СУБД

введение

Основные функции СУБД

- управление данными во внешней памяти (на дисках);
- управление данными в оперативной памяти с использованием дискового кэша;
- журнализация изменений;
- резервное копирование и восстановление базы данных после сбоев;
- поддержка языков БД:
  - язык определения данных,
  - язык манипулирования данными).

Компоненты современных СУБД

- **ядро**, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию,
- **процессор языка базы данных**, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода,
- **подсистему поддержки времени исполнения**, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД
- а также **сервисные программы** (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

Классификации СУБД

- По модели данных
- По степени распределённости
- По способу доступа к БД

Классификация по модели данных

- Иерархические
- Сетевые
- **Реляционные**
- Объектно-ориентированные
- Объектно-реляционные

По степени распределённости

- Локальные СУБД
- Распределенные СУБД

По способу доступа к БД

- Файл-серверные
  - Microsoft Access, Paradox, dBase, FoxPro, Visual FoxPro
- Клиент-серверные
  - Oracle, Firebird, Interbase, IBM DB2, Informix, MS SQL Server, Sybase Adaptive Server Enterprise, PostgreSQL, MySQL, Caché, Derby
- Встраиваемые
  - OpenEdge, SQLite, BerkeleyDB, Firebird Embedded, Microsoft SQL Server Compact, Derby

Элементы теории множеств

Множества

- Наиболее простая структура данных, используемая в математике, имеет место в случае, когда между отдельными изолированными данными отсутствуют какие-либо взаимосвязи. Совокупность таких данных представляет собой ***множество*.**
- Понятие множества является неопределяемым понятием.
- Множество не обладает внутренней структурой.

Элементы теории множеств

Для того чтобы некоторую совокупность элементов можно было назвать множеством, необходимо, чтобы выполнялись следующие условия:

- Должно существовать правило, позволяющее определить, принадлежит ли указанный элемент данной совокупности.
- Должно существовать правило, позволяющее отличать элементы друг от друга. (Это, в частности, означает, что множество не может содержать двух *одинаковых* элементов).

Элементы теории множеств

- Если элемент x принадлежит множеству A, то это обозначается: $x \in A$
- Если каждый элемент множества A является также и элементом множества B, то говорят, что множество A является *подмножеством* множества B: $A \subset B$
- Подмножество B множества A называется *собственным подмножеством*, если $A \neq B$
- Используя понятие множества можно построить более сложные и содержательные объекты.

Операции над множествами

- *Объединением* двух множеств называется новое множество

$$A \cup B = \{x \mid x \in A \text{ или } x \in B\}$$

Операции над множествами

- *Пересечением* двух множеств называется новое множество

$$A \cap B = \{x \,|x \ \in A \text{ и } x \ \in B\}$$

Операции над множествами

- *Разностью* двух множеств называется новое множество

$$A \setminus B = \{x \mid x \in A \text{ и } x \notin B\}$$

Операции над множествами

Если класс объектов, на которых определяются различные множества обозначить Ω(***Универсум***), то *дополнением* множества А называют разность:

$$\bar{A} = \Omega \backslash \mathrm{A}$$

Декартово произведение множеств

- Пусть A и B- множества. Выражение вида (a,b), где $a \in A$ и $b \in B$, называется ***упорядоченной парой*.**
- Равенство вида $(a,b) = (c,d)$ означает, что $a=c$ и $b=d$. В общем случае, можно рассматривать *упорядоченную n-ку* $(a_1,a_2,\ldots,a_n)$ из элементов $a_1 \in A_1$, $a_2 \in A_2$, … $a_n \in A_n$.
- Упорядоченные n-ки иначе называют ***наборы*** или ***кортежи***.

Декартово произведение множеств

Декартовым (прямым) произведением множеств A_1, A_2, … A_n называется множество упорядоченных n-ок (наборов, кортежей) вида

$$A_1 \times A_2 \times \dots \times A_n = \{(a_1, a_2, \dots, a_n) | a_i \in A_i\}$$

Степенью декартового произведения называют число множеств, входящих в это произведение.

Отношение

- Подмножество R декартового произведения множеств $A_1 \times A_2 \times \cdots \times A_n$ называется ***отношением степени n*** (***n-арным отношением***).
- Мощность множества кортежей, входящих в отношение R, называют *мощностью отношения R*

Отношение

- Понятие отношения фактически лежит в основе всей реляционной теории баз данных.
- Отношения являются математическим аналогом *таблиц*.
- Термин "реляционное представление данных", впервые введенный Коддом, происходит от термина ***relation***, понимаемом именно в смысле этого определения.

Отношение

- Любое множество можно рассматривать как декартовое произведение степени 1, поэтому любое подмножество, как и любое множество, можно считать отношением степени 1.
- Это не очень интересный пример, свидетельствующий лишь о том, что термины "отношение степени 1" и "подмножество" являются синонимами.
- Нетривиальность понятия отношения проявляется, когда степень отношения больше 1.

Отношение

- все элементы отношения есть *однотипные* кортежи.
- Однотипность кортежей позволяет считать их аналогами строк в простой таблице, т.е. в такой таблице, в которой все строки состоят из одинакового числа ячеек и в соответствующих ячейках содержатся одинаковые типы данных.

Отношение

Например, отношение, состоящее из трех следующих кортежей

{(1, "Иванов", 1000), (2, "Петров", 2000), (3, "Сидоров", 3000)} можно считать таблицей, содержащей данные о сотрудниках и их зарплатах.

Такая таблица будет иметь три строки и три колонки, причем в каждой колонке содержатся данные одного типа.

Отношение

За исключением крайнего случая, когда отношение есть само декартово произведение A_1 x A_2 x … x A_n, отношение включает в себя *не все возможные кортежи* из декартового произведения.

Это значит, что для каждого отношения имеется *критерий*, позволяющий определить, какие кортежи входят в отношение, а какие - нет.

Этот критерий, по существу, определяет для нас *смысл (семантику)* отношения.

Отношение

- Каждому отношению можно поставить в соответствие некоторое логическое выражение $P(x_1, x_2, x_3, \dots, x_n)$, зависящее от n параметров (n-местный предикат) и определяющее, будет ли кортеж $(a_1, a_2, a_3, \dots, a_n)$принадлежать отношению R.
- Это логическое выражение называют ***предикатом отношения*** **R.**

Отношение

Если это не вызывает путаницы, удобно и отношение, и его предикат обозначать одной и той же буквой. Например, отношение R имеет предикат $R(x_1, x_2, \ldots, x_n)$

Примеры отношений

- В математике большую роль играют бинарные отношения, т.е. отношения, заданные на декартовом произведении двух множеств $A_1 x A_2$

Отношение эквивалентности

- $(x,x) \in R$ для всех $x \in A$ (рефлексивность)
- Если $(x,y) \in R$, то $(y,x) \in R$ (симметричность)
- Если $(x,y) \in R$ и $(y,z) \in R$, то $(x,z) \in R$ (транзитивность)

Отношения порядка

- $(x,x) \in R$ для всех $x \in A$ (рефлексивность)
- Если $(x,y) \in R$ и $(y,x) \in R$, то $x=y$ (антисимметричность)
- Если $(x,y) \in R$ и $(y,z) \in R$, то $(x,z) \in R$ (транзитивность)

Функциональное отношение

- Если $(x,y) \in R$ и $(x,z) \in R$, то $y=z$ (однозначность функции).
- Обычно, функциональное отношение обозначают в виде *функциональной зависимости* – $(x,y) \in R$ тогда и только тогда, когда $y=f(x)$.
- Функциональные отношения (подмножества декартового произведения!) называют иначе *графиком функции* или *графиком функциональной зависимости*.

Еще пример бинарного отношения

Пусть множество есть следующее множество молодых людей:

{Вовочка, Петя, Маша, Лена},

причем известны следующие факты:

Еще пример бинарного отношения

1. Вовочка любит Вовочку (эгоист).
2. Петя любит Машу (взаимно).
3. Маша любит Петю (взаимно).
4. Маша любит Машу (себя не забывает).
5. Лена любит Петю (несчастная любовь).

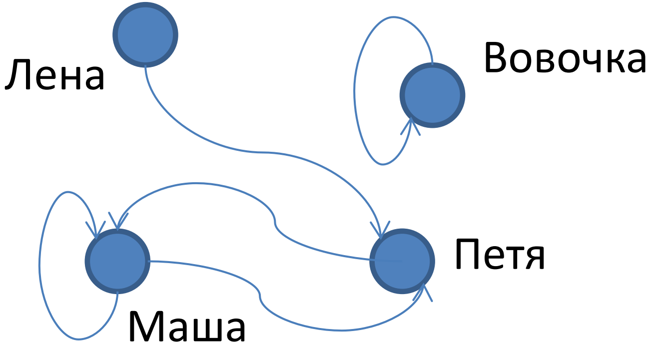
Еще пример бинарного отношения

- Информацию о взаимоотношения данных молодых людей можно описать бинарным отношением "любить", заданном на множестве . Это отношение можно описать несколькими способами.

Способы представления отношения

1. Перечисление фактов в виде произвольного текста (как это сделано выше).
2. В виде графа взаимоотношений.
3. При помощи матрицы взаимоотношений
4. При помощи таблицы фактов

Способы представления отношений



	Вовочка	Петя	Маша	Лена
Вовочка	Любит			
Петя			Любит	
Маша		Любит	Любит	
Лена		Любит		

Способы представления отношений

Кто любит	Кого любят
Вовочка	Вовочка
Петя	Маша
Маша	Петя
Маша	Маша
Лена	Петя

Предикат отношения

R(x,y) = {(x = "Вовочка" AND y = "Вовочка") OR (x = "Петя" AND y = "Маша") OR (x = "Маша" AND y = "Петя") OR (x = "Маша" AND y = "Маша") OR (x = "Лена" AND y = "Петя")}

Приведенное отношение не является ни транзитивным, ни симметричным или антисимметричным, ни рефлексивным, поэтому оно не является ни отношением эквивалентности, ни отношением порядка, ни каким-либо другим разумным отношением.

Изучением характеристик данного отношения и соответствующего ему предиката занималось (и продолжает заниматься) большое количество экспертов, таких как Толстой Л.Н., Шекспир В. и др.

Шутка ☺

Изучением характеристик данного отношения и соответствующего ему предиката занималось (и продолжает заниматься) большое количество экспертов, таких как Толстой Л.Н., Шекспир В. и др.

n-арные отношения (отношения степени n)

В некотором университете учатся студенты Иванов, Петров и Сидоров.

Лекции им читают преподаватели Дымо, Гайда и Тимофеев, причем известны следующие факты:

n-арные отношения (отношения степени n)

1. Тимофеев читает лекции по программированию и БД, соответственно, 40 и 80 часов в семестр.
2. Дымо читает лекции по UNIX, 50 часов в семестр.
3. Гайда читает лекции по программированию и UNIX, соответственно, 40 и 50 часов в семестр.
4. Студент Иванов посещает лекции по UNIX у Дымо и по БД у Тимофеева.
5. Студент Петров посещает лекции по программированию у Тимофеева и UNIX у Гайды.
6. Студент Сидоров посещает лекции по программированию у Гайды и по БД у Тимофеева.

A (Преподаватель)	B (Предмет)	Q (К-во часов)
Тимофеев	Программирование	40
Тимофеев	БД	80
Дымо	UNIX	50
Гайда	Программирование	40
Гайда	UNIX	50

C (Студент)	B (Предмет)	A (Преподаватель)
Иванов	UNIX	Дымо
Иванов	БД	Тимофеев
Петров	Программирование	Тимофеев
Петров	UNIX	Гайда
Сидоров	Программирование	Гайда
Сидоров	БД	Тимофеев

Табличная форма представления отношений

- Удобство использования табличной формы для задания отношения определяется в данном случае следующими факторами:
- Все используемые множества *конечны*.
- При добавлении или удалении студентов, предметов, преподавателей просто добавляются или удаляются соответствующие строки в таблице.

Замечание 1

- В таблицу "Посещать лекции" нельзя добавить две одинаковые строки, т.к. таблица изображает отношение R_2, а в отношении (как и в любом множестве) *не может быть двух одинаковых элементов*.
- Это пример ***синтаксического ограничения*** - такое ограничение задано в определении понятия отношение (одинаковых строк не может быть *ни в одной таблице*, задающей отношение).

Замечание 2

- В таблицу "Посещать лекции" нельзя добавить кортеж (Иванов, UNIX, Тимофеев). Действительно, из таблицы "Читает лекции по…", представляющей отношение R_1, следует, что Тимофеев *не читает* предмет “UNIX".
- Оказалось, что таблицы связаны друг с другом, и существенным образом! Это пример ***семантического ограничения*** - такое ограничение является следствием нашей трактовки данных, хранящихся в отношении (следствием понимания *смысла* данных).

Транзитивное замыкание отношений

Пусть отношение R задано на декартовом квадрате A x A некоторого множества A. ***Транзитивным замыканием*** отношения R называется новое отношение $\bar{R}$, состоящее из кортежей (x,y), для которых выполняется одно из двух условий:

Транзитивное замыкание отношений

- либо кортеж $(x, y) \in R$,
- либо найдется конечная последовательность элементов $(z_1, z_2, \ldots, z_n)$ такая, что все кортежи $(x, z_1), (z_1, z_2), \ldots, (z_{n-1}, z_n), (z_n, \mathrm{y})$ принадлежат отношению R.

Транзитивное замыкание

Конструкция	Где используется
Болт	Двигатель
Болт	Колесо
Гайка	Двигатель
Гайка	Колесо
Двигатель	Автомобиль
Колесо	Автомобиль
Ось	Колесо

Конструкция	Где используется
Болт	Двигатель
Болт	Колесо
Гайка	Двигатель
Гайка	Колесо
Двигатель	Автомобиль
Колесо	Автомобиль
Ось	Колесо
Болт	Автомобиль
Гайка	Автомобиль
Ось	Автомобиль

Выводы

- ***Множество***- это неопределяемое понятие, представляющее некоторую совокупность данных.
- Элементы множества можно отличать друг от друга, а также определять, принадлежит ли данный элемент данному множеству.
- Над множествами можно выполнять операции объединения, пересечения, разности и дополнения.

Выводы

- Новые множества можно строить при помощи понятия ***декартового произведения*** (конечно, есть и другие способы, но они нас в данный момент не интересуют).
- Декартово произведение нескольких множеств - это множество *кортежей*, построенное из элементов этих множеств.

Выводы

- ***Отношение***- это подмножество декартового произведения множеств. Отношения состоят из *однотипных* кортежей.
- Каждое отношение имеет ***предикат отношения*** и каждый n-местный предикат задает n-арное отношение.
- Отношение является математическим аналогом понятия "таблица".

Выводы

- Отношения обладают ***степенью и мощностью*.**
- **Степень отношения** - это количество элементов в каждом кортеже отношения (аналог количества столбцов в таблице).
- **Мощность отношения** - это мощность множества кортежей отношения (аналог количества строк в таблице).

Выводы

- В математике чаще всего используют ***бинарные отношения*** (отношения степени 2).
- В теории баз данных основными являются отношения степени **n**.
- В математике, как правило, отношения заданы на бесконечных множествах и имеют бесконечную мощность.
- В базах данных напротив, мощности отношений конечны (число хранимых строк в таблицах всегда конечно).

Базовые понятия реляционной модели данных

Основы реляционной модели

- Основы реляционной модели данных были впервые изложены в статье Е.Кодда в 1970 г:

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.)

- Эта работа послужила стимулом для большого количества статей и книг, в которых реляционная модель получила дальнейшее развитие.
- Наиболее распространенная трактовка реляционной модели данных принадлежит К.Дейту:

Дейт К. Введение в системы баз данных //6-издание. - Киев: Диалектика, 1998. - 784 с.

Основы реляционной модели

Согласно Дейту, реляционная модель состоит из трех частей:

- Структурной части.
- Целостной части.
- Манипуляционной части.

Основы реляционной модели

- ***Структурная часть*** описывает, какие объекты рассматриваются реляционной моделью.
- Постулируется, что единственной структурой данных, используемой в реляционной модели, являются нормализованные n-арные отношения.

Основы реляционной модели

- ***Целостная часть*** описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных базах данных.
- Это *целостность сущностей* и *целостность внешних ключей*.

Основы реляционной модели

- ***Манипуляционная часть*** описывает два эквивалентных способа манипулирования реляционными данными - *реляционную алгебру* и *реляционное исчисление*.

Типы данных

- Любые данные, используемые в программировании, имеют свои типы данных.
- *Важно!* Реляционная модель требует, чтобы типы используемых данных были *простыми*.
- Для уточнения этого утверждения рассмотрим, какие вообще типы данных обычно рассматриваются в программировании. Как правило, типы данных делятся на три группы:
  - Простые типы данных.
  - Структурированные типы данных.
  - Ссылочные типы данных.

Простые типы данных

Простые, или атомарные, типы данных не обладают внутренней структурой. Данные такого типа называют *скалярами*. К простым типам данных относятся следующие типы:

- **Логический**
- **Строковый**
- **Численный**

Структурированные типы данных

Структурированные типы данных предназначены для задания сложных структур данных. Структурированные типы данных конструируются из составляющих элементов, называемых компонентами, которые, в свою очередь, могут обладать структурой. В качестве структурированных типов данных можно привести следующие типы данных:

- **Массивы**
- **Записи (Структуры)**

С математической точки зрения массив представляет собой функцию с конечной областью определения.

Запись (или структура) представляет собой кортеж из некоторого декартового произведения множеств.

Ссылочные типы данных

Ссылочный тип данных (***указатели***) предназначен для обеспечения возможности указания на другие данные.

- Указатели характерны для языков процедурного типа, в которых есть понятие области памяти для хранения данных.
- Ссылочный тип данных предназначен для обработки сложных изменяющихся структур, например деревьев, графов, рекурсивных структур.

Типы данных, используемые в реляционной модели

- Для реляционной модели данных тип используемых данных не важен.
- Требование, чтобы тип данных был *простым*, нужно понимать так, что *в реляционных операциях не должна учитываться внутренняя структура данных*.
- Конечно, должны быть описаны действия, которые можно производить с данными как с единым целым, например, данные числового типа можно складывать, для строк возможна операция конкатенации и т.д.

Домены

Домен - это семантическое понятие. Домен можно рассматривать как подмножество значений некоторого типа данных имеющих определенный смысл. Домен характеризуется следующими свойствами:

- Домен имеет *уникальное имя* (в пределах базы данных).
- Домен определен на некотором *простом* типе данных или на другом домене.
- Домен может иметь некоторое *логическое условие*, позволяющее описать подмножество данных, допустимых для данного домена.
- Домен несет определенную *смысловую нагрузку*.

Домены

- Основное значение доменов состоит в том, что *домены ограничивают сравнения*.
- Некорректно, с логической точки зрения, сравнивать значения из различных доменов, даже если они имеют одинаковый тип. В этом проявляется смысловое ограничение доменов.
- Синтаксически правильный запрос "выдать список всех деталей, у которых вес детали больше имеющегося количества" не соответствует смыслу понятий "количество" и "вес".

Домены

- Понятие домена помогает правильно *моделировать* предметную область. При работе с реальной системой в принципе возможна ситуация когда требуется ответить на запрос, приведенный выше. Система даст ответ, но, вероятно, он будет бессмысленным.
- Не все домены обладают логическим условием, ограничивающим возможные значения домена.
  В таком случае множество возможных значений домена совпадает с множеством возможных значений типа данных.

Домены

- Не всегда очевидно, как задать логическое условие, ограничивающее возможные значения домена. Я буду благодарен тому, кто приведет мне условие на строковый тип данных, задающий домен "Фамилия сотрудника".
- Ясно, что строки, являющиеся фамилиями не должны начинаться с цифр, служебных символов, с мягкого знака и т.д. Но вот является ли допустимой фамилия "Гггггыыыыы"? Почему бы нет? Очевидно, нет! А может кто-то назло так себя назовет.

Отношения, атрибуты, кортежи отношения

- ***Атрибут отношения есть пара вида <Имя_атрибута : Имя_домена>.***
- Имена атрибутов должны быть уникальны в пределах отношения.
- Часто имена атрибутов отношения совпадают с именами соответствующих доменов.

Отношения, атрибуты, кортежи отношения

Отношение R, определенное на множестве доменов D_1, D_2,…,D_n(не обязательно различных), содержит две части: заголовок и тело.

- ***Заголовок*** *отношения* содержит фиксированное количество атрибутов отношения**:** $(<\boldsymbol{A_1}:\boldsymbol{D_1}>, <\boldsymbol{A_2}:\boldsymbol{D_2}>, \ldots, <\boldsymbol{A_n}:\boldsymbol{D_n}>)$
- ***Тело отношения*** содержит множество кортежей отношения. Каждый *кортеж отношения* представляет собой множество пар вида <Имя_атрибута : Значение_атрибута>:

$$(<\boldsymbol{A_1}:\boldsymbol{Val_1}>, <\boldsymbol{A_2}:\boldsymbol{Val_2}>, \ldots, <\boldsymbol{A_n}:\boldsymbol{Val_n}>)$$

таких что значение ***Val**$_i$* атрибута ***A**$_i$* принадлежит домену ***D**$_i$*

Отношения, атрибуты, кортежи отношения

- Отношение обычно записывается в виде:

$$\mathbf{R}(< A_1 : D_1 >, < A_2 : D_2 >, \dots, < A_n : D_n >),$$

или короче

$$\mathbf{R}(\mathbf{A}_1,\mathbf{A}_2,\dots,\mathbf{A}_n)$$

или просто

R.

Число атрибутов в отношении называют ***степенью*** (или - *арностью*) отношения.

Мощность множества кортежей отношения называют *мощностью* отношения.

Выводы

- Заголовок отношения описывает декартово произведение доменов, на котором задано отношение.
- Заголовок статичен, он не меняется во время работы с базой данных.
- Если в отношении изменены, добавлены или удалены атрибуты, то в результате получим уже *другое* отношение (пусть даже с прежним именем).

Выводы

- Тело отношения представляет собой набор кортежей, т.е. подмножество декартового произведения доменов.
- Таким образом, тело отношения собственно и является отношением в математическом смысле слова.
- Тело отношения может изменяться во время работы с базой данных - кортежи могут изменяться, добавляться и удаляться.

Определения

- ***Реляционной базой данных*** называется набор отношений.
- ***Схемой реляционной базы*** данных называется набор заголовков отношений, входящих в базу данных.

Определения

- Хотя любое отношение можно изобразить в виде таблицы, нужно четко понимать, что *отношения не являются таблицами*.
- Это близкие, но не совпадающие понятия.

Отношения и таблицы

Реляционный термин	Соответствующий «табличный» термин
База данных	Набор таблиц
Схема базы данных	Набор заголовков таблиц
Отношение	Таблица
Заголовок отношения	Заголовок таблицы
Тело отношения	Тело таблицы
Атрибут отношения	Наименование столбца таблицы
Кортеж отношения	Строка таблицы
Степень (-арность) отношения	Количество столбцов таблицы
Мощность отношения	Количество строк таблицы
Домены и типы данных	Типы данных в ячейках таблицы

Свойства отношений

- *В отношении нет одинаковых кортежей*. Действительно, тело отношения есть *множество* кортежей и, как всякое множество, не может содержать неразличимые элементы (см. понятие множества).
- Таблицы в отличие от отношений могут содержать одинаковые строки.

Свойства отношений

- *Кортежи не упорядочены (сверху вниз)*.
- Причина та же - тело отношения есть множество, а множество не упорядочено. Это вторая причина, по которой нельзя отождествить отношения и таблицы - строки в таблицах упорядочены.
- Одно и то же отношение может быть *изображено* разными таблицами, в которых *строки идут в различном порядке*.

Свойства отношений

- *Атрибуты не упорядочены (слева направо)*.
- Т.к. каждый атрибут имеет уникальное имя в пределах отношения, то порядок атрибутов не имеет значения. Это свойство несколько отличает отношение от математического определения отношения. Это также третья причина, по которой нельзя отождествить отношения и таблицы - столбцы в таблице упорядочены.
- Одно и то же отношение может быть *изображено* разными таблицами, в которых *столбцы идут в различном порядке*.

Свойства отношений

- *Все значения атрибутов атомарны*.
- Это следует из того, что лежащие в их основе атрибуты имеют атомарные значения. Это четвертое отличие отношений от таблиц - в ячейки таблиц можно поместить что угодно - массивы, структуры, и даже другие таблицы.

Свойства отношений

- Из свойств отношения следует, что *не каждая* таблица может задавать отношение.
- Для того, чтобы некоторая таблица задавала отношение, необходимо, чтобы таблица имела простую структуру (содержала бы только строки и столбцы, причем, в каждой строке было бы одинаковое количество полей), в таблице не должно быть одинаковых строк, любой столбец таблицы должен содержать данные только одного типа, все используемые типы данных должны быть простыми.

Свойства отношений

Каждое отношение можно считать ***классом эквивалентности таблиц***, для которых выполняются следующие условия:

- Таблицы имеют одинаковое количество столбцов.
- Таблицы содержат столбцы с одинаковыми наименованиями.
- Столбцы с одинаковыми наименованиями содержат данные из одних и тех же доменов.
- Таблицы имеют одинаковые строки с учетом того, что порядок столбцов может различаться.

Все такие таблицы есть различные *изображения* одного и того же отношения.

Нормальные формы отношений

1НФ

2НФ

3НФ

НФБК

4НФ

5НФ

1НФ

Кто знает определение?

1 НФ

- Труднее всего дать определение вещей, которые всем понятны.
- Если давать не строгое, описательное определение, то всегда остается возможность неправильной его трактовки.
- Если дать строгое формальное определение, то оно, как правило, или тривиально, или слишком громоздко.
- Дать определение 1НФ сложно ввиду его тривиальности.

1 НФ - объяснения

- Говорят, что отношение R находится в 1НФ, если оно удовлетворяет определению на слайде 69.
- Говорят, что отношение R находится в 1НФ, если его атрибуты содержат только скалярные (атомарные) значения.

Нарушение 1НФ

- Легко себе представить таблицу, у которой в некоторых ячейках содержатся массивы, в других ячейках - определенные пользователями сложные структуры, а в третьих ячейках - целые реляционные таблицы, которые в свою очередь могут содержать такие же сложные объекты.
- Именно такие возможности предоставляются некоторыми современными пост-реляционными и объектными СУБД.

Целостность реляционных данных

Во второй части реляционной модели данных определяются два ограничения, которые должны выполняться в *любой* реляционной базе данных. Это:

- Целостность сущностей.
- Целостность внешних ключей.

Null-значения

- Для представления информации в базе данных используются привычные для программистов типы данных - строковые, численные, логические и т.п.
- Однако в реальном мире часто встречается ситуация, когда данные неизвестны или не полны.
- Например, место жительства или дата рождения человека могут быть неизвестны (база данных разыскиваемых преступников).

Null-значения

- Если вместо неизвестного адреса уместно было бы вводить пустую строку, то что вводить вместо неизвестной даты?
- Ответ - пустую дату - не вполне удовлетворителен, т.к. простейший запрос "выдать список людей в порядке возрастания дат рождения" даст заведомо неправильный ответ.

Null-значения

- Практически все реализации современных реляционных СУБД позволяют использовать null-значения, несмотря на их недостаточную теоретическую обоснованность.
- Многие авторы полагают, что желательно избегать null-значений.

Трехзначная логика (3VL)

- Т.к. null-значение обозначает на самом деле тот факт, что значение неизвестно, то любые алгебраические операции (сложение, умножение, конкатенация строк и т.д.) должны давать также неизвестное значение, т.е. null.
- Действительно, если, например, вес детали неизвестен, то неизвестно также, сколько весят 10 таких деталей.

Потенциальные ключи

Пусть дано отношение . Подмножество атрибутов отношения будем называть ***потенциальным ключом,*** если обладает следующими свойствами:

- ***Свойством уникальности*** - в отношении не может быть двух различных кортежей, с одинаковым значением .
- ***Свойством неизбыточности*** - никакое подмножество в не обладает свойством уникальности.

Потенциальные ключи - пример

Табельный номер	Фамилия	Зарплата
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000

Потенциальные ключи - пример

Табельный номер	Фамилия	Зарплата
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000

A	B	C
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000

Целостность сущностей

Правило целостности сущностей.

Атрибуты, входящие в состав некоторого потенциального ключа не могут принимать null-значений.

Внешние ключи

- Различные объекты предметной области, информация о которых хранится в базе данных, всегда взаимосвязаны друг с другом.
- Например, накладная на поставку товара *содержит* список товаров с количествами и ценами, сотрудник предприятия *имеет* детей, *числится* в подразделении и т.д.

Внешние ключи

- Такие взаимосвязи отражаются в реляционных базах данных при помощи ***внешних ключей,*** связывающих несколько отношений.

Внешние ключи

Номер поставщика	Наименование поставщика	*Номер детали*	Наименование детали	Поставляемое количество
1	Иванов	*1*	Болт	100
1	Иванов	*2*	Гайка	200
1	Иванов	*3*	Винт	300
2	Петров	*1*	Болт	150
2	Петров	*2*	Гайка	250
3	Сидоров	*3*	Винт	1000

Проблемы возникают потому, что мы смешали в одном отношении различные объекты предметной области - и данные о поставщиках, и данные о деталях, и данные о поставках деталей. Говорят, что это отношение плохо нормализовано

Внешние ключи

Подмножество атрибутов FK отношения R будем называть *внешним ключом*, если:

- Существует отношение S (R и S не обязательно различны) с потенциальным ключом K.
- Каждое значение FK в отношении R всегда совпадает со значением K для некоторого кортежа из S, либо является null-значением.

Отношение называется *родительским отношением*, отношение называется *дочерним отношением*.

Внешние ключи

- Замечание. Внешний ключ, также как и потенциальный, может быть простым и составным.
- Замечание. Внешний ключ должен быть определен на тех же доменах, что и соответствующий первичный ключ родительского отношения.
- Замечание. Внешний ключ, как правило, *не обладает свойством уникальности*. Так и должно быть, т.к. в дочернем отношении может быть несколько кортежей, ссылающихся на один и тот же кортеж родительского отношения. Это, собственно, и дает тип отношения "один-ко-многим".
- Замечание. Если внешний ключ все-таки обладает свойством уникальности, то связь между отношениями имеет тип "***один-к-одному*****".** Чаще всего такие отношения объединяются в одно отношение, хотя это и не обязательно.

Внешние ключи

- Замечание. Хотя каждое значение внешнего ключа обязано совпадать со значениями потенциального ключа в некотором кортеже родительского отношения, то обратное, вообще говоря, неверно. Например, могут существовать поставщики, не поставляющие никаких деталей.
- Замечание. Для внешнего ключа не требуется, чтобы он был компонентом некоторого потенциального ключа (как получилось в примере с поставщиками и деталями).
- Замечание. Null-значения для атрибутов внешнего ключа допустимы только в том случае, когда атрибуты внешнего ключа не входят в состав никакого потенциального ключа

Целостность внешнего ключа

- Т.к. внешние ключи фактически служат ссылками на кортежи в другом (или в том же самом) отношении, то эти ссылки не должны указывать на несуществующие объекты.
- Это определяет следующее ***правило целостности внешних ключей*:**

Внешние ключи не должны быть несогласованными, т.е. для каждого значения внешнего ключа должно существовать соответствующее значение первичного ключа в родительском отношении.

Операции, могущие нарушить ссылочную целостность

- Ссылочная целостность может нарушиться в результате операций, изменяющих состояние базы данных.
- Таких операций три: вставка, обновление и удаление кортежей в отношениях.
- Т.к. в определении ссылочной целостности участвуют два отношения - родительское и дочернее, а в каждом из них возможны три операции: вставка, обновление, удаление, то нужно рассмотреть шесть различных вариантов.

Для родительского отношения

Вставка кортежа в родительском отношении.

- При вставке кортежа в родительское отношение возникает новое значение потенциального ключа.
- Т.к. допустимо существование кортежей в родительском отношении, на которые нет ссылок из дочернего отношения, то вставка кортежей в родительское отношение ***не нарушает ссылочной целостности***.

Для родительского отношения

Обновление кортежа в родительском отношении.

- При обновлении кортежа в родительском отношении может измениться значение потенциального ключа.
- Если есть кортежи в дочернем отношении, ссылающиеся на обновляемый кортеж, то значения их внешних ключей станут некорректными.
- Обновление кортежа в родительском отношении ***может привести к нарушению ссылочной целостности***, если это обновление затрагивает значение потенциального ключа

Для родительского отношения

Удаление кортежа в родительском отношении.

- При удалении кортежа в родительском отношении удаляется значение потенциального ключа.
- Если есть кортежи в дочернем отношении, ссылающиеся на удаляемый кортеж, то значения их внешних ключей станут некорректными.
- Удаление кортежей в родительском отношении ***может привести к нарушению ссылочной целостности***.

Для дочернего отношения

Вставка кортежа в дочернее отношение.

- Нельзя вставить кортеж в дочернее отношение, если вставляемое значение внешнего ключа некорректно.
- Вставка кортежа в дочернее отношение ***может привести к нарушению ссылочной целостности***.

Для дочернего отношения

Обновление кортежа в дочернем отношении.

- При обновлении кортежа в дочернем отношении можно попытаться некорректно изменить значение внешнего ключа.
- Обновление кортежа в дочернем отношении ***может привести к нарушению ссылочной целостности***.

Для дочернего отношения

***Удаление кортежа в дочернем отношении*.**

- При удалении кортежа в дочернем отношении ***ссылочная целостность не нарушается***

Возможно нарушение ссылочной целостности

- Обновление кортежа в родительском отношении.
- Удаление кортежа в родительском отношении.
- Вставка кортежа в дочернее отношение.
- Обновление кортежа в дочернем отношении.

Стратегии поддержания ссылочной целостности

- ***RESTRICT (ОГРАНИЧИТЬ)***- не разрешать выполнение операции, приводящей к нарушению ссылочной целостности. Это самая простая стратегия, требующая только проверки, имеются ли кортежи в дочернем отношении, связанные с некоторым кортежем в родительском отношении.

Стратегии поддержания ссылочной целостности

- ***CASCADE (КАСКАДИРОВАТЬ)***- разрешить выполнение требуемой операции, но внести при этом необходимые поправки в других отношениях так, чтобы не допустить нарушения ссылочной целостности и сохранить все имеющиеся связи. Изменение начинается в родительском отношении и каскадно выполняется в дочернем отношении.

2НФ (Вторая Нормальная Форма)

- Отношение находится во *второй нормальной форме* (*2НФ*) тогда и только тогда, когда отношение находится в 1НФ и *нет неключевых атрибутов, зависящих от части сложного ключа*. (*Неключевой атрибут* - это атрибут, не входящий в состав никакого потенциального ключа).

2НФ (Вторая Нормальная Форма)

- Отношение находится во *второй нормальной форме* (*2НФ*) тогда и только тогда, когда отношение находится в 1НФ и *нет неключевых атрибутов, зависящих от части сложного ключа*. (*Неключевой атрибут* - это атрибут, не входящий в состав никакого потенциального ключа).
- Замечание. Если потенциальный ключ отношения является простым, то отношение автоматически находится в 2НФ.

3НФ (Третья Нормальная Форма)

- Атрибуты называются ***взаимно независимыми,*** если ни один из них не является функционально зависимым от другого.
- Отношение находится в ***третьей нормальной форме*** **(*3НФ*)** тогда и только тогда, когда отношение находится в 2НФ и *все неключевые атрибуты взаимно независимы*.

Сравнение нормализованных и ненормализованных моделей

Критерий	Отношения слабо нормализованы (1НФ, 2НФ)	Отношения сильно нормализованы (3НФ)
Адекватность базы данных предметной области	ХУЖЕ (-)	ЛУЧШЕ (+)
Легкость разработки и сопровождения базы данных	ХУЖЕ (-)	ЛУЧШЕ (+)
Скорость выполнения вставки, обновления, удаления	ХУЖЕ (-)	ЛУЧШЕ (+)
Скорость выполнения выборки данных	ЛУЧШЕ (+)	ХУЖЕ (-)

Операторы SQL

- Операторы DDL (Data Definition Language) - операторы определения объектов базы данных
- Операторы DML (Data Manipulation Language) - операторы манипулирования данными
- Операторы защиты и управления данными

Операторы DDL (Data Definition Language)

- CREATE SCHEMA - создать схему базы данных
- DROP SCHEMA - удалить схему базы данных
- CREATE TABLE - создать таблицу
- ALTER TABLE - изменить таблицу
- DROP TABLE - удалить таблицу
- CREATE DOMAIN - создать домен
- ALTER DOMAIN - изменить домен
- DROP DOMAIN - удалить домен
- CREATE COLLATION - создать последовательность
- DROP COLLATION - удалить последовательность
- CREATE VIEW - создать представление
- DROP VIEW - удалить представление

Операторы DML (Data Manipulation Language)

- 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

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

Примеры использования операторов манипулирования данными

INSERT - вставка строк в таблицу

- Вставка одной строки в таблицу:

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

Примеры использования операторов манипулирования данными

INSERT - вставка строк в таблицу

- 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;
```

Примеры использования операторов манипулирования данными

UPDATE - обновление строк в таблице

Обновление нескольких строк в таблице:

```
UPDATE P
 SET PNAME = "Пушкин"
 WHERE P.PNUM = 1;
```

Примеры использования операторов манипулирования данными

DELETE - удаление строк в таблице

Удаление нескольких строк в таблице:

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

Удаление всех строк в таблице:

```
DELETE FROM P;
```

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.

Примеры использования оператора SELECT

- 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.

Отбор данных из одной таблицы

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.

Отбор данных из одной таблицы

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.

Отбор данных из одной таблицы

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

Отбор данных из одной таблицы

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.

Отбор данных из одной таблицы

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;
```

Отбор данных из одной таблицы

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

Отбор данных из одной таблицы

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

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

Отбор данных из одной таблицы

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

Отбор данных из одной таблицы

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;
```

Отбор данных из одной таблицы

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

Отбор данных из нескольких таблиц

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;
```

Отбор данных из нескольких таблиц

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

Отбор данных из нескольких таблиц

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.

Отбор данных из нескольких таблиц

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.

Отбор данных из нескольких таблиц

Естественное соединение трех таблиц:

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

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

Естественное соединение трех таблиц

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

Отбор данных из нескольких таблиц

Прямое произведение таблиц:

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

Прямое произведение таблиц

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

Прямое произведение таблиц

Соединение таблиц по произвольному условию.

Рассмотрим таблицы поставщиков и деталей, которыми присвоен некоторый статус

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

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

Прямое произведение таблиц

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;
```

Прямое произведение таблиц

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

Использование имен корреляции (алиасов, псевдонимов)

- 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.

Использование имен корреляции (алиасов, псевдонимов)

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;
```

Использование имен корреляции

В результате получим следующую таблицу:

PNAME	PSTATUS	PNAME2	PSTATUS2
Иванов	4	Петров	1
Иванов	4	Сидоров	2
Сидоров	2	Петров	1

Использование имен корреляции

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 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

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

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

Использование имен корреляции

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;
```

Использование имен корреляции

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

Использование имен корреляции

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;
```

Использование агрегатных функций в запросах

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

Использование агрегатных функций в запросах

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 таблицу с одной строкой

SM	MX	MN	AV
2000	1000	100	333.33333333

Использование агрегатных функций с группировками

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

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

Использование агрегатных функций с группировками

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

Использование агрегатных функций с группировками

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.

Использование агрегатных функций с группировками

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

<u>Замечание</u>. 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;
```

DNUM	SM
1	1250
2	450

Использование подзапросов

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*.

Использование подзапросов

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

- Выполнить *один раз* вложенный подзапрос и получить максимальное значение статуса.
- Просканировать таблицу поставщиков P, каждый раз сравнивая значение статуса поставщика с результатом подзапроса, и отобрать только те строки, в которых статус меньше максимального.

Использование подзапросов

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.

Использование подзапросов

Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

- Выполнить *один раз* вложенный подзапрос и получить список номеров поставщиков, поставляющих деталь номер 2.
- Просканировать таблицу поставщиков P, каждый раз проверяя, содержится ли номер поставщика в результате подзапроса.

Использование подзапросов

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);
```

Использование подзапросов

Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

- Просканировать таблицу поставщиков P, *каждый раз выполняя подзапрос* с новым значением номера поставщика, взятым из таблицы P.
- В результат запроса включить только те строки из таблицы поставщиков, для которых вложенный подзапрос вернул непустое множество строк.

Использование подзапросов

- 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.

Использование подзапросов

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.

Использование подзапросов

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*.

Использование объединения, пересечения и разности

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;
```

Использование объединения, пересечения и разности

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 результатирующем запросе будут автоматически взяты из результата первого запроса в объединении

Использование объединения, пересечения и разности

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;
```

Использование объединения, пересечения и разности

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;
```

Практика