КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра технологий программирования

Н.Р. БУХАРАЕВ

ВВЕДЕНИЕ В РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ И ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ SQL

Учебное пособие

Казань – 2018

УДК: 004.65 Система управления базами данных (СУБД)

ББК: 32.972.134 Системы управления базами данных

Принято на заседании кафедры технологий программирования
Протокол № 3 от 15 ноября 2018 года

Рецензенты:
кандидат физико-математических наук,
доцент кафедры технологий программирования КФУ **А.И.Еникеев;**
кандидат технических наук,
доцент кафедры технологий программирования КФУ **А.М.Гусенков**

Бухараев Н.Р.
Введение в реляционные базы данных и программирование на языке SQL /Н.Р.Бухараев. – Казань: Казан. ун-т, 2018. – 134 с.

Данное учебное пособие представляет собой краткое введение в понятийный аппарат реляционных баз данных и язык структурированных запросов SQL; оно адресуется преподавателям и студентам при изучении соответствующих разделов в курсе "Информатика" или иных вводных курсов программирования.

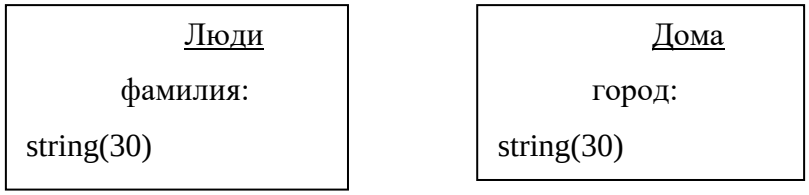
Цель пособия скромна - не заменить достаточно богатую "толстую" литературу по SQL и технологиям СУБД в целом, но скорее - подготовить к ее чтению начинающего осваивать эту обширную область, выделив (даже в ущерб если не точности, то полноте изложения) одновременно наиболее существенное и минимально достаточное.

Стиль изложения отличает полуформальный характер изложения, стимулирующий будущих разработчиков программного обеспечения к серьезному изучению фундаментальных математических дисциплин; в первую очередь - теории множеств и математической логики.

СОДЕРЖАНИЕ

§ 1. Базы данных как аппарат моделирования.

Как и в случае хорошо знакомого нам понятия типа данных, цель систем управления базами данных (СУБД) - достаточно полное и точное, в контексте решаемых задач, формальное описание (**моделирование**) объектов и процессов некоторой реальной ситуации, сферы деятельности (**предметной области**). По-прежнему, каждое из описывающих объекты понятий характеризуется набором характеристик или **атрибутов** (вообще говоря, различных типов), а процессы их преобразования - процедурами модификации значений атрибутов.



Однако - в отличие от понятия типа - здесь изначально внимание сфокусировано на структуре данных, а не структуре преобразований, не на описании правил преобразования объектов, но на форме представления данных - исходной, невычислимой, хранимой информации об объектах - в виде констатации некоторых имеющихся между ними взаимосвязей, или отношений.

Если понятие типа данных ориентируется более на динамическую, функционально-операционную сторону описания систем объектов, то БД - на статическую, описательную, дескриптивно-декларативную. Если, с точки зрения процедурного программирования, "приоритетно, все есть функция", то, с точки зрения представления информации в реляционных БД, "приоритетно, все есть отношение".

Напомним, что *декартовым произведением* $T_1 \times T_2 \times \ldots \times T_n$ множеств $T_1, T_2, \ldots, T_n$ называется множество всех кортежей длины n (или, попросту, n-ок)

$$T_1 \times T_2 \times \ldots \times T_n =_{def} \{<t_1,t_2,\ldots, t_n>: \forall\, i \in [1..n]\ (t_i \in T_i)\},$$

а n-арным *отношением (типа* $T_1 \times T_2 \times \ldots \times T_n$*)* - некоторое множество кортежей R, $R \subset T_1 \times T_2 \times \ldots \times T_n$, т.е. - любое подмножество некоторого декартова произведения. Часто в математике отношение связывают с некоторым свойством кортежей (пар, троек и т.д.), неявно отождествляя с предикатами (булевскими функциями):

$$R(t_1,t_2,\ldots, t_n)=true \approx <t_1,t_2,\ldots, t_n> \in R \approx$$

"элементы $t_1,t_2,\ldots, t_n$ обладают (связаны) свойством R".

Наоборот, с каждой n-местной функцией f мы привыкли отождествлять *график* функции, т.е. отношение $\{<x_1,x_2,\ldots, x_n, y>: <x_1,x_2,\ldots, x_n> \in Dom(f)\ ,\ y=f(x_1,x_2,\ldots, x_n) \in Val(f)\}$. Таким образом, с формальной стороны, языки теории функций и теории отношений равномощны как языки описаний. Содержательно же, - это существенно разные точки зрения; в том числе, с точки зрения практики математической - так, центральный и труднейший для математики вопрос - решение уравнений можно трактовать как переход от отношения к функции (или системе функций). Например, от отношения $ax^2+bx+c=0$ к системе $x_{1,2}=(-b+/-\sqrt{(b^2-ac)})/2a)$. Фундаментальная содержательная разница между отношениями и функциями – это разница между многозначностью и однозначностью. Не удивительно поэтому, что функции воспринимается нами обычно как более простое понятие, по сравнению с отношениями.

Отметим, что здесь - как обычно в программировании - мы в реальности имеем дело с *именованными* и *типизированными* аналогами классических понятий, т.е. фиксированной и явно заданной системой обозначений значений и функций (операторов).

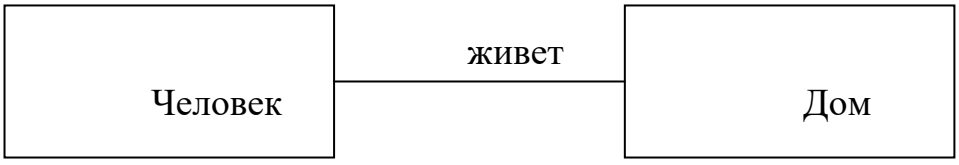
В данном случае – с *именованными* кортежами, элементами которых являются пары вида <имя, значение> (сравни с понятиями состояния переменных и, особенно - типа "запись" (record) в процедурном

программировании). Соответственно, именованным декартовым произведением множество всех именованных кортежей, при фиксированном множестве имен, принимающих всевозможные значения заданных типов.

Таблица есть *и*менованное отношение, или есть произвольное подмножество именованного декартова произведения.

Хотя далее мы упрощаем ситуацию, предполагая именование очевидным из контекста, и используем для именованного декартового произведения то же обозначение, что и для классического, просим читателя быть внимательным. В быту, чаще всего, мы не различаем имена и значения, что нередко – и в быту и, особенно, в программировании – приводит к большим недоразумениям. (Так, например – верно ли, что Маша состоит из 4 букв?)

Классические языки описания предикатов (отношений) - математическая логика (соответственно, теория множеств). Так, связь между людьми и домами можно выразить в виде предиката Живет: tЛюди × tДома → Boolean, Живет(x,y)=true ≈ <x,y> ∈ Живет ≈ " x живет в доме y". В программировании популярен более бедный, но и более наглядный язык диаграмм "сущность-связь", или ER-диаграмм (от англ. **E**ntity - цельность, сущность и **R**elations - отношение, связь).



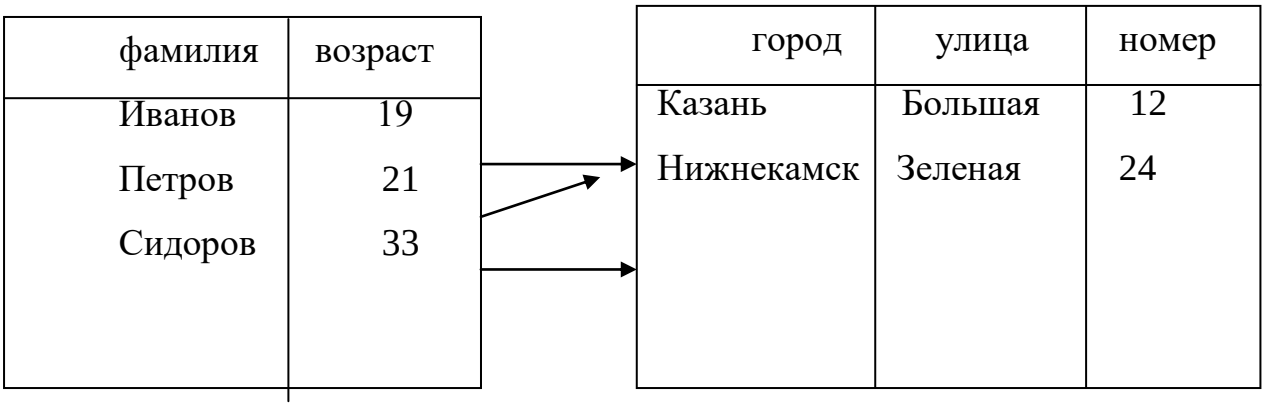
Пример наводит на мысль, что в естественном языке сущности выражаются существительными, а их связи – глаголами. Чаще всего, это действительно так; не стоит лишь забывать, что, в отличие от сущностей, связи *не обладают* собственными атрибутами. Единственное, что мы можем утверждать о связях – это факт их наличия либо отсутствия (на формальном уровне – факт принадлежности кортежа отношению как множеству). Потому, в случае богатых содержанием глаголов приходится предварительно преобразовывать их в

существительные, например, «агрегат *собирают из* деталей» (здесь подразумевается некоторый достаточно сложный процесс конструирования, требующий дополнительного описания) в «*сборка* агрегата осуществляется из деталей»

В любом случае, выделение, на основе экспертной информации, **сущностей** моделируемых объектов**,** их описание в виде совокупности атрибутов некоторых типов, а также их **связей,** (взаимо)отношений между объектами предметной области составляет первоначальную логическую основу, структуру БД. Исключительно сложную и важную проблему определения адекватной задачам предметной области и оптимальной в смысле компактности хранения и скорости доступа (выборки) к данным структуры БД называют *проектированием БД.*

Стандартная физическая реализация таблиц - файлы, т.е. "медленные", по сравнению с оперативной памятью, наборы данных на постоянных внешних носителях большой емкости как правило, с достаточно простой логической структурой – часто, последовательные. Поэтому такое описание должно быть, по возможности, минимальным и эффективным, т.е. направленным на экономию памяти при хранении файлов и высокую скорость их обработки. Связь БД с механическими по природе носителями, более простыми по структуре и функционально менее "гибкими", по сравнению с электронными, во многом определила специфику их эволюции.

Состояние БД, т.е. конкретные значения хранимых в ней на некоторый момент времени данных, описывается с помощью явного табличного задания конечных предикатов, описывающих конкретные состояния объектов и взаимосвязей.



Графическое отражение факта "Иванов, 19 лет и Петров, 21 год, живут по адресу Казань, ул. Большая, 12, а Сидоров, 33 лет, живет по адресу Нижнекамск, улица Зеленая, 24" или, в более традиционной математической записи: Живет □

{ <<Иванов, 19>, <Казань, Большая,12> >,

<<Петров, 21>, <Казань, Большая, 12> >,

< <Сидоров, 33>, <Нижнекамск, Зеленая, 24>>}

Здесь для краткости мы опустили именование вида

{<**Люди** → <**Фамилия** →Иванов, **Возраст** →19>, **Дома** → <**Город** → Казань, **Улица** → Большая, **Номер** →12>,….)

В терминологии СУБД, именованное n-арное отношение T принято называть **таблицей (table)**, составляющие его кортежи (n-ки) - строками или **записями (record)**, а их имена - столбцами или **полями (field)** записи.

Отметьте - множество всех состояний каждой таблицы состоит из всех отношений *одного* типа, т.е. подмножеств одного фиксированного в ходе проектирования декартового произведения. Таблицы - динамический тип «по строкам», т.е. могут содержать переменное число записей, но не «по столбцам», поскольку число и типы полей фиксированы в процессе проектирования БД.

Основной целью использования БД является ответы на *запросы* пользователей, т.е. получение интересующей их информации – не только непосредственно физически хранящейся в БД, так и логически из нее выводимой.

Программные системы, снабженные алгоритмами ("машиной") логического вывода такой информации, называют *экспертными системами* и *базами знаний*, а языки, ориентированные на их построение – языками *логического программирования*. Наиболее известным среди последних является язык ProLog. Мы оставим знакомство с такими системами до времени более глубокого освоения основ математической логики.

Примеры запросов - "По какому адресу живет Иванов 19 лет?", "Сколько человек живут на данной улице?", "Кто самый старший в данном доме?" и т.п.

С точки зрения математики, ответ на запрос есть вывод решения некоторого класса предикатных уравнений - например, Живет(<Иванов, 19>,x)=true - посредством некоторого заданного класса эквивалентных преобразований. С преобразованиями классического в СУБД классов – алгеброй отношений Кодда – мы познакомимся позже.

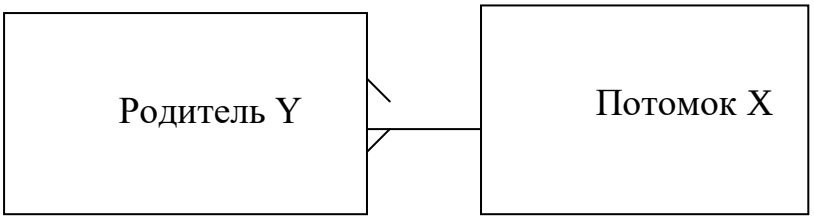
§ 2. Классификация бинарных отношений.

Пусть R - бинарное отношение, $R \subset Y \times X$, где Y, X - таблицы (т.е. снова отношения)

1) R - отношение *"один-ко-многим",* если каждому x соответствует в нем не более одного y, $\forall x \in X, \forall y,y' \in Y$ $(R(x,y) \& R(x,y') \rightarrow y=y')$

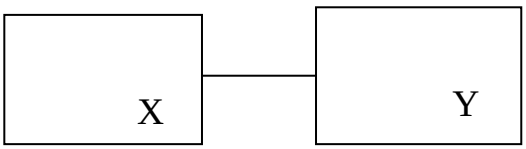
Пример. Отношение «человек x живет в доме y». Предполагается, что в одном доме могут жить много людей, но каждый живет (прописан) в одном.

В этом случае таблицу Y называют материнской или *родительской (parent)*, а таблицу X - дочерней или *потомком (child)*.

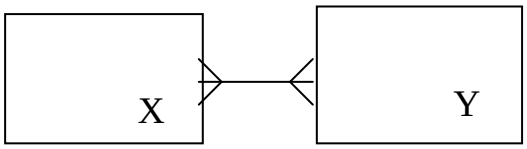


2) R - отношение *"один-ко-одному",* если каждому x соответствует не более одного y, и наоборот, $\forall x \in X, \forall y,y' \in Y$ (R(x,y) & R(x,y') → y=y')

Пример. Супружество, отношение «x женат на y» (европейский вариант). Каждый мужчина имеет не более одной жены и наоборот.



3) Оставшиеся отношения называют отношениями *"многие-ко-многим"*



Пример. Отношения знакомства - люди имеют много знакомых, которые, в свою очередь, тоже имеют много знакомых.

Заметим, что в последнем случае мы также имеем пример отношения - подмножества декартов квадрата, т.е. декартова произведения множества с

самим собой. Такие отношения мы будем называть *рефлексиями* (не путать - рефлексивным отношением называют отношение, содержащее тождественное отношение).

Нетрудно заметить, что на деле эти определения соответствуют определению (графика) функции, 1-1 функции (биекции) и собственного (т.е. не функционального) отношения.

Очень полезный, но реже употребляемый на практике в силу вычислительной сложности количественный подход к классификации отношений предусматривает оценку чисел N,M - "N записей одной таблицы может находиться в данном отношении с не более, чем M записями другой таблицы". Пропорцию N:M называют *кардинальностью* отношения.

В силу упомянутой выше большей простоты восприятия понятия функции, отношения "один-ко-многим" наиболее популярны в СУБД. Отношения "многие-ко-многим" трудны для восприятия и обработки - как правило, их стараются разложить, представить в виде соединения нескольких отношений "один-ко-многим". Отношения "один-к-одному" также достаточно редки - взаимно-однозначное соответствие "сущностей" обычно воспринимается нами как одна "сущность". Наличие связи "один-к-одному" означает (как правило), что удобнее иметь дело не с двумя, а с одним отношением (их композицией по этому соответствию).

Не случайно, в классической теоретико-множественной математики, как мы знаем, взаимно-однозначное соответствие составляет основу всякого отождествления. Заметим правда, что в приведенных выше определениях не требовалась всюду определенность (тотальность) определяемых функций. Так, весьма популярное в математике отношение "быть подмножеством" - пример отношения "один-ко-одному". В СУБД такая связь чаще применяется в случаях, когда лишь небольшое число записей одной таблице имеет соответствие в другой (т.е. является аналогом отношения “быть подмножеством”).

§ 3. Реляционные БД.

Итак, изначально, *база данных* - модель некоторой реальности (предметной области), отражающую информацию о ней в виде совокупности связанных отношениями таблиц и ориентированная на запросы пользователей о содержащейся в ней информации - как явно, так и неявно, логически.

Еще раз отметим, что это классическое определение БД определяет структуру данных, но не преобразований, цель ее использования, но не средства ее достижения. Позже нам придется вернуться к определению БД, существенно пополнившимся в ходе практики использования СУБД. “Данные – это (не только) значения”. Как мы увидим, осознание этого фундаментального факта привело к существенному сближению и взаимообогащению понятий база (модель) данных и типа данных - как логических понятий, хотя существенная прагматическая разница между обработкой файлов и данных оперативной памяти продолжает существовать. На удивление, кажется, что описываемая ниже первоначальная эволюция СУБД от структур данных с фиксированным и неявно заданным множеством преобразований (алгоритмов доступа) к “свободным структурам данных” свидетельствует об обратном. На деле, мы увидим, что схожий процесс происходил в эволюции понятия типа данных в первоначальной интерпретации процедурном программировании к понятию класса в объектном программировании.

Таблицы БД естественно реализуются последовательными файлами записей.

Ощутимая разница между теоретическим определением таблицы как отношения (множества) и ее фактической реализацией как последовательности (перечислением множества) нередко вносит неопределенность в семантику операций над таблицами.

Как реализовать отношения? Кажется совершенно естественным реализовать выделенные отношения программно, снабдив БД механизмом навигации, т.е. внутренними алгоритмами прохода по связям - например, от родителя к потомкам. Именно такой *навигационный* подход реализовывался в ранних *иерархических* и *сетевых* БД.

В названиях отражается применяемая в таких БД структура отношений в виде дерева (каждый потомок может иметь только одного родителя) и произвольного отношения (сети) - как правило, без связей "многие-ко-многим", но с возможностью каждому потомку иметь несколько родителей (множественное наследование).

Преимуществом навигационного подхода является высокая скорость доступа при выполнении запросов, недостатком - достаточно узкий круг разрешенных запросов; принято говорить, что навигационные БД не поддерживают незапланированные запросы.

Очевидно, что кроме явно выделенных, БД всегда содержит информацию о множестве иных, вторичных и неявных, но часто очень естественных с точки зрения пользовательских запросов отношений - таковы, например отношения-рефлексии "быть однофамильцем" или "быть старше (младше, ровесником)", никак не связанные с основным отношением "живет в доме", но логически выводимые из полной структуры БД.

Сегодня интерес к иерархическим БД вновь возрождается в виде т.н. объектно-ориентированных БД; возникающие вновь при этом проблемы с множественным наследованием мы обсудим позже в нашем курсе по ООП. Иерархические классификации – естественный упрощающий сложные реалии жизни аппарат нашего мышления. Проблема лишь в том, что реальность богаче любой фиксированной классификации. “Родитель”, в одной классификации – скажем, бюрократической иерархии, может оказаться “ребенком” в другой – например, иерархии генеалогической.

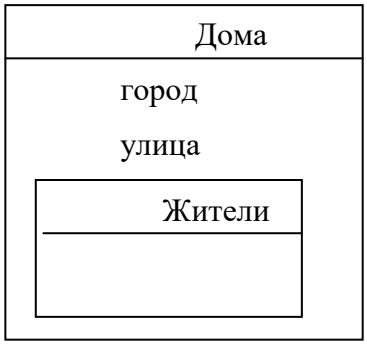
Подлинной революцией в развитии БД стали т.н. *реляционные* БД, в которых отношения не реализуются программно, но представляются в структуре БД с помощью специальных *ключевых выражений.*

Теоретический базис таких БД составила упомянутая выше реляционная алгебра Кодда.

Для полного понимания механизма ключей нам понадобится несколько определений.

Основным и, по существу, *единственным* структурным типом реляционных БД являются таблицы. Поля таблиц могут содержать лишь значения *базовых, или* **скалярных** типов, а также общее для всех таких типов специальное неопределенное значение NULL. В отличие от привычного описания сложных типов в процедурных языках, определение таблицы не рекурсивно и не итеративно, содержа лишь «один уровень» – таблицы *не могут* содержать в качестве полей другие таблицы.

А это значит, что нам придется отказаться от обычного при использовании структур оперативной памяти употребления неявного отношения включения при описании объектов, как в следующем примере



Единственный способ описания нетривиальных, сложно структурированных объектов и их взаимосвязей используется - явное задание **отношений.**

Идея реализации отношений между таблицами в реляционных БД исключительно проста и изящна - заменить рассмотрение отношений между сложными объектами (таблицами) простыми отношениями между простыми значениями базовых типов. А что может быть проще отношения равенства?

Рассмотрим работу механизма ключей в случае задания отношения "один-ко-многим". Пусть PrKey(r) - некоторое выражение над полями таблицы Parent. Можно считать, что PrKey *различает* две записи r1,r2∈Parent, если PrKey(r1)≠PrKey(r2). PrKey называется **первичным ключом (primary key)** таблицы, если оно позволяет идентифицировать каждую запись таблицы, отличить ее других:

$$\forall\ r1,r2 \in \text{Parent}\ (r1=r2 \Leftrightarrow \text{PrKey}(r1)=\text{PrKey}(r2))^1$$

В самом простом – и потому часто встречающемся на практике - случае ключом таблицы служит значение одного из ее полей, называемого в этом случае *ключевым*; в противном случае ключ называется **составным.**

[1] Для компактности мы используем, лишь в качестве стенографических, обозначения математической логики: ∀ - "для всех, для каждого", ∃ - "существует, найдется", ⇔ - "эквивалентно", ⇒ - "если…,то…", & -"и", ∨ -"или".

Согласно определению таблицы как отношения (т.е. множества), все поля записи должны образовывать ее первичный ключ - проще говоря, таблица не может содержать двух записей с полностью совпадающими значениями всех полей.

Здесь мы не уточняем определений зависящих от конкретной реляционной СУБД скалярных типов, операций над ними, равно как и точное понятие выражения. Так, в языках семейства **xBase** выражение-ключ – скалярного типа, а в рассматриваемом далее языке **SQL** таковым может быть произвольный *список имен столбцов*. В дальнейшем мы придерживаемся последнего, стандартного в теории БД определения.

Так, в предыдущем примере номер паспорта (но, в силу существования однофамильцев - *не* фамилия!) – первичный ключ таблицы ЛЮДИ, а первые три поля таблицы ДОМА (адрес дома) образуют составной первичный ключ этой таблицы.

Далее, пусть FKey(r) - выражение над полями другой таблицы Child. Запись r1, r1∈Child *ссылается (references)* на запись r2, r2∈Parent, если FKey(r1)=Parent(r2). FKey называется **внешним ключом (foreign key)** таблицы Child, если каждая запись в Child имеет соответствие в Parent в следующем смысле:

$$\forall\, r1 \in \text{Child}\ (\text{FKey}(r1) \neq \text{NULL} \rightarrow \exists r2 \in \text{Parent}\ (\text{FKey}(r1)=\text{PrKey}(r2))$$

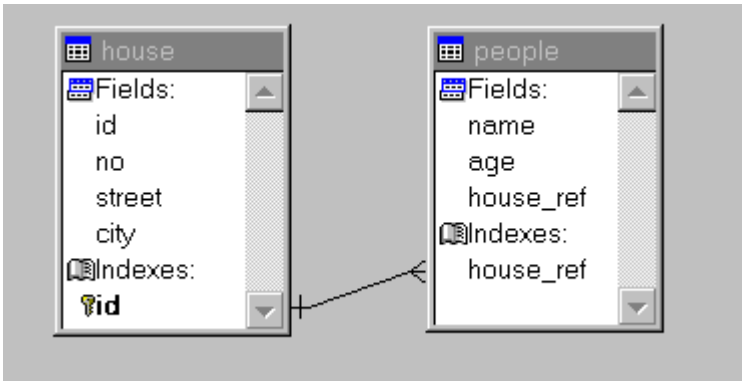
Очевидно, отношение "один-ко-многим" R между таблицами Parent, Child можно определить заданием соответствующих ключевых выражений FKey, PrKey таких, что

$$\forall\, r1 \in \text{Parent}\ \forall\, r2 \in \text{Child}\ (<r1,r2> \in R \Leftrightarrow \text{FKey}(r1)=\text{PrKey}(r2))$$

Аналогично, отношения "один-к-одному" могут быть заданы равенством первичных ключей, а отношения "многие-ко-многим" - равенством внешних ключей.

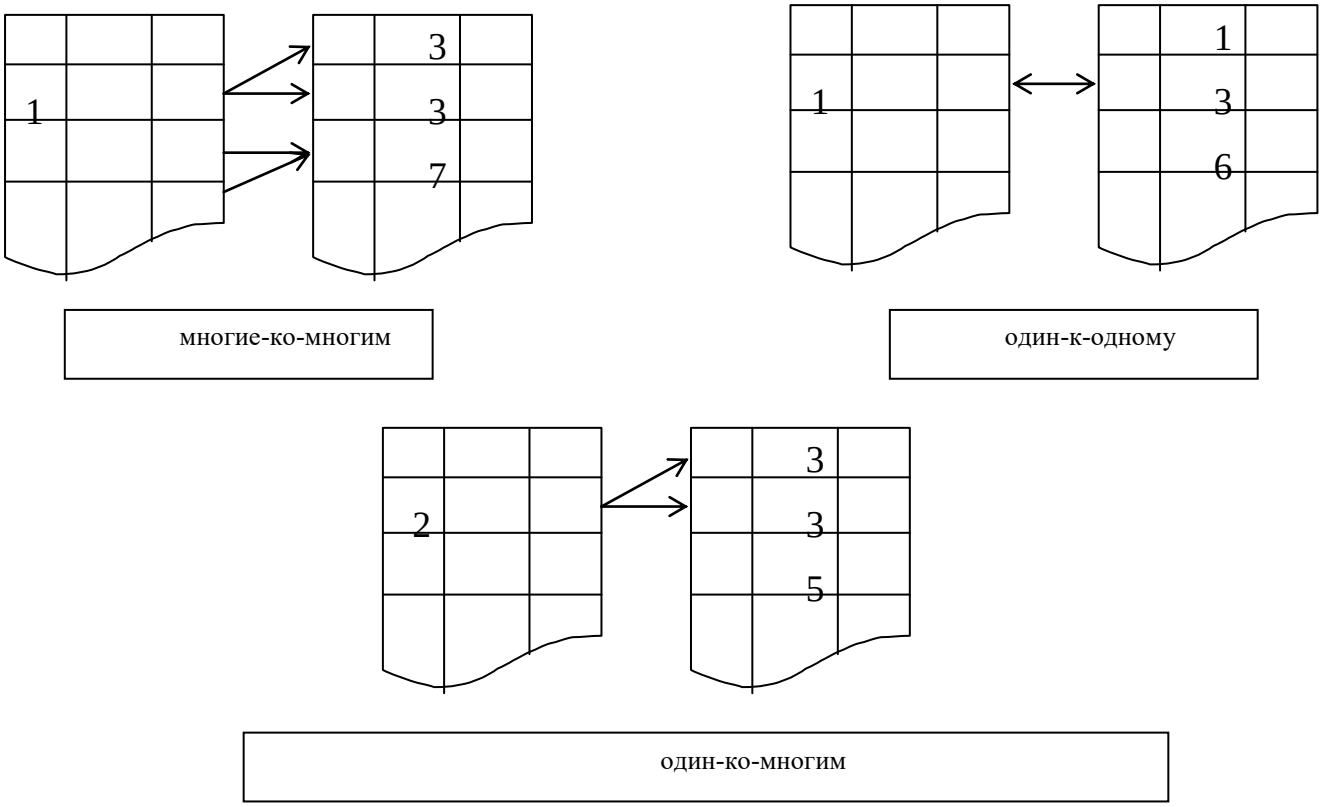
Как выбирать ключевые выражения? Как мы видели, во многих случаях такой ключ естественно возникает из логики задачи (номер паспорт, табельный номер и пр. - ничто иное, как уникальные идентификаторы, т.е. первичные ключи). Самый простой (а потому и самый популярный) способ "на все случаи жизни" - кодировка; можно добавить в каждую из связываемых таблиц искусственное (т.е. не связанное с логикой задачи) ключевое поле (числовой или иной код) - конечно же, следя за тем, чтобы их значения были равны для связываемых отношением записей.

Так, скажем, можно добавить в таблицу ДОМА поле Код, а в таблицу ЛЮДИ - ссылочное поле Дом, проставляя для каждого человека в описывающего его запись значение кода дома, в котором он живет.



Наиболее тривиальный способ - добавить все поля (образующий по определению первичный ключ) родительской таблицы в таблицы дочерней явно отпадает по меньшей мере по соображениям компактности хранения (о недопустимости дублирования информации - см. далее; исключение – связь 1-1, каждый человек живет в своем доме). Так, для того, чтобы отразить в нашей

модели отношения вида “(Этот) человек живет в (этом) доме”, мы должны будем внести значение первичного ключа таблицы ДОМА (адрес дома) в соответствующие поля таблицы ЛЮДИ.



Пример использования кодировки для реализации отношений.

§ 4. Проектирование БД как сжатие информации.

Какую структуру БД считать оптимальной? Не смотря на серьезные сложности как теоретического, так и практического характера, центральной руководящей идеей проектирования служит очевидный

Основной принцип – *не дублируй информацию* (за очевидным исключением ключей-ссылок)! Каждая ее часть должна храниться в одном, и только одном месте - таблице, определение которой согласовано с логикой

задачи. Сведения, которые могут быть выведены (вычислены) – не информация и не должны храниться вообще.

Кроме очевидной цели компактного хранения (а БД - это большие хранилища данных, которые могут содержать миллионы записей; при таких объемах выигрыш или проигрыш в один символ при хранении каждой записи может оказаться весьма ощутимым), недопустимость дублирования вызвана и не менее важной причиной сложности модификации - при наличии дублирования, при коррекции одной копии необходимо корректировать и все остальные (а для этого, как минимум, программист должен вспомнить - а где же они все же расположены?)

Вернемся к примеру с жителями. При ожидаемом реальном наличии большого числа "пересечений" в полях таблицы ДОМА (т.е. достаточно большого числа повторений названий городов и улиц), можно

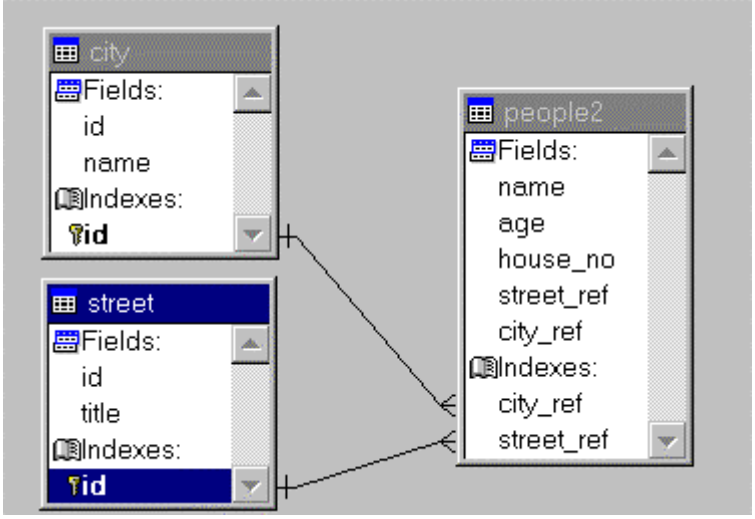
а) завести справочные (кодировочные) таблицы

УЛИЦЫ={<код_улицы, название_улицы>} и

ГОРОДА={<код_города,название_города>},

сопоставляющую длинным названиям улиц и городов некоторые краткие коды с тем, чтобы иметь возможность хранить в таблицах ДОМА и ЛЮДИ более краткие ключи вида

<номер_дома, код_улицы, код_города>



или же

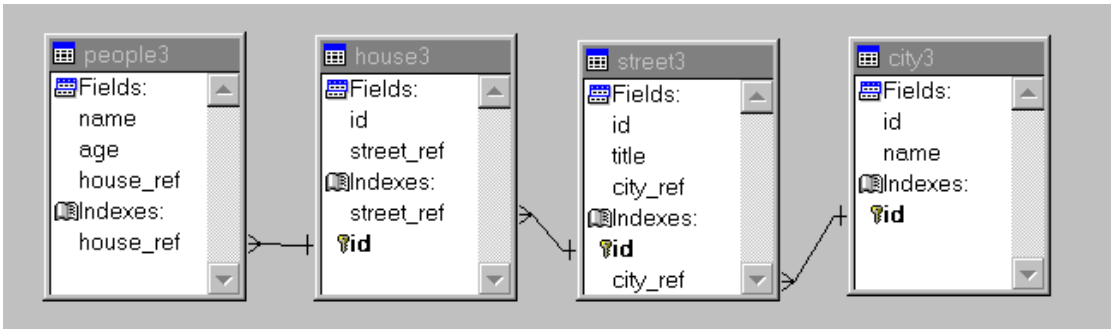
б) связать улицы с номерами имеющихся на них домов, а города – с названиями имеющихся в нем улиц, определив структуру введенных таблиц следующим образом

ЛЮДИ={<код_дома,...(другие атрибуты людей)>}

ДОМА={<код_дома, номер_дома, код_улицы,... (другие атрибуты домов)>},

УЛИЦЫ={<код_улицы, название_улицы, код_города>},

ГОРОДА={<код_города, название_города>},



Очевидно, что осуществив разбиение (декомпозицию) информации о связи между домами и улицами, а также улицами и городами (которую мы намеревались первоначально хранить в таблице ДОМА) и вынеся ее в отдельные таблицы УЛИЦЫ и ГОРОДА, мы получили новые связи “один-ко-многим” между таблицами УЛИЦЫ-ДОМА и ГОРОДА-УЛИЦЫ.

Приведенный пример показывает, что

соображения физической эффективности хранения и обработки данных могут потребовать уточнения (декомпозиции), но никак - *не ломки!* - начальной логической схемы. (Вспомним наш старый принцип - семантика основа прагматики!)

Будем считать в дальнейшем, что мы приняли последний вариант решения. Почему? Очевидно, такое сжатие будет тем более эффективным, чем больше его “коэффициент” - среднее количество дочерних записей, приходящихся на одну родительскую запись. В худшем случае – связи “один-ко-одному” – мы в реальности лишь увеличим объем хранимой информации за счет введения избыточных кодов. Таким образом, (весьма нелегкая в практическом воплощении) стратегия определения структуры БД должна преследовать цель максимизации коэффициента сжатия.

Что мы делали, с точки зрения математики?

Соединением (композицией - join) двух таблиц (отношений) $PT \subset X_1 \times \ldots \times PK \times \ldots \times X_n$, $CT \subset Y_1 \times \ldots \times FK \times \ldots \times Y_m$ по связи PK,FK называют таблицу

$J \subset X_1 \times \ldots \times PK \times \ldots \times X_n \times Y_1 \times \ldots \times FK \times \ldots \times Y_m$,

$J=\{<x_1,\ldots,pk,\ldots x_n,y_1,\ldots,fk,\ldots y_m>: <x_1,\ldots,pk,\ldots x_n> \in PT \ \& \ <y_1,\ldots,fk,\ldots y_m> \in CT \ \& \ pk=fk\}$

(слегка различные определения получим, если оставить в J лишь одно из равных значений pk,fk или исключить оба значения)

Соответственно, *декомпозиция* – разбиение таблицы на две, композицией которых по некоторому отношению она является.

Теперь можно переформулировать наш основной принцип.

- *Осуществляй декомпозицию* (с учетом кардинальности связи)

И все-таки, пока наш совет слишком общий для практического применения. Когда, в каких случаях можно и нужно осуществлять декомпозицию, какова стратегия декомпозиции?

§ 5. Понятие о нормализации БД.

Теоретическим фундаментом такой стратегии является понятие *нормальной формы* таблиц и, соответственно, интерпретация процесса проектирования как последовательного приведения к нормальным формам. Считается, что практически приемлемый результат дает приведение к первым 3 нормальным формам.

Первая н.ф. уточняет определение таблиц. Все поля таблиц должны быть *уникальными атомами,* а все содержательные отношения должны определяется с помощью внешних связей таблиц.

Под атомарностью имеется в виду неделимость логическая, т.е. зависящая от постановки задачи. Скажем, фамилия – строка символов, но она в большинстве случаев она неделима для нас как логическая единица, в то время как адрес – делим. Уникальность также понимается в логическом смысле. Поля могут быть одного типа, но они должны описывать разные атрибуты моделируемого объекта (в терминологии реляционной теории – принадлежать к разным *доменам*).

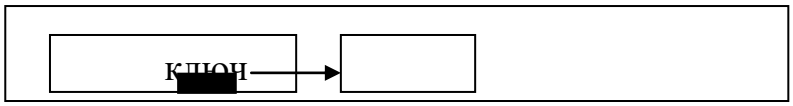
В отличие от изначального для реляционного подхода понятия отношения, таблицы как произвольного множества записей, в основе 2 и 3 нормальных форм (н.ф.) лежит понятие *таблично задаваемой функции* (графика, соответствия), взгляд на таблицу как график *зависимости* значений не ключевых полей от значений ключевых.

Пусть R – таблица, $R \subseteq T_1 \times \ldots \times T_n$. Поле T_i *зависит* от поля T_j, если

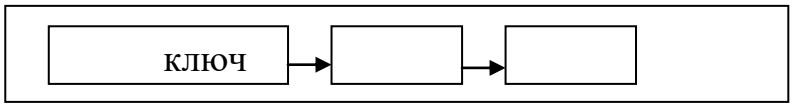
$\forall$ r1,r12$\in$ R ($r1_i = r2_i \rightarrow r1_j = r2_j$).

Аналогично определяется зависимость группы полей от группы полей (и выражений). По определению, все поля зависят от ключа.

Вторая н.ф. требует, чтобы в случае составных ключей ее аргументом служил *весь* ключ, а не отдельные входящие в него поля; т.е. для любой части ключа и любой части записи не существовало зависимости от части ключа.



Третья - то, чтобы ее аргументом служил *только* ключ, т.е. значения каждого не ключевого поля не зависели от значений никаких других полей, кроме ключевых. (В противном случае говорят о *транзитивной* зависимости)



Проиллюстрируем процесс нормализации на примере базы данных, описывающий факты купли-продажи: «Покупатели (Customer) делают заказы (Order) на товары (Product) продавцам (Employee)”

Первая нормальная форма. Атомарность.

Положим, что в контексте ситуации нас интересуют лишь следующие их характеристики «сущностей» Customer, Employee, Order и Product

Customer – cName (имя покупателя),

Employee - eName (имя продавца)

Order – oTotal (сумма заказа), oDate (дата заказа)

Product – pName (название товара), pPrice (цена товара), pAmount (количество проданного покупателю товара)

Положим также, что каждая из основных сущностей имеет некоторый естественно определяемый первичный ключ:

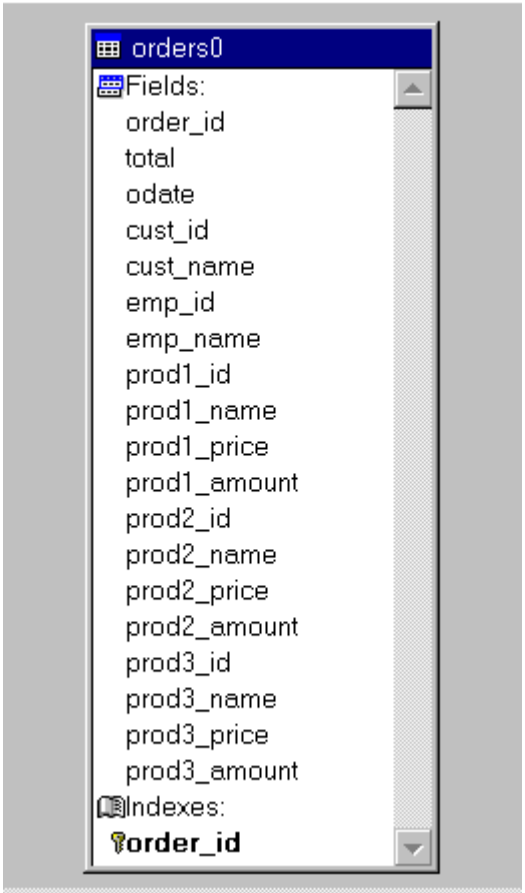
Cust_Id – скажем, номер паспорта покупателя

Emp_Id – табельный номер продавца

Order_Id – учетный номер заказа

Prod_Id – артикул товара

Изначально, мы имеем сложное 4-арное отношение между сущностями Customer, Employee, Order и Product, что соответствует самой простой и весьма далекой от оптимальной “плоской” структуре БД, состоящей из единственной таблицы.



Формально - нам нужно осуществить его декомпозицию (разбиение) на совокупность бинарных отношений, что мы и могли бы сделать, нормализуя исходную БД. Естественнее, однако, попытаться определить в логических терминах отношения между сущностями. Если не формальное определение, то смысл последних явно следует из постановки задачи.

Customer/Order – Покупатель делает заказ

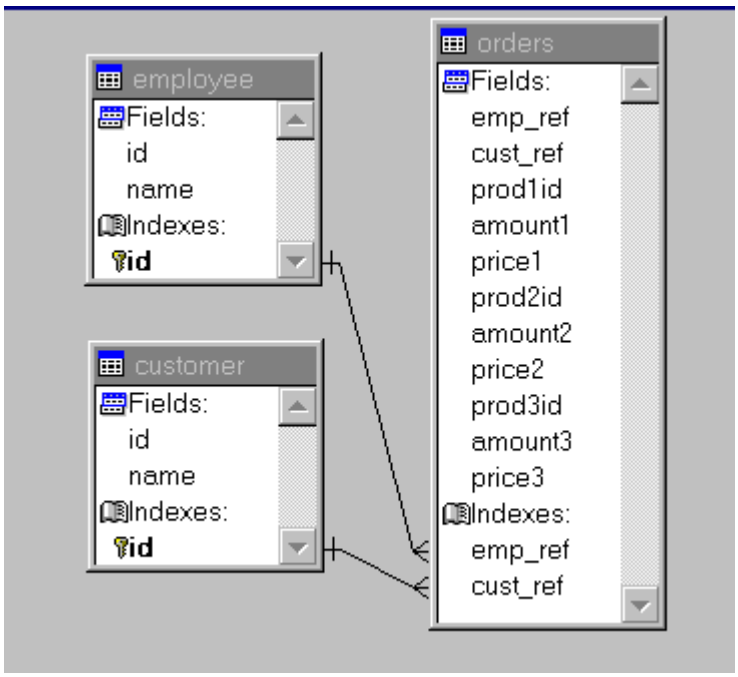
Employee/Order – Продавец принимает/исполняет заказ

Order/Product – Заказ состоит из (включает в себя) набора товаров

(отметьте – все отношения “один-ко-многим”)

Все остальные отношения между сущностями явно вторичны, косвенны, не бинарны – скажем отношение продавец/покупатель явно подразумевает наличие заказа. (Здесь и далее мы стараемся делать самые слабые допущения относительно предметной области. Нетрудно представить ситуацию, когда присутствуют и другие отношения между сущностями. Скажем, в данной фирме каждый покупатель может быть “прикреплен” к конкретному продавцу, что соответствуют наличию прямой связи Employee/Customer).

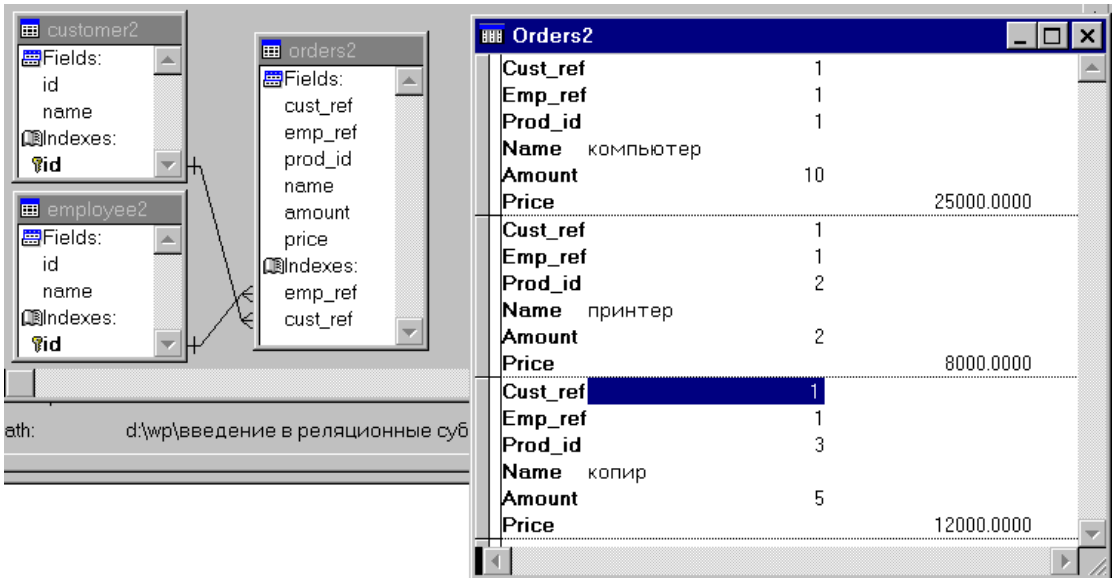
Первые два отношения нетрудно реализовать добавлением в таблицы Customer и Employee ключевых полей CustId и EmpId, и добавлением соответствующих полей - внешних ключей CustRef и EmpRef в таблицу Order. Ситуация с отношением Order/Product чуть сложнее. То, что сущность/понятие «заказ» включает в себя несколько экземпляров сущностей «товар» провоцирует на то, что структура таблицы Order должна включать в себя несколько ссылок на таблицу Product – скажем, ProdRef1,…, ProdRefn. Технически, это возможно, если известна явная константа – максимальное число товаров в заказе – скажем, не более 3. Но с точки зрения 1 НФ использование таких неявных отношений “один-ко-многим” недопустимо.



Плохо!

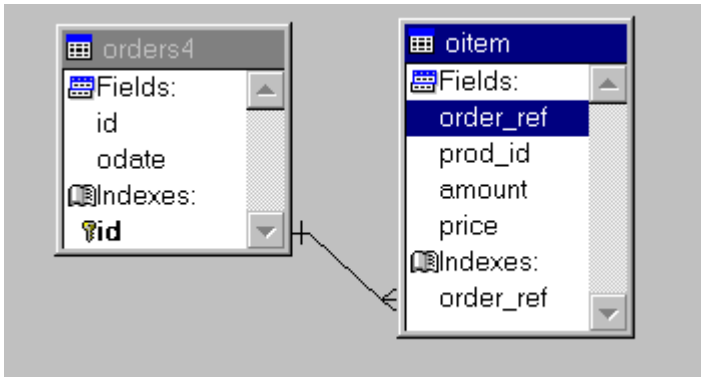
Замечание. Нормальные формы – лишь ориентир при проектировании. Реальная практика часто требует более тонкой классификации отношений и учета кардинальности связей. “Статичные” связи вида 1:Const и даже Const1:Const2 нередко на практике закладываются в определение таблиц. Пример – поквартальные или помесячные сводки – соответственно, связи вида 1:4 и 1:12. Категорически запрещается лишь фиксировать в структуре таблицы лишь отношения с переменной кардинальностью.

“Лобовой” способ удовлетворить 1РФ – превратить статичную структуру полей в динамическую структуру записей, изменив структуру таблицы Orders так, что бы «дублирующие сущность» поля Product1,Product2,… стали записями этой таблицы:



Плохо!

Однако, это приводит к тому, что теперь группа полей зависят от OrderId (собственно заказа), а другая часть – от OrderId,ProdId (отдельного пункта заказа), что противоречит 2НФ. Нужно оставить лишь поля – атрибуты самого заказа, выделив все лишнее – атрибуты *экземпляра* товара, *пункта* заказа – в отдельную таблицу. Это заставляет нас выделить явно *отношение включения* между заказом и экземпляром товара (OrderItem).



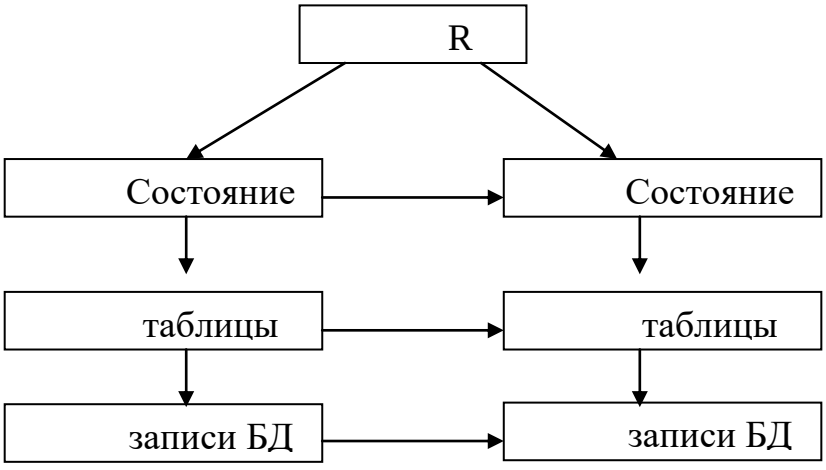
Наконец, посмотрим на таблицу oItem. Является ли пара Order_Ref, Prod_Id первичным ключом, т.е. могут ли в одном заказе одинаковые товары? Положим, что да. Для идентификации пункта добавим поле item_no – номер

пункта заказа. Тогда Order_Ref, Item_No – первичный ключ, он определяет товар (снова - в том лишь смысле, что для данного заказа и номера, существует единственный товар этого заказа с таким номером). Но цена товара определяется самим товаром, т.е. Prod_Id, не входящим в первичный ключ. Нарушено требование 3 н.ф. Выносим отношение Prod_Id/Price в отдельную таблицу Product и добавляем связь между oItem и Product. Логически, это связь - *отношение принадлежности* между товаром и экземпляром товара.

"Мораль" . Многие проблемы можно предупредить или решить на раннем этапе проектирования.

§ 6. Модификация БД. Нарушения целостности.

Итак, если логически реляционные БД – совокупность таблиц, связанных отношениями, декомпозиция некоторого отношения R на бинарные, то, с точки зрения реализации – это просто набор таблиц (файлов записей).



Не удивительно поэтому, что с точки зрения реализации, модификация (преобразование состояния) БД – преобразования таблиц, сводящиеся в свою очередь, к преобразованиям записей соответствующих файлов. *В процедурно-*

реляционно, «навигационных» языках (например, в языках популярного семейства языков СУБД, ориентированных на «малые» БД для персональных компьютеров xBase-dBase-FoxPro) алгоритмы преобразования записей таблиц осуществляются программистом, в *декларативно* (“собственно”) *реляционных* языках (пример – язык SQL, изначально – командный язык мэйнфреймов, «больших компьютеров» фирмы IBM) программист задает лишь спецификацию таких преобразований.

С точки зрения реализации, преобразование БД есть либо

- добавление (insert, append)
- удаление (delete)
- преобразование/модификация значения (replace, update)

Соответственно, в процедурных языках БД присутствуют команды добавления, удаления *записей* и модификации значений полей *записей,* в декларативных языках – команды добавления *таблицы* к таблице, удаления *подтаблицы* и изменения значений в *(под)таблице*.

Конкретную структуру и синтаксис команд классического декларативного языка мы рассмотрим ниже, сейчас отметим более существенный момент.

Внимание - проблема. Очевидно, любая модификация БД должна сохранять ее семантику (т.е. отношение R), преобразовывать одно корректное (т.е. верное, с точки зрения предметной области) состояние базы в другое. Реализация преобразований допускает некорректные состояния. В этом случае говорят о *нарушении целостности* данных БД.

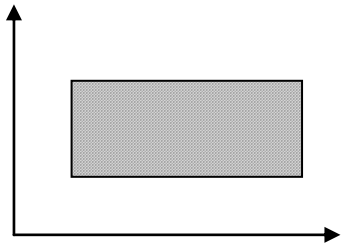
Разумеется, это не есть специфическая для СУБД проблема, но именно здесь она приобретает наибольшую остроту. Ведь в отличие от процедурного программирования, СУБД изначально ориентированы на долговременное

хранение данных и их корректность имеет здесь наивысший приоритет.

Нарушение целостности может происходить по 2 причинам. Первая из них связана с очень слабым аппаратом описания типов данных. Это может проявляться в следующих нарушениях целостности:

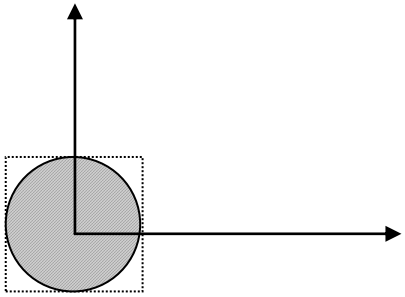
0) Нарушения “уровня столбца” (пример – уникальность, но не упорядоченность!)
1) Нарушения “уровня поля”, неточное описание доменов - областей допустимых, с точки зрения логики предметной области, значений полей.

Пример. Описание прямоугольника $R=\{<x,y>: x \in [1,3] \& y \in [1,2]\}$ в качестве таблицы (именованного отношения) как подмножества D^2 (D – множество вещественных чисел) – что верно, но не точно, поскольку технически допустима, например, запись $\{x \rightarrow 0, y \rightarrow 0\}$



2) Нарушения “уровня записи”. Неточное описание таблицы как отношения, взаимосвязи значений полей.

Пример. Описание круга $R=\{<x,y>: x^2+ y^2 \leq 1\}$ как подмножества D^2



Отметьте – ситуация действительно отличается от 1). Даже точное описание доменов полей – $R \subset [-1,1]^2$- не спасает от возможных ошибок.

Решение таких проблем почти очевидно – нужно описывать таблиц более точно, за счет добавления к определению таблиц предикатов, ограничивающих множества допустимых состояния полей и записей. Такие предикаты называют *правилами допустимости* (значений)– validation rules.

Нарушения “уровня связей”. Второй тип ошибок следует непосредственно из реализации. Команды СУБД преобразуют таблицы, а не связи, в то время как любое нетривиальное преобразование предполагает согласованное преобразование двух и более таблиц. Корректное изменение состояния таблицы не означает корректного изменения состояния БД.

Скажем, логическое действие “удалить из БД информацию о данном заказе” означает не только удаление соответствующей записи из таблицы Orders, но и удаление ее дочерних записей (пунктов заказа) из таблицы Items и даже, возможно, удаление информации о покупателе, сделавшем этот заказ в случае, если в БД больше нет информации о его заказах.

Данные – не только область допустимых значений. Описание данных предполагает и описание класса допустимых преобразований.

В современных СУБД рассматривается частный класс преобразований двух связанных отношением таблиц, сохраняющих *ссылочную целостность (referential integrity).*

Преобразование двух связанных отношением “один-ко-многим” таблиц не сохраняет ссылочную целостность, если после его применения в дочерней таблице остаются “*сироты*” - т.е. записи с несуществующим в родительской таблице значением внешнего ключа.

Различают несколько стандартных типов сохранения ссылочной целостности, которые мы рассмотрим ниже.

§ 7. К эволюции реляционных БД. Технология клиент-сервер.

Итак, с точки зрения изначальной *логики строения* информации, **база данных (БД)** есть формальная модель некоторой предметной области в виде совокупности таблиц, связанных отношениями.

С точки зрения технологии "клиент-сервер", это традиционное определение нуждается в существенном добавлении - отражения *логики практически возможного и логически допустимого изменения данных*, реализуемой с помощью ***хранимых,*** вместе со значениями данных, **процедур**.

Эта точка зрения отражает фундаментальный сдвиг в методологии программирования, нашедший на сегодня свое наиболее полное выражение в концепции *абстрактного типа данных* (или - *класса*, в объектно-ориентированном программировании, компонента в COM-).

Данные (информация) - это
не только (даже - не столько) значения,

Таким образом, понятие данных приобретает существенно *динамический* характер. С другой стороны, понятие абстрактного типа данных *статично* в том смысле, что система типов должно быть предварительно определена, хотя бы в виде абстрактной *схемы*, до разработки конкретных алгоритмов, с учетом всех возможных в дальнейшем случаев их использования. Такая существенная перемена приоритетов выдвигает на передний, центральный план этапы концептуального анализа и проектирования, с использованием соответствующих логико-математических методов. По статистическим оценкам, соотношение между этапами проектирования и собственно кодирования составляет около 80/20 - пропорция, *обратная* по отношению к традиционным методам!

Для того, чтобы в процессе модификации - а именно, вставки (insert), обновления (update) и удаления (delete) записей - БД не перестала быть **целостной**, т.е. не перестала адекватно отражать реальное содержание предметной области, необходимо выполнение следующих требований - правил целостности, или "бизнес-правил":

- таблицы БД не должны содержать нереальных данных, которые могут появиться как следствие ошибок программ обработки БД или некорректного ввода. Для этого в определение таблиц БД добавляют **правила проверки** (validation rules) значений поля или группы полей, определяющие необходимые условия их корректности;

  так, например, значение поля номер_дома в таблице ДОМА должен принадлежать некоторому допустимому интервалу натуральных чисел – скажем, от 1 до 1000 (при более точном моделировании мы могли бы отражать встречающиеся в реальности номера домов вида натуральное-буква и сформулировать более сложное условие проверки корректности значения поля);

В том - крайне нежелательном и опасном! - случае, когда задача моделирования подразумевает наличие связи между таблицами, но фактически некоторая запись дочерней таблицы не ссылается ни на одну запись родительской таблицы (т.е. содержит отсутствующий в родительской таблице значение ключа), то такую запись называют **сиротой** (orphan). По соглашению, запись с неопределенным явно, т.е. равным NULL, ключом сиротой *не* считается.

Так, в нашем примере запись в таблице ЛЮДИ будет сиротой, если она содержит ссылку на не существующий адрес дома – отсутствующее в таблице ДОМА значение поля КодДома. Запись со значением ссылки NULL ("временно не имеющий места жительства") сиротой не будет.

- БД должна удовлетворять условию **ссылочной целостности** (referential integrity), т.е. в таблицы БД, - в результате тех же причин - не должны попадать записи-"сироты". Следовательно, модификация связанных таблиц должна быть согласованной. К наиболее простым и популярным вариантам такого согласования относят ситуации, когда модификация записи родительской таблицы является

  a) **каскадной** (cascades), т.е. продолжается на соответствующие записи дочерней таблицы;

  b) **нулевой** (nulls), т.е. устанавливающей ключи дочерних записей в NULL;

  c) **ограниченной** (restricted), т.е. возможной и исполняемой лишь в тех случаях, когда у нее нет ни одной дочерней записи.

Как и правила проверки, варианты сохранения ссылочной целостности БД также являются частью ее определения. Обеспечивающие целостность

стандартные хранимые процедуры называются **триггерами.**[2] Разумеется, более сложные варианты согласования модификации двух и более таблиц могут потребовать реализации соответствующих пользовательских процедур-триггеров.

Так, в нашем примере наиболее естественно принять ограниченный вариант удаления – информация о доме не может быть удалена, пока таблица ЛЮДИ содержит хотя бы одну запись о человеке, в этом доме проживающем, и каскадный вариант модификации значений – изменение первичного ключа в любой записи таблицы ДОМА (т.е. одного из полей номер_дома, код_улицы, код_города) должно автоматически изменять атрибуты соответствующих записей дочерней таблицы ЛЮДИ.

С точки зрения реляционного подхода, все команды СУБД рассматриваются как преобразования содержимого одной или нескольких таблиц, результатом которых снова является таблица (новая или обновленная старая):

$$T := F(T1,\ldots,Tn)$$

(Стандартный) **SQL** – фактически, язык, состоящий из одних присваиваний. SQL - программа - композиция таких присваиваний.

К командам **модификации** (редактирования таблиц) БД относят команды вставки, удаления и изменения групп записей таблицы.

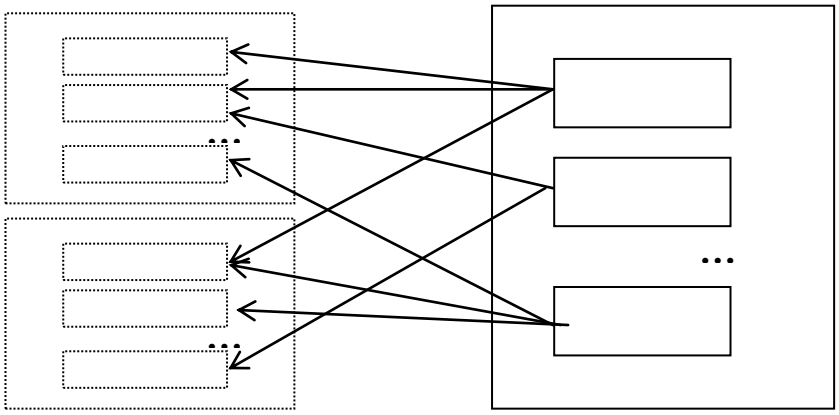
§ 8. Представления.

[2] Термин trigger - "спусковой крючок огнестрельного оружия" - подразумевает автоматический запуск такой процедуры при попытке модификации родительской таблицы.

В отличие от них, результатом команды **запроса** к БД является абстрактная, "виртуальная" таблица, не существующая физически в виде постоянно хранимого файла, но лишь отражающей текущее состояние реально физически хранимых, или – **базовых** таблиц БД. Цель использования таких таблиц очевидна – отобразить в развернутом, полном и доступном для пользователя виде – причем разном, для разных групп пользователей - интересующего именно его минимально достаточную по объему информацию о реальных объектах предметной области по компактно закодированной информации, содержащейся в базовых таблицах.

Так, естественная, для большинства пользователей, информация о домах должна включать не о введенных нами в целях экономии хранения кодах улиц и городов, но об их реальных названиях, хранящихся в базовых справочных таблицах УЛИЦЫ и ГОРОДА и, конечно, другие атрибуты домов из базовой таблицы ДОМА. В одних случаях, эта информация должна быть полной, включающей атрибуты всех проживающих в этих домах людей, в других - содержать лишь количество проживающих, относится лишь к одному городу и проч.

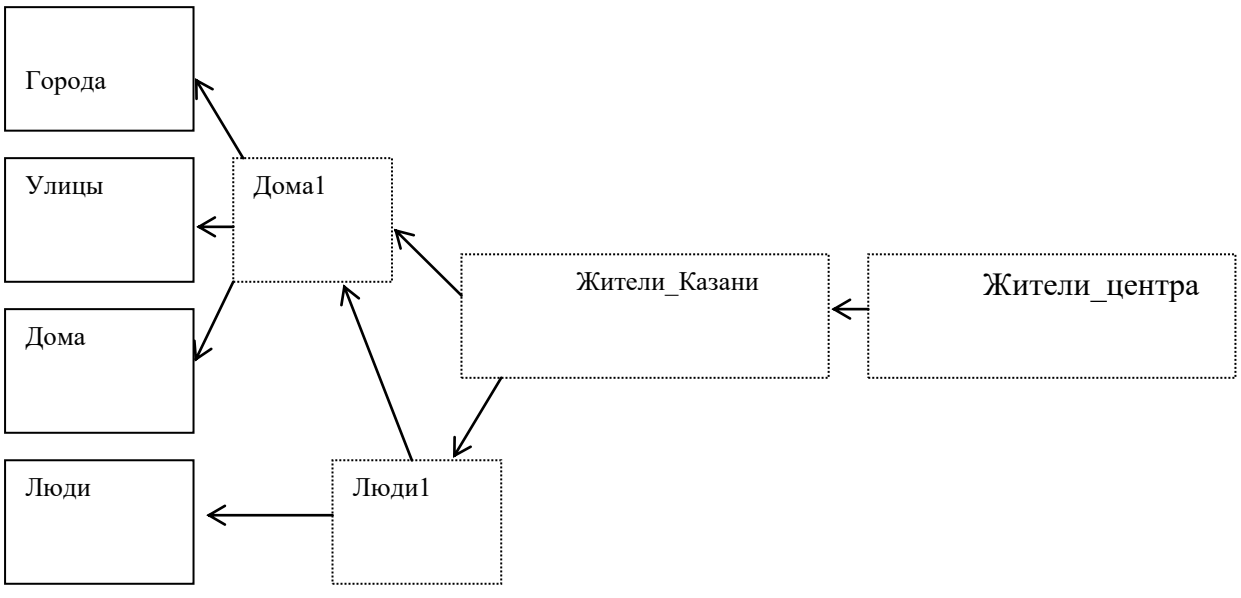
В реальности, каждая группа пользователей определяет некоторую специфическую схему сущностей и связей, которая и является, с точки зрения данной группы, логическим содержимым БД. С точки зрения реализации, такая



"виртуальная БД" есть набор именованных запросов или **представлений**, связываемых с данной группой пользователей.

Итак, **представления** (views) – это абстрактные (пользовательские) таблицы, *определение* (но *не содержимое!)* которых храниться в виде именованных команд запроса как часть БД. Подобно базовым таблицам, представления могут служить аргументами других запросов и представлений, позволяя таким образом программисту определять сложные родо-видовые иерархии объектов предметной области.

Так, в нашей модели-примере естественно определить представление ДОМА1 – именованном запросе, “хранящем” раскодированную по таблицам ДОМА, УЛИЦЫ и ГОРОДА полную информацию о домах в виде, естественном для пользователя БД, и представление ЛЮДИ1, “содержащую” (с точки зрения пользователя БД) информацию о людях, включающую ссылки на представление ДОМА1 и базовую таблицу ЛЮДИ. В свою очередь, эти представления могут служить базой более конкретных представлений ЖИТЕЛИ_КАЗАНИ и ЖИТЕЛИ_ЦЕНТРА.

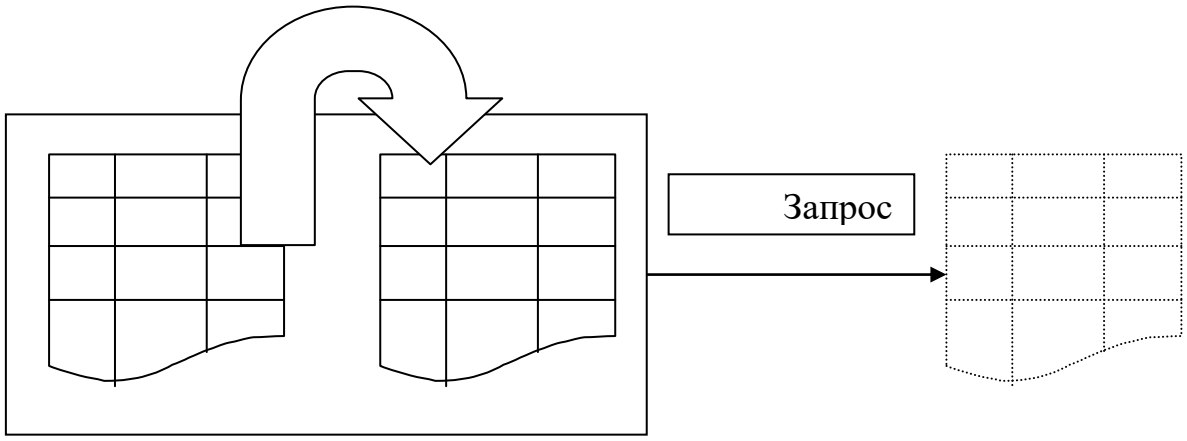


Пример иерархии представлений. Здесь стрелки отмечают направление ссылок определений (обратное направлению данных при запросах)

В какой степени можно относиться к абстрактным таблицам также, как к реальным, физически существующим? С точки зрения пользователя, “содержимое” **модифицируемых** представлений можно редактировать с помощью команд модификации – но поскольку такового, в реальности, не существует, последнее может означать лишь подходящую модификацию базовых таблиц.

Так, изменение, номера дома в некоторой записи представления ЖИТЕЛИ_ЦЕНТРА в реальности может означать лишь изменение значения дочернего ключа код_дома в таблице ЛЮДИ.[3]

По существу, модификация, т.е. преобразование представления T, основанном на запросе S к базовым таблицам T1,...,Tn, T=S(T1,..,Tn), к некоторому новому состоянию T’, означает требование к СУБД найти такое модификацию (преобразование) базовых таблиц к новому состоянию T1’,...,Tn’, которое вело бы к желаемому содержимому представления, T’=S(T1’,..,Tn’).



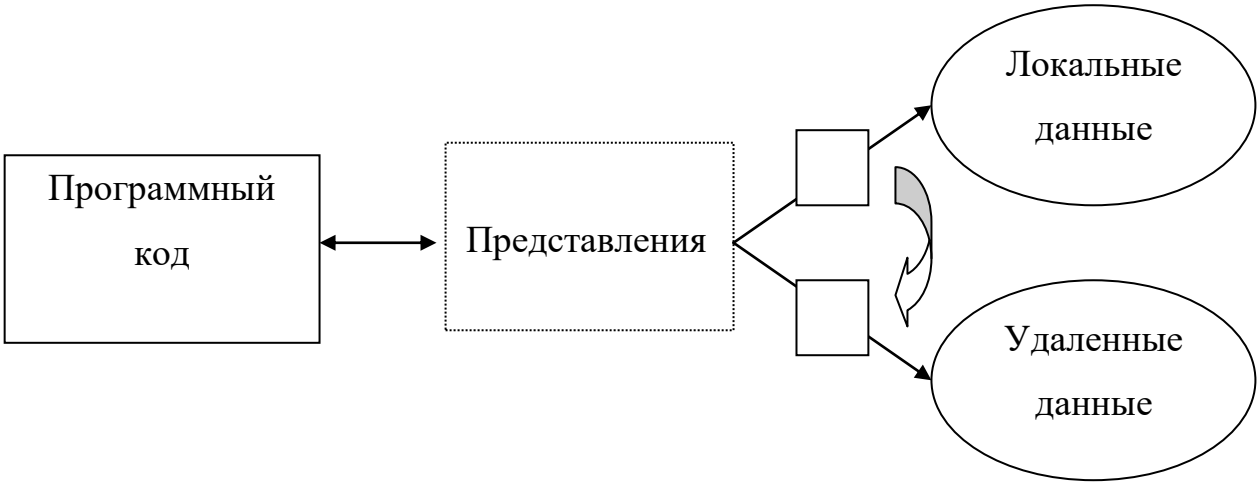
(i) Модификация представления в действительности означает модификацию данных базовых таблиц

[3] Неявно подразумевается случай, когда данный человек переезжает в дом, информация о котором уже содержится в БД. Что делать в противном случае? И что делать с записью о доме, на которую перестали ссылаться какие-либо записи о людях? Отметьте, что уже этот простой пример порождает проблемы, не покрываемые стандартными вариантами согласования изменений.

Понятно, что в целом такое обратное преобразование технически трудно реализуемо и, даже теоретически, может быть неоднозначным или просто не существовать; помимо этого, различные СУБД накладывают дополнительные ограничения на критерий модифицируемости представлений, исходя из соображения эффективности генерации требуемой модификации.

Пример неосуществимости модификации представления. Заведем в нашей модели-примере представление, содержащее единственное поле – средний доход людей, информация о которых хранится в таблице БД. Как повысить средний доход – т.е. изменить значения поля доход в таблице ЛЮДИ? Как подсказывает математика (и наш жизненный опыт), решение этой задачи далеко не однозначно.

Помимо фундаментальной роли в построении/определении логики программной системы, понятие представления служит для сокрытия реализации не только в смысле сокрытия реальной структуры БД, но и сокрытия реальных *источников данных*, которые могут быть как локальными (находится физически на клиенте), так и удаленными (находится на сервере). Этот факт существенно используется для создания и безопасного для реальных данных тестирования локального прототипа программной системы, оперирующего локальными представлениями (базирующихся на локальных копиях таблиц уже существующей или будущей серверной БД), после окончательной отладки превращаемого (полу)автоматической заменой ссылок в реальную систему, оперирующую с удаленными данными. Такое преобразование локального прототипа в реальное сетевое приложение называется *подъемом* (upsizing) БД.



Подъем БД. Базовые таблицы представлений:

1 - на этапе отладки

2- в процессе реального функционирования

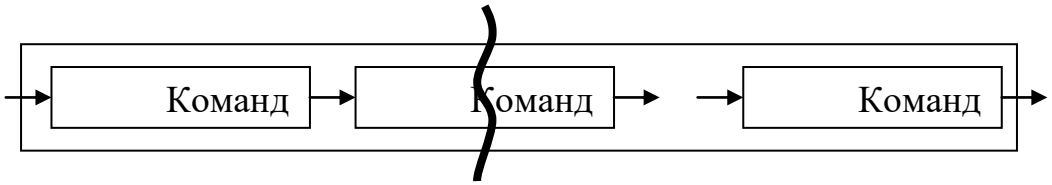
Введенное ранее понятие целостности БД - важнейший случай более общего понятия *безопасности данных,* подразумевающего включение в рассмотрение не только логической схемы самой БД, но и некоторую схему *всех аспектов ее использования* - от физического размещения и сохранности данных до определения прав пользователей - относимым к вопросам ведения или *администрирования* БД. Не касаясь здесь сколь-нибудь подробно этих вопросов, отметим лишь как существенно бóльшую безопасность данных, так и бóльшую сложность задач администрирования при использовании технологии "клиент-сервер"[4]. Одним из наиболее важных программных средств обеспечения безопасности данных является механизм поддержки транзакций.

Транзакция - логически единая операция по модификации базовых таблиц, состоящая из одной или нескольких команд СУБД, переводящая БД из одного целостного, корректного, с точки зрения предметной области, состояния в другое, также логически корректное. Поддержка транзакций гарантирует, что входящие в транзакцию команды либо выполнятся полностью, либо – если последнее невозможно по какой-либо причине (сбой энергоснабжения, конфликт действий пользователей в многопользовательской/ сетевой БД и пр.)

[4] Что, впрочем, касается в большей степени крупных серверных СУБД типа Oracle и MS SQL Server, а не таких СУБД клиентского типа, как рассматриваемая здесь Visual FoxPro.

– полное восстановление состояния БД на момент начала транзакции (или иной выбранной программистом контрольной точки соответствующей процедуры).

Примером транзакции может служить триггер каскадного удаления записи родительской таблицы, с последующим удалением записей дочерних таблиц. Все удаления должны быть либо осуществлены полностью, либо не осуществлены вообще – в противном случае в БД могут появиться записи-сироты.



Подобно традиционному понятию процедуры, транзакция – логическая операция, реализуемая с помощью команд языка. Отличие - в явном допущении возможности лишь частичного выполнения. Основной принцип транзакции - "все или ничего". Сбой при выполнении любой команды ведет к откату – невыполнению всей транзакции.

Помимо проблем логической целостности и компактного хранения данных, проблемы обеспечения высокой скорости доступа к данным также играют в СУБД немаловажную роль. Эти проблемы решаются в основном, за счет применения тех или иных вариантов алгоритмов быстрого дихотомического поиска в таблицах - требующих компактного хранения нужного порядка расположения записей таблиц в отдельных **индексных файлах.** Исходя из предполагаемой частоты тех или иных запросов, пользователь СУБД может затребовать создания ускоряющих выполнение этих запросов индексных файлов. В силу своей особой роли ключевые поля таблиц, как правило, индексируются автоматически.

Итак, с точки зрения логики моделирования

2. База данных =

3. (Базовые) Таблицы + Отношения + Правила целостности + Представления

а также, с точки зрения реализации - обеспечивающие выполнения правил целостности *хранимые процедуры* (в том числе *триггеры*) и ускоряющие обработку *индексные файлы.*

§ 9. Почему SQL? Введение в технологию "клиент-сервер".

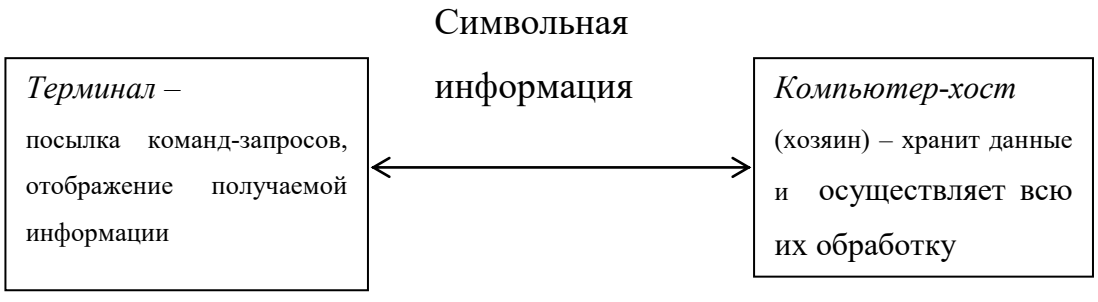
Как показывает практика, системы управления баз данных (СУБД) составляют наиболее значительную часть всех используемых программных систем – при этом часть, все более увеличивающуюся в связи с развитием локальных и, особенно, глобальных компьютерных сетей. Язык структурированных запросов SQL (Structured Query Language)[5] фактически играет роль *стандарта языка коммуникации* сетевых СУБД, часто работающих под управлением различных операционных систем и использующих разные форматы представления информации. "Встроенный" SQL не только является неотъемлемой частью любой современной СУБД, но и любой сколь-нибудь крупной системы программирования, ориентированной на использование технологии "клиент-сервер". Среди прочих достоинств SQL отметим максимальную *компактность*, обеспечивающую при этом полноценные средства обработки БД и логическую *простоту*, делающую SQL не только языком программирования, но и неоценимым средством проектирования и функционального описания достаточно больших

[5] В англоязычном мире принято произносить это сокращение как "Сиквэл", в русскоязычном же традиционно прижилось произношение "Эс-Ку-Эль".

программных систем. Сказанное объясняет несомненную важность изучения SQL.

Недостатки SQL – обратная сторона его достоинств. Стандартность SQL не идеальна – в действительности, различные СУБД использует отличающиеся диалекты языка, что делает далеко не тривиальным перенос определения и данных БД из одной среды в другую - в частности, т.н. "подъем" (upsizing) БД из локальной среды в сеть. Компактность и декларативность языка, отсутствие в нем традиционных структур управления (циклов, условных операторов и пр.) усложняет переход на SQL программистов, привыкших к процедурному программированию в стиле языков семейства xBase/dBase. В целом, характерный для SQL более абстрактный и высокоуровневый *реляционный подход*, т.е. взгляд на обработку данных как преобразование таблиц "в целом", сильно отличается по своей логике от характерного для языков этого семейства *навигационного подхода* - преобразования отдельных записей таблиц.

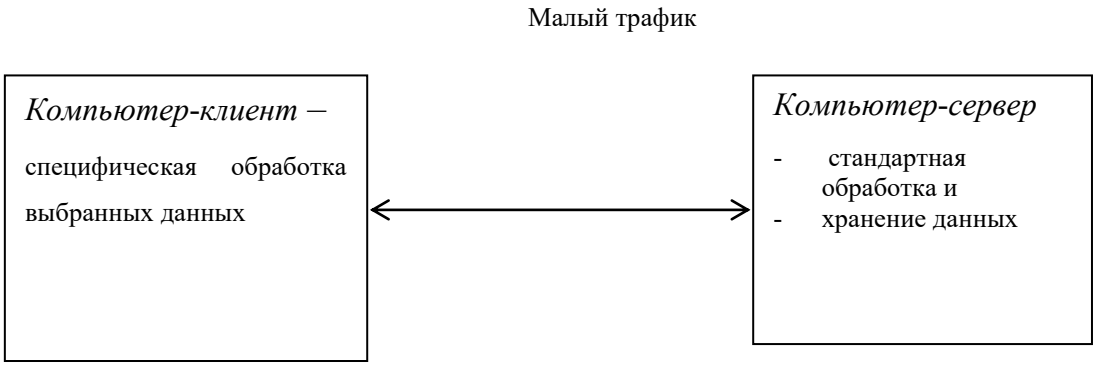
§ 10. К эволюции сетевых БД.



1970-е гг. (мэйнфреймы) Хост-архитектура

1980-е гг. (появление ПК) Архитектура файл-сервер - *разделение хранения и обработки данных*

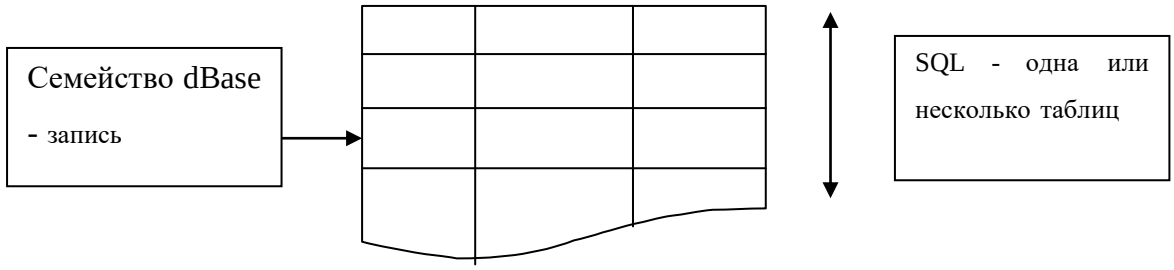
Как показывает принятое деление данных на локальные (local), т.е. хранимые на компьютере-клиенте и удаленные (remote), т.е. хранимые на сервере, в рамках этой технологии при необходимости возможно и распределение хранения данных.



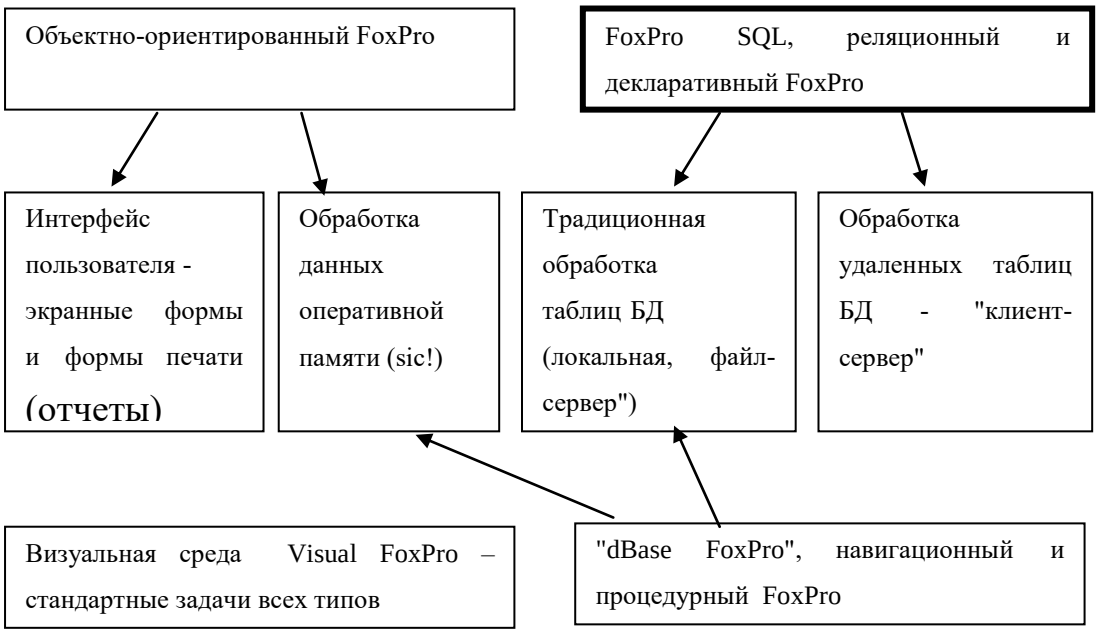
1990е гг. Архитектура клиент-сервер - распределение обработки данных

a)

§ 11. Навигационный и реляционный подход в СУБД FoxPro



Общее (в отличие от иерархической, сетевой и объектной моделей данных) - данные есть совокупность взаимосвязанных таблиц. Но если в первом случае объектом преобразований служат записи, то во втором таблицы. Несомненный интерес СУБД FoxPro для образовательных целей состоит в возможности сравнить оба подхода.



a) Сфера приложения FoxPro SQL,

b) в сравнении с другими частями языка

Во избежание недоразумений - Visual FoxPro не есть объектная или даже смешанная[6] СУБД. Аппарат ООП применяется лишь для обработки оперативных данных.

§ 12. Синтаксис FoxPro SQL

Традиционно, команды SQL делятся на команды (под)языка DDL (Data Definition Language – язык определения данных) и (под)языка DML (Data Manipulation Language – язык манипуляции данными).

Центральное место среди команд DML занимает команда создания таблиц CREATE TABLE

```
CREATE TABLE <table name>
   ( { <column name> <data type> I<size>]
[<colcnstrnt> ...]} .,.. );
   [<tabconstrnt>] .,.. );
```

где

ЭЛЕМЕНТ	ОПРЕДЕЛЕНИЕ
<table name>	Имя таблицы создаваемой этой командой.
<column name>	Имя столбца таблицы.
<data type>	Тип данных который может содержаться в столбце.
	Может быть любым из следующих:

[6] Под БД смешанного типа понимают реляционные, в целом, БД, способные хранить в таблицах объекты в качестве значений (в смысле ООП - не путать с OLE-объектами!).

	INTEGER	(ЦЕЛОЕ ЧИСЛО),
	CHARACTER	(СИМВОЛЬНОЕ),
	DECIMAL	(ДЕСЯТИЧНОЕ),
	NUMERIC	(ЧИСЛОВОЕ),
	SMALLINT	(НАИМЕНЬШЕЕ)
	FLOAT	(С ПЛАВАЮЩЕЙ ТОЧКОЙ)
	REAL	(РЕАЛЬНОЕ),
	DOUBLE PRECISION	(УДВОЕННАЯ ТОЧНОСТЬ С ПЛАВАЮЩЕЙ ТОЧКОЙ),
	LONG *	(ДЛИННОЕ *),
	VARCHAR *	(ПЕРЕМЕННОЕ СИМВОЛЬНОЕ) *,
	DATE	(ДАТА *),
	TIME	(ВРЕМЯ)
	(* - указывает на нестандартный для SQL тип данных)	
<size>	Размер.Его значение зависит от <data type>.	
<colconstrnt>	Может быть любым из следующих:	
	NOT NULL	(НЕ НУЛЕВОЙ),
	UNIQUE	(УНИКАЛЬНЫЙ),
	PRIMARY KEY	(ПЕРВИЧНЫЙ КЛЮЧ),
	CHECK(<predicate>)	(ПРОВЕРКА предиката),
	DEFAULT =	(ПО УМОЛЧАНИЮ =
	<value expression>	значимому выражению)
	REFERENCES <table name>	(ССЫЛКА НА имя таблицы
	[(<column name> .,..)]	[(имя столбца)])
<tabconstrnt>	Может быть любым из следующих:	

	UNIQUE	(УНИКАЛЬНЫЙ),
	PRIMARY KEY	(ПЕРВИЧНЫЙ КЛЮЧ),
	CHECK	(ПРОВЕРКА предиката)
	FOREIGN KEY(<column name>) (ВНЕШНИЙ КЛЮЧ)	
	REFERENCES <table name>	(ССЫЛКА НА имя таблицы
	[(<column name> .,..)]	[(имя столбца)].

```
Create Table Customer
(ID          integer       Primary Key NOT NULL,
 NAME        char(20),
CITY              char(20),
CREDIT      decimal,
BIRTHDAY         date
)
Create Table Employee
(ID          integer       Primary Key NOT NULL,
 NAME        char(20),
CITY              char(20),
COMM        decimal,
BIRTHDAY         date
)

Create Table Orders
(ID          integer       NOT NULL,
Cust_Ref     integer NOT NULL References Customers,
Emp_Ref      integer NOT NULL References Employee,
```

```
START       date,
FINISH      date
Check Start<Finish
)

Create Table Item
(Order_Ref  integer NOT NULL References Orders,
 Product_Ref        integer NOT NULL References Product,
Amount      integer
Update of Orders Cascades,
Delete of Orders Cascades
Delete of Product Restricted
)

Create Table Product
(Id     integer         Primary Key,
Name char(30),
Price   decimal,
Type char(10)
)
```

Как уже отмечалось выше, стандартность различных версий SQL далека от идеальной. Различные диалекты языка используют различные скалярные типы, функции и предикаты, поддерживают (или не поддерживают) различные опции команд и самый разнообразный синтаксис для них. В своем изложении языка мы следуем стандарту ANSI SQL.

Щадя читателя и следуя принципу "разумной достаточности", мы далее немного упрощаем синтаксис - в основном, за счет исключения синонимии и редко используемых опций. С другой стороны, мы стараемся упомянуть

наиболее часто встречающиеся на практике опции; такие языка, отсутствующие в стандарте (и, возможно - в используемой читателем версии) расширения SQL отмечены символом *.

Выражения и предикаты в SQL.

Итак, в рамках поставленной задачи нам необходимо и достаточно знания лишь "классического набора" выражений SQL, а именно - арифметических выражений и предикатов следующего вида:

а) **Сравнения**

Аргумент1 Знак_сравнения Аргумент2

где знак сравнения - один из символов отношений = (равенство), <>, !=, # (неравенство) и >, >=,<,<=, а также, для символьных строк, == (точное равенство) и [NOT] LIKE (сравнение по маске),

Заметьте, что, если для числовых типов и типа "дата" такие сравнения имеют обычный смысл, то для символьных строк имеется в виду лексикографическое (словарное) сравнение.

Предикат сравнения по маске имеет вид

Аргумент [NOT] LIKE Маска

где Маска - произвольная символьная строка, (возможно) содержащая специальные символы кратной замены % и одиночной замены _. Предикат [NOT] LIKE считается истинным, если маску можно превратить в стоящий

слева аргумент подстановкой некоторых слов вместо символа % и одиночных символов - вместо символов _.

б) **Булевские формулы,** как обычно, образуются с помощью операций конъюнкции AND, дизъюнкции OR и отрицания NOT

в) **"Синтаксический сахар"** - группа предикатов, выразимых через *булевские комбинации сравнений*, но более кратких и привычных по форме (особенно, для англоязычного пользователя)

Аргумент1 [NOT] BETWEEN Аргумент2 AND Аргумент3

- краткая форма записи предиката

[NOT] (Аргумент2 □ Аргумент1 AND Аргумент1□ Аргумент3)

Аргумент [NOT] IN (список_значений)

- краткая форма записи предиката

[NOT] (Аргумент=Аргумент1 OR Аргумент=□ Аргумент2 OR … Аргумент=АргументN)

г) **Предикаты, использующие выборку** образуют самый мощный - и самый сложный в освоении аппарат программирования SQL. Мы вернемся к ним в разделе «Вложенные подзапросы».

Имя_поля Знак_сравнения ALL (команда SELECT)

истинно, если сравнение выполняется для стоящего слева значения поля и *всех* значений, выбранных командой SELECT; при этом предполагается, что последняя выдает список значений - формально, таблицу с единственным

полем, совместимым по типу со значением поля, стоящим слева. Условие считается истинным также, если подвыборка пуста.

Имя_поля Знак_сравнения ANY | SOME (команда SELECT)

истинно, если сравнение выполняется для стоящего слева значения поля и хотя бы *одного* из значений, выбранных командой SELECT - снова предполагается, что последняя выдает список значений, совместимых по типу со значением поля. Если подвыборка пуста, условие считается ложным.

[NOT] EXISTS (команда_SELECT)

истинно, если подвыборка (не) пуста, т.е. (не) содержит по крайней мере одну строку.

Имя_поля [NOT] IN (команда_SELECT)

имеет тот же смысл, что и одноименный предикат IN, рассмотренный выше, с той лишь разницей, что список значений не задается явно, но является результатом подвыборки; истинен, если значение поля совпадает хотя бы с одним значением из (единственного поля) результата выборки.

§ 13. Создание и модификация структуры БД

Мы рекомендовали бы начать с более простого - изучения достаточно удобных визуальных средств модификации структуры БД. Тем не менее, далеко не все из них имеют средства "обратной" генерации SQL-кода, в то время как иметь "архивное" определение БД в символьном виде часто и удобно, и необходимо в целях безопасности.

Для опытных программистов заметим, что описание структуры БД Visual FoxPro содержит в файле с расширением .dbc, являющимся на деле .dbf-таблицей (что расширяет и без того достаточно богатые возможности программного определения и модификации структуры БД).

СОЗДАНИЕ ТАБЛИЦЫ

CREATE TABLE Имя_создаваемой_таблицы [NAME* Длинное - до 128 символов - имя_таблицы] [FREE*]

[(ОПИСАНИЕ ПОЛЯ 1 [ПРАВИЛА КОРРЕКТНОСТИ ПОЛЯ 1)],

[(ОПИСАНИЕ ПОЛЯ 2] [ПРАВИЛА КОРРЕКТНОСТИ ПОЛЯ 2])],

…..

[(ОПИСАНИЕ ПОЛЯ n] [ПРАВИЛА КОРРЕКТНОСТИ ПОЛЯ n])],

[ПРАВИЛА КОРРЕКТНОСТИ ЗАПИСИ]

(опция FREE - свободная таблица - используется в том случае, если программист не хочет включать создаваемую таблицу ни в одну из существующих БД; в противном случае, по умолчанию, таблица будет включена в БД, открытую на момент исполнения команды CREATE TABLE)

где

ОПИСАНИЕ ПОЛЯ -

Имя_поля

Тип_поля

[(Ширина_поля [,Точность (число знаков после запятой - для числовых типов])]

далее перечисляются допустимые в FoxPro скалярные типы, с указанием имени типа и допустимости для соответствующего типа, опций указания ширины поля (Ш) и точности (Т).

Тип	Ш	Т	Комментарий
C	n	-	Character - символьная строка некоторой длины n
D	-	-	Date - Дата
T	-	-	dateTime - Дата-время
N	n	d	Numeric - вещественное длины n , d знаков после запятой
F	n	d	Floating numeric - вещественное длины n, d знаков после запятой, в форме с плавающей точкой
I	-	-	Integer - целое
B	-	d	douBle - целое двойной точности
Y	-	-	currencY - денежная сумма
L	-	-	Logical - логический
M	-	-	Memo - строка неопределенной длины
G	-	-	General - ссылка на внешний объект

 Скалярные типы данных могут различаться как по синтаксису, так и по семантике в различных версиях SQL.

ПРАВИЛА КОРРЕКТНОСТИ ПОЛЯ - одна или несколько опций вида

[NULL | NOT NULL] - значением поля может (не может) быть неопределенное значение NULL; по умолчанию, значение опции определяется значением системной (SET-)переменной NULL; также по умолчанию, значение NULL не допустимо для первичных ключей и уникальных (UNIQUE) полей;

[CHECK Условие корректности поля [ERROR* Текст сообщения о нарушении условия]] - проверяется при каждой модификации, а также при добавлении "пустой" записи; сообщение об ошибке появляется лишь при работе в интерактивном режиме

[DEFAULT Значение поля по умолчанию]

[PRIMARY KEY | UNIQUE] - значение поля является (единственным) первичным ключом записи | должно быть уникальным для каждой записи таблицы; для поддержки соответствующего правила создается индекс с именем, совпадающим с именем поля:

[REFERENCES Имя_родительской_таблицы [TAG Имя (тег) индекса]] - значение поля является внешним ключом указанной родительской таблицы [по данному индексу]

ПРАВИЛА КОРРЕКТНОСТИ ЗАПИСИ - список из одной или несколько опций вида

[PRIMARY KEY Выражение TAG* Имя (тег) индекса |, UNIQUE Выражение TAG* Имя индекса] - указанное выражение определяет (единственный составной) первичный ключ записи | обязано быть уникальным для каждой записи; для поддержки соответствующего правила создается индекс с указанным именем (тегом);

[, FOREIGN KEY Выражение TAG* Имя_индекса
REFERENCES Имя_родительской таблицы [TAG* Имя индекса родительской таблицы]] - указанное выражение определяет (составной)

внешний ключ записи, поддерживаемым родительским индексом с указанным тегом

[, CHECK Условие корректности записи [ERROR* Текст сообщения о нарушении условия]]) - проверяется при каждой модификации, а также при добавлении "пустой" записи; сообщение об ошибке появляется лишь при работе в интерактивном режиме

УДАЛЕНИЕ ТАБЛИЦЫ

DROP TABLE Имя_таблицы | Имя_файла | ? [RECYCLE]

удаляет таблицу из текущей БД; опция ? выдает диалоговое окно выбора таблицы; подопция RECYCLE указывает на то, что удаленную таблицу нужно поместить в "корзину" ОС Windows с возможностью последующего восстановления, в противном случае восстановление невозможно.

УДАЛЕНИЕ ПРЕДСТАВЛЕНИЯ

DROP VIEW имя_представления

МОДИФИКАЦИЯ СТРУКТУРЫ ТАБЛИЦЫ

```
ALTER TABLE Имя_таблицы
        ADD | ALTER [COLUMN] Имя_поля
                Тип_поля [(Ширина_поля [, Точность])]
                [NULL | NOT NULL]
                [CHECK Правило_корректности_поля
                 [ERROR Текст_сообщения_о_нарушении_правила]]
                [DEFAULT Значение_по_умолчанию]
                [PRIMARY KEY | UNIQUE]
```

[REFERENCES Имя_родительской_таблицы

[TAG Тег(имя)_индекса]]

или

ALTER TABLE Имя_таблицы

ALTER [COLUMN] Имя_поля

[NULL | NOT NULL]

[SET DEFAULT Значение_по_умолчанию]

[SET CHECK Правило_корректности_поля

[ERROR Текст_сообщения_о_нарушении_правила]]

[DROP DEFAULT]

[DROP CHECK]

или

ALTER TABLE Имя_таблицы

[DROP [COLUMN] Имя_поля]

[SET CHECK Правило_корректности_поля

[ERROR Текст_сообщения_о_нарушении_правила]]

[DROP CHECK]

[ADD PRIMARY KEY Выражение_первичного_ключа

TAG Тег(имя)_индекса_первичного_ключа]

[DROP PRIMARY KEY]

[ADD UNIQUE Выражение [TAG Тег_индекса]]

[DROP UNIQUE TAG Тег_индекса]

[ADD FOREIGN KEY [Выражение_внешнего_ключа]

TAG Тег_индекса

REFERENCES Имя_родительской_таблицы

[TAG Тег_родительского_индекса]]

[DROP FOREIGN KEY TAG Тег_индекса]

[RENAME COLUMN Старое_имя_поля TO Новое_имя_поля]

Несмотря на устрашающий синтаксис, семантика команды ALTER TABLE легко выводится из пояснений к команде CREATE TABLE и значений английских слов ALTER - изменить, ADD - добавить, DROP - удалить, SET - положить равным, RENAME - переименовать.

Замечание. Пожалуй, единственным существенным упущением текущей версии FoxPro по сравнению с другими реализациями и стандартом SQL является отсутствие команды создания пользовательского индекса Create Index (и, соответственно команды удаления индекса Drop Index). Правда, это упущение – чисто синтаксическое, поскольку имеется достаточно мощная команда-аналог самого FoxPro – а именно, команда Index (см. документацию).

1. МОДИФИКАЦИЯ СОДЕРЖИМОГО ТАБЛИЦ

Добавление записи.

1. Добавление единичной записи.

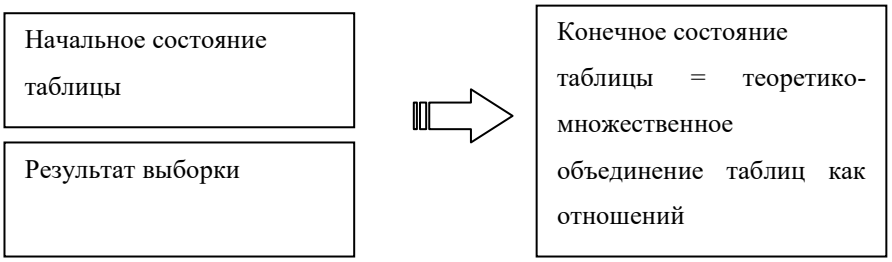
INSERT INTO имя_таблицы [(список_имен_полей])]
    VALUES (список_выражений)

В случае, когда список имен полей задан, он, очевидно, должен быть согласован по типу со списком выражений, задающим значения соответствующих полей в добавляемой записи; если же список имен полей отсутствует, то, по умолчанию, подразумевается список имен всех полей таблицы. В любом случае, необъявленные значения полей либо берутся равными значению по умолчанию (см. опцию DEFAULT в команде CREATE TABLE), либо - равными NULL.

2. Добавление результатов выборки.

INSERT INTO имя_таблицы (команда SELECT)

Подразумевается, что результатом команды SELECT является таблица, совпадающая по количеству и типу полей с заданной.



(a) Редактирование записей

UPDATE [Имя_базы_данных!]Имя_таблицы

SET Список выражений вида Имя_поля=Выражение

[WHERE Условие_обновления]

обновляет все указанные поля в заданной таблице значениями указанных выражений; при наличии опции WHERE, изменяются только записи, удовлетворяющие заданному условию.

(b) Логическое удаление записей

DELETE FROM [Имя_базы_данных!]Имя_таблицы

[WHERE Условие_удаления]

помечает записи, удовлетворяющие заданному условию, как удаленные.

Замечание. Как обычно в СУБД, логическое удаление записей *не означает* их фактического, т.е. физического удаления - производимого в FoxPro

командой упаковки таблицы PACK; до выполнения последней помеченные как удаленные записи могут быть восстановлены командой RECALL. Помеченные, но фактически не удаленные записи могут включаться или не включаться в последующую обработку в зависимости от значения системной SET-переменной DELETED.

§ 14. Запросы к БД – команда SELECT

SELECT
[ALL | DISTINCT]
[TOP числовое_выражение [PERCENT]]*
Список выборки

FROM Список_имен_базовых_таблиц

[[INTO Приемник_выборки]
| [TO FILE Имя_файла [ADDITIVE]
| TO PRINTER [PROMPT] | TO SCREEN]]*
[NOCONSOLE]*

[WHERE Условия_связи_и_условия_фильтрации]

[GROUP BY Список_столбцов_группировки]

[HAVING Условия_на_группу]

[UNION [ALL] команда SELECT]

[ORDER BY Список_столбцов_упорядочения]

где

ALL | DISTINCT - запрос на выборку всех (опция по умолчанию)| только различных значений списка выборки; последние считаются различными, если они различаются по значению хотя бы одной компоненты;

TOP числовое_выражение [PERCENT]* - запрос на выборку лишь нескольких первых различных строк выборки; количество различных строк (или процент от общего количества) задается значением числового выражения; используется только при задания порядка выборки - т.е. при использовании опции ORDER BY;

СПИСОК ВЫБОРКИ - список выражений вида

Компонента_выборки [AS Пользовательское имя поля]

Компонентой выборки может быть произвольное - в том числе константное - выражение над полями базовых таблиц; чаще всего это - просто имя поля базовой таблицы. Если базовые таблицы содержат одинаковые имена полей, во избежание коллизии (неоднозначности) имен необходимо использовать полные (квалифицированные) имена полей вида

[Имя или алиас таблицы.] Имя_поля

Имена полей в результирующей таблице (т.е. самой выборке) генерируются автоматически; опция AS позволяет давать им свои - более осмысленные имена.

Опция WHERE накладывает критерий, которому должны соответствовать выбираемые записи базовой таблицы (либо декартова произведения базовых таблиц, если их несколько).

Выборка из нескольких таблиц.

FROM Список_имен_базовых таблиц

список имен таблиц, "опрашиваемых" данной выборкой, вида

Имя_таблицы$_1$ [Локальный алиас$_1$],…, Имя_таблицы$_n$ [Локальный алиас$_n$]

Либо

Имя_таблицы [Локальный алиас]
[[INNER | LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER] JOIN *
Имя_таблицы [Локальный алиас]
[ON Условие_связи]

Первый вариант синтаксиса выборки из нескольких базовых таблиц означает (с точки зрения логики, но не реализации исполнения) выборку из их *именованного декартова произведения* - т.е. выборку всевозможных комбинаций строк базовых таблиц.

Такая полная выборка редко бывает необходимой, поэтому ее ограничивают наложением дополнительных условий (отношений, связей, объединения) на пары базовых таблиц. Ясно, что на практике чаще всего используются равенства первичного и внешнего ключей. В стандартном SQL связи реализуются с помощью опций WHERE.

Так, выборка

Select * from T1,T2 where B(r1,r2)

выбирает связанные отношением (заданным предикатом B(r1,r2)) записи r1, r2.

Расширенный синтаксис предлагает более непосредственное указание (и, по всей видимости, более оптимальную реализацию) такого произведения по условию – или, в терминах СУБД, соединения (JOIN) таблиц. Различают несколько вариантов такого соединения.

Внутреннее соединение (INNER JOIN) предполагает отбор из обеих таблиц только записей, удовлетворяющих соответствующему условию связи; кроме того, в *левое внешнее соединение* (LEFT [OUTER] JOIN) отбираются дополнительно все те строки из таблицы, стоящей слева от слова JOIN, в *правое внешнее соединение* (RIGHT [OUTER] JOIN) - все те строки из таблицы, стоящей справа от слова JOIN, и в *полное внешнее соединение* (FULL [OUTER] JOIN) - все те строки обеих таблиц, для которых соответствия в другой таблице не существует.

Более формально,

$T1 \text{ INNER JOIN } T2 \text{ ON } B(r1,r2) \approx \{r1+r2 : r2 \in T1 \,\&\, r2 \in T2 \,\&\, B(r1,r2)\}$

$T1 \text{ LEFT JOIN } T2 \text{ ON } B(r1,r2) \approx T1 \text{ INNER JOIN } T2 \text{ ON } B(r1,r2) \cup$

$\{r1+NULL(T2) : \text{not } \exists\, r2 \in T2\ B(r1,r2)\}$

здесь NULL(T2) – “пустая” или “фиктивная” запись таблицы T2, т.е. функция, принимающая значение NULL на всех именах полей из T2.

T1 RIGHT JOIN T2 ON B(r1,r2) ≈ T1 INNER JOIN T2 ON B(r1,r2) ∪

{NULL(T1)+r2 : not ∃ r1∈ T1 B(r1,r2)}

T1 FULL JOIN T2 ON B(r1,r2) ≈ T1 INNER JOIN T2 ON B(r1,r2) ∪

{NULL(T1)+r2 : not ∃ r1∈ T1 B(r1,r2)} ∪

{r1+NULL(T2) : not ∃ r2∈ T2 B(r1,r2)}

Разрешение коллизии имен. Квалифицированные имена и алиасы.

Синтаксис SQL не запрещает называть поля разных таблиц одним и тем же именем, что приводит к конфликту имен при формировании запроса из таких таблиц.

Квалифицированное (полное) имя - *Имя таблицы. Имя поля*

Замечание. Некоторые СУБД способны одновременно манипулировать с таблицами из разных БД – снова, возможно, одноименными, что приводит к необходимости дальнейшего уточнения имен, например Имя БД! Имя таблицы. Имя поля (синтаксис FoxPro)

Применение полных имен не спасает от коллизии в случае, когда команда SELECT ссылается на одну базовую таблицу несколько раз (например, при выборке из декартовой степени (двух копий) одной таблицы или – см. далее, в случае вложенных запросов, т.е. использования одной выборки в предикатах другой выборки). В таких случаях необходимо использовать псевдонимы/алиасы таблиц; последние задают временные имена (копий) таблицы, действительные только на момент выполнения данной выборки;

(а) Группировка и групповые вычисления. Опции Group By и Having

Синтаксис

GROUP BY List

где List – список (перечисление) полей группировки. При этом поля из списка могут быть заданы явно или числовой ссылкой на номер поля в списке выбора (в некоторых версиях также – выражением символьного типа) .

Список выборки при этом может состоять лишь из обращений к функциям, выдающим одно значение для каждой группы (групповым функциям) - по определению, таковыми являются значения полей группировки, а также следующие *агрегатные (т.е. групповые)* функции:

- COUNT(Выражение) - количество значений заданного выражения в группе, не равных NULL;
- COUNT(*) - число строк в группе;
- MIN(Выражение) - минимальное значение данного выражения по группе;
- MAX(Выражение) - максимальное значение данного выражения по группе;
- SUM(Числовое_выражение) - сумма значений заданного выражения по группе;
- AVG(Числовое_выражение), - среднее значение заданного выражения по группе;

Семантика –

1) производиться разбиение базовой таблицы (декартово произведение базовых

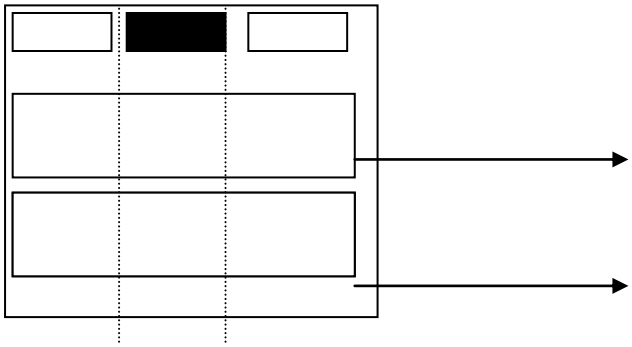
таблиц) на группы – классы эквивалентности, для которых поля из заданного списка принимают одинаковые значения.

R1,R2 принадлежат одной группе $g \approx R1 \approx_{list} R2 \approx R1(List)=R2(List)$

3) Для каждой группы вычисляются значения заданных списком выборки функций над группами

Select $F_1(List), \ldots, F_n(List)$ from T group by List =

$\{ <F_1(g), \ldots, F_n(g)>: g \in T/\approx list \}$



Примечание. Использование агрегатных функций возможно и без явного применения опции GROUP BY - в этом случае группой считается вся базовая таблица (декартово произведение базовых таблиц, если их несколько)

Опция

HAVING Условие_на_группу

по семантике схожа (и часто путается) с опцией WHERE, но, в отличие от последней, фильтрует не одиночные записи, а *группы.* Используется только совместно с GROUP BY. Соответственно, задаваемое условие на группу должно ссылаться только на групповые функции. Как правило, это условие

является проверкой значений некоторых агрегатных функций - для проверки значений полей списка группировки эффективнее использовать опцию WHERE.

Опция UNION - объединение выборок.

Обычно трактуется как опция команды SELECT, внося ненужную рекурсивность в определение этой команды. Более естественно трактовать как операцию над выборками – обычное теоретико-множественное объединение именованных отношений.

команда SELECT [UNION [ALL] команда SELECT

Объединяет результирующие таблицы двух или более выборок (совпадающих по структуре, т.е. количеству и типу компонентов выборки) в одну; в случае объединения более двух выборок необходимо расставить скобки.

Заметьте, что тип поля результирующих таблиц в FoxPro определяется по типу первого выбранного значения, что не всегда удобно, т.к. первое значение может оказаться короче последующих.

Довольно неожиданно, применение UNION по умолчанию убирает дубликаты из окончательной результирующей таблицы - используйте подопцию ALL (все), если такое положение не устраивает.

Отметим также, что

- UNION нельзя использовать в подзапросах;
- При использовании UNION разрешается использовать опцию упорядочения ORDER BY лишь последней объединяемой команды SELECT, со ссылкой на номер (не имя) столбца; в этом случае

упорядочение относится ко всей результирующей таблице-объединению.

Опция

ORDER BY список_полей_упорядочения [ASC|DESC]

как и в опции GROUP BY, список полей может состоять из имен полей, указанных явно либо выражением, значением которого является имя поля или его номер в списке выборки. В любом случае, подразумевается лексикографическое упорядочение по полям из заданного списка - либо по возрастанию (ASC - опция по умолчанию), либо по убыванию (DESC).

INTO Приемник_выборки| [TO FILE Имя_файла [ADDITIVE] | TO PRINTER [PROMPT] | TO SCREEN]]*

По умолчанию, результат выборки показывается в стандартном (BROWSE-) экранном окне. По желанию, можно указать другое место хранения результирующей таблицы · массив (опция INTO ARRAY имя_массива), временную таблицу "только для чтения", или - курсор (опция INTO CURSOR имя_курсора), таблица (т.е. DBF-файл - опция INTO DBF | TABLE Имя_таблицы [DATABASE Имя_БД [NAME Длинное_имя_таблицы]]), текстовый ASCII-файл (опция TO FILE имя_файла, подопция ADDITIVE указывается при дописывании в уже существующий файл), принтер (опция TO PRINTER, подопция PROMPT временно останавливает вывод для подготовки принтера), или главное экранное окно (опция TO SCREEN). Опция NOCONSOLE при этом подавляет "эхо"-вывод на экран.

Предикаты, использующие выборку. Вложенные подзапросы.

Предикаты – условия на запись, используемые в опции WHERE – как в запросах, так и командах модификации, в свою очередь, могут ссылаться на результаты запросов. Ниже следует перечисления таких предикатов.

1. r Знак_сравнения ALL (команда SELECT)

(здесь и далее r – некоторое выражение скалярного типа над полями записи некоторой таблицы; comp – знак сравнения)

Предикат истинен, если сравнение выполняется для стоящего слева значения поля и *всех* значений [SELECT], выбранных командой SELECT; при этом предполагается, что последняя выдает некоторый список значений List - формально, таблицу с единственным полем, совместимым по типу со значением выражения, стоящим слева. Условие считается истинным также, если подвыборка пуста.

r comp ALL (команда SELECT) = $\forall$ v$\in$ List (r comp v)

2. r comp ANY | SOME (команда SELECT)

истинно, если сравнение выполняется для стоящего слева значения поля и хотя бы *одного* из значений, выбранных командой SELECT - снова предполагается, что последняя выдает некоторый список значений, совместимых по типу со значением поля. Если подвыборка пуста, условие считается ложным.

r comp ANY | SOME (команда SELECT) = $\exists$ v$\in$ List (r comp v)

3. r [NOT] IN (команда_SELECT)

имеет тот же смысл, что и одноименный предикат IN, рассмотренный выше, с той лишь разницей, что список значений не задается явно, но является

результатом подвыборки; истинен, если значение поля совпадает хотя бы с одним значением из (единственного поля) результата выборки.

R IN [SELECT] = r ∈ [SELECT]

3. [NOT] EXISTS (команда_SELECT)

истинно, если подвыборка (не) пуста, т.е. (не) содержит по крайней мере одну строку.

EXISTS (SELECT) = Card([SELECT])>0 =∃ r (r∈ [SELECT])

Синтаксис SQL разрешает использовать во вложенном запросе ссылки на имена полей внешнего запроса. Такие вложенные запросы называют *соотнесенными*.

Пример. Выдать список имен богатых в своем городе покупателей (с кредитом выше среднего по городу, в котором они живут)

В какой степени к результатам выборки можно относиться как к реальным таблицам? Многие версии SQL предлагают опцию

INTO приемник выборки

позволяя тем самым сохранить результаты выборки в реальном файле-таблице – постоянной или временной (курсоре), либо в оперативной памяти (массиве и пр.), а также переслать их а экран монитора, принтер и т.п.

Представления дают способ сослаться на результаты сравнения, не прибегая к его сохранению.

СОЗДАНИЕ ПРЕДСТАВЛЕНИЯ

CREATE VIEW Имя_представления AS команда SELECT

В FoxPro допустимы

1) параметризованные представления, ссылающиеся на переменные FoxPro - при использовании параметров нужно поставить знак ? перед именем переменной.
2) модифицируемые представления; при этом суть определения модифицируемости состоит в следующем
   - каждая базовая таблица, в которой будет происходить реальная физическая модификация данных, обязана иметь первичный ключ;
   - этот первичный ключ обязан содержаться в модифицируемом представлении.

Более точное определение модифицируемости представлений можно найти в документации.

СОЗДАНИЕ КУРСОРА

Курсор (сокращение от "current set of records") – это временная таблица, с которой можно обращаться точно также, как с обычной таблицей БД, за одним, но важным исключением - в отличие от таблиц и представлений, курсор не считается частью БД; его содержимое и само определение действительно только в текущем сеансе работы и "теряется" немедленно по его закрытии.

CREATE CURSOR имя_курсора
(список_описания_полей_курсора)

где описание поля курсора имеет вид

(имя_поля тип_поля [(ширина_поля [, точностье])
[NULL | NOT NULL]
[CHECK Правило_корректности_поля
[ERROR Текст_сообщения_о_нарушении_правила]]
[DEFAULT значение_по_умолчанию]
[UNIQUE]

См. описание опций в команде CREATE TABLE.

Поскольку курсоры не считаются неотъемлемой частью SQL, мы снова отошлем к документации любознательного читателя, интересующегося деталями обработки курсоров в FoxPro.

§ 15. Транзакции

Введенное ранее понятие целостности БД - важнейший случай более общего понятия *безопасности данных,* подразумевающего включение в рассмотрение не только логической схемы самой БД, но и некоторую схему *всех аспектов ее использования* - от физического размещения и сохранности данных до определения прав пользователей - относимым к вопросам ведения или *администрирования* БД. Не касаясь здесь сколь-нибудь подробно этих вопросов, отметим лишь как существенно бóльшую безопасность данных, так и бóльшую сложность задач администрирования при использовании технологии "клиент-сервер"[7]. Одним из наиболее важных программных средств обеспечения безопасности данных является механизм поддержки транзакций.

Транзакция - логически единая операция по модификации базовых таблиц, состоящая из одной или нескольких команд СУБД, переводящая БД из

[7] Что, впрочем, касается в большей степени крупных серверных СУБД типа Oracle и MS SQL Server, а не таких СУБД клиентского типа, как рассматриваемая здесь Visual FoxPro.

одного целостного, корректного, с точки зрения предметной области, состояния в другое, также логически корректное.

Это понятие полностью совпадало бы с понятием процедуры, если бы ни одно обстоятельство, которое в данном случае нельзя игнорировать – недетерминизм, действия по модификации БД могут не выполниться, по не зависящим от программиста и программы внешним воздействиям. Состояние s команды/программы не полностью определяет результат ее выполнения F(s).

F(s,X)

X – скрытый фактор внешних воздействий, который может привести БД в некорректное состояние - F(s)□ F(s,X), для некоторых X.

В качестве такого скрытого фактора могут выступать самые разнообразные случайные факторы вроде сбоя электропитания, но даже при идеализации сетевой вычислительной среды нельзя игнорировать участия других пользователей, т.е. параллелизма действий различных модификаций.

Пример. Один пользователей удаляет запись, другой ее редактирует. Пользователь читает данные в момент выполнения другой команды.

Transaction(F,s,X)=if F(s)≠F(s,X) then F(s) [else E], ∀ X

Поддержка транзакций гарантирует, что входящие в транзакцию команды либо выполнятся полностью, либо – если последнее невозможно по какой-либо причине (сбой энергоснабжения, конфликт действий пользователей в многопользовательской/ сетевой БД и пр.) – полное восстановление состояния БД на момент начала транзакции (или иной выбранной программистом контрольной точки соответствующей процедуры).

"Утром деньги, вечером – стулья". Трансакция происходит от соответствующего финансового термина – банковская сделка по переводу денег. Элементарная сделка такого рода состоит из операция снятия денег с одного счета и записи соответствующей суммы на другой. Понятно, что без выполнения той или иной операции трансакцию нельзя считать успешной. Более сложный пример такого рода коллективный обмен – например, квартир.

Определяющие свойства трансакции определяеют как ACID – Atomicity, Consistency, Isolation, and Durability - атомность, непротиворечивость, изолированность и стойкость.

Под *атомностью* – «все или ничего», все вовлеченные в трансакцию изменения либо происходят целиком, либо не происходят вообще. Здесь "все или ничего" надо понимать скорее как "не навреди".

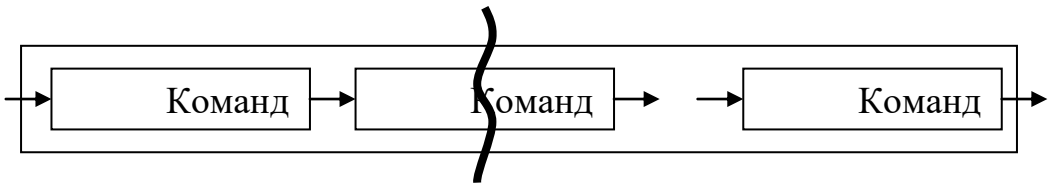
Непротиворечивость означает, что любые изменения необходимо приводят к корректному состоянию данных.

Изоляция – трансакции могут вкладываться, но не могут пересекаться (промежуточные результаты одной трансакции доступны другой, лишь если та полностью вложена в первую, т.е. составляет ее часть)

Стойкость означает, что подтвержденные результаты трансакции должны выдерживать все внешние воздействия (скажем, падение сервера)

Простые примеры трансакции – сами команды SQL. Это – автоматические, не объявленные явно трансакции.

Примером транзакции может служить также триггер каскадного удаления записи родительской таблицы, с последующим удалением записей дочерних таблиц. Все удаления должны быть либо осуществлены полностью, либо не осуществлены вообще – в противном случае в БД могут появиться записи-сироты.



Понятие трансакции отражает существо, и скрывает сложную реализацию двух основных механизмов поддержки многопользовательских БД – буферизации и блокировки.

Внутри транзакции модификация содержимого таблиц БД производиться над их копиями в буфере, затем производиться блокировка – запрет на действия по модификации оригиналов таблиц другими пользователями. Если блокировка удачна, т.е. соответствующие таблицы не заблокированы другим пользователем, содержимое буфера (результат трансакции) записывается в собственно таблицы БД, если нет – оригиналы остаются неизменными.

Начало транзакции оформляется командой SQL

BEGIN TRANSACTION

закачивается либо командой корректного завершения транзакции

END /COMMIT [TRANSACTION]

либо командой "отката", т.е. возврата к состоянию БД к моменту начала транзакции

ROLLBACK

FoxPro поддерживает до 5 уровней вложенности транзакций. Однако, поскольку другие пользователи сети не имеют доступа к записи таблиц БД,

модифицируемым во время исполнения транзакции, необходимо минимизировать время ее выполнения.

§ 16. Примерные типы заданий.

Пример простой БД «Учет заказов».

Некоторая торговая фирма просит вас автоматизировать учет заказов, поступающих от покупателя к обслуживающему его постоянно продавцу. При этом предполагается хранить следующую информацию.

Сведения о покупателях:

- УЧЕТный №;
- ФАМИЛИЯ;
- КРЕДИТ (сумма, на которую покупатель имеет право покупать товары у данной фирмы)
- ДАТА рождения
- ГОРОД проживания
- АДРЕС
- ТЕЛЕФОН
- ТАБЕЛЬный № обслуживающего продавца;

Сведения о продавцах:

- ТАБЕЛЬный №;
- ФАМИЛИЯ;
- КОМИССИЯ - число из интервала [0,1] - доля от суммы заказа, составляющая выручку продавца;

- ДАТА рождения
- ГОРОД проживания

Сведения о заказах:

- ПРОДАВЕЦ - табельный № продавца;
- ПОКУПАТЕЛЬ - учетный № покупателя;
- ТОВАР - артикул (идентификатор) товара
- ЗАКАЗАН (дата заказа);
- ПОСТАВЛЕН (дата поставки)
- СУММА заказа, в рублях;
- КОЛИЧЕСТВО поставляемого товара (в штуках или других подходящих единицах)

Упражнения.

1) Определите правила проверки корректности для полей и записей таблиц, делая разумные допущения о том, что такое:
- реальная дата заказа, если фирма основана 1.01.1990;
- реальная сумма заказа, если фирма продает автомобили ценой от 10000 до 100000 рублей, в текущих ценах;
- корректные сведения о заказе, если учитывать произошедшую 1.01.1998 деноминацию - смену масштаба цен.

2) Определите правила целостности в БД, предполагая, в частности, что:
- при увольнении продавца информация о его заказах сохраняется;
- при разрыве отношений с покупателем информация о его заказах далее не хранится

3) Определите структуру БД "Учет заказов" в вашей СУБД

4) Измените структуру БД, для того, чтобы отразить старшинство членов фирмы, полагая, что у каждого продавца, кроме главы фирмы, есть непосредственный начальник.

5) Дополните модель сведениями о продаваемых фирмой товарах – а именно, их

- АРТИКУЛ,
- НАИМЕНОВАНИЕ,
- код ТИПа товара (пример кодировки - 0101 - 'Автомобили', 0903 - 'Канцелярские товары' и пр.),
- коде предприятия-ИЗГОТОВИТЕЛЯ,
- коде ЕДИНИЦЫ измерения (метры, кв. метры, килограммы, штуки),
- ЦЕНЕ за единицу
- и имеющемся на складе КОЛИЧЕСТВЕ.

Как изменится структура БД, если допустить, что

- Каждый покупатель всегда покупает только один товар?
- Каждый продавец продает постоянно лишь один товар?
- В общем (и подразумеваемом далее) случае неверно ни то, ни другое?

ЗАПРОСЫ К БАЗЕ ДАННЫХ.

Введем несколько определений, полезных для компактной формулировки задач. Каждая строка таблицы БД описывает некоторый **объект** предметной области (или отношение между объектами, которое снова удобно считать объектом). **Атрибутами** объекта назовем значения одного или нескольких полей соответствующей ему таблицы. Свойством или **категорией** объектов назовем некоторое естественно определяемое, в терминах предметной

области, множество объектов. Категории объектов разделим на простые и сложные, в зависимости от того, содержится ли необходимая для определения категории информация в одной или нескольких таблицах БД. Примеры атрибутов и категорий для объектов БД «Учет заказов» приведены ниже.

Фильтрация таблиц. Определите атрибуты объектов по их свойствам, делая разумные допущения о предметной области и возможных значениях переменных.

Определите

(атрибуты покупателей:)

- учетные номера
- фамилии, телефоны и адреса
- даты рождения, по возрастанию
- возраст, по убыванию
- города проживания, в алфавитном порядке
- табельные номера обслуживающих продавцов (варианты: покупателей, простые категории покупателей:)

- живущих в данном городе (например, Казани)
- живущих в данных городах (например - Казани, Самаре или Москве)
- (не) совершеннолетних, старых, молодых
- (не) кредитоспособных

Определите
(атрибуты продавцов:)

- табельные номера

- фамилии и комиссионные, в процентах
- даты рождения, по убыванию
- возраст, по возрастанию
- города проживания, в алфавитном порядке

продавцов,

(простые категории продавцов:)

- живущих в данном городе (например, Казани)
- живущих в данных городах (например - Казани, Самаре или Москве)
- низко-, средне- высокооплачиваемых
- старых, молодых

Определите

(атрибуты заказов:)

- номера
- суммы
- артикулы товаров
- табельные номера продавцов
- учетные номера покупателей

заказов

(простые категории заказов:)

- на трехзначную сумму, по возрастанию суммы

- сделанных сегодня (вчера, на прошлой неделе, этим летом, в прошлом году), по возрастанию суммы заказа
- (не) выгодных для фирмы
- просроченных, по возрастанию даты заказа
- мелких, средних, крупных

Определите

(атрибуты товаров:)

- артикулы
- цену и названия, по возрастанию цены

товаров

(простые категории товаров:)

- дорогих (дешевых)
- имеющихся на складе в (не)достаточном количестве, отсутствующих на складе

Выборка из нескольких таблиц.

Фильтрация декартового произведения. Определите атрибуты двух и более объектов, связанных следующими отношениями:

- (не) этот продавец (не) обслуживает (не) этого покупателя
- (не) этот продавец (не) обслуживает (не) этот заказ
- (не) этот покупатель (не) сделал (не) этот заказ
- этот заказ - на (не) этот товар

- (не) этот покупатель (не) сделал (не этот) заказ (не) этому продавцу
- (не этот) покупатель (не) сделал (не этот) заказ (не этому) продавцу на (не) этот товар
- эти продавец и покупатель - однофамильцы
- эти продавец и покупатель живут в одном городе
- этот продавец старше (младше, ровесник) этого покупателя

Рефлексивные связи. Определите атрибуты двух и более объектов, связанных следующими отношениями

- эти покупатели - однофамильцы
- эти покупатели живут в одном городе
- этот покупатель старше (младше, ровесник) другого покупателя

- эти продавцы - однофамильцы
- эти продавцы живут в одном городе
- этот продавцы старше (младше, ровесник) другого продавца

- этот заказ - на более крупную сумму, чем другой
- этот заказ сделан раньше, чем другой
- эти заказы обслуживает один продавец
- эти заказы - одного покупателя
- эти заказы - на один товар

- это товар - дороже (дешевле, стоит столько же), чем другой
- эти товары (не) произведены на одном предприятии

Декодирование.

Определите атрибуты объектов по свойствам других (одного или нескольких) связанных с ними объектов.

Варианты задания

Определите фамилии продавцов,

- обслуживавших дорогие заказы
- обслуживавших кредитоспособных покупателей

Определите цену и наименования товаров

- для дорогих заказов
- для заказов, обсуживавшихся высокооплачиваемыми продавцами
- для заказов, обсуживавших молодых покупателей

Группировка и групповые вычисления.

Определите значения групповых функций для заданного определения группы

Подсчитайте

- средний, максимальный, минимальный возраст (дата рождения, сумма кредита) покупателя и количество покупателей

- по всем покупателям
- по всем категориям покупателям, перечисленных выше
- для каждого города

Подсчитайте

- средний, максимальный, минимальный процент комиссионных (возраст, дата рождения) продавца и количество продавцов

- по всем продавцам
- по всем категориям продавцов, перечисленных выше
- для каждого города

Подчитайте

- среднюю, максимальную, минимальную, совокупную сумму заказов и количество заказов

- по всем заказам
- по всем категориям заказов, перечисленных выше
- данного покупателя
- данного продавца
- для каждого покупателя
- для каждого продавца
- для каждой пары покупатель-продавец
-

Подчитайте

- минимальную, максимальную, среднюю цену товара
- минимальное, максимальное, среднее, совокупное количество единиц товара, хранящихся на складе
- количество единиц товара, по каждому типу товара

- по всем товарам
- по всем категориям товаров, перечисленным выше

Фильтрация групп. Подсчитайте значения групповых функций, выводя лишь значения, попадающие в заданный интервал.

Вариант задания. Определить количество товаров, которых мало на складе (т.е. это количество меньше некоторого заданного числа - скажем, 100)

Предикаты, использующие выборку. Вложенные запросы

Сравнение с выборкой. Определите атрибуты объектов, для которых значение некоторого поля больше (меньше, равно) среднего значения этого (или другого) поля этой (или другой) таблицы по группе.

Варианты задания.

Выведите атрибуты сравнительно молодых покупателей - т.е. покупателей, возраст которых меньше среднего значения по всем покупателям.

Выведите атрибуты сравнительно дорогих товаров, по каждому типу товаров.

Отношение принадлежности. Определите атрибуты объектов таких, что связанные с ними объекты принадлежат данной категории.

Варианты задания.

Определите атрибуты покупателей, обслуживающихся казанскими продавцами.

Определите атрибуты товаров, проданных за последний месяц молодыми продавцами.

Определите атрибуты продавцов, продавших хотя бы одно наименование из товаров, проданных данным продавцом.

Кванторные предикаты. Определите атрибуты объектов, для которых категория связанных с ними объектов (не)пуста.

Определите атрибуты продавцов, (не) обслуживающих хотя бы одного казанского покупателя.

Определите атрибуты продавцов, (не) продавших за последний месяц дорогие товары.

(Связанные запросы).

(Внешние) объединения. Определите совпадающие атрибуты объектов из разных таблиц.

Вариант задания. Определите фамилии и возраст покупателей и продавцов, проживающих в данном городе (например, Казани).

МОДИФИКАЦИЯ ТАБЛИЦ

Удалите из таблиц заданные категории объектов.

Вариант задания.

Удалите все сведения о товарах, не пользовавшихся спросом в последнее время (скажем, год).

Удалите сведения о продавцах, не осуществлявших никаких продаж за последнее время.

Удалите сведения о покупателях, не делавших покупок за последнее время.

Измените значения атрибутов объектов заданной категории.

Чуть более сложные задачи-многоходовки. Представления и транзакции.

В контексте заданной предметной области, любой человек характеризуется следующими данными.

- NAMe ФАМИЛИЯ (30 символов)
- ID ПАСПОРТ - № паспорта, 10 символов (очевидно, уникальный для каждого человека)
- MUM МАТЬ - № паспорта матери (NULL, если нет сведений)
- DAD ОТЕЦ - № паспорта отца (NULL, если нет сведений)
- СУПРУГ - № паспорта жены/мужа (NULL, если нет сведений)
- BIRTDAY ДАТА - дата рождения (тип дата)
- INCOME ДОХОД - ежемесячный доход (зарплата, стипендия и т.п.), вещественное 99.999.999,99
- CITY ГОРОД - город проживания (30 символов)
- TAX НАЛОГ - сумма подоходного налога, в % (целое)

Кроме того, в информацию о студентах дополнительно включаются сведения

- INSTITUTE ВУЗ - сокращенное название ВУЗа, 10 символов - например, ‘КГУ’, ‘КГМУ’ и т.п.
- SEX ПОЛ - один из символов ‘М’,’Ж’

БД включает таблицы СТУДЕНТЫ, ОТЦЫ и МАТЕРИ, включающую информацию о студентах и их родителях, соответственно.

ПРИМЕЧАНИЕ. Считаем, что допустимо применение имен на кириллице.

- Выдать список студентов, с указанием фамилий и дохода родителей, упорядоченный по возрасту студентов. Студентов, чьи доходы ниже 100 руб., в список не включать.
- Выдать упорядоченный по фамилиям список живущих в Казани, Москве и Самаре родителей, с указанием - мать это или отец, чей доход не превышает 300 руб.
- Выдать все пары студентов-однофамильцев из КГУ, с указанием фамилии и, для обоих, паспортных данных. Дублирующиеся пары не включать.
- Выбрать всю информацию об отцах, чьи доходы выше среднего по городу, в котором они живут.
- Выдать список живущих вне Казани матерей, имеющих не менее - двух сыновей-студентов, учащихся в Казани.
- Выбрать всю информацию об студентах, чьи доходы выше среднего для казанских студентов.

"Многоходовки" – т.е. задачи, решение которых не реализуется легко единственной командой SQL проще решать, используя представления или курсоры, например:

- Удалить информацию о студентах ВУЗА 'АГУ' а также их родителях - в случае, если у тех нет других детей-студентов (вариант посложнее - включить в число родителей студентов их бабушек и дедушек)

Вариант решения:

- запомнить студентов нужного ВУЗа и ссылки на их отцов,
- запомнить отцов, у которых все дети (если вообще есть таковые) учатся в нужном ВУЗе
- удалить информацию об отцах, входящих в первый и второй список
- … сделать то же с мамами

- удалить информацию о студентах нужного ВУЗа

1. **Create View Dad1 as Select Distinct Dad from Students where Institute=’АГУ’**
2. **Create View as Dad2 as Select Id from Dad where NOT Exists (Select Id from Students where Dad=Dad.Id and Institute<>’АГУ’)**
3. **Delete from Dad where Id in (Select Id from Dad1) and Id in (Select Id from Dad2)**
4. **…**
5. **Delete from Students where Institute=’АГУ’**

6. **Замечание. Стандарт ANSI SQL запрещает в командах модификации подзапрос – в том числе, неявный (через представление) - к модифицируемым таблицам. Если реализация следует в этом стандарту – прибегнуть к курсорам - Select Distinct Dad from Students where Institute=’АГУ’ into cursor Dad1 etc.**

- Снизить на 5% налог родителям, имеющим более 3 детей-студентов, обучающихся в Казани

Вариант решения:

- Разбив казанских студентов на группы детей одних родителей, отобрать и запомнить ссылки на тех отца и мать, для которых количество их детей больше 3
- Изменить записи о матерях, ссылки на которые были запомнены на первом шаге,
- Изменить записи об отцах, ссылки на которые были запомнены на первом шаге.

1. CreateView Parents as Select Mum, Dad,Count(*) from Students Group BY Mum, Dad Having Count(*)>3
2. Update Mum set Tax= Tax-5 where Id in (Select Mum from Parents)
3. Update Dad set Tax= Tax-5 where Id in (Select Dad from Parents)
4.

- Повысить на 50 руб. стипендию (т.е. доход) студентов, у которых нет хотя бы одного из родителей либо совокупный доход родителей не превышает 1000 руб.

Вариант решения:

- Подсчитать и запомнить совокупный доход всех родителей (вместе со ссылками на отца и мать), если он не более заданной суммы,
- Изменить соответственно информацию о студентах, для которых ссылки на родителей либо пусты, либо попали в запомненный список

1. Create View LowIncome as Student.Id as Id, Dad, Mum, Dad.Income+Mum.Income from Mums,Dads,Students where Student.Mum=Mum.Id and Student.Dad=Dad.Id and Dad.Income+Mum.Income<=1000
2. Update Students set Income=Income+50 where IsNull(Mum) or IsNull(Dad) or Id in (Select Id from LowIncome)

- Повысить на 5% налог отцам, у которых доход жены не ниже среднего для матерей, а совокупный доход детей превышает 1000 руб.

Вариант решения:

- Сгруппировав по отцам, подсчитать совокупный доход студентов по группе, запомнить ссылки на отцов, если подсчитанный доход больше заданной суммы.
- Запомнить ссылки на матерей, чей доход больше среднего по матерям

- Изменить соответственно информацию об отцах, попавших в первый список, у которых жена попала во второй список

Create View HignIncomeDads as Select Sum(Income),Dad from Students GroupBy Dad Having Sum(Income) >1000

Create View HignIncomeMum as Select Id from Mum where Income>(Select AVG(Income) from Mum)

Create View HignIncomeParents as Select Distinct Dad,Mum from Students where Dad in (Select Dad from HighIncomeDad) and Mum in (Select Id from HighIncomeMum)

Update Dads set Tax=Tax+5 where Id in (Select Distinct Dad from HignIncomeParents)

§ 17. Проектирование "реальной" БД.

Информационная система "Сборочное предприятие".

НЕФОРМАЛЬНАЯ МОДЕЛЬ

После тщательного изучения проблемной области и опроса экспертов - специалистов предприятия, вы обладаете следующей информацией о его структуре и функционировании.

Предприятие состоит из нескольких **подразделений** - цехов и участков**,** в которых трудятся **работники** различных **профессий.** Работники осуществляют, в соответствии с производственным планом предприятия, сборочные и иные **операции** по изготовлению готовых **изделий** из **компонент.** Компонентами могут служить **изделия** собственного производства и исходные **материалы** (сырье). Все операции осуществляются в согласии с установленными **нормам**

затрат труда и расхода материалов. Оплата труда работников производиться согласно их **выработке** и установленным **тарифам**. Исходные материалы **поступают** на **склады** предприятия от **поставщиков** в соответствии с заключенными с ними **договорам**. Готовые изделия **отгружают** со склада **покупателям** согласно их **заказам**.

Упражнение. Попробуйте сами выделить базовые понятия, их свойства и взаимосвязи на основе неформального и неполного описания. Разумеется, такое выделение далеко не однозначно. Обоснуйте (защитите) свой вариант, сравнив его с предложенным ниже.

ВЗАИМОСВЯЗЬ ПОНЯТИЙ

Дополнительно, каждое из выделенных базовых понятий и процессов и их взаимосвязи более детально характеризуется в соответствующих справочниках и учетных книгах предприятия следующим образом:

- **Одно ПОДРАЗДЕЛЕНИЕ может включать в себя другие ПОДРАЗДЕЛЕНИЯ (цеха состоят из участков)**
- **Каждый РАБОТНИК обладает некоторой ПРОФЕССИЕЙ**
- **У каждого РАБОТНИКа (кроме директора) имеется непосредственный начальник (другой РАБОТНИК)**
- **МАТЕРИАЛЫ участвуют в ОПЕРАЦИЯХ в качестве компонент**
- **ИЗДЕЛИЯ также участвуют в ОПЕРАЦИЯХ в качестве компонент**
- **ИЗДЕЛИЯ** являются также результатом **ОПЕРАЦИЙ**
- **ОПЕРАЦИИ** производятся некоторым **УЧАСТКОМ (их исполнителем)**
- **РАСХОД всегда связан с некоторым МАТЕРИАЛОМ**
- **РАСХОД материала осуществляется при выполнении некоторой ОПЕРАЦИИ**

- **РАСХОД материала осуществляется некоторым участком - исполнителем операций**
- **МАТЕРИАЛЫ** поставляются по **ДОГОВОРУ**
- **ДОГОВОР** заключается с **ПОСТАВЩИКОМ**
- **ПОСТАВКА осуществляется ПОСТАВЩИКОМ**
- **ПОСТАВКА осуществляется на СКЛАД**
-

МАТЕРИАЛЫ (исходные материалы и детали производства)

- уникальный **код** материалы;
- **наименование**;
- **характеристика**;
- **единица** измерения
- **цена** за единицу.

ИЗДЕЛИЯ (готовые изделия и сборочные единицы собственного производства)

- уникальный **код** изделия;
- **наименование**;
- **характеристика**;
- **цена**

СБОРКА (пооперационный процесс изготовления изделий и сборочных единиц из более простых компонент)

- код **компоненты**

- код сборочной единицы или готового **изделия**, в которую входит данная компонента;

- номер **операции** процесса сборки;

- используемое **количество** на операцию;

- номер **участка** исполнителя;

РАСХОД (нормы расхода материалов)

- код сборочной единицы или готового **изделия**;

- код **компоненты** (используемой при изготовлении детали или в монтажных работах при сборке изделия);

- номер **операции**;

- номер **цеха** (исполнителя работ);

- номер **участка**;

- **единица** измерения;

- норма **расхода** материала.

ЗАТРАТЫ (нормы затрат труда)

- код **изделия** ;

- номер **операции**;

- номер **цеха** (исполнителя работ);

- номер **участка**;

- код **профессии** рабочего;

- код условий труда и **тариф**ный разряд работы;

- **дополнение** - время на подготовку и заключение работы, в мин.;

- **время** штучное, в мин.

ТАРИФЫ

- уникальный **код** условий труда и тарифный разряд работы;
- часовая тарифная **ставка**, в рублях

ПРОФЕССИИ

- уникальный **код** профессии;
- **наименование** профессии.

ПОДРАЗДЕЛЕНИЯ

- **номер** цеха;
- **наименование** цеха или участка.

ПЛАН (производственный план предприятия)

- код **изделия**;
- распределение плана по всем месяцам - выпуск , в штуках, в **январе, феврале** и т.д.;
- дата **начала** действия плана.

ДОГОВОРЫ (на поставку компонент)

- код **поставщика**;
- уникальный **номер** договора;
- код **компонента** - материала или покупной детали;
- **единица** измерения;

- план **поставки** на год, в шт.;

- распределение плана по всем месяцам - поставки в **январе, феврале** и т.д.;;

- дата **начала** действия договора.

РАБОТНИКИ (личный состав, штат предприятия)

- номер **цеха**;
- табельный **номер** рабочего;
- код **профессии**;
- р**азряд** рабочего;
- часовая тарифная **ставка**;
- семейное поло**жение**;
- **фамилия** с инициалами.

ВЫРАБОТКА (учет выработки работников)

- номер **цеха**;
- номер учас**тка**;
- код **изделия**;
- номер **операции**;
- табельный номер **работника**;
- **количество** годных деталей;
- количество **брак**ованных деталей;
- **процент** оплаты брака;
- **дата** выполнения работ.

ПОСТАВЩИКИ

- уникальный **код** поставщика;
- **наименование** поставщика;
- **адрес** поставщика.

ПОСТАВКА компонент

- номер **склада**;
- код **поставщика**;
- код **компоненты**;
- **единица** измерения;
- **количество** ;
- **дата** поступления;
- уникальный **номер** документа.

ОТГРУЗКА готовой продукции

- номер **склада**;
- код поку**пателя**;
- код готового **изделия**;
- **единица** измерения;
- **количество**;
- **дата** отгрузки;
- уникальный **номер** документа.

ПОКУПАТЕЛИ

- уникальный **код** покупателя;
- **наименование**;
- **город;**

- почтовый **адрес**.

СКЛАДЫ

- **номер** склада;
- **фамилия** материально ответственного лица;
- код **детали** - компоненты или изделия;
- **единица** измерения;
- **количество**, имеющееся на складе;
- **дата** последней операции.

ЗАРПЛАТА (бухгалтерский учет начисления и удержания по зарплате)

- табельный номер **работника**;
- сумма **начисления**;
- сумма **удержания**;
- **дата** выдачи.

ЗАКАЗЫ (договоры на отгрузку готовой продукции покупателям)

- код **покупателя**;
- уникальный **номер** заказа;
- код **изделия**;
- **единица** измерения;

- **план** поставки на год, в шт.;

- распределение плана по всем месяцам - поставки в **январе, феврале** и т.д.;

- дата **начала** действия договора.

Упражнение. Постройте формальную модель предприятия в форме базы данных, определив таблицы КОМПОНЕНТЫ, ИЗДЕЛИЯ и т.д., используя выделенные слова (**код**, **наименование**, **характеристика**, **единица**, **цена** и т.д.) в качестве имен полей (здесь и далее мы для удобства используем кириллические имена произвольной длины; если ваша СУБД не поддерживает соответствующие идентификаторы, используйте латинскую транскрипцию и сокращения). Выясните, какие поля (или группа полей) являются первичными и внешними ключами. Правила целостности и корректности значений полей и записей таблиц (в частности, допустимость неопределенных значений) определите самостоятельно, исходя из содержательного смысла таблиц и отношений.

Упражнение. Проведите в компьютерном классе деловую игру, распределив роли директора предприятия, бухгалтера, мастера, поставщика, заказчика и т.п. по предлагаемому или - выдуманному самостоятельно "сценарию":

Заказчик - директору: "По нашему договору №…, от …. ваше предприятие недопоставило … изделий "…". Если Вы не поставите требуемые изделия в течении … дней, мы обратимся в суд"

Директор - заказчику "Минутку-минутку, сейчас уточним…У меня почему-то стоит другая дата…"

Директор - кладовщику "Сколько у нас на складе изделий "…"? Не хватает?"

Директор - начальнику цеха "Сможем в течении … дней собрать недостающие .. штук изделий "…"?

Далее следуют обращения

- начальника цеха на склад в поиске нужных компонент,
- начальника цеха - к директору с просьбой повысить тарифные расценки за срочную работу,
- директора - в бухгалтерию, с вопросом о финансовых возможностях предприятия выполнить эту просьбу,
- директора к поставщикам с просьбой срочно поставить недостающие компоненты,
- и т.д. - импровизируйте!

Разумеется, ваша игра должна быть достаточно деловой, т.е. серъезной - все обращения должны подкрепляться конкретными цифрами, датами, наименованиями, ссылками и т.д. - проще говоря, реальной работой по формированию запросов к БД и ее модификации.

ЛИТЕРАТУРА.

1) М.Нагао, Т.Катаяма, С.Уэмура. Структуры и базы данных – М.,Мир, 1986 – 196 с.
2) М.Грабер. Введение в SQL
3) А.Горев. Visual FoxPro 5.0. Книга для программистов – М., ТОО «Эдэль», 1997 – 552 с.
4) М.Антонович, Visual FoxPro для Windows, BINOM Publishers, 1996 – 688 с.
5) М.Базиян. Использование Visual FoxPro 6. Вильямс, 2000 - 925 с.
6) З.Пэддок, Дж.Петерсен, Р.Тэлмейдж. Visual FoxPro 6. Разработка корпоративных приложений. Изд-во ДМК 2000, 588 с.

ПРИЛОЖЕНИЕ. Пример проектирования базы данных[8].

Проектирование базы данных подразумевает собой процесс создания проекта базы данных, предназначенной для поддержки функционирования предприятия и способствующей достижению его целей. Основными целями проектирования базы данных являются:

1) представление данных и связей между ними, необходимых для всех основных областей применения данного приложения и любых существующих групп его пользователей.
2) создание модели данных, способной поддерживать выполнение любых требуемых транзакций обработки данных

[8] выполнено, под руководством автора, студенткой А.Юсуповой в качестве одного из разделов дипломной работы

3) разработка предварительного варианта проекта, структура которого позволяет удовлетворить все основные требования, предъявляемые к производительности системы.

Существует два основных подхода к проектированию систем баз данных: «нисходящий» и «восходящий». При восходящем подходе работа начинается с самого нижнего уровня- уровня определения атрибутов, т.е. свойств сущностей, которые на основе анализа существующих между ними связей группируются в отношения, представляющие типы сущностей и связи между ними. Более подходящей стратегий проектирования сложных баз данных является использование нисходящего подхода. Начинается этот подход с разработки моделей данных, которые содержат несколько высокоуровневых сущностей и связей, затем работа продолжается в виде серии нисходящих уточнений низкоуровневых сущностей, связей и относящихся к ним атрибутов.

Нисходящий подход демонстрируется в концепции «сущность-связь» В этом случае работа начинается с идентификации сущностей и связей между ними, интересующих данную организацию в наибольшей степени. Например, сначала можно было бы идентифицировать сущности PSTV (поставщик) и KOMP (материалы), а затем установить между ними связь «поставляет» и лишь после этого определить связанные с ними атрибуты - например, PSTV (k_pstv, pstv, gorod, adress) и KOMP (k_komp, komp, ed_izm, khar, zena_ed).

Существуют три основные фазы процесса проектирования базы данных: концептуальный, логический и физический.

Первая фаза процесса проектирования базы данных называется **концептуальным проектированием.** Она заключается в создании концептуальной модели данных для анализируемой части предприятия. Эта модель создается на основе информации, записанной в спецификации требований пользователей. Концептуальное проектирование абсолютно не

зависит от таких подробностей ее реализации, как тип выбранной целевой СУБД, набор создаваемых прикладных программ, используемые языки программирования. Созданная концептуальная модель данных предприятия является источником информации для фазы логического проектирования БД.

Вторая фаза проектирования БД называется **логическим проектированием БД.** Ее цель состоит в создании логической модели данных для исследуемой части предприятия. Концептуальная модель данных, созданная на предыдущем этапе, уточняется и преобразуется в логическую модель данных. Логическая модель данных учитывает особенности выбранной модели организации данных в целевой СУБД (например, реляционная или сетевая). Если концептуальная модель данных не зависит от любых физических аспектов реализации, то логическая модель данных создается на основе выбранной модели организации данных целевой СУБД. В процессе разработки логическая модель постоянно тестируется и проверяется на соответствие требованиям пользователей. Для проверки корректности логической модели данных используется метод нормализации. Нормализация гарантирует, что выведенные из соответствующей модели данных отношения не обладают избыточностью данных, способной вызвать аномалии обновления после их физической реализации.

Слияние представлений отдельных пользователей

Логическая модель, отражающая особенности представления о функционировании предприятия одновременно многих типов пользователей, называется глобальной логической моделью данных предприятия. Существует два основных подхода к созданию глобальной логической модели, это - централизованный подход и подход на основе интеграции представлений. Чаще используется второй подход, при котором осуществляется слияние отдельных моделей данных, отражающих представления разных групп пользователей и

называемых локальными логическими моделями данных, в единую глобальную логическую модель всего предприятия.

Физическое проектирование является третьей фазой процесса создания проекта базы данных. Основной целью физического проектирования БД является описание способа физической реализации логического проекта БД. В случае реляционной модели данных под этим подразумевается следующее:

- создание набора реляционных таблиц и ограничений для них на основе информации , представленной в глобальной логической модели данных;

-определение конкретных структур хранения данных и методов доступа к ним, обеспечивающих оптимальную производительность системы с базой данных;

-разработка средств защиты создаваемой системы.

Пример разработки концептуального, логического и физического проекта базы данных «сборочное предприятие».

Выполнение фазы сбора и анализа требований пользователей, являющейся первой в цикле разработки приложений баз данных, осуществлялось на предприятии «Двигатель». Был проведен опрос сотрудников, работающих на должностях начальника цеха, начальника склада, бухгалтера, менеджера по поставкам и по реализации , кадровик, а также проанализирована вся документация, используемая или создаваемая ими при выполнении своих служебных обязанностей. Результатом выполнения этой фазы разработки проекта явилась подготовка спецификаций требований для пользователей начальник цеха, начальник склада, бухгалтер, менеджер по поставкам и по реализации, кадровик. В этих спецификациях зафиксированы требования к информации, которая будет помещена в создаваемую базу данных, а также определены все транзакции, необходимые для пользователей,

находящихся на вышеупомянутых должностях, для выполнения их служебных обязанностей.

Спецификация требований для представления пользователя «Начальник склада »

Требования к данным.

1. Предприятие имеет два склада, на один из которых поставляются материалы и детали, а на другой - готовые изделия.
2. На каждом из складов есть материально- ответственное лицо и грузчики.
3. Каждая поставляемая деталь характеризуется уникальным номером, наименованием, характеристикой, ценой за единицу продукции.
4. Каждая изготавливаемая деталь и сам двигатель также характеризуются уникальным номером, названием, характеристикой, ценой за единицу.
5. Поставки на склад осуществляются поставщиками. Информация о поставщиках включает в себя код поставщика, название и адрес.
6. Со склада изделия отгружаются заказчикам (покупателям). В сведения о покупателях входят: код, название, адрес.
7. В обязанности начальника склада входит введение документации о поставках и отгрузках.

Требования к транзакциям.

К основным транзакциям, которые должны выполняться пользователем «Начальник склада», относятся следующие:

a) составление документа об отгрузке изделий

b) составление документа о поставке материалов и деталей

c) поиск изделий и деталей, удовлетворяющих различным требованиям.

Построение локальной концептуальной модели данных для представления пользователя «начальник склада».

Приступая к разработке локальной концептуальной модели данных для представления пользователя «начальник склада» в приложении «сборочное предприятие», прежде всего, следует выявить различные компоненты этой модели, используя имеющиеся спецификации требований пользователя. В каждую создаваемую модель данных входят следующие компоненты:

- типы сущностей
- типы связей
- атрибуты
- первичные ключи

Определение типов сущностей

Начнем работу с того, что определим основные типы сущностей исходя из имеющихся спецификаций. В спецификациях сущности обычно представлены как существительные. Анализ показывает, что основными сущностями, упоминаемыми в спецификациях, являются следующие:

СКЛАД	**SKLAD**
ИЗДЕЛИЕ	**IZDEL**
МАТЕРИАЛЫ И ДЕТАЛИ	**KOMP**
ПОСТАВЩИКИ	**PSTV**
ЗАКАЗЧИКИ	**POKUP**
ДОКУМЕНТ О ПОСТАВКЕ	**POST**
ДОКЕМЕНТ ОБ ОТГРУЗКЕ	**OTGR**

Определение типов связей

Следующий шаг состоит в определении типов связей, существующих между отдельными сущностями. Как правило, связи выражаются глаголами или глагольными сочетаниями.

тип сущности	тип связи	тип сущности
POST	осуществляется на	**SKLAD**
POST	осуществляется	**PSTV**
OTGR	делают на	**SKLAD**
OTGR	проводит	**POKUP**
IZDEL	отгружаются по	**OTGR**
KOMP	поставляются по	**POST**

Определение кардинальности и уровня участи отдельных типов связей

Следующий этап - определение кардинальности и уровня участия для каждого типа связей. Кардинальность любой связи может иметь значение либо **1:1**, либо **1:M**, либо **M:N.** Участие каждого из членов связи может быть определено как частичное или тотальное.

Связь **POST SKLAD.**

В спецификациях было написано, что поставка осуществляется только на один склад, следовательно, данная связь имеет кардинальность 1:1. Поскольку каждая поставка осуществляется на склад, степень участия сущности POST в связи POST SKLAD является полной.

Связь **POST PSTV.**

Кардинальность данной связи 1:1, поскольку каждая поставка осуществляется только одним поставщиком в данный момент времени. Степень участия сущности PSTV в данной связи является частичной.

Связь **OTGR SKLAD.**

В ходе аналогичных рассуждений получаем, что кардинальность данной связи 1:1, степень участия сущности SKLAD в связи OTGR SKLAD частичная.

Связь **OTGR POKUP.**

Кардинальность 1:1. Сущность OTGR участвует в связи OTGR□ POKUP полностью.

Связь **IZDEL□OTGR.**

Кардинальность 1:M. Сущность IZDEL участвует в данной связи частично.

Связь **KOMP□POST.**

Кардинальность данной связи 1:M, так как поставляться может несколько видов материалов и компонентов. Сущность POST полностью занята в связи KOMP□ POST.

Определение атрибутов и связывание их с типами сущностей и связей.

Следующим шагом является выделение атрибутов сущностей. Атрибут описывает некоторый аспект определенной сущности или связи.

Тип сущности	**Атрибут**
PSTV	**K_PSTV (код поставщика)**
	PSTV (название)
	GOROD (город)
	ADR (адрес)
POKUP	**K_POKUP(код покупателя)**
	POKUP (название)
	GOROD (город)
	ADR (адрес)
IZDEL	**K_IZDEL (код изделия)**
	IZDEL (наименование)
	KHAR (характеристика)
	ZENA_ED (цена за ед.)

KOMP	**K_KOMP (код детали)**
	KOMP (наименование)
	KHAR (характер.)
	ZENA_ED (цена за ед.)
POST	**N_POST (№ пост.)**
	KOL_VO (кол-во)
	DATA _POST (дата пост.)
OTGR	**N_OTGR (№ отгр.)**
	KOLVO (кол-во)
	DATA_OTGR (дата отгр.)
SKLAD	**N_SKLAD (№ склада)**
	FIO (отв. лицо)
	KOLVO (кол-во)
	DATA_OPER (дата опер.)

Определение атрибутов, являющихся потенциальными и первичными ключами.

Тип сущности	**Первичный ключ**	**Потен. ключ.**
IZDEL	**K_IZDEL**	**IZDEL**
KOMP	**K_KOMP**	**KOMP**
PSTV	**K_PSTV**	
POKUP	**K_POKUP**	
OTGR	**N_OTGR**	
POST	**N_POST**	

SKLAD **N_SKLAD, K_KOMP, DATA_OPER**

Определение набора отношений исходя из структуры локальной логической модели данных.

На этом этапе создаются отношения, представляющие сущности и связи, присутствующие в локальной логической модели данных представления пользователя «начальник склада» приложения «Сборочное предприятие». Связи между сущностями моделируются с помощью механизма первичных и внешних ключей.

Сильные типы сущностей

Для каждой сильной сущности в локальной модели данных создается отношение, включающее все простые атрибуты этой сущности. В случае составных атрибутов (например, адреса) в отношение включаются только составляющие их простые атрибуты (такие как город, улица, номер дома).

Слабые типы сущностей

Для каждой слабой сущности, присутствующей в логической модели, создается отношение, включающее все простые атрибуты этой сущности. Дополнительно в отношение включаются атрибуты внешнего ключа, соответствующего первичному ключу сущности-владельца. Первичный ключ слабой сущности частично или полностью зависит от ключа сущности-владельца.

Бинарные связи типа 1:1

Для каждой присутствующей в логической модели данных бинарной связи типа 1:1, установленной между сущностями E1 и E2, надо переслать атрибуты первичного ключа сущности E1 в отношение, представляющее сущность E2. Эти атрибуты будут использоваться в нем в качестве внешнего ключа. Определение родительской и дочерней сущностей зависит от

ограничений участия, наложенных на члены отношения E1 и E2. Сущность, которая частично участвует в связи, определяется как родительская, а та сущность, которая участвует в связи полностью определяется как дочерняя.

Бинарные связи типа 1:M

Для каждой бинарной связи типа 1:M, установленной в логической модели данных между сущностями E1 и E2, необходимо переслать копию атрибутов первичного ключа сущности E1 в отношение, представляющее сущность E2, где они будут играть роль внешнего ключа. Сущность, представляющая ’единичную’ сторону связи определяется как родительская. А сущность, представляющая ‘множественную ’ сторону - как дочерняя.

В результате окончательный вид отношений имеет следующий вид:

POST (N_POST, K_KOMP, K_PSTV, KOLVO, N_SKLAD, DATA_POST)

primary key N_POST;

foreign key (K_KOMP) references KOMP (K_KOMP);

foreign key (K_PSTV) references PSTV (K_PSTV);

foreign key (N_SKLAD) references SKLAD (N_SKLAD);

OTGR (N_OTGR, K_IZDEL, K_POKUP, KOLVO, N_SKLAD, DATA _OTGR)

primary key N_OTGR;

foreign key (K_IZDEL) references IZDEL (K_IZDEL);

foreign key (K_POKUP) references POKUP (K_POKUP);

foreign key (N_SKLAD) references SKLAD (N_SKLAD);

IZDEL (K_IZDEL, IZDEL KHAR, ZENA_ED)

primary key (K_IZDEL);

KOMP (K_KOMP, KOMP, KHAR, ZENA_ED);

primary key (K_KOMP);

PSTV (K_PSTV, PSTV GOROD, ADR);

primary key (K_PSTV);

POKUP (K_POKUP, POKUP, GOROD, ADR);

primary key (K_POKUP);

SKLAD (N_SKLAD, FIO, K_KOMP, K_IZDEL, KOLVO, DATA_OPER);

foreign key (K_KOMP) references KOMP (K_KOMP);

foreign key (K_IZDEL) references IZDEL (K_IZDEL);

Проверка модели с помощью правил нормализации

На этом этапе необходимо проверить созданный для представления пользователя «начальник склада» приложения «Сборочное предприятие» набор отношений на соответствие всем требованиям процедуры нормализации отношений и включает следующие действия:

- приведение к 1НФ, позволяющее удалить из отношений повторяющиеся группы атрибутов:
- приведение ко 2НФ, позволяющее устранить частичную зависимость атрибутов от первичного ключа;
- приведение к 3НФ, позволяющее удалить транзитивную зависимость атрибутов от первичного ключа;
- приведение к нормальной форме Бойса-Кодда, позволяющее удалить из функциональных зависимостей оставшиеся аномалии

Чтобы убедиться, что каждое из отношений находятся, как минимум, в НФБК, проанализируем функциональные зависимости между этими отношениями.

IZDEL (K_IZDEL, IZDEL, KHAR, ZENA_ED)

Pr. key K_IZDEL

K_IZDEL□ IZDEL, KHAR, ZENA_ED;

KOMP (K_KOMP, KOMP, KHAR, ZENA_ED)

Pr. key K_KOMP

K_KOMP□ KOMP, KHAR, ZENA_ED;

POKUP (K_POKUP, POKUP, GOROD, ADR);

Pr. key K_POKUP

K_POKUP□ POKUP, GOROD, ADR;

PSTV (K_PSTV, PSTV, GOROD, ADR)

Pr. key K_PSTV

K_PSTV□ PSTV, GOROD, ADR;

POST (N_POST, K_KOMP, K_PSTV, KOLVO, DATA_POST)

Pr. key N_POST

N_POST□ K_KOMP, K_PSTV, KOLVO, N_SKLAD, DATA_POST ;

OTGR (N_OTGR, K_IZDEL, K_POKUP, KOLVO, DATA_OTGR)

Pr. key N_OTGR

N_OTGR □ K_IZDEL, K_POKUP, N_SKLAD, KOLVO, DATA_OTGR ;

SKLAD (N_SKLAD, FIO K_KOMP, K_DET, KOLVO, DATA_OPER);

N_SKLAD, K_KOMP, DATA_OPER□ FIO, KOLVO;

Видим, что все эти отношения не содержат повторяющихся групп атрибутов и не имеют атрибутов, частично или транзитивно зависящих от первичных ключей этих отношений. Единственное, отношение SKLAD не находится в НФБК, но как мы видим оно не будет подвержено аномалиям обновления**.**

Спецификация требований представления пользователя «Менеджер по поставкам и по реализации».

Требования к данным

1. Предприятие заключает договора на продажу готовых изделий с другими организациями.
2. Каждая такая организация является покупателем, информация о котором содержит сведения об уникальном номере, названии, городе и адресе.
3. В обязанности менеджера входит составление договора на продажу. В этот документ входят план продажи на год, распределение плана по всем месяцам и дате начала действия договора. Каждый такой документ имеет уникальный номер.
4. Сведения об изделиях содержат информацию о его уникальном номере, названии, характеристики и цене за единицу продукции.
5. Предприятие не все составляющие двигателя изготавливает само. Часть сырья и деталей оно закупает у других предприятий.
6. Эти предприятия являются поставщиками. В информацию и них входит уникальный номер, название, город и адрес.
7. В обязанности менеджера по поставкам входит заключение договора с поставщиками на поставку сырья и компонентов.
8. Каждый такой договор имеет уникальный номер, план поставки на год, распределение по всем месяцам и дате начала действия договора.

9. Каждая поставляемая деталь характеризуется кодом, названием, характеристикой и ценой за единицу продукции.

Построение локальной концептуальной модели данных для представления пользователя «менеджер по поставкам и по продаже».

1. **Определение типов сущностей**

Договор	**DOGOVOR**
Заказ	**ZAKAZ**
Изделие	**IZDEL**
Материалы	**KOMP**
Покупатель	**POKUP**
Поставщик	**PSTV**

2. **Определение типов связей**

Тип сущности	**Тип связи**	**Тип сущности**
ZAKAZ	**заключается с**	**POKUP**
DOGOVOR	**заключается с**	**PSTV**
IZDEL	**отпускаются по**	**ZAKAZ**
KOMP	**поставляются по**	**DOGOVOR**

Теперь определим кардинальности и уровень участия этих типов сущностей в вышеперечисленных связях.

Связь ZAKAZ□ POKUP.

Каждый договор на продажу изделий заключается с покупателем. Связь типа 1:1. Сущность ZAKAZ участвует в связи полностью.

Связь **IZDEL□ ZAKAZ.** Кардинальность связи 1:M. Поскольку одно и о же изделие может продаваться по разным договорам. Степень участия IZDEL в данной связи частичная.

Связь **DOGOVOR□PSTV.** Кардинальность этой связи 1:1, так как каждый договор заключается только с одним поставщиком. Сущность DOGOVOR участвует в этой связи полностью.

Связь **KOMP□DOGOVOR.**

Кардинальность связи KOMP□ DOGOVOR 1:M, так как одна и та же деталь может поставляться по нескольким договорам.

Определение атрибутов и связывание их с типами сущностей.

A. Тип сущности	**Атрибут**
IZDEL	**K_IZDEL (код изделия)**
	IZDEL (наименование)
	KHAR (характеристика)
	ZENA_ED (цена за ед.)
POKUP	**K_POKUP (код покуп.)**
	POKUP (название)
	GOROD (город)
	ADR (адрес)
ZAKAZ	**N_ZAK (№ документа)**
	PLAN_GOD (план на год)
	YAN, FEV, MART…
	DATA_ZAK (дата начала договора)

PSTV	**K_PSTV**
	PSTV
	GOROD
	ADR
KOMP	**K_KOMP**
	KOMP
	GOROD
	ADR
DOGOVOR	**N_DOG (№ договора)**
	PLAN_GOD (план на год)
	YAN, FEV, MART…
	DATA_DOG (дата начала действия)

Определение атрибутов, являющихся потенциальными и первичными ключами.

Тип сущности	**Первичный ключ**	**Потенц. ключ**
PSTV	**K_PSTV**	**GOROD, ADR**
KOMP	**K_KOMP**	**KOMP**
DOGOVOR	**N_DOG**	
ZAKAZ	**N_ZAK**	
POKUP	**K_POKUP**	**GOROD,ADR**
IZDEL	**K_IZDEL**	**IZDEL**

Построение локальной логической модели данных для представления пользователя «менеджер по продаже и по поставкам».

На этом этапе обычно удаляются из концептуальной модели данных структуры, реализация которых в СУБД реляционного типа была бы затруднительна. Но, как видно, в нашем случае таких структур нет, т.е. нет связей типа M:N, сложных и рекурсивных связей. Поэтому сразу перейдем к следующему этапу.

Определение набора отношений. Напомню, что на данном этапе создаются отношения, представляющие сущности и связи локальной логической модели данных представления пользователя «менеджер по продаже и по поставкам». Связи между сущностями моделируются с помощью механизма первичных ключей и внешних ключей.

B. POKUP (K_POKUP, POKUP, GOROD, ADR)

Primary key K_POKUP

PSTV (K_PSTV, PSTV , GOROD, ADR)

Primary key K_PSTV

IZDEL (K_IZDEL , IZDEL, KHAR, ZENA_ED)

Primary key K_IZDEL

KOMP (K_KOMP, KOMP, KHAR, ZENA_ED)

Primary key K_KOMP

DOGOVOR (N_DOG, K_PSTV, K_KOMP, RLAN_GOD, YAN, FEV,…, DATA_DOG)

Primary key N_DOG

ZAKAZ (N_ZAK, K_POKUP, K_IZDEL, PLAN_GOD, YAN, FEV, …, DATA_DOG)

Primary key N_ZAK

Проверка модели с помощью правил нормализации

IZDEL (K_IZDEL, IZDEL, KHAR, ZENA_ED)

Pr. Key K_IZDEL

K_IZDEL□IZDEL, KHAR, ZENA_ED

POKUP (K_POKUP, POKUP, GOROD, ADR)

Pr.key K_POKUP

K_POKUP□POKUP, GOROD, ADR;

KOMP (K_KOMP, KOMP, KHAR, ZENA_ED)

Pr.key K_KOMP

K_KOMP□ KOMP, KHAR, ZENA_ED ;

PSTV (K_PSTV, PSTV, GOROD, ADR)

Pr. Key K_PSTV

K_PSTV□ PSTV, GOROD, ADR;

DOGOVOR (N_DOG, K_KOMP, K_PSTV, PLAN_GOD, YAN, FEV,…, DATA_DOG)

Pr. key N_DOG

N_DOG□ K_KOMP, K_PSTV, PLAN_GOD, YAN,…, DATA_DOG;

ZAKAZ (N_ZAK, K_IZDEL, K_POKUP, PLAN_GOD, YAN, FEV,…, DATA_DOG)

Pr. key N_ZAK

N_ZAK □ K_IZDEL, K_POKUP, PLAN_GOD, YAN,…, DATA_DOG;

Таким образом, мы видим, что все три отношения не содержат ни повторяющихся групп, ни атрибутов, частично или транзитивно зависящих от первичных ключей этих отношений. Более того, каждая из сущностей имеет только один детерминант, который является первичным ключом в соответствующем отношении. В результате можно сделать вывод, что эти отношения находятся в НФБК.

Спецификация требований представления пользователя «начальник цеха».

Требования к данным

1. Предприятие состоит из нескольких цехов, которые , в свою очередь, разбиты на участки. В сведения о цехах-участках входят их уникальный номер и название.

2. В цехах-участках трудятся рабочие. Информация о каждом рабочем содержит сведения о его табельном номере, фио, должности, семейном положении и о дате приема на работу.
3. В цехах изготавливаются составляющие двигателя и сам двигатель, который характеризуются своим уникальным номером, названием, характеристикой, единицей измерения и ценой за единицу продукции.
4. В цехах-участках протекают различные операции, в которых участвуют детали (сырье) и изделия.
5. Детали и сырье предприятие не изготавливает само, а закупает у других предприятий. Сведения о привозных компонентах содержат информацию о коде, названии, характеристике и цене за штуку.
6. На каждом участке в ходе каждой операции по изготовлению составляющих двигателя расходуется определенное количество сырья.
7. Каждый рабочий имеет профессию и разряд.
8. На каждом участке входе операции по изготовлению изделия рабочим ведется учет его выработки
9. На каждом участке на операцию по изготовлению изделия рабочим с определенной профессией и разрядом затрачивается определенное количество времени.

Требования к транзакциям

К основным транзакциям, которые должны выполняться пользователем «начальник цеха» относятся следующие:

a) составление списка изделий , производимых на участках данного цеха
b) составление списка используемых компонентов
c) составление перечня операций , протекающих в цехе
d) поиск работников удовлетворяющих различным требованиям
e) составление списка расхода материала
f) учет выработки работников

Определение типов сущностей

работа начинается с определения основных типов сущностей исходя из имеющихся спецификаций. В спецификациях сущности обычно представлены как существительные . Анализ показывает , что основными сущностями , упоминаемыми в спецификациях являются следующие:

ПОДРАЗДЕЛЕНИЯ	**ZECH**
РАБОЧИЕ	**RABOT**
ПРОФЕССИЯ	**PROF**
РАЗРЯД	**TARIF**
ИЗДЕЛИЯ	**IZDEL**
МАТЕРИАЛЫ	**KOMP**
ОПЕРАЦИИ	**OPER**

Определение типов связей

Следующий шаг состоит в определении типов связей, существующих между отдельными сущностями. Для выявления всех возможных типов связей вновь обратимся к спецификациям.

Тип сущности	**Тип связи**	**Тип сущности**
RABOT	**обладает**	**PROF**
RABOT	**имеет**	**TARIF**
KOMP	**участвуют**	**OPER**
KOMP	**расходуется**	**ZECH**
KOMP	**расходуется на**	**IZDEL**
KOMP	**расходуется в**	**OPER**
IZDEL	**участвует**	**OPER**
IZDEL	**изготавливается в**	**OPER**
IZDEL	**изготавливается в**	**ZECH**
IZDEL	**изготавливается**	**RABOT**
PROF	**затрачивается на**	**IZDEL**

TARIF затрачивается на IZDEL

Определение степени участия и кардинальности каждого типа связи.

Связь **IZDEL RABOT.** Кардинальность данной связи M:N, так как в процессе изготовления изделий принимают участие не один рабочий.

Связь **IZDEL ZECH.** Кардинальность этой связи также M:N, так как изделия изготавливаются во многих цехах.

Связь **IZDEL OPER.** Кардинальность данной связи M:N, так как изделия проходят разную степень обработки в разных цехах.

Связь **ZECH OPER.** Кардинальность связи 1:M. В цехе протекает несколько операций.

Связь **KOMP OPER (**материалы расходуются в операциях**)** Кардинальность M:N.

Связь **KOMP OPER (**материалы участвуют в операциях**)** Связь типа 1:M, так как один и тот же материал может участвовать в разных операциях.

Связь **RABOT PROF.** Связь типа 1:1.Каждый рабочий имеет только одну профессию.

Связь **RABOT TARIF.** Кардинальность данной связи также 1:1, так как каждый рабочий имеет только один разряд.

Определение атрибутов сущностей

Тип сущности	**Атрибут**
IZDEL	**K_IZDEL**

	IZDEL
	KHAR
	ED_IZM
KOMP	**K_KOMP**
	KOMP
	KHAR
	ED_IZM
RABOT	**TAB_N** (таб. №)
	FIO (ФИО)
	SEM_POL (семейное положение)
	DATA_ZACH (дата приема на работу)
OPER	**N_OPER** (№ операции)
TARIF	**N_TARIF** (№ разряда)
	TARIF (часовая тарифная ставка)
PROF	**K_PROF** (код проф.)
	PROF (название)

Определение первичных ключей и потенциальных ключей.

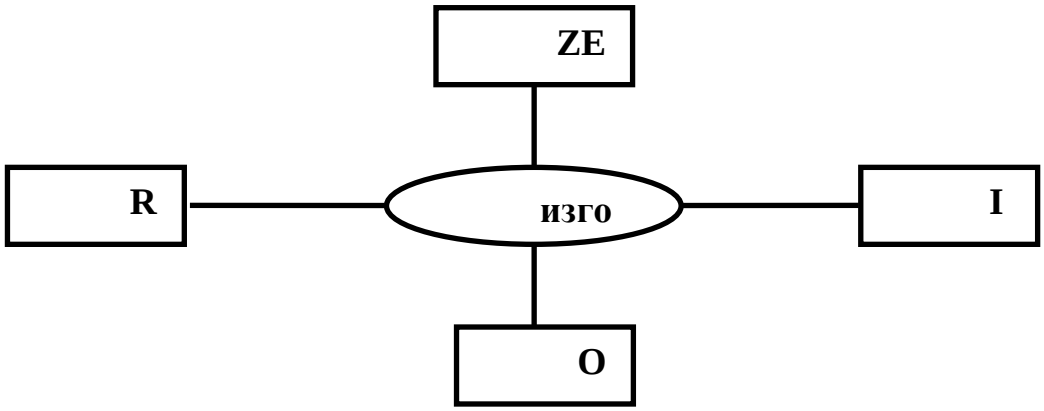
Тип сущности	**Первичный ключ**	**Потенц. Ключ**
IZDEL	**K_IZDEL**	**IZDEL**
KOMP	**K_KOMP**	**KOMP**
RABOT	**TAB_N**	
TARIF	**N_TARIF**	**TARIF**
PROF	**K_PROF**	**PROF**

Построение и проверка локальной логической модели для представления пользователя «начальник цеха».

В результате выполнения первого этапа мы получили набор локальных концептуальных моделей данных. Однако в них присутствуют структуры, реализация которых в СУБД реляционного типа будет затруднена. Имеются в виду сложные связи, существующие между отдельными типами сущностей.

Удаление сложных связей.

Прочитаем спецификацию N 9 : ’на каждом участке в ходе операции по изготовлению изделия рабочим ведется учет его выработки.’в этой связи участвуют четыре связи. Для удаления такой сложной связи объединим все эти сущности в промежуточную. И получим



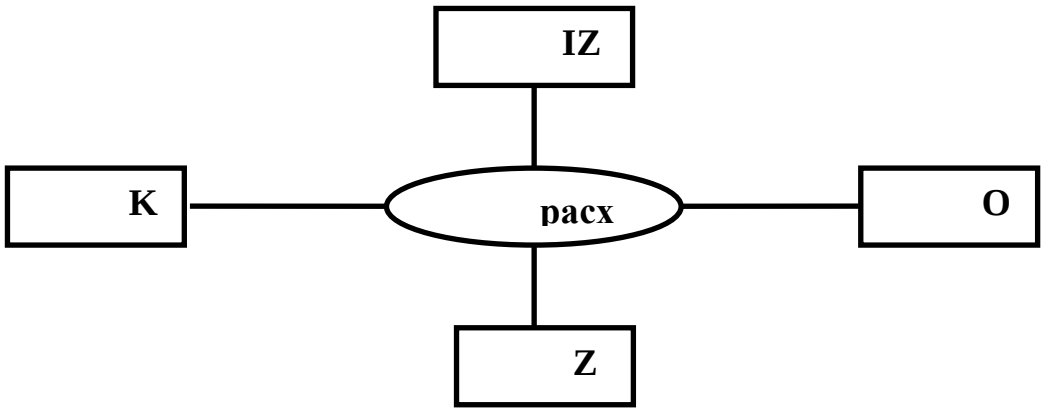
RABOT VIRABOTKA.Тип связи 1:M.

IZDEL VIRABOTKA. 1:M

OPER VIRABOTKA. 1:M

ZECH VIRABOTKA. 1:M

Спецификация N7: ‘На каждом участке в ходе каждой операции по изготовлению составляющих двигателя (т.е. изделий) расходуется



определенное количество сырья (т.е. материала)

Объединив эти четыре сущности KOMP, IZDEL, OPER, ZECH в единую промежуточную сущность RASHOD (расход) мы тем самым удалим еще одну сложную связь. В замен старой связи получим 4 связи типа 1:M

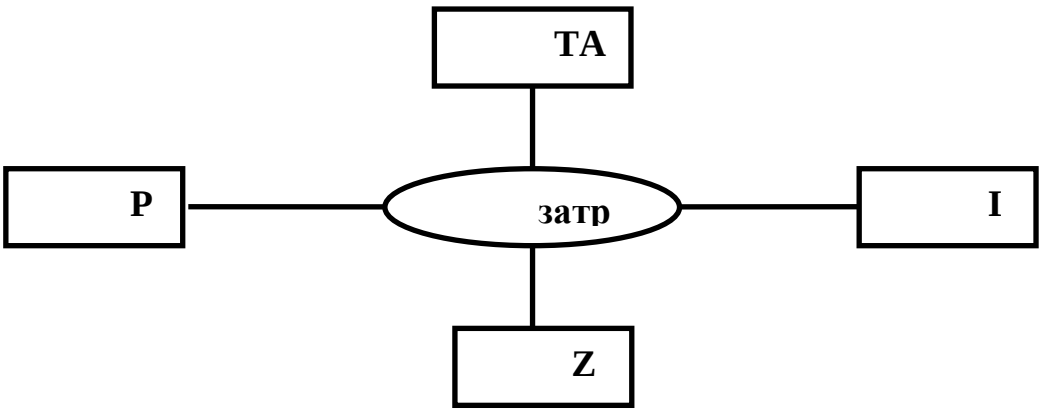
KOMP□RASHOD 1:M

IZDEL□RASHOD. 1:M

OPER□RASHOD. 1:M

ZECH□RASHOD.1:M

Спецификация N 10: ‘На каждом участке на каждую операцию по изготовлению изделия рабочим с определенной профессией и разрядом



затрачивается определенное количество времени’.

Для получения бинарной связи объединим сущности PROF, TARIF, IZDEL, ZECH в одну сущность под названием **ZATRATA** (нормы затрат труда).

Определение набора отношений

Определим сначала сильные типы сущностей

IZDEL (K_IZDEL, IZDEL, KHAR, ED_IZM)

primary key K_IZDEL

KOMP (K_KOMP, KOMP, KHAR, ED_IZM)

Primary key K_KOMP

PROF (K_PROF, PROF)

Primary key K_PROF

TARIF (N_TARIF, TARIF)

Primary key N_TARIF

Сущность **RABOT** имеет две родительские сущности **PROF** и **TARIF**-поэтому в ее отношение следует поместить копии первичных ключей обеих этих сущностей. Получим:

RABOT (TAB_N, FIO, K_PROF, N_TARIF, SEM_POL, DATA_ZACH)

Primary key TAB_N

Для каждой слабой сущности в модели данных следует создать отношение, в которое в качестве внешнего ключа следует поместить первичные ключи всех ее родительских сущностей. В нашем случае это сущности **RASHOD, VIRABOTKA, ZATRATA.**

RASHOD (K_KOMP, K_IZDEL, N_OPER, N-ZECH, ED-IZM, NORMA)

primary key K_KOMP, N_OPER

foreign key K_KOMP, K_IZDEL, N_OPER, N_ZECH

ZATRATA (K_IZDEL, N_OPER, N_ZECH, K_PROF, N_TARIF DOPOLN, VREMYA_ED)

primary key K-IZDEL, N_OPER

foreign key K_IZDEL, N_OPER, N_ECH, K_PROF, N_TARIF

VIRABOTKA (N_ECH, K_IZDEL, N_OPER, TAB_N, KOLV_GOD, KOLVO_BR, PROC_BR, DATA_VIR)

Primary key TAB_N, DATA_VIR

foreign key N_ZECH, K_IZDEL, N_OPER, TAB_N

Проверка модели с помощью правил нормализации

Чтобы убедиться, что каждое из полученных отношений находится в НФБК, проанализируем функциональные зависимости между этими отношениями.

IZDEL (...)

K_IZDEL□IZDEL, KHAR, ED_IZM;

KOMP(...)

K_KOMP□KOMP, KHAR, ED_IZM;

RABOT (...)

TAB_N□FIO, K_PROF, N_TARIF, SEM_POL, DATA_ZACH;

TARIF (...)

N_TARIF□TARIF

PROF (...)

K_PROF□PROF;

Эти пять отношений находятся в НФБК, поэтому никаким аномалиям обновления они подвержены не будут. Что касается остальных отношений RASHOD, VIRABOTKA, ZATRATA, то хоть они и не нормализованы до НФБК также не будут подвержены вышеупомянутым аномалиям.

Спецификация требований для представления пользователя «бухгалтер».

Требования к данным

1. Каждый сотрудник предприятия получает заработную плату.

2. Зарплата начисляется в соответствие с занимаемой должностью и учетом выработки (для рабочих)

Требования к транзакциям

a) Создание и редактирование записей о начислении заработной платы

b) создание отчета о начислении (квитанции для получения зарплаты в кассе)

Построение локальной концептуальной модели для пользователя «бухгалтер»

Определение сущностей

В спецификациях упоминаются следующие сущности:

СОТРУДНИКИ RABOT

РАЗРЯД TARIF

ЗАРПЛАТА ZARPLATA

Определение связей между сущностями

RABOT□ ZARPLATA. Каждый сотрудник получает зарплату. Тип связи 1:M

RABOT□TARIF. Каждый рабочий имеет разряд. Связь типа 1:1.Сущность RABOT участвует в этой связи полностью.

Определение атрибутов

Тип сущности	**Атрибут**
RABOT	**TAB_N**
	FIO
	POSITION
	SEM_POL
	DATA_ZACH
ZARPLATA	**SUM_NACH (начислено)**
	SUM_UDER (удержано)
	VIDACHA (к выдаче)
	DATA_VID (дата выдачи)
TARIF	**N_TARIF**
	TARIF

Определение первичных и потенциальных ключей

Тип сущности	**Первичный ключ**	**Потен. ключ**
RABOT	**TAB_N**	
TARIF	**N_TARIF**	**TARIF**
ZARPLATA		

Как видим, сущность ZARPLATA не имеет первичного ключа, т.е. она является слабой сущностью. Ее первичный ключ частично или полностью зависит от сущности владельца.

Определение отношений. Так как в логической модели у нас нет никаких ‘лишних’ структур, то сразу можно перейти к определению отношений.

RABOT (TAB_N, FIO, POSITION, N_TARIF, SEM_POL, DATA_ZACH)

Primary key TAB_N; foreign key N_TARIF;

ZARPLATA (TAB_N, SUM_NACH, SUM_UDER, VIDACHA, DATA_VID);

primary key TAB_N, DATA_VID;

foreign key TAB_N;

В отношении правил нормализации первые два отношения находятся в НФБК, так как нет ни повторяющихся групп, ни атрибутов. Частично или полностью зависящих от первичного ключа и, более того, детерминанты отношений являются их первичными ключами. А что касается отношения ZARPLATA, хоть оно и не находится в НФБК аномалиям обновления подвержено не будет.

Слияние локальных логических моделей в единую глобальную модель данных.

В небольших системах, насчитывающих 2-3 пользовательских представления с незначительным количеством типов сущностей и связей, задача сравнения локальных моделей с последующим слиянием и устранением любых возможных противоречий является относительно несложной. Существует систематический подход, который можно использовать для выполнения слияния локальных моделей и разрешения любых возникающих при этом проблем. Предлагаемый подход предусматривает выполнение следующих действий

1. Анализ имен сущностей и их первичных ключей
2. Анализ имен связей
3. Слияние общих сущностей из отдельных локальных моделей
4. Включение (без слияния) сущностей, уникальных для каждого локального представления
5. Слияние общих связей из отдельных локальных представлений
6. Включение (без слияния) связей, уникальных для каждого локального представления.

Таким образом, мы имеем следующее:

В результате слияния сущностей **IZDEL** из представлений начальник цеха, начальник склада, менеджер по продаже и по реализации получаем глобальное представление

IZDEL (K_IZDEL, IZDEL, ED_IZM, KHAR, ZENA_ED)

primary key K_IZDEL;

Глобальное представление сущности **KOMP,** полученное в результате слияния сущностей с таким именем из представлений начальник цеха, начальник склада, менеджер по продаже и по реализации.

KOMP (K_KOMP, KOMP, KHAR, ED_IZM, ZENA_ED)

primary key K_KOMP;

Глобальное представление сущности **RABOT**

RABOT (TAB_N, FIO, K_PROF, N_TARIF, POSITION, SEM_POL, DATA_ZACH)

Primary key TAB_N;

Глобальное представление сущности **PSTV**

PSTV (K_PSTV, PSTV, GOROD, ADRESS)

Primary key K_PSTV;

Глобальное представление сущности **POKUP**

POKUP (K_POKUP, POKUP, GOROD, ADRESS)

Primary key K_POKUP;