**Лекции по дисциплине «Базы данных»**

**Содержание**

Основные понятия и определения 4

Архитектура базы данных. 5

Процесс прохождения пользовательского запроса 6

Пользователи банков данных 7

Классификация моделей данных 8

Теоретико-графовые модели данных 9

Иерархическая модель данных 10

Сетевая модель данных 14

Реляционная модель данных 17

Основы реляционной алгебры 20

Операции над множествами. 20

Специальные операции. 22

Проектирование реляционных БД на основе принципов нормализации 25

Системный анализ предметной области 25

Инфологическая модель предметной области. 26

Нормальные формы ER-диаграмм 30

Даталогические модели 31

Получение реляционной схемы из ER-диаграммы 31

Физические модели 32

Проектирование реляционной базы данных 32

Универсальное отношение 32

Пример проектирования реляционной БД 38

**ВВЕДЕНИЕ**

В истории вычислительной техники можно проследить развитие двух основных областей ее использования. Первая область — применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную. Развитие этой области способствовало интенсификации методов численного решения сложных математических задач, появлению языков программирования, ориентированных на удобную запись численных алгоритмов, становлению обратной связи с разработчиками новых архитектур ЭВМ. Характерной особенностью данной области применения вычислительной техники является наличие сложных алгоритмов обработки, которые применяются к простым по структуре данным, объем которых сравнительно невелик.

Вторая область — это использование средств вычислительной техники в автоматических или автоматизированных информационных системах. Информационная система представляет собой программно-аппаратный комплекс, обеспечивающий выполнение следующих функций:

* надежное хранение информации в памяти компьютера;
* выполнение специфических для данного приложения преобразований информации и вычислений;
* предоставление пользователям удобного и легко осваиваемого интерфейса.

Классическими примерами информационных систем являются банковские системы, автоматизированные системы управления предприятиями, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т. д.

* + 1. **Основные понятия и определения**

Банк данных (БнД) — это система специальным образом организованных данных — баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

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

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

Программы, с помощью которых пользователи работают с базой данных, называются приложениями. В общем случае с одной базой данных могут работать множество различных приложений.

Например, если база данных моделирует некоторое предприятие, то для работы с ней может быть создано приложение, которое обслуживает:

* подсистему учета кадров,
* подсистему расчета заработной платы сотрудников,
* подсистему складского учета,
* планирование производственного процесса.

СУБД обеспечивает работу множества приложений с единой базой данных таким образом, чтобы каждое из них выполнялось корректно, но учитывало все изменения в базе данных, вносимые другими приложениями.

**2 Архитектура базы данных**

В процессе научных исследований, посвященных тому, как именно должна быть устроена СУБД, предлагались различные способы реализации. Самым жизнеспособным из них оказалась предложенная американским комитетом по стандартизации ANSI (American National Standards Institute) трехуровневая система организации БД, изображенная на рис. 1.



***Рис. 1 Архитектура базы данных.***

1. *Уровень внешних моделей* — самый верхний уровень, где каждая модель имеет свое «видение» данных. Этот уровень определяет точку зрения на БД отдельных приложений. Каждое приложение видит и обрабатывает только те данные, которые необходимы именно этому приложению. *Например,* система распределения работ использует сведения о квалификации сотрудника, но ее не интересуют сведения об окладе, домашнем адресе и телефоне сотрудника, и наоборот, именно эти сведения используются в подсистеме отдела кадров.
2. *Концептуальный уровень* — центральное управляющее звено. Здесь база данных представлена в наиболее общем виде, который объединяет данные, используемые всеми приложениями, работающими с данной базой данных. Фактически концептуальный уровень отражает обобщенную модель предметной области (объектов реального мира), для которой создавалась база данных. Как любая модель, концептуальная модель отражает только существенные, с точки зрения обработки, особенности объектов реального мира.
3. *Физический уровень* — собственно данные, расположенные в файлах или в страничных структурах, расположенных на внешних носителях информации.

Эта архитектура позволяет обеспечить логическую (между уровнями 1 и 2) и физическую (между уровнями 2 и 3) независимость при работе с данными. Логическая независимость предполагает возможность изменения одного приложения без корректировки других приложений, работающих с этой же базой данных. Физическая независимость предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с данной базой данных.

Выделение концептуального уровня позволило разработать аппарат централизованного управления базой данных.

**3 Процесс прохождения пользовательского запроса**

Рисунок 2 иллюстрирует взаимодействие пользователя, СУБД и ОС при обработке запроса на получение данных. Цифрами помечена последовательность взаимодействий:



***Рис. 2. Схема прохождения запроса к БД***

***Алгоритм прохождения запроса к БД:***

1. Пользователь посылает СУБД запрос на получение данных из БД.
2. Анализ прав пользователя и внешней модели данных, соответствующей данному пользователю, подтверждает или запрещает доступ данного пользователя к запрошенным данным.
3. В случае запрета на доступ к данным СУБД сообщает пользователю об этом (стрелка 12) и прекращает дальнейший процесс обработки данных, в противном случае СУБД определяет часть концептуальной модели, которая затрагивается запросом пользователя.
4. СУБД получает информацию о запрошенной части концептуальной модели.
5. СУБД запрашивает информацию о местоположении данных на физическом уровне (файлы или физические адреса).
6. В СУБД возвращается информация о местоположении данных в терминах операционной системы.
7. СУБД вежливо просит операционную систему предоставить необходимые данные, используя средства операционной системы.
8. Операционная система осуществляет перекачку информации из устройства хранения и пересылает ее в системный буфер.
9. Операционная система оповещает СУБД об окончании пересылки.
10. СУБД выбирает из доставленной информации, находящейся в системном буфере, только то, что нужно пользователю, и пересылает эти данные в рабочую область пользователя.

***БМД*** — это База Метаданных, в которой хранится вся информация:

* об используемых структурах данных;
* о логической организации данных;
* о правах доступа пользователей;
* о физическом расположении данных.

Для управления БМД существует специальное программное обеспечение администрирования баз данных, которое предназначено для корректного использования единого информационного пространства многими пользователями.

Всегда ли запрос проходит полный цикл? Конечно, нет. СУБД обладает достаточно развитым интеллектом, который позволяет ей не повторять бессмысленных действий. И поэтому, например, если этот же пользователь повторно обратится к СУБД с новым запросом, то для него уже не будут проверяться внешняя модель и права доступа, а если дальнейший анализ запроса покажет, что данные могут находиться в системном буфере, то СУБД осуществит только 11 и 12 шаги в обработке запроса.

Разумеется, механизм прохождения запроса в реальных СУБД гораздо сложнее, но и эта упрощенная схема показывает, насколько серьезными и сложными должны быть механизмы обработки запросов, поддерживаемые реальными СУБД.

**4 Классификация моделей данных**

Одними из основополагающих в концепции баз данных являются обобщенные категории «данные» и «модель данных».

Понятие ***«данные»*** в концепции баз данных — это набор конкретных значений, параметров, характеризующих объект, условие, ситуацию или любые другие факторы.

Примеры данных: Петров Николай Степанович, $30 и т. д.

Данные не обладают определенной структурой, данные становятся информацией тогда, когда пользователь задает им определенную структуру, то есть осознает их смысловое содержание. Поэтому центральным понятием в области баз данных является понятие модели. Не существует однозначного определения этого термина, у разных авторов эта абстракция определяется с некоторыми различиями, но, тем не менее, можно выделить нечто общее в этих определениях.

***Модель данных***— это сведения, содержащие не только данные, но и взаимосвязь между ними. Это некоторая абстракция, которая приложена к конкретным данным и позволяет пользователям и разработчикам трактовать их как информацию.



Рис. 3 Классификация моделей данных.

В соответствии с рассмотренной трехуровневой архитектурой мы сталкиваемся с понятием модели данных по отношению к каждому уровню:

* 1. Физическая модель данных оперирует категориями, касающимися организации внешней памяти и структур хранения, используемых в данной операционной среде.
  2. Концептуальная модель данных. Наибольший интерес вызывают модели данных, используемые на концептуальном уровне. По отношению к ним внешние модели называются подсхемами и используют те же абстрактные категории, что и концептуальные модели данных.

Инфологические модели данных используются на ранних стадиях проектирования для описания структур данных в процессе разработки приложения, а даталогические модели уже поддерживаются конкретной СУБД.

Документальные модели данных соответствуют представлению о слабоструктурированной информации, ориентированной в основном на свободные форматы документов, текстов на естественном языке.

Тезаурусные модели основаны на принципе организации словарей, содержат определенные языковые конструкции и принципы их взаимодействия в заданной грамматике. Эти модели эффективно используются в системах-переводчиках, особенно многоязыковых переводчиках. Принцип хранения информации в этих системах и подчиняется тезаурусным моделям.

Дескрипторные модели — самые простые из документальных моделей, они широко использовались на ранних стадиях использования документальных баз данных. В этих моделях каждому документу соответствовал дескриптор — описатель. Этот дескриптор имел жесткую структуру и описывал документ в соответствии с теми характеристиками, которые требуются для работы с документами в разрабатываемой документальной БД. Например, для БД, содержащей описание патентов, дескриптор содержал название области, к которой относился патент, номер патента, дату выдачи патента и еще ряд ключевых параметров, которые заполнялись для каждого патента. Обработка информации в таких базах данных велась исключительно по дескрипторам, то есть по тем параметрам, которые характеризовали патент, а не по самому тексту патента.

**5 Модели данных**

Модели данных отражают совокупность объектов реального мира в виде графа взаимосвязанных информационных объектов. В зависимости от типа графа выделяют: иерархическую и сетевую модели. Исторически эти модели появились раньше, и в настоящий момент они используются реже, чем более современная реляционная модель данных. Однако до сих пор существуют системы, работающие на основе этих моделей, а одна из концепций развития объектно-ориентированных баз данных предполагает объединение принципов сетевой модели с концепцией реляционной.

**6 Иерархическая модель данных**

Появление иерархической модели связано с тем, что в реальном мире очень многие связи соответствуют иерархии, когда один объект выступает как родительский, а с ним может быть связано множество подчиненных объектов. Иерархия проста и естественна в отображении взаимосвязи между классами объектов.

Основными информационными единицами в иерархической модели являются: база данных (БД), сегмент и поле. Поле данных определяется как минимальная, неделимая единица данных, доступная пользователю с помощью СУБД. Например, если в задачах требуется печатать в документах адрес клиента, но не требуется дополнительного анализа полного адреса, то есть города, улицы, дома, квартиры, то мы можем принять весь адрес за элемент данных, и он будет храниться полностью, а пользователь сможет получить его только как полную строку символов из БД.

Если же в задачах существует анализ частей, составляющих адрес, например города, где расположен клиент, то нам необходимо выделить город как отдельный элемент данных, только в этом случае пользователь может получить к нему доступ и выполнить, например, запрос на поиск всех клиентов, которые проживают в конкретном городе, например в Париже. Однако если пользователю понадобится и полный адрес клиента, то остальную информацию по адресу также необходимо хранить в отдельном поле, которое может быть названо, например, Сокращенный адрес. В этом случае для каждого клиента в БД хранится как Город, так и Сокращенный адрес.

Сегмент в терминологии Американской Ассоциации по базам данных называется записью, при этом в рамках иерархической модели определяются два понятия: тип сегмента или тип записи и экземпляр сегмента или экземпляр записи.

***Тип сегмента*** — это поименованная совокупность типов элементов данных, в него входящих. ***Экземпляр сегмента*** образуется из конкретных значений полей или элементов данных, в него входящих. Каждый тип сегмента в рамках иерархической модели образует некоторый набор однородных записей. Для возможности различия отдельных записей в данном наборе каждый тип сегмента должен иметь ключ или набор ключевых атрибутов (полей, элементов данных). ***Ключом*** называется набор элементов данных, однозначно идентифицирующих экземпляр сегмента. Например, рассматривая тип сегмента, описывающий сотрудника организации, мы должны выделить те характеристики сотрудника, которые могут его однозначно идентифицировать в рамках БД предприятия. Если предположить, что на предприятии могут работать однофамильцы, то, вероятно, наиболее надежным будет идентифицировать сотрудника по его табельному номеру. Однако если мы будем строить БД, содержащую описание множества граждан, например нашей страны, то, скорее всего, нам придется в качестве ключа выбрать совокупность полей, отражающих его паспортные данные.

В иерархической модели сегменты объединяются в ориентированный древовидный граф. При этом полагают, что направленные ребра графа отражают иерархические связи между сегментами: каждому экземпляру сегмента, стоящему выше по иерархии и соединенному с данным типом сегмента, соответствует несколько (множество) экземпляров данного (подчиненного) типа сегмента. Тип сегмента, находящийся на более высоком уровне иерархии, называется логически исходным по отношению к типам сегментов, соединенным с данным направленными иерархическими ребрами, которые в свою очередь называются логически подчиненными по отношению к этому типу сегмента. Иногда исходные сегменты называют сегментами-предками, а подчиненные сегменты называют сегментами-потомками.

На концептуальном уровне определяется понятие схемы БД в терминологии иерархической модели.

Схема иерархической БД представляет собой совокупность отдельных деревьев, каждое дерево в рамках модели называется физической базой данных. Каждая физическая БД удовлетворяет следующим иерархическим ограничениям:

* в каждой физической БД существует один корневой сегмент, то есть сегмент, у которого нет логически исходного (родительского) типа сегмента;
* каждый логически исходный сегмент может быть связан с произвольным числом логически подчиненных сегментов;
* каждый логически подчиненный сегмент может быть связан только с одним логически исходным (родительским) сегментом.

Между экземплярами сегментов также существуют иерархические связи. Рассмотрим, например, иерархический граф, представленный на рис. 5.



***Рис. 5 Пример структуры иерархического дерева***

Каждый тип сегмента может иметь множество соответствующих ему экземпляров. Между экземплярами сегментов также существуют иерархические связи.

На рис. 6 представлены 2 экземпляра иерархического дерева



***Рис. 6. Пример двух экземпляров данного дерева***

Экземпляры-потомки одного типа, связанные с одним экземпляром сегмента-предка, называют «близнецами». Так, для нашего примера экземпляры b1, b2 и b3 являются «близнецами», но экземпляр b4 подчинен другому экземпляру родительского сегмента, и он не является «близнецом» по отношению к экземплярам b1, b2 и b3. Набор всех экземпляров сегментов, подчиненных одному экземпляру корневого сегмента, называется физической записью. Количество экземпляров-потомков может быть разным для разных экземпляров родительских сегментов, поэтому в общем случае физические записи имеют разную длину. Так, используя принцип линейной записи иерархических графов, пример на рис 5 можно представить в виде двух записей:



Как видно из нашего примера, физические записи в иерархической модели различаются по длине и структуре.

***Рассмотрим пример иерархической БД.***

Наша организация занимается производством и продажей компьютеров, в рамках производства мы комплектуем компьютеры из готовых деталей по индивидуальным заказам. У нас существует несколько базовых моделей, которые мы продаем без предварительных заказов по наличию на складе. В организации существуют несколько филиалов (рис. 7) и несколько складов, на которых хранятся комплектующие. Нам необходимо вести учет продаваемой продукции.



***Рис.7 Физическая БД «Филиалы»***

Какие задачи нам надо решать в ходе разработки приложения?

* При приеме заказа мы должны выяснить, какую модель заказывает заказчик: типичную или индивидуальную комплектацию.
* Если заказывается типичная модель, то выясняется, какая модель и есть ли она в наличии, если модель есть, то надо уменьшить количество компьютеров данной модели в данном филиале на покупаемое количество. На этом будем считать заказ выполненным, однако при оформлении заказа может потребоваться задание полной спецификации покупаемого изделия.
* Если заказывается индивидуальная модель, то требуется описать весь состав новой модели.

Для того чтобы можно было бы принимать заказы на индивидуальные модели, нам понадобится информация о наличие конкретных деталей на складе, в этом случае нам необходимо второе дерево — Склады (см. рис. 8).



***Рис. 8. Физическая модель «Склады»***

**7 Сетевая модель данных**

Базовыми объектами модели являются:

* элемент данных;
* агрегат данных;
* запись;
* набор данных,

*Элемент данных* — то же, что и в иерархической модели, то есть минимальная информационная единица, доступная пользователю с использованием СУБД.

*Агрегат данных* соответствует следующему уровню обобщения в модели. В модели определены агрегаты двух типов: агрегат типа *вектор* и агрегат типа *повторяющаяся группа*.

Агрегат данных имеет имя, и в системе допустимо обращение к агрегату по имени. Агрегат типа вектор соответствует линейному набору элементов данных. Например, агрегат Адрес может быть представлен следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| **Адрес** | | | |
| Город | Улица | дом | квартира |

Агрегат типа повторяющаяся группа соответствует совокупности векторов данных. Например, агрегат Зарплата соответствует типу повторяющаяся группа с числом повторений 12.

|  |  |
| --- | --- |
| **Зарплата** | |
| Месяц | Сумма |
|  |  |

Записью называется совокупность агрегатов или элементов данных, моделирующая некоторый класс объектов реального мира. Понятие записи соответствует понятию «сегмент» в иерархической модели. Для записи, так же как и для сегмента, вводятся понятия типа записи и экземпляра записи.

Следующим базовым понятием в сетевой модели является понятие «Набор». Набором называется двухуровневый граф, связывающий отношением «один-ко многим» два типа записи.



Набор фактически отражает иерархическую связь между двумя типами записей. Родительский тип записи в данном наборе называется владельцем набора, а дочерний тип записи — членом того же набора.

Для любых двух типов записей может быть задано любое количество наборов, которые их связывают. Фактически наличие подобных возможностей позволяет промоделировать отношение «многие-ко-многим» между двумя объектами реального мира, что выгодно отличает сетевую модель от иерархической. В рамках набора возможен последовательный просмотр экземпляров членов набора, связанных с одним экземпляром владельца набора.

Между двумя типами записей может быть определено любое количество наборов: например, можно построить два взаимосвязанных набора. Существенным ограничением набора является то, что один и тот же тип записи не может быть одновременно владельцем и членом набора.

В качестве примера рассмотрим таблицу, на основе которой организуем два набора и определим связь между ними:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Преподаватель** | | **Группа** | | **День недели** | | № **пары** | | **Аудитория** | | |
| Иванов | | 4306 | | Понедельник | | 1 | | 22-13 | | |
| Иванов | | 4307 | | Понедельник | | 2 | | 22-13 | | |
| Карпова | | 4307 | | Вторник | | 2 | | 22-14 | | |
| Карпова | | 4309 | | Вторник | | 4 | | 22-14 | | |
| Карпова | | 84305 | | Вторник | | 1 | | 22-14 | | |
| Смирнов | | 4306 | | Вторник | | 3 | | 23-07 | | |
| Смирнов | | 4309 | | Вторник | | 4 | | 23-07 | | |
|  |  | |  | |  | |  | |  |  |



Экземпляров набора Ведет занятия будет 3 (по числу преподавателей), экземпляром набора Занимается у будет 4 (по числу групп).

В общем случае сетевая база данных представляет совокупность взаимосвязанных наборов, которые образуют на концептуальном уровне некоторый граф.

8 Реляционная модель данных

Р

***Рис 9. Основные понятия реляционной модели***

Реляционная модель является удобной и наиболее привычной фор­мой представления данных в виде таблицы. В отличие от иерархиче­ской и сетевой моделей, такой способ представления:

1) понятен пользователю-непрограммисту;

2) позволяет легко изменять схему – присоединять новые элементы данных и записи без изменения соот­ветствующих подсхем;

3) обеспечивает необходимую гибкость при обработке непредвиденных запросов.

Одним из основных преимуществ реляционной модели является ее однородность. Все данные рассматриваются как хранимые в таб­лицах, в которых каждая строка имеет один и тот же формат. Каждая строка в таблице представляет некоторый объект реального мира или соотношение между объектами.

К основными понятиям, с помощью которых определяется реля­ционная модель, относятся:

* домен,
* отношение,
* кортеж,
* кардинальность,
* атрибуты,
* степень,
* первичный ключ.

*Таблица*

|  |  |
| --- | --- |
| Домен | Совокупность допустимых значений |
| Кортеж | Таблица |
| Кардинальность | Количество строк в таблице |
| Атрибут | Поле, столбец таблицы |
| Степень отношения | Количество полей (столбцов) |
| Первичный ключ | Уникальный идентификатор |

*Домен —* это совокупность значений, из которой берутся значе­ния соответствующих атрибутов определенного отношения. С точки зрения программирования, домен — это тип данных, определяемый системой (стандартный) или пользователем.

*Первичный ключ —* это столбец или некоторое подмножество столбцов, которые уникально, т. е. единственным образом опреде­ляют строки. Первичный ключ, который включает более одного столбца, называется множественным, или комбинированным, или составным. Правило целостности объектов утверждает, что первич­ный ключ не может быть полностью или частично пустым. (Остальные ключи, которые можно также использовать в качест­ве первичных, называются потенциальными или *альтернативными* ключами).

*Внешний ключ* — это столбец или подмножество одной таблицы, который может служить в качестве первичного ключа для другой таблицы. Внешний ключ таблицы является ссылкой на первичный ключ другой таблицы. Правило ссылочной целостности гласит, что внешний ключ может быть либо пустым, либо соответствовать зна­чению первичного ключа, на который он ссылается. Внешние клю­чи являются неотъемлемой частью реляционной модели, поскольку реализуют связи между таблицами базы данных.

Внешний ключ, как и первичный, тоже может представ­лять собой комбинацию столбцов. На практике внешний ключ все­гда будет составным (состоящим из нескольких столбцов), если он ссылается на составной первичный ключ в другой таблице. Количество столбцов и их типы данных в первичном и внешнем ключах совпадают.

Если таблица связана с несколькими другими таблицами, она может иметь несколько внешних ключей.

Модель предъявляет к таблицам следующие требования:

1) данные в ячейках таблицы должны быть структурно неделимыми;

2) данные в одном столбце должны быть одного типа;

3) каждый столбец должен быть уникальным (недопустимо дуб­лирование столбцов);

4) столбцы размещаются в произвольном порядке;

5) строки размещаются в таблице также в произвольном по­рядке;

6) столбцы имеют уникальные наименования.

Концепция реляционной модели определяется следую­щими двенадцатью правилами.

1. *Правило информации.* Вся информация в базе данных должна быть предоставлена исключительно на логическом уровне и только одним способом — в виде значений, содержащихся в таблицах.

2. *Правило гарантированного доступа.* Логический доступ ко всем и каждому элементу данных (атомарному значению) в реляци­онной базе данных должен обеспечиваться путем использования комбинации имени таблицы, первичного ключа и имени столбца.

3. *Правило поддержки недействительных значений.* В реляцион­ной базе данных должна быть реализована поддержка недействи­тельных значений, которые отличаются от строки символов нулевой длины, строки пробельных символов, от нуля или любого другого числа и используются для представления отсутствующих данных не­зависимо от типа этих данных.

4. *Правило динамического каталога, основанного на реляционной модели.* Описание базы данных на логическом уровне должно быть представлено в том же виде, что и основные данные, чтобы пользо­ватели, обладающие соответствующими правами, могли работать с ним с помощью того же реляционного языка, который они приме­няют для работы с основными данными.

5. *Правило исчерпывающего подъязыка данных.* Реляционная сис­тема может поддерживать различные языки и режимы взаимодейст­вия с пользователем (например, режим вопросов и ответов). Однако должен существовать по крайней мере один язык, операторы кото­рого можно представить в виде строк символов в соответствии с не­которым четко определенным синтаксисом и который в полной мере поддерживает следующие элементы:

• определение данных;

• определение представлений;

• обработку данных (интерактивную и программную);

• условия целостности;

• идентификацию прав доступа;

• границы транзакций (начало, завершение и отмена).

6. *Правило обновления представлений.* Все представления, которые теоретически можно обновить, должны быть доступны для обновления,

7. *Правило добавления, обновления и удаления.* Возможность рабо­тать с отношением как с одним операндом должна существовать не

только при чтении данных, но и при добавлении, обновлении и уда­лении данных.

8. *Правило независимости физических данных.* Прикладные про­граммы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при любых изменениях способов хранения данных или методов доступа к ним.

9. *Правило независимости логических данных.* Прикладные про­граммы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при внесении в базовые таблицы любых изменений, которые теоретически позволяют сохранить не­тронутыми содержащиеся в этих таблицах данные.

10. *Правило независимости условий целостности.* Должна существо­вать возможность определять условия целостности, специфические для конкретной реляционной базы данных, на подъязыке реляционной базы данных и хранить их в каталоге, а не в прикладной программе.

11. *Правило независимости распространения.* Реляционная СУБД не должна зависеть от потребностей конкретного клиента.

12. *Правило единственности.* Если в реляционной системе есть низкоуровневый язык (обрабатывающий одну запись за один раз), то должна отсутствовать возможность использования его для того, чтобы обойти правила и условия целостности, выраженные на реля­ционном языке высокого уровня (обрабатывающем несколько запи­сей за один раз).

Правило 2 указывает на роль первичных ключей при поиске ин­формации в базе данных. Имя таблицы позволяет найти требуемую таблицу, имя столбца позволяет найти требуемый столбец, а первичный ключ позволяет найти строку, содержащую искомый элемент данных.

Правило 3 требует, чтобы отсутствующие данные можно было представить с помощью недействительных значений (NULL).

Правило 4 гласит, что реляционная база данных должна сама себя описывать. Другими словами, база данных должна содержать на­бор *системных таблиц,* описывающих структуру самой базы данных.

Правило 5 требует, чтобы СУБД использовала язык реляци­онной базы данных, например SQL. Такой язык должен поддержи­вать все основные функции СУБД — создание базы данных, чтение и ввод данных, реализацию защиты базы данных и т. д.

Правило 6 касается *представлений,* которые являются вир­туальными таблицами, позволяющими показывать различным поль­зователям различные фрагменты структуры базы данных. Это одно из правил, которые сложнее всего реализовать на практике.

Правило 7 акцентирует внимание на том, что базы данных по своей природе ориентированы на множества. Оно требует, чтобы операции добавления, удаления и обновления можно было выпол­нять над множествами строк. Это правило предназначено для того, чтобы запретить реализации, в которых поддерживаются только операции над одной строкой.

Правила 8 и 9 означают отделение пользователя и прикладной программы от низкоуровневой реализации базы дан­ных. Они утверждают, что конкретные способы реализации хране­ния или доступа, используемые в СУБД, и даже изменения структу­ры таблиц базы данных не должны влиять на возможность пользо­вателя работать с данными.

Правило 10 гласит, что язык базы данных должен поддерживать ограничительные условия, налагаемые на вводимые данные и дейст­вия, которые могут быть выполнены над данными.

Правило 11 гласит, что язык базы данных должен обеспечивать возможность работы с распределенными данными, расположенны­ми на других компьютерных системах.

Правило 12 предотвращает использование других возможностей для работы с базой данных, помимо языка базы дан­ных, поскольку это может нарушить ее целостность.

**9 Основы реляционной алгебры**

Реляционная алгебра состоит из двух групп по четыре оператора.

1. Традиционные операции над множествами: объединение, пересечение, разность и декартово произведение.

2. Специальные реляционные операции: выборка, проекция, со­единение, деление.

**10 Операции над множествами**

***Объединение***возвращает отношение, содержащее все кортежи, которые принадлежат либо одному из двух заданных отношений, либо им обоим.

***R1***

|  |  |  |  |
| --- | --- | --- | --- |
| *FIO* | *Year* | *Job* | *Chair* |
| Цветкова С.С. | 1960 | Доцент | 23 |
| Козлов К. К. | 1959 | Доцент | 23 |
| Петров П. П. | 1960 | Ст. преп. | 24 |
| Лютикова Л.Л. | 1977 | Ассистент | 24 |
| ***R2*** |  |  |  |
| *FIO* | *Year* | *Job* | *Chair* |
| Иванов И. И. | 1948 | Зав каф. | 22 |
| Сидоров С. С. | 1953 | Проф. | 22 |
| Гиацинтова Г. Г | 1945 | Проф. | 22 |
| Цветкова С. С. | 1960 | Доцент | 23 |
| Козлов К. К. | 1959 | Доцент | 23 |

**(ОБЪЕДИНЕНИЕ** **)**

|  |  |  |  |
| --- | --- | --- | --- |
| *FIO* | *Year* | *Job* | *Chair* |
| Иванов И. И. | 1948 | Зав каф. | 22 |
| Сидоров С. С. | 1953 | Проф. | 22 |
| Гиацинтова Г. Г | 1945 | Проф. | 22 |
| Цветкова С. С. | 1960 | Доцент | 23 |
| Козлов К. К. | 1959 | Доцент | 23 |
| Петров П. П. | 1960 | Ст. преп. | 24 |
| Лютикова Л.Л. | 1977 | Ассистент | 24 |

***Пересечение***возвращает отношение, содержащее все кортежи, которые принадлежат одновременно двум заданным отношениям.

**ПЕРЕСЕЧЕНИЕ**



|  |  |  |  |
| --- | --- | --- | --- |
| *FIO* | *Year* | *Job* | *Chair* |
| Цветкова С. С. | 1960 | Доцент | 23 |
| Козлов К. К. | 1959 | Доцент | 23 |

***Разность*** возвращает отношение, содержащее все кортежи, ко­торые принадлежат первому из двух заданных отношений и не при­надлежат второму.

**РАЗНОСТЬ**



|  |  |  |  |
| --- | --- | --- | --- |
| *FIO* | *Year* | *Job* | *Chair* |
| Иванов И. И. | 1948 | Зав каф. | 22 |
| Сидоров С. С. | 1953 | Проф. | 22 |
| Гиацинтова Г. Г | 1945 | Проф. | 22 |

|  |
| --- |
| *Job* |
| Зав каф. |
| Проф. |
| Ст. преп. |
| Доцент |
| Ассистент |

***Произведение***возвращает отношение, содержащее все возмож­ные кортежи, которые являются сочетанием двух кортежей, принад­лежащих соответственно двум заданным отношениям.

|  |
| --- |
| *Chair* |
| 22 |
| 23 |

***R3 R4***

**ПРОИЗВЕДЕНИЕ**



|  |  |
| --- | --- |
| *Job* | *Chair* |
| Зав каф. | 22 |
| Зав каф. | 23 |
| Проф. | 22 |
| Проф. | 23 |
| Ст. преп. | 22 |
| Ст. преп. | 23 |
| Доцент | 22 |
| Доцент | 23 |
| Ассистент | 22 |
| Ассистент | 23 |

**ПРИМЕР.**

Исходными являются три отношения R1 R2 и R3. Все они имеют эквивалентные схемы.

* R1= (ФИО, Паспорт, Школа);
* R2= (ФИО, Паспорт, Школа);
* R3= (ФИО, Паспорт, Школа).

Рассмотрим ситуацию поступления в высшие учебные заведения когда разрешены репетиционные вступительные экзамены, которые сдаются раньше основных вступительных экзаменов в вуз. Отношение R1 содержит список абитуриентов, сдававших репетиционные экзамены. Отношение, R2 содержит список абитуриентов, сдававших экзамены на общих условиях. Отношение R3 содержит список абитуриентов, принятых в институт. Будем считать, что при неудачной сдаче репетиционных экзаменов абитуриент мог делать вторую попытку и сдавать экзамены в общем потоке, поэтому некоторые абитуриенты могут присутствовать как в первом, так и во втором отношении.

Ответим на следующие вопросы:

1. Список абитуриентов, которые поступали два раза и не поступили в вуз. R = R1 https://refdb.ru/images/833/1665877/523cde0c.pngR2 \ R3
2. Список абитуриентов, которые поступили в вуз с первого раза. R = (R1 \ R2 https://refdb.ru/images/833/1665877/523cde0c.pngR3)https://refdb.ru/images/833/1665877/m468959a2.png (R2 \ R1https://refdb.ru/images/833/1665877/523cde0c.png R3)
3. Список абитуриентов, которые поступили в вуз только со второго раза.

Прежде всего это те абитуриенты, которые присутствуют в отношениях R1 и R2, потому что они поступали два раза, и присутствуют в отношении R3, потому что они поступили. R = R1 https://refdb.ru/images/833/1665877/523cde0c.pngR2 https://refdb.ru/images/833/1665877/523cde0c.pngR3

1. Список абитуриентов, которые поступали только один раз и не поступили.

Это прежде всего те абитуриенты; которые присутствуют в R1и не присутствуют в R2, и те, кто присутствуют в R2 и не присутствуют в R1. И разумеется, никто из них не присутствует в R3. R = (R1 \ R2) https://refdb.ru/images/833/1665877/m468959a2.png(R2 \ R1) \ R3

**11 Специальные операции**

***Выборка***возвращает отношение, содержащие все кортежи из за­данного отношения, которые удовлетворяют указанным условиям.

**ВЫБОРКА R=R1[Chair=22]**

|  |  |  |  |
| --- | --- | --- | --- |
| *FIO* | *Year* | *Job* | *Chair* |
| Иванов И. И. | 1948 | Зав каф. | 22 |
| Сидоров С. С. | 1953 | Проф. | 22 |
| Гиацинтова Г. Г | 1945 | Проф. | 22 |

**ВЫБОРКА R=R1[(Chair=24) AND (Year<1970)]**

|  |  |  |  |
| --- | --- | --- | --- |
| *FIO* | *Year* | *Job* | *Chair* |
| Петров П. П. | 1960 | Ст. преп. | 24 |

***Проекция***возвращает отношение, содержащее все кортежи (подкортежи) заданного отношения, которые остались в этом отноше­нии после исключения из него некоторых атрибутов.

***ПРОЕКЦИЯ R=R1[FIO, Job]***

|  |  |
| --- | --- |
| *FIO* | *Job* |
| Иванов И. И. | Зав каф. |
| Сидоров С. С. | Проф. |
| Гиацинтова Г. Г | Проф. |
| Цветкова С. С. | Доцент |
| Козлов К. К. | Доцент |
| Петров П. П. | Ст. преп. |
| Лютикова Л. Л. | Ассистент |

***Соединение*** возвращает отношение, содержащее все возможные кортежи, которые представляют собой комбинацию атрибутов двух кортежей, принадлежащих двум заданным, при условии, что в этих двух комбинированных кортежах присутствуют одинаковые значе­ния в одном или нескольких общих для исходных отношений атри­бутах (причем эти общие значения в результирующем кортеже появ­ляются один раз, а не дважды).

**СОЕДИНЕНИЕ**

**R3**

|  |  |
| --- | --- |
| *Job* | *Pay* |
| Зав каф. | 3000 |
| Проф. | 2500 |
| Доцент | 2000 |
| Ст. преп. | 1500 |
| Ассистент | 1200 |

**R=(R1[R1.Job=R3.Job]R3)**

|  |  |  |  |
| --- | --- | --- | --- |
| *FIO* | *Job* | *Chair* | *Pay* |
| Иванов И. И. | Зав каф. | 22 | 3000 |
| Сидоров С. С. | Проф. | 22 | 2500 |
| Гиацинтова Г. Г | Проф. | 22 | 2500 |
| Цветкова С. С. | Доцент | 23 | 2000 |
| Козлов К. К. | Доцент | 23 | 2000 |
| Петров П. П. | Ст. преп. | 24 | 1500 |
| Лютикова Л. Л. | Ассистент | 24 | 1200 |

***Деление*** для заданных двух унарных отношений и одного бинар­ного возвращает отношение, содержащее все кортежи из первого унарного отношения, которые содержатся также в бинарном отно­шении и соответствуют всем кортежам во втором унарном отноше­нии.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Job*** |  | ***Job*** | ***Chair*** |  | ***Chair*** |
| Зав. Каф. |  | Зав. Каф. | 22 |  | 22 |
| Проф. |  | Проф. | 22 |  |  |
| Доцент |  | Доцент | 22 |  |  |
| Ст. преп. |  | Зав. Каф. | 23 |  |  |
| Ассистент |  | Доцент | 23 |  | ***Chair*** |
|  |  | Ст. преп. | 24 |  | 22 |
|  |  | Ассистент | 24 |  | 23 |

**ДЕЛЕНИЕ**

|  |  |  |
| --- | --- | --- |
| ***Job*** |  | ***Job*** |
| Зав. Каф. |  | Зав. Каф. |
| Проф. |  | Доцент |
| Доцент |  |  |

**ПРИМЕР.**

Возьмем набор отношений, которые моделируют сдачу сессии студентами некоторого учебного заведения. Тема весьма понятная и привычная.

R1 = <ФИО, Дисциплина, Оценка>;

R2 = <ФИО, Группа>;

R3 = < Группы, Дисциплина>,

где R1 — информация о попытках (как успешных, так и неуспешных) сдачи экзаменов студентами; R2 — состав групп; R3 — список дисциплин, которые надо сдавать каждой группе. Домены для атрибутов формально задавать не будем, но, ориентируясь на здравый смысл, будем считать, что доменом для атрибута Дисциплина будет множество всех дисциплин, преподающихся в ВУЗе, доменом для атрибута Группа будет множество всех групп ВУЗа и т. д.

Покажем, каким образом можно получить из этих таблиц интересующие нас сведения с помощью реляционной алгебры. В каждом из приведенных примеров путем операции над исходными отношениями R1, R2, R3 формируются промежуточные отношения и результирующее отношение S, содержащее требуемую информацию.

* Список студентов, которые сдали экзамен по БД на «отлично». Результат может быть получен применением операции фильтрации по сложному условию к отношению R1 и последующим проектированием на атрибут «ФИО» (нам ведь требуется только список фамилий).

S = (R1[Оценка = 5^Дисциплина = «БД»])[ФИО];

* Список тех, кто должен был сдавать экзамен по БД, но пока еще не сдавал. Сначала найдем всех, кто должен был сдавать экзамен по БД. В отношении R3 находится список всех дисциплин, по которым каждая группа должна была сдавать экзамены, ограничим перечень дисциплин только «БД». Для того чтобы получить список студентов, нам надо соединить отношение R3 с отношением R2, в котором определен список студентов каждой группы.

R4 = (R2[R3.Номер группы=R2.НомерГруппы^R3.Дисциплина = «БД»] R3)[ФИО];

* Теперь получим список всех, кто сдавал экзамен по «БД» (нас пока не интересует результат сдачи, а интересует сам факт попытки сдачи, то есть присутствие в отношении R1):

R5 = (R1 [Дисциплина = «БД»])[ФИО];

и, наконец, результат — все, кто есть в первом множестве, но нет во втором:

S = R4 \ R5;

* Список круглых отличников. Строим список всех пар <студент—дисциплина>, которые в принципе должны быть сданы:

R4 = (R2[R2Группа = R3Группa] R3)[ФИО, Дисциплина];

Строим список пар <студент- дисциплина>, где получена оценка «отлично»:

R5 = (R1[Оценка = 5])[ФИО, Дисциплина];

Строим список студентов, что-либо не сдавших на отлично:

R6=(R4\R5)[ФИО].

Наконец, исключив последнее отношение из общего списка студентов, получаем результат:

R2[ФИО] \ R6

**12 Проектирование реляционных БД на основе принципов нормализации**

Проектирование реляционной БД - это набор взаимосвязанных отношений, в которых определены все атрибуты, заданы первичные ключи отношений и заданы еще некоторые дополнительные свойства отношений, которые относятся к принципам поддержки целостности. Этапы жизненного цикла базы данных изображены на рис.

  
Процесс проектирования БД представляет собой последовательность переходов от неформального словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели. В общем случае можно выделить следующие этапы проектирования:

1. Системный анализ и словесное описание информационных объектов предметной области.
2. Проектирование инфологической модели предметной области — частично формализованное описание объектов предметной области в терминах некоторой семантической модели.
3. Даталогическое или логическое проектирование БД, то есть описание БД в терминах принятой диалогической модели данных.

Физическое проектирование БД, то есть выбор эффективного размещения БД на внешних носителях для обеспечения наиболее эффективной работы приложения.

**13 Системный анализ предметной области**

С точки зрения проектирования БД в рамках системного анализа, необходимо осуществить первый этап, то есть провести подробное словесное описание объектов предметной области и реальных связей, которые присутствуют между описываемыми объектами. Желательно, чтобы данное описание позволяло корректно определить все взаимосвязи между объектами предметной области.

В общем случае существуют два подхода к выбору состава и структуры предметной области:

* *Функциональный подход —* он реализует принцип движения «от задач» и применяется тогда, когда заранее известны функции некоторой группы лиц и комплексов задач, для обслуживания информационных потребностей которых создается рассматриваемая БД. В этом случае мы можем четко выделить минимальный необходимый набор объектов предметной области, которые должны быть описаны.
* *Предметный подход —* когда информационные потребности будущих пользователей БД жестко не фиксируются. Они могут быть многоаспектными и весьма динамичными. Мы не можем точно выделить минимальный набор объектов предметной области, которые необходимо описывать. В описание предметной области в этом случае включаются такие объекты и взаимосвязи, которые наиболее характерны и наиболее существенны для нее. БД, конструируемая при этом, называется предметной, то есть она может быть использована при решении множества разнообразных, заранее не определенных задач. Конструирование предметной БД в некотором смысле кажется гораздо более заманчивым, однако трудность всеобщего охвата предметной области с невозможностью конкретизации потребностей пользователей может привести к избыточно сложной схеме БД, которая для конкретных задач будет неэффективной.

Чаще всего на практике рекомендуется использовать некоторый компромиссный вариант, который, с одной стороны, ориентирован на конкретные задачи или функциональные потребности пользователей, а с другой стороны, учитывает возможность наращивания новых приложений.

Системный анализ должен заканчиваться подробным описанием информации об объектах предметной области, которая требуется для решения конкретных задач, и которая должна храниться в БД, формулировкой конкретных задач, с кратким описанием алгоритмов их решения, описанием выходных документов, которые должны генерироваться в системе, описанием входных документов, которые служат основанием для заполнения данными БД.

**14 Инфологическая модель предметной области**

**Модель «Сущность - связь»**

Одной из наиболее популярных средств формализованного пред­ставления предметной области является модель «сущ­ность — связь» (ER-модели).

Семантическую основу ER-модели составляют следующие пред­положения:

• та часть реального мира (совокупность взаимосвязанных объ­ектов), сведения о которых должны быть помещены в базу данных, может быть *представлена* как совокупность *сущностей;*

• каждая сущность обладает характеристическими свойствами (атрибутами), отличающими ее от других сущностей и позво­ляющими ее *идентифицировать;*

• сущности можно классифицировать по типам сущностей: каждый экземпляр сущности (представляющий некоторый объект) может быть отнесен к классу — *типу сущностей,* каж­дый экземпляр которого обладает общими для них и отличающими их от сущностей других классов свойствами;

• систематизация представления, основанная на классах, в об­щем случае предполагает иерархическую зависимость типов: сущность типа является *подтипом* сущности *В,* если каждый экземпляр типа *A* является экземпляром сущности типа *В;*

• взаимосвязи объектов могут быть представлены как *связи* — сущности, которые служат для фиксирования (представле­ния) взаимозависимости двух или нескольких сущностей.

ER-модель должна опреде­лить объекты и взаимосвязи между ними, т. е. установить связи сле­дующих двух типов.

1.Связи между объектами и наборами характеристических свойств, и таким образом определить сами объекты.

2.Связи между объектами, задающие характер и функциональ­ную природу их взаимозависимости.

ER-моделирование предметной облас­ти базируется на использовании графических диаграмм.



***Рис. 10 Пример ER-диаграммы***

**Сущность.** Сущность, с помощью которой моделируется класс однотипных объектов, определяется как «предмет, который может быть четко идентифицирован». Сущность должна *определяться* таким *набором атрибутов,* который позволял бы различать отдельные экземпляры сущности. Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности. Например, для од­нозначной идентификации каждого экземпляра сущности «Сотруд­ник» вводится атрибут «Табельный номер», который вследствие сво­ей природы будет всегда иметь уникальное значение в рамках пред­приятия. Уникальным идентификатором сущности может являться атрибут, комбинация атрибутов, комбинация связей или комбинация связей и атрибутов, однозначно отличающая любой эк­земпляр сущности от других экземпляров сущности того же типа.

Сущность имеет *имя,* уникальное в пределах модели. При этом *имя сущности* — это *имя типа,* а не некоторого конкретного экземп­ляра.

Сущности подразделяются на *сильные* и *слабые.* Сущность явля­ется слабой, если ее существование зависит от другой сущности — сильной по отношению к ней. Например, сущность «Подчиненный» является слабой по отношению к сущности «Сотрудник»: если будет удалена запись, соответствующая некоторому сотруднику, имеюще­му подчиненных, то сведения о подчинении также должны быть удалены.

**Свойства.** Свойство может быть *множественным* или *единичным* — т. е. ат­рибут, задающий свойство, может одновременно иметь несколько значений или, соответственно, только одно. Например, сотрудник может иметь несколько специальностей, но единственное значе­ние — «Табельный номер».

Свойство может быть *простым* (не подлежащим дальнейшему делению с точки зрения прикладных задач) или *составным —* если его значение составляется из значений простых свойств. Например, свойство «Год рождения» является простым, а свойство «Адрес» — составным, так как включает значения простых свойств «Город», «Улица», «Дом».

В некоторых случаях полезно различать *базовые* и *производные* свойства. Например, «Поставщик» может иметь свойство «Общее количество поставляемых деталей», которое вычисляется суммиро­ванием количества деталей, поставляемых им по проекту.

Если наличие некоторого свойства для всех экземпляров сущно­сти не является обязательным, то такое свойство называется *услов­ным.* Например, не все сотрудники обладают свойством «ученая сте­пень».

Значения свойств могут быть *статическими* или *динамическими,* т. е. меняться со временем. Например, свойство «Табельный номер» является статическим, а «Адрес» — динамиче­ским. Свойство может быть *неопределенным,* если оно является ди­намическим, но его текущее значение еще не задано.

Свойство может рассматриваться как *ключевое,* если его значе­ние уникально и, возможно, в определенном контексте, однозначно идентифицирует сущность. Например, подчиненный некоторого определенного сотрудника.

**Связи.** Кроме связей между объектом и его свойствами, модель отражает связи между объектами разных классов. *Связь* определяется как «ассоциация, объединяющая несколь­ко сущностей». Эта ассоциация всегда может существовать между разными сущностями или между сущностью и ею же самой (рекур­сивная связь).

Сущности, объединяемые связью, называются *участниками. Степень связи* определяется количеством участников связи.

Если каждый экземпляр сущности участвует, по крайней мере, в одном экземпляре связи, то такое участие этой сущности называет­ся *полным* (или *обязательным);* в противном случае — *неполным* (или *необязательным).*

Количественный характер участия экземпляров сущностей задается *типом связи* (или *мощностью связи).* Возможны следующие типы: *«один к одному»* (1:1), *«один ко многим»* (1:М), *«многие к одному»* (М:1), *«многие ко многим»* (М:М).

Следует отметить, что инструмент связей — это средство пред­ставления *сложных объектов,* каждый из которых может рассматри­ваться как множество некоторым образом взаимосвязанных *про­стых объектов.* Деление на простые и сложные объекты, также как и характер взаимосвязи, является условным и определяется осо­бенностями анализа предметной области, т. е. в конце концов — характером использования данных о предметах в решаемых прикладных задачах. При этом с точки зрения, например, конструкто­ра, ДЕТАЛЬ является сложным объектом, а с точки зрения Постав­щика — простым.

Среди многих разновидностей взаимосвязей наиболее частыми являются такие отношения иерархического типа, как «часть — це­лое», «род — вид».

Отношение «часть — целое» используются для представления *составных объектов.* Например, МАШИНЫ состоят из УЗЛОВ, УЗЛЫ состоят из ДЕТАЛЕЙ. Здесь возможны как отношения *«один ко многим»,* так и *«многие ко многим».*

Отношение «род — вид» — для представления *обобщенных объек­тов.* Например, СОТРУДНИКИ подразделяются по профессии на КОНСТРУКТОРОВ, ПРОГРАММИСТОВ, РАБОЧИХ; ПРОГРАМ­МИСТЫ - на ПРИКЛАДНЫХ ПРОГРАММИСТОВ и СИСТЕМ­НЫХ ПРОГРАММИСТОВ. Иерархические отношения, и в частно­сти — «родовидовые», обычно используются как основа классифи­кации объектов по наборам характеристических признаков. Причем, «видовые» объекты *наследуют* свойства «родовых».

Другой широко используемой разновидностью взаимосвязи яв­ляется агрегирование — объединение простых объектов в сложный по принципу их принадлежности *агрегату* или их совместного уча­стия в некотором процессе. Агрегирование, рассматриваемое здесь как более общий случай иерархических отношений, объединяет объекты разной природы с единственным общим свойством «совме­стное участие». Агрегированные объекты именуются обычно отгла­гольными существительными, например, *«Состав»:* ПОДРАЗДЕЛЕ­НИЕ *состоит из* СОТРУДНИКОВ; *«Поставка»:* ПОСТАВЩИК *по­ставляет* ДЕТАЛИ.

**Супертипы и подтипы.** Сущность может быть расщеплена на два или более взаимоисключающих *подтипов,* каждый из которых включает общие атрибуты и/или связи. Эти общие атрибуты и/или связи явно определяются один раз на более высоком уровне. В под­типах могут определяться собственные атрибуты и/или связи. В принципе выделение подтипов может продолжаться на более низ­ких уровнях, но в большинстве случаев оказывается достаточно двух-трех уровней.

Сущность, на основе которой определяются подтипы, называет­ся *супертипом.* Подтипы должны образовывать полное множество, т. е. любой экземпляр супертипа должен относиться к некоторому подтипу. Иногда для полноты множества надо определять дополни­тельный подтип, например, ПРОЧИЕ.

Подтип наследует свойства и связи супертипа. Например, тип сущности ПРОГРАММИСТ является подтипом сущности СОТРУД­НИК. Программисты обладают всеми свойствами сотрудников и участвуют во всех связях, однако обратные утверждения неверны.

Тип сущности, его подтипы, подтипы этих подтипов и т. д. обра­зуют *иерархию типов сущности,* пример которой приведен на рис. 11.



***Рис. 11 Пример иерархии типов сущности***

**ER-диаграмма**

ER-диаграмма является очень удачным решени­ем мо­делирования. В ней сочетаются функциональный и информационный подхо­ды, что позволяет представлять как совокупность выполняемых функций, так и отношения между элементами системы, задаваемые структурами данных.

**Сущности.** Каждый тип сущности в ER-диаграммах представля­ется в виде прямоугольника, содержащего имя сущности. В качестве имени обычно используются существительные (или обороты суще­ствительного) в единственном числе. Для отражения сущностей сла­бых типов используются прямоугольники, стороны которых рису­ются двойными линиями. Например, в рассматриваемой далее ER-диаграмме, приведенной на рис. 5.4, ПОДЧИНЕННЫЙ — сущ­ность слабого типа.

**Свойства.** Свойства служат для уточнения, идентификации, ха­рактеристики или выражения состояния сущности или связи. Свой­ства отображаются в виде эллипсов, содержащих имя свойства. Эл­липс соединяется с соответствующей сущностью или связью линией.

Имена ключевых свойств подчеркиваются, например, свойство «Табельный номер» сущности СОТРУДНИК.

Контур эллипса рисуется двойной линией, если свойство мно­гозначное, например, свойство «Специальность» сущности СОТ­РУДНИК.

Контур эллипса рисуется штриховой линией, если свойство про­изводное, например, свойство «Кол-во» сущности ПОСТАВЩИК.

Эллипс соединяется пунктирной линией, если свойство условное, например, свойство «Иностранный язык» сущности СОТРУДНИК.

Если свойство составное, то составляющие его свойства отобра­жаются другими эллипсами, соединенными с эллипсом составного, например, свойство «Адрес» сущности СОТРУДНИК состоит из простых свойств «Город», «Улица», «Дом».

**Связи.** Связь — это графически изображаемая ассоциация, уста­навливаемая между сущностями. Каждый тип связи на ER-диаграмме отображается в виде ромба с именем связи внутри. В качестве имени обычно используются отглагольные существительные.

Стороны ромба рисуют двойными линиями, если это связь сущ­ности слабого типа с сущностью, от которой она зависит. Например, связь «Подчинение», связывающая сущность слабого типа ПОДЧИ­НЕННЫЙ с сущностью СОТРУДНИК, от которой она зависит.

Участники связи соединены со связью линиями. Двойная линия обозначает полное участие сущности в связи с данной стороны. Напри­мер, связь «Подчинение» со стороны сущности ПОДЧИНЕННЫЙ.

Связь может быть модифицирована указанием роли. Напри­мер, для рекурсивной связи «Состав» указаны роли: «Деталь *состо­ит из* ...» и «Деталь входит *в состав* ...».

Тип связи указывается индексами «1» или «М» над соответст­вующей линией. Например, связь «Руководство» имеет тип «один ко многим»: один сотрудник может руководить многими проектами; связь «Участие» имеет тип «многие ко многим»: один сотрудник мо­жет участвовать во многих проектах, и в проекте могут участвовать многие сотрудники.

**15 Нормальные формы ER-диаграмм**

В *первой нормальной форме* ER-диаграммы устраняются повто­ряющиеся атрибуты или группы атрибутов, т. е. производится выяв­ление неявных сущностей, «замаскированных» под атрибуты.

Во *второй нормальной форме* устраняются атрибуты, зависящие только от части уникального идентификатора. Эта часть уникально­го идентификатора определяет отдельную сущность.

В *третьей нормальной форме* устраняются атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. Эти атри­буты являются основой отдельной сущности.

На рис. 12 представлена ER-диаграмма рис. 10 в третьей нор­мальной форме.



***Рис. 12 Пример ER-диаграммы в третьей нормальной форме***

**16 Даталогические модели**

Задачей следующей стадии проектирования системы базы данных является выбор подходящей СУБД и отображение в ее среду (струк­туру данных) спецификаций инфологической модели предметной области. Результатом даталогического проектирования базы данных является концептуальная схема базы данных, включающая определение всех информационных элементов (единиц) и связей, в том числе задание типов, характеристик и имен.

Даталогическое проектирование оперирует логическими понятиями, связанными со структурой базы данных, но особенности представления данных, правила и языки агрегирования и манипулирования данными имеют опреде­ляющее влияние. Не все виды связей, например, «многие ко многим», могут быть непосредственно отображены в логической модели.

Может быть много вариантов отображения инфоло­гической модели предметной области в даталогическую модель базы. Следует учитывать влияние двух факторов.

Во-первых, связи предметной области могут отображаться двумя путями: как декларативным — в логической схеме, так и процедур­ным — отработкой связей через программные модули, обрабаты­вающие (связывающие) соответствующие хранимые данные.

Во-вторых, существенным фактором может оказаться характер обработки информации. Например, частые обращения к совместно обрабатываемым данным, очевидно, предполагают их совместное хранение, а данные (особенно большого объема), к которым обра­щаются редко, целесообразно хранить отдельно от часто исполь­зуемых.

Рассмотрим по шагам общий подход к построению реляцион­ной базы данных на основе инфологической модели, представлен­ной ER-диаграммой.

**17 Получение реляционной схемы из ER-диаграммы**

1.Каждая простая сущность превращается в таблицу (отношение). Имя сущности становится именем таблицы.

2*.*Каждый атрибут становится возможным столбцом с тем же именем. Столбцы, соответствующие необязательным атрибутам, мо­гут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, — не могут. Если атрибут является множе­ственным, то для него строится отдельное отношение.

3.Компоненты уникального идентификатора сущности превра­щаются в первичный ключ. Если имеется несколько возможных уникальных идентификаторов, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, то к чис­лу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

4.Связи «многие к одному» и «один к одному» становятся внешними ключами. Т.е. создается копия уникального идентифика­тора с конца связи «один», и соответствующие столбцы составляют внешний ключ.

5. Индексы создаются для первичного ключа (уникальный ин­декс), а также внешних ключей и тех атрибутов, которые будут час­то использоваться в запросах.

6. Если в концептуальной схеме присутствуют подтипы, то воз­можны два варианта.

Все подтипы хранятся в одной таблице, которая создается для самого внешнего супертипа, а для подтипов создаются представле­ния. В таблицу добавляется по крайней мере один столбец, содер­жащий код ТИПА, и он становится частью первичного ключа.

Во втором случае для каждого подтипа создается отдельная таб­лица (для более нижних — представления) и для каждого подтипа первого уровня супертип воссоздается следующим образом: из всех таблиц подтипов выбираются общие столбцы — столбцы супертипа.

7. Если остающиеся внешние ключи все принадлежат одному до­мену, т. е. имеют общий формат, то создаются два столбца: иденти­фикатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей. Столбец идентификатора сущности используется для хранения значений уникального иденти­фикатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, соз­даются явные столбцы внешних ключей.

**18 Физические модели**

Стадия физического проектирования базы данных в общем случае включает:

•выбор способа организации базы данных;

•разработку спецификации внутренней схемы средствами мо­дели данных ее внутреннего уровня;

•описание отображения концептуальной схемы во внутреннюю.

Важно заметить, что в отличие от ранних СУБД, многие совре­менные системы не предоставляют разработчику какого-либо выбо­ра на этой стадии. Реально к вопросам проектирования физической модели можно отнести выбор схемы размещения данных.

Способ хранения базы данных определяется механизмами СУБД автоматически «по умолчанию» на основе спецификаций концептуальной схемы базы данных, и внутренняя схема в явном виде в таких системах не используется.

Внешние схемы базы данных обыч­но конструируются на стадии разработки приложений.

**19 Проектирование реляционной базы данных**

Задача проектирования БД для предметной области состоит в том, чтобы обеспечить поддержку не только любых ныне используемых, но и будущих приложений. Таким образом, БД создают основу для обработки неформализованных, изменяющихся и неизвестных за­просов и приложений, для которых невозможно заранее определить требования к данным. Это позволяет в дальнейшем строить на основе предметных БД достаточно стабильные информа­ционные системы, т. е. системы, в которых большинство изменений можно осуществить без переписывания старых приложений.

Задача проектирования БД — это сокращение из­быточности хранимых данных, а следовательно, экономия объема используемой памяти, уменьшение затрат на многократные опера­ции обновления избыточных копий и устранение возможности воз­никновения противоречий из-за хранения в разных местах сведений об одном и том же объекте. Такой проект БД можно создать, используя методологию нормализации отношений.

**20 Универсальное отношение**

Рассмотрим задачу проектирования БД на базе сводной таблицы, пример которой приведен на рис. 6.1. Предложенная таблица отра­жает результаты сдачи сессии (шкала оценок: 0 — незачет; 1 — за­чет; 2, 3, 4, 5 — экзаменационная оценка).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ФИО**  **студента** | **Семестр** | **Дисциплина** | **Форма отчетности** | **Оценка** | **Количество часов** | **ФИО**  **преподавателя** |
| Иванов В. П. | 1 | Английский язык | зачет | 1 | 60 | Цветкова А.Ю. |
| Математический анализ | зачет | 1 | 28 | Рыбин К. К. |
| Математический анализ | экзамен | 5 | 32 | Раков И.И. |
| Программирование | зачет | 1 | 36 | Незабудкина З.П. |
| Программирование | экзамен | 5 | 32 | Зайчиков А. А. |
| Линейная алгебра | зачет | 1 | 24 | Волков Г. И. |
| Линейная алгебра | экзамен | *А* | 28 | Волков Г.И. |
| История Отечества | экзамен | 5 | 24 | Москвин А. П. |
| Петрова А Л. | 1 | Английский язык | зачет | 1 | 60 | Цветкова А.Ю. |
| Математический анализ | зачет | 1 | 28 | Рыбин К. К. |
| Математический анализ | экзамен | 3 | 32 | Раков И.И. |
| Программирование | зачет | 1 | 36 | Незабудкина З.П. |
| Программирование | экзамен | 4 | 32 | Зайчиков А. А. |
| Линейная алгебра | зачет | 1 | 24 | Волков Г.И. |
| Линейная алгебра | экзамен | 4 | 28 | Волков Г. И. |
| История Отечества | экзамен | 5 | 24 | Москвин А. П. |
| Сидоров К.К. | 3 | Английский язык | зачет | 1 | 60 | Цветкова А.Ю. |
| Математический анализ | зачет | 1 | 20 | Карпов К.Ю. |
| Математический анализ | экзамен | 5 | 28 | Раков И.И. |
| Алгоритмы и структу­ры данных | экзамен | 5 | 32 | Зайчиков А. А. |
| Теория вероятностей и математическая статистика | экзамен | 4 | 32 | Соболев И.Г. |
| Операционные систе­мы, среды и оболочки | зачет | 1 | 36 | Незабудкина З.П. |
| Операционные систе­мы, среды и оболочки | экзамен | 4 | 32 | Незабудкина З.П. |
| Экономическая теория | зачет | 1 | 24 | Лабиринтов Е.Н. |

***Рис. 13. Исходные данные для создания БД «Сессия»***

Этот вариант Таблицы «Сессия» не является отношением, так как большинство ее столбцов не атомарны. Атомарными являются лишь значения столбцов «ФИО студента», «Семестр». Остальные столбцы таблицы — множественные.

Для преобразования данных в отношение необходимо реконст­руировать таблицу с помощью простого процесса встав­ки, результат которой показан на рис. 14.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ФИО**  **студента** | **Семестр** | **Дисциплина** | **Форма отчетности** | **Оценка** | **Количество часов** | **ФИО**  **преподавателя** |
| Иванов В. П. | 1 | Английский язык | зачет | 1 | 60 | Цветкова А.Ю. |
| Математический анализ | зачет | 1 | 28 | Рыбин К. К. |
| Математический анализ | экзамен | 5 | 32 | Раков И.И. |
| Программирование | зачет | 1 | 36 | Незабудкина З.П. |
| Программирование | экзамен | 5 | 32 | Зайчиков А. А. |
| Линейная алгебра | зачет | 1 | 24 | Волков Г. И. |
| Линейная алгебра | экзамен | *А* | 28 | Волков Г.И. |
| История Отечества | экзамен | 5 | 24 | Москвин А. П. |
| Петрова А Л. | 1 | Английский язык | зачет | 1 | 60 | Цветкова А.Ю. |
| Математический анализ | зачет | 1 | 28 | Рыбин К. К. |
| Математический анализ | экзамен | 3 | 32 | Раков И.И. |
| Программирование | зачет | 1 | 36 | Незабудкина З.П. |
| Программирование | экзамен | 4 | 32 | Зайчиков А. А. |
| Линейная алгебра | зачет | 1 | 24 | Волков Г.И. |
| Линейная алгебра | экзамен | 4 | 28 | Волков Г. И. |
| История Отечества | экзамен | 5 | 24 | Москвин А. П. |
| Сидоров К.К. | 3 | Английский язык | зачет | 1 | 60 | Цветкова А.Ю. |
| Математический анализ | зачет | 1 | 20 | Карпов К.Ю. |
| Математический анализ | экзамен | 5 | 28 | Раков И.И. |
| Алгоритмы и структу­ры данных | экзамен | 5 | 32 | Зайчиков А. А. |
| Теория вероятностей и математическая статистика | экзамен | 4 | 32 | Соболев И.Г. |
| Операционные систе­мы, среды и оболочки | зачет | 1 | 36 | Незабудкина З.П. |
| Операционные систе­мы, среды и оболочки | экзамен | 4 | 32 | Незабудкина З.П. |
| Экономическая теория | зачет | 1 | 24 | Лабиринтов Е.Н. |

***Рис. 14 Преобразование данных***

Такое преобразование приводит к возникнове­нию большого объема избыточных данных.

Таблица на рис. 14 представляет собой корректное отношение. Такое отношение называют универсальным отношением проекти­руемой БД. В одно универсальное отношение включаются все пред­ставляющие интерес атрибуты, и оно может содержать все данные, которые предполагается размещать в БД в будущем. При проекти­ровании некоторых БД универсальное отношение может использо­ваться в качестве отправной точки.

Однако при использовании универсального отношения возни­кают, по крайней мере, две проблемы.

1. *Избыточность данных.* Значения столбцов таблицы много­кратно повторяются. Повторяются также и некоторые наборы зна­чений столбцов, например, данные о дисциплине.

2. *Потенциальная противоречивость.* Если при вводе данных, на­пример, количества часов для дисциплины «Английский язык», была допущена ошибка, то для ее исправления необходимо найти все строки, содержащие сведения об этой дисциплине, и во всех этих строках произвести изменения. Более того, при заполнении та­кой таблицы могут быть использованы различные формы записи одного и того же значения, например: «Англ. язык» и «Английский язык», «Мат. анализ» и «Математический анализ».

Решение этих проблем состоит в разделении данных и связей, т. е. в выделении в отдельные таблицы сведений о студентах, препо­давателях, дисциплинах и результатах сдачи экзаменов (рис. 15).

**Студенты Преподаватели Дисциплины**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | **ФИО студента** |  | № | ФИО преподавателя |  | № | Дисциплина |
| 1 | Иванов В. П. |  | 1 | Волков Г. И. |  | 1 | Алгоритмы и структуры данных |
| 2 | Петрова А.П. |  | 2 | Зайчиков А. А. |  | 2 | Английский язык |
| 3 | Сидоров **К.** К. |  | 3 | Карпов К. Ю. |  | 3 | История Отечества |
|  |  |  | 4 | Лабиринтов Е. Н. |  | 4 | Линейная алгебра |
|  |  |  | 5 | Москвин А. П. |  | 5 | Математический анализ |
|  |  |  | 6 | Незабудкина З. П. |  | 6 | Операционные систе­мы, среды и оболочки |
|  |  |  | 7 | Пиков И. И. |  | 7 | Программирование |
|  |  |  | 8 | Рыбин К. К. |  | 8 | Теория вероятностей и математическая статистика |
|  |  |  | 9 | Соболев И. Г. |  | 9 | Экономическая теория |
|  |  |  | 10 | Цветкова А. Ю. |  |  |  |

***Рис. 15 Разделение данных и связей***

Заменим в таблицах «Результаты сессии» и «Учебный план» конкретные значения на их номера в других таблицах и получим, помимо значительного упрощения процедуры модификации тексто­вых значений, дополнительные возможности по включению строк в таблицы «Студенты», «Преподаватели», «Дисциплины», что значи­тельно расширяет возможности БД.

**Учебный план**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Дисциплина | Семестр | Кол-во  часов | Форма отчетности | Преподаватель |
| 1 | 2 | 1 | 60 | зачет | 10 |
| 2 | 3 | 1 | 24 | экзамен | 5 |
| 3 | 4 | 1 | 24 | зачет | 1 |
| 4 | 4 | 1 | 28 | экзамен | 1 |
| 5 | 5 | 1 | 28 | зачет | 8 |
| 6 | 5 | 1 | 32 | экзамен | 7 |
| 7 | 7 | 1 | 36 | зачет | 6 |
| 8 | 7 | 1 | 32 | экзамен | 2 |
| 9 | 2 | 3 | 60 | зачет | 10 |
| 10 | 5 | 3 | 20 | зачет | 3 |
| 11 | 5 | 3 | 28 | экзамен | 7 |
| 12 | 1 | 3 | 32 | экзамен | 2 |
| 13 | 8 | 3 | 32 | экзамен | 9 |
| 14 | 6 | 3 | 36 | зачет | 6 |
| 15 | 6 | 3 | 32 | экзамен | 6 |
| 16 | 9 | 3 | 24 | зачет | 4 |

**Результаты сессии**

|  |  |  |
| --- | --- | --- |
| Студент | Учебный  план | Оценка |
| 1 | 1 | 1 |
| 1 | 2 | 5 |
| 1 | 3 | 1 |
| 1 | 4 | 4 |
| 1 | 5 | 1 |
| 1 | 6 | 5 |
| 1 | 7 | 1 |
| 1 | 8 | 5 |
| 2 | 1 | 1 |
| 2 | 2 | 5 |
| 2 | 3 | 1 |
| 2 | 4 | 4 |
| 2 | 5 | 1 |
| 2 | 6 | 3 |
| 2 | 7 | 1 |
| 2 | 8 | 4 |

***Рис. 16. Разделение универсального отношения «Сессия»***

Теперь при изменении названия «Математический анализ» на «Мат. анализ» исправляется единственное значение в таблице «Дис­циплины». И даже если оно вводится с ошибкой, то это не может повлиять на связь между дисциплиной, преподавателем и студентом (в связующей таблице «Результаты сессии» используются номера дисциплин учебного плана, а не их названия).

**Функциональная и многозначная зависимости**

*Функциональная зависимость,* по сути, является связью типа «многие к одному» между множествами атрибутов (столбцов) рас­сматриваемого отношения.

Например, в таблице «Учебный план» столбцы *Дисциплина, Семестр* и *Форма отчетности* функционально зависят от ключа *№ (порядковый номер)* в учебном плане, а в таблице «Ре­зультаты сессии» столбец *Оценка* функционально зависит от состав­ного ключа (*Студент, Учебный план).*

*Многозначная зависимость.* Говорят, что один атрибут таблицы многозначно определяет другой атрибут той же таблицы, если для каждого значения первого атрибута существует хорошо определен­ное множество соответствующих значений второго атрибута.

**Нормальные формы**

*Таблица находится в первой нормальной форме* ***(1НФ)*** тогда **и** только тогда, когда в любом допустимом значении этой таблицы ка­ждая ее строка содержит только одно значение для каждого атрибу­та (столбца).

Из таблиц, рассмотренных ранее, не удовлетворяет этим требо­ваниям (т. е. не находится в 1НФ) только таблица на рис.13.

*Таблица находится во второй нормальной форме (2НФ),* если она удовлетворяет определению 1НФ и все ее атрибуты (столбцы), не входящие в первичный ключ, связаны полной функциональной за­висимостью с первичным ключом.

Не удовлетворяют этим требованиям таблицы, представленные на рис. 13 и на рис. 14. Таблица 14 имеет составной первичный ключ (*ФИО студента, Семестр, Дисциплина, Форма отчетности)* и содержит множество неключевых атрибутов (*Оценка, Количество ча­сов, ФИО преподавателя),* зависящих лишь от той или иной части первичного ключа. Так, атрибуты *Количество часов* и *ФИО* препода­вателя зависят только от атрибутов *Семестр, Дисциплина, Форма от­четности.* Следовательно, эти атрибуты не связаны с первичным ключом полной функциональной зависимостью.

Ко второй нормальной форме приведены все таблицы рис. 15.

*Таблица находится в третьей нормальной форме (ЗНФ),* если она удовлетворяет определению 2НФ и ни один из ее неключевых атри­бутов не связан функциональной зависимостью с любым другим не­ключевым атрибутом.

Таблица «Учебный план» (рис. 15), очевидно, не находилась бы в третьей нормальной форме, если включала бы в себя столбец *Должность преподавателя.* В этом случае необходимо было бы про­вести декомпозицию таблицы «Учебный план» и в результате полу­чить дополнительную таблицу «Кадровый состав» с атрибутами: *№, ФИО преподавателя, Должность преподавателя.*

Следует отметить, что в таблице «Учебный план» на самом деле существует функциональная зависимость между атрибутами *Количе­ство часов* и *ФИО преподавателя,* с одной стороны, и совокупно­стью атрибутов *Семестр, Дисциплина* и *Форма отчетности* — с дру­гой. Однако тройка атрибутов *{Семестр, Дисциплина* и *Форма от­четности)* в свою очередь может выступать в качестве первичного ключа, который представлен в таблице атрибутом *Порядковый номер.* Чтобы избегать в процессе нормализации подобных противоречий, Кодд и Бойс обосновали и предложили более строгое определение для ЗНФ, которое учитывает, что в таблице может быть несколько первичных ключей.

Таблица находится в *нормальной форме Бойса-Кодда (НФБК)* тог­да и только тогда, когда любая функциональная зависимость между ее атрибутами сводится к полной функциональной зависимости от *возможного* первичного ключа.

В соответствии с этой формулировкой таблица «Учебный план» находится в НФБК или в ЗНФ.

В следующих нормальных формах (4НФ и 5НФ) учитываются не только функциональные, но и многозначные зависимости между атрибутами. Для того чтобы привести определения этих нормальных форм, введем понятие полной декомпозиции таблицы.

*Полной декомпозицией таблицы* называют такую совокупность произвольного числа ее проекций, соединение которых полностью совпадает с содержимым таблицы.

Таблица находится в *пятой нормальной форме (ЗНФ)* тогда и только тогда, когда в каждой ее полной декомпозиции все проекции содержат возможный ключ. Таблица, не имеющая ни одной полной декомпозиции, также находится в 5НФ.

*Четвертая нормальная форма (4НФ)* является частным случаем 5НФ, когда полная декомпозиция должна быть соединением ровно двух проекций. На практике непросто подобрать реальную таблицу, которая находилась бы в 4НФ, но не была бы в 5НФ.

**Процедура нормализации**

В соответствии с определениями нормальных форм можно дать и другое определение нормализации: *нормализация* — это процесс по­следовательной замены таблицы ее полными декомпозициями до тех пор, пока все они не будут находиться в 5НФ. Однако оказыва­ется, что достаточно привести таблицы к НФБК и с большой гаран­тией считать, что они находятся в 5НФ (это утверждение нуждается в проверке, но пока не существует эффективного алгоритма такойпроверки).

Рассмотрим процедуру приведения таблиц к НФБК.

К, где К — первичный ключ, а А — некото­рый атрибут. Принцип «один факт в одном месте» говорит о том, что не должно существовать в рамках таблицы никаких других функциональных зависимостей. Цель нормализации и состоит в удалении этих «других» функциональных зависимостей. Рассмотрим два возможных случая.∀Такая процедура основывается на том, что единственными функциональными зависимостями в любой таблице должны быть зависимости вида А

1. Таблица имеет составной первичный ключ вида, скажем, (К1, К2), и включает также атрибут А, который функционально зависит от части этого ключа (например, от К2), но не от полного ключа. В этом случае рекомендуется сформировать другую таблицу, содер­жащую атрибуты К2 и А (первичный ключ — К2), и удалить атрибут А из первоначальной таблицы:

2. Таблица имеет первичный (возможный) ключ К, атрибут А1, который не является возможным ключом, но функционально зави­сит от К, и другой неключевой атрибут А2, который функционально зависит от А1. Решение здесь, по существу, то же самое, что и преж­де — формируется другая таблица, содержащая атрибуты А1 и А2, с первичным ключом А1, а атрибут А2 удаляется из первоначальной таблицы.

**21 Пример проектирования реляционной БД**

Рассмотрим следующую задачу: пусть необходимо обеспечить сбор и обработку данных по результатам сдачи экзаменов и зачетов сту­дентами факультета. Организация данных должна поддерживать:

• выполнение текущего учебного плана;

• формирование ведомостей по отдельным дисциплинам для групп студентов;

• формирование листов зачетных книжек студентов;

• формирование сводной ведомости курса;

• расчет среднего балла по дисциплинам и т. п.

Приведем этапы построения инфологической и даталогической моделей (ER-диаграммы и реляционной схемы) для решения такой задачи.

**Построение ER-диаграммы**

Представим предметную область как взаимодействие двух сущно­стей — «Дисциплина учебного плана» и «Студент»: каждый студент сдает экзамен или зачет по некоторой дисциплине учебного плана и получает оценку, которая должна быть зафиксирована в модели данных.

«Дисциплина учебного плана» с точки зрения решаемой задачи должна быть представлена группой свойств, позволяющих характе­ризовать дисциплину в рамках каждого отдельного семестра: наиме­нование дисциплины, семестр, количество часов, форма отчетности (экзамен или зачет) и данные о преподавателе, читающем дисципли­ну. Необходимость задания таких свойств обусловлена, с одной сто­роны, задачей организации хранения результатов сдачи экзаменов и зачетов (наименование дисциплины, семестр и форма отчетности), и с другой стороны — задачей формирования листов зачетных книжек (количество часов и данные о преподавателе). Отдельный экземпляр такой сущности однозначно идентифицируется тройкой свойств — наименование дисциплины, семестр и форма отчетности.

Сущность «Студент» для обеспечения выполнения объявленных функций должна характеризоваться следующими свойствами: фами­лия, имя, отчество и номер группы. Однако следует отметить, что даже набор значений всех этих свойств не может однозначно характе­ризовать экземпляр сущности, так как можно предполагать наличие в одной группе полных тезок. Таким образом, для идентификации от­дельного экземпляра сущности необходимо ввести дополнительное (ключевое) свойство — идентификационный номер студента.

Определим для сущности «Студент» еще два дополнительных свойства, которые не будут непосредственно обеспечивать решение поставленной задачи, но могут служить для реализации дополни­тельных (сервисных) функций (например, организации почтовой или телефонной связи): домашний адрес и номер телефона. Свойст­во «Домашний адрес», являясь по сути составным, будет на самом деле рассматриваться в контексте решаемых задач как простое, а свойство «Номер телефона» — как условное.

Взаимодействие сущностей реализуется связью «Сводная ведо­мость», т. е. Студент сдает экзамен (зачет) по Дисциплине учебного плана. Мощность связи — «многие ко многим» (М: М). Для иденти­фикации связи отдельных экземпляров сущностей в этом случае не­обходимо наличие у связи следующих дополнительных свойств: оценка и дата сдачи экзамена (зачета).

ER-диаграмма рассматриваемой задачи представлена на рис 17



***Рис. 17. ER-диаграмма рассматриваемой задачи***

Построенная ER-диаграмма находится в *первой нормальной фор­ме,* так как сущности не имеют повторяющихся групп свойств. Однако при рассмотрении свойств сущности «Дисциплина учебного плана» можно заметить, что свойство «Преподаватель» за­висит только от части ключевых свойств, — а именно от свойств «Наименование дисциплины» и, возможно, «Форма отчетности». Следовательно, для того чтобы привести ER-диаграмму ко второй нормальной форме, необходимо выделить свойство «Преподава­тель» в отдельную сущность.

Новая сущность «Преподаватель» характеризуется группой ос­новных свойств — фамилия, имя, отчество, и группой дополнитель­ных свойств — кафедра, должность, домашний адрес и телефон. Так же, как и для сущности «Студент», для сущности «Преподаватель» необходимо ввести дополнительное (ключевое) свойство — иденти­фикационный номер преподавателя.

  
Взаимодействие новой сущности с сущностью «Дисциплина учебного плана» осуществляется посредством новой связи «Читает». Мощность связи — «Многие к одному» (М:1), т. е. несколько дис­циплин учебного плана может читать один преподаватель.

***Рис. 18. Нормализованная ER-диаграмма***

Измененная ER-диаграмма представлена на рис. 18. Новый ва­риант ER-диаграммы находится *в третьей нормальной форме,* так как сущности не имеют свойств, зависящих от неключевых.

**Построение реляционной схемы**

Следующий этап проектирования — построение даталогической мо­дели. В рассматриваемом случае задача этого этапа — преобразова­ние ER-диаграммы в реляционную схему.

Реляционный подход, в основе которого лежит принцип разде­ления данных и связей, обеспечивает, с одной стороны, независи­мость данных, а с другой — более простые способы хранения и об­новления.

Первые шаги преобразования состоят в превращении каждой сущности в отношение (таблицу). Связь типа М: М, которую назы­вают «сущность—связь», тоже превращается в отдельное отношение. Каждое свойство становится атрибутом — столбцом соответствую­щей таблицы.

После реализации этих шагов получаем реляционную схему, изображенную на рис. 19, где представлены таблицы «Студенты», «Сводная ведомость», «Учебный план» и «Кадровый состав», ото­бражающие соответственно сущности «Студент», «Сводная ведо­мость», «Дисциплина учебного плана» и «Преподаватель».

Далее необходимо преобразовать связи во внешние ключи. Связь «многие ко многим», реализуемая отношением «Сводная ве­домость», должна содержать уникальные идентификаторы сущно­стей — участников связи. При этом, если для однозначной иденти­фикации студента достаточно добавить в таблицу столбец *ID\_Cmyдент,* то однозначная идентификация дисциплины потребует добавления в таблицу столбцов *Наименование, Семестр* и *Форма\_отчетности.* Хранение всей этой информации явно приведет к *избыточности* данных и их потенциальной *противоречивости* (на­пример, если при переносе дисциплины на другой семестр обновить только строку таблицы «Учебный план», то содержимое таблицы «Сводная ведомость» станет неактуальным).

*** Рис. 19. Реляционная схема после первого этапа преобразования***

Для ликвидации избыточности и потенциальной противоречи­вости данных добавим в таблицу «Учебный план» столбец ID\_*План,* содержимое которого будет однозначно идентифицировать каждую строку таблицы. Теперь этот новый столбец станет первичным клю­чом, и одноименный столбец должен быть добавлен в таблицу «Сводная ведомость».

Связь «Читает» предполагает добавление в таблицу «Учебный план» столбца *ID Преподаватель.* Реляционная схема со связями представлена на рис. 20.



***Рис. 20. Реляционная схема со связями***

**Нормализация таблиц**

Все построенные таблицы находятся в *первой нормальной форме,* так как каждый столбец таблицы неделим и в рамках одной табли­цы нет столбцов с одинаковыми по смыслу значениями.

Таблица «Сводная ведомость» через столбцы ID*\_Студент* и ID\_*План* связывает информацию о студенте с информацией о кон­кретной дисциплине и фиксирует оценку, полученную студентом. Оценка и дата сдачи экзамена (зачета) однозначно зависят от содер­жимого столбцов ID\_*Студент* и ID\_*План,* которые представляют собой составной первичный ключ. Таким образом, все таблицы имеют первичные ключи, которые однозначно определяют строки и неизбыточны, и можно говорить о том, что таблицы находятся *во второй нормальной форме.*

Рассмотрим подробнее таблицу «Учебный\_план», которая со­держит перечень дисциплин текущего учебного плана. Первичным ключом таблицы служит столбец ID\_*План,* который однозначно ха­рактеризует каждую дисциплину учебного плана с точностью до се­местра, т. е. для дисциплин, протяженность изучения которых более одного семестра, в таблице будет отведено столько строк, сколько семестров длится изучение дисциплины. Тогда хранение наимено­ваний дисциплин в таблице «Учебный\_план» становится избыточ­ным: например, если изучение английского языка длится шесть се­местров, то наименование «Английский язык» будет повторено в шести записях и есть вероятность сделать шесть различных ошибок при вводе одного и того же наименования.

Чтобы избежать этого, проведем декомпозицию отношения «Учебный план», выделив наименования дисциплин в отдельное от­ношение. В результате получим дополнительную таблицу «Дисцип­лины» со столбцами ID*\_Дисциплина* и *Наименование,* а столбец *На­именование* в таблице «Учебный\_план» заменим столбцом ID\_Д*исциплина,* сформировав тем самым вторичный ключ, связывающий новую таблицу с таблицей «Учебный\_план».

Таблица «Студенты»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Ограничения |
| **ID\_Студент** | **Целое число** | **Значение уникально** |
| **Фамилия** | **Строка символов размером 30** | **Значение не должно быть пустым** |
| **Имя** | **Строка символов размером 15** | **Значение не должно быть пустым** |
| **Отчество** | **Строка символов размером 20** | **Значение не должно быть**  **пустым** |
| **Номер группы** | **Целое число** | **Значение не должно быть пустым** |
| **Адрес** | **Строка символов размером 30** |  |
| **Телефон** | **Строка символов размером 8** |  |

Таблица «Дисциплины»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Ограничения |
| **ID\_Дисциплина** | **Целое число** | **Значение уникально** |
| **Наименование** | **Строка символов размером 20** | **Значение уникально** |

Таблица «Кадровый\_состав»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Ограничения |
| **ID\_Преподаватель** | **Целое число** | **Значение уникально** |
| **Фамилия** | **Строка символов размером 30** | **Значение не должно быть пустым** |
| **Имя** | **Строка символов размером 15** | **Значение не должно быть пустым** |
| **Отчество** | **Строка символов размером 20** | **Значение не должно быть пустым** |
| **Должность** | **Строка символов размером 20** | **Значение не должно быть пустым** |
| **Кафедра** | **Строка символов размером 3** | **Значение не должно быть пустым** |
| **Адрес** | **Строка символов размером 30** |  |
| **Телефон** | **Строка символов размером 8** |  |

Таблица «Учебный план»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Ограничения |
| **ID\_План** | **Целое число** | **Значение уникально** |
| **ID\_Дисциплина** | **Целое число** | **Значение не должно быть пустым** |
| **Семестр** | **Целое число** | **Значение не должно быть пустым и должно находиться в интервале от 1 до 10** |
| **Количество часов** | **Целое число** |  |
| **ID\_Преподаватель** | **Целое число** |  |

Таблица «Сводная ведомость»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Ограничения |
| **ID\_Студент** | **Целое число** | **Значение не должно быть пустым** |
| **ID\_План** | **Целое число** | **Значение не должно быть пустым** |
| **Оценка** | **Целое число** | **Значение не должно быть пустым и должно находиться в интервале от 0 до 5** |
| **Дата сдачи** | **Дата-время** | **Значение не должно быть пустым, по умолчанию — текущая дата** |

**ВВЕДЕНИЕ В SQL**

Язык SQL используется для реализации всех функцио­нальных возможностей, необходимых для управления БД:

* *организация данных* - позволяет определять и изменять структуру представления данных, а также устанавливать отно­шения;
* *обработка данных -* позволяет изменять содержимое базы данных: добавлять новые данные, удалять или обновлять уже имеющиеся в ней данные;
* *управление доступом -* позволяет ограничивать возмож­ности пользователя по чтению и изменению данных (защита данных от несанкционированного доступа) и координировать их совместное использование пользователями, работающими параллельно.

Таким образом, хотя SQL и не объявляется полноценным язы­ком программирования, он является достаточно полным и мощ­ным языком для управления взаимодействием с СУБД. SQL являет­ся *подъязыком* баз данных, предназначенным для управления базами данных. SQL на сегодняш­ний день является *единственным* стандартным языком для работы с реляционными базами данных.

Операторы SQL *встраиваются* в базовый язык, например, PASCAL, FORTRAN или С, и дают возможность получать доступ к базам данных из прикладных программ.

**Основные понятия и компоненты.**

**Инструкции и имена**

SQL представлен множеством инструкций, каждая из которых пред­писывает СУБД выполнить определенное действие. Инструк­ция SQL начинается с *команды —* ключевого слова, описывающего действие, выполняемое инструкцией. Следом за командой указывается одно или не­сколько *предложений.* Предложение описывает данные, с которыми должна работать инструкция, или уточняет действие, выполняемое инструкцией. Предложения в инструкции делятся на обязательные и необязательные. Каждое предложение начинается с ключевого слова. Многие предложения в качестве параметров содержат имена таблиц или столбцов; некоторые из них могут содержать дополнительные ключевые слова, константы и выражения.

У каждого объекта в базе данных есть уникальное *имя.* Имена используются в инструкциях SQL и указывают, над каким объектом базы данных инструкция должна выполнить действие. В соответст­вии со стандартом ANSI/ISO имена в SQL могут содержать от 1 до 128 символов, начинаться с буквы и не должны включать пробелов или специальных символов пунктуации.

В инструкциях SQL могут использоваться как полные имена объектов, так и короткие. Полное имя таблицы (в отличие от корот­кого) содержит имя пользователя и короткое имя таблицы, разде­ленные точкой:

<Имя\_пользователя>.<Имя\_таблицы>

При этом уникальность именования таблицы сохраняется в слу­чае, если в рамках одной базы данных разные пользователи создают таблицы с одинаковыми именами.

Полное имя столбца в свою очередь состоит из полного (или короткого) имени таблицы, которой принадлежит столбец, и корот­кого имени столбца, разделенных точкой:

<Имя\_пользователя>.<Имя\_таблицы>.<Имя\_столбца> или <Имя\_таблицы>.<Имя столбца>

В рамках одной таблицы не может быть определено двух столб­цов с одинаковыми именами, но в разных таблицах это возможно. При этом в инструкциях SQL необходимо использовать полное именование столбцов.

**Типы данных**

Современные СУБД позволяют обрабатывать данные разнообразных типов, среди которых наиболее распространенными можно назвать следующие.

Целые числа (INT, SMALLINT). В столбцах, имеющих такой тип данных, обычно хранятся данные о количестве и возрасте со­трудников, идентификаторы.

Десятичные числа (NUMERIC, DECIMAL). В столбцах данного типа хранятся числа, имеющие дробную часть с фиксированным количеством знаков после запятой, например курсы валют и про­центы.

Числа с плавающей запятой (REAL, FLOAT). Числа с плаваю­щей запятой представляют больший диапазон действительных зна­чений, чем десятичные числа.

Строки символов постоянной длины (CHAR). В столбцах, имеющих этот тип данных, хранятся имена и фамилии, географиче­ские названия, адреса и т. п.

Строки символов переменной длины (VARCHAR). Столбцы этого типа позволяют хранить символьные строки, длина которых изменяется в заданном диапазоне.

Денежные величины (MONEY, SMALLMONEY). Наличие от­дельного типа данных для хранения денежных величин позволяет правильно форматировать их и снабжать признаком валюты перед выводом на экран.

Дата и время (DATETIME, SMALLDATETIME). Поддержка особого типа данных для значений дата/время. Как правило, с этим типом данных связа­ны особые операции и процедуры обработки.

Булевы величины (BIT). Столбцы такого типа данных позволя­ют хранить логические значения True (1) и False (0).

Длинный текст (TEXT). Многие СУБД поддерживают хранение в столбцах текстовых строк длиной до 32КБ или 64КБ символов, а в некоторых случаях и больше. Это позволяет хранить в базе данных целые документы.

Неструктурированные потоки байтов (BINARY, VARBINARY, IMAGE). Современные СУБД позволяют хранить и извлекать не­структурированные потоки байтов переменной длины. Такой тип данных обычно используется для хранения графических и видео­изображений, исполняемых файлов и других неструктурированных данных.

**Встроенные функции**

Язык SQL содержит так называемые встроенные функции, которые реализуют некоторые наиболее распространенные алгоритмы. Ос­новной особенностью этих функций является возможность их ис­пользования при построении выражений.

Встроенные функции, доступные при работе с SQL, можно ус­ловно разделить на следующие группы:

• математические функции;

• строковые функции;

• функции для работы с величинами типа дата-время;

• функции конфигурирования;

• системные функции;

• функции системы безопасности;

• функции управления метаданными;

• статистические функции.

В таблице приведены наиболее часто используемые функции первых трех групп.

|  |  |
| --- | --- |
| Функция | Назначение |
| АВS(число) | Вычисляет абсолютную величину числа |
| ISNUMERIC(выражение) | Определяет, имеет ли выражение числовой тип данных |
| SIGN(число) | Определяет знак числа |
| RAND(целое число) | Вычисляет случайное число |
| ROUND(число, точность) | Выполняет округление числа с указанной точ­ностью |
| POWER (число, степень) | Возводит число в степень |
| SQRT(число) | Извлекает квадратный корень из числа |
| SIN (угол) | Вычисляет синус угла, указанного в радианах |
| COS(угол) | Вычисляет косинус угла, указанного в радианах |
| ЕХР(число) | Вычисляет экспоненту числа |
| LOG(число) | Вычисляет натуральный логарифм числа |
| LEN (строка) | Вычисляет длину строки в символах |
| LTRIM(строка) | Удаляет пробелы в начале строки |
| RTRIM(cтрокa) | Удаляет пробелы в конце строки |
| LEFT(строка, количество) | Возвращает указанное количество символов строки, начиная с самого левого символа |
| RIGHT(строка, количество) | Возвращает указанное количество символов строки, начиная с самого правого символа |
| LOWER (строка) | Приводит символы строки к нижнему регистру |
| UPPER (строка) | Приводит символы строки к верхнему регистру |
| STR (число) | Выполняет конвертирование числового значе­ния в символьный формат |
| SUBSTRING (строка, индекс, длина) | Возвращает для строки подстроку заданной длины, начиная с символа заданного индекса |
| GETDATE() | Возвращает текущую системную дату |
| ISDATE(строка) | Проверяет строку на соответствие одному из форматов даты и времени |
| DAY(дата) | Возвращает число указанной даты |
| MONTH (дата) | Возвращает месяц указанной даты |
| YEAR(дата) | Возвращает год указанной даты |
| DATEADD(тип, число, дата) | Прибавляет к дате указанное число единиц за­данного типа (год, месяц, день, час и т. п.) |

**Значения NULL**

При заполнении таблиц базы данных отдельные элементы в них мо­гут отсутствовать. Например, при заполнении таблицы «Студенты» или «Кадровый\_состав» может быть не задан для некоторых строк номер телефона, тем не менее, строка должна быть введена в табли­цу и должна участвовать в запросах на выдачу информации.

SQL поддерживает обработку не определенных (не заданных) данных с помощью использования так называемого отсутствующего значения (NULL). Это значение показывает, что в конкретной строке конкретный элемент данных отсутствует. При этом NULL не является значением данных и в связи с этим не имеет определенно­го типа. Это всего лишь признак, показывающий, что значение эле­мента данных не задано.

Правила обработки значений NULL в различных инструкциях и предложениях включены в синтаксис языка.

**Ограничения целостности.**

***Первичный ключ таблицы***

Всякая таблица обычно содержит один или несколько столбцов, значение или совокупность значений которых *уникально идентифи­цируют* каждую строку в таблице. Этот столбец (или столбцы) назы­вается *первичным ключом* (Primary Key, PK) таблицы.

Если в первичный ключ входит более одного столбца, значения в пределах одного столбца могут дублироваться, но любая совокуп­ность значений всех столбцов первичного ключа при этом должна быть уникальна. Например, в таблице «Дисциплины» один столбец (*ID\_Дисциплина*) определен как первичный ключ (рис. 21), а для таблицы «Сводная ведомость» задан составной первичный ключ - в него входят значения столбцов *ID\_Студент* и *ID\_Дисциплина*.

Таблица может иметь *только один* первичный ключ, причем ни­какой столбец, входящий в первичный ключ, не может хранить зна­чение NULL.

***Р  
ис. 21. Первичный ключ таблицы «Сводная\_ведомость»***

Еще одним назначением первичного ключа является обеспечение ссылочной целостности данных в нескольких таблицах. Естественно, это может быть реализовано только при наличии соответствующих *внешних ключей* (FOREIGN KEY) в других (дочерних) таблицах.



***Рис. 22. Первичный ключ таблицы «Учебный\_план»***

Если по столбцу строится первичный ключ, столбцу должен быть приписан атрибут PRIMARY KEY (ограничение целостности на уровне столбца), например, описание столбца *ID\_План* для таб­лицы «Учебный\_план» (см. рис. 22) может выглядеть так: *ID\_Дисциплина INTEGER NOT NULL PRIMARY KEY*

Первичный ключ может быть также построен с помощью от­дельного предложения PRIMARY KEY (ограничение целостности на уровне таблицы) - путем включения имени (имен) ключевого столбца (столбцов) в качестве параметров. Например, первичный ключ для таблицы «Сводная\_ведомость» (рис. 1) может быть задан следующим образом:

*PRIMARY KEY (ID\_Дисциплина, ID\_Студент)*

***Внешний ключ таблицы***

Внешний ключ строится в дочерней (зависимой) таблице для соеди­нения родительской (главной) и дочерних таблиц БД.

Это ограничение целостности предназначено для организации ссылочной целостности данных. Внешний ключ связывается с по­тенциальным первичным ключом в другой таблице. Внешний ключ при этом может ссылаться либо на столбец (или столбцы) с ограни­чением целостности PRIMARY KEY, либо на столбец (столбцы) с ограничением целостности UNIQUE.

Таблицу, в которой определен внешний ключ, будем называть *зависимой,* а таблицу с первичным ключом - *главной.* Ссылочная целостность данных двух таблиц обеспечивается следующим обра­зом: в зависимую таблицу нельзя вставить строку, если внешний ключ не имеет соответствующего значения в главной таблице, а из главной таблицы нельзя удалить строку, если значение первичного ключа используется в зависимой таблице.

Столбцы внешнего ключа (в отличие от столбцов первичного ключа) могут содержать значения типа NULL, однако при этом проверка на ограничение FOREIGN KEY будет пропускаться. За­дать внешний ключ можно как при создании, так и при изменении таблиц.

Синтаксис определения внешнего ключа следующий:

FOREIGN KEY (<список столбцов внешнего ключа>)

REFERENCES <имя родительской таблицы>

[[<список столбцов родительской таблицы>]

[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT| SET NULL}]]

*Список столбцов внешнего ключа* определяет столбцы дочерней таблицы, по которым строится внешний ключ.

*Имя родительской таблицы* определяет таблицу, в которой опи­сан первичный ключ (или столбец с атрибутом UNIQUE). На этот ключ (столбец) должен ссылаться внешний ключ дочерней таблицы для обеспечения ссылочной целостности.

*Список столбцов родительской таблицы,* определяющий ссылоч­ную целостность, необязателен при ссылке на первичный ключ ро­дительской таблицы. При ссылке в родительской таблице на стол­бец с атрибутом UNIQUE этот список лучше привести.

Параметры ON DELETE, ON UPDATE задают способы измене­ния подчиненных записей дочерней таблицы при удалении (ON DELETE) или изменении (ON UPDATE) поля связи в записи роди­тельской таблицы. Перечислим эти способы:

- NO ACTION - запрещает удаление/изменение родительской записи при наличии подчиненных записей в дочерней таблице;

- CASCADE - при удалении записи родительской таблицы (ис­пользуется совместно с ON DELETE) происходит удаление всех подчиненных записей в дочерней таблице; при измененииполя связи в записи родительской таблицы (используется совместно с ON UPDATE) происходит изменение на то же значение поля внешнего ключа у всех подчиненных записей в дочерней таблице;

- SET DEFAULT - в поле внешнего ключа записей дочерней таблицы заносится значение этого поля по умолчанию, ука­занное при определении поля (параметр DEFAULT);

- SET NULL - в поле внешнего ключа записей дочерней таб­лицы заносится значение NULL.

Установим связь между таблицами «Студенты», «Учебный\_план» и «Сводная\_ведомость»:

ALTER TABLE Сводная ведомость

ADD FOREIGN KEY (ID\_План)

REFERENCES Учебный\_план

ALTER TABLE Сводная ведомость

ADD FOREIGN KEY (ID\_Студент)

REFERENCES Студенты

Хотя в рассмотренном примере имена столбцов первичного и внешнего ключей в обеих таблицах совпадают, это не является обя­зательным. Первичный ключ может быть определен для столбца с одним именем, в то время как столбец, на который наложено огра­ничение FOREIGN KEY, может иметь совершенно другое имя. Од­нако лучше давать таким столбцам идентичные названия, чтобы по­казать связь между ними (рис. 23).

***  
Рис. 23. Связь внешнего и первичного ключей***

***Определение уникального столбца***

Ограничение целостности UNIQUE предназначено для того, чтобы обеспечить уникальность значений в столбце (или нескольких столбцах). Если столбцу приписан атрибут UNIQUE, это означает, что в столбце не могут содержаться два одинаковых значения.

Для ограничения целостности PRIMARY KEY автоматически гарантируется уникальность значений. Однако в каждой таблице можно определить всего один первичный ключ. Если же необходи­мо дополнительно обеспечить уникальность значений еще в одном или более столбцах помимо первичного ключа, то нужно использо­вать ограничение целостности UNIQUE.

Ограничение целостности UNIQUE, в отличие от PRIMARY KEY,допускает существование значения NULL. При этом к значе­нию NULL также предъявляется требование уникальности, поэтому в столбце с ограничением целостности UNIQUE допускается суще­ствование лишь единственного значения NULL.

Таким образом, ограничение UNIQUE используется в том слу­чае, когда столбец не входит в состав первичного ключа, но, тем не менее, его значение всегда должно быть уникальным. Например, для таблицы «Дисциплины» первичный ключ строится по номеру дисциплины *ID\_Дисциплина,* введенному для сокращения объема первичного ключа и времени поиска по нему (объем ключа по столбцу типа INTEGER много меньше объема ключа по символьно­му полю). Однако и название дисциплины (столбец *Наименование)* должно быть уникальным, для чего ему приписан атрибут UNIQUE:

CREATE TABLE Дисциплины

(ID\_Дисциплина INTEGER NOT NULL PRIMARY KEY,

Наименование VARCHAR (20) NOT NULL UNIQUE)

Уникальность может быть определена и на уровне таблицы:

CREATE TABLE Дисциплины

(ID\_Дисциплина INTEGER NOT NULL,

Наименование VARCHAR (20) NOT NULL,

PRIMARY KEY (ID\_Дисциплина),

UNIQUE (Наименование))

***Определение проверочных ограничений***

Ограничение целостности CHECK задает диапазон возможных зна­чений для столбца. Например, если в столбце хранится процентное значение, то необходимо гарантировать, что оно будет лежать в пределах от 0 до 100. Для этого можно использовать тип данных, допус­кающий хранение целых значений в диапазоне от 0 до 255, совмест­но с ограничением целостности CHECK, которое будет обеспечи­вать соответствующую проверку значений.

Преимуществом ограничения целостности CHECK является возможность определения для одного столбца множества правил контроля значений.

В основе ограничения целостности CHECK лежит проверка логического выражения, которое возвращает значение TRUE (ис­тина) либо значение FALSE (ложь). Если возвращается значение TRUE, то ограничение целостности выполняется, и операция из­менения или вставки данных разрешается. Когда же возвращается значение FALSE, то операция изменения или вставки данных от­меняется.

Например, для обеспечения правильности задания значения для столбца *Семестр* в таблице «Учебный\_план» (оно должно находить­ся в диапазоне от 1 до 10) можно использовать следующее логиче­ское выражение:

((Семестр >= 1) OR (Семестр <= 10)))

Ограничение целостности при этом может быть задано на уров­не столбца:

Семестр INTEGER NOT NULL CHECK ((Семестр >= 1) OR (Семестр <= 10)))

Или на уровне таблицы:

CHECK ((Семестр >= 1) OR (Семестр <= 10)))

Как уже было сказано, допускается применение нескольких ог­раничений CHECK к одному и тому же столбцу. В этом случае они будут применены в той последовательности, в какой они указаны в инструкции.

***Определение значения по умолчанию***

При вводе записи (строки) в таблицу каждый столбец должен со­держать какое-либо значение. Если значение для столбца не указа­но, то столбец заполняется значениями NULL (конечно, если для него разрешено хранение значений NULL). Однако это нежела­тельно. Наилучшим решением в подобных ситуациях может быть определение для столбца *значений по умолчанию.* Например, часто ноль определяется как значение по умолчанию для числовых столбцов, а «n/а» (не определено) — как значение по умолчанию для символьных столбцов. Таким образом, определение для столбца значения по умолчанию гарантирует автоматическую подстановку этого значения, если при вставке новых строк значение для столб­ца не указано.

Использование значений по умолчанию позволяет ускорить процесс ввода информации. Значитель­но расширяет область применения значений по умолчанию возмож­ность вызова встроенных функций.

**УПРАВЛЕНИЕ ТАБЛИЦАМИ**

***Команда создания таблицы — CREATE TABLE***

Создание таблицы выполняется при помощи команды CREATE TABLE. Обобщенный синтаксис команды следующий:

CREATE TABLE имя\_таблицы

({<определение\_\_столбца>|<определение\_ограничения\_таблицы>}

[,...,{<определение\_стол6ца>|<определение\_ограничения\_таблицы >}])

То есть после задания имени таблицы через запятую в круглых скобках должны быть перечислены все предложения, определяю­щие отдельные элементы таблицы, — столбцы или ограничения це­лостности:

*имя\_таблицы* — идентификатор создаваемой таблицы, который в общем случае строится из имени базы данных, имени владельца таблицы и имени самой таблицы. При этом комбинация имени таб­лицы и ее владельца должна быть уникальной в пределах базы дан­ных. Если таблица создается не в текущей базе данных, в ее иденти­фикатор необходимо включить имя базы данных;

*определение\_столбца* — задание имени, типа данных и парамет­ров отдельного столбца таблицы. Названия столбцов должны соот­ветствовать правилам для идентификаторов и быть уникальными в пределах таблицы;

*определение\_ограничения\_таблицы* — задание некоторого ограни­чения целостности на уровне таблицы.

*Описание столбцов*

Как видно из синтаксиса команды CREATE TABLE, для каждо­го столбца указывается предложение <определение\_столбца>, с по­мощью которого и задаются свойства столбца. Предложение имеет следующий синтаксис:

<Имя\_столбца> <тип\_данных>

[<ограничение\_столбца> ] [,...,<ограничение\_столбца>]

*Имя\_столбца* — идентификатор, задающий имя столбца табли­цы;

*Тип\_данных —* задает тип данных столбца. Если при определе­нии столбца явно не указано ограничение на хранение значений NULL, то будут использованы свойства типа данных, т. е. если вы­бранный тип данных позволяет хранить значения NULL, то и в столбце можно будет хранить значения NULL. Если же при опреде­лении столбца в команде CREATE TABLE явно будет разрешено или запрещено хранение значений NULL, то свойства типа данных будут перекрыты установленным на уровне столбца ограничением. Например, если тип данных позволяет хранить значения NULL, a на уровне столбца будет установлен запрет, то попытка вставки зна­чения NULL в столбец закончится ошибкой;

*Ограничение\_столбца* — с помощью этого предложения указыва­ются ограничения, которые будут определены для столбца. Синтак­сис предложения следующий:

<ограничение\_столбца>::=[ CONSTRAINT <имя\_ограничения > ]

{[ DEFAULT <выражение>]

| [NULL | NOT NULL]

| [PRIMARY KEY | UNIQUE]

| [FOREIGN KEY

REFERENCES <имя\_главной\_таблицы>[(<имя\_столбца> [,...,n])]

[ON DELETE {CASCADE | NO ACTION}]

[ON UPDATE {CASCADE | NO ACTION}]

]

| [CHECK (<логическое\_выражение>)]

}

CONSTRAINT — необязательное ключевое слово, после кото­рого указывается название ограничения на значения столбца (имя\_ограничения). Имена ограничений должны быть уникальны в пределах базы данных.

DEFAULT — задает значение по умолчанию для столбца. Это значение будет использовано при вставке строки, если для столбца явно не указано никакое значение.

NULL|NOT NULL — ключевые слова, разрешающие (NULL) или запрещающие (NOT NULL) хранение в столбце значений NULL. Если для столбца не задано значение по умолчанию, то при вставке строки с неизвестным значением для столбца будет пред­приниматься попытка вставки в столбец значения NULL. Если при этом для столбца указано ограничение NOT NULL, то попытка вставки строки будет отклонена, и пользователь получит соответст­вующее сообщение об ошибке.

PRIMARY KEY — определение первичного ключа на уровне од­ного столбца (т. е. первичный ключ будет состоять только из значе­ний одного столбца). Если необходимо сформировать первичный ключ на базе двух и более столбцов, то такое ограничение целостно­сти должно быть задано на уровне таблицы. При этом следует пом­нить, что для каждой таблицы может быть создан только один пер­вичный ключ.

UNIQUE — указание на создание для столбца ограничения це­лостности UNIQUE, что позволит гарантировать уникальность каж­дого отдельного значения в столбце в пределах этого столбца. В таб­лице может быть создано несколько ограничений целостности UNIQUE.

FOREIGN KEY ... REFERENCES — указание на то, что столбец будет служить внешним ключом для таблицы, имя которой задается с помощью параметра <имя\_главной\_таблицы>.

*(имя столбца* [,...,n]*)* — столбец или список перечисленных че­рез запятую столбцов главной таблицы, входящих в ограничение FOREIGN KEY. При этом столбцы, входящие во внешний ключ, могут ссылаться только на столбцы первичного ключа или столбцы с ограничением UNIQUE таблицы.

ON DELETE {CASCADE | NO ACTION} — эти ключевые слова определяют действия, предпринимаемые при удалении строки из главной таблицы. Если указано ключевое слово CASCADE, то при удалении строки из главной (родительской) таблицы строка в зави­симой таблице также будет удалена. При указании ключевого слова NO ACTION в подобном случае будет выдана ошибка. Значением по умолчанию является вариант NO ACTION.

ON UPDATE {CASCADE | NO ACTION} - эти ключевые слова определяют действия, предпринимаемые при модификации строки главной таблицы. Если указано ключевое слово CASCADE, то при модификации строки из главной (родительской) таблицы строка в зависимой таблице также будет модифицирована. При использова­нии ключевого слова NO ACTION в подобном случае будет выдана ошибка. Значением по умолчанию является вариант NO ACTION.

CHECK — ограничение целостности, инициирующее контроль вводимых в столбец (или столбцы) значений;

*Логическое выражение* — логическое выражение, используемое для ограничения CHECK.

*Ограничения на уровне таблицы*

Синтаксис команды CREATE TABLE предусматривает исполь­зование предложения <ограничение таблицы>, с помощью которо­го определяются ограничения целостности на уровне таблицы. Син­таксис предложения следующий:

<ограничение таблицы>::= [CONSTRAINT <имя ограничения>]

{[{PRIMARY KEY | UNIQUE }

{(<имя колонки> [ASC | DESC] [,...,n] )}]

| FOREIGN KEY

[(<имя\_колонки>[,..., n])]

REFERENCES <внешняя таблица> [(<имя\_колонки\_внешней\_таблицы> [, ..., n])]

[ON DELETE {CASCADE | NO ACTION}]

[ON UPDATE {CASCADE | NO ACTION}]

| CHECK (<логическое выражение>)

Назначение параметров совпадает с назначением аналогичных параметров предложения <ограничение столбца>.

*Имя колонки* — столбец (или список столбцов), на которые не­обходимо наложить какие-либо ограничения целостности;

[ASC | DESC] — метод упорядочивания данных в индексе. Ин­декс создается при указании ключевых слов PRIMARY KEY, UNIQUE. При указании значения ASC данные в индексе будут упо­рядочены по возрастанию, при указании значения DESC — по убы­ванию. По умолчанию используется значение ASC.

***Примеры создания таблиц***

В качестве примера рассмотрим инструкции создания таблиц базы данных «Сессия».

Таблица «Студенты» состоит из следующих столбцов:

ID\_Студент — тип данных INTEGER, уникальный ключ;

Фамилия — тип данных CHAR, длина 30;

Имя — тип данных CHAR, длина 15;

Отчество — тип данных CHAR, длина 20;

Номер группы — тип данных CHAR, длина 6;

Адрес — тип данных CHAR, длина 30;

Телефон — тип данных CHAR, длина 8.

Создание таблицы выполнялось с помощью следующей команды:

CREATE TABLE Студенты

(ID\_Студент INTEGER NOT NULL,

Фамилия CHAR(30) NOT NULL,

Имя CHAR(15) NOT NULL,

Отчество CHAR(20) NOT NULL,

Номер\_группы INTEGER NOT NULL,

Адрес CHAR(30),

Телефон CHAR(8),

PRIMARY KEY (ID\_Студент) )

На все столбцы таблицы, кроме столбцов *Адрес* и *Телефон,* нало­жены ограничения NOT NULL, запрещающие ввод строки при не­определенном значении столбца.

Для создания таблицы «Дисциплины» была использована команда:

CREATE TABLE Дисциплины

(ID\_Дисциплина INTEGER NOT NULL,

Наименование VARCHAR(40) NOT NULL,

PRIMARY KEY (ID\_Дисциплина),

UNIQUE (Наименование))

Таблица содержит два столбца (ID*\_Дисциплина, Наименование).*

На столбцы ID*\_Дисциплина, Наименование* наложены ограниче­ния NOT NULL, запрещающие ввод строки при неопределенном значении столбца.

Столбец ID*\_Дисциплина* объявлен первичным ключом, а на зна­чения, вводимые в столбец *Наименование,* наложено условие уни­кальности.

Таблица «Учебный\_план» включает в себя следующие столбцы:

ID\_План — тип данных INTEGER, столбец уникального ключа;

ID\_Дисциплина — тип данных INTEGER;

Семестр — тип данных INTEGER;

Количество\_часов — тип данных INTEGER;

ID\_Преподаватель — тип данных INTEGER.

Создание таблицы выполнялось с помощью следующей команды:

CREATE TABLE Учебный\_план

(ID\_План INTEGER NOT NULL,

ID\_Дисциплина INTEGER NOT NULL,

Семестр INTEGER NOT NULL,

Количество\_часов INTEGER,

ID\_Преподаватель INTEGER,

PRIMARY KEY (ID\_План),

CHECK ((Семестр >= 1) OR (Семестр <= 10)))

Для значений столбца *Семестр* сформулировано логическое вы­ражение, разрешающее вводить только значения от 1 до 10.

Таблица «Своднаяведомость» состоит из следующих столбцов:

ID\_Студент — тип данных INTEGER, столбец уникального ключа;

ID\_План — тип данных INTEGER, столбец уникального ключа;

Оценка — тип данных INTEGER;

Дата\_сдачи — тип данных DATETIME;

ID\_Преподаватель — тип данных INTEGER.

Создание таблицы выполнялось с помощью следующей команды:

CREATE TABLE Сводная\_ведомость

(ID\_Студент INTEGER NOT NULL,

ID\_План INTEGER NOT NULL,

Оценка INTEGER NOT NULL,

Дата\_сдачи DATETIME NOT NULL,

PRIMARY KEY (ID\_Студент, ID\_Дисциплина),

CHECK ((Оценка >= 0) OR (Оценка <= 5)))

На все столбцы таблицы наложены ограничения NOT NULL, запрещающие ввод строки при неопределенном значении столбца.

Для значений столбца *Оценка* сформулировано логическое вы­ражение, разрешающее вводить только значения от 0 до 5: 0 — неза­чет, 1 — зачет, 2 — неудовлетворительно, 3 — удовлетворительно, 4 — хорошо, 5 — отлично.

И, наконец, перечислим столбцы таблицы «Кадровый состав»:

ID\_Преподаватель — тип данных INTEGER, уникальный ключ;

Фамилия — тип данных CHAR, длина 30;

Имя — тип данных CHAR, длина 15;

Отчество — тип данных CHAR, длина 20;

Должность — тип данных CHAR, длина 20;

Кафедра — тип данных CHAR, длина 3;

Адрес — тип данных CHAR, длина 30;

Телефон — тип данных CHAR, длина 8.

Создание таблицы выполнялось с помощью следующей команды:

CREATE TABLE Кадровый\_состав

(ID\_ Преподаватель INTEGER NOT NULL,

Фамилия CHAR(30) NOT NULL,

Имя CHAR(15) NOT NULL,

Отчество CHAR(20) NOT NULL,

Должность CHAR(20) NOT NULL,

Кафедра CHAR(3) NOT NULL,

Адрес CHAR(30),

Телефон CHAR(8),

PRIMARY KEY (ID Преподаватель))

На все столбцы таблицы, кроме столбцов *Адрес* и *Телефон,* нало­жены ограничения NOT NULL, запрещающие ввод строки при не­определенном значении столбца.

Для таблиц «Учебный\_план» и «Сводная\_ведомость» должны быть построены внешние ключи, связывающие таблицы базы дан­ных «Сессия»:

FК\_Дисциплина — внешний ключ, связывающий таблицы «Учебный\_план» и «Дисциплины» по столбцу *ID\_ Дисциплина;*

FК\_Кадровый\_состав — внешний ключ, связывающий табли­цы «Учебный\_план» и «Кадровый\_состав» по столбцу *ID\_Преподаватель;*

FК\_Студент — внешний ключ, связывающий таблицы «Сводная\_ведомость» и «Студенты» по столбцу *ID\_Студент;*

FК\_План — внешний ключ, связывающий таблицы «Сводная\_ведомость» и «Учебный\_план» по столбцу *ID\_План.*

Добавление внешних ключей в таблицы будет описано при рас­смотрении возможностей команды ALTER TABLE.

***Изменение структуры таблицы — команда ALTER TABLE***

Специальная команда ALTER TABLE предназначена для *моди­фикации* структуры таблицы. С ее помощью можно изменять свойства существующих столбцов, удалять или добавлять в таблицу столбцы, а также управлять ограничениями целостности как на уровне столбца, так и на уровне таблицы, т. е. выполнять следую­щие функции:

• добавить в таблицу определение нового столбца;

• удалить столбец из таблицы;

• изменить значение по умолчанию для какого-либо столбца;

• добавить или удалить первичный ключ таблицы;

• добавить или удалить внешний ключ таблицы;

• добавить или удалить условие уникальности;

• добавить или удалить условие на значение.

Рассмотрим обобщенный синтаксис команды ALTER TABLE:

ALTER TABLE <имя\_таблицы>

[ALTER COLUMN <имя столбца> [SET DEFAULT <выражение>]|

[DROP DEFAULT]]

|[ADD <определение\_столбца>]

|[DROP COLUMN <имя\_столбца> [CASCADE]|[RESTRICT]]

|[ADD [<определение\_первичного\_ключа>]|[<определение\_внешнего ключа>]|

[<условис\_уникальности>]|[<условие\_на\_значение>]]

|[DROP CONSTRAINT <имя\_ограничения> [CASCADE]|[RESTRICT]]

Команда ALTER TABLE берет на себя все действия по копиро­ванию данных во временную таблицу, удалению старой таблицы, созданию вместо нее новой таблицы с нужной структурой и после­дующим переписыванием в нее данных.

Назначение многих параметров и ключевых слов команды ALTER TABLE аналогично назначению соответствующих парамет­ров и ключевых слов команды CREATE TABLE (например, синтак­сис конструкции <определение\_столбца> совпадает с синтаксисом аналогичной конструкции команды CREATE TABLE).

Основные режимы использования команды ALTER TABLE сле­дующие:

• добавление столбца;

• удаление столбца;

• модификация столбца;

• изменение, добавление и удаление ограничений (первичных и внешних ключей, значений по умолчанию).

***Добавление столбца***

Для добавления нового столбца следует использовать ключевое слово ADD, после которого должно стоять определение столбца.

Добавим, например, в таблицу «Студенты» столбец *Год\_поступления* следующим образом:

ALTER TABLE Студенты

ADD Год\_поступления INTEGER NOT NULL DEFAULT YEAR(GETDATE())

После выполнения этой команды в структуру таблицы «Сту­дент» будет добавлен еще один столбец со значением по умолча­нию, равным текущему году (значение по умолчанию вычисляется с помощью двух встроенных функций — YEAR() и GETDATE()).

***Модификация столбца***

Для модификации существующего столбца таблицы служит ключевое слово ALTER COLUMN. Изменение свойств столбца не­возможно, если:

столбец участвует в ограничениях PRIMARY KEY или FOREIGN KEY;

на столбец наложены ограничения целостности CHECK или UNIQUE (исключение составляют столбцы, имеющие тип данных переменной длины, т. е. типы данных, начинающиеся на var);

со столбцом связано значение по умолчанию (в этом случае допускается изменение длины, общего количества цифр или количества цифр после десятичной точки при неизменном типе данных).

Определяя для столбца новый тип данных, следует помнить о том, что старый тип данных должен конвертироваться в новый.

Пример модификации столбца «Номер\_группы» таблицы «Сту­денты» (тип данных INTEGER заменяется на CHAR):

ALTER TABLE Студенты

ALTER COLUMN Номер\_группы CHAR(6) NOT NULL

***Удаление столбца***

Для удаления столбца из таблицы используется предложение DROP COLUMN <имя\_столбца>. При удалении столбцов следует учитывать, что нельзя удалять столбцы с ограничениями целостно­сти CHECK, FOREIGN KEY, UNIQUE или PRIMARY KEY, а так­же столбцы, для которых определены значения по умолчанию (в виде ограничения целостности на уровне столбца или на уровне таблицы).

Рассмотрим, например, команду удаления из таблицы «Студен­ты» столбца «Год\_поступления»:

ALTER TABLE Студенты

DROP COLUMN Год\_поступления

Эта команда выполнена не будет, так как при добавлении столбца было определено значение по умолчанию.

***Добавление ограничений на уровне таблицы***

Для добавления ограничений на уровне таблицы используется предложение ADD CONSTRAINT <имя\_ограничения>.

В качестве примера рассмотрим команды добавления внешних ключей в таблицы базы данных «Сессия».

Добавление внешних ключей в таблицу «Учебный\_план» (созда­ние связи с именем FК\_Дисциплина и связи с именем FK\_ Кадровый\_состав):

ALTER TABLE Учебный\_план

ADD CONSTRAINT FК\_Дисциплина

FOREIGN KEY (ID\_Дисциплина)

REFERENCES Дисциплины

ALTER TABLE Учебный\_план

ADD CONSTRAINT FК\_Кадровый\_состав

FOREIGN KEY (ID\_Преподаватель)

REFERENCES Кадровый\_состав

Добавление внешних ключей в таблицу «Сводная\_ведомость» (создание связи с именем FK\_Студент и связи с именем FK\_План):

ALTER TABLE Сводная\_ведомость

ADD CONSTRAINT FK\_Студент

FOREIGN KEY (ID\_Студент)

REFERENCES Студенты

ALTER TABLE Сводная\_ведомость

ADD CONSTRAINT FK\_План

FOREIGN KEY (ID\_План)

REFERENCES Учебный\_план

С помощью конструкции ADD CONSTRAINT создается *поиме­нованное* ограничение. Необходимо отметить, что удаление любого ограничения на уровне таблицы происходит только по его имени, поэтому ограничение должно быть поименовано (чтобы его можно было удалить).

Рассмотрим еще один пример — добавление значения по умол­чанию для столбца *Номер\_группы:*

ALTER TABLE Студент

ADD CONSTRAINT DEF\_Номер\_группы DEFAULT 1 FOR Номер\_группы

В результате выполнения этой команды на уровне таблицы будет создано ограничение целостности с именем DEF\_Номер\_группы.

***Удаление ограничений***

Для удаления из таблицы ограничения целостности использует­ся предложение DROP CONSTRAINT <имя\_ограничения>.

Удаление ограничения целостности возможно только в том слу­чае, когда оно поименовано (т. е. предложение <определение\_ограничения> содержит именование ограничения CONSTRAINT).

Команда удаления построенного внешнего ключа FK\_Дисциплина из таблицы «Учебный\_план» выглядит следующим образом:

ALTER TABLE Учебный\_план

DROP CONSTRAINT FK\_Дисциплина

Удалить же построенное ограничение DEF\_Номер\_группы можно с помощью следующей команды:

ALTER TABLE Студент

DROP CONSTRAINT DEF\_Номер\_группы

***Удаление таблиц. Команда DROP TABLE***

Удаление таблицы выполняется при помощи команды DROP TABLE:

DROP TABLE <имя\_таблицы>

Единственный аргумент команды задает имя таблицы, которую необходимо удалить.

Операция удаления таблицы в некоторых случаях требует опреде­ленного внимания. Невозможно удалить таблицу, если на нее с по­мощью ограничения целостности FOREIGN KEY ссылается другая таблица: попытка удаления таблицы «Дисциплины» вызовет сообще­ние об ошибке, так как на таблицу «Дисциплины» ссылается таблица «Учебный\_план». Например, в ответ на использование команды:

DROP TABLE Дисциплины

будет выдано сообщение об ошибке, гласящее, что невозможно уда­лить таблицу, поскольку есть ограничение целостности FOREIGN KEY, ссылающееся на таблицу «Дисциплины».

**УПРАВЛЕНИЕ ДАННЫМИ**

***Извлечение данных — команда SELECT***

Основным инструментом выборки данных в языке SQL является команда SELECT. С помощью этой команды можно получить дос­туп к данным, представленным как совокупность таблиц практиче­ски любой сложности.

Чаще всего используется упрощенный вариант команды SELECT, имеющий следующий синтаксис:

SELECT <Список\_выбора>

[INTO <Новая\_таблица>]

FROM <Исходная\_таблица>

[WHERE<Условие\_отбора>]

[GROUP BY <Ключи группировки>]

[HAVING <Условие\_отбора>]

[ORDER BY <Ключи\_сортировки> [ASC | DESC]]

Инструкция SELECT разбивается на отдельные разделы, каж­дый из которых имеет свое назначение. Из приведенного синтакси­ческого описания видно, что обязательными являются только разде­лы SELECT и FROM, а остальные разделы могут быть опущены. Полный список разделов следующий:

SELECT

INTO

FROM

WHERE

GROUP BY

HAVING

UNION

ORDER BY

COMPUTE

FOR

OPTION

**Раздел SELECT**

Основное назначение раздела SELECT — задание набора столбцов, возвращаемых после выполнения запроса, т. е. внешнего вида результата. В простейшем случае возвращается столбец одной из таблиц, участвующих в запросе. В более сложных ситуациях набор значений в столбце формируется как результат вычисления выражения. Такие столбцы называются *вычисляемыми* и по умолчанию им не присваивается никакого имени. Пользователь может указать для столбца, возвращаемого после выполнения запроса, произвольное имя. Та­кое имя называется *псевдонимом* (alias). Наиболее часто это требуется при работе с раз­делом INTO, в котором каждый из возвращаемых столбцов должен иметь имя, и это имя должно быть уникально.

Синтаксис раздела SELECT следующий:

SELECT [ALL | DISTINCT]

[TOP n [PERCENT] [WITH TIES]]

< Список\_выбора >

***Ключевые слова ALL| DISTINCT***

При указании ключевого слова ALL в результат запроса выводят­ся *все* строки, удовлетворяющие сформулированным условиям, тем самым разрешается включение в результат одинаковых строк. Параметр ALL используется по умолчанию.

Если в запросе SELECT указывается ключевое слово DISTINCT, то в результат выборки не будет включаться более од­ной повторяющейся строки. Таким образом, каждая возвращенная строка будет уникальной.

***Ключевое слово TOP n [PERCENT] [WITH TIES]***

Использование ключевого слова ТОР n, где n — числовое значе­ние, позволяет отобрать в результат не все строки, а только n пер­вых. При этом выбираются первые строки результата выборки, а не исходных данных. Поэтому набор строк в результате выборки при указании ключевого слова ТОР может меняться в зависимости от порядка сортировки. Если в запросе используется раздел WHERE, то ключевое слово ТОР работает с набором строк, возвращенных после применения логического условия, определенного в разделе WHERE.

Продемонстрируем использование ключевого слова ТОР:

SELECT TOP 5 \* FROM Студенты

Из таблицы Студенты было выбрано 5 первых строк.

Можно также выбирать не фиксированное количество строк, а определенный процент от всех строк, удовлетворяющих условию. Для этого необходимо добавить ключевое слово PERCENT:

SELECT TOP 10 PERCENT FROM Студенты

Если указанное количество процентов строк представляет собой нецелое число, то всегда выполняется округление в большую сторону.

При указании вместе с предложением ORDER BY ключевого слова WITH TIES в результат будут включены строки, совпадающие по значению колонки сортировки с последними выведен­ными строками запроса SELECT TOP n [PERCENT].

Использование ключевого слова WITH TIES в следующем примере позволит обеспечить выдачу в ответ на запрос информации обо *всех* студентах первой по порядку группы:

SELECT TOP 10 PERCENT WITH TIES \*

FROM Студенты

ORDER BY Номер\_группы

*Предложение <Список\_выбора>*

Синтаксис предложения <Список\_выбора> следующий:

<Список выбора>::=

{\*

|{<Имя\_таблицы> | <Псевдоним\_таблицы>}.\*

|{<Имя\_столбца> | <Выражение>}

[[AS] <Псевдоним\_столбца>]

|<Псевдоним\_столбца> = <Выражение>

} [,…,n]

Символ «\*» означает включение в результат всех столбцов, имеющихся в списке таблиц раздела FROM.

Если в результат не нужно включать все столбцы *всех* таблиц, то можно явно указать имя объекта, из которого необходимо выбрать все столбцы (<Имя\_таблицы>.\* или <Псевдоним\_таблицы>.\*).

Отдельный столбец таблицы в результат выборки включается явным указанием имени столбца (параметр <Имя\_столбца>). Стол­бец должен принадлежать одной из таблиц, указанных в разделе FROM. Если столбец с указанным именем имеется более чем в од­ном источнике данных, перечисленных в разделе FROM, то необхо­димо явно указать имя источника данных, к которому принадлежит столбец в формате <Имя\_таблицы>.<Имя\_столбца>. В противном случае будет выдано сообщение об ошибке.

Например, попробуем выбрать данные из столбца *ID\_Дисциплина,* который имеется в таблицах «Дисциплина» и «Учебный\_план»:

SELECT ID\_Дисциплина, Наименование, Семестр

FROM Дисциплина, Учебный\_план

В ответ будет выдано сообщение об ошибке, указывающее на некорректное использование имени *ID\_Дисциплина.*

То есть в этом случае необходимо явно указать имя источника данных, которому принадлежит столбец, например:

SELECT Дисциплина.ID\_Дисциплина, Наименование, Семестр

FROM Дисциплина, Учебный план

Столбцам, возвращаемым как результат выполнения запроса, могут быть присвоены *псевдонимы.* Псевдонимы позволяют изме­нить имя исходного столбца или поименовать столбец, содержимое которого получено как результат вычисления выражения. Имя псев­донима указывается с помощью параметра [AS] <Псевдоним\_столбца>. Ключевое слово AS необязательно при задании псевдонима.

В общем случае не требуется уникальности имен столбцов ре­зультата выборки, поэтому разные столбцы могут иметь одинаковые имена или псевдонимы.

Столбцы в результате выборки могут быть не только копией столбца одной из исходных таблиц, но и формироваться на основе вычисления выражения. Такой столбец в списке выбора задается с помощью конструкции <Выражение> [[AS] <Псевдоним\_столбца>]. Выражение при этом может содержать константы, имена столбцов, функции, а также их комбинации. Дополнительно столбцу, форми­руемому на основе вычисления выражения, можно присвоить псев­доним, указав его с помощью параметра [AS] <Псевдоним\_столбца>. По умолчанию вычисляемый столбец не имеет имени.

Другой способ формирования вычисляемого столбца состоит в использовании конструкции со знаком равенства: <Псевдоним\_столбца> = <Выражение>. Единственным отличием этого спо­соба от предыдущего является необходимость *обязательного* задания псевдонима. В простейшем случае выражение является именем столбца, константой, переменной или функцией. Если в качестве выражения выступает имя столбца, то получаем еще один способ задания псевдонима для столбца.

Рассмотрим следующий пример. Пусть для таблицы «Студенты» необходимо построить запрос, представляющий фамилию, имя и отчество в одной колонке. Используя операцию конкатенации (сло­жения) символьных строк и значение ФИО в качестве псевдонима столбца, построим запрос:

SELECT ТОР 10 Фамилия + ' ' + Имя + ' ' + Отчество as ФИО, Номер\_Группы

FROM Студенты

**Раздел FROM**

С помощью раздела FROM определяются источники данных, с которыми будет работать запрос.

Синтаксис раздела FROM следующий:

FROM {<Источник\_данных>} [,...,n]

В основном работа с разделом FROM — это перечисление че­рез запятую источников данных, с которыми должен работать за­прос. Собственно источник данных указывается с помощью предложения <Источник\_данных>, синтаксис которого следующий:

<Источник данных>::= <имя\_таблицы> [[AS] <псевдоним\_таблцы>]| <связка\_таблиц>

С помощью параметра <имя\_таблицы> указывается имя обыч­ной таблицы. Параметр <псевдоним\_таблицы> используется для присвоения таблице псевдонима, под которым на нее нужно будет ссылаться в запросе. Часто псевдонимы таблиц применяют, чтобы ссылку на нужную таблицу сделать более удобной и короткой. На­пример, если в запросе часто упоминается имя таблицы «Учеб-ный\_план», то можно воспользоваться псевдонимом, например, tpl. Указание ключевого слова AS не является при этом обязательным.

Конструкция <связка\_таблиц> реализует один из наиболее сложных методов задания источника данных. С помощью нее мож­но связать данные двух и более таблиц в единый набор данных, ука­зав критерии связывания. Синтаксис конструкции <связка\_таблиц> следующий:

<связка таблиц>::=<левая таблица> <тип\_связывания> <правая таблица> ON <условие\_связывания>

Конструкция <тип\_связывания> описывает тип связывания двух таблиц. Исходная таблица указывается слева от конструкции <тип\_связывания> (<левая\_таблица>), а справа указывается зависи­мая таблица (<правая\_таблица>).

Общий синтаксис конструкции <тип\_связывания> следующий:

<тип\_связывания>::= [INNER | {{LEFT | RIGHT | FULL} [OUTER]}]JOIN

Как видно, обязательным в конструкции является ключевое слово JOIN.

Конструкция ON <условие\_связывания> задает логическое ус­ловие связывания двух таблиц. Допустимы операторы сравнения (например, =, <, >, <=, >=, !-, <>). Чаще всего используется опера­тор равенства, например:

ON Учебный план.ID Дисциплина = Дисциплины.ID Дисциплина

В этом примере устанавливается связь между таблицами «Учеб-ныйплан» и «Дисциплина» по столбцу *ID\_Дисциплина,* имеющему­ся в каждой из таблиц.

***Ключевое слово INNER***

Этот тип связи используется по умолчанию. Указание сочетания INNER JOIN равносильно указанию только ключевого слова JOIN. В качестве кандидатов на включение в результат запроса рассматри­ваются пары строк, удовлетворяющие критерию связывания в обеих таблицах. Затем строки из левой таблицы, для которых не имеется пары в связанной таблице, в результат не включаются. Также не включаются в результат и строки правой таблицы, для которых нет соответствующей строки в левой таблице.

В приведенном ниже примере выполняется выборка данных из таблиц «Дисциплины» и «Учебный\_план» с помощью запроса SELECT. Таблицы связаны по ключевому полю *ID\_Дисциплина,* имеющемуся в каждой из них. Для каждой строки таблицы «Учебный\_план» ищется строка с совпадающим значением поля *ID\_*Д*исциплина* в таблице «Дисциплины». Все строки таблицы «Учеб-ный\_план», для которых нет строк с соответствующим значением поля *ID\_Дисциплина,* игнорируются и не включаются в конечный результат. Аналогично не включаются в результат все строки табли­цы «Дисциплины», для которых нет соответствующей строки в таб­лице «Учебный\_план» (что, однако, невозможно для данного при­мера, так как столбец *ID\_Дисциплина* таблицы «Учебный\_план» свя­зан внешним ключом со столбцом *ID\_Дисциплина* таблицы «Дисциплины»).

SELECT Наименование, Семестр, Количество часов

FROM Учебный\_план INNER JOIN Дисциплины ON

Учебный\_план.ID\_Дисциплина = Дисциплины.ID\_Дисциплина

WHERE Количество\_часов > 60

***Ключевое слово LEFT [OUTER]***

При использовании ключевого слова LEFT в результат будут включены все строки левой таблицы, независимо от того, есть для них соответствующая строка в правой таблице или нет. В случае от­сутствия строки в правой таблице для столбцов правой таблицы, включенных в результат выборки, устанавливается значение NULL. В приведенном ниже примере иллюстрируется использование клю­чевого слова LEFT [OUTER] для выборки данных.

SELECT Наименование, Семестр, Отчетность

FROM Дисциплины LEFT OUTER JOIN Учебный\_план ON

Учебный\_план.ID Дисциплина = Дисциплины.ID Дисциплина

WHERE (Наименование LIKE '%информатик%')

Как видно, по сравнению с использованием ключевого слова INNER, в результат запроса добавлена строка из таблицы «Дисци­плины», которая удовлетворяет сформулированному условию отбо­ра, но для которой не существует соответствующей строки в табли­це «Учебный\_план». В столбцах *Семестр* и *Отчетность* (относя­щихся к таблице «Учебный\_план») для этих строк установлено значение NULL.

***Ключевое слово RIGHT [OUTER]***

При использовании этого ключевого слова в результат будут включены все строки правой таблицы, независимо от того, есть ли для них соответствующая строка в левой таблице. Для соответст­вующих столбцов левой таблицы, включенных в запрос, устанавли­вается значение NULL. Приведем пример такого запроса:

SELECT Отчетность, Семестр, Наименование

FROM Учебный\_план RIGHT OUTER JOIN Дисциплины ON

Учебный план.ID Дисциплина = Дисциплины.ID Дисциплина

WHERE (Наименование LIKE '%информатик%')

Этот пример основывается на тех же данных, что и предыду­щий, но связь таблиц устанавливается в обратном порядке.

***Ключевое слово FULL [OUTER]***

При использовании ключевого слова FULL в результат будут включены все строки как правой, так и левой таблицы. Применение ключевого слова FULL [OUTER] можно рассматривать как одно­временное применение ключевых слов LEFT [OUTER] и RIGHT [OUTER].

**Раздел WHERE**

Раздел WHERE предназначен для наложения вертикальных фильтров на данные, обрабатываемые запросом. Другими словами, с помощью раздела WHERE можно сузить набор строк, включаемых в результат выборки. Для этого указывается логическое условие, от которого зависит, будет ли строка включена в выборку по запросу, или нет. Строка включается в результат выборки, только если логи­ческое выражение возвращает значение TRUE.

В общем случае логическое выражение содержит имена столб­цов таблиц, с которыми работает запрос. Для каждой строки, возвращенной запросом, вычисляется логическое выражение путем подстановки вместо имен столбцов конкретных значений из соот­ветствующей строки. Если при вычислении выражения возвращает­ся значение TRUE, то есть выражение истинно, то строка будет включена в конечный результат. В противном случае строка в ре­зультат не включается. При необходимости можно указать более од­ного логического выражения, объединив их с помощью логических операторов OR и AND.

Рассмотрим синтаксис раздела WHERE.

WHERE <условие\_отбора>

| <имя\_столбца> {= | \*= | =\*} <имя\_столбца>

В конструкции <условие\_отбора> можно определить любое ло­гическое условие, при выполнении которого строка будет включена в результат. Хотя и было сказано, что обычно логическое условие содержит имена столбцов, оно может быть и произвольным, в том числе и совсем не связанным с данными. Например, в следующей команде условие WHERE никогда не выполнится и ни одна строка не будет возвращена:

SELECT \* FROM Дисциплины WHERE 3=5

Приведенный пример демонстрирует логику работы раздела WHERE. Более удачное использование логического условия приве­дено в следующем примере:

SELECT Фамилия, Имя, Отчество, Номер Группы, Год\_поступления

FROM Студенты

WHERE Год\_поступления < 2000

Помимо операций сравнения (=, >, <, >=, <=) и логических операторов OR, AND, NOT при формировании условия отбора мо­гут быть использованы дополнительные логические операторы, рас­ширяющие возможности управления данными. Рассмотрим некото­рые из этих операторов.

***Оператор BETWEEN***

С помощью этого оператора можно определить, лежит ли значе­ние указанной величины в заданном диапазоне. Синтаксис исполь­зования оператора следующий:

<выражение> [NOT] BETWEEN <начало\_диапазона> AND <конец\_диапазона>

<Выражение> задает проверяемую величину, а аргументы <начало\_диапазона> и <конец\_диапазона> определяют возможные гра­ницы ее изменения. Использование оператора NOT совместно с оператором BETWEEN позволяет задать диапазон, *вне* которого мо­жет изменяться проверяемая величина.

При выполнении оператор BETWEEN преобразуется в конст­рукцию из двух операций сравнения:

(<выражение> >= <начало\_диапазона>)

AND (<выражение> <= <конец\_диапазона>)

Рассмотрим пример использования оператора BETWEEN:

SELECT Наименование, Семестр, Количество\_часов

FROM Учебный\_план INNER JOIN Дисциплины

ON Учебный план.ID Дисциплина = Дисциплины.ID\_Дисциплина

WHERE Количество\_часов BETWEEN 50 AND 100

***Оператор IN***

Оператор позволяет задать в условии отбора множество возмож­ных значений для проверяемой величины. Синтаксис использова­ния оператора следующий:

<выражение> [NOT] IN (<выpaжeниel>,...,<выpaжeниeN>)

<Выражение> указывает проверяемую величину, а аргументы <выражение1>,..., <выражениеN> задают перечислением через за­пятую набор значений, которые может принимать проверяемая ве­личина. Ключевое слово NOT выполняет логическое отрицание.

Рассмотрим пример применения оператора IN.

SELECT Наименование, Семестр, Количество часов

FROM Учебный\_план INNER JOIN Дисциплины

ON Учебный\_план.ID\_Дисциплина = Дисциплины.ID\_Дисциплина

WHERE Наименование IN ('Английский язык', 'Физическая культура')

***Оператор LIKE***

С помощью оператора LIKE можно выполнять сравнение выра­жения символьного типа с заданным шаблоном. Синтаксис опера­тора следующий:

<Символьное выражение> [NOT] LIKE <образец>

<Образец> задает символьный шаблон для сравнения и заклю­чается в кавычки. Шаблон может содержать символы-разделители.

Допускается использование следующих символов-разделителей

|  |  |
| --- | --- |
| **Символы-разделители** | **Значение** |
| % | Может быть заменен в символьном выражении любым количест­вом произвольных символов. Например, образец '%кош%' позво­ляет отобрать слова: **'кошка', 'окошко', 'лукошко', 'кошма'** и т. п. |
| \_ | Может быть заменен в символьном выражении любым, но толь­ко одним символом. Например, образец **'программ\_'** позволяет отобрать слова: **'программа', 'программ', 'программы',** но не 'программист' или 'программой' |
| **[АВС0-9]** | Может быть заменен в символьном выражении только одним символом из указанного в квадратных скобках набора. Дефис ис­пользуется для указания диапазона. Например, образец любой последовательности символов, начинающейся с буквы латинско­го алфавита, может быть задан следующим образом: **'[A-Z]%'** |
| **[^АВС0-9]** | Может быть заменен в символьном выражении только одним символом, кроме тех, что указаны в квадратных скобках. Дефис используется для указания диапазона. Например, образец любой последовательности символов, которая не должна заканчиваться цифрой, может быть задан следующим образом: '%[^0-9]' |

Рассмотрим пример использования оператора:

SELECT Фамилия, Имя, Отчество, Должность

FROM Кадровый состав

WHERE Должность LIKE '%пр%'

***Связывание таблиц***

Раздел WHERE может быть использован для связывания таб­лиц. В этом случае условие связывания должно присоединяться к логическому выражению с помощью логической операции AND (логическое умножение).

Рассмотрим пример, уточняющий один из представленных выше:

SELECT Наименование, Семестр, Количество часов

FROM Учебный\_план INNER JOIN Дисциплины

ON Учебный план.ID\_Дисциплина = Дисциплины.ID\_Дисциплина

WHERE (Количество\_часов > 60) AND (Семестр = 1)

Перенесем условие связывания в логическое выражение:

SELECT Наименование, Семестр, Количество\_часов

FROM Учебный план, Дисциплины

WHERE (Учебный план.ID Дисциплина = Дисциплины.ID Дисциплина) AND (Количество\_часов > 60) AND (Семестр = 1)

Использование только условия связывания в разделе WHERE аналогично связыванию ключевым словом INNER в разделе FROM. Например, результаты следующих запросов одинаковы:

SELECT TOP 10 Наименование, Семестр, Количество\_часов

FROM Учебный\_план, Дисциплины

WHERE (Учебный\_план.ID\_Дисциплина = Дисциплины.ID Дисциплина)

SELECT TOP 10 Наименование, Семестр, Количество\_часов

FROM Учебный\_план INNER JOIN Дисциплины

ON Учебный план.ID Дисциплина = Дисциплины.ID\_Дисциплина

Содержимое обеих таблиц можно посмотреть с помощью сле­дующих запросов:

SELECT TOP 10 \*

FROM Учебный план

SELECT TOP 10 \*

FROM Дисциплины

Аналогом использования ключевых слов LEFT OUTER JOIN является указание в разделе WHERE условия с помощью символов \*=. Приведенные примеры возвращают одинаковый набор данных:

SELECT Наименование, Семестр, Отчетность

FROM Дисциплины LEFT OUTER JOIN Учебный\_план

ON Учебный\_план.ID\_Дисциплина = Дисциплины.ID\_Дисциплина

WHERE (Наименование LIKE '%информатик%')

SELECT Наименование, Семестр, Отчетность

FROM Дисциплины, Учебный план

WHERE (Учебный\_план.ID Дисциплина \*= Дисциплины.ID Дисциплина)

AND (Наименование LIKE '%информатик%')

Аналогом использования ключевых слов RIGHT OUTER JOIN является указание условия с помощью символов =\*. Приведенные примеры возвращают одинаковый набор данных:

SELECT Отчетность, Семестр, Наименование

FROM Учебный\_план RIGHT OUTER JOIN Дисциплины

ON Учебный\_план.ID\_Дисциплина = Дисциплины.ID\_Дисциплина

WHERE (Наименование LIKE '%информатик%')

SELECT Отчетность, Семестр, Наименование

FROM Учебный план, Дисциплины

WHERE (Учебный план.ID\_Дисциплина =\* Дисциплины.ID\_Дисциплина)

AND (Наименование LIKE '%информатик%')

Следует отметить, что при использовании специальных ключе­вых слов INNER | {LEFT | RIGHT | FULL} [OUTER] данные пред­ставляются по-иному, чем при указании условия WHERE. Скорость выполнения запроса в первом случае оказывается выше, поскольку организуется *связывание* данных, тогда как при использовании кон­струкции WHERE происходит их *фильтрация.* При выполнении за­просов на небольших наборах данных это не играет существенной роли, поэтому удобнее обращаться к конструкции WHERE из-за на­глядности и простоты синтаксиса этого варианта, но при построе­нии сложных запросов, выполняющих обработку тысяч строк, все же лучше использовать конструкцию связывания.

**Раздел ORDER BY**

Раздел ORDER BY предназначен для упорядочения набора дан­ных, возвращаемого после выполнения запроса.

Полный синтаксис раздела ORDER BY следующий:

ORDER BY {<условие\_сортировки> [ASC | DESC]} [,...,n]

Параметр <условие\_сортировки> требует задания выражения, в соответствии с которым будет осуществляться сортировка строк. В простейшем случае это выражение представляет собой имя столб­ца одного из источников данных запроса.

Следует отметить, что в выражении, в соответствии с которым осуществляется сортировка строк, могут использоваться и столбцы, не указанные в разделе SELECT, то есть не входящие в результат выборки.

Раздел ORDER BY разрешает использование ключевых слов ASC и DESC, с помощью которых можно явно указать, каким обра­зом следует упорядочить строки. При указании ключевого слова ASC данные будут отсортированы по возрастанию. Если необходи­мо отсортировать данные по убыванию, указывается ключевое сло­во DESC. По умолчанию используется сортировка по возрастанию.

Данные можно отсортировать по нескольким столбцам. Для этого необходимо ввести имена столбцов через запятую по порядку сортировки. Сначала данные сортируются по столбцу, имя которого было указано в разделе ORDER BY первым. Затем, если имеется множество строк с одинаковыми значениями в первом столбце, вы­полняется дополнительная сортировка этих строк по второму столбцу (внутри группы с одинаковым значением в первом столб­це) и т. д.

Приведем пример сортировки по двум столбцам:

SELECT TOP 20 Наименование, Семестр, Количество\_часов

FROM Учебный план, Дисциплины

WHERE (Учебный план.ID\_Дисциплина = Дисциплины.ID\_Дисциплина)

ORDER BY Семестр, Количество\_часов DESC

**Раздел GROUP BY**

Раздел GROUP BY позволяет выполнять группировку строк таб­лиц по определенным критериям. Для каждой группы можно вы­полнить специальные функции агрегирования, которые применяют­ся ко всем строкам в группе. Одним из примеров использования раздела GROUP BY является суммирование однотипных значений.

Синтаксис раздела GROUP BY следующий:

GROUP BY [ALL] <условие\_группировки> [,...,n]

При использовании группировки (раздела GROUP BY) на раз­дел SELECT накладываются дополнительные ограничения. В непо­средственном виде разрешается указание только имен столбцов, пе­речисленных в разделе GROUP BY, то есть тех столбцов, по кото­рым осуществляется группировка. Значения других столбцов не могут быть выведены в непосредственном виде, так как обычно ка­ждая группа содержит множество строк, а в результате выборки для каждой группы должно быть указано единственное значение. По­этому, чтобы вывести значения столбцов, не задающих критерии группировки, необходимо использовать функции агрегирования.

Аргумент <условие\_группировки> определяет условие группи­ровки. Обычно в качестве условия группировки указывается имя столбца, однако в общем случае разрешается использование и выра­жений, включающих ссылки на столбцы.

Функции агрегирования позволяют выполнять статистическую обработку данных, подсчитывая количество, сумму, среднее значе­ние и другие величины для всего набора данных. Во многих функ­циях агрегирования допускается использование ключевых слов ALL и DISTINCT. Ключевое слово ALL выполняет агрегирование всех строк исходного набора данных. При указании ключевого слова DISTINCT будет выполняться агрегирование только уникальных строк. Все повторяющиеся строки будут проигнорированы. По умолчанию выполняется агрегирование всех строк, то есть исполь­зуется ключевое слово ALL. Далее приведены описания некоторых функций агрегирования.

***AVG()***

Эта функция вычисляет *среднее значение* для указанного столбца Функция имеет следующий синтаксис:

AVG ([ALL | DISTINCT] <выражение>)

При выполнении группировки (GROUP BY) вычисляет среднее значение для каждой группы. Если группировка не используется, то вычисляет среднее по всему столбцу. Например:

SELECT AVG (Количество\_часов) FROM Учебный\_план

Теперь рассмотрим пример использования функции AVG со­вместно с разделом GROUP BY при выполнении группировки по столбцу *Семестр:*

SELECT Семестр, AVG (Количество\_часов)

FROM Учебный\_план

GROUP BY Семестр

***COUNT()***

Функция подсчитывает количество строк в группе (при выпол­нении группировки) или количество строк результата запроса. Син­таксис функции COUNT следующий:

COUNT ({[ALL | DISTINCT] <выражение>] | \*})

Параметр <выражение> в простейшем случае представляет со­бой имя столбца. Если обрабатываемая строка в соответствующем столбце содержит значение не NULL, то счетчик будет увеличен на единицу. Указание символа (\*) предписывает считать *общее ко­личество строк* независимо от того, содержат они значения NULL или нет.

Пример использования функции COUNT:

SELECT COUNT (\*) AS 'Всего сотрудников',

COUNT(Телефон) AS 'С домашним телефоном'

FROM Кадровый\_состав

Этот запрос подсчитывает общее количество строк в таблице, а также количество ненулевых значений в столбце *Телефон.*

Результат выполнения запроса:

Всего сотрудников С домашним телефоном

14 10

Пример использования функции COUNT() при выполнении группировки:

SELECT Должность, COUNT (\*)

FROM Кадровый состав

GROUP BY Должность

Данный запрос возвращает количество строк в каждой группе столбца *Должность:*

Должность

Ассистент 3

Доцент 4

Зав.каф. 2

Проф. 3

Ст.преп. 2

***MAX()***

Функция возвращает *максимальное* значение в указанном диапа­зоне. Эта функция может использоваться как в обычных запросах, так и в запросах с группировкой. Синтаксис функции следующий:

MAX ([ALL | DISTINCT] <выражение>)

Пример использования функции:

SELECT MAX (Количество\_часов),

МАХ (Количество\_часов/2)

FROM Учебный план

***MIN()***

Функция возвращает минимальное значение в указанном диапа­зоне. Синтаксис функции следующий:

MIN ([ALL | DISTINCT] <выражение>)

Пример использования функции:

SELECT MIN (Количество\_часов) FROM Учебный план

***SUM()***

Функция выполняет обычное суммирование значений в ука­занном диапазоне. В качестве такого диапазона может рассматри­ваться группа или весь набор строк (без использования раздела GROUP BY).

Синтаксис функции следующий:

SUM ([ALL | DISTINCT] <выражение>)

В качестве примера просто суммируем значения в столбце *Коли­чество часов:*

SELECT SUM (Количество\_часов), COUNT(\*),

SUM (Количество\_часов)/COUNT(\*),

AVG (Количество\_часов)

FROM Учебный\_план

Теперь вновь обратимся к разделу SELECT и приведем пример группировки значений таблицы «Учебный\_план». Произведем груп­пировку строк по семестрам (столбец *Семестр)* и подсчитаем общую нагрузку в часах за каждый семестр:

SELECT Семестр, SUM (Количество\_часов) AS 'Нагрузка'

FROM [Учебный\_план]

GROUP BY Семестр

Рассмотрим теперь запрос, подсчитывающий количество экза­менов в каждом семестре:

SELECT Семестр, COUNT(\*) AS 'Экзамены'

FROM [Учебный\_план]

WHERE Отчетность = 'э'

GROUP BY Семестр

Предложение группировки может со­держать ключевое слово ALL. Назначение этого слова следующее. Нередко при вы­полнении группировки используется раз­дел WHERE, то есть группировка должна выполняться не над всеми строками, а лишь над определенной частью строк. Ре­зультатом такого подхода может явиться то, что одна или более групп не будет со­держать ни одной строки. Если группа не содержит ни одной строки, то по умолча­нию эта группа не включается в результат выборки. Однако в некоторых ситуациях все же требуется, чтобы были выведены все группы, в том числе и не содержащие ни одной строки. Для этого и необходимо указывать в разделе GROUP BY ключевое слово ALL. В этом случае будет выводиться список всех групп, но для групп, не содержащих строк, не будут выполняться функции агрегирования.

Рассмотрим это на примере. Для начала выполним группировку без использования ключевого слова ALL, но с вертикальной фильт­рацией (с помощью раздела WHERE) — в таблице «Учебный\_план» посчитаем для каждого семестра количество дисциплин с нагрузкой более 60 часов:

SELECT Семестр, COUNT(\*) AS 'Количество часов > 60'

FROM [Учебный\_план]

WHERE Количество\_часов > 60

GROUP BY Семестр

Добавим в раздел GROUP BY ключевое слово ALL:

**Раздел COMPUTE**

Этот раздел предназначен для выполнения групповых операций над содержимым столбцов выборки. Групповые операции задаются с помощью функций агрегирования. Результат агрегирования выво­дится в отдельной строке после всех данных столбца.

Синтаксис раздела COMPUTE следующий:

COMPUTE <Функция\_агрегирования> (<столбец\_агрегирования>)}[,...,n]

[ BY <столбец\_группировки> [,...,n]]

Аргумент <столбец\_агрегирования> должен содержать имя агре­гируемого столбца. Этот столбец должен быть включен в результат выборки. Ключевое слово BY указывает, что результат вычисления следует сгруппировать. Следующий за этим ключевым словом аргу­мент <столбец\_группировки> содержит имя столбца, по которому будет производиться группировка. Результат необходимо предвари­тельно отсортировать по этому столбцу, то есть столбец должен быть указан в разделе ORDER BY. Приведем простой пример при­менения раздела COMPUTE для вычисления количества дисцип­лин, читаемых в семестре, и общей суммы часов:

SELECT Наименование, Семестр, Количество\_часов

FROM Учебный план, Дисциплины

WHERE (Учебный\_план.ID\_Дисциплина = Дисциплины.ID\_Дисциплина) AND (Семестр = 2)

COMPUTE SUM (Количество\_часов), COUNT(Семестр)

Рассмотрим пример группировки при использовании раздела COMPUTE (составление списков групп и вычисление количества студентов в группе):

SELECT Фамилия, Имя, Отчество, Номер\_Группы

FROM Студенты

ORDER BY Номер\_Группы

COMPUTE COUNT (Номер\_\_Группы)

BY Номер\_Группы

**Раздел UNION**

Раздел UNION служит для объединения результатов выборки, возвращаемых двумя и более запросами. Рассмотрим синтаксис раздела UNION:

<Спецификация\_Запроса\_1>

UNION [ALL]

<Спецификация\_Запроса\_2>

…

[UNION [ALL]]

<Спецификация Запроса\_ n>

Чтобы к результатам запросов можно было применить опера­цию объединения, они должны соответствовать следующим требо­ваниям:

•запросы должны возвращать одинаковый набор столбцов (причем необходимо гарантировать одинаковый порядок сле­дования столбцов в каждом из запросов);

•типы данных соответствующих столбцов второго и последую­щих запросов должны поддерживать неявное преобразование или совпадать с типом данных столбцов первого запроса;

•ни один из результатов не может быть отсортирован с помо­щью раздела ORDER BY (однако общий результат может быть отсортирован, как будет показано ниже).

Указание ключевого слова ALL предписывает включать в ре­зультат повторяющиеся строки. По умолчанию повторяющиеся строки в результат не включаются.

Продемонстрируем применение раздела UNION. Рассмотрим таблицы «Кадровый\_Состав» и «Студенты» и попробуем построить, например, общий список и учащихся, и преподавателей, номер те­лефона которых начинается на 120.

Сначала построим запрос для таблицы «Кадровый\_Состав»:

SELECT Фамилия, Имя, Отчество, Должность, Телефон

FROM Кадровый состав

WHERE Телефон LIKE '120%'

Затем построим запрос для таблицы «Студенты»:

SELECT Фамилия, Имя, Отчество, Телефон

FROM Студенты

WHERE Телефон LIKE '120%'

Теперь объединим два запроса, чтобы в результате получить единую таблицу. Заметим, что столбец *Должность* отсутствует в таб­лице «Студенты». Чтобы в обшей таблице выделить студентов, вве­дем в запрос для таблицы «Студенты» столбец, содержащий стро­ку — константу «Студент» для всех записей, и объединим два запро­са с помощью раздела UNION:

SELECT Фамилия, Имя, Отчество, Должность, Телефон

FROM Кадровый\_состав

WHERE Телефон LIKE '120%'

UNION

SELECT Фамилия, Имя, Отчество, Новый\_столбец = 'Студент', Телефон

FROM Студенты

WHERE Телефон LIKE '120%'

При объединении таблиц столбцам итогового набора данных всегда присваиваются те же имена, что были указаны в первом из объединяемых запросов.

Упорядочим полученный список по алфавиту, добавив предло­жение ORDER BY:

SELECT Фамилия, Имя, Отчество, Должность, Телефон

FROM Кадровый\_состав

WHERE Телефон LIKE '120%'

UNION

SELECT Фамилия, Имя, Отчество, Новый столбец = 'Студент', Телефон

FROM Студенты

WHERE Телефон LIKE '120%'

ORDER BY Фамилия

**Раздел INTO. Использование команды SELECT...INTO**

При указании этой конструкции результат выполнения запроса будет сохранен в новой таблице. Синтаксис раздела INTO следую­щий:

INTO <имя\_новой\_таблицы>

Аргумент <имя\_новой\_таблицы> определяет имя таблицы, в ко­торую будут вставлены результаты.

При выполнении запроса SELECT...INTO автоматически созда­ется новая таблица с нужной структурой и в нее заносится получен­ный набор строк. При этом в базе данных не должно существовать таблицы, имя которой совпадает с именем таблицы, указанной в ко­манде SELECT...INTO. Если необходимо быстро создать таблицу со структурой, позволяющей сохранить результат выполнения запроса, то лучшим выходом будет использование команды SELECT...INTO.

Синтаксис команды SELECT...INTO следующий:

SELECT {<имя столбца> [[AS] <псевдоним\_столбца>] [, ..., n]}

INTO <имя\_новой\_таблицы> FROM {<имя\_исходной таблицы> [,..., n]}

Приведенный вариант синтаксиса далеко не исчерпывает все возможности вставки данных с помощью команды SELECT...INTO. Допускаются практически все варианты синтаксиса запроса SELECT, то есть можно выполнять группировку, сортировку, объе­динение и т. д.

Рассмотрим назначение аргументов команды.

<имя\_столбца> [[AS] <псевдоним\_столбца>].

Аргумент <имя\_столбца> задает имя столбца таблицы, который будет включен в резуль­тат. Указанный столбец должен принадлежать одной из таблиц, пе­речисленных в списке FROM {<имя\_исходной\_таблицы> [,..., n]}. Если столбцы, принадлежащие разным таблицам, имеют одинако­вые имена, то для столбцов необходимо использовать псевдонимы. В противном случае произойдет попытка создать таблицу со столб­цами, имеющими одинаковые имена, что приведет к ошибке, и вы­полнение запроса будет прервано. Указание псевдонимов также обязательно для столбцов, значения в которых формируются на ос­нове вычисления выражений (по умолчанию такие столбцы не име­ют никакого имени, что недопустимо для таблицы) и когда пользо­ватель хочет задать столбцам в создаваемой таблице новые имена (отличные от исходных). Имя псевдонима задается с помощью па­раметра <псевдоним\_колонки>.

INTO <имя\_новой\_таблицы>.

Аргумент <имя\_новой\_таблицы> содержит имя создаваемой таблицы. Это имя должно быть уникаль­ным в пределах базы данных.

FROM {<имя\_исходной\_таблицы> [,..., n]}.

В простейшем слу­чае конструкция FROM содержит список исходных таблиц. В более сложных запросах с помощью этой конструкции определяются ус­ловия связывания двух и более таблиц.

С помощью команды SELECT..INTO, например, можно разде­лить таблицу «Студенты» на две, выделив в отдельную таблицу «Контакты» адреса и телефоны, а затем удалив эти столбцы из таб­лицы «Студенты»:

SELECT ID\_Студент, Адрес, Телефон

INTO Контакты

FROM Студенты

Запрос для таблицы «Контакты»:

SELECT \*

FROM Контакты

WHERE Телефон LIKE '120%'

Построим внешний ключ для таблицы «Контакты», обеспечив связь с таблицей «Студенты»:

ALTER TABLE Контакты ADD CONSTRAINT FK\_Kонтакт

FOREIGN KEY (ID\_Студент)

REFERENCES Студенты

Модифицируем запрос для таблицы «Контакты»:

SELECT \*

FROM Студенты

INNER JOIN Контакты

ON Студенты.ID\_Студент = Контакты. ID\_Студент

WHERE Телефон LIKE '120%'

***Добавление данных — команда INSERT***

Рассмотрим некоторые возможности заполнения таблиц. Данные в таблицу могут быть внесены различными способами:

• с помощью команды INSERT. Используя команду INSERT, можно добавить как одну строку, так и множество строк;

• с помощью команды SELECT INTO. В этом случае на основе результата выборки, возвращаемого запросом, автоматически создается новая таблица (аппарат использования команды рассмотрен выше).

Рассмотрим процесс внесения данных в таблицу с помощью ко­манды INSERT. Как уже было сказано, эта команда может быть ис­пользована для вставки как одной, так и множества строк.

***Вставка одной строки***

В простейшем случае вставка данных с помощью команды INSERT предполагает использование конструкции INSERT-VALUES:

INSERT [INTO] <имя\_таблицы> [(<список колонок>)] VALUES (<список\_значений>)

С помощью этой команды можно добавить только одну строку.

Аргумент <имя\_таблицы> идентифицирует имя таблицы, в ко­торую необходимо вставить строку данных. Необязательный пара­метр <список\_столбцов> задает имена столбцов, в которые будет производиться добавление данных.

Рассмотрим процесс добавления данных в таблицу «Сводная\_ведомость». Каждая строка этой таблицы содержит результат сдачи экзамена (зачета) по отдельной дисциплине отдельным сту­дентом. Если студент, ID\_*Студент* которого равен 10, сдал экзамен по дисциплине со значением 3 в столбце ID*\_Дисциплина* на оценку «пять», то команда добавления этих данных в таблицу «Сводная\_ведомость» выглядит следующим образом:

INSERT Сводная\_векомость VALUES (10, 3, 5)

Для назначения произвольного порядка и состава столбцов в этом случае можно использовать следующую команду:

INSERT INTO Сводная\_ведомость (ID\_Дисциплина, ID\_Студент) VALUES (3, 10)

Если для столбца *Оценка* определено значение по умолчанию или разрешено хранение значений NULL, то значение для этого столбца можно вообще не указывать.

Мы рассматривали вставку строк в таблицу, значения для которых были заданы с помощью констант. Однако вставляемые значе­ния можно идентифицировать и с помощью переменных, функций, а также любых сложных выражений. Единственным требованием является совпадение типов данных столбца и значения, возвращае­мого выражением.

***Вставка результата запроса***

Приведем упрощенный синтаксис команды INSERT:

INSERT [INTO]

<имя таблицы>

{[(<список\_колонок>)]

{VALUES

( { DEFAULT | NULL | <выражение>} [, ..., n] )

| <результирующая\_таблица>

}

}

| DEFAULT VALUES

Рассмотрим назначение каждого из аргументов.

INTO — дополнительное ключевое слово, которое может быть использовано между словом INSERT и именем таблицы для обозна­чения того, что следующий параметр является именем таблицы, в которую будут вставлены данные;

<имя\_таблицы> — имя таблицы, в которую необходимо вста­вить данные;

<список\_столбцов> — содержит список столбцов, в которые бу­дет производиться вставка данных. Если он опущен, то данные бу­дут вставляться последовательно во все столбцы, начиная с первого. Значения для столбцов указываются после ключевого слова VALUES. Для каждого столбца должно быть задано выражение, имеющее соответствующий тип данных. Если список столбцов не указан, то количество значений VALUES должно соответствовать количеству столбцов таблицы. Если же список столбцов явно задан, то это определяет порядок значений VALUES (и, соответственно, их типы). Можно не указывать явно значения для столбцов, если для них определено значение по умолчанию или разрешено хранение значений NULL.

VALUES ({DEFAULT | NULL | <выражение>} [,..., n]) - опре­деляет набор данных, которые будут вставлены в таблицу. Количе­ство аргументов VALUES определяется количеством столбцов в таб­лице или количеством столбцов в списке (если таковой имеется). Для каждого столбца таблицы можно указать один из трех возмож­ных вариантов:

DEFAULT — будет вставлено значение по умолчанию, опреде­ленное для столбца. Если для столбца разрешено хранение значений NULL, а значение по умолчанию не определено, то в столбец будет вставлено значение NULL.

NULL — в столбец будет вставлено значение NULL. Естествен­но, вставка таких значений будет успешной, если для столбца была разрешена возможность хранения значений NULL. Следует пом­нить, что для столбцов, входящих в первичный ключ, возможность хранения значений NULL не предусмотрена.

<выражение> — задает значение, которое будет вставлено в столбец таблицы. Этот параметр должен иметь тот же тип данных, что и столбец, а также удовлетворять ограничениям целостности, определенным для соответствующего столбца.

<результирующая\_таблица> — этот параметр подразумевает указание запроса SELECT, с помощью которого будет формиро­ваться набор данных, вставляемых в таблицу. Количество столбцов, порядок их перечисления и их типы данных должны соответство­вать столбцам, указанным в списке <список\_столбцов>. Если по­следний отсутствует, то запрос должен возвращать значения для всех столбцов таблицы.

DEFAULT VALUES — при указании этого параметра строка будет содержать только значения по умолчанию. Если для столбца не уста­новлено значение по умолчанию, но разрешено хранение значений NULL, то в столбец будет вставлено значение NULL. Если же для столбца не разрешено хранение значений NULL, нет значения по умолчанию и в команде INSERT не указано значение для вставки, то будет выдано сообщение об ошибке и выполнение команды прервется.

Более сложный случай вставки данных предполагает использо­вание конструкции INSERT INTO...SELECT:

INSERT INTO <имя\_таблицы>

SELECT <выражение\_запроса>

Аргумент <имя\_таблицы> содержит имя таблицы, в которую бу­дут вставляться выбранные данные. Таблица должна иметь соответ­ствующую структуру и быть предварительно создана.

<Выражение\_запроса> определяет тело запроса SELECT, с по­мощью которого производится выборка данных из одной или не­скольких таблиц.

Например, для выборки данных из таблицы «Сту­денты» обо всех студентах, поступивших в ВУЗ в 2000 г., и сохране­ния их в таблице «Студент\_2000» можно использовать такую последовательность инструкций:

CREATE TABLE Студент\_2000

(ID\_Студент\_2000 INTEGER NOT NULL,

Фамилия CHAR(30) NOT NULL,

Имя CHAR(15) NOT NULL,

Отчество CHAR(20) NOT NULL,

Адрес CHAR(30),

Телефон CHAR(8),

PRIMARY KEY (ID\_Студент\_2000))

INSERT INTO Студент\_2000

SELECT ID\_Студент, Фамилия, Имя, Отчество, Адрес, Телефон

FROM Студенты

WHERE Год\_поступления = 2000

Приведенный пример иллюстрирует вставку строк данных в таблицу на основе результата выполнения запроса, обращающегося к одной таблице. Более сложные запросы могут обращаться к мно­жеству таблиц одной или нескольких баз данных.

В качестве еще одного примера рассмотрим помещение в новую таблицу «Преподаватель-дисциплина» информации о том, какой преподаватель какую дисциплину ведет.

Для этого мы будем работать с тремя таблицами: «Кадровый\_состав», «Учебный\_план» и «Дисциплины». В первой таблице содер­жится список преподавателей, тогда как в третьей — список дис­циплин. С помощью таблицы «Учебный\_план» устанавливается связь «многие ко многим» между таблицами «Кадровый\_состав» и «Дисциплины».

Прежде чем приступать к вставке данных, необходимо создать таблицу, которая будет содержать интересующие нас данные. Поми­мо столбцов для хранения информации об имени и фамилии препо­давателя и названии дисциплины, предусмотрим столбцы для хра­нения идентификационных номеров преподавателей и дисциплин:

CREATE TABLE Преподаватель\_дисциплина

(ID\_Дисциплина INTEGER NOT NULL,

ID\_Преподаватель INTEGER NOT NULL,

Наименование CHAR (20) NOT NULL,

Фамилия CHAR(30) NOT NULL,

Имя CHAR(15) NOT NULL,

Отчество CHAR(20) NOT NULL,

Должность CHAR(20) NOT NULL)

Теперь вставим в созданную таблицу нужные нам данные, вы­полнив для этого следующий запрос:

INSERT INTO Преподаватель\_дисциплина

SELECT DISTINCT Дисциплины.ID\_\_Дисциплина, Кадровый\_состав.ID\_Преподаватель,

Наименование, Фамилия, Имя, Отчество, Должность

FROM Кадровый\_состав, Учебный\_план, Дисциплины

WHERE Кадровый\_состав.ID\_Преподаватель = Учебный план.ID Преподаватель

AND Дисциплины.ID Дисциплина = Учебный план.ID Дисциплина

***Изменение данных — команда UPDATE***

Для внесения изменений в данные таблиц служит команда UPDATE, позволяющая выполнять как простое обновление данных в столбце, так и сложные операции модификации данных во множестве строк таблицы. Рассмотрим упрощенный синтаксис этой команды:

UPDATE <имя\_таблицы>

SET {<имя\_колонки> = {<выражение> | DEFAULT | NULL}}[,...,n]

{[ FROM {<имя\_исходной таблицы>} [,...,n]]

[ WHERE <условие\_отбора>]}

Рассмотрим назначение каждого из аргументов.

<имя\_таблицы> — имя таблицы, в которой необходимо произ­вести изменение данных.

SET — с этого ключевого слова начинается блок, в котором оп­ределяется список изменяемых столбцов. За один вызов UPDATE можно изменить данные в нескольких столбцах множества строк одной таблицы.

<имя\_столбца> = {<выражение>|DEFAULT|NULL} — для каждого изменяемого столбца нужно задать значение, которое он примет после выполнения изменения. С помощью ключевого сло­ва DEFAULT можно присвоить столбцу значение, определенное для него по умолчанию. Можно также установить для столбца зна­чение NULL. Изменению подвергнутся все строки, удовлетворяю­щие критериям ограничения области действия запроса UPDATE, которые задаются с помощью раздела WHERE. При составлении выражения можно ссылаться на любые столбцы таблицы, включая изменяемые. При этом следует учитывать, что изменения в данные вносятся только после выполнения команды. Таким образом, при ссылке на изменяемые столбцы будут использоваться старые зна­чения.

FROM {<имя\_исходной\_таблицы>} — если при изменении дан­ных в таблице необходимо учесть состояние данных в других табли­цах, то эти источники данных необходимо указать в разделе FROM. Собственно источник данных описывается с помощью конструкции <имя\_исходной\_таблицы>.

WHERE <условие\_отбора> — назначение раздела WHERE, ис­пользуемого в запросе UPDATE, полностью соответствует назначе­нию, которое раздел имеет в запросе SELECT, т. е. с помощью раздела WHERE можно сузить диапазон строк, в которых будет выполняться изменение данных. Необходимо указать логическое условие, на осно­ве которого будет приниматься решение об изменении данных конкретной строки. Если в контексте значений строки указанное логиче­ское условие выполняется (т. е. возвращает значение TRUE), то дан­ные этой строки будут изменены. В противном случае изменение не выполняется. Предполагается, что логическое условие включает име­на столбцов изменяемой таблицы, однако это необязательно.

Приведем простейший пример изменения данных. Добавим в таблицу «Учебный\_план» по два часа в столбец *Количество часов* для дисциплин 1-го семестра с формой отчетности «экзамен».

Выведем сначала исходное состояние данных:

SELECT \*

FROM Учебный\_план

WHERE (Отчетность = ‘э’) AND (Семестр = 1)

Затем выполним изменения.

UPDATE Учебный\_план

SET *Количество часов = Количество часов + 2*

WHERE (Отчетность = 'э') AND (Семестр = 1)

SELECT \*

FROM Учебный\_план

WHERE (Отчетность = 'э') AND (Семестр = 1)

***Удаление данных — команда DELETE***

Удаление данных из таблицы выполняется построчно. За одну операцию можно выполнить удаление как одной строки, так и не­скольких тысяч строк. Если необходимо удалить из таблицы *все* данные, то можно удалить саму таблицу. Естественно, при этом будут удалены и все хранящиеся в ней данные. Однако этот способ следует использовать лишь в самых крайних случаях, так как поми­мо данных будет удалена и структура таблицы.

Чаще всего удаление данных выполняется с помощью команды DELETE, удаляющей строки таблицы. Синтаксис команды, чаще всего использующийся на практике, следующий:

DELETE <Имя\_таблицы>

[WHERE <Условие\_отбора>]

Таким образом, в большинстве случаев требуется указание лишь имени таблицы, из которой необходимо удалить данные, и логиче­ского условия, ограничивающего диапазон удаляемых строк. При­чем последнее вовсе не обязательно, и при отсутствии условия из таблицы будут удалены все имеющиеся строки. Как и при выборке и изменении строк, диапазон удаляемых строк формируется с помо­щью раздела WHERE, использование которого было подробно рас­смотрено ранее.

Пусть из таблицы «Учебный\_план» необходимо удалить дисцип­лины первого семестра с формой отчетности «зачет», т. е. строки, у которых значение в столбце *Отчетность* равно 'з'. Команда, кото­рая позволит выполнить эту функцию, имеет следующий вид:

DELETE Учебный\_\_план

WHERE (Отчетность = 'з') AND (Семестр = 1)