значение 1> [, столбец 2 = <значение 2>…]
[*where* <условие поиска>]

Если опустить *where* <условие поиска>, то будут изменены все записи, в противном случае только записи, удовлетворяющие условию.

Например, заменить дату 10.10.00 и увеличить количество товара на 10 единиц для всех записей с датой 4.10.00 в таблице *prihod*:

Update prihod
set date_prihod = "10-*oct*-2000",
kolvo = *kolvo*+10,
where date_prihod = "4-*oct*-2000";

Оператор *delete* применяется для удаления групп записей из объекта (таблиц или *view*).

Формат:

Delete from <объект>
[*where* <условие поиска>];

Удаляются все записи если нет *where* <условие поиска>, иначе только записи, удовлетворяющие условию.

Пример: удалить все записи детали *D*1 за 10.10.2000.

Delete from prihod
where (*name_det* = "*D*1") *and*
(*date_prihod* = "10-*oct*-2000");

10 ПРОСМОТР *View*

В БД может быть определен просмотр, являющийся виртуальной таблицей, в которой представлены все записи из одной или нескольких таблиц.

Создать просмотр:

Create view имя просмотра [(столб_*view*1 [, столб_*view*2…])]
as <оператор *select*> [*with check option*]; (если указан, то все изменения, не удовлетворяющие условиям, будут отвергаться)

Удалить просмотр:

Drop view имя просмотра;

Преимущества использования просмотров:

однажды определив *view*, не нужно всякий раз формировать *select* для запроса;

представляет подмножество столбцов из таблиц, что усиливает безопасность данных.

Пример. Создать просмотр, содержащий дату прихода, имя детали, количество из таблицы *prihod* и цену из таблицы *detal*:

Create view full
as
select p.date_prihod, *p.name_det*, *p.kolvo*, *d.zena_ed*
from prihod p, *detal d*
where p.name_det = d.name_det;

Теперь обращение с запросом становится простым:

*Select * from full*;

Действие над просмотром не повредит исходной таблице.

10.1 Способы формирования *view*

1 вертикальный срез таблицы (включается подмножество столбцов):

create view p_vert
as
select name_det, *kolvo*
from prihod;

2 горизонтальный срез таблицы (все столбцы, но не все записи)

create view p_horiz
as
*select **
from prihod
where name_det = "*D*1";

3 вертикально-горизонтальный срез таблицы (не все столбцы и не все строки)

create view p_vh
as
select name_det, *kolvo*
where name_det = "*D*1";

4 подмножество строк и столбцов разных таблиц:

create view full_p
as
select p.name_det, *p.kolvo*, *d.zena_ed*
from prihod p, *detal d*
where p.name_det = d.name_det;

Имя столбца указывается в имени просмотра, если столбец вычисляемый.

Чтобы просмотр можно было обновлять, необходимо:

−формировать *view* из записей одной таблицы;

−включить все столбцы с атрибутами *not null* в исходной таблице;

−оператор *select* просмотра не должен использовать агрегирующих функций, а также *distinct*, *having*, *join*, хранимых процедур и функций.

Если *view* удовлетворяет этим требованиям, то к нему применимы операторы *insert*, *update* и *delete*.

СПИСОК ЛИТЕРАТУРЫ

1 Вейнеров О.М., Самохвалов Э.Н. Проектирование баз данных САПР. М.: Высш. шк., 1990. 144 с. Кн. 4.

2 Ульман Дж. Основы систем баз данных. М., 1983.

3 Федорук В.Г., Черненький В.М. Системы автоматизированного проектирования. Кн. 3: Информационное и прикладное программное обеспечение. Минск: Высш. шк., 1988. 157 с.

4 Тиори Т., Фрай Дж. Проектирование структур баз данных. М.: Мир, 1985. 287 с. Т. 1, 2.

5 Хорафас Д., Легг С. Конструкторские базы данных. М.: Машиностроение, 1990. 224 с.

6 Диго С.М. Проектирование и использование баз данных. М.: Финансы и статистика, 1995. 208 с.

7 Дейт К. Дж. Введение в системы баз данных. Киев: Диалектика, 1998. 784 с.

8 Фаронов В.В., Шумаков П.В. Delphi 4. Руководство разработчика баз данных. М.: Нолидж, 1999. 560 с.

9 Карпова Т.С. Базы данных: модели, разработка, реализация. СПб.: Питер, 2001. 304 с.

оглавление