МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

А.С. Митин, К.И. Дадьков

Программное и информационное обеспечение

Курсовое проектирование

Учебное пособие для студентов инженерно-технических специальностей

Учебное электронное издание

Минск ◊ БНТУ ◊ 2008

УДК 025.17:371.385 (075.8)
ББК 76.400.6я7
С 88

Авторы:
А.С. Митин – ассистент кафедры «Стандартизация, метрология и информационные системы» БНТУ;
К.И. Дадьков – старший преподаватель кафедры «Стандартизация, метрология и информационные системы» БНТУ

Рецензенты:
В.Л. Гуревич – директор Белорусского государственного института стандартизации и сертификации (БелГИСС);
С.А. Новиков – заместитель директора по научной и инновационной работе Института прикладной физики НАН Беларуси, кандидат технических наук

Учебное пособие содержит данные о порядке выполнения курсовой работы по дисциплине «Программное и информационное обеспечение». Методические указания, приведенные в пособии, могут быть использованы для самостоятельной работы студентов как дневного, так и заочного отделений высших учебных заведений.

Белорусский национальный технический университет
пр-т Независимости, 65, г. Минск, Республика Беларусь
Тел.(017) 232-77-52 факс (017) 232-91-37
E-mail: kdadzkou@smis-bntu.com
http://www.smis-bntu.com
Регистрационный № _____

Содержание

Введение

На сегодняшний день, практически любая информационная система имеет в своем составе базу данных. База данных — это каким-то образом структурированная информация, а абсолютно любая информационная система как раз с информацией и оперирует. Соответственно, должны существовать (и существуют) эффективные механизмы представления, хранения и обработки информации. Этим как раз и занимается отрасль информационных технологий — базы данных.

Основные задачи, которые решаются в базах данных:

- как правильно представить информацию;
- как эффективно с представленной информацией работать.

Современные информационные системы, основанные на концепции банков данных и баз знаний, характеризуются большими объемами хранимой информации, их сложной организацией, необходимостью удовлетворять разнообразные требования пользователей. Важным компонентом этой концепции является единая методология проектирования баз данных. Базы данных, являясь информационной моделью непрерывно меняющегося реального мира, также должны меняться, чтобы адекватно отображать действительность. Поэтому для сопровождения и эксплуатации информационных систем требуется постоянное использование процедур проектирования баз данных.

Методология проектирования автоматизированных банков данных может рассматриваться как совокупность методов и средств, последовательное применение которых обеспечивает разработку проекта баз данных, удовлетворяющего заданным целям. Рассматриваемая методология позволяет пользователю лучше понять, как следует специфицировать требования к данным.

Следуя этой методологии, проектировщик может выполнять более глубокий и содержательный анализ требований к данным, осуществлять контроль и управление кодом проектирования.

Цель работы

Целью курсовой работы по дисциплине «Программное и информационное обеспечение» является закрепление студентами специальности «Метрология, стандартизация и сертификация» практических навыков, полученных при изучение основ проектирования баз данных в рамках данной учебной дисциплины. В процессе выполнения курсовой работы студенты должны усвоить теоретические основы организации баз данных, включая принципы построения на

концептуальном, логическом и физическом уровнях, научиться ставить и решать практические задачи проектирования и эксплуатации баз данных.

При выполнении курсовой работы моделируются этапы разработки баз данных и подготовки их сопроводительной документации. Успешное выполнение курсовой работы повышает степень готовности будущих молодых специалистов для решения реально существующих задач на организациях промышленных отраслей Республики Беларусь.

Исходным заданием на курсовую работу по дисциплине «Программное и информационное обеспечение» является предметная область для разработки профильной базы данных. В рамках курсового проектирования студенты должны детально изучить предметную область, спроектировать инфологическую модель базы данных, спроектировать даталогическую модель базы данных, разработать и реализовать базу данных, оформить программную документацию на разработанную базу данных.

1 Терминология

Сущность — множество объектов.

Объект должен иметь признаки (свойства, атрибуты), описывающие этот объект качественно и количественно.

Объекты должны иметь общие признаки, по которым можно утверждать, что объекты принадлежат одному и тому же множеству.

Объекты, принадлежащие одному множеству, должны иметь уникальные значения признаков — тем самым они различимы между собой в рамках одного и того же множества.

Например, сущность *человек* — множество всех людей на планете. Всех людей на планете можно объединить, например, по признаку ДНК (у каждого человека есть ДНК). Конкретным представителем (так называемым экземпляром сущности) множества *человек* являетесь — вы. Ваше ДНК имеет конкретную химическую формулу, отличную от других людей — так вас можно однозначно отличить от других людей.

Экземпляр сущности — конкретный объект (представитель) какой-то сущности, для которого определен первичный ключ (установлены значения атрибутов, входящих в первичный ключ).

Неключевые атрибуты экземпляра сущности *могут* быть определены (а могут и не быть определены).

Атрибут — поименованная характеристика (признак, свойство) сущности.

Наименование атрибута должно быть уникальным для конкретной сущности, но может быть одинаковым для различных сущностей (например, *цвет* может быть определен для многих сущностей: *собака*, *автомобиль*, *дым* и т.д.). Атрибут *цвет*, в свою очередь, может принимать ряд значений: красный, синий, банановый и т.д., однако каждому экземпляру сущности присваивается только одно значение атрибута.

Любой атрибут, в различном контексте, может выступать как отдельная сущность. Например, для сущности *автомобиль*, *цвет* является атрибутом. А, например, для художника, *цвет* может выступать как отдельная сущность с атрибутом *название*. Тогда сущность *картина* состоит (включает в себя) несколько *цветов*. И один и тот же *цвет* может присутствовать в одной или нескольких *картинах*.

Первичный ключ — минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Для сущности.

Связь — ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако одно из основных требований к организации базы данных — это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить между ними определенные связи.

1.1 Связи

Различают три типа связей:

- «один к одному» (1:1) — одному экземпляру сущности A соответствует 0 или 1 экземпляр сущности B и одному экземпляру сущности B соответствует 0 или 1 экземпляр сущности A;
- «один к многим» (1:N) — одному экземпляру сущности A соответствуют 0, 1 или несколько (N) экземпляров сущности B и одному экземпляру сущности B соответствует 0 или 1 экземпляр сущности A;
- «многие к многим» (N:M) — одному экземпляру сущности A соответствует 0, 1 или несколько (M) экземпляров сущности B и одному экземпляру сущности B соответствует 0, 1 или несколько (N) экземпляров сущности A.

1.2 Первичные и внешние ключи

Первичный ключ или возможный (потенциальный) ключ — это минимальный набор атрибутов, по значениям которых можно однозначно найти (определить) каждый экземпляр сущности. Каждая сущность обладает хотя бы одним возможным ключом. Один из них принимается за первичный ключ.

При выборе первичного ключа следует отдавать предпочтение несоставным ключам или ключам, составленным из минимального числа атрибутов. Нецелесообразно также использовать ключи с длинными текстовыми значениями (предпочтительнее использовать целочисленные атрибуты).

Например, для сущности *студент* можно установить в качестве первичного ключа атрибут *номер зачетной книжки* (он уникален в рамках одного учебного заведения), либо несколько атрибутов: *фамилию*, *имя*, *отчество*, *номер группы*. Поскольку может быть ситуация появления в одной группе двух студентов с одинаковым именем, отчеством и фамилией, придется включить в первичный ключ дополнительные атрибуты, например, домашний адрес.

Не допускается, чтобы первичный ключ (любой атрибут, участвующий в первичном ключе) принимал неопределенное значение. Иначе возникнет противоречивая ситуация: появится не обладающий индивидуальностью экземпляр сущности.

1.3 Внешние ключи

Связь между двумя сущностями осуществляется между атрибутами сущностей (причем тип данных связываемых атрибутов двух сущностей должен быть одинаковым). Следовательно, чтобы связь между сущностями была, значения атрибутов должны дублироваться, при этом, одна сущность определяется как родительская, другая — дочерняя. При этом значения связываемого атрибута дочерней сущности должны браться из множества значений связываемого атрибута родительской сущности, либо принимать неопределенное значение NULL. Связываемый атрибут в дочерней сущности называется **внешним ключом**.

1.4 Целостность данных

Целостность (от англ. integrity — нетронутость, неприкосновенность, сохранность, целостность) — понимается как правильность данных в любой момент времени. Но эта цель может быть достигнута лишь в определенных пределах: СУБД не может контролировать правильность каждого отдельного значения, вводимого в базу данных (хотя каждое значение можно проверить на правдоподобность). Например, нельзя обнаружить, что вводимое значение 5

(представляющее номер дня недели) в действительности должно быть равно 3. С другой стороны, значение 9 явно будет ошибочным и СУБД должна его отвергнуть. Однако для этого ей следует сообщить, что номера дней недели должны принадлежать набору (1,2,3,4,5,6,7).

Поддержание целостности базы данных может рассматриваться как защита данных от неверных изменений или разрушений (не следует путать с незаконными изменениями и разрушениями, являющимися проблемой безопасности). Современные СУБД имеют ряд средств для обеспечения поддержания целостности (так же, как и средств обеспечения поддержания безопасности).

Выделяют три группы правил целостности:

1. Целостность по сущностям: не допускается, чтобы какой-либо атрибут, участвующий в первичном ключе, принимал неопределенное значение.
2. Целостность по ссылкам: значение внешнего ключа должно либо:

a. быть равным значению первичного ключа родительской сущности;
b. быть полностью неопределенным, т.е. каждое значение атрибута, участвующего во внешнем ключе должно быть неопределенным (NULL).

3. Целостность, определяемая пользователем: для любой конкретной базы данных существует ряд дополнительных специфических правил, которые относятся к ней одной и определяются разработчиком.

1.5 Нормализация

Нормализация — это разбиение таблицы на две или более, обладающих лучшими свойствами при включении, изменении и удалении данных.

Окончательная **цель нормализации** сводится к получению такого проекта базы данных, в котором *каждый факт появляется лишь в одном месте*, т.е. исключена избыточность информации. Это делается не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных.

1.5.1 Функциональная зависимость

Поле B таблицы функционально зависит от поля A той же таблицы в том и только в том случае, когда в любой заданный момент времени для каждого из различных значений поля A обязательно существует только одно из различных значений поля B. Отметим, что здесь допускается, что поля A и B могут быть составными.

Например, есть сущность *человек* с атрибутами *фамилия* и *пол*. Соответственно, если атрибут *фамилия* принимает значение *Иванов*, то атрибут *пол* может принимать значение только *мужской*. В таком случае говорят: атрибут *пол* функционально зависит от атрибута *фамилия*.

1.5.2 Полная функциональная зависимость

Поле B находится в полной функциональной зависимости от составного поля A, если оно функционально зависит от A и не зависит функционально от любого подмножества поля A.

1.5.3 Первая нормальная форма

Таблица находится в *первой нормальной форме* (*1НФ*) тогда и только тогда, когда ни одна из ее строк не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто.

1.5.4 Вторая нормальная форма

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

1.5.5 Третья нормальная форма

Таблица находится в *третьей нормальной форме* (*3НФ*), если она удовлетворяет определению 2НФ и не одно из ее неключевых полей не зависит функционально от любого другого неключевого поля.

1.5.6 Четвертая нормальная форма

Таблица находится в *нормальной форме Бойса-Кодда* (*НФБК*), если и только если любая функциональная зависимость между его полями сводится к полной функциональной зависимости от *возможного* ключа.

1.5.7 Пятая нормальная форма

Таблица находится в *пятой нормальной форме* (*5НФ*) 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.

2 Теория

2.1 Основные понятия БД и СУБД

Взаимосвязанные данные, которые позволяют описать ту или иную реальную систему, называются **информационной системой**. Каждая информационная система ориентирована на конкретную предметную область, которую она описывает.

Основная особенность СУБД — это наличие процедур для ввода и хранения не только самих данных, но и описаний их структуры. Файлы, снабженные описанием хранимых в них данных и находящиеся под управлением СУБД, стали называть банки данных, а затем «Базы данных» (БД).

2.2 Модели данных

Любая СУБД основывается на конкретной модели данных. Модель данных отражает взаимосвязи между объектами, описываемыми в БД. Компонентами в модели данных являются объекты и их взаимосвязи. В настоящее время имеется три основные модели данных:

- иерархическая;
- сетевая;
- реляционная.

Основное различие между указанными выше типами моделей данных состоит в способах представления взаимосвязей:

- между объектами;
- между атрибутами одного и того же объекта;
- между атрибутами различных объектов.

2.2.1 Задачи СУБД

СУБД должна предоставлять доступ к данным любым пользователям, включая и тех, которые практически не имеют и (или) не хотят иметь представления о:

- физическом размещении в памяти данных и их описаний;
- механизмах поиска запрашиваемых данных;
- проблемах, возникающих при одновременном запросе одних и тех же данных многими пользователями (прикладными программами);
- способах обеспечения защиты данных от некорректных обновлений и (или) несанкционированного доступа;

- поддержании баз данных в актуальном состоянии и множестве других функций СУБД.

2.3 Этапы разработки базы данных

2.3.1 Введение

Проект базы данных надо начинать с анализа предметной области и выявления требований к ней отдельных пользователей (сотрудников организации, для которых создается база данных). Проектирование обычно поручается человеку (группе лиц) — администратору базы данных (АБД). Им может быть как специально выделенный сотрудник организации, так и будущий пользователь базы данных, достаточно хорошо знакомый с машинной обработкой данных.

Объединяя частные представления о содержимом базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в будущих приложениях, АБД сначала создает обобщенное неформальное описание создаваемой базы данных. Это описание, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных, называют инфологической моделью данных.

Такая человеко-ориентированная модель полностью независима от физических параметров среды хранения данных. В конце концов, этой средой может быть память человека, а не ЭВМ. Поэтому инфологическая модель не должна изменяться до тех пор, пока какие-то изменения в реальном мире не потребуют изменения в ней некоторого определения, чтобы эта модель продолжала отражать предметную область.

Остальные модели, показанные на рисунке 1.1, являются компьютеро-ориентированными. С их помощью СУБД дает возможность программам и пользователям осуществлять доступ к хранимым данным лишь по их именам, не заботясь о физическом расположении этих данных. Нужные данные отыскиваются СУБД на внешних запоминающих устройствах по физической модели данных.

Рис. 1.1 – Уровни моделей данных

Так как указанный доступ осуществляется с помощью конкретной СУБД, то модели должны быть описаны на языке описания данных этой СУБД. Такое описание, создаваемое АБД по инфологической модели данных, называют даталогической моделью данных.

Трехуровневая архитектура (инфологический, даталогический и физический уровни) позволяет обеспечить независимость хранимых данных от использующих их программ. АБД может при необходимости переписать хранимые данные на другие носители информации и (или) реорганизовать их физическую структуру, изменив лишь физическую модель данных. АБД может подключить к системе любое число новых пользователей (новых приложений), дополнив, если надо, даталогическую модель. Указанные изменения физической и даталогической моделей не будут замечены существующими пользователями системы (окажутся «прозрачными» для них), так же как не будут замечены и новые пользователи. Следовательно, независимость данных обеспечивает возможность развития системы баз данных без разрушения существующих приложений.

2.3.2 Цель разработки БД

Целью разработки любой базы данных является хранение и использование информации о какой-либо предметной области.

2.3.3 Уровни моделирования

При разработке базы данных обычно выделяется несколько уровней моделирования, при помощи которых происходит переход от предметной области к конкретной реализации базы данных средствами конкретной СУБД. Можно выделить следующие уровни:

- сама предметная область;
- модель предметной области;
- логическая модель данных;
- физическая модель данных;
- собственно база данных и приложения.

2.3.4 Предметная область

Это часть реального мира, данные о которой мы хотим отразить в базе данных. Например, в качестве предметной области можно выбрать бухгалтерию какого-либо предприятия, отдел кадров, банк, магазин и т.д. Предметная область бесконечна и содержит как существенно важные понятия и данные, так и малозначащие или вообще не значащие данные. Так, если в качестве предметной области выбрать *учет товаров на складе*, то понятия *накладная* и *счет-фактура* являются существенно важными понятиями, а то, что *сотрудница*, принимающая накладные, *имеет двоих детей* — это для учета товаров неважно. Однако, с точки зрения отдела кадров данные о наличии детей являются существенно важными. Таким образом, вывод: **важность данных зависит от выбора предметной области**.

2.3.5 Модель предметной области

Модель предметной области — это наши знания о предметной области. Знания могут быть как в виде неформальных знаний в мозгу эксперта, так и выражены формально при помощи каких-либо средств. В качестве таких средств могут выступать текстовые описания предметной области, наборы должностных инструкций, правила ведения дел в компании и т.п. Опыт показывает, что текстовый способ представления модели предметной области крайне неэффективен. Гораздо более информативными и полезными при разработке баз данных являются описания предметной области, выполненные при помощи специализированных графических нотаций.

Имеется большое количество методик описания предметной области. Из наиболее известных можно назвать методику структурного анализа SADT и основанную на нем IDEF0, диаграммы потоков данных Гейна-Сарсона, методику объектно-ориентированного анализа UML, и др.

Модель предметной области описывает скорее процессы, происходящие в предметной области и данные, используемые этими процессами. От того, насколько правильно смоделирована предметная область, зависит успех дальнейшей разработки приложений.

2.3.6 Логическая модель данных

На следующем, более низком уровне находится логическая модель данных предметной области. Логическая модель описывает понятия (сущности) предметной области, их взаимосвязь, а также ограничения на данные, налагаемые предметной областью. Примеры понятий — *сотрудник*, *отдел*, *проект*, *зарплата*. Примеры взаимосвязей между понятиями — *«сотрудник числится ровно в одном отделе»*, *«сотрудник может выполнять несколько проектов»*, *«над одним проектом может работать несколько сотрудников»*. Примеры ограничений — *«возраст сотрудника не менее 16 и не более 60 лет»*.

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но без привязки к конкретной СУБД. Более того, логическая модель данных необязательно должна быть выражена средствами именно реляционной модели данных. Основным средством разработки логической модели данных в настоящий момент являются различные варианты ER-диаграмм (*Entity-Relationship*, диаграммы *сущность-связь*).

Решения, принятые на предыдущем уровне, при разработке модели предметной области, определяют некоторые границы, в пределах которых можно развивать логическую модель данных, в пределах же этих границ можно принимать различные решения. Например, модель предметной области *складского учета* содержит понятия *склад*, *накладная*, *товар*. При разработке соответствующей реляционной модели эти термины обязательно должны быть использованы, но различных способов реализации тут много — можно создать одну сущность, в которой будут присутствовать в качестве атрибутов *склад*, *накладная*, *товар*, а можно создать три отдельных сущности, по одному на каждое понятие.

2.3.6.1 Инфологическая модель данных «Сущность-связь»

2.3.6.1.1 Цель

Цель инфологического моделирования — обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Поэтому инфологическую модель данных пытаются строить по аналогии с естественным

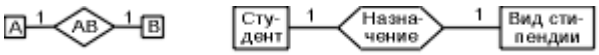
языком (последний не может быть использован в чистом виде из-за сложности компьютерной обработки текстов и неоднозначности любого естественного языка). Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты).

2.3.6.1.2 Характеристика связей и язык моделирования

При построении инфологических моделей можно использовать язык ER-диаграмм (от англ. Entity-Relationship, т.е. сущность-связь). В них сущности изображаются помеченными прямоугольниками, ассоциации — помеченными ромбами или шестиугольниками, атрибуты — помеченными овалами, а связи между ними — ненаправленными ребрами, над которыми может проставляться степень связи (1 или буква, заменяющая слово «много») и необходимое пояснение.

Между двумя сущностям, например, A и B возможны три вида связей.

Первый тип — связь *один-к-одному* (1:1): одному экземпляру сущности A соответствует 0 или 1 экземпляр сущности B и одному экземпляру сущности B соответствует 0 или 1 экземпляр сущности A:



Второй тип — связь *один-ко-многим* (1:N): одному экземпляру сущности A соответствуют 0, 1 или несколько (N) экземпляров сущности B и одному экземпляру сущности B соответствует 0 или 1 экземпляр сущности A.

Третий тип — *многие-ко-многим* (N:M): одному экземпляру сущности A соответствует 0, 1 или несколько экземпляров сущности B и одному экземпляру сущности B соответствует 0, 1 или несколько экземпляров сущности A.

Характер связей между сущностями не ограничивается перечисленными. Существуют и более сложные связи:

- множество связей между одними и теми же сущностями;
- тернарные связи;
- связи более высоких порядков, семантика (смысл) которых иногда очень сложна.

2.3.7 Физическая модель данных

На еще более низком уровне находится физическая модель данных. Физическая модель данных описывает данные средствами конкретной СУБД.

Сущности, разработанные на стадии формирования логической модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц, для ключевых атрибутов создаются уникальные индексы, домены преображаются в типы данных, принятые в конкретной СУБД. Связи становятся внешними ключами.

Ограничения, имеющиеся в логической модели данных, реализуются различными средствами СУБД, например, при помощи индексов, декларативных ограничений целостности, триггеров, хранимых процедур. При этом опять-таки решения, принятые на уровне логического моделирования определяют некоторые границы, в пределах которых можно развивать физическую модель данных. Точно также, в пределах этих границ можно принимать различные решения. Например, отношения, содержащиеся в логической модели данных, должны быть преобразованы в таблицы, но для каждой таблицы можно дополнительно объявить различные индексы, повышающие скорость обращения к данным. Здесь многое зависит от конкретной СУБД.

2.3.8 Собственно база данных и приложения

Как результат предыдущих этапов появляется собственно сама база данных. База данных реализована на конкретной программно-аппаратной основе, и выбор этой основы позволяет существенно повысить скорость работы с базой данных. Например, можно выбирать различные типы компьютеров, менять количество процессоров, объем оперативной памяти, дисковые подсистемы и т.п. Очень большое значение имеет также настройка СУБД в пределах выбранной программно-аппаратной платформы.

Но опять решения, принятые на предыдущем уровне — уровне физического проектирования,— определяют границы, в пределах которых можно принимать решения по выбору программно-аппаратной платформы и настройки СУБД.

2.3.9 Вывод

Решения, принятые на каждом этапе моделирования и разработки базы данных, будут сказываться на дальнейших этапах. Поэтому особую роль играет принятие правильных решений на ранних этапах моделирования.

3 Элементы языка SQL

3.1 Введение

Язык SQL (structured query language — структурированный язык запросов) стал фактически стандартным языком доступа к базам данных. Все СУБД, претендующие на название «реляционные», реализуют тот или иной диалект SQL. Многие нереляционные системы также имеют в настоящее время средства доступа к реляционным данным. Целью стандартизации является переносимость приложений между различными СУБД.

Нужно заметить, что в настоящее время ни одна система не реализует стандарт SQL в полном объеме. Кроме того, во всех диалектах языка имеются возможности, не являющиеся стандартными. Таким образом, можно сказать, что каждый диалект — это надмножество некоторого подмножества стандарта SQL. Это затрудняет переносимость приложений, разработанных для одних СУБД в другие СУБД.

Язык SQL оперирует терминами, несколько отличающимися от терминов реляционной теории, например, вместо «отношений» используются «таблицы», вместо «кортежей» — «строки», вместо «атрибутов» — «колонки» или «столбцы».

Стандарт языка SQL, хотя и основан на реляционной теории, но во многих местах отходит от нее.

Язык SQL является реляционно полным. Это означает, что любой оператор реляционной алгебры может быть выражен подходящим оператором SQL.

3.2 Цель

База данных — это хранилище, задачей которой является непосредственно хранение данных.

СУБД — позволяет к этим данным обращаться.

Чтобы обращаться к данным, необходим определенный язык — так называемый язык запросов.

Язык запросов — существует с целью управлять структурой БД и данными в БД — изменять с течением времени (добавлять/изменять/удалять).

3.3 Операторы SQL

Основу языка SQL составляют операторы, условно разбитые на несколько групп по выполняемым функциям.

Можно выделить следующие группы операторов (перечислены не все операторы SQL):

3.3.1 Операторы DDL (Data Definition Language) — операторы определения объектов базы данных

- CREATE SCHEMA — создать схему базы данных;
- DROP SHEMA — удалить схему базы данных;
- CREATE TABLE — создать таблицу;
- ALTER TABLE — изменить таблицу;
- DROP TABLE — удалить таблицу;
- CREATE VIEW — создать представление;
- DROP VIEW — удалить представление.

3.3.2 Операторы DML (Data Manipulation Language) — операторы манипулирования данными

- SELECT — отобрать строки из таблиц;
- INSERT — добавить строки в таблицу;
- UPDATE — изменить строки в таблице;
- DELETE — удалить строки в таблице;
- COMMIT — зафиксировать внесенные изменения;
- ROLLBACK — откатить внесенные изменения.

3.3.3 Операторы защиты и управления данными

- CREATE ASSERTION — создать ограничение;
- DROP ASSERTION — удалить ограничение;
- GRANT — предоставить привилегии пользователю или приложению на манипулирование объектами;
- REVOKE — отменить привилегии пользователя или приложения.

3.4 Пример использования операторов манипулирования данными

3.4.1 INSERT — вставка строк в таблицу

Пример 1. Вставка одной строки в таблицу:

```
INSERT INTO
  P (PNUM, PNAME)
  VALUES (4, «Иванов»);
```

Пример 2. Вставка в таблицу нескольких строк, выбранных из другой таблицы (в таблицу TMP_TABLE вставляются данные о поставщиках из таблицы P, имеющие номера, большие 2):

```
INSERT INTO
  TMP_TABLE (PNUM, PNAME)
  SELECT PNUM, PNAME
    FROM P
    WHERE P.PNUM>2;
```

3.4.2 UPDATE — обновление строк в таблице

Пример 3. Обновление нескольких строк в таблице:

```
UPDATE P
  SET PNAME = «Пушников»
  WHERE P.PNUM = 1;
```

3.4.3 DELETE — удаление строк в таблице

Пример 4. Удаление нескольких строк в таблице:

```
DELETE FROM P
  WHERE P.PNUM = 1;
```

Пример 5. Удаление всех строк в таблице:

```
DELETE FROM P;
```

3.5 Примеры использования оператора SELECT

Оператор SELECT является фактически самым важным для пользователя и самым сложным оператором SQL. Он предназначен для выборки данных из таблиц, т.е. он, собственно, и реализует одно их основных назначение базы данных — предоставлять информацию пользователю.

Оператор SELECT всегда выполняется над некоторыми таблицами, входящими в базу данных (причем в одну либо несколько).

На самом деле в базах данных могут быть не только постоянно хранимые таблицы, а также временные таблицы и так называемые представления (views). Представления — это просто хранящиеся в базе данные SELECT-выражения. С точки зрения пользователей представления — это таблица, которая не хранится постоянно в базе данных, а «возникает» в момент обращения к ней. С точки зрения оператора SELECT и постоянно хранимые таблицы, и временные таблицы и представления выглядят совершенно одинаково. Конечно, при реальном выполнении оператора SELECT системой учитываются различия между

хранимыми таблицами и представлениями, но эти различия скрыты от пользователя.

Результатом выполнения оператора SELECT всегда является таблица. Таким образом, по результатам действий оператор SELECT похож на операторы реляционной алгебры. Любой оператор реляционной алгебры может быть выражен подходящим образом сформулированным оператором SELECT. Сложность оператора SELECT определяется тем, что он содержит в себе все возможности реляционной алгебры, а также дополнительные возможности, которых в реляционной алгебре нет.

3.5.1 Отбор данных из одной таблицы

Пример 6. Выбрать все данные из таблицы поставщиков (ключевые слова ***SELECT***… ***FROM***…):

```
SELECT *
  FROM P;
```

Замечание. В результате получим новую таблицу, содержащую полную копию данных из исходной таблицы P.

Пример 7. Выбрать все строки из таблицы поставщиков, удовлетворяющих некоторому условию (ключевое слово ***WHERE***…):

```
SELECT *
  FROM P
  WHERE P.PNUM > 2;
```

Замечание. В качестве условия в разделе WHERE можно использовать сложные логические выражения, использующие поля таблиц, константы, сравнения (>, <, = и т.д.), скобки, союзы AND и OR, отрицание NOT.

Пример 8. Выбрать некоторые колонки из исходной таблицы (указание списка отбираемых колонок):

```
SELECT P.NAME
  FROM P;
```

Замечание. В результате получим таблицу с одной колонкой, содержащую все наименования поставщиков.

Замечание. Если в исходной таблице присутствовало несколько поставщиков с разными номерами, но одинаковыми наименованиями, то в результирующей таблице *будут строки с повторениями* — дубликаты строк автоматически не отбрасываются.

Пример 9. Выбрать некоторые колонки из исходной таблицы, удалив из результата повторяющиеся строки (ключевое слово ***DISTINCT***):

```
SELECT DISTINCT P.NAME
  FROM P;
```

Замечание. Использование ключевого слова DISTINCT приводит к тому, что в результирующей таблице будут удалены все повторяющиеся строки.

Пример 10. Использование скалярных выражений и переименований колонок в запросах (ключевое слово ***AS***…):

```
SELECT
    TOVAR.TNAME,
    TOVAR.KOL,
    TOVAR.PRICE,
    "=" AS EQU,
    TOVAR.KOL*TOVAR.PRICE AS SUMMA
  FROM TOVAR;
```

В результате получим таблицу с колонками, которых не было в исходной таблице TOVAR:

TNAME	KOL	PRICE	EQU	SUMMA
Болт	10	100	=	1000
Гайка	20	200	=	4000
Винт	30	300	=	9000

Пример 11.Упорядочение результатов запроса (ключевое слово ***ORDER BY***…):

```
SELECT
    PD.PNUM,
    PD.DNUM,
    PD.VOLUME
  FROM PD
  ORDER BY DNUM;
```

В результате получим следующую таблицу, упорядоченную по полю DNUM:

PNUM	DNUM	VOLUME
1	1	100
2	1	150

3	1	1000
1	2	200
2	2	250
1	3	300

Пример 12. Упорядочение результатов запроса по нескольким полям с возрастанием или убыванием (ключевые слова ***ASC***, ***DESC***):

```
SELECT
    PD.PNUM,
    PD.DNUM,
    PD.VOLUME
  FROM PD
  ORDER BY
    DNUM ASC,
    VOLUME DESC;
```

В результате получим таблицу, в которой строки идут в порядке возрастания значения поля DNUM, а строки, с одинаковым значением DNUM идут в порядке убывания значения поля VOLUME:

PNUM	DNUM	VOLUME
3	1	1000
2	1	150
1	1	100
2	2	250
1	2	200
1	3	300

Замечание. Если явно не указаны ключевые слова ASC или DESC, то по умолчанию принимается упорядочение по возрастанию (ASC).

3.5.2 Отбор данных из нескольких таблиц

Пример 13. Естественное соединение таблиц (способ 1 — явное указание условий соединения):

```
SELECT
    P.PNUM,
    P.PNAME,
```

```
    PD.DNUM,
    PD.VOLUME
  FROM P, PD
  WHERE P.PNUM = PD.PNUM;
```

В результате получим новую таблицу, в которой строки с данными о поставщиках соединены со строками с данными о поставках деталей:

PNUM	PNAME	DNUM	VOLUME
1	Иванов	1	100
1	Иванов	2	200
1	Иванов	3	300
2	Петров	1	150
2	Петров	2	250
3	Сидоров	1	1000

Замечание. Соединяемые таблицы перечислены в разделе FROM оператора, условие соединения приведено в разделе WHERE. Раздел WHERE, помимо условия соединения таблиц, может также содержать и условия отбора строк.

Пример 14. Естественное соединение таблиц (способ 2 — ключевые слова ***JOIN… USING…***):

```
SELECT
    P.PNUM,
    P.PNAME,
    PD.DNUM,
    PD.VOLUME
  FROM P JOIN PD USING PNUM;
```

Замечание. Ключевое слово USING позволяет *явно указать*, по каким из *общих* колонок таблиц будет производиться соединение.

Пример 15. Естественное соединение таблиц (способ 3 — ключевое слово ***NATURAL JOIN***):

```
SELECT
    P.PNUM,
    P.PNAME,
    PD.DNUM,
    PD.VOLUME
  FROM P NATURAL JOIN PD;
```

Замечание. В разделе FROM не указано, по каким полям производится соединение. NATURAL JOIN автоматически соединяет *по всем одинаковым полям* в таблицах.

Пример 16. Естественное соединение трех таблиц:

```
SELECT
    P.PNAME,
    D.DNAME,
    PD.VOLUME
  FROM
    P NATURAL JOIN PD NATURAL JOIN D;
```

В результате получим следующую таблицу:

PNAME	DNAME	VOLUME
Иванов	Болт	100
Иванов	Гайка	200
Иванов	Винт	300
Петров	Болт	150
Петров	Гайка	250
Сидоров	Болт	1000

Пример 17. Прямое произведение таблиц:

```
SELECT
    P.PNUM,
    P.PNAME,
    D.DNUM,
    D.DNAME
  FROM P, D;
```

В результате получим следующую таблицу:

PNUM	PNAME	DNUM	DNAME
1	Иванов	1	Болт
1	Иванов	2	Гайка
1	Иванов	3	Винт
2	Петров	1	Болт

2	Петров	2	Гайка
2	Петров	3	Винт
3	Сидоров	1	Болт
3	Сидоров	2	Гайка
3	Сидоров	3	Винт

Замечание. Т.к. не указано условие соединения таблиц, то *каждая* строка первой таблицы соединится с *каждой* строкой второй таблицы.

Пример 18. Соединение таблиц по произвольному условию. Рассмотрим таблицы поставщиков и деталей, которыми присвоен некоторый статую:

PNUM	PNAME	PSTATUS
1	Иванов	4
2	Петров	1
3	Сидоров	2

Таблица 1 – Отношение P (Поставщики)

DNUM	DNAME	DSTATUS
1	Болт	3
2	Гайка	2
3	Винт	1

Таблица 2 – Отношение D (Детали)

Ответ на вопрос «какие поставщики имеют право поставлять какие детали?» дает следующий запрос:

```
SELECT
    P.PNUM,
    P.PNAME,
    P.PSTATUS,
    D.DNUM,
    D.DNAME,
    D.DSTATUS
  FROM P, D
  WHERE P.PSTATUS >= D.DSTATUS;
```

В результате получим следующую таблицу:

PNUM	PNAME	PSTATUS	DNUM	DNAME	DSTATUS
1	Иванов	4	1	Болт	3
1	Иванов	4	2	Гайка	2
1	Иванов	4	3	Винт	1
2	Петров	1	3	Винт	1
3	Сидоров	2	2	Гайка	2
3	Сидоров	2	3	Винт	1

Остальные модели, показанные на рисунке 1.1, являются компьютеро-ориентированными. С их помощью СУБД дает возможность программам и пользователям осуществлять доступ к хранимым данным лишь по их именам, не заботясь о физическом расположении этих данных. Нужные данные отыскиваются СУБД на внешних запоминающих устройствах по физической модели данных.

3.5.3 Использование имен корреляции (алиасов, псевдонимов — aliases)

Иногда приходится выполнять запросы, в которых таблица соединяется сама с собой, или одна таблица соединяется дважды с другой таблицей. При этом используются имена корреляции (алиасы, псевдонимы), которые позволяют различать соединяемые копии таблиц. Имена корреляции вводятся в разделе FROM и идут через пробел после имени таблицы. Имена корреляции должны использоваться в качестве префикса перед именем столбца и отделяются от имени столбца точкой. Если в запросе указываются одни и те же поля из разных экземпляров одной таблицы, они должны быть переименованы для устранения неоднозначности в именованиях колонок результирующей таблицы. Определение имени корреляции действует только во время выполнения запроса.

Пример 19. Отобрать все пары поставщиков таким образом, чтобы первый поставщик в паре имел статус, больший статуса второго поставщика:

```
SELECT
    P1.PNAME AS PNAME1,
    P1.PSTATUS AS PSTATUS1,
    P2.PNAME AS PNAME2,
```

```
  P2.PSTATUS AS PSTATUS2
FROM
  P P1, P P2
WHERE P1.PSTATUS1 > P2.PSTATUS2;
```

В результате получим следующую таблицу:

PNAME1	PSTATUS1	PNAME2	PSTATUS2
Иванов	4	Петров	1
Иванов	4	Сидоров	2
Сидоров	2	Петров	1

Пример 20. Рассмотрим ситуацию, когда некоторые поставщики (назовем их контрагенты) могут выступать как в качестве поставщиков деталей, так и в качестве получателей. Таблицы, хранящие данные могут иметь следующий вид:

Номер контрагента NUM	Наименование контрагента NAME
1	Иванов
2	Петров
3	Сидоров

Таблица 3 – Отношение CONTRAGENTS

Номер детали DNUM	Наименование детали DNAME
1	Болт
2	Гайка
3	Винт

Таблица 4 – Отношение DETAILS (Детали)

Номер поставщика PNUM	Номер получателя CNUM	Номер детали DNUM	Поставляемое количество VOLUME
1	2	1	100
1	3	2	200
1	3	3	300

2	3	1	150
2	3	2	250
3	1	1	1000

Таблица 5 – Отношение CD (Поставки)

В таблице CD (поставки) поля PNUM и CNUM являются внешними ключами, ссылающимися на потенциальный ключ NUM в таблице CONTRAGENTS.

Ответ на вопрос «кто кому что в каком количестве поставляет» дается следующим запросом:

```
SELECT
    P.NAME AS PNAME,
    C.NAME AS CNAME,
    DETAILS.DNAME,
    CD.VOLUME
  FROM
    CONTRAGENTS P,
    CONTRAGENTS C,
    DETAILS,
    CD
  WHERE
    P.NUM = CD.PNUM AND
    C.NUM = CD.CNUM AND
    D.DNUM = CD.DNUM;
```

В результате получим следующую таблицу:

Наименование поставщика PNAME	**Наименование получателя CNAME**	**Наименование детали DNAME**	**Поставляемое количество VOLUME**
Иванов	Петров	Болт	100
Иванов	Сидоров	Гайка	200
Иванов	Сидоров	Винт	300
Петров	Сидоров	Болт	150
Петров	Сидоров	Гайка	250
Сидоров	Иванов	Болт	1000

Замечание. Этот же запрос может быть выражен очень большим количеством способов, например, так:

```
SELECT
    P.NAME AS PNAME,
    C.NAME AS CNAME,
    DETAILS.DNAME,
    CD.VOLUME
  FROM
    CONTRAGENTS P,
    CONTRAGENTS C,
    DETAILS NATURAL JOIN CD
  WHERE
    P.NUM = CD.PNUM AND
    C.NUM = CD.CNUM;
```

3.5.4 Использование агрегатных функций в запросах

Пример 21. Получить общее количество поставщиков (ключевое слово ***COUNT***):

```
SELECT COUNT(*) AS N
  FROM P;
```

В результате получим таблицу с одним столбцом и одной строкой, содержащей количество строк из таблицы P:

N
3

Пример 22. Получить общее, максимальное, минимальное и среднее количества поставляемых деталей (ключевые слова ***SUM***, ***MAX***, ***MIN***, ***AVG***):

```
SELECT
    SUM(PD.VOLUME) AS SM,
    MAX(PD.VOLUME) AS MX,
    MIN(PD.VOLUME) AS MN,
    AVG(PD.VOLUME) AS AV
  FROM PD;
```

В результате получим следующую таблицу с одной строкой:

SM	MX	MN	AV
2000	1000	100	333.33333333

3.5.5 Использование агрегатных функций с группировками

Пример 23. Для каждой детали получить суммарное поставляемое количество (ключевое слово ***GROUP BY***…):

```
SELECT
    PD.DNUM,
    SUM(PD.VOLUME) AS SM
  GROUP BY PD.DNUM;
```

Этот запрос будет выполняться следующим образом. Сначала строки исходной таблицы будут сгруппированы так, чтобы в каждую группу попали строки с одинаковыми значениями DNUM. Потом внутри каждой группы будет просуммировано поле VOLUME. От каждой группы в результирующую таблицу будет включена одна строка:

DNUM	SM
1	1250
2	450
3	300

Замечание. В списке отбираемых полей оператора SELECT, содержащего раздел GROUP BY можно включать *только* агрегатные функции и поля, *которые входят в условие группировки*. Следующий запрос выдаст синтаксическую ошибку:

```
SELECT
    PD.PNUM,
    PD.DNUM,
    SUM(PD.VOLUME) AS SM
  GROUP BY PD.DNUM;
```

Причина ошибки в том, что в список отбираемых полей включено поле PNUM, которое не входит в раздел GROUP BY. И действительно, в каждую полученную группу строк может входить несколько строк с различными значениями поля PNUM. Из каждой группы строк будет сформировано по одной итоговой строке. При этом нет однозначного ответа на вопрос, какое значение выбрать для поля PNUM в итоговой строке.

Замечание. Некоторые диалекты SQL не считают это за ошибку. Запрос будет выполнен, но предсказать, какие значения будут внесены в поле PNUM в результирующей таблице, невозможно.

Пример 24. Получить номера деталей, суммарное поставляемое количество которых превосходит 400 (ключевое слово ***HAVING***…):

Замечание. Условие, что суммарное поставляемое количество должно быть больше 400 не может быть сформулировано в разделе WHERE, т.к. в этом разделе нельзя использовать агрегатные функции. Условия, использующие агрегатные функции должны быть размещены в специальном разделе HAVING:

```
SELECT
    PD.DNUM,
    SUM(PD.VOLUME) AS SM
  GROUP BY PD.DNUM
  HAVING SUM(PD.VOLUME) > 400;
```

В результате получим следующую таблицу:

DNUM	SM
1	1250
2	450

Замечание. В одном запросе могут встретиться как условия отбора строк в разделе WHERE, так и условия отбора групп в разделе HAVING. Условия отбора групп нельзя перенести из раздела HAVING в раздел WHERE. Аналогично и условия отбора строк нельзя перенести из раздела WHERE в раздел HAVING, за исключением условий, включающих поля из списка группировки GROUP BY.

3.5.6 Использование подзапросов

Очень удобным средством, позволяющим формулировать запросы более понятным образом, является возможность использования подзапросов, вложенных в основной запрос.

Пример 25. Получить список поставщиков, статус которых меньше максимального статуса в таблице поставщиков (сравнение с подзапросом):

```
SELECT *
  FROM P
  WHERE P.STATUS < (SELECT MAX(P.STATUS) FROM P);
```

Замечание. Т.к. поле P.STATUS сравнивается с результатом подзапроса, то подзапрос должен быть сформулирован так, чтобы возвращать таблицу, состоящую *ровно из одной строки и одной колонки*.

Замечание. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

1. Выполнить *один раз* вложенный подзапрос и получить максимальное значение статуса.

2. Просканировать таблицу поставщиков P, каждый раз сравнивая значение статуса поставщика с результатом подзапроса, и отобрать только те строки, в которых статус меньше максимального.

Пример 26. Использование предиката ***IN***. Получить список поставщиков, поставляющих деталь номер 2:

```
SELECT *
  FROM P
  WHERE P.PNUM IN
    (SELECT DISTINCT PD.PNUM
        FROM PD
        WHERE PD.DNUM = 2);
```

Замечание. В данном случае вложенный подзапрос может возвращать таблицу, содержащую несколько строк.

Замечание. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

1. Выполнить *один раз* вложенный подзапрос и получить список номеров поставщиков, поставляющих деталь номер 2.

2. Просканировать таблицу поставщиков P, каждый раз проверяя, содержится ли номер поставщика в результате подзапроса.

Пример 27. Использование предиката ***EXIST***. Получить список поставщиков, поставляющих деталь номер 2:

```
SELECT *
  FROM P
  WHERE EXIST
    (SELECT *
        FROM PD
        WHERE
            PD.PNUM = P.PNUM AND
            PD.DNUM = 2);
```

Замечание. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

1. Просканировать таблицу поставщиков P, *каждый раз выполняя подзапрос* с новым значением номера поставщика, взятым из таблицы P.

2. В результат запроса включить только те строки из таблицы поставщиков, для которых вложенный подзапрос вернул непустое множество строк.

Замечание. В отличие от двух предыдущих примеров, вложенный подзапрос содержит параметр (внешнюю ссылку), передаваемый из основного запроса — номер поставщика P.PNUM. Такие подзапросы называются ***коррелируемыми*** (***correlated***). Внешняя ссылка может принимать различные значения для каждой строки-кандидата, оцениваемого с помощью подзапроса, поэтому подзапрос должен выполняться заново для каждой строки, отбираемой в основном запросе. Такие подзапросы характерны для предиката EXIST, но могут быть использованы и в других подзапросах.

Замечание. Может показаться, что запросы, содержащие коррелируемые подзапросы будут выполняться медленнее, чем запросы с некоррелируемыми подзапросами. На самом деле это не так, т.к. то, как пользователь, сформулировал запрос, *не определяет*, как этот запрос будет выполняться. Язык SQL является непроцедурным, а декларативным. Это значит, что пользователь, формулирующий запрос, просто описывает, *каким должен быть результат запроса*, а как этот результат будет получен — за это отвечает сама СУБД.

Пример 28. Использование предиката ***NOT EXIST***. Получить список поставщиков, не поставляющих деталь номер 2:

```
SELECT *
  FROM P
  WHERE NOT EXIST
    (SELECT *
        FROM PD
        WHERE
            PD.PNUM = P.PNUM AND
            PD.DNUM = 2);
```

Замечание. Также как и в предыдущем примере, здесь используется коррелируемый подзапрос. Отличие в том, что в основном запросе будут отобраны те строки из таблицы поставщиков, для которых вложенный подзапрос не выдаст ни одной строки.

Пример 29. Получить имена поставщиков, поставляющих все детали:

```
SELECT DISTINCT PNAME
  FROM P
  WHERE NOT EXIST
    (SELECT *
        FROM D
        WHERE NOT EXIST
          (SELECT *
              FROM PD
              WHERE
                  PD.DNUM = D.DNUM AND
                  PD.PNUM = P.PNUM));
```

Замечание. Данный запрос содержит два вложенных подзапроса и реализует реляционную операцию *деления отношений*.

Самый внутренний подзапрос параметризован двумя параметрами (D.DNUM, P.PNUM) и имеет следующий смысл: отобрать все строки, содержащие данные о поставках поставщика с номером PNUM детали с номером DNUM. Отрицание NOT EXIST говорит о том, что данный поставщик не поставляет данную деталь. Внешний к нему подзапрос, сам являющийся вложенным и параметризованным параметром P.PNUM, имеет смысл: отобрать список деталей, которые не поставляются поставщиком PNUM. Отрицание NOT EXIST говорит о том, что для поставщика с номером PNUM не должно быть деталей, которые не поставлялись бы этим поставщиком. Это в точности означает, что во внешнем запросе отбираются только поставщики, поставляющие все детали.

3.5.7 Использование объединения, пересечения и разности

Пример 30. Получить имена поставщиков, имеющих статус, больший 3 или поставляющих хотя бы одну деталь номер 2 (объединение двух подзапросов — ключевое слово ***UNION***):

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
  UNION
  SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
        PD.DNUM = 2;
```

Замечание. Результирующие таблицы объединяемых запросов должны быть совместимы, т.е. иметь одинаковое количество столбцов и одинаковые типы столбцов в порядке их перечисления. *Не требуется*, чтобы объединяемые таблицы имели бы одинаковые имена колонок. Это отличает операцию объединения запросов в SQL от операции объединения в реляционной алгебре. Наименования колонок в результирующем запросе будут автоматически взяты из результата первого запроса в объединении.

Пример 31. Получить имена поставщиков, имеющих статус, больший 3 и одновременно поставляющих хотя бы одну деталь номер 2 (пересечение двух подзапросов — ключевое слово ***INTERSECT***):

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
  INTERSECT
  SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
        PD.DNUM = 2;
```

Пример 32. Получить имена поставщиков, имеющих статус, больший 3, за исключением тех, кто поставляет хотя бы одну деталь номер 2 (разность двух подзапросов — ключевое слово ***EXCEPT***):

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
EXCEPT
  SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
        PD.DNUM = 2;
```

3.5.8 Синтаксис оператора выборки данных (SELECT)

3.5.8.1 BNF-нотация

Опишем синтаксис оператора выборки данных (оператора SELECT) более точно. При описании синтаксиса операторов обычно используются условные обозначения, известные как стандартные формы Бэкуса-Наура (BNF).

В BNF обозначениях используются следующие элементы:

- Символ «::=» означает равенство по определению. Слева от знака стоит определяемое понятие, справа — собственно определение понятия.
- Ключевые слова записываются прописными буквами. Они зарезервированы и составляют часть оператора.
- Метки-заполнители конкретных значений элементов и переменных записываются курсивом.
- Необязательные элементы оператора заключены в квадратные скобки [].
- Вертикальная черта | указывает на то, что все предшествующие ей элементы списка являются необязательными и могут быть заменены любым другим элементом списка после этой черты.
- Фигурные скобки {} указывают на то, что все находящееся внутри них является единым целым.
- Троеточие «…» означает, что предшествующая часть оператора может быть повторена любое количество раз.
- Многоточие, внутри которого находится запятая «.,..» указывает, что предшествующая часть оператора, состоящая из нескольких элементов, разделенных запятыми, может иметь произвольное число повторений. Запятую нельзя ставить после последнего элемента. Замечание: данное соглашение не входит в стандарт BNF, но позволяет более точно описать синтаксис операторов SQL.
- Круглые скобки являются элементом оператора.

3.5.8.2 Синтаксис оператора выборки

В довольно сильно упрощенном виде оператор выборки данных имеет следующий синтаксис (для некоторых элементов мы дадим не BNF-определения, а словесное описание):

Оператор выборки ::=
Табличное выражение
[**ORDER BY**
{{*Имя столбца-результата* [**ASC** | **DESC**]} | {*Положительное целое* [**ASC** | **DESC**]}},..];

Табличное выражение ::=
Select-выражение
[
{**UNION** | **INTERSECT** | **EXCEPT**} [**ALL**]
{*Select-выражение* | **TABLE** *Имя таблицы* | *Конструктор значений таблицы*}
]

Select-выражение ::=
SELECT [**ALL** | **DISTINCT**]
{{{*Скалярное выражение* | *Функция агрегирования* | *Select-выражение*} [**AS** *Имя столбца*]}.,..}
| {{*Имя таблицы*|*Имя корреляции*}.*}
| *
FROM {
{*Имя таблицы* [**AS**] [*Имя корреляции*] [(*Имя столбца*.,..)]}
| {*Select-выражение* [**AS**] *Имя корреляции* [(*Имя столбца*.,..)]}
| *Соединенная таблица* }.,..
[**WHERE** *Условное выражение*]
[**GROUP BY** {[{*Имя таблицы*|*Имя корреляции*}.]*Имя столбца*}.,..]
[**HAVING** *Условное выражение*]

Замечание. Select-выражение в разделе SELECT, используемое в качестве значения для отбираемого столбца, должно возвращать таблицу, состоящую из одной строки и одного столбца, т.е. скалярное выражение.

Замечание. Условное выражение в разделе WHERE должно вычисляться для каждой строки, являющейся кандидатом в результирующее множество строк. В этом условном выражении можно использовать подзапросы. Синтаксис условных выражений, допустимых в разделе WHERE рассматривается ниже.

Замечание. Раздел HAVING содержит условное выражение, вычисляемое для каждой группы, определяемой списком группировки в разделе GROUP BY. Это условное выражение может содержать функции агрегирования, вычисляемые для каждой группы. Условное выражение, сформулированное в разделе WHERE, может быть перенесено в раздел HAVING. Перенос условий из раздела HAVING в раздел WHERE невозможен, если условное выражение содержит агрегатные функции. Перенос условий из раздела WHERE в раздел HAVING является плохим стилем программирования — эти разделы предназначены для различных по смыслу условий (условия для строк и условия для групп строк).

Замечание. Если в разделе SELECT присутствуют агрегатные функции, то они вычисляются по-разному в зависимости от наличия раздела GROUP BY. Если раздел GROUP BY отсутствует, то результат запроса возвращает не более одной строки. Агрегатные функции вычисляются по всем строкам, удовлетворяющим условному выражению в разделе WHERE. Если раздел GROUP BY присутствует, то агрегатные функции вычисляются по отдельности для каждой группы, определенной в разделе GROUP BY.

Скалярное выражение — в качестве скалярных выражений в разделе SELECT могут выступать либо имена столбцов таблиц, входящих в раздел FROM, либо простые функции, возвращающие скалярные значения.

Функция агрегирования ::=
COUNT (*) |
{
{**COUNT** | **MAX** | **MIN** | **SUM** | **AVG**} ([**ALL** | **DISTINCT**] *Скалярное выражение*)
}

Конструктор значений таблицы ::=
VALUES *Конструктор значений строки*.,..

Конструктор значений строки ::=
Элемент конструктора | (*Элемент конструктора*.,..) | *Select-выражение*

Замечание. Select-выражение, используемое в конструкторе значений строки, обязано возвращать ровно одну строку.

Элемент конструктора ::=
Выражение для вычисления значения | **NULL** | **DEFAULT**

3.5.9 Синтаксис соединенных таблиц

В разделе FROM оператора SELECT можно использовать соединенные таблицы. Пусть в результате некоторых операций мы получаем таблицы A и B. Такими операциями могут быть, например, оператор SELECT или другая соединенная таблица. Тогда синтаксис соединенной таблицы имеет следующий вид:

Соединенная таблица ::=
Перекрестное соединение
| *Естественное соединение*
| *Соединение посредством предиката*
| *Соединение посредством имен столбцов*
| *Соединение объединения*

Тип соединения ::=
INNER
| **LEFT** [**OUTER**]
| **RIGTH** [**OUTER**]
| **FULL** [**OUTER**]

Перекрестное соединение ::=
Таблица A **CROSS JOIN** *Таблица B*

Естественное соединение ::=
Таблица А [**NATURAL**] [*Тип соединения*] **JOIN** *Таблица В*

Соединение посредством предиката ::=
Таблица А [*Тип соединения*] **JOIN** *Таблица В* **ON** Предикат

Соединение посредством имен столбцов ::=
Таблица А [*Тип соединения*] **JOIN** *Таблица В* **USING** (Имя столбца.,..)

Соединение объединения ::=
Таблица А **UNION JOIN** *Таблица В*

Опишем используемые термины.

CROSS JOIN — Перекрестное соединение возвращает просто декартово произведение таблиц. Такое соединение в разделе FROM может быть заменено списком таблиц через запятую.

NATURAL JOIN — Естественное соединение производится по всем столбцам таблиц А и В, имеющим одинаковые имена. В результирующую таблицу одинаковые столбцы вставляются только один раз.

JOIN … ON — Соединение посредством предиката соединяет строки таблиц А и В посредством указанного предиката.

JOIN … USING — Соединение посредством имен столбцов соединяет отношения подобно естественному соединению по тем общим столбцам таблиц А и Б, которые указаны в списке USING.

OUTER — Ключевое слово OUTER (внешний) не является обязательными, оно не используется ни в каких операциях с данными.

INNER — Тип соединения «внутреннее». Внутренний тип соединения используется по умолчанию, когда тип явно не задан. В таблицах А и В соединяются только те строки, для которых найдено совпадение.

LEFT (OUTER) — Тип соединения «левое (внешнее)». Левое соединение таблиц А и В включает в себя все строки из левой таблицы А и те строки из правой таблицы В, для которых обнаружено совпадение. Для строк из таблицы А, для которых не найдено соответствия в таблице В, в столбцы, извлекаемые из таблицы В, заносятся значения NULL.

RIGHT (OUTER) — Тип соединения «правое (внешнее)». Правое соединение таблиц А и В включает в себя все строки из правой таблицы В и те строки из левой таблицы А, для которых обнаружено совпадение. Для строк из таблицы В,

для которых не найдено соответствия в таблице A, в столбцы, извлекаемые из таблицы A заносятся значения NULL.

FULL (OUTER) — Тип соединения «полное (внешнее)». Это комбинация левого и правого соединений. В полное соединение включаются все строки из обеих таблиц. Для совпадающих строк поля заполняются реальными значениями, для несовпадающих строк поля заполняются в соответствии с правилами левого и правого соединений.

UNION JOIN — Соединение объединения является обратным по отношению к внутреннему соединению. Оно включает только те строки из таблиц A и B, для которых не найдено совпадений. В них используются значения NULL для столбцов, полученных из другой таблицы. Если взять полное внешнее соединение и удалить из него строки, полученные в результате внутреннего соединения, то получится соединение объединения.

Использование соединенных таблиц часто облегчает восприятие оператора SELECT, особенно, когда используется естественное соединение. Если не использовать соединенные таблицы, то при выборе данных из нескольких таблиц необходимо явно указывать условия соединения в разделе WHERE. Если при этом пользователь указывает сложные критерии отбора строк, то в разделе WHERE смешиваются семантически различные понятия — как условия связи таблиц, так и условия отбора строк.

3.5.10 Синтаксис условных выражений раздела WHERE

Условное выражение, используемое в разделе WHERE оператора SELECT должно вычисляться для каждой строки-кандидата, отбираемой оператором SELECT. Условное выражение может возвращать одно из трех значений истинности: TRUE, FALSE или UNKNOUN. Строка-кандидат отбирается в результатирующее множество строк только в том случае, если для нее условное выражение вернуло значение TRUE.

Условные выражения имеют следующий синтаксис (в целях упрощения изложения приведены не все возможные предикаты):

Условное выражение ::=
[( ] [**NOT**]
{*Предикат сравнения*
| *Предикат between*
| *Предикат in*
| *Предикат like*

| *Предикат null*
| *Предикат количественного сравнения*
| *Предикат exist*
| *Предикат unique*
| *Предикат match*
| *Предикат overlaps*}
[{**AND** | **OR**} *Условное выражение*] [ )]
[**IS** [**NOT**] {**TRUE** | **FALSE** | **UNKNOWN**}]

Предикат сравнения ::=
Конструктор значений строки {= | < | > | <= | >= | <>} *Конструктор значений строки*

Пример 33. Сравнение поля таблицы и скалярного значения:

```
POSTAV.VOLUME > 100
```

Пример 34. Сравнение двух сконструированных строк:

```
(PD.PNUM, PD.DNUM) = (1, 25)
```

Этот пример эквивалентен условному выражению

```
PD.PNUM = 1 AND PD.DNUM = 25
```

Предикат between ::=
Конструктор значений строки [**NOT**] **BETWEEN**
Конструктор значений строки **AND** *Конструктор значений строки*

Пример 35. PD.VOLUME BETWEEN 10 AND 100

Предикат in ::=
Конструктор значений строки [**NOT**] **IN**
{(*Select-выражение*) | (*Выражение для вычисления значения*.,..)}

Пример 36.

```
P.PNUM IN (SELECT PD.PNUM FROM PD WHERE PD.DNUM=2)
```

Пример 37.

```
P.PNUM IN (1, 2, 3, 5)
```

Предикат like ::=
Выражение для вычисления значения строки-поиска [**NOT**] **LIKE**
Выражение для вычисления значения строки-шаблона [**ESCAPE** *Символ*]

Замечание. Предикат LIKE производит поиск строки-поиска в строке-шаблоне. В строке-шаблоне разрешается использовать два трафаретных символа:

- Символ подчеркивания «_» может использоваться вместо любого единичного символа в строке-поиска,
- Символ процента «%» может заменять набор любых символов в строке-поиска (число символов в наборе может быть от 0 и более).

Предикат null ::=
Конструктор значений строки **IS** [**NOT**] **NULL**

Замечание. Предикат NULL применяется специально для проверки, не равно ли проверяемое выражение null-значению.

Предикат количественного сравнения ::=
Конструктор значений строки {= | < | > | <= | >= | <>}
{**ANY** | **SOME** | **ALL**} (*Select-выражение*)

Замечание. Кванторы ANY и SOME являются синонимами и полностью взаимозаменяемы.

Замечание. Если указан один из кванторов ANY и SOME, то предикат количественного сравнения возвращает TRUE, если сравниваемое значение совпадает *хотя бы с одним* значением, возвращаемом в подзапросе (select-выражении).

Замечание. Если указан квантор ALL, то предикат количественного сравнения возвращает TRUE, если сравниваемое значение совпадает *с каждым* значением, возвращаемом в подзапросе (select-выражении).

Пример 38.

```
P.PNUM = SOME (SELECT PD.PNUM FROM PD WHERE PD.DNUM=2)
```

Предикат exist ::=
EXIST (*Select-выражение*)

Замечание. Предикат EXIST возвращает значение TRUE, если результат подзапроса (select-выражения) не пуст.

Предикат unique ::=
UNIQUE (*Select-выражение*)

Замечание. Предикат UNIQUE возвращает TRUE, если в результате подзапроса (select-выражения) нет совпадающих строк.

Предикат match ::=
Конструктор значений строки **MATCH** [**UNIQUE**]
[**PARTIAL** | **FULL**] (*Select-выражение*)

Замечание. Предикат MATCH проверяет, будет ли значение, определенное в конструкторе строки совпадать со значением любой строки, полученной в результате подзапроса.

Предикат overlaps ::=
Конструктор значений строки **OVERLAPS** *Конструктор значений строки*

Замечание. Предикат OVERLAPS, является специализированным предикатом, позволяющем определить, будет ли указанный период времени перекрывать другой период времени.

4 Задания к выполнению курсовой работы

4.1 Порядок выполнения курсовой работы

При выполнении курсового проекта последовательность действий должна быть следующей:

1. Изучить основные теоретические разделы проектирования БД из методических указаний и рекомендованной литературы.
2. Изучить предметную область.
3. Спроектировать инфологическую модель.
4. Спроектировать даталогическую модель.
5. Разработать и реализовать базу данных.
6. Оформить пояснительную записку.

4.2 Типовые задания

- Система учета одежды, подлежащей обязательной сертификации.
- Система учета стиральных машин, подлежащих сертификации.
- Система учета средств связи, подлежащих сертификации.
- Система учета холодильников, подлежащих сертификации.
- Система учета компьютерной техники, подлежащей сертификации.
- Система учета продуктов питания, подлежащих сертификации.
- Система учета сертифицированных туристических операторов.
- Система учета сертифицированной медицинской техники.
- Система учета сертифицированных медицинских препаратов.
- Информационно-поисковая система технической литературы.
- Учебное заведение.

Литература


D:\work\BGPA\Пояснительная запи

1. Пример выполнения пояснительно записки к курсовой работе студентки группы №113523, Ермоленко Галины Дмитриевны.
2. Википедия (http://ru.wikipedia.org)
3. Дейт К. Руководство по реляционной СУБД DB2. – М.: Финансы и статистика, 1988. – 320 с.
4. Дейт К. Введение в системы баз данных //6-издание. – Киев: Диалектика, 1998. – 784 с.
5. Кузнецов С.Д. Введение в системы управления базами данных //СУБД. – 1995. – №1,2,3,4, 1996. – №1,2,3,4,5.
6. Codd E.F. Relation Model of Data for Large Shared Data Banks //Comm. ACM. – 1970. – V.13, №.6. – P.377-383. (Имеется перевод: Кодд Е.Ф. Реляционная модель данных для больших совместно используемых банков данных //СУБД. – 1995. – №1. – С.145-160.)