# ВВЕДЕНИЕ В СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

#### Учебное пособие

Учебное пособие было опубликовано в 1999 г. в двух частях издательством Башкирского государственного университета.

Выходные данные: УДК 519.6 ББК 32.973-018.2

Пушников А.Ю. Введение в системы управления базами данных. Часть 1. Реляционная модель данных: Учебное пособие/Изд-е Башкирского ун-та. - Уфа, 1999. - 108 с. - ISBN 5-7477-0350-1.

Пушников А.Ю. Введение в системы управления базами данных. Часть 2. Нормальные формы отношений и транзакции: Учебное пособие/Изд-е Башкирского ун-та. - Уфа, 1999. - 138 с. - ISBN 5-7477-0351-X.

Рецензенты: канд.тех.н., доц. Мартынов В.В. (Уфимский государственный авиационный техни-ческий университет); кафедра автоматизированных систем управления УГАТУ.

В настоящий момент автор работает в Уфимском представительстве компании AйTu (web page: <u>www.it.ru</u>, <u>www.itc.ufanet.ru</u>).



Связаться с автором можно:

Раб.тел.: (3472) 25-37-71, 25-38-11, 25-38-53

Дом.тел.: (3472) 54-90-84

E-mail: apushnik@itc.ufanet.ru, apushnik@mail.ru

## Оглавление

## **ВВЕДЕНИЕ**

#### ГЛАВА 1. ЭЛЕМЕНТЫ ТЕОРИИ МНОЖЕСТВ

Множества

Операции над множествами

Декартово произведение множеств

Отношение

Примеры отношений

Бинарные отношения (отношения степени 2)

Отношение эквивалентности

Отношения порядка

Функциональное отношение

Еще пример бинарного отношения

<u>n-арные отношения (отношения степени n)</u>

Транзитивное замыкание отношений

Выводы

# ГЛАВА 2. БАЗОВЫЕ ПОНЯТИЯ РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ

Общая характеристика реляционной модели данных

Типы данных

Простые типы данных

Структурированные типы данных

Ссылочные типы данных

Типы данных, используемые в реляционной модели

Домены

Отношения, атрибуты, кортежи отношения

Определения и примеры

Свойства отношений

Первая нормальная форма

Выводы

## ГЛАВА 3. ЦЕЛОСТНОСТЬ РЕЛЯЦИОННЫХ ДАННЫХ

Null-значения

Трехзначная логика (3VL)

Потенциальные ключи

Целостность сущностей

Внешние ключи

Целостность внешних ключей

Замечания к правилам целостности сущностей и внешних ключей

Операции, могущие нарушить ссылочную целостность

Для родительского отношения

Для дочернего отношения

Стратегии поддержания ссылочной целостности

Применение стратегий поддержания ссылочной целостности

При обновлении кортежа в родительском отношении

При удалении кортежа в родительском отношении

При вставке кортежа в дочернее отношение

При обновлении кортежа в дочернем отношении

Выводы

#### ГЛАВА 4. РЕЛЯЦИОННАЯ АЛГЕБРА.

Обзор реляционной алгебры

Замкнутость реляционной алгебры

Отношения, совместимые по типу

Оператор переименования атрибутов

Теоретико-множественные операторы

Объединение

Пересечение

Вычитание

Декартово произведение

Специальные реляционные операторы

Выборка (ограничение, селекция)

Проекция

Соединение

Общая операция соединения

Тэта-соединение

Экви-соединение

Естественное соединение

Деление

Примеры использования реляционных операторов

Зависимые реляционные операторы

Оператор соединения

Оператор пересечения

Оператор деления

Примитивные реляционные операторы

Оператор декартового произведения

Оператор проекции

Оператор выборки

Операторы объединения и вычитания

Запросы, невыразимые средствами реляционной алгебры

Плохая нормализация отношений

Невыразимость транзитивного замыкания реляционными операторами

Кросс-таблицы

Выводы

#### ГЛАВА 5. ЭЛЕМЕНТЫ ЯЗЫКА SQL.

Операторы SQL

Операторы DDL (Data Definition Language) - операторы определения объектов базы ланных

<u>Операторы DML (Data Manipulation Language) - операторы манипулирования</u> данными

Операторы защиты и управления данными

Примеры использования операторов манипулирования данными

INSERT - вставка строк в таблицу

UPDATE - обновление строк в таблице

DELETE - удаление строк в таблице

Примеры использования оператора SELECT

Отбор данных из одной таблицы

Отбор данных из нескольких таблиц

Использование имен корреляции (алиасов, псевдонимов)

Использование агрегатных функций в запросах

Использование агрегатных функций с группировками

Использование подзапросов

Использование объединения, пересечения и разности

Синтаксис оператора выборки данных (SELECT)

BNF-нотация

Синтаксис оператора выборки

Синтаксис соединенных таблиц

Синтаксис условных выражений раздела WHERE

Порядок выполнения оператора SELECT

Стадия 1. Выполнение одиночного оператора SELECT

Стадия 2. Выполнение операций UNION, EXCEPT, INTERSECT

Стадия 3. Упорядочение результата

Как на самом деле выполняется оператор SELECT

Реализация реляционной алгебры средствами оператора

SELECT (Реляционная полнота SQL)

Оператор декартового произведения

Оператор проекции

Оператор выборки

Оператор объединения

Оператор вычитания

Оператор соединения

Оператор пересечения

Оператор деления

Выводы

# ГЛАВА 6. НОРМАЛЬНЫЕ ФОРМЫ ОТНОШЕНИЙ

Этапы разработки базы данных

Критерии оценки качества логической модели данных

Адекватность базы данных предметной области

Легкость разработки и сопровождения базы данных

Скорость операций обновления данных (вставка, обновление, удаление)

Скорость операций выборки данных

Основной пример

1НФ (Первая Нормальная Форма)

Аномалии обновления

Аномалии вставки (INSERT)

Аномалии обновления (UPDATE)

Аномалии удаления (DELETE)

Функциональные зависимости

Определение функциональной зависимости

Функциональные зависимости отношений и математическое понятие

функциональной зависимости

2НФ (Вторая Нормальная Форма)

Анализ декомпозированных отношений

Оставшиеся аномалии вставки (INSERT)

Оставшиеся аномалии обновления (UPDATE)

Оставшиеся аномалии удаления (DELETE)

3НФ (Третья Нормальная Форма)

Алгоритм нормализации (приведение к 3НФ)

Анализ критериев для нормализованных и ненормализованных моделей данных

Сравнение нормализованных и ненормализованных моделей

OLTP и OLAP-системы

Корректность процедуры нормализации - декомпозиция без потерь. Теорема Хеза

Выводы

#### ГЛАВА 7. НОРМАЛЬНЫЕ ФОРМЫ БОЛЕЕ ВЫСОКИХ ПОРЯДКОВ

НФБК (Нормальная Форма Бойса-Кодда)

4НФ (Четвертая Нормальная Форма)

5НФ (Пятая Нормальная Форма)

Продолжение алгоритма нормализации (приведение к 5НФ)

Выводы

#### ГЛАВА 8. ЭЛЕМЕНТЫ МОДЕЛИ "СУЩНОСТЬ-СВЯЗЬ"

Основные понятия ER-диаграмм

Пример разработки простой ER-модели

## Концептуальные и физические ER-модели

Выволы

## ГЛАВА 9. ТРАНЗАКЦИИ И ЦЕЛОСТНОСТЬ БАЗ ДАННЫХ

Пример нарушения целостности базы

Понятие транзакции

Ограничения целостности

Классификация ограничений целостности

Классификация ограничений целостности по способам реализации

Классификация ограничений целостности по времени проверки

Классификация ограничений целостности по области действия

Ограничения домена

Ограничения атрибута

Ограничения кортежа

Ограничения отношения

Ограничения базы данных

Реализация декларативных ограничений целостности средствами SQL

Общие принципы реализации ограничений средствами SQL

Синтаксис ограничений стандарта SQL

Синтаксис операторов SQL, использующих ограничения

Выводы

## ГЛАВА 10. ТРАНЗАКЦИИ И ПАРАЛЛЕЛИЗМ

Работа транзакций в смеси

Проблемы параллельной работы транзакций

Проблема потери результатов обновления

Проблема незафиксированной зависимости (чтение "грязных" данных,

неаккуратное считывание)

Проблема несовместимого анализа

Неповторяемое считывание

Фиктивные элементы (фантомы)

Собственно несовместимый анализ

Конфликты между транзакциями

Блокировки

Решение проблем параллелизма при помощи блокировок

Проблема потери результатов обновления

Проблема незафиксированной зависимости (чтение "грязных" данных,

неаккуратное считывание)

Проблема несовместимого анализа

Неповторяемое считывание

Фиктивные элементы (фантомы)

Собственно несовместимый анализ

Разрешение тупиковых ситуаций

<u>Преднамеренные блокировки</u>

Предикатные блокировки

Метод временных меток

Механизм выделения версий данных

Теорема Есварана о сериализуемости

Реализация изолированности транзакций средствами SQL

Уровни изоляции

# <u>Синтаксис операторов SQL, определяющих уровни изоляции</u> Выводы

## ГЛАВА 11. ТРАНЗАКЦИИ И ВОССТАНОВЛЕНИЕ ДАННЫХ

Виды восстановления данных

Индивидуальный откат транзакции

Восстановление после мягкого сбоя

Восстановление после жесткого сбоя

Восстановление данных и стандарт SQL

Выводы

## СПИСОК ЛИТЕРАТУРЫ

#### Введение

Основное назначение данного учебного пособия - дать систематическое введение в основы реляционной модели данных и принципы функционирования реляционных баз данных.

Реляционная модель описывает, какие данные могут храниться в реляционных базах данных, а также способы манипулирования такими данными. В упрощенном виде основная идея реляционной модели состоит в том, что данные должны храниться в таблицах и только в таблицах. Эта, кажущаяся тривиальной, идея оказывается вовсе не простой при рассмотрении вопроса, а что, собственно, представляет собой таблица? В данный момент существуем много различных систем обработки данных, оперирующих понятием "таблица", например, всем известные, электронные таблицы, таблицы текстового редактора MS Word, и т.п. Ячейки электронной таблицы могут хранить разнотипные данные, например, числа, строки текста, формулы, ссылающиеся на другие ячейки. Собственно, на одном листе электронной таблицы можно разместить несколько совершенно независимых таблиц, если под таблицей понимать прямоугольную область, расчерченную на клеточки и заполненную данными. Таблицы текстовых редакторов вообще могут иметь совершенно произвольную структуру, например, как на рисунке:

Отдел	Сотрудники	Дети сотрудников (интересы)		
Цех	Иванов И.И.	Маша	ЛЕГО	
		Петя	Книги	Видео
		Саша	Компьютеры	
		Дима	Спорт	
	Петров П.П.	Артур	Ничем не интересуется	
	Сидоров С.С.	Сергей	Компьютеры Книги	
		Валерий	Книги	
		Станислав	Видео	
Бухгалтерия	•••			

Таблица 1 Таблица произвольной формы

Конечно, и электронные таблицы, и текстовые редакторы позволяют хранить и обрабатывать данные очень гибко, но как быть, если требуется хранить информацию обо всех сотрудниках большого предприятия и периодически выдавать ответы на запросы типа "представить список всех сотрудников, принятых на работу не позднее трех лет назад, имеющих по крайней мере одного ребенка, не имеющих взысканий и с зарплатой не выше 1000 р.". Для получения ответов на подобные запросы и предназначены *Системы Управления Базами Данных (СУБД)*.

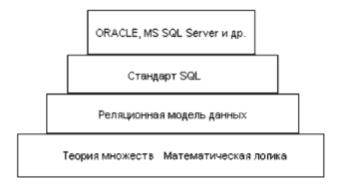
Классическая реляционная модель данных требует, чтобы данные хранились в так называемых плоских таблицах. Более точно, пользователи и приложения, обращающиеся к данным, должны работать с данными так, как если бы они размещались в таких таблицах. В упрощенном виде плоская таблица - это таблица, каждая ячейка которой может быть однозначно идентифицирована указанием строки и столбца таблицы. Кроме того, в одном столбце все ячейки должны содержать данные одного простого типа. Точное определение понятия "плоская таблица" дается в реляционной модели данных.

Реляционная модель основана на теории множеств и математической логике. Такой фундамент обеспечивает математическую строгость реляционной модели данных.

В свою очередь, на основе реляционной модели были разработаны различные языки для доступа к реляционным данным, такие как SEQUEL, SQL, QUEL и другие. Фактическим промышленным стандартом в настоящее время стал язык SQL (Structured Query Language - язык структурированных запросов).

Различные фирмы, производители СУБД, предлагают свои реализации языка SQL. Эти реализации отличаются как друг от друга, так и от стандартизованного языка SQL. Это и хорошо и плохо. Хорошо это тем, что конкретная реализация языка, может включать в себя более широкие возможности по сравнению со стандартизованными SQL, например, больше типов данных, большее количество команд, больше дополнительных опций имеющихся команд. Такие возможности делают работу с конкретной СУБД более эффективной. Кроме того, такие нестандартные возможности языка проходят практическую апробацию и со временем могут быть включены в стандарт. Плохо же это тем, что различия в синтаксисе реализаций SQL затрудняют перенос приложений из одной системы в другую. Например, если приложение было написано для базы данных MS SQL Server с использованием своего диалекта SQL - языка Transact-SQL, то при переносе системы в базу данных ORACLE, не все конструкции языка будут понятны соответствующему диалекту SQL - языку PL/SQL.

Взаимосвязь реляционной модели данных, стандарта языка SQL и различных его реализаций можно условно изобразить в виде следующей пирамиды:



Каждый более высокий уровень основывается на понятиях, определенных на более низком уровне. На каждом из уровней используется своя терминология. Например, на уровне теории множеств мы говорим "множество", "подмножество декартового произведения", "кортеж". На уровне реляционной модели используем термины "домен", "отношение", "кортеж". На уровне стандарта SQL и конкретных реализаций используем термины "тип данных", "таблица", "строка таблицы". И каждый раз речь идет об одном и том же.

Учебное пособие имеет следующую структуру.

Первая глава содержит небольшое введение в математическую теорию множеств, необходимое для введения фундаментального понятия "отношение".

Следующие три главы содержат описание собственно реляционной модели данных. Во второй главе вводятся базовые понятия реляционной модели данных, такие как "домен", "отношение", "атрибут", "кортеж", "ключ", "внешний ключ". В третьей главе вводится понятие целостности реляционных данных. Даются понятия целостности сущностей и целостности внешних ключей. В четвертой главе описывается язык доступа к реляционным данным - реляционная алгебра.

Пятая глава содержит краткое описание и примеры применения стандартного языка доступа к реляционным данным - языка SQL.

Шестая и седьмая главы посвящены важным вопросам правильного проектирования отношений. В этих главах вводятся нормальные формы отношений. Понятие нормальных форм необходимо для проектирования непротиворечивых и неизбыточных таблиц базы данных.

В восьмой главе описывается альтернативный способ разработки таблиц в нормальной форме - модель "сущность-связь".

Последние три главы посвящены важному для баз данных понятию "транзакция". Понятие транзакции является фундаментальным при рассмотрении таких вопросов как поддержание целостности базы данных, независимой одновременной работы большого количества пользователей, восстановления данных после сбоев системы.

## Глава 1. Элементы теории множеств

## Множества

Наиболее простая структура данных, используемая в математике, имеет место в случае, когда между отдельными изолированными данными отсутствуют какие-либо взаимосвязи. Совокупность таких данных представляет собой *множество*. Понятие множества является неопределяемым понятием. Множество не обладает внутренней структурой. Множество можно представить себе как совокупность элементов, обладающих некоторым общим свойством. Для того чтобы некоторую совокупность элементов можно было назвать множеством, необходимо, чтобы выполнялись следующие условия:

1. Должно существовать правило, позволяющее определить, принадлежит ли указанный элемент данной совокупности.

2. Должно существовать правило, позволяющее отличать элементы друг от друга. (Это, в частности, означает, что множество не может содержать двух *одинаковых* элементов).

Множества обычно обозначаются заглавными латинскими буквами. Если элемент x принадлежит множеству A, то это обозначается:

$$x \in A$$

Если каждый элемент множества  $^{B}$  является также и элементом множества  $^{A}$ , то говорят, что множество  $^{B}$  является **подмножеством** множества  $^{A}$ :

$$B \subset A$$

Подмножество B множества A называется собственным подмножеством, если

$$B \neq A$$

Используя понятие множества можно построить более сложные и содержательные объекты.

# Операции над множествами

Основными операциями над множествами являются *объединение*, *пересечение* и *разность*.

Определение 1. Объединением двух множеств называется новое множество

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

Определение 2. Пересечением двух множеств называется новое множество

$$A \cap B = \{x \mid x \in A \text{ if } x \in B\}$$

Определение 3. Разностью двух множеств называется новое множество

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

Если класс объектов, на которых определяются различные множества обозначить  $\Omega$  (*Универсум*), то *дополнением* множества A называют разность

$$\overline{A} = \Omega \setminus A$$

## Декартово произведение множеств

Одним из способов конструирования новых объектов из уже имеющихся множеств является *декартово произведение множеств*.

Пусть  $A_{\mathsf{H}}$  B - множества. Выражение вида (a,b), где  $a \in A_{\mathsf{H}}$   $b \in B$ , называется упорядоченной парой. Равенство вида (a,b) = (c,d) означает, что  $a = c_{\mathsf{H}}$  b = d. В общем

случае, можно рассматривать *упорядоченную п-ку*  $(a_1,a_2,\dots,a_n)$  из элементов  $a_1\in A_1,\dots a_n\in A_n$  . Упорядоченные n-ки иначе называют *наборы* или *кортежи*.

Определение 4. **Декартовым** (**прямым**) **произведением множеств**  $A_1, A_2, \dots, A_n$  называется множество упорядоченных n-ок (наборов, кортежей) вида

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

*Определение 5. Степенью декартового произведения*  $A_1 \times A_2 \times ... \times A_n$  называется число множеств n, входящих в это декартово произведение.

<u>Замечание</u>. Если все множества  $^{A_{i}}$  одинаковы, то используют обозначение

$$A^n = A \times A \times ... \times A$$

#### Отношение

Определение 6. Подмножество R декартового произведения множеств  $A_1 \times A_2 \times ... \times A_n$  называется **отношением степени п** (**n-арным отношением**).

*Определение* 7. Мощность множества кортежей, входящих в отношение R, называют **мощностью отношения** R.

Замечание. Понятие отношения является очень важным не только с математической точки зрения. Понятие отношения фактически лежит в основе всей реляционной теории баз данных. Как будет показано ниже, отношения являются математическим аналогом *таблиц*. Сам термин "реляционное представление данных", впервые введенный Коддом [43], происходит от термина *relation*, понимаемом именно в смысле этого определения.

Т.к. любое множество можно рассматривать как декартовое произведение степени 1, то любое подмножество, как и любое множество, можно считать отношением степени 1. Это не очень интересный пример, свидетельствующий лишь о том, что термины "отношение степени 1" и "подмножество" являются синонимами. Нетривиальность понятия отношения проявляется, когда степень отношения больше 1. Ключевыми здесь являются два момента:

Во-первых, все элементы отношения есть однотипные кортежи. Однотипность кортежей позволяет считать их аналогами строк в простой таблице, т.е. в такой таблице, в которой все строки состоят из одинакового числа ячеек и в соответствующих ячейках содержатся одинаковые типы данных. Например, отношение, состоящее из трех следующих кортежей {(1, "Иванов", 1000), (2, "Петров", 2000), (3, "Сидоров", 3000)} можно считать таблицей, содержащей данные о сотрудниках и их зарплатах. Такая таблица будет иметь три строки и три колонки, причем в каждой колонке содержатся данные одного типа.

В противоположность этому рассмотрим множество  $\{(1), (1, 2), (1, 2, 3)\}$ , состоящее из разнотипных числовых кортежей. Это множество не является отношением ни в  $\mathbb{R}$ , ни в  $\mathbb{R}^3$ , ни в  $\mathbb{R}^3$ . Из кортежей, входящих в это множество нельзя составить простую таблицу. Правда, можно считать это множество отношением степени 1 на множестве всех

$$\Omega = R \cup R^2 \cup R^3 \cup ... = \bigcup_{k=1}^{\infty} R^k$$

возможных числовых кортежей всех возможных степеней такая трактовка ничего нового, по сравнению с понятием подмножества, не дает.

Во-вторых. За исключением крайнего случая, когда отношение есть само декартово произведение  $A_1 \times A_2 \times \ldots \times A_n$ , отношение включает в себя не все возможные кортежи из декартового произведения. Это значит, что для каждого отношения имеется критерий, позволяющий определить, какие кортежи входят в отношение, а какие - нет. Этот критерий, по существу, определяет для нас *смысл* (*семантику*) отношения.

Действительно, каждому отношению можно поставить в соответствие некоторое логическое выражение  $P(x_1,x_2,...,x_n)$ , зависящее от п параметров (п-местный предикат) и определяющее, будет ли кортеж  $(a_1,a_2,...,a_n)$  принадлежать отношению R. Это логическое выражение называют *предикатом отношения* R. Более точно, кортеж  $(a_1,a_2,...,a_n)$  принадлежит отношению R тогда и только тогда, когда предикат этого отношения  $P(a_1,a_2,...,a_n)$  принимает значение "истина". В свою очередь, каждый пместный предикат задает некоторое R0 принимает значение. Таким образом, существует взаимно однозначное соответствие между R1 принимает значения и R2 принимает значение. Таким образом, существует взаимно однозначное соответствие между R3 принимает значения и R4 предикатами.

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

# Примеры отношений

#### Бинарные отношения (отношения степени 2)

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

## Отношение эквивалентности

*Определение* 8. Отношение R на множестве  $A^2$  называется *отношением* эквивалентности, если оно обладает следующими свойствами:

- 1.  $(x,x) \in \mathbb{R}$  для всех  $x \in A$  (рефлексивность)
- 2. Если  $(x,y) \in R$ , то  $(y,x) \in R$  (симметричность)
- 3. Если  $(x,y) \in R_{\mathsf{H}}(y,z) \in R_{\mathsf{T}}(x,z) \in R_{\mathsf{T}}(x,z)$  (транзитивность)

Обычно отношение эквивалентности обозначают знаком " =" или "  $\approx$ " и говорят, что оно (отношение) задано на множестве A (а не на  $A^2$ ). Условия 1-3 в таких обозначениях выглядят более естественно:

- 1. x \* x для всех x ∈ A (рефлексивность)
- 2. Если  $x \approx y$ , то  $y \approx x$  (симметричность)
- 3. Если  $x \approx y$ и  $y \approx z$ , то  $x \approx z$  (транзитивность)

Легко доказывается, что если на множестве  $^{A}$  задано отношение эквивалентности, то множество  $^{A}$  разбивается на взаимно непересекающиеся подмножества, состоящие из эквивалентных друг другу элементов (классы эквивалентности).

<u>Пример 1</u>. Рассмотрим на множестве вещественных чисел  ${\bf R}$  отношение, заданное просто равенством чисел. Предикат такого отношения:

$$R(x,y) = \begin{cases} "ucmund, ecnu x = y \\ "noжcb", ecnux \neq y \end{cases}, или просто x = y$$

Условия 1-3, очевидно, выполняются, поэтому данное отношение является отношением эквивалентности. Каждый класс эквивалентности этого отношения состоит из одного числа.

<u>Пример 2</u>. Рассмотрим более сложное отношение эквивалентности. На множестве целых чисел **Z** зададим отношение "равенство по модулю п" следующим образом: два числа  $^a$  и  $^b$  *равны по модулю п*, если их остатки при делении на п равны. Например, по модулю 5 равны числа 2, 7, 12 и т.д.

Условия 1-3 легко проверяются, поэтому равенство по модулю является отношением эквивалентности. Предикат этого отношения имеет вид:

$$R(x,y) = \begin{cases} \text{"ucmund, ecnu } x = y \text{ modn} \\ \text{"nower", ecnu } x \neq y \text{ modn} \end{cases}$$

Классы эквивалентности этого отношения состоят из чисел, дающих при делении на n одинаковые остатки. Таких классов ровно n:

$$[0] = \{0, n, 2n, ...\}$$

$$[1] = \{1, n+1, 2n+1, ...\}$$
...
$$[n-1] = \{n-1, n+n-1, 2n+n-1, ...\}$$

#### Отношения порядка

*Определение 9.* Отношение R на множестве  $A^2$  называется *отношением порядка*, если оно обладает следующими свойствами:

- 1.  $(x,x) \in \mathbb{R}$  для всех  $x \in A$  (рефлексивность)
- 2. Если  $(x,y) \in R_{\mathsf{H}}(y,x) \in R$ , то x = y (антисимметричность)
- 3. Если  $(x,y) \in R_{\mathsf{H}}(y,z) \in R_{\mathsf{T}}(x,z) \in R_{\mathsf{T}}(x,z)$

Обычно отношение порядка обозначают знаком " $\leq$ ". Если для двух элементов x и y выполняется  $x \leq y$ , то говорят, что x "предшествует" y. Как и для отношения эквивалентности, условия 1-3 в таких обозначениях выглядят более естественно:

- 1.  $x \le x$  для всех  $x \in A$  (рефлексивность)
- 2. Если  $x \le y_{\rm H} \ y \le x_{\rm TO} \ x = y_{\rm (антисимметричность)}$

3. Если  $x \le y_{\mathsf{H}} \quad y \le z$ , то  $x \le z$  (транзитивность)

<u>Пример 3</u>. Простым примером отношения порядка является отношение, задаваемое обычным неравенством " $\leq$ " на множестве вещественных чисел  $\mathbf R$  . Заметим, что *для любых чисел* x и y выполняется либо  $x \leq y$ , либо  $y \leq x$ , т.е. *любые* два числа сравнимы между собой. Такие отношения называются *отношениями полного порядка*.

Предикат данного отношения есть просто утверждение  $x \le y$ .

<u>Пример 4</u>. Рассмотрим на множестве  $^{A}$  всех сотрудников некоторого предприятия отношение, задаваемое следующим образом: сотрудник  $^{x}$  предшествует сотруднику  $^{y}$  тогда и только тогда, когда выполняется одно из условий:

- x = y
- x является начальником (не обязательно непосредственным) y

Назовем такое отношение "быть начальником". Легко проверить, что отношение "быть начальником" является отношением порядка. Заметим, что в отличие от предыдущего примера, *существуют такие пары* сотрудников x и y, для которых не выполняется ни  $x \le y$ , ни  $y \le x$  (например, если x и y являются сослуживцами). Такие отношения, в которых есть несравнимые между собой элементы, называют *отношениями частичного порядка*.

## Функциональное отношение

*Определение 10.* Отношение R на декартовом произведении двух множеств  $A_1 \times A_2$  называется **функциональным отношением**, если оно обладает следующим свойством:

1. Если 
$$(x,y) \in R_{\mathsf{H}}(x,z) \in R$$
, то  $y = z$  (однозначность функции).

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

Предикат функционального отношения есть просто выражение функциональной зависимости y = f(x)

## Еще пример бинарного отношения

<u>Пример 5</u>. Пусть множество  $^{A}$  есть следующее множество молодых людей: {Вовочка, Петя, Маша, Лена}, причем известны следующие факты:

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

Информацию о взаимоотношения данных молодых людей можно описать бинарным отношением "любить", заданном на множестве  $A^2$ . Это отношение можно описать несколькими способами.

Способ 1. Перечисление фактов в виде произвольного текста (как это сделано выше).

Способ 2. В виде графа взаимоотношений:

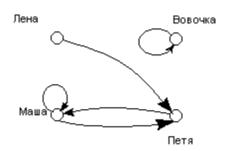


Рисунок 1 Граф взаимоотношений

Способ 3. При помощи матрицы взаимоотношений:

Кого	Вовочка	Петя	Маша	Лена
Кто				
Вовочка	Любит			
Петя			Любит	
Маша		Любит	Любит	
Лена		Любит		

Таблица 1. Матрица взаимоотношений

Способ 4. При помощи таблицы фактов:

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

Таблица 2 Таблица фактов

С точки зрения реляционных баз данных наиболее предпочтительным является четвертый способ, т.к. он допускает наиболее удобный способ хранения и манипулирования информацией. Действительно, перечисление фактов как текстовая форма хранения

информации уместна для литературного произведения, но с трудом поддается алгоритмической обработке. Изображение в виде графа наглядно, и его удобно использовать как конечную форму представления информации для пользователя, но хранить данные в графическом виде неудобно. Матрица взаимоотношений уже больше соответствует требованиям информационной системы. Матрица удобна в обработке и компактно хранится. Но одно небольшое изменение, например, появился еще Вася и влюбился в несчастную Лену, требует перестройки всей матрицы, а именно, добавления и колонок, и столбиов. Таблица фактов свободна от всех этих недостатков - при добавлении новых действующих лиц просто добавляются новые строки.

Что касается предиката данного отношения, то он имеет следующий вид (дизъюнктивная нормальная форма):

```
R(x,y) = \{(x = "Вовочка" AND y = "Вовочка") OR (x = "Петя" AND y = "Маша") OR (x = "Маша" AND y = "Петя") OR (x = "Лена" AND y = "Петя") <math>\{(x = "Петя")\}
```

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

Замечание. Большая часть мировой литературы существует и имеет смысл лишь постольку, поскольку бинарное отношение "любить" не является отношением эквивалентности. В частности, по этой причине человечество не разбивается на классы эквивалентности взаимно любящих особей. Изучением характеристик данного отношения и соответствующего ему предиката занималось (и продолжает заниматься) большое количество экспертов, таких как Толстой Л.Н., Шекспир В. и др.

#### n-арные отношения (отношения степени n)

В математике п-арные отношения рассматриваются относительно редко, в отличие от баз данных, где наиболее важными являются именно отношения, заданные на декартовом произведении более чем двух множеств.

<u>Пример 6</u>. В некотором университете на математическом факультете учатся студенты Иванов, Петров и Сидоров. Лекции им читают преподаватели Пушников, Цыганов и Шарипов, причем известны следующие факты:

- 1. Пушников читает лекции по алгебре и базам данных, соответственно, 40 и 80 часов в семестр.
- 2. Цыганов читает лекции по геометрии, 50 часов в семестр.
- 3. Шарипов читает лекции по алгебре и геометрии, соответственно, 40 и 50 часов в семестр.
- 4. Студент Иванов посещает лекции по алгебре у Шарипова и по базам данных у Пушникова.
- 5. Студент Петров посещает лекции по алгебре у Пушникова и по геометрии у Цыганова.
- 6. Студент Сидоров посещает лекции по геометрии у Цыганова и по базам данных у Пушникова.

Для того чтобы формально описать данную ситуацию (например, в целях разработки информационной системы, учитывающей данные о ходе учебного процесса), введем три множества:

- Множество преподавателей  $A = \{\Pi y \text{шников}, \text{Цыганов}, \text{Шарипов}\}.$
- Множество предметов  $B = \{ Aлгебра, Геометрия, Базы данных \}.$
- Множество студентов  $C = \{ \text{Иванов}, \text{Петров}, \text{Сидоров} \}.$

Имеющиеся факты можно разделить на две группы. 1 группа (факты 1-3) - факты о преподавателях, 2 группа (факты 4-6) - факты о студентах.

Для того чтобы отразить факты 1-3 (характеризующие преподавателей и читаемые ими лекции), введем отношение  $R_1$  на декартовом произведении  $A \times B \times Q$ , где Q- множество рациональных чисел. А именно, упорядоченная тройка  $(x,y,n) \in R_1$  тогда и только тогда, когда преподаватель x читает лекции по предмету y в количестве y часов в семестр. Назовем такое отношение "Читает лекции по...". Множество кортежей, образующих отношение x0 удобно представить в виде таблицы:

А (Преподаватель)	В (Предмет)	Q (Количество часов)	
Пушников	Алгебра	40	
Пушников	Базы данных	80	
Цыганов	Геометрия	50	
Шарипов	Алгебра	40	
Шарипов	Геометрия	50	

Таблица 3 Отношение "Читает лекции по..."

Для того чтобы отразить факты 4-6 (характеризующие посещение студентами лекций), введем отношение  $R_2$  на декартовом произведении  $C \times B \times A$ . Упорядоченная тройка  $(z,y,x) \in R_2$  тогда и только тогда, когда студент z посещает лекции по предмету y у преподавателя x. Назовем это отношение "Посещать лекции". Его также представим в виде таблицы:

С (студент)	В (предмет)	А (Преподаватель)	
Иванов	Алгебра	Шарипов	
Иванов	Базы данных	Пушников	
Петров	Алгебра	Пушников	
Петров	Геометрия	Цыганов	
Сидоров	Геометрия	Цыганов	
Сидоров Базы данных		Пушников	

Таблица 4 Отношение "Посещать лекции"

Рассмотрим отношение  $R_2$  подробнее. Оно задано на декартовом произведении  $\Omega = C \times B \times A$ . Это произведение, содержащее 3\*3\*3=27 кортежей, можно назвать "Студенты-Лекции-Преподаватели". Множество  $\Omega$  представляет собой совокупность всех возможных вариантов посещения студентами лекций. Отношение же  $R_2$  показывает мекущее состояние учебного процесса. Очевидно, что отношение  $R_2$  является изменяемым во времени отношением.

Итак, факты о ходе учебного процесса удалось отразить в виде двух отношений третьей степени (3-арных), а сами отношения изобразить в виде таблиц с тремя колонками.

Удобство использования табличной формы для задания отношения определяется в данном случае следующими факторами:

- 1. Все используемые множества конечны.
- 2. При добавлении или удалении студентов, предметов, преподавателей просто добавляются или удаляются соответствующие строки в таблице.

Нас сейчас не интересует вопрос, хороши ли полученные отношения. Заметим пока только, что, как показывают следующие замечания, не любую строку можно добавить в таблицу "Посещать лекции".

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

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

## Транзитивное замыкание отношений

Введем понятие *танзитивного замыкания*, связанное с бинарными отношениями, которое понадобится в дальнейшем.

Определение 11. Пусть отношение R задано на декартовом квадрате  $A^2$  некоторого множества A. **Транзитивным замыканием** отношения R называется новое отношение  $\overline{R}$ , состоящее из кортежей (x,y), для которых выполняется:

- либо кортеж (x,y) ∈ R,
- либо найдется конечная последовательность элементов  $z_1, z_2, ..., z_n \in A$ , такая, что все кортежи  $(x, z_1), (z_1, z_2), ..., (z_n, y)$  принадлежат отношению R.

Очевидно, что  $R \subseteq \overline{R}$ .

<u>Пример 7</u>. Пусть множество  $^{A}$  представляет собой следующее множество деталей и конструкций:

 $A = \{$ Болт, Гайка, Двигатель, Автомобиль, Колесо, Ось $\}$ 

причем некоторые из деталей и конструкций могут использоваться при сборке других конструкций. Взаимосвязь деталей описывается отношением R ("непосредственно используется в") и состоит из следующих кортежей:

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

Таблица 5 Отношение R

Транзитивное замыкание  $\overline{R}$  состоит из кортежей (добавленные кортежи помечены серым цветом):

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

Таблица 6 Транзитивное замыкание отношения R

Очевидный смысл замыкания  $\overline{R}$  состоит в описании включения деталей друг в друга не только непосредственно, а через использование их в промежуточных деталях, например, болт используется в автомобиле, т.к. он используется в двигатель используется в автомобиле.

#### Выводы

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

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

**Отношение**- это подмножество декартового произведения множеств. Отношения состоят из *однотипных* кортежей. Каждое отношение имеет *предикат отношения* и каждый пместный предикат задает n-apное отношение.

Отношение является математическим аналогом понятия "таблица".

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

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

## Глава 2. Базовые понятия реляционной модели данных

# Общая характеристика реляционной модели данных

Основы реляционной модели данных были впервые изложены в статье Е.Кодда [43] в 1970 г. Эта работа послужила стимулом для большого количества статей и книг, в которых реляционная модель получила дальнейшее развитие. Наиболее распространенная трактовка реляционной модели данных принадлежит К.Дейту [11]. Согласно Дейту, реляционная модель состоит из трех частей:

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

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

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

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

В данной главе рассматривается структурная часть реляционной модели.

#### Типы данных

Любые данные, используемые в программировании, имеют свои типы данных.

Важно! Реляционная модель требует, чтобы типы используемых данных были простыми.

Для уточнения этого утверждения рассмотрим, какие вообще типы данных обычно рассматриваются в программировании. Как правило, типы данных делятся на три группы:

- Простые типы данных.
- Структурированные типы данных.
- Ссылочные типы данных.

## Простые типы данных

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

- Логический.
- Строковый.
- Численный.

Различные языки программирования могут расширять и уточнять этот список, добавляя такие типы как:

- Целый.
- Вещественный.
- Дата.
- Время.
- Денежный.
- Перечислимый.
- Интервальный.
- И т.д....

Конечно, понятие атомарности довольно относительно. Так, строковый тип данных можно рассматривать как одномерный массив символов, а целый тип данных - как набор битов. Важно лишь то, что при переходе на такой низкий уровень теряется *семантика* (*смысл*) *данных*. Если строку, выражающую, например, фамилию сотрудника, разложить в массив символов, то при этом теряется смысл такой строки как единого целого.

#### Структурированные типы данных

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

- Массивы
- Записи (Структуры)

С математической точки зрения массив представляет собой функцию с конечной областью определения. Например, рассмотрим конечное множество натуральных чисел

$$A = \{1, 2, ..., n\}$$

называемое множеством индексов. Отображение

$$F:A\to \mathbb{R}$$

из множества  $^{A}$  во множество вещественных чисел  $^{R}$  задает одномерный вещественный массив. Значение этой функции для некоторого значения индекса  $^{i}$  называется элементом массива, соответствующим  $^{i}$ . Аналогично можно задавать многомерные массивы.

Запись (или структура) представляет собой кортеж из некоторого декартового произведения множеств. Действительно, запись представляет собой именованный упорядоченный набор элементов  $r_i$ , каждый из которых принадлежит типу  $T_i$ . Таким образом, запись  $r = (r_1, r_2, ..., r_n)$  есть элемент множества  $T = T_1 \times T_2 \times ... \times T_n$ . Объявляя новые типы записей на основе уже имеющихся типов, пользователь может конструировать сколь угодно сложные типы данных.

Общим для структурированных типов данных является то, что они *имеют внутреннюю* структуру, используемую на том же уровне абстракции, что и сами типы данных.

Поясним это следующим образом. При работе с массивами или записями можно манипулировать массивом или записью и как с единым целым (создавать, удалять, копировать целые массивы или записи), так и поэлементно. Для структурированных типов данных есть специальные функции - конструкторы типов, позволяющие создавать массивы или записи из элементов более простых типов.

Работая же с простыми типами данных, например с числовыми, мы манипулируем ими как неделимыми целыми объектами. Чтобы "увидеть", что числовой тип данных на самом деле сложен (является набором битов), нужно перейти на более низкий уровень абстракции. На уровне программного кода это будет выглядеть как ассемблерные вставки в код на языке высокого уровня или использование специальных побитных операций.

# Ссылочные типы данных

*Ссылочный тип данных* (*указатели*) предназначен для обеспечения возможности указания на другие данные. Указатели характерны для языков процедурного типа, в которых есть понятие области памяти для хранения данных. Ссылочный тип данных предназначен для обработки сложных изменяющихся структур, например деревьев, графов, рекурсивных структур.

#### Типы данных, используемые в реляционной модели

Собственно, для реляционной модели данных тип используемых данных не важен. Требование, чтобы тип данных был *простым*, нужно понимать так, что *в реляционных операциях не должна учитываться внутренняя структура данных*. Конечно, должны

быть описаны действия, которые можно производить с данными как с единым целым, например, данные числового типа можно складывать, для строк возможна операция конкатенации и т.д.

С этой точки зрения, если рассматривать массив, например, как единое целое и не использовать поэлементных операций, то массив можно считать простым типом данных. Более того, можно создать свой, сколь угодно сложных тип данных, описать возможные действия с этим типом данных, и, если в операциях не требуется знание внутренней структуры данных, то такой тип данных также будет простым с точки зрения реляционной теории. Например, можно создать новый тип - комплексные числа как запись вида z = (x,y), где  $x \in \mathbb{R}, y \in \mathbb{R}$ . Можно описать функции сложения, умножения, вычитания и деления, и все действия с компонентами x и y выполнять только внутри этих операций. Тогда, если в действиях с этим типом использовать только описанные операции, то внутренняя структура не играет роли, и тип данных извне выглядит как атомарный.

Именно так в некоторых пост-реляционных СУБД реализована работа со сколь угодно сложными типами данных, создаваемых пользователями.

## Домены

В реляционной модели данных с понятием тип данных тесно связано понятие домена, которое можно считать уточнением типа данных.

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

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

Например, домен D, имеющий смысл "возраст сотрудника" можно описать как следующее подмножество множества натуральных чисел:

$$D = \{n \in \mathbb{N} : n \ge 18 \text{ and } n \le 60\}$$

Если тип данных можно считать множеством всех возможных значений данного типа, то домен напоминает подмножество в этом множестве.

Отличие домена от понятия подмножества состоит именно в том, что *домен отражает семантику*, определенную предметной областью. Может быть несколько доменов, совпадающих как подмножества, но несущие различный смысл. Например, домены "Вес детали" и "Имеющееся количество" можно одинаково описать как множество неотрицательных целых чисел, но смысл этих доменов будет различным, и это будут *различные* домены.

Основное значение доменов состоит в том, что *домены ограничивают сравнения*. Некорректно, с логической точки зрения, сравнивать значения из различных доменов,

даже если они имеют одинаковый тип. В этом проявляется смысловое ограничение доменов. Синтаксически правильный запрос "выдать список всех деталей, у которых вес детали больше имеющегося количества" не соответствует смыслу понятий "количество" и "вес".

<u>Замечание</u>. Понятие домена помогает правильно *моделировать* предметную область. При работе с реальной системой в принципе возможна ситуация когда требуется ответить на запрос, приведенный выше. Система даст ответ, но, вероятно, он будет бессмысленным.

<u>Замечание</u>. Не все домены обладают логическим условием, ограничивающим возможные значения домена. В таком случае множество возможных значений домена совпадает с множеством возможных значений типа данных.

Замечание. Не всегда очевидно, как задать логическое условие, ограничивающее возможные значения домена. Я буду благодарен тому, кто приведет мне условие на строковый тип данных, задающий домен "Фамилия сотрудника". Ясно, что строки, являющиеся фамилиями не должны начинаться с цифр, служебных символов, с мягкого знака и т.д. Но вот является ли допустимой фамилия "Ггггггыыыыы"? Почему бы нет? Очевидно, нет! А может кто-то назло так себя назовет. Трудности такого рода возникают потому, что смысл реальных явлений далеко не всегда можно формально описать. Просто мы, как все люди, интуитивно понимаем, что такое фамилия, но никто не может дать такое формальное определение, которое отличало бы фамилии от строк, фамилиями не являющимися. Выход из этой ситуации простой - положиться на разум сотрудника, вводящего фамилии в компьютер.

# Отношения, атрибуты, кортежи отношения

#### Определения и примеры

Фундаментальным понятием реляционной модели данных является понятие *отношения*. В определении понятия отношения будем следовать книге К. Дейта [11].

Определение 1. Атрибут отношения есть пара вида <Имя атрибута: Имя домена>.

Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов.

*Определение 2. Отношение* R, определенное на множестве доменов  $D_1, D_2, \dots, D_n$  (не обязательно различных), содержит две части: заголовок и тело.

Заголовок отношения содержит фиксированное количество атрибутов отношения:

$$(\mathrel{\i} A_1\mathrel{:} D_1\mathrel{\gt},\mathrel{\i} A_2\mathrel{:} D_2\mathrel{\gt},...,\mathrel{\i} A_n\mathrel{:} D_n\mathrel{\gt})$$

**Тело отношения** содержит множество кортежей отношения. Каждый кортеж отношения представляет собой множество пар вида <Имя\_атрибута: Значение\_атрибута>:

$$(\langle A_1 : Val_1 \rangle, \langle A_2 : Val_2 \rangle, ..., \langle A_n : Val_n \rangle)$$

таких что значение  ${}^{Val_i}$ атрибута  ${}^{A_i}$ принадлежит домену  ${}^{D_i}$ 

Отношение обычно записывается в виде:

$$R(\mathrel{<} A_1\mathrel{:} D_1\mathrel{>},\mathrel{<} A_2\mathrel{:} D_2\mathrel{>},\ldots,\mathrel{<} A_{\mathbf{n}}\mathrel{:} D_{\mathbf{n}}\mathrel{>})$$

или короче

$$R(A_1, A_2, \dots, A_n)$$

или просто

 $R_{\perp}$ 

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

Мощность множества кортежей отношения называют мощностью отношения.

Возвращаясь к математическому понятию отношения, введенному в предыдущей главе, можно сделать следующие выводы:

<u>Вывод 1</u>. Заголовок отношения описывает декартово произведение доменов, на котором задано отношение. Заголовок статичен, он не меняется во время работы с базой данных. Если в отношении изменены, добавлены или удалены атрибуты, то в результате получим уже *другое* отношение (пусть даже с прежним именем).

<u>Вывод 2</u>. Тело отношения представляет собой набор кортежей, т.е. подмножество декартового произведения доменов. Таким образом, тело отношения собственно и является отношением в математическом смысле слова. Тело отношения может изменяться во время работы с базой данных - кортежи могут изменяться, добавляться и удаляться.

<u>Пример 1</u>. Рассмотрим отношение "Сотрудники" заданное на доменах "Номер\_сотрудника", "Фамилия", "Зарплата", "Номер\_отдела". Т.к. все домены различны, то имена атрибутов отношения удобно назвать так же, как и соответствующие домены. Заголовок отношения имеет вид:

Сотрудники (Номер\_сотрудника, Фамилия, Зарплата, Номер\_отдела)

Пусть в данный момент отношение содержит три кортежа:

- (1,Иванов, 1000, 1)
- (2, Петров, 2000, 2)
- (3, Сидоров, 3000, 1)

такое отношение естественным образом представляется в виде таблицы:

Номер_сотрудника	Фамилия	Запплата	Номер отделя	,
Tromep_corp_gmmu	- CONTINUE IN	Jupinuu	ттошер_отден	٠

1	Иванов	1000	1
2	Петров	2000	2
3	Сидоров	3000	1

Таблица 1 Отношение "Сотрудники"

Определение 3. Реляционной базой данных называется набор отношений.

*Определение 4. Схемой реляционной базы* данных называется набор заголовков отношений, входящих в базу данных.

Хотя любое отношение можно изобразить в виде таблицы, нужно четко понимать, что отношения не являются таблицами. Это близкие, но не совпадающие понятия. Различия между отношениями и таблицами будут рассмотрены ниже.

Термины, которыми оперирует реляционная модель данных, имеют соответствующие "табличные" синонимы:

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

#### Свойства отношений

Свойства отношений непосредственно следуют из приведенного выше определения отношения. В этих свойствах в основном и состоят различия между отношениями и таблицами.

- 1. *В отношении нет одинаковых кортежей*. Действительно, тело отношения есть *множество* кортежей и, как всякое множество, не может содержать неразличимые элементы (см. понятие множества в гл.1.). Таблицы в отличие от отношений могут содержать одинаковые строки.
- 2. Кортежи не упорядочены (сверху вниз). Действительно, несмотря на то, что мы изобразили отношение "Сотрудники" в виде таблицы, нельзя сказать, что сотрудник Иванов "предшествует" сотруднику Петрову. Причина та же тело отношения есть множество, а множество не упорядочено. Это вторая причина, по которой нельзя отождествить отношения и таблицы строки в таблицах

- упорядочены. Одно и то же отношение может быть *изображено* разными таблицами, в которых *строки идут в различном порядке*.
- 3. Атрибуты не упорядочены (слева направо). Т.к. каждый атрибут имеет уникальное имя в пределах отношения, то порядок атрибутов не имеет значения. Это свойство несколько отличает отношение от математического определения отношения (см. гл.1 компоненты кортежей там упорядочены). Это также третья причина, по которой нельзя отождествить отношения и таблицы столбцы в таблице упорядочены. Одно и то же отношение может быть изображено разными таблицами, в которых столбцы идут в различном порядке.
- 4. *Все значения атрибутов атомарны*. Это следует из того, что лежащие в их основе атрибуты имеют атомарные значения. Это четвертое отличие отношений от таблиц в ячейки таблиц можно поместить что угодно массивы, структуры, и даже другие таблицы.

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

<u>Замечание</u>. Каждое отношение можно считать *классом эквивалентности таблиц*, для которых выполняются следующие условия:

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

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

#### Первая нормальная форма

Труднее всего дать определение вещей, которые всем понятны. Если давать не строгое, описательное определение, то всегда остается возможность неправильной его трактовки. Если дать строгое формальное определение, то оно, как правило, или тривиально, или слишком громоздко. Именно такая ситуация с определением отношения в *Первой Нормальной Форме* (*1НФ*). Совсем не говорить об этом нельзя, т.к. на основе 1НФ строятся более высокие нормальные формы, которые рассматриваются далее в гл. 6 и 7. Дать определение 1НФ сложно ввиду его тривиальности. Поэтому, дадим просто несколько объяснений.

*Объяснение 1*. Говорят, что отношение R находится в 1Н $\Phi$ , если оно удовлетворяет определению 2.

Это, собственно, тавтология, ведь из определения 2 следует, что других отношений не бывает. Действительно, определение 2 описывает, что является отношением, а что - нет, следовательно, отношений в непервой нормальной форме просто нет.

*Объяснение* 2. Говорят, что отношение R находится в 1НФ, если его атрибуты содержат только скалярные (атомарные) значения.

Опять же, определение 2 опирается на понятие домена, а домены определены на простых типах данных.

Непервую нормальную форму можно получить, если допустить, что атрибуты отношения могут быть определены на сложных типах данных - массивах, структурах, или даже на других отношениях. Легко себе представить таблицу, у которой в некоторых ячейках содержатся массивы, в других ячейках - определенные пользователями сложные структуры, а в третьих ячейках - целые реляционные таблицы, которые в свою очередь могут содержать такие же сложные объекты. Именно такие возможности предоставляются некоторыми современными пост-реляционными и объектными СУБД.

Требование, что отношения должны содержать только данные простых типов, объясняет, почему отношения иногда называют *плоскими таблицами* (*plain table*). Действительно, таблицы, задающие отношения двумерны. Одно измерение задается списком столбцов, второе измерение задается списком строк. Пара координат (Номер строки, Номер столбца) однозначно идентифицирует ячейку таблицы и содержащееся в ней значение. Если же допустить, что в ячейке таблицы могут содержаться данные сложных типов (массивы, структуры, другие таблицы), то такая таблица будет уже не плоской. Например, если в ячейке таблицы содержится массив, то для обращения к элементу массива нужно знать *три* параметра (Номер строки, Номер столбца, номер элемента в массиве).

Таким образом появляется третье объяснение Первой Нормальной Формы:

*Объяснение 3.* Отношение R находится в 1Н $\Phi$ , если оно является плоской таблицей.

Мы сознательно ограничиваемся рассмотрением только классической реляционной теории, в которой все отношения имеют только атомарные атрибуты и заведомо находятся в 1НФ.

#### Выводы

Реляционная модель данных состоит из трех частей:

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

В классической реляционной модели используются только *простые* (*атомарные*) *типы данных*. Простые типы данных не обладают внутренней структурой.

**Домены** - это типы данных, имеющие некоторый смысл (семантику). Домены ограничивают сравнения - некорректно, хотя и возможно, сравнивать значения из различных доменов.

**Отношение** состоит из двух частей - *заголовка отношения* и *тела отношения*. Заголовок отношения - это аналог заголовка таблицы. Заголовок отношения состоит из атрибутов. Количество атрибутов называется *степенью отношения*. Тело отношения - это аналог тела таблицы. Тело отношения состоит из *кортежей*. Кортеж отношения

является аналогом строки таблицы. Количество кортежей отношения называется **мощностью отношения**.

Отношение обладает следующими свойствами:

- В отношении нет одинаковых кортежей.
- Кортежи не упорядочены (сверху вниз).
- Атрибуты не упорядочены (слева направо).
- Все значения атрибутов атомарны.

Реляционной базой данных называется набор отношений.

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

Отношение находится в *Первой Нормальной Форме* ( $1H\Phi$ ), если оно содержит только скалярные (атомарные) значения.

# Глава 3. Целостность реляционных данных

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

- Целостность сущностей.
- Целостность внешних ключей.

Прежде, чем говорить о целостности сущностей, опишем использование null-значений в реляционных базах данных.

#### Null-значения

Основное назначение баз данных состоит в том, чтобы хранить и предоставлять информацию о реальном мире. Для представления этой информации в базе данных используются привычные для программистов типы данных - строковые, численные, логические и т.п. Однако в реальном мире часто встречается ситуация, когда данные неизвестны или не полны. Например, место жительства или дата рождения человека могут быть неизвестны (база данных разыскиваемых преступников). Если вместо неизвестного адреса уместно было бы вводить пустую строку, то что вводить вместо неизвестной даты? Ответ - пустую дату - не вполне удовлетворителен, т.к. простейший запрос "выдать список людей в порядке возрастания дат рождения" даст заведомо неправильных ответ.

Для того чтобы обойти проблему неполных или неизвестных данных, в базах данных могут использоваться типы данных, пополненные так называемым *null-значением*. Null-значение - это, собственно, не значение, а некий маркер, показывающий, что значение неизвестно.

Таким образом, в ситуации, когда возможно появление неизвестных или неполных данных, разработчик имеет на выбор два варианта.

Первый вариант состоит в том, чтобы ограничиться использованием обычных типов данных и не использовать null-значения, а вместо неизвестных данных вводить либо нулевые значения, либо значения специального вида - например, договориться, что строка

"АДРЕС НЕИЗВЕСТЕН" и есть те данные, которые нужно вводить вместо неизвестного адреса. В любом случае на пользователя (или на разработчика) ложится ответственность на правильную трактовку таких данных. В частности, может потребоваться написание специального программного кода, который в нужных случаях "вылавливал" бы такие данные. Проблемы, возникающие при этом очевидны - не все данные становятся равноправны, требуется дополнительный программный код, "отслеживающий" эту неравноправность, в результате чего усложняется разработка и сопровождение приложений.

Второй вариант состоит в использовании null-значений вместо неизвестных данных. За кажущейся естественностью такого подхода скрываются менее очевидные и более глубокие проблемы. Наиболее бросающейся в глаза проблемой является необходимость использования трехзначной логики при оперировании с данными, которые могут содержать null-значения. В этом случае при неаккуратном формулировании запросов, даже самые естественные запросы могут давать неправильные ответы. Есть более фундаментальные проблемы, связанные с теоретическим обоснованием корректности введения null-значений, например, непонятно вообще, входят ли null-значения в домены или нет.

Подробное обсуждение проблем использования null-значений выходит за пределы данной работы. Можно только сказать о том, что этот вопрос в теории реляционных баз данных окончательно не решен. Основоположник реляционного подхода Кодд считал null-значения неотъемлемой частью реляционной модели. К.Дейт, один из крупнейших теоретиков реляционной модели выступает категорически против null-значений (подробное обсуждение проблем, возникающих при использовании null-значений приведено в книге [11].

Практически все реализации современных реляционных СУБД позволяют использовать null-значения, несмотря на их недостаточную теоретическую обоснованность. Такую ситуацию можно сравнить с ситуацией, сложившейся в начале века с теорией множеств. Почти сразу после создания Кантором теории множеств, в ней были обнаружены внутренние противоречия (антиномии). Были разработаны более строгие теории, позволяющие избежать этих противоречий (конструктивная теория множеств). Однако в реальной работе большинство математиков пользуется классической теорией множеств, т.к. более строгие теории более ограничены и негибки в применении именно в силу своей большей строгости.

Мнение автора (очень скромное по сравнению с мнением корифеев реляционной теории) состоит в том, что желательно избегать null-значений. Тем не менее, приведем здесь описание трехзначной логики, необходимой для работы с null-значениями.

## Трехзначная логика (3VL)

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

При сравнении выражений, содержащих null-значения, результат также может быть неизвестен, например, значение истинности для выражения a = b есть null, если один или оба аргумента есть null. Таким образом, определение истинности логических выражений базируется на *мрехзначной логике* (*three-valued logic*, *3VL*), в которой кроме значений T -

ИСТИНА и F - ЛОЖЬ, введено значение U - НЕИЗВЕСТНО. Логическое значение U - это то же самое, что и null-значение. Трехзначная логика базируется на следующих таблицах истинности:

AND	F	T	U
F	F	F	F
T	F	T	U
U	F	U	U

Таблина 1 Таблина истинности AND

OR	F	T	U
F	F	T	U
T	T	T	Т
U	U	Т	U

Таблица 2 Таблица истинности OR

NOT	
F	T
T	F
U	U

Таблица 3 Таблица истинности NOT

Имеется несколько парадоксальных следствий применения трехзначной логики.

<u>Парадокс 1.</u> Null-значение не равно самому себе. Действительно, выражение null = null дает значение не ИСТИНА, а НЕИЗВЕСТНО. Значит выражение x = x не обязательно ИСТИНА!

<u>Парадокс 2.</u> Неверно также, что null-значение не равно самому себе! Действительно, выражение null ≠ null также принимает значение не ИСТИНА, а НЕИЗВЕСТНО! Значит также, что и выражение  $x \neq x$ тоже не обязательно ЛОЖЬ!

<u>Парадокс 3.</u> a or not(a) не обязательно ИСТИНА. Значит, в трехзначной логике не работает принцип исключенного третьего (любое высказывание либо истинно, либо ложно).

Таких парадоксов можно построить сколько угодно. Конечно, это на самом деле не парадоксы, а просто следствия из аксиом трехзначной логики.

#### Потенциальные ключи

По определению, тело отношения есть *множество* кортежей, поэтому отношения не могут содержать одинаковые кортежи. Это значит, что каждый кортеж должен обладать

свойством уникальности. На самом деле, свойством уникальности в пределах отношения могут обладать отдельные атрибуты кортежей или группы атрибутов. Такие уникальные атрибуты удобно использовать для идентификации кортежей.

*Определение 1.* Пусть дано отношение R. Подмножество атрибутов K отношения R будем называть *потенциальным ключом*, если K обладает следующими свойствами:

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

Любое отношение имеет по крайней мере один потенциальный ключ. Действительно, если никакой атрибут или группа атрибутов не являются потенциальным ключом, то, в силу уникальности кортежей, все атрибуты вместе образуют потенциальный ключ.

Потенциальный ключ, состоящий из одного атрибута, называется *простым*. Потенциальный ключ, состоящий из нескольких атрибутов, называется *составным*.

Отношение может иметь несколько потенциальных ключей. Традиционно, один из потенциальных ключей объявляется *первичным*, а остальные - *альтернативными*. Различия между первичным и альтернативными ключами могут быть важны в конкретной реализации реляционной СУБД, но с точки зрения реляционной модели данных, нет оснований выделять таким образом один из потенциальных ключей.

<u>Замечание</u>. Понятие потенциального ключа является *семантическим* понятием и отражает некоторый смысл (трактовку) понятий из конкретной предметной области. Для того чтобы проиллюстрировать этот факт рассмотрим следующее отношение "Сотрудники":

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

Таблица 4 Отношение "Сотрудники"

При первом взгляде на таблицу, изображающую это отношение, может показаться, что в таблице имеется три потенциальных ключа - в каждой колонке таблицы содержатся уникальные данные. Однако среди сотрудников могут быть однофамильцы и сотрудники с одинаковой зарплатой. Табельный же номер по сути свой уникален для каждого сотрудника. Какие же соображения привели нас к пониманию того, что в данном отношении только один потенциальный ключ - "Табельный номер"? Именно понимание смысла данных, содержащихся в отношении.

Попробуем представить это отношение в другом виде, изменив наименования атрибутов:

A	В	C
1	Иванов	1000

2	Петров	2000
3	Сидоров	3000

Предъявим кому-нибудь эту таблицу и не сообщим смысл наименований атрибутов. Очевидно, что невозможно судить, не понимая смысла данных, может или не может в этом отношении появиться, например, кортеж (1, Петров, 3000). Если бы, кстати, такой кортеж появился (что, на первый взгляд, вполне возможно, т.к. не нарушается уникальность кортежей), то мы точно смогли бы сказать, что не является альтернативным ключом - ни один из атрибутов по отдельности. Но мы не сможем сказать, что же является первичным ключом.

<u>Замечание</u>. Потенциальные ключи служат *средством идентификации* объектов предметной области, данные о которых хранятся в отношении. Объекты предметной области должны быть различимы.

<u>Замечание</u>. Потенциальные ключи служат единственным *средством адресации на уровне кортежей* в отношении. Точно указать какой-нибудь кортеж можно только зная значение его потенциального ключа.

## Целостность сущностей

Т.к. потенциальные ключи фактически служат идентификаторами объектов предметной области (т.е. предназначены для *различения* объектов), то значения этих идентификаторов не могут содержать неизвестные значения. Действительно, если бы идентификаторы могли содержать null-значения, то мы не могли бы дать ответ "да" или "нет" на вопрос, совпадают или нет два идентификатора.

Это определяет следующее правило целостности сущностей:

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

#### Внешние ключи

Различные объекты предметной области, информация о которых хранится в базе данных, всегда взаимосвязаны друг с другом. Например, накладная на поставку товара *содержит* список товаров с количествами и ценами, сотрудник предприятия *имеет* детей, *числится* в подразделении и т.д. Термины "содержит", "имеет", "числится" отражают взаимосвязи между понятиями "накладная" и "список товаров", "сотрудник" и "дети", "сотрудник" и "подразделение". Такие взаимосвязи отражаются в реляционных базах данных при помощи *внешних ключей*, связывающих несколько отношений.

Рассмотрим пример с поставщиками и поставками деталей. Предположим, что нам требуется хранить информацию о наименовании поставщиков, наименовании и количестве поставляемых ими деталей, причем каждый поставщик может поставлять несколько деталей и каждая деталь может поставляться несколькими поставщиками. Можно предложить хранить данные в следующем отношении:

Номер	Наименование	Номер	Наименование	Поставляемое
поставщика	поставщика	детали	детали	количество

1	Иванов	1	Болт	100
1	Иванов	2	Гайка	200
1	Иванов	3	Винт	300
2	Петров	1	Болт	150
2	Петров	2	Гайка	250
3	Сидоров	3	Винт	1000

Таблица 5 Отношение "Поставщики и поставляемые детали"

Потенциальным ключом этого отношения может выступать пара атрибутов {"Номер поставщика", "Номер детали"} - в таблице они выделены курсивом.

Приведенный способ хранения данных обладает рядом недостатков.

Что произойдет, если изменилось наименование поставщика? Т.к. наименование поставщика повторяется во многих кортежах отношения, то это наименование нужно *одновременно* изменить *во всех* кортежах, где оно встречается, иначе данные станут противоречивыми. То же самое с наименованиями деталей. Значит, данные хранятся в нашем отношении с большой избыточностью.

Далее, как отразить факт, что некоторый поставщик, например Петров, временно прекратил поставки деталей? Если мы удалим все кортежи, в которых хранится информация о поставках этого поставщика, то мы *потеряем данные* о самом Петрове как потенциальном поставщике. Выйти из этого положения, оставив в отношении кортеж типа (2, Петров, NULL, NULL, NULL) мы не можем, т.к. атрибут "Номер детали" входит в состав потенциального ключа и не может содержать null-значений. То же самое произойдет, если некоторая деталь временно не поставляется никаким поставщиком. Получается, что мы не можем хранить информацию о том, что есть некий поставщик, если он не поставляет хотя бы одну деталь, и не можем хранить информацию о том, что есть некоторая деталь, если она никем не поставляется.

Подобные проблемы возникают потому, что мы смешали в одном отношении различные объекты предметной области - и данные о поставщиках, и данные о деталях, и данные о поставках деталей. Говорят, что это отношение *плохо нормализовано* (просто нормализованным оно является хотя бы потому, что оно есть отношение и, следовательно, автоматически находится в 1НФ).

О том, как правильно нормализовать отношения, будет сказано в следующих главах, сейчас же предложим разнести данные по трем отношениям - "Поставщики", "Детали", "Поставки". Для нас важно выяснить, каким образом данные, хранящиеся в этих отношениях взаимосвязаны друг с другом. Эта связь определяется семантикой предметной области и описывается фразами: "Поставщики выполняют Поставки", "Детали поставляются через Поставки". Эти две взаимосвязи косвенно определяют новую взаимосвязь между "Поставщиками" и "Деталями": "Детали поставляются Поставщиками".

Эти фразы отражают различные типы взаимосвязей. Чтобы более точно отразить предметную область, можно иначе переформулировать фразы: "Один Поставщик может

выполнять *несколько* Поставок", "*Одна* Деталь может поставляться *несколькими* Поставками". Это пример взаимосвязи типа "*один-ко-многим*".

Взаимосвязь между "Поставщиками" и "Деталями" можно переформулировать так: "*Несколько* Деталей может поставляться *несколькими* Поставщиками". Это пример взаимосвязи типа "*много-ко-многим*".

В реляционных базах данных основными являются взаимосвязи типа "один-ко-многим". Взаимосвязи типа "много-ко-многим" реализуются использованием нескольких взаимосвязей типа "один-ко-многим". Отношение, входящее в связь со стороны "один" (например, "Поставщики"), называют родительским отношением. Отношение, входящее в связь со стороны "много" (например, "Поставки"), называется дочернем отношением.

Механизм реализации взаимосвязи "один-ко-многим" состоит в том, что в дочернее отношение добавляются атрибуты, являющиеся ссылками на ключевые атрибуты родительского отношения. Эти атрибуты и являются внешними ключами, определяющими, с какими кортежами родительского отношения связаны кортежи дочернего отношения. Такие атрибуты еще называют *мигрирующими* из родительского отношения.

Таким образом, наш пример с поставщиками и поставляемыми деталями должен выглядеть следующим образом:

Номер поставщика	Наименование поставщика
1	Иванов
2	Петров
3	Сидоров

Таблица 6 Отношение "Поставщики"

Номер детали	Наименование детали
1	Болт
2	Гайка
3	Винт

Таблица 7 Отношение "Детали"

Номер поставщика	Номер детали	Поставляемое количество
1	1	100
1	2	200
1	3	300
2	1	150
2	2	250
3	3	1000

# Таблица 8 Отношение "Поставки"

В отношении "Поставки" атрибуты "Номер поставщика" и "Номер детали" являются ссылками на ключевые атрибуты отношений "Поставщики" и "Детали", и, следовательно, являются внешними ключами. Заметим, что данные отношения свободны от недостатков, описанных выше, когда все данные предлагалось хранить в одном отношении. Действительно, при изменении наименования поставщика или детали, это изменение происходит только в одном месте. Если поставщик прекратил поставки всех деталей, то удаляются соответствующие кортежи в отношении "Поставки", данные же о самом поставщике остаются без изменений.

Дадим точное определение.

*Определение* 2. Пусть дано отношение R. Подмножество атрибутов FK отношения R будем называть *внешним ключом*, если:

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

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

Замечание. Внешний ключ, также как и потенциальный, может быть простым и составным.

Замечание. Внешний ключ должен быть определен на тех же доменах, что и соответствующий первичный ключ родительского отношения.

<u>Замечание</u>. Внешний ключ, как правило, *не обладает свойством уникальности*. Так и должно быть, т.к. в дочернем отношении может быть несколько кортежей, ссылающихся на один и тот же кортеж родительского отношения. Это, собственно, и дает тип отношения "один-ко-многим".

<u>Замечание</u>. Если внешний ключ все-таки обладает свойством уникальности, то связь между отношениями имеет тип "*один-к-одному*". Чаще всего такие отношения объединяются в одно отношение, хотя это и не обязательно.

Замечание. Хотя каждое значение внешнего ключа обязано совпадать со значениями потенциального ключа в некотором кортеже родительского отношения, то обратное, вообще говоря, неверно. Например, могут существовать поставщики, не поставляющие никаких деталей.

<u>Замечание</u>. Для внешнего ключа не требуется, чтобы он был компонентом некоторого потенциального ключа (как получилось в примере с поставщиками и деталями).

<u>Замечание</u>. Null-значения для атрибутов внешнего ключа допустимы только в том случае, когда атрибуты внешнего ключа не входят в состав никакого потенциального ключа

## Целостность внешних ключей

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

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

## Замечания к правилам целостности сущностей и внешних ключей

На самом деле приведенные правила целостности сущностей и внешних ключей прямо следуют из определений понятий "потенциальный ключ" и "внешний ключ".

Действительно, в определении потенциального ключа требуется, чтобы потенциальный ключ обладал свойством уникальности. Это фактически означает, что мы должны уметь различать значения потенциальных ключей, т.е. при сравнении двух значений потенциального ключа мы всегда должны получать значения либо ИСТИНА, либо ЛОЖЬ. Но любое сравнение, в которое входит null-значение, принимает значение U - НЕИЗВЕСТНО, откуда следует, что атрибуты потенциального ключа не могут содержать null-значений.

Для внешних ключей правило целостности фактически входит в определение (п. 2 определения 2).

Таким образом, с точки зрения реляционной теории, явная формулировка правил целостности является излишней - они автоматически вытекают из определений понятий ключа и внешнего ключа.

Тем не менее, явная формулировка правил целостности имеет определенный практический смысл. В большинстве серьезных СУБД за выполнением этих ограничений следит сама СУБД, если, конечно, пользователь явно объявил потенциальные и внешние ключи. Но, во-первых, для некоторых систем можно допустить, чтобы эти ограничения не выполнялись, а во-вторых, некоторые системы просто не поддерживают понятия целостности, например, некоторые "настольные" СУБД типа FoxPro 2.5. В этих случаях за целостностью данных должен следить сам пользователь, или программист, разрабатывающий приложение для пользователя.

Явная формулировка правил целостности помогает четко понять, какие опасности несет в себе пренебрежение этими правилами.

## Операции, могущие нарушить ссылочную целостность

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

## Для родительского отношения

• Вставка кортежа в родительском отношении

- . При вставке кортежа в родительское отношение возникает новое значение потенциального ключа. Т.к. допустимо существование кортежей в родительском отношении, на которые нет ссылок из дочернего отношения, то вставка кортежей в родительское отношение не нарушает ссылочной целостности.
- Обновление кортежа в родительском отношении
  - . При обновлении кортежа в родительском отношении может измениться значение потенциального ключа. Если есть кортежи в дочернем отношении, ссылающиеся на обновляемый кортеж, то значения их внешних ключей станут некорректными. Обновление кортежа в родительском отношении может привести к нарушению ссылочной целостности, если это обновление затрагивает значение потенциального ключа.
- Удаление кортежа в родительском отношении
  - . При удалении кортежа в родительском отношении удаляется значение потенциального ключа. Если есть кортежи в дочернем отношении, ссылающиеся на удаляемый кортеж, то значения их внешних ключей станут некорректными. Удаление кортежей в родительском отношении может привести к нарушению ссылочной целостности.

## Для дочернего отношения

- Вставка кортежа в дочернее отношение
  - . Нельзя вставить кортеж в дочернее отношение, если вставляемое значение внешнего ключа некорректно. Вставка кортежа в дочернее отношение *привести к* нарушению ссылочной целостности.
- Обновление кортежа в дочернем отношении
  - . При обновлении кортежа в дочернем отношении можно попытаться некорректно изменить значение внешнего ключа. Обновление кортежа в дочернем отношении может привести к нарушению ссылочной целостности.
- Удаление кортежа в дочернем отношении
  - . При удалении кортежа в дочернем отношении ссылочная целостность не нарушается.

Таким образом, ссылочная целостность в принципе может быть нарушена при выполнении одной из четырех операций:

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

# Стратегии поддержания ссылочной целостности

Существуют две основные стратегии поддержания ссылочной целостности:

- **RESTRICT** (**OГРАНИЧИТЬ**)- не разрешать выполнение операции, приводящей к нарушению ссылочной целостности. Это самая простая стратегия, требующая только проверки, имеются ли кортежи в дочернем отношении, связанные с некоторым кортежем в родительском отношении.
- *CASCADE* (*KACKAДИРОВАТЬ*)- разрешить выполнение требуемой операции, но внести при этом необходимые поправки в других отношениях так, чтобы не допустить нарушения ссылочной целостности и сохранить все имеющиеся связи. Изменение начинается в родительском отношении и каскадно выполняется в дочернем отношении. В реализации этой стратегии имеется одна тонкость, заключающаяся в том, что дочернее отношение само может быть родительским для некоторого третьего отношения. При этом может дополнительно потребоваться выполнение какой-либо стратегии и для этой связи и т.д. Если при этом какая-либо из каскадных операций (любого уровня) не может быть выполнена, то необходимо отказаться от первоначальной операции и вернуть базу данных в исходное состояние. Это самая сложная стратегия, но она хороша тем, что при этом не нарушается связь между кортежами родительского и дочернего отношений.

Эти стратегии являются стандартными и присутствуют во всех СУБД, в которых имеется поддержка ссылочной целостности.

Можно рассмотреть *дополнительные стратегии поддержания ссылочной* иелостности:

- SET NULL (УСТАНОВИТЬ В NULL) разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на null-значения. Эта стратегия имеет два недостатка. Во-первых, для нее требуется допустить использование null-значений. Во-вторых, кортежи дочернего отношения теряют всякую связь с кортежами родительского отношения. Установить, с каким кортежем родительского отношения были связаны измененные кортежи дочернего отношения, после выполнения операции уже нельзя.
- **SET DEFAULT (УСТАНОВИТЬ ПО УМОЛЧАНИЮ)** разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на некоторое значение, принятое по умолчанию. Достоинство этой стратегии по сравнению с предыдущей в том, что она позволяет не пользоваться null-значеними. Недостатки заключаются в следующем. Во-первых, в родительском отношении должен быть некий кортеж, потенциальный ключ которого принят как значение по умолчанию для внешних ключей. В качестве такого "кортежа по умолчанию" обычно принимают специальный кортеж, заполненный нулевыми значениями (не null-значениями!). Этот кортеж нельзя удалять из родительского отношения, и в этом кортеже нельзя изменять значение потенциального ключа. Таким образом, не все кортежи родительского отношения становятся равнозначными, поэтому приходится прилагать дополнительные усилия для отслеживания этой неравнозначности. Это плата за отказ от использования null-значений. Во-вторых, как и в предыдущем случае, кортежи дочернего отношения теряют всякую связь с кортежами родительского отношения. Установить, с каким кортежем родительского отношения были связаны измененные кортежи дочернего отношения, после выполнения операции уже нельзя.

В некоторых реализация СУБД рассматривается еще одна стратегия поддержания ссылочной иелостности:

• *IGNORE (ИГНОРИРОВАТЬ*) - выполнять операции, не обращая внимания на нарушения ссылочной целостности.

Конечно, это не стратегия, а отказ от поддержки ссылочной целостности. В этом случае в дочернем отношении могут появляться некорректные значения внешних ключей, и вся ответственность за целостность базы данных ложится на пользователя.

В дополнение к приведенным стратегиям пользователь может придумать свою уникальную стратегию поддержания ссылочной целостности.

# Применение стратегий поддержания ссылочной целостности

Рассмотрим, как применяются стратегии поддержания ссылочной целостности при выполнении операций модификации базы данных.

#### При обновлении кортежа в родительском отношении

Допустимые стратегии:

**RESTRICT** (**ОГРАНИЧИТЬ**) - не разрешать обновление, если имеется хотя бы один кортеж в дочернем отношении, ссылающийся на обновляемый кортеж.

**CASCADE** (**КАСКАДИРОВАТЬ**) - выполнить обновление и каскадно изменить значения внешних ключей во всех кортежах дочернего отношения, ссылающихся на обновляемый кортеж.

**SET NULL (УСТАНОВИТЬ В NULL)** - выполнить обновление и во всех кортежах дочернего отношения, ссылающихся на обновляемый кортеж, изменить значения внешних ключей на null-значение.

**SET DEFAULT (УСТАНОВИТЬ ПО УМОЛЧАНИЮ)** - выполнить обновление и во всех кортежах дочернего отношения, ссылающихся на обновляемый кортеж, изменить значения внешних ключей на некоторое значение, принятое по умолчанию.

*IGNORE* (*ИГНОРИРОВАТЬ*) - выполнить обновление, не обращая внимания на нарушения ссылочной целостности.

## При удалении кортежа в родительском отношении

Допустимые стратегии:

**RESTRICT** (**ОГРАНИЧИТЬ**) - не разрешать удаление, если имеется хотя бы один кортеж в дочернем отношении, ссылающийся на удаляемый кортеж.

**CASCADE** (**КАСКАДИРОВАТЬ**) - выполнить удаление и каскадно удалить кортежи в дочернем отношении, ссылающиеся на удаляемый кортеж.

**SET NULL (УСТАНОВИТЬ В NULL)** - выполнить удаление и во всех кортежах дочернего отношения, ссылающихся на удаляемый кортеж, изменить значения внешних ключей на null-значение.

**SET DEFAULT (УСТАНОВИТЬ ПО УМОЛЧАНИЮ)** - выполнить удаление и во всех кортежах дочернего отношения, ссылающихся на удаляемый кортеж, изменить значения внешних ключей на некоторое значение, принятое по умолчанию.

*IGNORE (ИГНОРИРОВАТЬ*) - выполнить удаление, не обращая внимания на нарушения ссылочной целостности.

## При вставке кортежа в дочернее отношение

Допустимые стратегии:

**RESTRICT** (**ОГРАНИЧИТЬ**) - не разрешать вставку, если внешний ключ во вставляемом кортеже не соответствует ни одному значению потенциального ключа родительского отношения.

**SET NULL (УСТАНОВИТЬ В NULL)** - вставить кортеж, но в качестве значения внешнего ключа занести не предлагаемое пользователем некорректное значение, а null-значение.

**SET DEFAULT (УСТАНОВИТЬ ПО УМОЛЧАНИЮ)** - вставить кортеж, но в качестве значения внешнего ключа занести не предлагаемое пользователем некорректное значение, а некоторое значение, принятое по умолчанию.

*IGNORE (ИГНОРИРОВАТЬ)* - вставить кортеж, не обращая внимания на нарушения ссылочной целостности.

#### При обновлении кортежа в дочернем отношении

Допустимые стратегии:

**RESTRICT** (**ОГРАНИЧИТЬ**) - не разрешать обновление, если внешний ключ в обновляемом кортеже становится не соответствующим ни одному значению потенциального ключа родительского отношения.

 $SET\ NULL\ (YCTAHOBUTЬ\ B\ NULL)$  - обновить кортеж, но в качестве значения внешнего ключа занести не предлагаемое пользователем некорректное значение, а null-значение.

**SET DEFAULT (УСТАНОВИТЬ ПО УМОЛЧАНИЮ)** - обновить кортеж, но в качестве значения внешнего ключа занести не предлагаемое пользователем некорректное значение, а некоторое значение, принятое по умолчанию.

*IGNORE* (*ИГНОРИРОВАТЬ*) - обновить кортеж, не обращая внимания на нарушения ссылочной целостности.

#### Выводы

Современные СУБД допускают использование *null-значений*, т.к. данные часто бывают неполными или неизвестными. Споры о допустимости использования null-значений ведутся до сих пор. Использование null-значения связано с применением *трехзначной логики* (*three-valued logic*, *3VL*).

Средством, позволяющим однозначно идентифицировать кортежи отношения, являются потенциальные ключи отношения.

Потенциальный ключ отношения - это набор атрибутов отношения, обладающий свойствами *уникальности* и *неизбыточности*. Доступ к конкретному кортежу можно получить, лишь зная значение потенциального ключа для этого кортежа.

Традиционно один из потенциальных ключей объявляется *первичным ключом*, остальные - *альтернативными ключами*.

Потенциальный ключ, состоящий из одного атрибута, называется *простым*. Потенциальный ключ, состоящий из нескольких атрибутов, называется *составным*.

Отношения связываются друг с другом при помощи внешних ключей.

Внешний ключ отношения - это набор атрибутов отношения, содержащий ссылки на потенциальный ключ другого (или того же самого) отношения. Отношение, содержащее потенциальный ключ, на который ссылается некоторый внешний ключ, называется родительским отношением. Отношение, содержащее внешний ключ, называется дочерним отношением.

Внешний ключ не обязан обладать свойством уникальности. Поэтому, одному кортежу родительского отношения может соответствовать несколько кортежей дочернего отношения. Такой тип связи между отношениями называется "один-ко-многим".

Связи типа "*много-ко-многим*" реализуются использованием нескольких отношений типа "один-ко-многим".

В любой реляционной базе данных должны выполняться два ограничения:

- Целостность сущностей
- Целостность внешних ключей

Правило **целостности сущностей** состоит в том, что атрибуты, входящие в состав некоторого потенциального ключа не могут принимать null-значений.

Правило *целостности внешних ключей* состоит в том, что внешние ключи не должны ссылаться на отсутствующие в родительском отношении кортежи, т.е. внешние ключи должны быть корректны.

Ссылочную целостность могут нарушить операции, изменяющие состояние базы данных. Такими операциями являются операции вставки, обновления и удаления кортежей.

Для поддержания ссылочной целостности обычно используются две *основные стратегии*:

- *RESTRICT* (*OГРАНИЧИТЬ*) не разрешать выполнение операции, приводящей к нарушению ссылочной целостности.
- *CASCADE* (*KACKAДИРОВАТЬ*) разрешить выполнение требуемой операции, но внести каскадные изменения в другие отношения так, чтобы не допустить нарушения ссылочной целостности.

Дополнительными стратегиями поддержания ссылочной целостности являются:

- *SET NULL (УСТАНОВИТЬ В NULL)* все некорректные значения внешних ключей изменять на null-значения.
- SET DEFAULT (УСТАНОВИТЬ ПО УМОЛЧАНИЮ) все некорректные значения внешних ключей изменять на некоторое значение, принятое по умолчанию.

В реальных СУБД можно также отказаться от использования какой-либо стратегии поддержания ссылочной целостности:

• *IGNORE* (*ИГНОРИРОВАТЬ*) - выполнять операции, не обращая внимания на нарушения ссылочной целостности.

Пользователь может разработать свою уникальную стратегию поддержания ссылочной целостности.

# Глава 4. Реляционная алгебра

# Обзор реляционной алгебры

Третья часть реляционной модели, манипуляционная часть, утверждает, что доступ к реляционным данным осуществляется при помощи реляционной алгебры или эквивалентного ему реляционного исчисления.

В реализациях конкретных реляционных СУБД сейчас не используется в чистом виде ни реляционная алгебра, ни реляционное исчисление. Фактическим стандартом доступа к реляционным данным стал язык SQL (Structured Query Language). Язык SQL представляет собой смесь операторов реляционной алгебры и выражений реляционного исчисления, использующий синтаксис, близкий к фразам английского языка и расширенный дополнительными возможностями, отсутствующими в реляционной алгебре и реляционном исчислении. Вообще, язык доступа к данным называется реляционно полным, если он по выразительной силе не уступает реляционной алгебре (или, что то же самое, реляционному исчислению), т.е. любой оператор реляционной алгебры может быть выражен средствами этого языка. Именно таким и является язык SQL.

В данной главе будут рассмотрены основы реляционной алгебры.

#### Замкнутость реляционной алгебры

Реляционная алгебра представляет собой набор операторов, использующих отношения в качестве аргументов, и возвращающие отношения в качестве результата. Таким образом, реляционный оператор f выглядит как функция с отношениями в качестве аргументов:

$$R = f(R_1, R_2, \dots, R_n)$$

Реляционная алгебра является замкнутой, т.к. в качестве аргументов в реляционные операторы можно подставлять другие реляционные операторы, подходящие по типу:

$$R = f(f_1(R_{11}, R_{12}, ...), f_2(R_{21}, R_{22}, ...), ...)$$

Таким образом, в реляционных выражениях можно использовать вложенные выражения сколь угодно сложной структуры.

Каждое отношение обязано иметь уникальное имя в пределах базы данных. Имя отношения, полученного в результате выполнения реляционной операции, определяется в левой части равенства. Однако можно не требовать наличия имен от отношений, полученных в результате реляционных выражений, если эти отношения подставляются в качестве аргументов в другие реляционные выражения. Такие отношения будем называть неименованными отношениями. Неименованные отношения реально не существуют в базе данных, а только вычисляются в момент вычисления значения реляционного оператора.

Традиционно, вслед за Коддом [43], определяют восемь реляционных операторов, объединенных в две группы.

Теоретико-множественные операторы:

- Объединение
- Пересечение
- Вычитание
- Декартово произведение

Специальные реляционные операторы:

- Выборка
- Проекция
- Соединение
- Деление

Не все они являются независимыми, т.е. некоторые из этих операторов могут быть выражены через другие реляционные операторы.

## Отношения, совместимые по типу

Некоторые реляционные операторы (например, объединение) требуют, чтобы отношения имели одинаковые заголовки. Действительно, отношения состоят из заголовка и тела. Операция объединения двух отношений есть просто объединение двух множеств кортежей, взятых из тел соответствующих отношений. Но будет ли результат отношением? Во-первых, если исходные отношения имеют разное количество атрибутов, то, очевидно, что множество, являющееся объединением таких разнотипных кортежей нельзя представить в виде отношения. Во-вторых, пусть даже отношения имеют одинаковое количество атрибутов, но атрибуты имеют различные наименования. Как тогда определить заголовок отношения, полученного в результате объединения множеств кортежей? В-третьих, пусть отношения имеют одинаковое количество атрибутов, атрибуты имеют одинаковые наименования, но определенны на различных доменах. Тогда снова объединение кортежей не будет образовывать отношение.

*Определение 1.* Будем называть отношения *совместимыми по типу*, если они имеют идентичные заголовки, а именно,

- Отношения имеют *одно и то же множество имен атрибутов*, т.е. для любого атрибута в одном отношении найдется атрибут с таким же наименованием в другом отношении,
- Атрибуты с одинаковыми именами определены на одних и тех же доменах.

Некоторые отношения не являются совместимыми по типу, но становятся таковыми после некоторого переименования атрибутов. Для того чтобы такие отношения можно было использовать в реляционных операторах, вводится вспомогательный *оператор переименования атрибутов*.

## Оператор переименования атрибутов

Оператор переименования атрибутов имеет следующий синтаксис:

R RENAME  $Atr_1$ ,  $Atr_2$ ,... AS  $NewAtr_1$ ,  $NewAtr_2$ ,...

где

R - отношение.

 $Atr_1, Atr_2, \dots$  исходные имена атрибутов,

 $NewAtr_1$ ,  $NewAtr_2$ ,...- новые имена атрибутов.

В результате применения оператора переименования атрибутов получаем новое отношение, с измененными именами атрибутов.

# Пример 1.

Следующий оператор возвращает неименованное отношение, в котором атрибут City - Mum переименован в Cityld:

City RENAME City Num AS Cityld

## Теоретико-множественные операторы

## Объединение

Определение 2. Объединением двух совместимых по типу отношений  $^{A}$  и  $^{B}$  называется отношение с тем же заголовком, что и у отношений  $^{A}$  и  $^{B}$ , и телом, состоящим из кортежей, принадлежащих или  $^{A}$ , или  $^{B}$ , или обоим отношениям.

Синтаксис операции объединения:

A UNION B

<u>Замечание</u>. Объединение, как и любое отношение, не может содержать одинаковых кортежей. Поэтому, если некоторый кортеж входит и в отношение  $^{A}$ , и отношение  $^{B}$ , то в объединение он входит один раз.

<u>Пример 2</u>. Пусть даны два отношения  $A_{\rm H}$   $B_{\rm C}$  информацией о сотрудниках:

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

Таблина 1 Отношение А

Табельный номер	Фамилия	Зарплата
1	Иванов	1000
2	Пушников	2500
4	Сидоров	3000

Таблипа 2 Отношение В

Объединение отношений  $^{A}$ и  $^{B}$  будет иметь вид:

Табельный номер	Фамилия	Зарплата
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000
2	Пушников	2500
4	Сидоров	3000

Таблица 3 Отношение A UNION В

<u>Замечание</u>. Как видно из приведенного примера, потенциальные ключи, которые были в отношениях  $^{A}$  и  $^{B}$  не наследуются объединением этих отношений. Поэтому, в объединении отношений  $^{A}$  и  $^{B}$  атрибут "Табельный номер" может содержать дубликаты значений. Если бы это было не так, и ключи наследовались бы, то это противоречило бы понятию объединения как "объединение множеств". Конечно, объединение отношений  $^{A}$  и  $^{B}$  имеет, как и любое отношение, потенциальный ключ, например, состоящий из всех атрибутов.

## Пересечение

Определение 3. **Пересечением** двух совместимых по типу отношений  $^{A}$  и  $^{B}$  называется отношение с тем же заголовком, что и у отношений  $^{A}$  и  $^{B}$ , и телом, состоящим из кортежей, принадлежащих одновременно обоим отношениям  $^{A}$  и  $^{B}$ .

Синтаксис операции пересечения:

AINTERSECT B

<u>Пример 3</u>. Для тех же отношений A и B, что и в предыдущем примере пересечение имеет вид:

Табельный номер	Фамилия	Зарплата
1	Иванов	1000

Таблица 4 Отношение A INTERSECT В

Замечание. Казалось бы, что в отличие от операции объединения, потенциальные ключи могли бы наследоваться пересечением отношений. Однако это не так. Вообще, никакие реляционные операторы не передают результатирующему отношению никаких данных о потенциальных ключах. В качестве причины этого можно было бы привести тривиальное соображение, что так получается более просто и симметрично - все операторы устроены одинаково. На самом деле причина более глубока, и заключается в том, что потенциальный ключ - семантическое понятие, отражающее различимость объектов предметной области. Наличие потенциальных ключей не выводится из структуры отношения, а явно задается для каждого отношения, исходя из его смысла. Реляционные же операторы являются формальными операциями над отношениями и выполняются одинаково, независимо от смысла данных, содержащихся в отношениях. Поэтому, реляционные операторы ничего не могут "знать" о смысле данных. Трактовка результата реляционных операций - дело пользователя.

#### Вычитание

Определение 4. **Вычитанием** двух совместимых по типу отношений  $^{A}$ и  $^{B}$  называется отношение с тем же заголовком, что и у отношений  $^{A}$ и  $^{B}$ , и телом, состоящим из кортежей, принадлежащих отношению  $^{A}$ и не принадлежащих отношению  $^{B}$ .

Синтаксис операции вычитания:

AMINUS B

<u>Пример 4</u>. Для тех же отношений  $^{A}$  и  $^{B}$ , что и в предыдущем примере вычитание имеет вил:

Табельный номер	Фамилия	Зарплата
2	Петров	2000
3	Сидоров	3000

Таблица 5 Отношение A MINUS В

## Декартово произведение

Определение 5. Декартовым произведением двух отношений  ${}^{A(A_1,\,A_2,\,\ldots,\,A_n)}$  и  ${}^{B(B_1,\,B_2,\,\ldots,\,B_m)}$  называется отношение, заголовок которого является сцеплением заголовков отношений  ${}^{A}$  и  ${}^{B}$  :

$$(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$$

а тело состоит из кортежей, являющихся *сцеплением кортежей* отношений  $A_{\rm H}$  B:

$$(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m)$$

таких, что 
$$(a_1, a_2, ..., a_n) \in A$$
,  $(b_1, b_2, ..., b_m) \in B$ .

Синтаксис операции декартового произведения:

#### ATIMES B

<u>Замечание</u>. Мощность произведения  $^{A\, T\, {\rm IM}\, {\rm ES}\, B}$  равна произведению мощностей отношений  $^{A}\, {\rm u}^{\ B}\, ,$  т.к. каждый кортеж отношения  $^{A}\, {\rm coe}$  диняется с каждым кортежем отношения  $^{B}\, .$ 

<u>Замечание</u>. Если в отношения  $^{A}$ и  $^{B}$  имеются атрибуты с одинаковыми наименованиями, то перед выполнением операции декартового произведения такие атрибуты необходимо переименовать.

Замечание. Перемножать можно любые два отношения, совместимость по типу при этом не требуется.

<u>Пример 5</u>. Пусть даны два отношения Aи Bс информацией о поставщиках и деталях:

Номер поставщика	Наименование поставщика	
1	Иванов	
2	Петров	
3	Сидоров	

Таблица 6 Отношение А (Поставщики)

Номер детали	Наименование детали
1	Болт
2	Гайка
3	Винт

Таблица 7 Отношение В (Детали)

Декартово произведение отношений  $A_{\rm H}$  B будет иметь вид:

Номер поставщика	Наименование поставщика	Номер детали	Наименование детали
1	Иванов	1	Болт
1	Иванов	2	Гайка
1	Иванов	3	Винт

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

Таблица 8 Отношение A TIMES В

Замечание. Сама по себе операция декартового произведения не очень важна, т.к. она не дает никакой новой информации, по сравнению с исходными отношениями. Для реальных запросов эта операция почти никогда не используется. Однако операция декартового произведения важна для выполнения специальных реляционных операций, о которых речь пойлет ниже.

# Специальные реляционные операторы

## Выборка (ограничение, селекция)

Определение 6. Выборкой (ограничением, селекцией) на отношении  $^{A}$ с условием  $^{c}$  называется отношение с тем же заголовком, что и у отношения  $^{A}$ , и телом, состоящем из кортежей, значения атрибутов которых при подстановке в условие  $^{c}$  дают значение ИСТИНА.  $^{c}$  представляет собой логическое выражение, в которое могут входить атрибуты отношения  $^{A}$ и (или) скалярные выражения.

В простейшем случае условие  $^{\mathcal{C}}$  имеет вид  $^{X \oplus Y}$ , где  $^{\Theta}$  - один из операторов сравнения (=,  $\neq$ ,  $\leq$   $\geq$ ,  $\geq$  и т.д.), а  $^{\mathcal{X}}$  и  $^{\mathcal{Y}}$  - атрибуты отношения  $^{\mathcal{A}}$  или скалярные значения. Такие выборки называются  $^{\Theta}$  -выборки (тэта-выборки) или  $^{\Theta}$  -ограничения,  $^{\Theta}$  -селекции.

Синтаксис операции выборки:

A WHERE c

или

A WHERE X®Y

<u>Пример 6</u>. Пусть дано отношение  $^{A}$ с информацией о сотрудниках:

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

Таблица 9 Отношение А

Результат выборки  $^{A}$  WHERE Зартиата < 3000 будет иметь вид:

Табельный номер	Фамилия	Зарплата
1	Иванов	1000
2	Петров	2000

# Таблица 10 Отношение A WHERE Зарплата<3000

Смысл операции выборки очевиден - выбрать кортежи отношения, удовлетворяющие некоторому условию. Таким образом, операция выборки дает "горизонтальный срез" отношения по некоторому условию.

## Проекция

Определение 7. **Проекцией** отношения  $^A$  по атрибутам  $^{X,Y,...,Z}$ , где каждый из атрибутов принадлежит отношению  $^A$ , называется отношение с заголовком  $^{(X,Y,...,Z)}$  и телом, содержащим множество кортежей вида  $^{(x,y,...,z)}$ , таких, для которых в отношении  $^A$  найдутся кортежи со значением атрибута  $^X$  равным  $^x$ , значением атрибута  $^Y$  равным  $^y$ , ..., значением атрибута  $^Z$  равным  $^Z$ .

Синтаксис операции проекции:

<u>Замечание</u>. Операция проекции дает "*вертикальный срез*" отношения, в котором удалены все возникшие при таком срезе дубликаты кортежей.

<u>Пример 7</u>. Пусть дано отношение  $^{A}$ с информацией о поставщиках, включающих наименование и месторасположение:

Номер поставщика	Наименование поставщика	Город поставщика
1	Иванов	Уфа
2	Петров	Москва
3	Сидоров	Москва
4	Сидоров	Челябинск

Таблица 11 Отношение А (Поставщики)

Проекция  $A[\Gamma \circ p \circ d]$  поставщика ] будет иметь вид:

Город поставщика
Уфа
Москва
Челябинск

## Таблица 12 Отношение А[Город поставщика]

#### Соединение

Операция соединения отношений, наряду с операциями выборки и проекции, является одной из наиболее важных реляционных операций.

Обычно рассматривается несколько разновидностей операции соединения:

- Общая операция соединения
- Экви-соединение
- Естественное соединение

Наиболее важным из этих частных случаев является операция естественного соединения. Все разновидности соединения являются частными случаями общей операции соединения.

# Общая операция соединения

Определение 8. Соединением отношений  $^A$ и  $^B$  по условию  $^c$  называется отношение (ATIMES  $^B$ ) WHERE  $^c$ 

 $^{c}$  представляет собой логическое выражение, в которое могут входить атрибуты отношений  $^{A}$  и  $^{B}$  и (или) скалярные выражения.

Таким образом, операция соединения есть результат последовательного применения операций декартового произведения и выборки. Если в отношениях  $^{A}$ и  $^{B}$  имеются атрибуты с одинаковыми наименованиями, то перед выполнением соединения такие атрибуты необходимо переименовать.

#### Тэта-соединение

*Определение* 9. Пусть отношение  $^{A}$  содержит атрибут  $^{X}$ , отношение  $^{B}$  содержит атрибут  $^{Y}$ , а  $^{\odot}$  - один из операторов сравнения ( $^{=,\neq,\,\leqslant,\,\leqslant\,>,\,\geq}$  и т.д.). Тогда  $^{\odot}$  -*соединением* отношения  $^{A}$  по атрибуту  $^{Y}$  с отношением  $^{B}$  по атрибуту  $^{Y}$  называют отношение

(ATIMES B) WHERE X Y

Это частный случай операции общего соединения.

Иногда, для операции  $^{\odot}$  -соединения применяют следующий, более короткий синтаксис:  $A[X\Theta Y]B$ 

<u>Пример 8</u>. Рассмотрим некоторую компанию, в которой хранятся данные о поставщиках и поставляемых деталях. Пусть поставщикам и деталям присвоен некий статус. Пусть бизнес компании организован таким образом, что поставщики имеют право поставлять только те детали, статус которых не выше статуса поставщика (смысл этого может быть в

том, что хороший поставщик с высоким статусом может поставлять больше разновидностей деталей, а плохой поставщик с низким статусом может поставлять только ограниченный список деталей, важность которых (статус детали) не очень высока).

Номер поставщика	Наименование поставщика	X
		(Статус поставщика)
1	Иванов	4
2	Петров	1
3	Сидоров	2

Таблица 13 Отношение А (Поставщики)

Номер детали	<b>Наименование</b> детали	Y
		(Статус
		детали)
1	Болт	3
2	Гайка	2
3	Винт	1

Таблица 14 Отношение В (Детали)

Ответ на вопрос "какие поставщики имеют право поставлять какие детали?" дает  $^{\boxdot}$  - соединение  $^{A[X \geq Y]B}$  :

Номер поставщика	Наименование поставщика	Х (Статус поставщика)	<b>Номер</b> детали	Наименование детали	Ү (Статус детали)
1	Иванов	4	1	Болт	3
1	Иванов	4	2	Гайка	2
1	Иванов	4	3	Винт	1
2	Петров	1	3	Винт	1
3	Сидоров	2	2	Гайка	2
3	Сидоров	2	3	Винт	1

Таблица 15 Отношение "Какие поставщики поставляют какие детали"

## Экви-соединение

Наиболее важным частным случаем  $^{\tiny lacktrightarrow}$  -соединения является случай, когда  $^{\tiny lacktrightarrow}$  есть просто равенство.

#### Синтаксис экви-соединения:

**<u>Пример 9.</u>** Пусть имеются отношения  $^{P}$ ,  $^{D}$ и  $^{PD}$ , хранящие информацию о поставщиках, деталях и поставках соответственно (для удобства введем краткие наименования атрибутов):

Номер поставщика	Наименование поставщика
PNUM	PNAME
1	Иванов
2	Петров
3	Сидоров

Таблица 16 Отношение Р (Поставщики)

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

Таблица 17 Отношение D (Детали)

Номер поставщика	Номер детали	Поставляемое количество
PNUM	<b>DNUM</b>	VOLUME
1	1	100
1	2	200
1	3	300
2	1	150
2	2	250
3	1	1000

Таблица 18 Отношение РD (Поставки)

Ответ на вопрос, какие детали поставляются поставщиками, дает экви-соединение P[PNUM = PNUM]PD. На самом деле, т.к. в отношениях имеются одинаковые атрибуты, то требуется сначала переименовать атрибуты, а потом выполнить экви-соединение. Запись становится более громоздкой:

(PRENAMEPNUM AS PNUM 1) [PNUM 1 = PNUM 2] (PDRENAME PNUM AS PNUM 2)

Обычно, такой сложной формой записи не пользуются. Но как бы то ни было, в результате имеем отношение:

Номер поставщика	Наименование поставщика	Номер поставщика	Номер детали	Поставляемое количество
PNUM1	PNAME	PNUM2	DNUM	VOLUME
1	Иванов	1	1	100
1	Иванов	1	2	200
1	Иванов	1	3	300
2	Петров	2	1	150
2	Петров	2	2	250
3	Сидоров	3	1	1000

Таблица 19 Отношение "Какие детали поставляются какими поставщиками"

Недостатком экви-соединения является то, что если соединение происходит по атрибутам с одинаковыми наименованиями (а так чаще всего и происходит!), то в результатирующем отношении появляется два атрибута с одинаковыми значениями. В нашем примере атрибуты PNUM1 и PNUM2 содержат дублирующие данные. Избавиться от этого недостатка можно, взяв проекцию по всем атрибутам, кроме одного из дублирующих. Именно так действует естественное соединение.

#### Естественное соединение

Определение 10. Пусть даны отношения  $A(A_1,A_2,...,A_n,X_1,X_2,...,X_p)$  и  $B(X_1,X_2,...,X_p,B_1,B_2,...,B_m)$ , имеющие одинаковые атрибуты  $X_1,X_2,...,X_p$  (т.е. атрибуты с одинаковыми именами и определенные на одинаковых доменах).

Тогда *естественным соединением* отношений A и B называется отношение с заголовком  $(A_1,A_2,\ldots,A_n,X_1,X_2,\ldots,X_p,B_1,B_2,\ldots,B_m)$  и телом, содержащим множество кортежей  $(a_1,a_2,\ldots,a_n,x_1,x_2,\ldots,x_p,b_1,b_2,\ldots,b_m)$ , таких, что  $(a_1,a_2,\ldots,a_n,x_1,x_2,\ldots,x_p)\in A$  и  $(x_1,x_2,\ldots,x_p,b_1,b_2,\ldots,b_m)\in B$ 

Естественное соединение настолько важно, что для него используют специальный синтаксис:

#### AJOIN B

<u>Замечание</u>. В синтаксисе естественного соединения не указываются, по каким атрибутам производится соединение. Естественное соединение производится *по всем* одинаковым атрибутам.

<u>Замечание</u>. Естественное соединение эквивалентно следующей последовательности реляционных операций:

- 1. Переименовать одинаковые атрибуты в отношениях
- 2. Выполнить декартово произведение отношений
- 3. Выполнить выборку по совпадающим значениям атрибутов, имевших одинаковые имена
- 4. Выполнить проекцию, удалив повторяющиеся атрибуты
- 5. Переименовать атрибуты, вернув им первоначальные имена

<u>Замечание</u>. Можно выполнять последовательное естественное соединение нескольких отношений. Нетрудно проверить, что естественное соединение (как, впрочем, и соединение общего вида) обладает свойством *ассоциативности*, т.е.

$$(A \text{ JOIN } B) \text{ JOIN } C = A \text{ JOIN } (B \text{ JOIN } C)$$

поэтому такие соединения можно записывать, опуская скобки:

#### AJOIN BJOIN C

**Пример 10**. В предыдущем примере ответ на вопрос "какие детали поставляются поставщиками", более просто записывается в виде естественного соединения трех отношений P JOIN PD JOIN D (для удобства просмотра порядок атрибутов изменен, это является допустимым по свойствам отношений):

Номер поставщика	Наименование поставщика	Номер детали	Наименование детали	Поставляемое количество
PNUM	PNAME	DNUM	DNAME	VOLUME
1	Иванов	1	Болт	100
1	Иванов	2	Гайка	200
1	Иванов	3	Винт	300
2	Петров	1	Болт	150
2	Петров	2	Гайка	250
3	Сидоров	1	Болт	1000

Таблица 20 Отношение Р JOIN PD JOIN D

#### Деление

Определение 11. Пусть даны отношения  $A(X_1,X_2,...,X_n,Y_1,Y_2,...,Y_m)$  и  $B(Y_1,Y_2,...,Y_m)$ , причем атрибуты  $Y_1,Y_2,...,Y_m$  общие для двух отношений. Делением отношений A на B называется отношение с заголовком  $(X_1,X_2,...,X_n)$  и телом, содержащим множество кортежей  $(x_1,x_2,...,x_n)$ , таких, что для acex кортежей  $(y_1,y_2,...,y_m) \in B$  в отношении A найдется кортеж  $(x_1,x_2,...,x_n,y_1,y_2,...,y_m)$ .

Отношение  $^{A}$  выступает в роли  $^{deлимого}$ , отношение  $^{B}$  выступает в роли  $^{deлимеля}$ . Деление отношений аналогично делению чисел с остатком.

Синтаксис операции деления:

#### A DEVIDBY B

<u>Замечание</u>. Типичные запросы, реализуемые с помощью операции деления, обычно в своей формулировке имеют слово "все" - "какие поставщики поставляют *все* детали?".

<u>Пример 11</u>. В примере с поставщиками, деталями и поставками ответим на вопрос, "какие поставщики поставляют *все* детали?".

В качестве делимого возьмем проекцию X = PD[PNUM, DNUM], содержащую номера поставщиков и номера поставляемых ими деталей:

Номер поставщика	Номер детали
PNUM	DNUM
1	1
1	2
1	3
2	1
2	2
3	1

Таблица 21 Проекция X=PD[PNUM,DNUM]

В качестве делителя возьмем проекцию Y = D[DNUM], содержащую список номеров всех деталей (не обязательно поставляемых кем-либо):

Номер детали
DNUM
1
2
3

Таблица 22 Проекция Y=D[DNUM]

Деление X DEVIDEBY Y дает список номеров поставщиков, поставляющих все детали:

Номер поставщика
PNUM
1

#### Таблица 23 Отношение X DEVIDEBY Y

Оказалось, что только поставщик с номером 1 поставляет все детали.

# Примеры использования реляционных операторов

Пример 12. Получить имена поставщиков, поставляющих деталь номер 2.

Решение:

((DP JOIN P) WHERE DNUM = 2)[PNAME]

<u>Пример 13</u>. Получить имена поставщиков, поставляющих по крайней мере одну гайку.

Решение:

 $(((D \text{ WHERE } DNAME = \Gamma a \tilde{u} \kappa a) \text{ JOIN } DP) \text{ JOIN } P)[PNAME]$ 

Ответ на этот запрос можно получить и иначе:

((( D JOIN DP) JOIN P) WHERE DNAME = \( \text{Ta\tilde{u}} \text{ra} ) [ PNAME ]

Пример 14. Получить имена поставщиков, поставляющих все детали.

Решение:

((DP[PNUM, DNUM] DEVIDEBY D[DNUM]) JOIN P)[PNAME]

Пример 15. Получить имена поставщиков, не поставляющих деталь номер 2.

Решение:

 $((P[PNUM] \, \texttt{MINUS} ((P \, \texttt{JOIN} \, DP) \, \, \texttt{WHERE} \, DNUM = 2)[PNUM])$   $\texttt{JOIN} \, P)[PNAME]$ 

Ответ на этот запрос можно получить и пошагово:

 $T_1 = P[PNUM]_{-}$  получить список номеров всех поставщиков

 $T_2$  =  $P\,\mathrm{JOIN}\,DP$  - соединить данные о поставщиках и поставках

 $T_3 = T_2$  W HERE DNUM = 2 - в данных о поставщиках и поставках оставить только данные о поставках детали номер 2.

 $T_4 = T_3 [PNUM]_-$  получить список номеров поставщиков, поставляющих деталь номер 2.

 $T_5 = T_1 \text{ MINUS} T_4$  - получить список номеров поставщиков, не поставляющих деталь номер 2.

 $T_6 = T_5 \text{ JOIN } P$  - соединить список номеров поставщиков, не поставляющих деталь номер 2 с данными о поставщиках (получатся полные данные о поставщиках, не поставляющих деталь номер 2).

 $T_7 = T_6 [\mathit{PNAME}]_-$  искомый ответ (имена поставщиков, не поставляющих деталь номер 2).

## Зависимые реляционные операторы

Как было сказано в начале главы, не все операторы реляционной алгебры являются независимыми - некоторые из них выражаются через другие реляционные операторы.

## Оператор соединения

Оператор соединения определяется через операторы декартового произведения и выборки. Для оператора естественного соединения добавляется оператор проекции.

## Оператор пересечения

Оператор пересечения выражается через вычитание следующим образом:

A INTERSECT B = A MINUS (A MINUS B)

# Оператор деления

Оператор деления выражается через операторы вычитания, декартового произведения и проекции следующим образом:

$$A \text{ DEVIDEBY } B = A[X] \text{ MINUS } ((A[X] \text{ TIMES } B) \text{ MINUS } A)[X]$$

Таким образом показано, что операторы *соединения*, *пересечения* и *деления* можно выразить через другие реляционные операторы, т.е. эти операторы не являются примитивными.

## Примитивные реляционные операторы

Оставшиеся реляционные операторы (объединение, вычитание, декартово произведение, выборка, проекция) являются **примитивными операторами** - их нельзя выразить друг через друга.

## Оператор декартового произведения

Оператор декартового произведения - это единственный оператор, увеличивающий количество атрибутов, поэтому его нельзя выразить через объединение, вычитание, выборку, проекцию.

## Оператор проекции

Оператор проекции - единственный оператор, уменьшающий количество атрибутов, поэтому его нельзя выразить через объединение, вычитание, декартово произведение, выборку.

# Оператор выборки

Оператор выборки - единственный оператор, позволяющий проводить сравнения по атрибутам отношения, поэтому его нельзя выразить через объединение, вычитание, декартово произведение, проекцию.

#### Операторы объединения и вычитания

Доказательство примитивности операторов объединения и вычитания более сложны и мы их здесь не приводим.

#### Запросы, невыразимые средствами реляционной алгебры

Несмотря на мощь языка реляционной алгебры, имеется ряд типов запросов, которые принципиально нельзя выразить только при помощи операторов реляционной алгебры. Это вовсе не означает, что ответы на эти запросы нельзя получить вообще. Просто, для получения ответов на подобные запросы приходится применять процедурные расширения реляционных языков.

## Плохая нормализация отношений

Данный пример взят из книги Гилуа М.М. [6, стр.43].

**Пример 16**. Пусть имеется отношение ХИМИЧЕСКИЙ\_СОСТАВ\_ВЕЩЕСТВ с набором атрибутов (Наименование вещества, Водород, Гелий, ..., 105\_элемент). Значением атрибута "Вещество" являются наименования химических веществ, значениями остальных атрибутов - процентный состав соответствующих элементов в этом веществе. Такое отношение могло бы иметь, к примеру, следующий вид:

Наименование вещества	Водород	Гелий	•••	105 элемент
Дезоксирибону-клеиновая кислота	5	3		0.01
Бензин	50	0		0
	•••			

Таблица 24 Отношение ХИМИЧЕСКИЙ\_СОСТАВ\_ВЕЩЕСТВ

Рассмотрим запрос "Найти все химические элементы, содержание которых в каком-либо из веществ превышает заданный процент (скажем, 90)".

С алгоритмической точки зрения этот запрос выполняется элементарно - просматриваются все столбцы таблицы, если в столбце присутствует хотя бы одно значение, большее 90, то запоминается заголовок этого столбца. Набор наименований запомненных столбцов и является ответом на запрос.

Формально невозможно выразить этот запрос в рамках реляционной алгебры, т.к. ответом на этот запрос должен быть *список атрибутов* отношений, удовлетворяющих определенному условию. В реляционной алгебре нет операторов, манипулирующих с наименованиями атрибутов.

На самом деле, этот пример показывает, что таблица плохо нормализована (нормализация отношений рассматривается в гл.6 и 7). В таблице есть набор однотипных атрибутов ("Водород", "Гелий" и т.д. в количестве 105 столбцов).

Правильнее разбить это отношение на три различных отношения:

- 1. ВЕЩЕСТВО(НОМ ВЕЩЕСТВА, ВЕЩЕСТВО),
- 2. ЭЛЕМЕНТЫ(НОМ\_ЭЛЕМЕНТА, ЭЛЕМЕНТ),
- 3. ХИМИЧЕСКИЙ\_СОСТАВ\_ВЕЩЕСТВ(<u>НОМ\_ВЕЩЕСТВА</u>, <u>НОМ\_ЭЛЕМЕНТА</u>, ПРОЦЕНТ).

ном_вещества	вещество
1	Дезоксирибонуклеиновая кислота
2	Бензин

Таблица 25 Отношение ВЕЩЕСТВО

ном_элемента	ЭЛЕМЕНТ
1	Водород
2	Гелий
105	

Таблица 26 Отношение ЭЛЕМЕНТЫ

ном_вещества	ном_элемента	ПРОЦЕНТ
1	1	5
1	2	3
1	105	0.01
2	1	50

Таблица 27 Отношение ХИМИЧЕСКИЙ\_СОСТАВ\_ВЕЩЕСТВ

Для отношений, нормализованных таким образом, исходный запрос реализуется следующей последовательностью операторов:

- 1. R1(HOMEP\_BEЩЕСТВА, HOM\_ЭЛЕМЕНТА, ПРОЦЕНТ)= XИМИЧЕСКИЙ\_СОСТАВ\_ВЕЩЕСТВ[ПРОЦЕНТ>90]. (Выборка из отношения).
- 2. R2(HOM\_ЭЛЕМЕНТА) = R1[HOM\_ЭЛЕМЕНТА]. (Проекция отношения).
- 3. R3(HOM\_ЭЛЕМЕНТА,ЭЛЕМЕНТ)= R2[HOM\_ЭЛЕМЕНТА=HOM\_ЭЛЕМЕНТА]ЭЛЕМЕНТЫ. (Естественное соединение)
- 4. ОТВЕТ(ЭЛЕМЕНТ) = R3[ЭЛЕМЕНТ]. (Проекция таблицы).

На языке SQL такой запрос реализуется одной командой:

**SELECT** ЭЛЕМЕНТЫ. ЭЛЕМЕНТ

**FROM** ЭЛЕМЕНТЫ, ХИМИЧЕСКИЙ СОСТАВ ВЕЩЕСТВ

WHERE

ЭЛЕМЕНТЫ. HOM\_ЭЛЕМЕНТА=ХИМИЧЕСКИЙ\_СОСТАВ\_ВЕЩЕСТВ. HOM\_ЭЛЕМЕНТА AND ХИМИЧЕСКИЙ\_СОСТАВ\_ВЕЩЕСТВ. ПРОЦЕНТ>90;

#### Невыразимость транзитивного замыкания реляционными операторами

Следующий пример иллюстрирует класс запросов, невыразимых средствами реляционной алгебры или реляционного исчисления по причине невыразимости средствами реляционной алгебры транзитивного замыкания отношений (см. гл. 1).

<u>Пример 17</u>. Рассмотрим отношение, описывающее сотрудников некоего предприятия. Отношение содержит данные о табельном номере сотрудника, фамилии, должности и табельном номере руководителя сотрудника – СОТРУДНИКИ (<u>ТАБ НОМ</u>, ФАМИЛИЯ, ДОЛЖНОСТЬ, ТАБ НОМ РУК):

ТАБ_НОМ	ФАМИЛИЯ	должность	ТАБ_НОМ_РУК
1	Иванов	Директор	1
2	Петров	Глав.бухгалтер	1
3	Сидоров	Бухгалтер	2
4	Васильев	Начальник цеха	1
5	Сухов	Мастер	4
6	Шарипов	Рабочий	5
	•••		

Таблица 28 Отношение СОТРУДНИКИ

Рассмотрим запрос "Перечислить всех руководителей (прямых и непрямых) данного сотрудника".

Ответом на запрос может быть получен при помощи понятия транзитивного замыкания. Однако транзитивное замыкание не может быть выражено операторами реляционной алгебры.

## Кросс-таблицы

Одной из задач, связанных с представлением табличных данных является построение так называемых кросс-таблиц.

Пусть имеется отношение с тремя атрибутами и потенциальным ключом, включающим первые два атрибута. Примером такого отношения могут быть данные с объемами продаж различных товаров за некоторые промежутки времени:

Товар	Месяц	Количество
Компьютеры	Январь	100
Принтеры	Январь	200
Сканеры	Январь	300

Компьютеры	Февраль	150
Принтеры	Февраль	250
Сканеры	Февраль	350

Таблица 29 Данные о продажах

Требуется представить эти данные в виде таблицы, по строкам которой идут наименования товаров, по столбцам - месяцы, а в ячейках содержатся объемы продаж. Это и будет кросс-таблицей:

Товар	Январь Февралі		•••
Компьютеры	100	150	
Принтеры	200	250	
Сканеры	300	350	

Таблица 30 Кросс-таблица

Построение кросс-таблицы средствами реляционной алгебры невозможно, т.к. для этого требуется превратить данные в ячейках таблицы в наименования новых столбцов таблицы.

#### Выводы

Доступ к реляционным данным возможен при помощи операторов реляционной алгебры. Реляционная алгебра представляет собой набор операторов, использующих отношения в качестве аргументов, и возвращающие отношения в качестве результата. Реляционная алгебра замкнута таким образом, что результаты одних реляционных выражений можно использовать в других выражениях.

Традиционно определяют восемь реляционных операторов, объединенных в две группы.

Теоретико-множественные операторы: *объединение*, *пересечение*, *вычитание*, *декартово произведение*.

Специальные реляционные операторы: выборка, проекция, соединение, деление.

Для выполнения некоторых реляционных операторов требуется, чтобы отношения были *совместимы по типу*.

Не все операторы реляционной алгебры являются независимыми - некоторые из них выражаются через другие реляционные операторы. Операторы *соединения*, *пересечения* и *деления* можно выразить через другие реляционные операторы, т.е. эти операторы не являются примитивными. Оставшиеся реляционные операторы (объединение, вычитание, декартово произведение, выборка, проекция) являются примитивными операторами - их нельзя выразить друг через друга.

Имеется несколько типов запросов, которые нельзя выразить средствами реляционной алгебры. К ним относятся запросы, требующие дать в ответе список атрибутов, удовлетворяющих определенным условиям, построение транзитивного замыкания отношений, построение кросс-таблиц. Для получения ответов на подобные запросы приходится использовать процедурные расширения реляционных языков.

# Глава 5. Элементы языка SQL

В данной главе рассматриваются элементы языка SQL (Structured Query Language). Текущая версия стандарта языка SQL принята в 1992 г. (Официальное название стандарта - Международный стандарт языка баз данных SQL (1992) (International Standart Database Language SQL), неофициальное название - SQL/92, или SQL-92, или SQL2). Документ, описывающий стандарт, содержит более 600 страниц. Мы дадим только некоторые понятия языка.

Язык SQL стал фактически стандартным языком доступа к базам данных. Все СУБД, претендующие на название "реляционные", реализуют тот или иной диалект SQL. Многие нереляционные системы также имеют в настоящее время средства доступа к реляционным данным. Целью стандартизации является переносимость приложений между различными СУБД.

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

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

Стандарт языка SQL, хотя и основан на реляционной теории, но во многих местах отходит он нее. Например, отношение в реляционной модели данных не допускает наличия одинаковых кортежей, а таблицы в терминологии SQL могут иметь одинаковые строки. Имеются и другие отличия.

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

## Операторы SQL

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

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

# Oператоры DDL (Data Definition Language) - операторы определения объектов базы данных

- CREATE SCHEMA создать схему базы данных
- DROP SHEMA удалить схему базы данных
- CREATE TABLE создать таблицу

- ALTER TABLE изменить таблицу
- DROP TABLE удалить таблицу
- CREATE DOMAIN создать домен
- ALTER DOMAIN изменить домен
- DROP DOMAIN удалить домен
- CREATE COLLATION создать последовательность
- DROP COLLATION удалить последовательность
- CREATE VIEW создать представление
- DROP VIEW удалить представление

# Операторы DML (Data Manipulation Language) - операторы манипулирования ланными

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

## Операторы защиты и управления данными

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

Кроме того, есть группы операторов установки параметров сеанса, получения информации о базе данных, операторы статического SQL, операторы динамического SQL.

Наиболее важными для пользователя являются операторы манипулирования данными (DML).

# Примеры использования операторов манипулирования данными

## INSERT - вставка строк в таблицу

Пример 1. Вставка одной строки в таблицу:

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

<u>Пример 2</u>. Вставка в таблицу нескольких строк, выбранных из другой таблицы (в таблицу TMP\_TABLE вставляются данные о поставщиках из таблицы P, имеющие номера, большие 2):

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

#### UPDATE - обновление строк в таблице

# **Пример 3**. Обновление нескольких строк в таблице:

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

## DELETE - удаление строк в таблице

Пример 4. Удаление нескольких строк в таблице:

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

## Пример 5. Удаление всех строк в таблице:

```
DELETE FROM P;
```

## Примеры использования оператора SELECT

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

Оператор SELECT всегда выполняется над некоторыми таблицами, входящими в базу данных.

Замечание. На самом деле в базах данных могут быть не только постоянно хранимые таблицы, а также временные таблицы и так называемые представления. Представления - это просто хранящиеся в базе данные SELECT-выражения. С точки зрения пользователей представления - это таблица, которая не хранится постоянно в базе данных, а "возникает" в момент обращения к ней. С точки зрения оператора SELECT и постоянно хранимые таблицы, и временные таблицы и представления выглядят совершенно одинаково. Конечно, при реальном выполнении оператора SELECT системой учитываются различия между хранимыми таблицами и представлениями, но эти различия скрыты от пользователя.

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

# Отбор данных из одной таблицы

<u>Пример 6</u>. Выбрать все данные из таблицы поставщиков (ключевые слова SELECT... FROM...):

```
SELECT * FROM P;
```

<u>Замечание</u>. В результате получим новую таблицу, содержащую полную копию данных из исходной таблицы Р.

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

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

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

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

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

<u>Замечание</u>. В результате получим таблицу с одной колонкой, содержащую все наименования поставшиков.

<u>Замечание</u>. Если в исходной таблице присутствовало несколько поставщиков с разными номерами, но одинаковыми наименованиями, то в результатирующей таблице *будут строки с повторениями* - дубликаты строк автоматически не отбрасываются.

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

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

<u>Замечание</u>. Использование ключевого слова DISTINCT приводит к тому, что в результатирующей таблице будут удалены все повторяющиеся строки.

<u>Пример 10</u>. Использование скалярных выражений и переименований колонок в запросах (ключевое слово AS...):

```
SELECT

TOVAR.TNAME,

TOVAR.KOL,

TOVAR.PRICE,

"=" AS EQU,

TOVAR.KOL*TOVAR.PRICE AS SUMMA
FROM TOVAR;
```

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

TNAME	KOL	PRICE	EQU	SUMMA
Болт	10	100	=	1000
Гайка	20	200	=	4000

Винт	30	300	=	9000
Dilli		300	=	9000

<u>Пример 11</u>. Упорядочение результатов запроса (ключевое слово *ORDER BY*...):

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

В результате получим следующую таблицу, упорядоченную по полю DNUM:

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

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

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

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

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

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

# Отбор данных из нескольких таблиц

<u>Пример 13</u>. Естественное соединение таблиц (способ 1 - явное указание условий соединения):

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

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

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

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

<u>Пример 14</u>. Естественное соединение таблиц (способ 2 - ключевые слова *JOIN*... USING...):

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

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

# <u>Пример 15</u>. Естественное соединение таблиц (способ 3 - ключевое слово NATURAL *JOIN*):

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

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

# <u>Пример 16</u>. Естественное соединение трех таблиц:

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

В результате получим следующую таблицу:

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

# <u>Пример 17</u>. Прямое произведение таблиц:

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

В результате получим следующую таблицу:

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

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

<u>Пример 18</u>. Соединение таблиц по произвольному условию. Рассмотрим таблицы поставщиков и деталей, которыми присвоен некоторый статую (см. пример 8 из предыдущей главы):

PNUM	PNAME	<b>PSTATUS</b>
1	Иванов	4
2	Петров	1
3	Сидоров	2

Таблица 1 Отношение Р (Поставщики)

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

Таблица 2 Отношение D (Детали)

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

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

В результате получим следующую таблицу:

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

## Использование имен корреляции (алиасов, псевдонимов)

Иногда приходится выполнять запросы, в которых таблица соединяется сама с собой, или одна таблица соединяется дважды с другой таблицей. При этом используются *имена корреляции* (*алиасы*, *псевдонимы*), которые позволяют различать соединяемые копии таблиц. Имена корреляции вводятся в разделе FROM и идут через пробел после имени

таблицы. Имена корреляции должны использоваться в качестве префикса перед именем столбца и отделяются от имени столбца точкой. Если в запросе указываются одни и те же поля из разных экземпляров одной таблицы, они должны быть переименованы для устранения неоднозначности в именованиях колонок результатирующей таблицы. Определение имени корреляции действует только во время выполнения запроса.

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

```
SELECT
P1.PNAME AS PNAME1,
P1.PSTATUS AS PSTATUS1,
P2.PNAME AS PNAME2,
P2.PSTATUS AS PSTATUS2
FROM
P P1, P P2
WHERE P1.PSTATUS1 > P2.PSTATUS2;
```

В результате получим следующую таблицу:

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

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

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

Таблица 3 Отношение CONTRAGENTS

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

Таблица 4 Отношение DETAILS (Детали)

Номер поставщика	Номер получателя	Номер детали	Поставляемое количество
PNUM	CNUM	DNUM	VOLUME

1	2	1	100
1	3	2	200
1	3	3	300
2	3	1	150
2	3	2	250
3	1	1	1000

Таблица 5 Отношение СD (Поставки)

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

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

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

В результате получим следующую таблицу:

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

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

```
SELECT
P.NAME AS PNAME,
C.NAME AS CNAME,
DETAILS.DNAME,
CD.VOLUME
```

```
FROM
CONTRAGENTS P,
CONTRAGENTS C,
DETAILS NATURAL JOIN CD
WHERE
P.NUM = CD.PNUM AND
C.NUM = CD.CNUM;
```

# Использование агрегатных функций в запросах

<u>Пример 21</u>. Получить общее количество поставщиков (ключевое слово COUNT):

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

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



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

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

В результате получим следующую таблицу с одной строкой:

SM	MX	MN	AV
2000	1000	100	333.33333333

#### Использование агрегатных функций с группировками

<u>Пример 23</u>. Для каждой детали получить суммарное поставляемое количество (ключевое слово *GROUP BY*...):

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

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

DNUM	SM
1	1250

2	450
3	300

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

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

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

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

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

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

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

В результате получим следующую таблицу:

DNUM	SM
1	1250
2	450

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

#### Использование подзапросов

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

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

```
SELECT *
  FROM P
  WHERE P.STATYS <
     (SELECT MAX(P.STATUS)
        FROM P);</pre>
```

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

<u>Замечание</u>. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

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

<u>Пример 26</u>. Использование предиката IN. Получить список поставщиков, поставляющих деталь номер 2:

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

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

<u>Замечание</u>. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

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

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

```
SELECT *
FROM P
WHERE EXIST
(SELECT *
FROM PD
```

```
WHERE

PD.PNUM = P.PNUM AND

PD.DNUM = 2);
```

<u>Замечание</u>. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

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

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

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

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

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

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

## **Пример 29**. Получить имена поставщиков, поставляющих все детали:

```
SELECT DISTINCT PNAME

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

<u>Замечание</u>. Данный запрос содержит два вложенных подзапроса и реализует реляционную операцию *деления отношений*.

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

# Использование объединения, пересечения и разности

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

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

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

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

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

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

```
SELECT P.PNAME
FROM P
WHERE P.STATUS > 3
```

```
EXCEPT

SELECT P.PNAME

FROM P, PD

WHERE P.PNUM = PD.PNUM AND

PD.DNUM = 2;
```

# Синтаксис оператора выборки данных (SELECT)

#### BNF-нотация

Опишем синтаксис оператора выборки данных (оператора SELECT) более точно. При описании синтаксиса операторов обычно используются условные обозначения, известные как  $\it cmandapmhile формы Бэкуса-Haypa (BNF)$ .

В BNF обозначениях используются следующие элементы:

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

# Синтаксис оператора выборки

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

```
Оператор выборки ::=
Табличное выражение
[ORDER BY
{{Имя столбца-результата [ASC | DESC]} | {Положительное целое [ASC | DESC]}}.,..];
Табличное выражение ::=
Select-выражение
[
{UNION | INTERSECT | EXCEPT} [ALL]
```

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

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

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

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

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

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

```
Функция агрегирования ::=
COUNT (*) |
{
```

# {COUNT | MAX | MIN | SUM | AVG} ([ALL | DISTINCT] Скалярное выражение) }

Конструктор значений таблицы ::=

VALUES Конструктор значений строки.,..

Конструктор значений строки ::=

Элемент конструктора | (Элемент конструктора.,..) | Select-выражение

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

Элемент конструктора ::=

Выражение для вычисления значения | NULL | DEFAULT

## Синтаксис соединенных таблиц

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

Соединенная таблица ::=

Перекрестное соединение

Естественное соединение

| Соединение посредством предиката

| Соединение посредством имен столбцов

Соединение объединения

Тип соединения ::=

**INNER** 

| LEFT [OUTER]

| RIGTH [OUTER]

| FULL [OUTER]

Перекрестное соединение ::=

Таблица A CROSS JOIN Таблица В

Естественное соединение ::=

Таблица A [NATURAL] [Тип соединения] JOIN Таблица В

Соединение посредством предиката ::=

*Таблица А* [*Tun соединения*] **JOIN** *Таблица В* **ON** Предикат

Соединение посредством имен столбцов ::=

Таблица А [Тип соединения] **JOIN** Таблица В **USING** (Имя столбца.,..)

Соединение объединения ::=

Таблица A UNION JOIN Таблица В

Опишем используемые термины.

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

NATURAL JOIN - Естественное соединение производится по всем столбцам таблиц A и B, имеющим одинаковые имена. В результатирующую таблицу одинаковые столбцы вставляются только один раз.

JOIN ... ON - Соединение посредством предиката соединяет строки таблиц A и B посредством указанного предиката.

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

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

INNER - Тип соединения "внутреннее". Внутренний тип соединения используется по умолчанию, когда тип явно не задан. В таблицах A и B соединяются только те строки, для которых найдено совпадение.

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

RIGHT (OUTER) - Тип соединения "правое (внешнее)". Правое соединение таблиц A и B включает в себя все строки из правой таблицы B и те строки из левой таблицы A, для которых обнаружено совпадение. Для строк из таблицы B, для которых не найдено соответствия в таблице A, в столбцы, извлекаемые из таблицы A заносятся значения NULL.

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

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

Использование соединенных таблиц часто облегчает восприятие оператора SELECT, особенно, когда используется естественное соединение. Если не использовать соединенные таблицы, то при выборе данных из нескольких таблиц необходимо явно указывать условия соединения в разделе WHERE. Если при этом пользователь указывает сложные критерии отбора строк, то в разделе WHERE смешиваются семантически различные понятия - как условия связи таблиц, так и условия отбора строк (см. примеры 13, 14, 15 данной главы).

# Синтаксис условных выражений раздела WHERE

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

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

```
Условное выражение ::=
[ ( ] [NOT]
{Предикат сравнения
| Предикат between
| Предикат іп
| Предикат like
| Предикат null
Предикат количественного сравнения
| Предикат exist
Предикат ипідие
Предикат match
| Предикат overlaps}
[{AND | OR} Условное выражение][)]
[IS [NOT] {TRUE | FALSE | UNKNOWN}]
Предикат сравнения ::=
Конструктор значений строки \{= |<|>|<=|>>\} Конструктор значений строки
Пример 33. Сравнение поля таблицы и скалярного значения:
POSTAV.VOLUME > 100
```

# Пример 34. Сравнение двух сконструированных строк:

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

Этот пример эквивалентен условному выражению

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

Предикат between ::=

Конструктор значений строки [NOT] **BETWEEN** 

Конструктор значений строки AND Конструктор значений строки

# Пример 35. PD. VOLUME BETWEEN 10 AND 100

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

## Пример 36.

## Пример 37.

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

Предикат like ::=

Выражение для вычисления значения строки-поиска [NOT] LIKE Выражение для вычисления значения строки-шаблона [ESCAPE Символ]

<u>Замечание</u>. Предикат LIKE производит поиск строки-поиска в строке-шаблоне. В строкешаблоне разрешается использовать два трафаретных символа:

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

Предикат null ::=

Конструктор значений строки IS [NOT] NULL

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

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

<u>Замечание</u>. Кванторы ANY и SOME являются синонимами и полностью взаимозаменяемы.

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

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

# Пример 38.

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

Предикат exist ::=

EXIST (Select-выражение)

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

Предикат unique ::= UNIQUE (Select-выражение)

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

Предикат match ::=
Конструктор значений строки **MATCH** [**UNIQUE**]

[PARTIAL | FULL] (Select-выражение)

<u>Замечание</u>. Предикат МАТСН проверяет, будет ли значение, определенное в конструкторе строки совпадать со значением любой строки, полученной в результате подзапроса.

Предикат overlaps ::=

Конструктор значений строки OVERLAPS Конструктор значений строки

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

# Порядок выполнения оператора SELECT

Для того чтобы понять, как получается результат выполнения оператора SELECT, рассмотрим концептуальную схему его выполнения. Эта схема является именно концептуальной, т.к. гарантируется, что результат будет таким, как если бы он выполнялся шаг за шагом в соответствии с этой схемой. На самом деле, реально результат получается более изощренными алгоритмами, которыми "владеет" конкретная СУБД.

# Стадия 1. Выполнение одиночного оператора SELECT

Если в операторе присутствуют ключевые слова UNION, EXCEPT и INTERSECT, то запрос разбивается на несколько независимых запросов, каждый из которых выполняется отдельно:

*Шаг 1 (FROM)*. Вычисляется прямое декартовое произведение всех таблиц, указанных в обязательном разделе FROM. В результате шага 1 получаем таблицу А.

Шаг 2 (WHERE). Если в операторе SELECT присутствует раздел WHERE, то сканируется таблица A, полученная при выполнении шага 1. При этом для каждой строки из таблицы A вычисляется условное выражение, приведенное в разделе WHERE. Только те строки, для которых условное выражение возвращает значение TRUE, включаются в результат. Если раздел WHERE опущен, то сразу переходим к шагу 3. Если в условном выражении участвуют вложенные подзапросы, то они вычисляются в соответствии с данной концептуальной схемой. В результате шага 2 получаем таблицу В.

*Шаг 3 (GROUP BY)*. Если в операторе SELECT присутствует раздел GROUP BY, то строки таблицы B, полученной на втором шаге, группируются в соответствии со списком группировки, приведенным в разделе GROUP BY. Если раздел GROUP BY опущен, то сразу переходим к шагу 4. В результате шага 3 получаем таблицу C.

*Шаг 4 (HAVING)*. Если в операторе SELECT присутствует раздел HAVING, то группы, не удовлетворяющие условному выражению, приведенному в разделе HAVING, исключаются. Если раздел HAVING опущен, то сразу переходим к шагу 5. В результате шага 4 получаем таблицу D.

Шаг 5 (SELECT). Каждая группа, полученная на шаге 4, генерирует одну строку результата следующим образом. Вычисляются все скалярные выражения, указанные в разделе SELECT. По правилам использования раздела GROUP BY, такие скалярные выражения должны быть одинаковыми для всех строк внутри каждой группы. Для каждой группы вычисляются значения агрегатных функций, приведенных в разделе SELECT. Если раздел GROUP BY отсутствовал, но в разделе SELECT есть агрегатные функции, то считается, что имеется всего одна группа. Если нет ни раздела GROUP BY, ни агрегатных функций, то считается, что имеется столько групп, сколько строк отобрано к данному моменту. В результате шага 5 получаем таблицу Е, содержащую столько колонок, сколько элементов приведено в разделе SELECT и столько строк, сколько отобрано групп.

# Стадия 2. Выполнение операций UNION, EXCEPT, INTERSECT

Если в операторе SELECT присутствовали ключевые слова UNION, EXCEPT и INTERSECT, то таблицы, полученные в результате выполнения 1-й стадии, объединяются, вычитаются или пересекаются.

# Стадия 3. Упорядочение результата

Если в операторе SELECT присутствует раздел ORDER BY, то строки полученной на предыдущих шагах таблицы упорядочиваются в соответствии со списком упорядочения, приведенном в разделе ORDER BY.

## Как на самом деле выполняется оператор SELECT

Если внимательно рассмотреть приведенный выше концептуальный алгоритм вычисления результата оператора SELECT, то сразу понятно, что выполнять его непосредственно в таком виде чрезвычайно накладно. Даже на самом первом шаге, когда вычисляется декартово произведение таблиц, приведенных в разделе FROM, может получиться таблица огромных размеров, причем практически большинство строк и колонок из нее будет отброшено на следующих шагах.

На самом деле в РСУБД имеется *оптимизатор*, функцией которого является нахождение такого *оптимального алгоритма* выполнения запроса, который гарантирует получение правильного результата.

Схематично работу оптимизатора можно представить в виде последовательности нескольких шагов:

*Шаг 1 (Синтаксический анализ)*. Поступивший запрос подвергается синтаксическому анализу. На этом шаге определяется, правильно ли вообще (с точки зрения синтаксиса SQL) сформулирован запрос. В ходе синтаксического анализа вырабатывается некоторое внутренне представление запроса, используемое на последующих шагах.

Шаг 2 (Преобразование в каноническую форму). Запрос во внутреннем представлении подвергается преобразованию в некоторую каноническую форму. При преобразовании к канонической форме используются как синтаксические, так и семантические преобразования. Синтаксические преобразования (например, приведения логических выражений к конъюнктивной или дизъюнктивной нормальной форме, замена выражений "x AND NOT x" на "FALSE", и т.п.) позволяют получить новое внутренне представление запроса, синтаксически эквивалентное исходному, но стандартное в некотором смысле. Семантические преобразования используют дополнительные знания, которыми владеет

система, например, ограничения целостности. В результате семантических преобразований получается запрос, синтаксически *не эквивалентный* исходному, но дающий *тот же самый результат*.

Шаг 3 (Генерация планов выполнения запроса и выбор оптимального плана). На этом шаге оптимизатор генерирует множество возможных планов выполнения запроса. Каждый план строится как комбинация низкоуровневых процедур доступа к данным из таблиц, методам соединения таблиц. Из всех сгенерированных планов выбирается план, обладающий минимальной стоимостью. При этом анализируются данные о наличии индексов у таблиц, статистических данных о распределении значений в таблицах, и т.п. Стоимость плана это, как правило, сумма стоимостей выполнения отдельных низкоуровневых процедур, которые используются для его выполнения. В стоимость выполнения отдельной процедуры могут входить оценки количества обращений к дискам, степень загруженности процессора и другие параметры.

*Шаг 4.* (*Выполнение плана запроса*). На этом шаге план, выбранный на предыдущем шаге, передается на реальное выполнение.

Во многом качество конкретной СУБД определяется качеством ее оптимизатора. Хороший оптимизатор может повысить скорость выполнения запроса на несколько порядков. Качество оптимизатора определяется тем, какие методы преобразований он может использовать, какой статистической и иной информацией о таблицах он располагает, какие методы для оценки стоимости выполнения плана он знает.

# Реализация реляционной алгебры средствами оператора SELECT (Реляционная полнота SQL)

Для того, чтобы показать, что язык SQL является реляционно полным, нужно показать, что любой реляционный оператор может быть выражен средствами SQL. На самом деле достаточно показать, что средствами SQL можно выразить любой из *примитивных* реляционных операторов.

# Оператор декартового произведения

```
      Реляционная алгебра: A T IM ES B

      Оператор SQL:

      SELECT A.Поле1, A.Поле2, ..., B.Поле1, B.Поле2, ...

      FROM A, B;

      или

      SELECT A.Поле1, A.Поле2, ..., B.Поле1, B.Поле2, ...

      FROM A CROSS JOIN B;
```

## Оператор проекции

Реляционная алгебра: A[X,Y,...,Z]

Оператор SQL:

```
SELECT DISTINCT X, Y, ..., Z
FROM A;
```

# Оператор выборки

```
Реляционная алгебра: A \text{ WHERE } c,
```

Оператор SQL:

```
SELECT *
FROM A
WHERE c;
```

# Оператор объединения

```
Реляционная алгебра: ^{A} UNION ^{B}
```

Оператор SQL:

```
SELECT *
FROM A
UNION
SELECT *
FROM B;
```

# Оператор вычитания

Реляционная алгебра: A MINUS B

```
Onepatop SQL:
SELECT *
FROM A
EXCEPT
SELECT *
FROM B
```

Реляционный оператор переименования RENAME выражается при помощи ключевого слова AS в списке отбираемых полей оператора SELECT. Таким образом, язык SQL является реляционно полным.

Остальные операторы реляционной алгебры (соединение, пересечение, деление) выражаются через примитивные, следовательно, могут быть выражены операторами SQL. Тем не менее, для практических целей приведем их.

# Оператор соединения

```
Реляционная алгебра: (ATIMES B) WHERE c
Оператор SQL:
SELECT A.Поле1, A.Поле2, ..., B.Поле1, В.Поле2, ...
FROM A, B
WHERE c;
```

```
SELECT A.Поле1, A.Поле2, ..., B.Поле1, B.Поле2, ...
FROM A CROSS JOIN B
WHERE c;
```

## Оператор пересечения

Реляционная алгебра: A INTERSECT B

## Оператор SQL:

```
SELECT *
FROM A
INTERSECT
SELECT *
FROM B;
```

# Оператор деления

Реляционная алгебра: A(X,Y) DEVIDBY B(Y)

# Оператор SQL:

```
SELECT DISTINCT A.X

FROM A
WHERE NOT EXIST
(SELECT *
FROM B
WHERE NOT EXIST
(SELECT *
FROM A A1
WHERE
A1.X = A.X AND
A1.Y = B.Y));
```

<u>Замечание</u>. Оператор SQL, реализующий деление отношений трудно запомнить, поэтому дадим пример эквивалентного преобразования выражений, представляющих суть запроса.

Пусть отношение A содержит данные о поставках деталей, отношение B содержит список всех деталей, которые могут поставляться. Атрибут X является номером поставщика, атрибут Y является номером детали.

Разделить отношение A на отношение B означает в данном примере "отобрать номера поставщиков, которые поставляют *все* детали".

Преобразуем текст выражения:

"Отобрать номера поставщиков, которые поставляют все детали" эквивалентно

"Отобрать те номера поставщиков из таблицы A, для которых *не существует* непоставляемых деталей в таблице B" эквивалентно

"Отобрать те номера поставщиков из таблицы A, для которых *не существует* тех номеров деталей из таблицы B, которые *не поставляются* этим поставщиком" эквивалентно

"Отобрать те номера поставщиков из таблицы A, для которых *не существует* тех номеров деталей из таблицы B, для которых *не существует* записей о поставках в таблице A для этого поставшика и этой детали".

Последнее выражение дословно переводится на язык SQL. При переводе выражения на язык SQL нужно учесть, что во внутреннем подзапросе таблица А должна быть переименована, для того чтобы отличать ее от экземпляра этой же таблицы, используемой во внешнем запросе.

#### Выводы

Фактически стандартным языком доступа к базам данных в настоящее время стал язык SQL (Structured Query Language).

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

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

Основу языка SQL составляют операторы, условно разбитые не несколько групп по выполняемым функциям:

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

Одним из основных операторов DML является оператор SELECT, позволяющий извлекать данные из таблиц и получать ответы на различные запросы. Оператор SELECT содержит в себе все возможности реляционной алгебры. Это означает, что любой оператор реляционной алгебры может быть выражен при помощи подходящего оператора SELECT. Этим доказывается реляционная полнота языка SQL.

Различают концептуальную схему выполнения оператора SELECT и фактическую схему его выполнения. Концептуальная схема описывает, в какой логической последовательности должны выполняться операции, чтобы получить результат. При реальном выполнении оператора SELECT на первый план выступает достижение максимальной скорости выполнения запроса. Для этого используется *оптимизатор*, который, анализируя различные планы выполнения запроса, выбирает наилучший из них.

# Глава 6. Нормальные формы отношений

# Этапы разработки базы данных

Целью разработки любой базы данных является хранение и использование информации о какой-либо предметной области. Для реализации этой цели имеются следующие инструменты:

- 1. Реляционная модель данных удобный способ представления данных предметной области.
- 2. Язык SQL универсальный способ манипулирования такими данными.

Однако очевидно, что для одной и той же предметной области реляционные отношения можно спроектировать множеством различных способов. Например, можно спроектировать несколько отношений с большим количеством атрибутов, или наоборот, разнести все атрибуты по большому числу мелких отношений. Как определить, по каким признакам нужно помещать атрибуты в те или иные отношения?

В данной главе рассматриваются способы "хорошего" или "правильного" проектирования реляционных отношений. Сначала мы обсудим, что значит "хорошие" или "правильные" модели данных. Потом будут введены понятия первой, второй и третьей нормальных форм отношений ( $1H\Phi$ ,  $2H\Phi$ ,  $3H\Phi$ ) и показано, что "хорошими" являются отношения в третьей нормальной форме.

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

- Сама предметная область
- Модель предметной области
- Логическая модель данных
- Физическая модель данных
- Собственно база данных и приложения

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

Модель предметной области. Знания могут быть как в виде неформальных знаний в мозгу эксперта, так и выражены формально при помощи каких-либо средств. В качестве таких средств могут выступать текстовые описания предметной области, наборы должностных инструкций, правила ведения дел в компании и т.п. Опыт показывает, что текстовый способ представления модели предметной области крайне неэффективен. Гораздо более информативными и полезными при разработке баз данных являются описания предметной области, выполненные при помощи специализированных графических нотаций. Имеется большое количество методик описания предметной области. Из наиболее известных можно назвать методику структурного анализа SADT и основанную на нем IDEFO, диаграммы потоков данных Гейна-Сарсона, методику объектно-ориентированного анализа UML, и др. Модель предметной области описывает скорее процессы, происходящие в предметной области и данные, используемые этими процессами. От того, насколько правильно смоделирована предметная область, зависит успех дальнейшей разработки приложений.

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

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но без привязки к конкретной СУБД. Более того, логическая модель данных необязательно должна быть выражена средствами именно реляционной модели данных. Основным средством разработки логической модели данных в настоящий момент являются различные варианты **ER-диаграмм** (Entity-Relationship, диаграммы сущность-связь). Одну и ту же ER-модель можно преобразовать как в реляционную модель данных, так и в модель данных для иерархических и сетевых СУБД, или в постреляционную модель данных. Однако, т.к. мы рассматриваем именно реляционные СУБД, то можно считать, что логическая модель данных для нас формулируется в терминах реляционной модели данных.

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

При разработке логической модели данных возникают вопросы: хорошо ли спроектированы отношения? Правильно ли они отражают модель предметной области, а следовательно и саму предметную область?

Физическая модель данных. На еще более низком уровне находится физическая модель данных. Физическая модель данных описывает данные средствами конкретной СУБД. Мы будем считать, что физическая модель данных реализована средствами именно реляционной СУБД, хотя, как уже сказано выше, это необязательно. Отношения, разработанные на стадии формирования логической модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц, для ключевых атрибутов создаются уникальные индексы, домены преображаются в типы данных, принятые в конкретной СУБД.

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

дополнительно объявить различные индексы, повышающие скорость обращения к данным. Многое тут зависит от конкретной СУБД.

При разработке физической модели данных возникают вопросы: хорошо ли спроектированы таблицы? Правильно ли выбраны индексы? Насколько много программного кода в виде триггеров и хранимых процедур необходимо разработать для поддержания целостности данных?

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

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

Таким образом ясно, что решения, принятые на каждом этапе моделирования и разработки базы данных, будут сказываться на дальнейших этапах. Поэтому особую роль играет принятие правильных решений *на ранних этапах моделирования*.

## Критерии оценки качества логической модели данных

Цель данной главы - описать некоторые принципы построения *хороших логических моделей данных*. Хороших в том смысле, что решения, принятые в процессе логического проектирования приводили бы к хорошим физическим моделям и в конечном итоге к хорошей работе базы данных.

Для того чтобы оценить качество принимаемых решений на уровне логической модели данных, необходимо сформулировать некоторые критерии качества в терминах физической модели и конкретной реализации и посмотреть, как различные решения, принятые в процессе *погического* моделирования, влияют на качество физической модели и на скорость работы базы данных.

Конечно, таких критериев может быть очень много и выбор их в достаточной степени произволен. Мы рассмотрим некоторые из таких критериев, которые являются безусловно важными с точки зрения получения качественной базы данных:

- Адекватность базы данных предметной области
- Легкость разработки и сопровождения базы данных
- Скорость выполнения операций обновления данных (вставка, обновление, удаление кортежей)
- Скорость выполнения операций выборки данных

# Адекватность базы данных предметной области

База данных должна адекватно отражать предметную область. Это означает, что должны выполняться следующие условия:

- 1. Состояние базы данных в каждый момент времени должно соответствовать состоянию предметной области.
- 2. Изменение состояния предметной области должно приводить к соответствующему изменению состояния базы данных
- 3. Ограничения предметной области, отраженные в модели предметной области, должны некоторым образом отражаться и учитываться базе данных.

# Легкость разработки и сопровождения базы данных

Практически любая база данных, за исключением совершенно элементарных, содержит некоторое количество программного кода в виде триггеров и хранимых процедур.

**Хранимые процедуры** - это процедуры и функции, хранящиеся непосредственно в базе данных в откомпилированном виде и которые могут запускаться пользователями или приложениями, работающими с базой данных. Хранимые процедуры обычно пишутся либо на специальном процедурном расширении языка SQL (например, PL/SQL для ORACLE или Transact-SQL для MS SQL Server), или на некотором универсальном языке программирования, например, C++, с включением в код операторов SQL в соответствии со специальными правилами такого включения. Основное назначение хранимых процедур - реализация бизнес-процессов предметной области.

Триггеры - это хранимые процедуры, связанные с некоторыми событиями, происходящими во время работы базы данных. В качестве таких событий выступают операции вставки, обновления и удаления строк таблиц. Если в базе данных определен некоторый триггер, то он запускается автоматически всегла при возникновении события, с которым этот триггер связан. Очень важным является то, что пользователь не может обойти триггер. Триггер срабатывает независимо от того, кто из пользователей и каким способом инициировал событие, вызвавшее запуск триггера. Таким образом, основное назначение триггеров - автоматическая поддержка целостности базы данных. Триггеры могут быть как достаточно простыми, например, поддерживающими ссылочную целостность, так и довольно сложными, реализующими какие-либо сложные ограничения предметной области или сложные действия, которые должны произойти при наступлении некоторых событий. Например, с операцией вставки нового товара в накладную может быть связан триггер, который выполняет следующие действия - проверяет, есть ли необходимое количество товара, при наличии товара добавляет его в накладную и уменьшает данные о наличии товара на складе, при отсутствии товара формирует заказ на поставку недостающего товара и тут же посылает заказ по электронной почте поставщику.

Очевидно, что чем больше программного кода в виде триггеров и хранимых процедур содержит база данных, тем сложнее ее разработка и дальнейшее сопровождение.

# Скорость операций обновления данных (вставка, обновление, удаление)

На уровне логического моделирования мы определяем реляционные отношения и атрибуты этих отношений. На этом уровне мы не можем определять какие-либо физические структуры хранения (индексы, хеширование и т.п.). Единственное, чем мы можем управлять - это распределением атрибутов по различным отношениям. Можно описать мало отношений с большим количеством атрибутов, или много отношений, каждое из которых содержит мало атрибутов. Таким образом, необходимо попытаться ответить на вопрос - влияет ли количество отношений и количество атрибутов в отношениях на скорость выполнения операций обновления данных. Такой вопрос, конечно, не является достаточно корректным, т.к. скорость выполнения операций с базой

данных сильно зависит от физической реализации базы данных. Тем не менее, попытаемся качественно оценить это влияние при одинаковых подходах к физическому моделированию.

Основными операциями, изменяющими состояние базы данных, являются операции вставки, обновления и удаления записей. В базах данных, требующих постоянных изменений (складской учет, системы продаж билетов и т.п.) производительность определяется скоростью выполнения большого количества небольших операций вставки, обновления и удаления.

Рассмотрим операцию вставки записи в таблицу. Вставка записи производится в одну из свободных страниц памяти, выделенной для данной таблицы. СУБД постоянно хранит информацию о наличии и расположении свободных страниц. Если для таблицы не созданы индексы, то операция вставки выполняется фактически с одинаковой скоростью независимо от размера таблицы и от количества атрибутов в таблице. Если в таблице имеются индексы, то при выполнении операции вставки записи индексы должны быть перестроены. Таким образом, скорость выполнения операции вставки уменьшается при увеличении количества индексов у таблицы и мало зависит от числа строк в таблице.

Рассмотрим операции обновления и удаления записей из таблицы. Прежде, чем обновить или удалить запись, ее необходимо найти. Если таблица не индексирована, то единственным способом поиска является последовательное сканирование таблицы в поиске нужной записи. В этом случае, скорость операций обновления и удаления существенно увеличивается с увеличением количества записей в таблице и не зависит от количества атрибутов. Но на самом деле неиндексированные таблицы практически никогда не используются. Для каждой таблицы обычно объявляется один или несколько индексов, соответствующий потенциальным ключам. При помощи этих индексов поиск записи производится очень быстро и практически не зависит от количества строк и атрибутов в таблице (хотя, конечно, некоторая зависимость имеется). Если для таблицы объявлено несколько индексов, то при выполнении операций обновления и удаления эти индексы должны быть перестроены, на что тратится дополнительное время. Таким образом, скорость выполнения операций обновления и удаления также уменьшается при увеличении количества индексов у таблицы и мало зависим от числа строк в таблице.

Можно предположить, что чем больше атрибутов имеет таблица, тем больше для нее будет объявлено индексов. Эта зависимость, конечно, не прямая, но *при одинаковых подходах* к физическому моделированию обычно так и происходит. Таким образом, можно принять допущение, что *чем больше атрибутов имеют отношения*, разработанные в ходе логического моделирования, *тем медленнее будут выполняться операции обновления данных*, за счет затраты времени на перестройку большего количества индексов.

Дополнительные соображения в пользу приведенного тезиса о замедлении выполнения операций обновления данных (влияние журнализации, длины строк таблиц) приведены в работе А.Прохорова [27].

## Скорость операций выборки данных

Одно из назначений базы данных - предоставление информации пользователям. Информация извлекается из реляционной базы данных при помощи оператора SQL - SELECT. Одной из наиболее дорогостоящих операций при выполнении оператора SELECT является операция соединение таблиц. Таким образом, чем больше

взаимосвязанных отношений было создано в ходе логического моделирования, тем больше вероятность того, что при выполнении запросов эти отношения будут соединяться, и, следовательно, тем медленнее будут выполняться запросы. Таким образом, увеличение количества отношений приводит к замедлению выполнения операций выборки данных, особенно, если запросы заранее неизвестны.

# Основной пример

Рассмотрим в качестве предметной области некоторую организацию, выполняющую некоторые проекты. Модель предметной области опишем следующим неформальным текстом:

- 1. Сотрудники организации выполняют проекты.
- 2. Проекты состоят из нескольких заданий.
- 3. Каждый сотрудник может участвовать в одном или нескольких проектах, или временно не участвовать ни в каких проектах.
- 4. Над каждым проектом может работать несколько сотрудников, или временно проект может быть приостановлен, тогда над ним не работает ни один сотрудник.
- 5. Над каждым заданием в проекте работает ровно один сотрудник.
- 6. Каждый сотрудник числится в одном отделе.
- 7. Каждый сотрудник имеет телефон, находящийся в отделе сотрудника.

В ходе дополнительного уточнения того, какие данные необходимо учитывать, выяснилось следующее:

- 1. О каждом сотруднике необходимо хранить табельный номер и фамилию. Табельный номер является уникальным для каждого сотрудника.
- 2. Каждый отдел имеет уникальный номер.
- 3. Каждый проект имеет номер и наименование. Номер проекта является уникальным.
- 4. Каждая работа из проекта имеет номер, уникальный в пределах проекта. Работы в разных проектах могут иметь одинаковые номера.

# 1НФ (Первая Нормальная Форма)

Понятие первой нормальной формы уже обсуждалось в главе 2. *Первая нормальная форма* ( $1H\Phi$ ) - это обычное отношение. Согласно нашему определению отношений, любое отношение автоматически уже находится в  $1H\Phi$ . Напомним кратко свойства отношений (это и будут свойства  $1H\Phi$ ):

- В отношении нет одинаковых кортежей.
- Кортежи не упорядочены.
- Атрибуты не упорядочены и различаются по наименованию.
- Все значения атрибутов атомарны.

В ходе логического моделирования на первом шаге предложено хранить данные в одном отношении, имеющем следующие атрибуты:

СОТРУДНИКИ\_ОТДЕЛЫ\_ПРОЕКТЫ ( $H\_COTP$ , ФАМ,  $H\_OTД$ , ТЕЛ,  $H\_\Pi PO$ , ПРОЕКТ,  $H\_3AДAH$ )

*H COTP* - табельный номер сотрудника

ФАМ - фамилия сотрудника

Н\_ОТД - номер отдела, в котором числится сотрудник

ТЕЛ - телефон сотрудника

 $H_{-}\Pi PO$  - номер проекта, над которым работает сотрудник

ПРОЕКТ - наименование проекта, над которым работает сотрудник

Н\_ЗАДАН - номер задания, над которым работает сотрудник

Т.к. каждый сотрудник в каждом проекте выполняет ровно одно задание, то в качестве потенциального ключа отношения необходимо взять пару атрибутов  $\{H\_COTP, H\_\Pi PO\}$ .

В текущий момент состояние предметной области отражается следующими фактами:

- Сотрудник Иванов, работающий в 1 отделе, выполняет в первом проекте "Космос" задание 1 и во втором проекте "Климат" задание 1.
- Сотрудник Петров, работающий в 1 отделе, выполняет в первом проекте "Космос" задание 2.
- Сотрудник Сидоров, работающий во 2 отделе, выполняет в первом проекте "Космос" задание 3 и во втором проекте "Климат" задание 2.

Это состояние отражается в таблице (курсивом выделены ключевые атрибуты):

H_COTP	ФАМ	н_отд	ТЕЛ	Н_ПРО	ПРОЕКТ	Н_ЗАДАН
1	Иванов	1	11- 22- 33	1	Космос	1
1	Иванов	1	11- 22- 33	2	Климат	1
2	Петров	1	11- 22- 33	1	Космос	2
3	Сидоров	2	33- 22- 11	1	Космос	3
3	Сидоров	2	33- 22- 11	2	Климат	2

Таблица 1 Отношение СОТРУДНИКИ\_ОТДЕЛЫ\_ПРОЕКТЫ

Аномалии обновления

Даже одного взгляда на таблицу отношения СОТРУДНИКИ\_ОТДЕЛЫ\_ПРОЕКТЫ достаточно, чтобы увидеть, что данные хранятся в ней с большой избыточностью. Во многих строках повторяются фамилии сотрудников, номера телефонов, наименования проектов. Кроме того, в данном отношении хранятся вместе независимые друг от друга данные - и данные о сотрудниках, и об отделах, и о проектах, и о работах по проектам. Пока никаких действий с отношением не производится, это не страшно. Но как только состояние предметной области изменяется, то, при попытках соответствующим образом изменить состояние базы данных, возникает большое количество проблем.

Исторически эти проблемы получили название *аномалии обновления*. Попытки дать строгое понятие аномалии в базе данных не являются вполне удовлетворительными [51, 7]. В данных работах аномалии определены как противоречие между моделью предметной области и физической моделью данных, поддерживаемых средствами конкретной СУБД. "Аномалии возникают в том случае, когда наши знания о предметной области оказываются, по каким-то причинам, невыразимыми в схеме БД или входящими в противоречие с ней" [7]. Мы придерживаемся другой точки зрения, заключающейся в том, что аномалий в смысле определений упомянутых авторов нет, а есть либо *неадекватность модели данных* предметной области, либо некоторые *дополнительные трудности в реализации ограничений* предметной области средствами СУБД. Более глубокое обсуждение проблемы строгого определения понятия аномалий выходит за пределы данной работы.

Таким образом, мы будем придерживаться интуитивного понятия аномалии как неадекватности модели данных предметной области, (что говорит на самом деле о том, что логическая модель данных попросту неверна!) или как необходимости дополнительных усилий для реализации всех ограничений определенных в предметной области (дополнительный программный код в виде триггеров или хранимых процедур).

Т.к. аномалии проявляют себя при выполнении операций, изменяющих состояние базы данных, то различают следующие виды аномалий:

- Аномалии вставки (INSERT)
- Аномалии обновления (UPDATE)
- Аномалии удаления (DELETE)

В отношении СОТРУДНИКИ\_ОТДЕЛЫ\_ПРОЕКТЫ можно привести примеры следующих аномалий:

## Аномалии вставки (INSERT)

В отношение **СОТРУДНИКИ\_ОТДЕЛЫ\_ПРОЕКТЫ** нельзя вставить данные о сотруднике, который пока не участвует ни в одном проекте. Действительно, если, например, во втором отделе появляется новый сотрудник, скажем, Пушников, и он пока не участвует ни в одном проекте, то мы должны вставить в отношение кортеж (4, Пушников, 2, 33-22-11, null, null, null). Это сделать невозможно, т.к. атрибут *Н\_ПРО* (номер проекта) входит в состав потенциального ключа, и, следовательно, не может содержать null-значений.

Точно также нельзя вставить данные о проекте, над которым пока не работает ни один сотрудник.

<u>Причина аномалии</u> - хранение в одном отношении разнородной информации (и о сотрудниках, и о проектах, и о работах по проекту).

<u>Вывод</u> - логическая модель данных неадекватна модели предметной области. База данных, основанная на такой модели, будет работать неправильно.

# Аномалии обновления (UPDATE)

Фамилии сотрудников, наименования проектов, номера телефонов повторяются во многих кортежах отношения. Поэтому если сотрудник меняет фамилию, или проект меняет наименование, или меняется номер телефона, то такие изменения необходимо одновременно выполнить во всех местах, где эта фамилия, наименование или номер телефона встречаются, иначе отношение станет некорректным (например, один и тот же проект в разных кортежах будет называться по-разному). Таким образом, обновление базы данных одним действием реализовать невозможно. Для поддержания отношения в целостном состоянии необходимо написать триггер, который при обновлении одной записи корректно исправлял бы данные и в других местах.

<u>Причина аномалии</u> - избыточность данных, также порожденная тем, что в одном отношении хранится разнородная информация.

<u>Вывод</u> - увеличивается сложность разработки базы данных. База данных, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

# Аномалии удаления (DELETE)

При удалении некоторых данных может произойти потеря другой информации. Например, если закрыть проект "Космос" и удалить все строки, в которых он встречается, то будут потеряны все данные о сотруднике Петрове. Если удалить сотрудника Сидорова, то будет потеряна информация о том, что в отделе номер 2 находится телефон 33-22-11. Если по проекту временно прекращены работы, то при удалении данных о работах по этому проекту будут удалены и данные о самом проекте (наименование проекта). При этом если был сотрудник, который работал только над этим проектом, то будут потеряны и данные об этом сотруднике.

<u>Причина аномалии</u> - хранение в одном отношении разнородной информации (и о сотрудниках, и о проектах, и о работах по проекту).

<u>Вывод</u> - логическая модель данных неадекватна модели предметной области. База данных, основанная на такой модели, будет работать неправильно.

## Функциональные зависимости

Отношение **СОТРУДНИКИ\_ОТДЕЛЫ\_ПРОЕКТЫ** находится в 1НФ, при этом, как было показано выше, логическая модель данных не адекватна модели предметной области. Таким образом, первой нормальной формы *недостаточно* для правильного моделирования данных.

# Определение функциональной зависимости

Для устранения указанных аномалий (а на самом деле *для правильного проектирования модели данных*!) применяется метод нормализации отношений. Нормализация основана на понятии функциональной зависимости атрибутов отношения.

Определение 1. Пусть R- отношение. Множество атрибутов Y функционально зависимо от множества атрибутов X (X функционально определяем Y) тогда и только тогда, когда для любого состояния отношения R для любых кортежей  $r_1, r_2 \in R$  из того, что  $r_1 X = r_2 X$  следует что  $r_1 Y = r_2 Y$  (т.е. во всех кортежах, имеющих одинаковые значения атрибутов X, значения атрибутов Y также совпадают G любом состоянии отношения G0. Символически функциональная зависимость записывается

$$X \to Y$$

Множество атрибутов X называется **детерминантом функциональной зависимости**, а множество атрибутов Y называется **зависимой частью**.

<u>Замечание</u>. Если атрибуты X составляют потенциальный ключ отношения X, то *любой* атрибут отношения X функционально зависит от X.

<u>Пример 1</u>. В отношении **СОТРУДНИКИ\_ОТДЕЛЫ\_ПРОЕКТЫ** можно привести следующие примеры функциональных зависимостей:

Зависимость атрибутов от ключа отношения:

 $\{H\_COTP, H\_\Pi PO\} \rightarrow \Phi AM$ 

 $\{H\_COTP, H\_\Pi PO\} \rightarrow H\_OTД$ 

 $\{H\_COTP, H\_\Pi PO\} \rightarrow$  ТЕЛ

 $\{H \ COTP, H \ \Pi PO\} \rightarrow \Pi POEKT$ 

 $\{H\_COTP, H\_\Pi PO\} \rightarrow$  H\_ЗАДАН

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:

H COTP  $\rightarrow \Phi$ AM

 $H\_COTP \rightarrow H\_OTД$ 

 $H_{-}COTP \rightarrow$ ТЕЛ

Зависимость наименования проекта от номера проекта:

H ПРО →ПРОЕКТ

Зависимость номера телефона от номера отдела:

 $H_{O}TJ \rightarrow TEJ$ 

Замечание. Приведенные функциональные зависимости не выведены из внешнего вида отношения, приведенного в таблице 1. Эти зависимости отражают взаимосвязи, обнаруженные между объектами предметной области и являются дополнительными ограничениями, определяемыми предметной областью. Таким образом, функциональная зависимость - семантическое понятие. Она возникает, когда по значениям одних данных в предметной области можно определить значения других данных. Например, зная табельный номер сотрудника, можно определить его фамилию, по номеру отдела можно определить телефона. Функциональная зависимость задает дополнительные ограничения на данные, которые могут храниться в отношениях. Для корректности базы данных (адекватности предметной области) необходимо при выполнении операций модификации базы данных проверять все ограничения, определенные функциональными зависимостями.

# Функциональные зависимости отношений и математическое понятие функциональной зависимости

Функциональная зависимость атрибутов отношения напоминает понятие функциональной зависимости в математике. *Но это не одно и то же.* Для сравнения напомним математическое понятие функциональной зависимости:

Определение 2. Функциональная зависимость ( $\pmb{\phi}$ ункция) - это тройка объектов  ${\{X,Y,f\}}$  , где

X - множество (область определения),

У - множество (*множество* значений),

f - правило, согласно которому каждому элементу  $x \in X$  ставится в соответствие один и только один элемент  $y \in Y$  (правило функциональной зависимости).

Функциональная зависимость обычно обозначается как  $f: X \to Y$  или y = f(x).

<u>Замечание</u>. Правило f может быть задано любым способом - в виде формулы (чаще всего), при помощи таблицы значений, при помощи графика, текстовым описанием и т.д.

Функциональная зависимость атрибутов отношения тоже *напоминает* это определение. Действительно:

- В качестве области определения выступает домен, на котором определен атрибут X (или декартово произведение доменов, если X является множеством атрибутов)
- В качестве множества значений выступает домен, на котором определен атрибут Y (или декартово произведение доменов)
- Правило f реализуется следующим алгоритмом 1) по данному значению атрибута f найти любой кортеж отношения, содержащий это значение, 2) значение атрибута f в этом кортеже и будет значением функциональной зависимости, соответствующим данному f . Определение функциональной зависимости в отношении гарантирует, что найденное значение f не зависим от выбора кортежа, поэтому правило f

определено корректно.

Отличие от математического понятия отношения состоит в том, что, если рассматривать математическое понятие функции, то *для фиксированного значения*  $x \in X$  соответствующее значение функции y = f(x) всегда одно и то же. Например, если задана функция  $y = x^2$ , то для значения x = 2 соответствующее значение y всегда будет равно 4. В противоположность этому в отношениях значение зависимого атрибута может принимать различные значения в различных состояниях базы данных. Например, атрибут **ФАМ** функционально зависит от атрибута **H\_COTP**. Предположим, что сейчас сотрудник с табельным номером 1 имеет фамилию Иванов, т.е. при значении детерминанта равного 1, значение зависимого аргумента равно "Иванов". Но сотрудник может сменить фамилию, например на "Сидоров". Теперь при том же значении детерминанта, равного 1, значение зависимого аргумента равно "Сидоров".

Таким образом, понятие функциональной зависимости атрибутов нельзя считать полностью эквивалентным математическому понятию функциональной зависимости, т.к. значение этой зависимости различны при разных состояниях отношения, и, самое главное, эти значения могут меняться непредсказуемо.

Функциональная зависимость атрибутов утверждает лишь то, что для каждого конкретного состояния базы данных по значению одного атрибута (детерминанта) можно однозначно определить значение другого атрибута (зависимой части). Но конкретные значение зависимой части могут быть различны в различных состояниях базы данных.

# 2НФ (Вторая Нормальная Форма)

Определение 3. Отношение R находится во **второй нормальной форме** (**2**H**Ф**) тогда и только тогда, когда отношение находится в 1HФ и нет неключевых атрибутов, зависящих от части сложного ключа. (**Неключевой атрибут** - это атрибут, не входящий в состав никакого потенциального ключа).

<u>Замечание</u>. Если потенциальный ключ отношения является простым, то отношение автоматически находится в  $2H\Phi$ .

Отношение СОТРУДНИКИ\_ОТДЕЛЫ\_ПРОЕКТЫ не находится в 2НФ, т.к. есть атрибуты, зависящие от части сложного ключа:

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника является зависимостью от части сложного ключа:

$$H COTP \rightarrow \Phi AM$$

$$H\_COTP \rightarrow H\_OTД$$

$$H_{-}COTP \rightarrow$$
ТЕЛ

Зависимость наименования проекта от номера проекта является зависимостью от части сложного ключа:

 $H_{\bot}$ ПРО →ПРОЕКТ

Для того, чтобы устранить зависимость атрибутов от части сложного ключа, нужно произвести *декомпозицию* отношения на несколько отношений. При этом *те атрибуты, которые зависят от части сложного ключа,* выносятся в отдельное отношение.

Отношение **СОТРУДНИКИ\_ОТДЕЛЫ\_ПРОЕКТЫ** декомпозируем на три отношения - **СОТРУДНИКИ\_ОТДЕЛЫ**, **ПРОЕКТЫ**, **ЗАДАНИЯ**.

Отношение СОТРУДНИКИ\_ОТДЕЛЫ (Н\_СОТР, ФАМ, Н\_ОТД, ТЕЛ):

Функциональные зависимости:

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:

H  $COTP \rightarrow \Phi AM$ 

 $H\_COTP \rightarrow H\_OTД$ 

H сотр  $\rightarrow$  тел

Зависимость номера телефона от номера отдела:

 $H_OTД$  →TЕЛ

H_COTP	ФАМ	н_отд	ТЕЛ
1	Иванов	1	11-22-33
2	Петров	1	11-22-33
3	Сидоров	2	33-22-11

Таблица 2 Отношение СОТРУДНИКИ\_ОТДЕЛЫ

Отношение ПРОЕКТЫ (Н ПРО, ПРОЕКТ):

Функциональные зависимости:

H  $\Pi PO$  →  $\Pi POEKT$ 

Н_ПРО	ПРОЕКТ
1	Космос
2	Климат

Таблица 3 Отношение ПРОЕКТЫ

Отношение ЗАДАНИЯ (*H\_COTP*, *H\_ПРО*, H\_ЗАДАН):

Функциональные зависимости:

 $\{H\_COTP, H\_\Pi PO\} \rightarrow$  H\_ЗАДАН

H_COTP	Н_ПРО	Н_ЗАДАН
1	1	1
1	2	1
2	1	2
3	1	3
3	2	2

Таблица 4 Отношения ЗАДАНИЯ

## Анализ декомпозированных отношений

Отношения, полученные в результате декомпозиции, находятся в  $2H\Phi$ . Действительно, отношения **СОТРУДНИКИ\_ОТДЕЛЫ** и **ПРОЕКТЫ** имеют простые ключи, следовательно автоматически находятся в  $2H\Phi$ , отношение **ЗАДАНИЯ** имеет сложный ключ, но единственный неключевой атрибут **Н\_ЗАДАН** функционально зависит от всего ключа { $H\_COTP, H\_\Pi PO$ }.

Часть аномалий обновления устранена. Так, данные о сотрудниках и проектах теперь хранятся в различных отношениях, поэтому при появлении сотрудников, не участвующих ни в одном проекте просто добавляются кортежи в отношение **СОТРУДНИКИ\_ОТДЕЛЫ**. Точно также, при появлении проекта, над которым не работает ни один сотрудник, просто вставляется кортеж в отношение **ПРОЕКТЫ**.

Фамилии сотрудников и наименования проектов теперь хранятся без избыточности. Если сотрудник сменит фамилию или проект сменит наименование, то такое обновление будет произведено в одном месте.

Если по проекту временно прекращены работы, но требуется, чтобы сам проект сохранился, то для этого проекта удаляются соответствующие кортежи в отношении **ЗАДАНИЯ**, а данные о самом проекте и данные о сотрудниках, участвовавших в проекте, остаются в отношениях **ПРОЕКТЫ** и **СОТРУДНИКИ\_ОТДЕЛЫ**.

Тем не менее, часть аномалий разрешить не удалось.

#### Оставшиеся аномалии вставки (INSERT)

В отношение СОТРУДНИКИ\_ОТДЕЛЫ нельзя вставить кортеж (4, Пушников, 1, 33-22-11), т.к. при этом получится, что два сотрудника из 1-го отдела (Иванов и Пушников) имеют разные номера телефонов, а это противоречит модели предметной области. В этой ситуации можно предложить два решения, в зависимости от того, что реально произошло в предметной области. Другой номер телефона может быть введен по двум причинам - по ошибке человека, вводящего данные о новом сотруднике, или потому что номер в отделе действительно изменился. Тогда можно написать триггер, который при вставке записи о сотруднике проверяет, совпадает ли телефон с уже имеющимся телефоном у другого сотрудника этого же отдела. Если номера отличаются, то система должна задать вопрос, оставить ли старый номер в отделе или заменить его новым. Если нужно оставить старый номер (новый номер введен ошибочно), то кортеж с данными о новом сотруднике будет вставлен, но номер телефона будет у него будет тот, который уже есть в отделе (в данном случае, 11-22-33). Если же номер в отделе действительно изменился, то кортеж будет

вставлен с новым номером, и одновременно будут изменены номера телефонов у всех сотрудников этого же отдела. И в том и в другом случае не обойтись без разработки громоздкого триггера.

<u>Причина аномалии</u> - избыточность данных, порожденная тем, что в одном отношении хранится разнородная информация (о сотрудниках и об отделах).

<u>Вывод</u> - увеличивается сложность разработки базы данных. База данных, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

## Оставшиеся аномалии обновления (UPDATE)

Одни и те же номера телефонов повторяются во многих кортежах отношения. Поэтому если в отделе меняется номер телефона, то такие изменения необходимо одновременно выполнить во всех местах, где этот номер телефона встречаются, иначе отношение станет некорректным. Таким образом, обновление базы данных одним действием реализовать невозможно. Необходимо написать триггер, который при обновлении одной записи корректно исправляет номера телефонов в других местах.

<u>Причина аномалии</u> - избыточность данных, также порожденная тем, что в одном отношении хранится разнородная информация.

<u>Вывод</u> - увеличивается сложность разработки базы данных. База данных, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

## Оставшиеся аномалии удаления (DELETE)

При удалении некоторых данных по-прежнему может произойти потеря другой информации. Например, если удалить сотрудника Сидорова, то будет потеряна информация о том, что в отделе номер 2 находится телефон 33-22-11.

<u>Причина аномалии</u> - хранение в одном отношении разнородной информации (и о сотрудниках, и об отделах).

<u>Вывод</u> - логическая модель данных неадекватна модели предметной области. База данных, основанная на такой модели, будет работать неправильно.

Заметим, что при переходе ко второй нормальной форме отношения стали *почти* адекватными предметной области. Остались также трудности в разработке базы данных, связанные с необходимостью написания триггеров, поддерживающих целостность базы данных. Эти трудности теперь связаны только с одним отношением СОТРУДНИКИ\_ОТДЕЛЫ.

# ЗНФ (Третья Нормальная Форма)

*Определение 4*. Атрибуты называются *взаимно независимыми*, если ни один из них не является функционально зависимым от другого.

Определение 5. Отношение R находится в **тремьей нормальной форме** (**3**H**Ф**) тогда и только тогда, когда отношение находится в 2HФ и все неключевые атрибуты взаимно независимы.

Отношение **СОТРУДНИКИ\_ОТДЕЛЫ** не находится в 3НФ, т.к. имеется функциональная зависимость неключевых атрибутов (зависимость номера телефона от номера отдела):

 $H_{OTJ} \rightarrow TEJ$ 

Для того, чтобы устранить зависимость неключевых атрибутов, нужно произвести декомпозицию отношения на несколько отношений. При этом *те неключевые атрибуты*, которые являются зависимыми, выносятся в отдельное отношение.

Отношение СОТРУДНИКИ\_ОТДЕЛЫ декомпозируем на два отношения - СОТРУДНИКИ, ОТДЕЛЫ.

Отношение СОТРУДНИКИ (*H\_COTP*, ФАМ, H\_ОТД):

Функциональные зависимости:

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:

 $H\_COTP \rightarrow \Phi AM$ 

 $H\_COTP \rightarrow H\_OTД$ 

H сотр  $\rightarrow$  тел

H_COTP	ФАМ	н_отд
1	Иванов	1
2	Петров	1
3	Сидоров	2

Таблица 5 Отношение СОТРУДНИКИ

Отношение ОТДЕЛЫ (*H\_ОТД*, ТЕЛ):

Функциональные зависимости:

Зависимость номера телефона от номера отдела:

 $H_{-}$ ОТД $\rightarrow$ ТЕЛ

Н_ОТД	ТЕЛ
1	11-22-33
2	33-22-11

Таблица 6 Отношение ОТДЕЛЫ

Обратим внимание на то, что атрибут **H\_OT**Д, *не являвшийся ключевым* в отношении **СОТРУДНИКИ\_ОТДЕЛЫ**, *становится потенциальным ключом* в отношении **ОТДЕЛЫ**. Именно за счет этого устраняется избыточность, связанная с многократным хранением одних и тех же номеров телефонов.

<u>Вывод</u>. Таким образом, все обнаруженные аномалии обновления устранены. Реляционная модель, состоящая из четырех отношений **СОТРУДНИКИ**, **ОТДЕЛЫ**, **ПРОЕКТЫ**, **ЗАДАНИЯ**, находящихся в третьей нормальной форме, является адекватной описанной модели предметной области, и требует наличия только тех триггеров, которые поддерживают ссылочную целостность. Такие триггеры являются стандартными и не требуют больших усилий в разработке.

## Алгоритм нормализации (приведение к ЗНФ)

Итак, алгоритм нормализации (т.е. алгоритм приведения отношений к  $3H\Phi$ ) описывается следующим образом.

Шаг 2 (Приведение к  $2H\Phi$ ). Если в некоторых отношениях обнаружена зависимость атрибутов от части сложного ключа, то проводим декомпозицию этих отношений на несколько отношений следующим образом: те атрибуты, которые зависят от части сложного ключа выносятся в отдельное отношение вместе с этой частью ключа. В исходном отношении остаются все ключевые атрибуты:

Исходное отношение:  $R(K_1, K_2, A_1, ..., A_n, B_1, ..., B_m)$ .

Ключ:  $\{K_1, K_2\}$  - сложный.

Функциональные зависимости:

 $(K_1, K_2) \to (A_1, ..., A_m, B_1, ..., B_m)$  - зависимость всех атрибутов от ключа отношения.

 $\{K_1\} \to \{A_1,..,A_n\}$  - зависимость некоторых атрибутов от части сложного ключа.

Декомпозированные отношения:

$$R_1(K_1,K_2,B_1,...,B_m)$$
 - остаток от исходного отношения. Ключ  $\{K_1,K_2\}$  .

 $R_2(K_1,A_1,...,A_n)$  - атрибуты, вынесенные из исходного отношения вместе с частью сложного ключа. Ключ  $K_1$ .

*Шаг 3 (Приведение к 3НФ)*. Если в некоторых отношениях обнаружена зависимость некоторых неключевых атрибутов других неключевых атрибутов, то проводим декомпозицию этих отношений следующим образом: те неключевые атрибуты, которые

зависят других неключевых атрибутов выносятся в отдельное отношение. В новом отношении ключом становится детерминант функциональной зависимости:

Исходное отношение:  $R(K, A_1, ..., A_n, B_1, ..., B_m)$ .

Ключ: K.

Функциональные зависимости:

$$K \to \{A_1, ..., A_n, B_1, ..., B_m\}$$
 - зависимость всех атрибутов от ключа отношения.

 $\{A_1,...,A_n\} \to \{B_1,...,B_m\}$  - зависимость некоторых неключевых атрибутов других неключевых атрибутов.

Декомпозированные отношения:

$$R_1(K,A_1,...,A_n)$$
 - остаток от исходного отношения. Ключ  $K$  .

 $R_2(A_1,...,A_n,B_1,...,B_m)$  - атрибуты, вынесенные из исходного отношения вместе с детерминантом функциональной зависимости. Ключ  $\{A_1,...,A_n\}$  .

Замечание. На практике, при создании логической модели данных, как правило, не следуют прямо приведенному алгоритму нормализации. Опытные разработчики обычно сразу строят отношения в ЗНФ. Кроме того, основным средством разработки логических моделей данных являются различные варианты ЕR-диаграмм. Особенность этих диаграмм в том, что они сразу позволяют создавать отношения в ЗНФ. Тем не менее, приведенный алгоритм важен по двум причинам. Во-первых, этот алгоритм показывает, какие проблемы возникают при разработке слабо нормализованных отношений. Во-вторых, как правило, модель предметной области никогда не бывает правильно разработана с первого шага. Эксперты предметной области могут забыть о чем-либо упомянуть, разработчик может неправильно понять эксперта, во время разработки могут измениться правила, принятые в предметной области, и т.д. Все это может привести к появлению новых зависимостей, которые отсутствовали в первоначальной модели предметной области. Тут как раз и необходимо использовать алгоритм нормализации хотя бы для того, чтобы убедиться, что отношения остались в ЗНФ и логическая модель не ухудшилась.

# Анализ критериев для нормализованных и ненормализованных моделей данных

# Сравнение нормализованных и ненормализованных моделей

Соберем воедино результаты анализа критериев, по которым мы хотели оценить влияние логического моделирования данных на качество физических моделей данных и производительность базы данных:

Критерий	Отношения слабо нормализованы (1НФ, 2НФ)	Отношения сильно нормализованы (ЗНФ)
Адекватность базы данных	ХУЖЕ (-)	ЛУЧШЕ (+)

предметной области		
Легкость разработки и сопровождения базы данных	СЛОЖНЕЕ (-)	ЛЕГЧЕ (+)
Скорость выполнения вставки, обновления, удаления	МЕДЛЕННЕЕ (-)	БЫСТРЕЕ (+)
Скорость выполнения выборки данных	БЫСТРЕЕ (+)	МЕДЛЕННЕЕ (-)

Как видно из таблицы, более сильно нормализованные отношения оказываются лучше спроектированы (три плюса, один минус). Они больше соответствуют предметной области, легче в разработке, для них быстрее выполняются операции модификации базы данных. Правда, это достигается ценой некоторого замедления выполнения операций выборки данных.

У слабо нормализованных отношений единственное преимущество - если к базе данных обращаться только с запросами на выборку данных, то для слабо нормализованных отношений такие запросы выполняются быстрее. Это связано с тем, что в таких отношениях уже как бы произведено соединение отношений и на это не тратится время при выборке данных.

Таким образом, выбор степени нормализации отношений зависит от характера запросов, с которыми чаще всего обращаются к базе данных.

#### OLTP и OLAP-системы

Можно выделить некоторые классы систем, для которых больше подходят сильно или слабо нормализованные модели данных.

Сильно нормализованные модели данных хорошо подходят для так называемых *OLTP*приложений (On-Line Transaction Processing (OLTP)- оперативная обработка *транзакций*). Типичными примерами OLTP-приложений являются системы складского учета, системы заказов билетов, банковские системы, выполняющие операции по переводу денег, и т.п. Основная функция подобных систем заключается в выполнении большого количества коротких транзакций. Сами транзакции выглядят относительно просто, например, "снять сумму денег со счета А, добавить эту сумму на счет В". Проблема заключается в том, что, во-первых, транзакций очень много, во-вторых, выполняются они одновременно (к системе может быть подключено несколько тысяч одновременно работающих пользователей), в-третьих, при возникновении ошибки, транзакция должна целиком откатиться и вернуть систему к состоянию, которое было до начала транзакции (не должно быть ситуации, когда деньги сняты со счета А, но не поступили на счет В). Практически все запросы к базе данных в ОLTР-приложениях состоят из команд вставки, обновления, удаления. Запросы на выборку в основном предназначены для предоставления пользователям возможности выбора из различных справочников. Большая часть запросов, таким образом, известна заранее еще на этапе проектирования системы. Таким образом, критическим для ОLTP-приложений является скорость и надежность выполнения коротких операций обновления данных. Чем выше уровень нормализации данных в OLTP-приложении, тем оно, как правило, быстрее и надежнее. Отступления от этого правила могут происходить тогда, когда уже на этапе разработки известны некоторые часто возникающие запросы, требующие соединения отношений и от скорости выполнения которых существенно зависит работа приложений. В этом случае можно пожертвовать нормализацией для ускорения выполнения подобных запросов.

Другим типом приложений являются так называемые *OLAP-приложения* (*On-Line Analitical Processing* (*OLAP*) - *оперативная аналитическая обработка данных*). Это обобщенный термин, характеризующий принципы построения *систем поддержки принятия решений* (*Decision Support System - DSS*), *хранилищ данных* (*Data Warehouse*), *систем интеллектуального анализа данных* (*Data Mining*). Такие системы предназначены для нахождения зависимостей между данными (например, можно попытаться определить, как связан объем продаж товаров с характеристиками потенциальных покупателей), для проведения анализа "что если...". OLAP-приложения оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми их электронных таблиц или из других источников данных. Такие системы характеризуются следующими признаками:

- Добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из OLTP-приложения).
- Данные, добавленные в систему, обычно никогда не удаляются.
- Перед загрузкой данные проходят различные процедуры "очистки", связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления для одних и тех же понятий, данные могут быть некорректны, ошибочны.
- Запросы к системе являются нерегламентированными и, как правило, достаточно сложными. Очень часто новый запрос формулируется аналитиком для уточнения результата, полученного в результате предыдущего запроса.
- Скорость выполнения запросов важна, но не критична.

Данные OLAP-приложений обычно представлены в виде одного или нескольких гиперкубов, измерения которого представляют собой справочные данные, а в ячейках самого гиперкуба хранятся собственно данные. Например, можно построить гиперкуб, измерениями которого являются: время (в кварталах, годах), тип товара и отделения компании, а в ячейках хранятся объемы продаж. Такой гиперкуб будет содержать данных о продажах различных типов товаров по кварталам и подразделениям. Основываясь на этих данных, можно отвечать на вопросы вроде "у какого подразделения самые лучшие объемы продаж в текущем году?", или "каковы тенденции продаж отделений Юго-Западного региона в текущем году по сравнению с предыдущим годом?"

Физически гиперкуб может быть построен на основе специальной *многомерной модели данных* (*MOLAP* - *Multidimensional OLAP*) или построен средствами реляционной модели данных (*ROLAP* - *Relational OLAP*).

Возвращаясь к проблеме нормализации данных, можно сказать, что в системах OLAP, использующих реляционную модель данных (ROLAP), данные целесообразно хранить в виде слабо нормализованных отношений, содержащих заранее вычисленные основные итоговые данные. Большая избыточность и связанные с ней проблемы тут не страшны, т.к. обновление происходит только в момент загрузки новой порции данных. При этом происходит как добавление новых данных, так и пересчет итогов.

Корректность процедуры нормализации - декомпозиция без потерь. Теорема Хеза

Как было показано выше, алгоритм нормализации состоит в выявлении функциональных зависимостей предметной области и соответствующей декомпозиции отношений. Предположим, что мы уже имеем работающую систему, в которой накоплены данные. Пусть данных корректны в текущий момент, т.е. факты предметной области правильно отражаются текущим состоянием базы данных. Если в предметной области обнаружена новая функциональная зависимость (либо она была пропущена на этапе моделирования предметной области, либо просто изменилась предметная область), то возникает необходимость заново нормализовать данные. При этом некоторые отношения придется декомпозировать в соответствии с алгоритмом нормализации. Возникают естественные вопросы - что произойдет с уже накопленными данными? Не будут ли данные потеряны в ходе декомпозиции? Можно ли вернуться обратно к исходным отношениям, если будет принято решение отказаться от декомпозиции, восстановятся ли при этом данные?

Для ответов на эти вопросы нужно ответить на вопрос - что же представляет собой декомпозиция отношений с точки зрения операций реляционной алгебры? При декомпозиции мы из одного отношения получаем два или более отношений, каждое из которых содержит часть атрибутов исходного отношения. В полученных новых отношениях необходимо удалить дубликаты строк, если таковые возникли. Это в точности означает, что декомпозиция отношения есть не что иное, как взятие одной или нескольких проекций исходного отношения так, чтобы эти проекции в совокупности содержали (возможно, с повторениями) все атрибуты исходного отношения. Т.е., при декомпозиции не должны теряться атрибуты отношений. Но при декомпозиции также не должны потеряться и сами данные. Данные можно считать не потерянными в том случае, если возможна обратная операция - по декомпозированным отношениям можно восстановить исходное отношение в точности в прежнем виде. Операцией, обратной операции проекции, является операция соединения отношений. Имеется большое количество видов операции соединения (см. гл. 4). Т.к. при восстановлении исходного отношения путем соединения проекций не должны появиться новые атрибуты, то необходимо использовать естественное соединение.

Определение 6. Проекция  $^{R[X]}$  отношения  $^{R}$  на множество атрибутов  $^{X}$  называется собственной, если множество атрибутов  $^{X}$  является собственным подмножеством множества атрибутов отношения  $^{R}$  (т.е. множество атрибутов  $^{X}$  не совпадает с множеством всех атрибутов отношения  $^{R}$ ).

Определение 7. Собственные проекции  $^{R_1}$ и  $^{R_2}$ отношения  $^{R}$  называются **декомпозицией без потерь**, если отношение  $^{R}$  точно восстанавливается из них при помощи естественного соединения  $^{\partial n}$  любого состояния отношения  $^{R}$ :

$$R_1 \text{ JOIN } R_2 = R$$
.

Рассмотрим пример, показывающий, что декомпозиция без потерь происходит не всегда.

**Пример 2**. Пусть дано отношение R:

HOMEP	ФАМИЛИЯ	ЗАРПЛАТА
1	Иванов	1000
2	Петров	1000

## Таблица 7 Отношение R

Рассмотрим первый вариант декомпозиции отношения R на два отношения:

HOMEP	ЗАРПЛАТА
1	1000
2	1000

Таблица 8 Отношение  $R_1$ 

ФАМИЛИЯ	ЗАРПЛАТА
Иванов	1000
Петров	1000

Таблица 9 Отношение  $R_2$ 

Естественное соединение этих проекций, имеющих общий атрибут "ЗАРПЛАТА", очевидно, будет следующим (каждая строка одной проекции соединится с каждой строкой другой проекции):

HOMEP	ФАМИЛИЯ	ЗАРПЛАТА
1	Иванов	1000
1	Петров	1000
2	Иванов	1000
2	Петров	1000

Таблица 10 Отношение  $R_1 \text{ JOIN } R_2 \neq R$ 

Итак, данная декомпозиция не является декомпозицией без потерь, т.к. исходное отношение *не восстанавливается в точном виде* по проекциям (серым цветом выделены лишние кортежи).

Рассмотрим другой вариант декомпозиции:

HOMEP	ФАМИЛИЯ
1	Иванов
2	Петров

Таблица 11 Отношение  $R_1$ 

HOMEP	ЗАРПЛАТА
1	1000

2 1000
--------

# Таблица 12 Отношение $R_2$

По данным проекциям, имеющие общий атрибут "HOMEP", исходное отношение восстанавливается в точном виде. Тем не менее, нельзя сказать, что данная декомпозиция является декомпозицией без потерь, т.к. мы рассмотрели только *одно конкретное состояние* отношения  $^R$ , и не можем сказать, будет ли и в других состояниях отношение  $^R$  восстанавливаться точно. Например, предположим, что отношение  $^R$  перешло в состояние:

HOMEP	ФАМИЛИЯ	ЗАРПЛАТА
1	Иванов	1000
2	Петров	1000
2	Сидоров	2000

Таблица 13 Отношение R

Кажется, что этого не может быть, т.к. значения в атрибуте "НОМЕР" повторяются. Но мы же ничего не говорили о ключе этого отношения! Сейчас проекции будут иметь вид:

HOMEP	ФАМИЛИЯ
1	Иванов
2	Петров
2	Сидоров

Таблица 14 Отношение  $R_1$ 

HOMEP	ЗАРПЛАТА
1	1000
2	1000
2	2000

Таблица 15 Отношение  $R_2$ 

Естественное соединение этих проекций будет содержать лишние кортежи:

HOMEP	ФАМИЛИЯ	ЗАРПЛАТА
1	Иванов	1000
2	Петров	1000
2	Петров	2000
2	Сидоров	1000

2 Сидоров	2000
-----------	------

# Таблица 16 Отношение $R_1 \text{ JOIN } R_2 \neq R$

<u>Вывод</u>. Таким образом, *без дополнительных ограничений* на отношение R нельзя говорить о декомпозиции без потерь.

Такими дополнительными ограничениями и являются функциональные зависимости. Имеет место следующая теорема Xeзa [54]:

<u>Теорема (Хеза)</u>. Пусть R(A,B,C) является отношением, и A,B,C - атрибуты или множества атрибутов этого отношения. Если имеется функциональная зависимость  $A \to B$ , то проекции  $R_1 = R[A,B]_H$   $R_2 = R[A,C]_H$  образуют декомпозицию без потерь.

<u>Доказательство</u>. Необходимо доказать, что  $R_1$  JOIN  $R_2 = R$  для любого состояния отношения R. В левой и правой части равенства стоят *множества* кортежей, поэтому для доказательства достаточно доказать два включения для двух *множеств* кортежей:  $R_1$  JOIN  $R_2 \supseteq R_{_{\mathbf{U}}}$   $R_1$  JOIN  $R_2 \subseteq R$ 

Докажем первое включение. Возьмем произвольный кортеж  $r = (a,b,c) \in R$ . Докажем, что он включается также и в  $R_1$  JOIN  $R_2$ . По определению проекции, кортежи  $r_1 = (a,b) \in R_1$  и  $r_2 = (a,c) \in R_2$ . По определению естественного соединения кортежи  $r_1$  и  $r_2$ , имеющие одинаковое значение a общего атрибута A, будут соединены в процессе естественного соединения в кортеж  $(a,b,c) \in R_1$  JOIN  $R_2$ . Таким образом, включение доказано.

Докажем обратное включение. Возьмем произвольный кортеж  $r=(a,b,c)\in R_1$  JOIN  $R_2$ . Докажем, что он включается также и в R. По определению естественного соединения получим, что в имеются кортежи  $r_1=(a,b)\in R_1$  и  $r_2=(a,c)\in R_2$ . Т.к.  $R_1=R[A,B]$ , то существует некоторое значение  $c_1$ , такое что кортеж  $c_1=(a,b,c_1)\in R$ . Аналогично, существует некоторое значение  $c_1=(a,b,c_1)\in R$ . Кортежи  $c_1=(a,b,c_1)\in R$ . Обратное включение доказано.  $c_1=(a,b,c_1)\in R$ . Обратное включение доказано.  $c_1=(a,b,c_1)\in R$ .

Т.к. алгоритм нормализации (приведения отношений к 3HФ) основан на имеющихся в отношениях функциональных зависимостях, то теорема Хеза показывает, что алгоритм нормализации является корректным, т.е. в ходе нормализации *не происходит потери информации*.

#### Выводы

При разработке базы данных можно выделить несколько уровней моделирования:

- Сама предметная область
- Модель предметной области
- Логическая модель данных
- Физическая молель ланных
- Собственно база данных и приложения

Ключевые решения, определяющие качество будущей базы данных закладываются на этапе разработки логической модели данных. "Хорошие" модели данных должны удовлетворять определенным критериям:

- Адекватность базы данных предметной области
- Легкость разработки и сопровождения базы данных
- Скорость выполнения операций обновления данных (вставка, обновление, удаление)
- Скорость выполнения операций выборки данных

**Первая нормальная форма** ( $1H\Phi$ ) - это обычное отношение. Отношение в  $1H\Phi$  обладает следующими свойствами:

- В отношении нет одинаковых кортежей.
- Кортежи не упорядочены.
- Атрибуты не упорядочены.
- Все значения атрибутов атомарны.

Отношения, находящиеся в 1НФ являются "плохими" в том смысле, что они не удовлетворяют выбранным критериям - имеется большое количество аномалий обновления, для поддержания целостности базы данных требуется разработка сложных триггеров.

Отношение R находится во *второй нормальной форме* ( $2H\Phi$ ) тогда и только тогда, когда отношение находится в  $1H\Phi$  и нет неключевых атрибутов, зависящих от части сложного ключа.

Отношения в  $2H\Phi$  "лучше", чем в  $1H\Phi$ , но еще недостаточно "хороши" - остается часть аномалий обновления, по-прежнему требуются триггеры, поддерживающие целостность базы данных.

Отношение R находится в *темьей нормальной форме* ( $3H\Phi$ ) тогда и только тогда, когда отношение находится в  $2H\Phi$  и все неключевые атрибуты взаимно независимы.

Отношения в 3HФ являются самыми "хорошими" с точки зрения выбранных нами критериев - устранены аномалии обновления, требуются только стандартные триггеры для поддержания ссылочной целостности.

Переход от ненормализованных отношений к отношениям в 3НФ может быть выполнен при помощи *алгоритма нормализации*. Алгоритм нормализации заключается в последовательной декомпозиции отношений для устранения функциональных

зависимостей атрибутов от части сложного ключа (приведение к 2HФ) и устранения функциональных зависимостей неключевых атрибутов друг от друга (приведение к 3HФ).

Корректность процедуры нормализации (декомпозиция без потери информации) доказывается *теоремой Хеза*.

## Глава 7. Нормальные формы более высоких порядков

В предыдущей главе были рассмотрены нормальные формы вплоть до третьей нормальной формы (3НФ). В большинстве случаев этого вполне достаточно, чтобы разрабатывать вполне работоспособные базы данных. В данной главе рассматриваются нормальные формы более высоких порядков, а именно, нормальная форма Бойса-Кодда (НФБК), четвертая нормальная форма (4НФ), пятая нормальная форма (5НФ).

## НФБК (Нормальная Форма Бойса-Кодда)

При приведении отношений при помощи алгоритма нормализации к отношениям в ЗНФ неявно предполагалось, что все отношения содержат один потенциальный ключ. Это не всегда верно. Рассмотрим следующий пример отношения, содержащего два ключа.

<u>Пример 1</u>. Пусть требуется хранить данные о поставках деталей некоторыми поставщиками. Предположим, что наименования поставщиков являются уникальными. Кроме того, каждый поставщик имеет свой уникальный номер. Данные о поставках можно хранить в следующем отношении:

Номер поставщика PNUM	Наименование поставщика PNAME	Hoмер детали DNUM	Поставляемое количество VOLUME
1	Фирма 1	1	100
1	Фирма 1	2	200
1	Фирма 1	3	300
2	Фирма 2	1	150
2	Фирма 2	2	250
3	Фирма 3	1	1000

Таблица 1 Отношение "Поставки"

Данное отношение содержит два потенциальных ключа - {PNUM, DNUM} и {PNAME, DNUM}. Видно, что данные хранятся в отношении с избыточностью - при изменении наименования поставщика, это наименование нужно изменить во всех кортежах, где оно встречается. Можно ли эту аномалию устранить при помощи алгоритма нормализации, описанного в предыдущей главе? Для этого нужно выявить имеющиеся функциональные зависимости (как обычно, курсивом выделены ключевые атрибуты):

 $PNUM \rightarrow PNAME$  - наименование поставщика зависит от номера поставщика.

 $PNAME \rightarrow PNUM$  - номер поставщика зависит от наименования поставщика.

 $\{PNUM, DNUM\} \rightarrow VOLUME$  - поставляемое количество зависит от первого ключа отношения.

 $\{PNUM, DNUM\} \xrightarrow{} PNAME$  - наименование поставщика зависит от первого ключа отношения.

 $\{PNAME, DNUM\} \rightarrow VOLUME$  - поставляемое количество зависит от второго ключа отношения.

 $\{PNAME, DNUM\} \rightarrow PNUM$  - номер поставщика зависит от второго ключа отношения.

Данное отношение не содержит *неключевых* атрибутов, зависящих от части сложного ключа (см. определение  $2H\Phi$ ). Действительно, от части сложного ключа зависят атрибуты *PNAME* и *PNUM*, но они сами являются ключевыми. Таким образом, отношение находится в  $2H\Phi$ .

Кроме того, отношение не содержит зависимых друг от друга *неключевых* атрибутов, т.к. неключевой атрибут всего один - **VOLUME** (см. определение  $3H\Phi$ ). Таким образом, показано, что отношение "Поставки" находится в  $3H\Phi$ .

Таким образом, описанный ранее алгоритм нормализации неприменим к данному отношению. Очевидно, однако, что аномалия данного отношения устраняется путем декомпозиции его на два следующих отношения:

Номер поставщика PNUM	Наименование поставщика PNAME
1	Фирма 1
2	Фирма 2
3	Фирма 3

Таблица 2 Отношение "Поставшики"

Номер поставщика PNUM	Номер детали DNUM	Поставляемое количество VOLUME
1	1	100
1	2	200
1	3	300
2	1	150
2	2	250
3	1	1000

Таблица 3 Отношение "Поставки-2"

Определение 1. Отношение R находится в **нормальной форме Бойса-Кодда** (**НФБК**) тогда и только тогда, когда детерминанты всех функциональных зависимостей являются потенциальными ключами.

<u>Замечание</u>. Если отношение находится в НФБК, то оно автоматически находится и в 3НФ. Действительно, это сразу следует из определения 3НФ.

Отношение "Поставки" не находится в НФБК, т.к. имеются зависимости ( $PNUM \rightarrow PNAME$  и  $PNAME \rightarrow PNUM$ ), детерминанты которых не являются потенциальными ключами.

Для того чтобы устранить зависимость от детерминантов, не являющихся потенциальными ключами, необходимо провести декомпозицию, вынося эти детерминанты и зависимые от них части в отдельное отношение. Отношения "Поставщики" и "Поставки-2", полученные в результате декомпозиции находятся в НФБК.

Замечание. Приведенная декомпозиция отношения "Поставки" на отношения "Поставщики" и "Поставки-2" не является единственно возможной. Альтернативной декомпозицией является декомпозиция на следующие отношения:

Номер поставщика PNUM	Наименование поставщика PNAME
1	Фирма 1
2	Фирма 2
3	Фирма 3

Таблина 4 Отношение "Поставшики"

Наименование поставщика PNAME	Номер детали DNUM	Поставляемое количество VOLUME
Фирма 1	1	100
Фирма 1	2	200
Фирма 1	3	300
Фирма 2	1	150
Фирма 2	2	250
Фирма 3	1	1000

Таблица 5 Отношение "Поставки-3"

На первый взгляд, такая декомпозиция хуже, чем первая. Действительно, наименования поставщиков по-прежнему повторяются, и при изменении наименования поставщика, это наименование придется менять одновременно в нескольких местах (тем более сразу в двух отношениях!). Кажется, что ситуация стала еще хуже, чем была до декомпозиции. Однако такое ощущение возникает от того, что мы интуитивно считаем, что наименования поставщиков могут меняться, а номера - нет. Если же предположить, что номера поставщиков тоже могут меняться (почему бы нет - директор приказал перенумеровать поставщиков!), то первая декомпозиция получается такой же "плохой" как и вторая - повторяющиеся номера придется менять одновременно в нескольких местах и также сразу в двух отношениях.

На самом деле никакого противоречия тут нет. В отношении "Поставки-3" атрибут "Наименование поставщика" (PNAME) является внешним ключом, служащим для связи с отношением "Поставщики". Поэтому, при изменении наименования поставщика, это изменение производится в отношении "Поставщики" и каскадно (см. стратегии поддержания ссылочной целостности в гл. 3) распространяется на отношение "Поставки-3" совершенно так, как изменение номера поставщика каскадно распространяется на отношение "Поставки-2". Поэтому, формально обе декомпозиции совершенно равноправны. В реальной работе разработчик выберет, конечно, первую декомпозицию, но тут важно подчеркнуть, что его выбор основан совсем на других соображениях, не имеющих отношения к формальной теории нормальных форм.

Замечание. Отношение "Поставки-2", полученное в результате декомпозиции имеет всего один потенциальный ключ. Поэтому, для анализа отношения "Поставки-2" не требуется привлекать определение НФБК, достаточно определения ЗНФ. Хотя отношение "Поставщики" имеет два потенциальных ключа, но, т.к. других атрибутов в нем нет, оно уже так просто устроено, что упростить его дальше нельзя. Возникает вопрос, имеются ли нетривиальные примеры отношений в НФБК, не находящиеся в ЗНФ и не такие простые, как отношение "Поставшики"?

<u>Пример 2</u>. Предположим, что нам по-прежнему необходимо учитывать поставки, но каждый акт поставки должен иметь некоторый уникальный номер (назовем его "сквозной номер поставки"). Отношение может иметь следующий вид:

Номер поставщика PNUM	Номер детали DNUM	Поставляемое количество VOLUME	Сквозной номер поставки NN
1	1	100	1
1	2	200	2
1	3	300	3
2	1	150	4
2	2	250	5
3	1	1000	6

Таблица 6 Отношение "Поставки-с-номером"

Одним потенциальным ключом данного отношения является, как и раньше, пара атрибутов  $\{PNUM, DNUM\}$ . Другим ключом, в силу уникальности сквозного номера, является атрибут NN. В данном отношении имеются следующие функциональные зависимости:

Зависимость атрибутов от первого ключа отношения:

 $\{PNUM, DNUM\} \rightarrow VOLUME,$ 

 $\{PNUM, DNUM\} \rightarrow NN.$ 

Зависимость атрибутов от второго ключа отношения:

```
NN \rightarrow PNUM.
```

 $NN \rightarrow DNUM$ ,

 $NN \rightarrow VOLUME$ .

Зависимости, являющиеся следствием зависимостей от ключей отношения:

 $\{PNUM, DNUM\} \rightarrow \{VOLUME, NN\},\$ 

 $NN \rightarrow \{PNUM, DNUM\},$ 

 $NN \rightarrow \{PNUM, VOLUME\},\$ 

 $NN \rightarrow \{DNUM, VOLUME\},$ 

 $NN \rightarrow \{PNUM, DNUM, VOLUME\}.$ 

Как можно заметить, детерминанты всех зависимостей являются потенциальными ключами, поэтому данное отношение находится в НФБК. Особенностью данного отношения является то, что оно имеет два совершенно независимых потенциальных ключа.

### 4НФ (Четвертая Нормальная Форма)

Рассмотрим следующий пример. Пусть требуется учитывать данные об абитуриентах, поступающих в ВУЗ. При анализе предметной области были выделены следующие требования:

- Каждый абитуриент имеет право сдавать экзамены на несколько факультетов одновременно.
- Каждый факультет имеет свой список сдаваемых предметов.
- Один и тот же предмет может сдаваться на нескольких факультетах.
- Абитуриент обязан сдавать все предметы, указанные для факультета, на который он поступает, несмотря на то, что он, может быть, уже сдавал такие же предметы на другом факультете.

Предположим, что нам требуется хранить данные о том, какие предметы должен сдавать каждый абитуриент. Попытаемся хранить данные в одном отношении "Абитуриенты-Факультеты-Предметы":

Абитуриент	Факультет	Предмет
Иванов	Математический	Математика
Иванов	Математический	Информатика
Иванов	Физический	Математика
Иванов	Физический	Физика
Петров	Математический	Математика
Петров	Математический	Информатика

### Таблица 7 Отношение "Абитуриенты-Факультеты-Предметы"

В данный момент в отношении хранится информация о том, что абитуриент Иванов поступает на два факультета (математически и физический), а абитуриент Петров - только на математический. Кроме того, можно сделать вывод, что на математическом факультете нужно сдавать математику и информатику, а на физическом - математику и физику.

Кажется, что в отношении имеется аномалия обновления, связанная с тем, что дублируются фамилии абитуриентов, наименования факультетов и наименования предметов. Однако эта аномалия легко устраняется стандартным способом - вынесением всех наименований в отдельные отношения, оставляя в исходном отношении только соответствующие номера:

Номер Абитуриента	Номер Факультета	Номер Предмета
1	1	1
1	1	2
1	2	1
1	2	3
2	1	1
2	1	2

## Таблица 8 Модифицированное отношение "Абитуриенты-Факультеты-Предметы"

Номер Абитуриента	Абитуриент
1	Иванов
2	Петров

## Таблица 9 Отношение "Абитуриенты"

Номер Факультета	Факультет
1	Математический
2	Физический

### Таблица 10 Отношение "Факультеты"

Номер Предмета	Предмет
1	Математика
2	Информатика
3	Физика

### Таблица 11 Отношение "Предметы"

Теперь каждое наименование встречается только в одном месте.

И все-таки как в исходном, так и в модифицированном отношении имеются аномалии обновления, возникающие при попытке вставить или удалить кортежи.

Аномалия вставки. При попытке добавить в отношение "Абитуриенты-Факультеты-Предметы" новый кортеж, например (Сидоров, Математический, Математика), мы обязаны добавить также и кортеж (Сидоров, Математический, Информатика), т.к. все абитуриенты математического факультета обязаны иметь один и тот же список сдаваемых предметов. Соответственно, при попытке вставить в модифицированное отношении кортеж (3, 1, 1), мы обязаны вставить в него также и кортеж (3, 1, 2).

Аномалия удаления. При попытке удалить кортеж (Иванов, Математический, Математика), мы *обязаны* удалить также и кортеж (Иванов, Математический, Информатика) по той же самой причине.

Таким образом, вставка и удаление кортежей не может быть выполнена независимо от других кортежей отношения.

Кроме того, если мы удалим кортеж (Иванов, Физический, Математика), а вместе с ним и кортеж (Иванов, Физический, Физика), то будет потеряна информация о предметах, которые должны сдаваться на физическом факультете.

Декомпозиция отношения "Абитуриенты-Факультеты-Предметы" для устранения указанных аномалий не может быть выполнена на основе функциональных зависимостей, т.к. это отношение *не содержит никаких функциональных зависимостей*. Это отношение является полностью ключевым, т.е. ключом отношения является все множество атрибутов. Но ясно, что какая-то взаимосвязь между атрибутами имеется. Эта взаимосвязь описывается понятием *многозначной зависимости*.

*Определение 2.* Пусть R - отношение, и X, Y, Z - некоторые из его атрибутов (или *непересекающиеся* множества атрибутов).

Тогда атрибуты (множества атрибутов) Y и Z многозначно зависят от X (обозначается  $X \longrightarrow Y \mid Z$ ), тогда и только тогда, когда из того, что в отношении R содержатся кортежи  $r_1 = (x,y,z_1)_{\mathsf{H}} \ r_2 = (x,y_1,z)_{\mathsf{Cледует}}$ , что в отношении R содержится также и кортеж R

Замечание. Меняя местами кортежи  $^{r_1}$ и  $^{r_2}$ в определении многозначной зависимости, получим, что в отношении  $^R$  должен содержаться также и кортеж  $^{r_4}$  =  $(x,y_1,z_1)$ . Таким образом, атрибуты  $^Y$  и  $^Z$ , многозначно зависящие от  $^X$ , ведут себя "симметрично" по отношению к атрибуту  $^X$ .

В отношении "Абитуриенты-Факультеты-Предметы" имеется многозначная зависимость  $\Phi$ акультет $\longrightarrow$  Абитуриент|Предмет.

Словами это можно выразить так - для каждого факультета (для каждого значения из X) каждый поступающий на него абитуриент (значение из Y) сдает *один и тот же список* предметов (набор значений из Z), и для каждого факультета (для каждого значения из X) каждый сдаваемый на факультете экзамен (значение из Z) сдается *одним и тем же списком* абитуриентов (набор значений из Y). Именно наличие этой зависимости не позволяет независимо вставлять и удалять кортежи. Кортежи обязаны вставляться и удаляться одновременно *целыми наборами*.

Замечание. Если в отношении X имеется не менее трех атрибутов X, Y, Z и есть функциональная зависимость  $X \to Y$ , то есть и многозначная зависимость  $X \to Y \mid Z$ .

Действительно, действуя формально в соответствии с определением многозначной зависимости, предположим, что в отношении R содержатся кортежи  $r_1=(x,y,z_1)_{\rm H}$   $r_2=(x,y_1,z)$ . В силу функциональной зависимости  $X\to Y$  отсюда следует, что  $Y=y_1$ . Но тогда кортеж  $r_3=(x,y,z)$  в точности совпадает с кортежем  $r_2=(x,y_1,z)_{\rm H}$ , следовательно, содержится в отношении R. Таким образом, имеется многозначная зависимость  $X\to\to Y\mid Z$ 

Таким образом, понятие многозначной зависимости является обобщением понятия функциональной зависимости.

Определение 3. Многозначная зависимость  $X \longrightarrow Y \mid Z$  называется **нетривиальной многозначной зависимостью**, если не существует функциональных зависимостей  $X \to Y$  и  $X \to Z$ 

В отношении "Абитуриенты-Факультеты-Предметы" имеется именно нетривиальная многозначная зависимость Факультет  $\rightarrow$  Абитуриент|Предмет. В силу нетривиальности этой зависимости мы не можем воспользоваться теоремой Хеза для декомпозиции отношения. Однако Фейджином Р. [52] доказана следующая теорема:

<u>Теорема (Фейджина)</u>. Пусть X, Y, Z - непересекающиеся множества атрибутов отношения R(X,Y,Z).

Декомпозиция отношения R на проекции  $R_1 = R[X,Y]_{\mathsf{H}} \ R_2 = R[X,Z]_{\mathsf{будет}}$  декомпозицией без потерь тогда и только тогда, когда имеется многозначная зависимость  $X \longrightarrow Y \mid Z$ .

<u>Замечание</u>. Если зависимость  $X \to Y \mid Z$  является тривиальной, т.е. существует одна из функциональных зависимостей  $X \to Y$  или  $X \to Z$ , то получаем теорему Xeза.

Доказательство теоремы.

<u>Необходимость</u>. Пусть декомпозиция отношения R на проекции  $R_1 = R[X,Y]_{\mathcal{U}}$   $R_2 = R[X,Z]_{\mathsf{ЯВЛЯЕТСЯ}}$  декомпозицией без потерь. Докажем что  $X \longrightarrow Y \mid Z$ .

Предположим, что отношение R содержит кортежи  $r_1 = (x, y, z_1)_H$   $r_2 = (x, y_1, z)_L$ . Необходимо доказать, что кортеж  $r_3 = (x, y, z)_L$  также содержится в R. По определению

проекций, кортеж  $r_1'=(x,y)$  содержится в  $R_1=R[X,Y]$ , а кортеж  $r_2'=(x,z)$  содержится в  $R_2=R[X,Z]$ . Тогда кортеж  $r_3=(x,y,z)$  содержится в естественном соединении  $R_1$  JOIN  $R_2$ , а в силу того, что декомпозиция является декомпозицией без потерь, этот кортеж содержится и в R. Heoбxodumocmb dokasaha.

<u>Достаточность</u>. Пусть имеется многозначная зависимость  $X \longrightarrow Y \mid Z$ . Докажем, что декомпозиция отношения R на проекции  $R_1 = R[X,Y]_{\mathcal{U}} R_2 = R[X,Z]_{\mathsf{ЯВЛЯЕТСЯ}}$  декомпозицией без потерь.

Как и в доказательстве теоремы Хеза, нужно доказать, что  $R_1$  JOIN  $R_2 = R$  для любого состояния отношения R.

Включение  $R_1 \text{ JOIN } R_2 \supseteq R$  доказывается как в теореме Xeзa. Такое включение выполняется всегда для любой декомпозиции отношения R.

Докажем включение  $R_1$  ЈОІN  $R_2 \subseteq R$ . Пусть кортеж  $r = (x,y,z) \in R_1$  ЈОІN  $R_2$ . Это означает, что в проекции  $R_1 = R[X,Y]$  содержится кортеж  $r_1' = (x,y)$ , а в проекции  $R_2 = R[X,Z]$  содержится кортеж  $r_2' = (x,z)$ . По определению проекции, найдется такое значение  $Z_1$  атрибута Z, что отношение  $Z_1$  содержит кортеж  $Z_2 = (x,y,z_1)$ . Аналогично, найдется такое значение  $Z_2 = (x,y,z_1)$  доглатрибута  $Z_2 = (x,y_1,z)$  доглато определению многозначной зависимости кортеж  $Z_2 = (x,y_1,z)$  доглато определению многозначной зависимости кортеж  $Z_2 = (x,y,z)$  включение доказано.  $Z_2 = (x,y,z) \in R$  включение доказана.  $Z_2 = (x,y,z) \in R$  включение доказана.

Определение 4. Отношение R находится в **четвертой нормальной форме** (**4**H**Ф**) тогда и только тогда, когда отношение находится в НФБК и *не содержит нетривиальных многозначных зависимостей*.

Отношение "Абитуриенты-Факультеты-Предметы" находится в НФБК, но не в 4НФ. Согласно теореме Фейджина, это отношение можно без потерь декомпозировать на отношения:

Факультет	Абитуриент
Математический	Иванов
Физический	Иванов
Математический	Петров

Таблица 12 Отношение "Факультеты-Абитуриенты"

Факультет	Предмет	
Математический	Математика	
Математический	Информатика	
Физический	Математика	
Физический	Физика	

### Таблица 13 Отношение "Факультеты-Предметы"

В полученных отношениях устранены аномалии вставки и удаления, характерные для отношения "Абитуриенты-Факультеты-Предметы".

Заметим, что полученные отношения остались полностью ключевыми, и в них попрежнему нет функциональных зависимостей.

Отношения с нетривиальными многозначными зависимостями возникают, как правило, в результате естественного соединения двух отношений по общему полю, которое *не является ключевым ни в одном из отношений*. Фактически это приводит к попытке хранить в одном отношении информацию о двух *независимых* сущностях. В качестве еще одного примера можно привести ситуацию, когда сотрудник может иметь много работ и много детей. Хранение информации о работах и детях в одном отношении приводит к возникновению нетривиальной многозначной зависимости Работник  $\xrightarrow{}$  Работа|Дети.

### 5НФ (Пятая Нормальная Форма)

Функциональные и многозначные зависимости позволяют произвести декомпозицию исходного отношения без потерь на *две* проекции. Можно, однако, привести примеры отношений, которые нельзя декомпозировать без потерь ни на какие две проекции.

**Пример 3**. Рассмотрим следующее отношение R:

X	Y	Z
1	1	2
1	2	1
2	1	1
1	1	1

Таблица 14 Отношение R

Всевозможные проекции отношения R, включающие по два атрибута, имеют вид:

X	Y
1	1
1	2
2	1

Таблица 15 Проекция R1=R[X,Y]

X	Z
1	2
1	1
2	1

### Таблица 16 Проекция R2=R[X,Z]

### Таблица 17 Проекция R3=R[Y,Z]

Как легко заметить, отношение R не восстанавливается ни по одному из *попарных* соединений  $R_1$  JOIN  $R_2$ ,  $R_1$  JOIN  $R_3$  или  $R_2$  JOIN  $R_3$ . Действительно, соединение  $R_1$  JOIN  $R_2$  имеет вил:

X	Y	Z
1	1	2
1	1	1
1	2	2
1	2	1
2	1	1

Таблина 18 R1 JOIN R2

Серым цветом выделен лишний кортеж, отсутствующий в отношении R. Аналогично (в силу соображений симметрии) и другие попарные соединения не восстанавливают отношения R.

Однако отношение R восстанавливается соединением всех mpex проекций:

$$R_1 \text{ JOIN } R_2 \text{ JOIN } R_3 = R$$

Это говорит о том, что между атрибутами этого отношения также имеется некоторая зависимость, но эта зависимость не является ни функциональной, ни многозначной зависимостью.

Определение 5. Пусть R является отношением, а A, B, ..., Z - произвольными (возможно пересекающимися) подмножествами множества атрибутов отношения R. Тогда отношение R удовлетворяет зависимости соединения

$$*(A, B, ..., Z)$$

тогда и только тогда, когда оно равносильно соединению всех своих проекций с подмножествами атрибутов  $A, B, \ldots, Z$ , т.е.

$$R = R[A] \text{ JOIN } R[B] \text{ JOIN } ... \text{ JOIN } R[Z]$$

Можно предположить, что отношение R в примере 3 удовлетворяет следующей зависимости соединения:

Утверждать, что это именно так мы пока не можем, т.к. определение зависимости соединения должно выполняться для *любого* состояния отношения R, а не только для состояния, приведенного в примере.

Покажем, что зависимость соединения является обобщением понятия многозначной зависимости. Действительно, согласно теореме Фейджина, отношение R(X,Y,Z) может быть декомпозировано без потерь на проекции  $R_1 = R[X,Y]_{\mathcal{U}} R_2 = R[X,Z]_{\mathsf{ТОГДА}}$  и только тогда, когда имеется многозначная зависимость  $X \longrightarrow Y \mid Z$ . Согласно определению зависимости соединения, теорема Фейджина может быть переформулирована следующим образом:

<u>Теорема Фейджина (другая формулировка)</u>. Отношение  ${}^{R(X,Y,Z)}$ удовлетворяет зависимости соединения  ${}^{*}(XY,XZ)$  тогда и только тогда, когда имеется многозначная зависимость  ${}^{X} \longrightarrow Y \mid Z$ .

Т.к. теорема Фейджина является взаимно обратной, то ее можно взять в качестве определения многозначной зависимости. Таким образом, многозначная зависимость является частным случаем зависимости соединения, т.е., если в отношении имеется многозначная зависимость, то имеется и зависимость соединения. Обратное, конечно, неверно.

Определение 6. Зависимость соединения \*(A, B, ..., Z) называется **нетривиальной** зависимостью соединения, если выполняется два условия:

- Одно из множеств атрибутов
  - $^{A,\,B,\ldots,Z}$  не содержит потенциального ключа отношения  $^{R}$  .
- Hu одно из множеств атрибутов не совпадает со всем множеством атрибутов отношения R.

Для удобства работы сформулируем это определение так же и в отрицательной форме:

Определение 7. Зависимость соединения \*(A,B,...,Z) называется **тривиальной** зависимостью соединения, если выполняется одно из условий:

- Либо все множества атрибутов  $^{A,B,...,Z}$  содержат потенциальный ключ отношения  $^{R}$  .
- Либо одно из множеств атрибутов совпадает со всем множеством атрибутов отношения R.

Определение 8. Отношение R находится в **пятой нормальной форме** (**5**H**Ф**) тогда и только тогда, когда любая имеющаяся зависимость соединения является тривиальной.

Определения 5H $\Phi$  может стать более понятным, если сформулировать его в отрицательной форме:

Определение 9. Отношение R не находится в **5**H $\Phi$ , если в отношении найдется нетривиальная зависимость соединения.

Возвращаясь к примеру 3, становится понятно, что не зная ничего о том, какие потенциальные ключи имеются в отношении и как взаимосвязаны атрибуты, нельзя делать выводы о том, находится ли данное отношение в  $5H\Phi$  (как, впрочем, и в других нормальных формах). По данному конкретному примеру можно только предположить, что отношение в примере 3 не находится в  $5H\Phi$ . Предположим, что анализ предметной области позволил выявить следующие зависимости атрибутов в отношении R:

- (i) Отношение R является полностью ключевым (т.е. потенциальным ключом отношения является все множество атрибутов).
- (іі) Имеется следующая зависимость (довольно странная, с практической точки зрения): если в отношении R содержатся кортежи  $r_1 = (x, y, z_1)$ ,  $r_2 = (x, y_1, z)$   $r_3 = (x_1, y, z)$ , то отсюда следует, что в отношении R содержится также и кортеж  $r_4 = (x, y, z)$ .

 $\underline{\mathit{Утверждениe}}$ . Докажем, что при наличии ограничений (i) и (ii), отношение находится в  $4H\Phi$ , но не в  $5H\Phi$ .

<u>Доказательство</u>. Покажем, что отношение <sup>R</sup> находится в 4НФ. Согласно определению 4НФ, необходимо показать, что отношение находится в НФБК и не содержит нетривиальных многозначных зависимостей. Т.к. отношение является полностью ключевым, то оно автоматически находится в НФБК. Если бы в отношении имелась многозначная зависимость (необязательно нетривиальная), то, согласно теореме Фейджина, отношение можно было бы декомпозировать без потерь на две проекции. Но пример 3 показывает, что таких декомпозиций нет (здесь мы воспользовались тем, что для доказательства возможности декомпозиции необходимо доказать ее для всех возможных состояний отношения, а для доказательства невозможности достаточно привести один контрпример). Поэтому в отношении нет никаких многозначных зависимостей.

Покажем, что отношение не находится в 5НФ. Для этого нужно привести пример нетривиальной зависимости соединения. Естественным кандидатом на нее является \*(XY,XZ,YZ). Если это действительно зависимость соединения, то она нетривиальна. Действительно, ни одно из множеств атрибутов XY, XZ и YZ не совпадает с множеством всех атрибутов отношения X и не содержит потенциального ключа.

Но является ли такая декомпозиция именно зависимостью соединения? Для этого нужно показать, что декомпозиция на три проекции  $R_1 = R[X,Y]$ ,  $R_2 = R[X,Z]_H$   $R_3 = R[Y,Z]$  является декомпозицией без потерь для *любого состояния* отношения R (именно здесь содержится ключевая тонкость, обычно пропускаемая при анализе конкретного состояния отношения R в примере 3, и именно здесь нам понадобятся знания о предметной области, выраженные в утверждении (ii)).

Как и в предыдущих доказательствах, нужно доказать, что  $R_1 \text{ JOIN } R_2 \text{ JOIN } R_3 = R_{\text{для}}$  любого состояния отношения R.

Включение  $R_1$  JOIN  $R_2$  JOIN  $R_3 \supseteq R$  доказывается как в теореме Xe3a. Такое включение выполняется всегда для любой декомпозиции отношения R.

Докажем включение  $R_1 \text{ JOIN} R_2 \text{ JOIN} R_3 \subseteq R$  .

Пусть кортеж  $r=(x,y,z)\in R_1$  ЈОІN  $R_2$  ЈОІN  $R_3$ . Это означает, что в проекции  $R_1=R[X,Y]$  содержится кортеж  $r_1'=(x,y)$ , в проекции  $R_2=R[X,Z]$  содержится кортеж  $r_2'=(x,z)$ , а в проекции  $R_3=R[Y,Z]$  содержится кортеж  $r_3'=(y,z)$ . По определению проекции, найдутся такие значения  $x_1$ ,  $y_1$ ,  $z_1$  атрибутов x,  $y_1$   $z_2$  соответственно, что отношение  $x_2$  содержит кортежи  $x_1=(x,y,z_1)$ ,  $x_2=(x,y_1,z)$   $x_3=(x_1,y,z)$ . Но тогда по условию (іі) в отношении  $x_1=(x,y,z_1)$  содержится также и кортеж  $x_1=(x,y,z_1)$  . Этим доказано необходимое включение.  $x_1=(x,y,z_1)$  диверждение доказано.

## Продолжение алгоритма нормализации (приведение к 5НФ)

В предыдущей главе был описан алгоритм нормализации как алгоритм приведения отношений к  $3H\Phi$ . Теперь мы можем продолжить этот алгоритм, доведя его до алгоритма приведения к  $5H\Phi$ .

*Шаг 4 (Приведение к НФБК)*. Если имеются отношения, содержащие несколько потенциальных ключей, то необходимо проверить, имеются ли функциональные зависимости, детерминанты которых не являются потенциальными ключами. Если такие функциональные зависимости имеются, то необходимо провести дальнейшую декомпозицию отношений. Те атрибуты, которые зависят от детерминантов, не являющихся потенциальными ключами выносятся в отдельное отношение вместе с детерминантами.

*Шаг 5 (Приведение к 4НФ)*. Если в отношениях обнаружены нетривиальные многозначные зависимости, то необходимо провести декомпозицию для исключения таких зависимостей.

*Шаг 5 (Приведение к 5НФ)*. Если в отношениях обнаружены нетривиальные зависимости соединения, то необходимо провести декомпозицию для исключения и таких зависимостей.

#### Выводы

Обобщением  $3H\Phi$  на случай, когда отношение имеет более одного потенциального ключа, является нормальная форма Бойса-Кодда.

Отношение R находится в *нормальной форме Бойса-Кодда* ( $H\Phi EK$ ) тогда и только тогда, когда детерминанты всех функциональных зависимостей являются потенциальными ключами.

Нормализация отношений вплоть до нормальной формы Бойса-Кодда основывалась на понятии функциональной зависимости и теореме Хеза, гарантировавшей, что декомпозиция будет происходить без потерь информации.

Дальнейшая нормализация связана уже с обобщением понятия функциональной зависимости.

Атрибуты (множества атрибутов) Y и Z **многозначно зависям** от X, ( $X \longrightarrow Y \mid Z$ ), тогда и только тогда, когда из того, что в отношении R содержатся кортежи  $r_1 = (x, y, z_1)_{\mathsf{U}}$   $r_2 = (x, y_1, z)_{\mathsf{Cледует}}$ , что в отношении R содержится также и кортеж  $\mathsf{K}^{r_3} = (x, y, z)_{\mathsf{C}}$ .

Корректность дальнейшей декомпозиции основывается на теореме Фейджина, которая говорит о том, что декомпозиция отношения на две проекции является декомпозицией без потерь тогда и только тогда, когда в отношении имеется некоторая многозначная зависимость.

Если в отношении имеется функциональная зависимость, то автоматически имеется и тривиальная многозначная зависимость, определяемая этой функциональной зависимостью.

Многозначная зависимость  $X \longrightarrow Y \mid Z$  называется **нетривиальной многозначной** зависимостью, если не существует функциональных зависимостей  $X \to Y$  и  $X \to Z$ .

Отношение R находится в **четвертой нормальной форме** (**4**H**Ф**) тогда и только тогда, когда отношение находится в НФБК и не содержит *нетривиальных* многозначных зависимостей.

Имеют место зависимости специального вида, когда отношение не может быть подвергнуто декомпозиции без потерь на две проекции, но может быть декомпозировано на большее число проекций. Такие зависимости называются зависимостями соединения и являются обобщением понятия многозначной зависимости.

Отношение R находится в **пятой нормальной форме** ( $5H\Phi$ ) тогда и только тогда, когда любая имеющаяся зависимость соединения является тривиальной.

## Глава 8. Элементы модели "сущность-связь"

Моделирование структуры базы данных при помощи алгоритма нормализации, описанного в предыдущих главах, имеет серьезные недостатки:

- 1. Первоначальное размещение всех атрибутов в одном отношении является очень неестественной операцией. Интуитивно разработчик сразу проектирует несколько отношений в соответствии с обнаруженными сущностями. Даже если совершить насилие над собой и создать одно или несколько отношений, включив в них все предполагаемые атрибуты, то совершенно неясен смысл полученного отношения.
- 2. Невозможно сразу определить полный список атрибутов. Пользователи имеют привычку называть разными именами одни и те же вещи или наоборот, называть одними именами разные вещи.

3. Для проведения процедуры нормализации необходимо выделить зависимости атрибутов, что тоже очень нелегко, т.к. необходимо *явно выписать все зависимости*, даже те, которые являются очевидными.

В реальном проектировании структуры базы данных применяются другой метод - так называемое, *семантическое моделирование*. Семантическое моделирование представляет собой моделирование структуры данных, опираясь на смысл этих данных. В качестве инструмента семантического моделирования используются различные варианты *диаграмм сущность-связь* (*ER - Entity-Relationship*).

Первый вариант модели сущность-связь был предложен в 1976 г. Питером Пин-Шэн Ченом [37]. В дальнейшем многими авторами были разработаны свои варианты подобных моделей (нотация Мартина, нотация IDEF1X, нотация Баркера и др.). Кроме того, различные программные средства, реализующие одну и ту же нотацию, могут отличаться своими возможностями. По сути, все варианты диаграмм сущность-связь исходят из одной идеи - рисунок всегда нагляднее текстового описания. Все такие диаграммы используют графическое изображение сущностей предметной области, их свойств (атрибутов), и взаимосвязей между сущностями.

Мы опишем работу с ER-диаграммами близко к нотации Баркера, как довольно легкой в понимании основных идей. Данная глава является скорее иллюстрацией методов семантического моделирования, чем полноценным введением в эту область.

#### Основные понятия ER-диаграмм

*Определение 1. Сущность* - это класс однотипных объектов, информация о которых должна быть учтена в модели.

Каждая сущность должна иметь наименование, выраженное существительным в единственном числе.

Примерами сущностей могут быть такие классы объектов как "Поставщик", "Сотрудник", "Накладная".

Каждая сущность в модели изображается в виде прямоугольника с наименованием:



#### Рис. 1

Определение 2. Экземпляр сущности - это конкретный представитель данной сущности.

Например, представителем сущности "Сотрудник" может быть "Сотрудник Иванов".

Экземпляры сущностей должны быть *различимы*, т.е. сущности должны иметь некоторые свойства, уникальные для каждого экземпляра этой сущности.

*Определение 3. Атрибут сущности* - это именованная характеристика, являющаяся некоторым свойством сущности.

Наименование атрибута должно быть выражено существительным в единственном числе (возможно, с характеризующими прилагательными).

Примерами атрибутов сущности "Сотрудник" могут быть такие атрибуты как "Табельный номер", "Фамилия", "Имя", "Отчество", "Должность", "Зарплата" и т.п.

Атрибуты изображаются в пределах прямоугольника, определяющего сущность:



Рис. 2

Определение 4. **Ключ сущности** - это неизбыточный набор атрибутов, значения которых в совокупности являются *уникальными* для каждого экземпляра сущности. Неизбыточность заключается в том, что удаление любого атрибута из ключа нарушается его уникальность.

Сущность может иметь несколько различных ключей.

Ключевые атрибуты изображаются на диаграмме подчеркиванием:

Сотрудник	
<u>Табельный номер</u>	
Фамилия	
Имя	
Отчество	
Должность	
Зарплата	

Рис. 3

*Определение 5.* **Связь** - это некоторая ассоциация между *двумя* сущностями. Одна сущность может быть связана с другой сущностью или сама с собою.

Связи позволяют по одной сущности находить другие сущности, связанные с нею.

Например, связи между сущностями могут выражаться следующими фразами - "СОТРУДНИК может иметь несколько ДЕТЕЙ", "каждый СОТРУДНИК обязан числиться ровно в одном ОТДЕЛЕ".

Графически связь изображается линией, соединяющей две сущности:



Рис. 4

Каждая связь имеет два конца и одно или два наименования. Наименование обычно выражается в неопределенной глагольной форме: "иметь", "принадлежать" и т.п. Каждое из наименований относится к своему концу связи. Иногда наименования не пишутся ввиду их очевидности.

Каждая связь может иметь один из следующих *типов связи*:

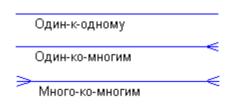


Рис. 5

Связь типа *один-к-одному* означает, что один экземпляр первой сущности (левой) связан с одним экземпляром второй сущности (правой). Связь один-к-одному чаще всего свидетельствует о том, что на самом деле мы имеем всего одну сущность, неправильно разделенную на две.

Связь типа *один-ко-многим* означает, что один экземпляр первой сущности (левой) связан с несколькими экземплярами второй сущности (правой). Это наиболее часто используемый тип связи. Левая сущность (со стороны "один") называется *родительской*, правая (со стороны "много") - *дочерней*. Характерный пример такой связи приведен на Рис. 4.

Связь типа *много-ко-многим* означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности. Тип связи много-ко-многим является *временным* типом связи, допустимым на ранних этапах разработки модели. В дальнейшем этот тип связи должен быть заменен двумя связями типа один-ко-многим путем создания промежуточной сущности.

Каждая связь может иметь одну из двух модальностей связи:

Может Должен

Рис. 6

Модальность "*может*" означает, что экземпляр одной сущности *может быть связан* с одним или несколькими экземплярами другой сущности, *а может быть и не связан* ни с одним экземпляром.

Модальность "*должен*" означает, что экземпляр одной сущности *обязан быть связан не менее чем с одним* экземпляром другой сущности.

Связь может иметь разную модальность с разных концов (как на Рис. 4).

Описанный графический синтаксис позволяет *однозначно* читать диаграммы, пользуясь следующей схемой построения фраз:

<Каждый экземпляр СУЩНОСТИ 1> <МОДАЛЬНОСТЬ СВЯЗИ> <НАИМЕНОВАНИЕ СВЯЗИ> <ТИП СВЯЗИ> <экземпляр СУЩНОСТИ 2>.

Каждая связь может быть прочитана как слева направо, так и справа налево. Связь на Рис. 4 читается так:

Слева направо: "каждый сотрудник может иметь несколько детей".

Справа налево: "Каждый ребенок обязан принадлежать ровно одному сотруднику".

### Пример разработки простой ER-модели

При разработке ER-моделей мы должны получить следующую информацию о предметной области:

- 1. Список сущностей предметной области.
- 2. Список атрибутов сущностей.
- 3. Описание взаимосвязей между сущностями.

ER-диаграммы удобны тем, что процесс выделения сущностей, атрибутов и связей является итерационным. Разработав первый приближенный вариант диаграмм, мы уточняем их, опрашивая экспертов предметной области. При этом документацией, в которой фиксируются результаты бесед, являются сами ER-диаграммы.

Предположим, что перед нами стоит задача разработать информационную систему по заказу некоторой оптовой торговой фирмы. В первую очередь мы должны изучить предметную область и процессы, происходящие в ней. Для этого мы опрашиваем сотрудников фирмы, читаем документацию, изучаем формы заказов, накладных и т.п.

Например, в ходе беседы с менеджером по продажам, выяснилось, что он (менеджер) считает, что проектируемая система должна выполнять следующие действия:

- Хранить информацию о покупателях.
- Печатать накладные на отпущенные товары.
- Следить за наличием товаров на складе.

Выделим все существительные в этих предложениях - это будут потенциальные кандидаты на сущности и атрибуты, и проанализируем их (непонятные термины будем выделять знаком вопроса):

- Покупатель явный кандидат на сущность.
- Накладная явный кандидат на сущность.
- Товар явный кандидат на сущность
- (?)Склад а вообще, сколько складов имеет фирма? Если несколько, то это будет кандидатом на новую сущность.
- (?)Наличие товара это, скорее всего, атрибут, но атрибут какой сущности?

Сразу возникает очевидная связь между сущностями - "покупатели могут покупать много товаров" и "товары могут продаваться многим покупателям". Первый вариант диаграммы выглядит так:



Рис. 7

Задав дополнительные вопросы менеджеру, мы выяснили, что фирма имеет несколько складов. Причем, каждый товар может храниться на нескольких складах и быть проданным с любого склада.

Куда поместить сущности "Накладная" и "Склад" и с чем их связать? Спросим себя, как связаны эти сущности между собой и с сущностями "Покупатель" и "Товар"? Покупатели покупают товары, получая при этом накладные, в которые внесены данные о количестве и цене купленного товара. Каждый покупатель может получить несколько накладных. Каждая накладная обязана выписываться на одного покупателя. Каждая накладная обязана содержать несколько товаров (не бывает пустых накладных). Каждый товар, в свою очередь, может быть продан нескольким покупателям через несколько накладных. Кроме того, каждая накладная должна быть выписана с определенного склада, и с любого склада может быть выписано много накладных. Таким образом, после уточнения, диаграмма будет выглядеть следующим образом:

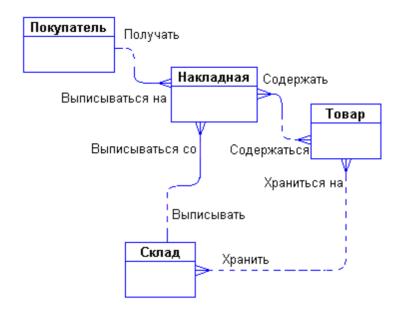


Рис. 8

Пора подумать об атрибутах сущностей. Беседуя с сотрудниками фирмы, мы выяснили следующее:

- Каждый покупатель является юридическим лицом и имеет наименование, адрес, банковские реквизиты.
- Каждый товар имеет наименование, цену, а также характеризуется единицами измерения.
- Каждая накладная имеет уникальный номер, дату выписки, список товаров с количествами и ценами, а также общую сумму накладной. Накладная выписывается с определенного склада и на определенного покупателя.
- Каждый склад имеет свое наименование.
- Снова выпишем все существительные, которые будут потенциальными атрибутами, и проанализируем их:
- *Юридическое лицо* термин риторический, мы не работаем с физическими лицами. Не обращаем внимания.
- Наименование покупателя явная характеристика покупателя.
- Адрес явная характеристика покупателя.
- Банковские реквизиты явная характеристика покупателя.
- Наименование товара явная характеристика товара.
- (?) Цена товара похоже, что это характеристика товара. Отличается ли эта характеристика от цены в накладной?
- Единица измерения явная характеристика товара.
- Номер накладной явная уникальная характеристика накладной.
- Дата накладной явная характеристика накладной.
- (?) Список товаров в накладной список не может быть атрибутом. Вероятно, нужно выделить этот список в отдельную сущность.
- (?) Количество товара в накладной это явная характеристика, но характеристика чего? Это характеристика не просто "товара", а "товара в накладной".
- (?) Цена товара в накладной опять же это должна быть не просто характеристика товара, а характеристика товара в накладной. Но цена товара уже встречалась выше это олно и то же?

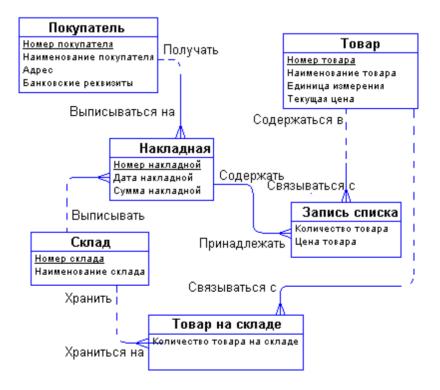
- *Сумма накладной* явная характеристика накладной. Эта характеристика не является независимой. Сумма накладной равна сумме стоимостей всех товаров, входящих в накладную.
- Наименование склада явная характеристика склада.

В ходе дополнительной беседы с менеджером удалось прояснить различные понятия цен. Оказалось, что каждый товар имеет некоторую текущую цену. Эта цена, по которой товар продается в данный момент. Естественно, что эта цена может меняться со временем. Цена одного и того же товара в разных накладных, выписанных в разное время, может быть различной. Таким образом, имеется две цены - цена товара в накладной и текущая цена товара.

С возникающим понятием "Список товаров в накладной" все довольно ясно. Сущности "Накладная" и "Товар" связаны друг с другом отношением типа много-ко-многим. Такая связь, как мы отмечали ранее, должна быть расщеплена на две связи типа один-комногим. Для этого требуется дополнительная сущность. Этой сущностью и будет сущность "Список товаров в накладной". Связь ее с сущностями "Накладная" и "Товар" характеризуется следующими фразами - "каждая накладная обязана иметь несколько записей из списка товаров в накладной", "каждая запись из списка товаров в накладной обязана включаться ровно в одну накладную", "каждый товар может включаться в несколько записей из списка товаров в накладной", " каждая запись из списка товаров в накладной обязана быть связана ровно с одним товаром". Атрибуты "Количество товара в накладной" и "Цена товара в накладной" являются атрибутами сущности " Список товаров в накладной".

Точно также поступим со связью, соединяющей сущности "Склад" и "Товар". Введем дополнительную сущность "Товар на складе". Атрибутом этой сущности будет "Количество товара на складе". Таким образом, товар будет числиться на любом складе и количество его на каждом складе будет свое.

Теперь можно внести все это в диаграмму:



### Концептуальные и физические ER-модели

Разработанный выше пример ER-диаграммы является примером концептуальной онаграммы. Это означает, что диаграмма не учитывает особенности конкретной СУБД. По данной концептуальной диаграмме можно построить физическую онаграмму, которая уже будут учитываться такие особенности СУБД, как допустимые типы и наименования полей и таблиц, ограничения целостности и т.п. Физический вариант диаграммы, приведенной на Рис. 9 может выглядеть, например, следующим образом:

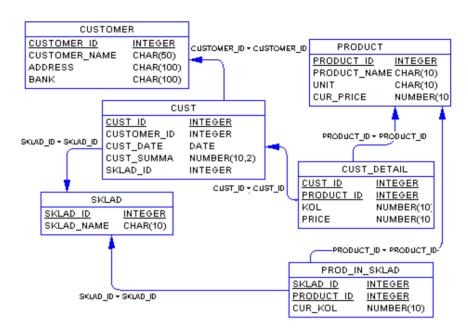


Рис. 10

На данной диаграмме каждая сущность представляет собой таблицу базы данных, каждый атрибут становится колонкой соответствующей таблицы. Обращаем внимание на то, что во многих таблицах, например, "CUST\_DETAIL" и "PROD\_IN\_SKLAD", соответствующих сущностям "Запись списка накладной" и "Товар на складе", появились новые атрибуты, которых не было в концептуальной модели - это ключевые атрибуты родительских таблиц, *мигрировавших* в дочерние таблицы для того, чтобы обеспечить связь между таблицами посредством внешних ключей.

Легко заметить, что полученные таблицы сразу находятся в ЗНФ.

### Выводы

Реальным средством моделирования данных является не формальный метод нормализации отношений, а так называемое семантическое моделирование.

В качестве инструмента семантического моделирования используются различные варианты *диаграмм сущность-связь* (*ER - Entity-Relationship*).

Диаграммы сущность-связь позволяют использовать наглядные графические обозначения для моделирования сущностей и их взаимосвязей.

Различают концептуальные и физические ER-диаграммы. Концептуальные диаграммы не учитывают особенностей конкретных СУБД. Физические диаграммы строятся по концептуальным и представляют собой прообраз конкретной базы данных. Сущности, определенные в концептуальной диаграмме становятся таблицами, атрибуты становятся колонками таблиц (при этом учитываются допустимые для данной СУБД типы данных и наименования столбцов), связи реализуются путем миграции ключевых атрибутов родительских сущностей и создания внешних ключей.

При правильном определении сущностей, полученные таблицы будут сразу находиться в 3HФ. Основное достоинство метода состоит в том, модель строится методом последовательных уточнений первоначальных диаграмм.

В данной главе, являющейся иллюстрацией к методам ER-моделирования, не рассмотрены более сложные аспекты построения диаграмм, такие как подтипы, роли, исключающие связи, непереносимые связи, идентифицирующие связи и т.п.

## Глава 9. Транзакции и целостность баз данных

В данной и в последующих главах изучается фундаментальное понятие транзакции. Это понятие не входит в реляционную модель данных, т.к. транзакции рассматриваются не только в реляционных СУБД, но и в СУБД других типов, а также и в других типах информационных систем.

Транзакция - это неделимая, с точки зрения воздействия на СУБД, последовательность операций манипулирования данными. Для пользователя транзакция выполняется по принципу "все или ничего", т.е. либо транзакция выполняется целиком и переводит базу данных из одного целостного состояния в другое целостное состояние, либо, если по каким-либо причинам, одно из действий транзакции невыполнимо, или произошло какоелибо нарушение работы системы, база данных возвращается в исходное состояние, которое было до начала транзакции (происходит откат транзакции). С этой точки зрения, транзакции важны как в многопользовательских, так и в однопользовательских системах. В однопользовательских системах транзакции - это логические единицы работы, после выполнения которых база данных остается в целостном состоянии. Транзакции также являются единицами восстановления данных после сбоев - восстанавливаясь, система ликвидирует следы транзакций, не успевших успешно завершиться в результате программного или аппаратного сбоя. Эти два свойства транзакций определяют атомарность (неделимость) транзакции. В многопользовательских системах, кроме того, транзакции служат для обеспечения изолированной работы отдельных пользователей пользователям, одновременно работающим с одной базой данных, кажется, что они работают как бы в однопользовательской системе и не мешают друг другу.

### Пример нарушения целостности базы

Для иллюстрации возможного нарушения целостности базы данных рассмотрим следующий пример:

<u>Пример 1</u>. Пусть имеется система, в которой хранятся данные о подразделениях и работающих в них сотрудниках. Список подразделений хранится в таблице DEPART(Dep\_Id, Dep\_Name, Dept\_Kol), где Dept\_Id - идентификатор подразделения, Dept\_Name - наименование подразделения, Dept\_Kol - количество сотрудников в подразделении. Список сотрудников хранится в таблице PERSON(Pers\_Id, Pers\_Name,

Dept\_Id), где Pers\_Id - идентификатор сотрудника, Pers\_Name - имя сотрудника, Dept\_Id - идентификатор подразделения, в котором работает сотрудник:

Dept_Id	Dept_Name	Dept_Kol
1	Кафедра алгебры	3
2	Кафедра программирования	2

Таблина 1 DEPART

Pers_Id	Pers_Name	Dept_Id
1	Иванов	1
2	Петров	2
3	Сидоров	1
4	Пушников	2
5	Шарипов	1

Таблица 2 PERSON

Ограничение целостности этой базы данных состоит в том, что поле Dept\_Kol не может заполняться произвольными значениями - это поле должно содержать количество сотрудников, реально числящихся в подразделении.

С учетом этого ограничения можно заключить, что вставка нового сотрудника в таблицу *не может быть выполнена одной операцией*. При вставке нового сотрудника необходимо одновременно увеличить значение поля Dept\_Kol:

- Шаг 1. Вставить сотрудника в таблицу PERSON: INSERT INTO PERSON (6, Муфтахов, 1)
- Шаг 2. Увеличить значение поля Dept\_Kol: UPDATE DEPART SET Dept=Dept+1 WHERE Dept\_Id=1

Если после выполнения первой операции и до выполнения второй произойдет сбой системы, то реально будет выполнена только первая операция и база данных остается в нецелостном состоянии.

#### Понятие транзакции

Определение 1. **Транзакция** - это последовательность операторов манипулирования данными, выполняющаяся *как единое целое* (все или ничего) и переводящая базу данных из одного целостного состояния в другое целостное состояние.

Транзакция обладает четырьмя важными свойствами, известными как свойства АСИД:

- (*A*) *Атомарность*. Транзакция выполняется как атомарная операция либо выполняется вся транзакция целиком, либо она целиком не выполняется.
- (*C*) Согласованность. Транзакция переводит базу данных из одного согласованного (целостного) состояния в другое согласованное (целостное) состояние. Внутри транзакции согласованность базы данных может нарушаться.

- (*И*) *Изоляция*. Транзакции разных пользователей не должны мешать друг другу (например, как если бы они выполнялись строго по очереди).
- (Д) Долговечность. Если транзакция выполнена, то результаты ее работы должны сохраниться в базе данных, даже если в следующий момент произойдет сбой системы.

Транзакция обычно начинается автоматически с момента присоединения пользователя к СУБД и продолжается до тех пор, пока не произойдет одно из следующих событий:

- Подана команда COMMIT WORK (зафиксировать транзакцию).
- Подана команда ROLLBACK WORK (откатить транзакцию).
- Произошло отсоединение пользователя от СУБД.
- Произошел сбой системы.

Команда COMMIT WORK завершает текущую транзакцию и автоматически начинает новую транзакцию. При этом гарантируется, что результаты работы завершенной транзакции фиксируются, т.е. сохраняются в базе данных.

<u>Замечание</u>. Некоторые системы (например, Visual FoxPro), требуют подать явную команду BEGIN TRANSACTION для того, чтобы начать новую транзакцию.

Команда ROLLBACK WORK приводит к тому, что все изменения, сделанные текущей транзакцией откатываются, т.е. отменяются так, как будто их вообще не было. При этом автоматически начинается новая транзакция.

При отсоединении пользователя от СУБД происходит автоматическая фиксация транзакций.

При сбое системы происходят более сложные процессы. Кратко суть их сводится к тому, что при последующем запуске системы происходит анализ выполнявшихся до момента сбоя транзакций. Те транзакции, для которых была подана команда COMMIT WORK, но результаты работы которых не были занесены в базу данных выполняются снова (накатываются). Те транзакции, для которых не была подана команда COMMIT WORK, откатываются. Более подробно восстановление после сбоев рассматривается далее.

Свойства АСИД транзакций не всегда выполняются в полном объеме. Особенно это относится к свойству И (изоляция). В идеале, транзакции разных пользователей не должны мешать друг другу, т.е. они должны выполняться так, чтобы у пользователя создавалась иллюзия, что он в системе один. Простейший способ обеспечить абсолютную изолированность состоит в том, чтобы выстроить транзакции в очередь и выполнять их строго одну за другой. Очевидно, при этом теряется эффективность работы системы. Поэтому реально одновременно выполняется несколько транзакций (смесь транзакций). Различается несколько уровней изоляции транзакций. На низшем уровне изоляции транзакции могут реально мешать друг другу, на высшем они полностью изолированы. За большую изоляцию транзакций приходится платить большими накладными расходами системы и замедлением работы. Пользователи или администратор системы могут по своему усмотрению задавать различные уровни всех или отдельных транзакций. Более подробно изоляция транзакций рассматривается в следующей главе.

Свойство Д (долговечность) также не является абсолютными свойством, т.к. некоторые системы допускают вложенные транзакции. Если транзакция Б запущена внутри транзакции А, и для транзакции Б подана команда COMMIT WORK, то фиксация данных

транзакции Б является условной, т.к. внешняя транзакция А может откатиться. Результаты работы внутренней транзакции Б будут окончательно зафиксированы только если будет зафиксирована внешняя транзакция А.

### Ограничения целостности

Свойство (С) - согласованность транзакций определяется наличием понятия согласованности базы данных.

*Определение 2. Ограничение целостности* - это некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния базы данных.

Примерами ограничений целостности могут служить следующие утверждения:

<u>Пример 2</u>. Возраст сотрудника не может быть меньше 18 и больше 65 лет.

<u>Пример 3</u>. Каждый сотрудник имеет уникальный табельный номер.

Пример 4. Сотрудник обязан числиться в одном отделе.

<u>Пример 5</u>. Сумма накладной обязана равняться сумме произведений цен товаров на количество товаров для всех товаров, входящих в накладную.

Как видно из этих примеров, некоторые из ограничений целостности являются ограничениями реляционной модели данных (см. гл. 3). Пример 3 представляет ограничение, реализующее целостность сущности. Пример 4 представляет ограничение, реализующее ссылочную целостность. Другие ограничения являются достаточно произвольными утверждениями (примеры 2 и 5). Любое ограничение целостности является семантическим понятием, т.е. появляется как следствие определенных свойств объектов предметной области и/или их взаимосвязей.

*Определение 3*. База данных находится в *согласованном* (*целостном*) *состоянии*, если выполнены (удовлетворены) все ограничения целостности, определенные для базы данных.

В данном определении важно подчеркнуть, что должны быть выполнены не все вообще ограничения предметной области, а только те, которые *определены* в базе данных. Для этого необходимо, чтобы СУБД *обладала* развитыми средствами поддержки ограничений целостности. Если какая-либо СУБД не может отобразить все необходимые ограничения предметной области, то такая база данных хотя и будет находиться в целостном состоянии с точки зрения СУБД, но это состояние не будет правильным с точки зрения пользователя.

Таким образом, согласованность базы данных есть формальное свойство базы данных. База данных не понимает "смысла" хранимых данных. "Смыслом" данных для СУБД является весь набор ограничений целостности. Если все ограничения выполнены, то СУБД считает, что данные корректны.

Вместе с понятием целостности базы данных возникает понятие *реакции системы на попытку нарушения целостности*. Система должна не только проверять, не нарушаются ли ограничения в ходе выполнения различных операций, но и должным образом реагировать, если операция приводит к нарушению целостности. Имеется два типа реакции на попытку нарушения целостности:

- 1. Отказ выполнить "незаконную" операцию.
- 2. Выполнение компенсирующих действий.

Например, если система знает, что в поле "Возраст\_Сотрудника" должны быть целые числа в диапазоне от 18 до 65, то система отвергает попытку ввести значение возраста 66. При этом может генерироваться какое-нибудь сообщение для пользователя.

В противоположность этому, в примере 1 система допускает вставку записи о новом сотруднике (что приводит к нарушению целостности базы данных), но *автоматически* производит компенсирующие действия, изменяя значение поля Dept\_Kol в таблице DEPART.

Работу системы по проверке ограничений можно изобразить на следующем рисунке:

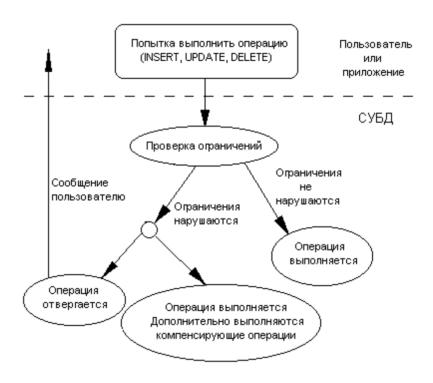


Рисунок 1

В некоторых случаях система может не выполнять проверку на нарушение ограничений, а сразу выполнять компенсирующие операции. Действительно, в примере 1 при вставке или удалении сотрудника проверку производить не нужно, т.к. результаты ее известны заранее - ограничение обязательно будет нарушено. В этом случае необходимо сразу приступать к компенсированию возникшего нарушения.

#### Классификация ограничений целостности

Ограничения целостности можно классифицировать несколькими способами:

- По способам реализации.
- По времени проверки.
- По области действия.

#### Классификация ограничений целостности по способам реализации

Каждая система обладает своими средствами поддержки ограничений целостности. Различают два способа реализации:

- Декларативная поддержка ограничений целостности.
- Процедурная поддержка ограничений целостности.

Определение 4. Декларативная поддержка ограничений целостности заключается в определении ограничений средствами языка определения данных (DDL - Data Definition Language). Обычно средства декларативной поддержки целостности (если они имеются в СУБД) определяют ограничения на значения доменов и атрибутов, целостность сущностей (потенциальные ключи отношений) и ссылочную целостность (целостность внешних ключей). Декларативные ограничения целостности можно использовать при создании и модификации таблиц средствами языка DDL или в виде отдельных утверждений (ASSERTION).

Например, следующий оператор создает таблицу PERSON и определяет для нее некоторые ограничения целостности:

```
CREATE TABLE PERSON

(Pers_Id INTEGER PRIMARY KEY,
Pers_Name CHAR(30) NOT NULL,
Dept_Id REFERENCES DEPART(Dept_Id) ON UPDATE CASCADE ON DELETE CASCADE);
```

После выполнения оператора для таблицы PERSON будут объявлены следующие ограничения целостности:

- Поле Pers\_Id образует потенциальный ключ отношения.
- Поле Pers\_Name не может содержать null-значений.
- Поле Dept\_Id является внешней ссылкой на родительскую таблицу DEPART, причем, при изменении или удалении строки в родительской таблице *каскадно* должны быть внесены соответствующие изменения в дочернюю таблицу.

Средства декларативной поддержки ограничений описаны в стандарте SQL и более подробно рассматриваются ниже.

*Определение 5.* **Процедурная поддержка ограничений целостности** заключается в использовании триггеров и хранимых процедур.

Не все ограничения целостности можно реализовать декларативно. Примером такого ограничения может служить требование из примера 1, утверждающее, что поле Dept\_Kol таблицы DEPART должно содержать количество сотрудников, реально числящихся в подразделении. Для реализации этого ограничения необходимо создать триггер, запускающийся при вставке, модификации и удалении записей в таблице PERSON, который корректно изменяет значение поля Dept\_Kol. Например, при вставке в таблицу PERSON новой строки, триггер увеличивает на единицу значение поля Dept\_Kol, а при удалении строки - уменьшает. Заметим, что при модификации записей в таблице PERSON могут потребоваться даже более сложные действия. Действительно, модификация записи в таблице PERSON может заключаться в том, что мы переводим сотрудника из одного отдела в другой, меняя значение в поле Dept\_Id. При этом необходимо в старом подразделении уменьшить количество сотрудников, а в новом - увеличить.

Кроме того, необходимо защититься от неправильной модификации строк таблицы DEPART. Действительно, пользователь может попытаться модифицировать запись об

отделе, введя неверное значение поля Dept\_Kol. Для предотвращения подобных действий необходимо создать также триггеры, запускающиеся при вставке и модификации записей в таблице DEPART. Триггер, запускающийся при удалении записей из таблицы DEPART не нужен, т.к. уже имеется ограничение ссылочной целостности, каскадно удаляющее записи из таблицы PERSON при удалении записи из таблицы DEPART.

По сути, наличие ограничения целостности (как декларативного, так и процедурного характера) *всегда* приводит к созданию или использованию некоторого программного кода, реализующего это ограничение. Разница заключается в том, где такой код хранится и как он создается.

Если ограничение целостности реализовано в виде триггеров, то этот программный код является просто телом триггера. Если используется декларативное ограничение целостности, то возможны два подхода:

- 1. При декларировании (объявлении) ограничения текст ограничения хранится в виде некоторого объекта СУБД, а для реализации ограничения используются встроенные в СУБД функции, и тогда этот код представляет собой внутренние функции ядра СУБД.
- 2. При декларировании ограничения СУБД автоматически генерирует триггеры, выполняющие необходимые действия по проверке ограничений.

Примером использования функций ядра для проверки декларативных ограничений является автоматическая проверка уникальности индексов, соответствующих потенциальным ключам отношений. В качестве другого примера можно привести поддержку ссылочной целостности средствами СУБД ORACLE. Ограничение ссылочной целостности является в ORACLE объектом базы данных, хранящим формулировку этого ограничения. Проверка ограничения выполняется функциями ядра ORACLE со ссылкой на этот объект. Ограничение целостности в этом случае нельзя модифицировать иначе, как используя декларативные операторы создания и модификации ограничений.

Примером генерации новых триггеров для реализации декларативных ограничений, может служить система Visual FoxPro. Триггеры, автоматически сгенерированные Visual FoxPro при объявлении ограничений ссылочной целостности можно посмотреть и даже внести в них изменения, так чтобы они могли выполнять некоторые дополнительные действия.

Если система не поддерживает ни декларативную поддержку ссылочной целостности, ни триггеры (как, например, FoxPro 2.5), то программный код, следящий за корректностью базы данных, приходится размещать в пользовательском приложении (такой ведь код все равно необходим!). Это сильно затрудняет разработку программ и не защищает от попыток пользователей напрямую внести некорректные данные в базу данных. Особенно сложно становится в том случае, когда имеется сложная база данных и множество различных приложений, работающих с ней (например, к базе данных торгового предприятия может обращаться несколько приложений, таких как "Складской учет", "Прием заказов", "Главный бухгалтер" и т.п.). Каждое из таких приложений должно содержать один и тот же код, отвечающий за поддержание целостности базы данных. Особенно весело становится разработчику, когда необходимо внести изменения в логику поддержания целостности. Приходится заменять во всех программах одни и те же места, перекомпилировать все приложения и распространять по рабочим местам новые версии.

Классификация ограничений целостности по времени проверки

По времени проверки ограничения делятся на:

- Немедленно проверяемые ограничения.
- Ограничения с отложенной проверкой.

Определение 6. **Немедленно проверяемые ограничения** проверяются непосредственно в момент выполнения операции, могущей нарушить ограничение. Например, проверка уникальности потенциального ключа проверяется в момент вставки записи в таблицу. Если ограничение нарушается, то такая операция отвергается. Транзакция, внутри которой произошло нарушение немедленно проверяемого утверждения целостности, обычно откатывается.

Определение 7. Ограничения с отложенной проверкой проверяется в момент фиксации транзакции оператором COMMIT WORK. Внутри транзакции ограничение может не выполняться. Если в момент фиксации транзакции обнаруживается нарушение ограничения с отложенной проверкой, то транзакция откатывается. Примером ограничения, которое не может быть проверено немедленно является ограничение из примера 1. Это происходит оттого, что транзакция, заключающаяся во вставке нового сотрудника в таблицу PERSON, состоит не менее чем из двух операций - вставки строки в таблицу PERSON и обновления строки в таблице DEPART. Ограничение, безусловно, неверно после первой операции и становится верным после второй операции.

### Классификация ограничений целостности по области действия

По области действия ограничения делятся на:

- Ограничения домена
- Ограничения атрибута
- Ограничения кортежа
- Ограничения отношения
- Ограничения базы данных

### Ограничения домена

Определение 8. Ограничения целостности домена представляют собой ограничения, накладываемые только на допустимые значения домена. Фактически, ограничения домена обязаны являться частью определения домена (см. определение домена в гл. 2).

Например, ограничением домена "Возраст сотрудника" может быть условие "Возраст сотрудника не менее 18 и не более 65".

<u>Проверка ограничения</u>. Ограничения домена сами по себе не проверяются. Если на какомлибо домене основан атрибут, то ограничение соответствующего домена становится ограничением этого атрибута.

### Ограничения атрибута

Определение 9. **Ограничение целостности атрибута** представляют собой ограничения, накладываемые на допустимые значения атрибута вследствие того, что атрибут основан на каком-либо домене. Ограничение атрибута в точности совпадают с ограничениями соответствующего домена. Отличие ограничений атрибута от ограничений домена в том, что ограничения атрибута *проверяются*.

Если логика предметной области такова, что на значения атрибута необходимо наложить дополнительные ограничения, помимо ограничений домена, то такие ограничения переходят в следующую категорию.

<u>Проверка ограничения</u>. Ограничение атрибута является *немедленно проверяемым* ограничением. Действительно, ограничение атрибута не зависит ни от каких других объектов базы данных, кроме домена, на котором основан атрибут. Поэтому никакие изменения в других объектах не могут повлиять на истинность ограничения.

#### Ограничения кортежа

Определение 10. Ограничения целостности кортежа представляют собой ограничения, накладываемые на допустимые значения *отдельного* кортежа отношения, и *не являющиеся* ограничением целостности атрибута. Требование, что ограничение относится к *отдельному* кортежу отношения, означает, что для его проверки *не требуется* никакой информации о других кортежах отношения.

<u>Пример 6</u>. Атрибут "Возраст сотрудника" в таблице "Спецподразделение", может иметь дополнительное ограничение "Возраст сотрудника не менее 25 и не более 45", помимо того, что этот атрибут уже имеет ограничение, определяемое доменом - "Возраст сотрудника не менее 18 и не более 65".

Приведенное ограничение кортежа, по сути, является дополнительным ограничением на значения *одного* атрибута. В этом случае допустимы два решения. Можно объявить новый домен "Возраст сотрудника спецподразделения" и тогда ограничение кортежа становится ограничением домена и атрибута, либо рассматривать это ограничение именно как ограничение кортежа. Оба решения имеют свои положительные и отрицательные стороны.

Замечание. Тут имеются некоторые возможности для оптимизации. Формально, при изменении значения данного атрибута необходимо проверить два ограничения - ограничение атрибута и ограничение кортежа. Но в данном случае ограничение кортежа сильнее ограничения атрибута и достаточно проверить только ограничение кортежа. Разумно построенная СУБД могла бы выявлять такие случаи и уменьшать лишнюю работу.

<u>Пример 7</u>. Для отношения "Сотрудники" можно сформулировать следующее ограничение: если атрибут "Должность" принимает значение "Директор", то атрибут "Зарплата" содержит значение не менее 1000\$.

Это ограничение связывает два атрибута одного кортежа.

<u>Пример 8</u>. В накладной можно установить следующую взаимосвязь атрибутов - "Цена\*Количество=Сумма", связывающую атрибуты "Цена", "Количество", "Сумма".

Данный пример кажется неестественным, т.к. сумма является явно избыточным атрибутом, значение которого просто выводятся из значений других атрибутов. Поэтому кажется, что лучше хранить только два базовых атрибута "Цена" и "Количество", а сумму вычислять во время выполнения запросов по мере необходимости. Так, собственно, требует реляционная теория, стремящаяся свести избыточность к минимуму. В практике, однако, дело обстоит сложнее. Например, каждая строка реальной накладной может содержать следующие данные о товаре:

Наиме-нование атрибута	Описание атрибута	Базовый ли атрибут	Формула для вычислимого атрибута
Name	Наименование товара	Да	
N	Количество	Да	
P1	Учетная цена товара	Да	
S1	Учетная сумма на все количество		S1 = N*P1
PerSent	Процент наценки на единицу товара	Да	
P2	Наценка на единицу товара		P2 = P1*PerSent/100
S2	Сумму наценки на все количество		S2 = N*P2
P3	Цену товара с учетом наценки		P3 = P1+P2
S3	Сумму на все количество с учетом наценки		S3 = N*P3
NDS	Процент НДС	Да	
P4	Сумма НДС на единицу товара		P4 = P2*NDS/100
S4	Сумма НДС на все количество		S4 = N*P4
P5	Цена товара с НДС		P5 = P3 + P4
S5	Сумма на все количество с НДС		S5 = N*P5

# Таблица 3 Атрибуты товара

Базовыми, т.е. требующими ввода данных являются всего 5 атрибутов (выделены серым цветом). Все остальные атрибуты вычисляются по базовым. Нужно ли хранить в отношении только базовые атрибуты, или желательно хранить все атрибуты, пересчитывая значения вычислимых атрибутов каждый раз при изменении базовых?

Решение 1. Пусть в отношении решено хранить только базовые атрибуты.

#### Достоинства решения:

- Структура отношения полностью неизбыточна.
- Не требуется дополнительного программного кода для поддержания целостности кортежа.
- Экономится дисковое пространство.
- Уменьшается трафик сети.

#### Недостатки решения:

• В бухгалтерии для формирования проводок используются, как правило, не базовые, а вычислимые атрибуты. Одни и те же формулы используются во многих

- местах, поэтому все операторы отбора данных будут содержать одинаковые фрагменты кода с одними и теми же формулами. Имеется риск в разных местах вычислять одни и те же данные по *разным формулам*.
- При изменении логики вычислений (что бывает довольно часто при изменении законодательства), необходимо изменить одни и те же фрагменты кода во всех местах, где они встречаются. Это сильно затрудняет модификацию приложений.
- Если возникает нерегламентированный запрос, то человек, формулирующий запрос должен *помнить* все эти формулы.

<u>Решение 2</u>. Предположим, что в отношении решено хранить *все атрибуты*, в том числе и вычислимые.

#### Достоинства решения:

- Код, поддерживающий целостность кортежа (и содержащий формулы для вычислимых атрибутов), хранится в одном месте, например в триггере, связанном с данным отношением.
- При изменении логики вычислений, изменения в формулы требуется внести только в одном месте (в триггере).
- Запросы к базе данных содержат меньше формул и поэтому более просты.
- Легче формулировать нерегламентированные запросы, т.к. в запросе используются атрибуты, имеющие для бухгалтера конкретный смысл.

#### Недостатки решения:

- При изменении логики расчета надобность в некоторых атрибутах может исчезнуть, зато может появиться потребность в новых атрибутах. Это потребует *перестройки структуры* отношения, что является весьма болезненной операцией для работающей системы.
- Структура отношения становится более сложной и запутанной.
- Увеличивается объем базы данных.
- Увеличивается трафик сети.

Как видим, оба решения имеют свои достоинства и недостатки. Важно то, что программный код, содержащий эти формулы, не исчезает ни в каком из этих решений (да и куда он денется, раз такова предметная область!). Только в одном случае код хранится в одном месте, а в другом может быть "размазан" по всему приложению.

На самом деле данный пример сильно упрощен, т.к. еще одной неприятной особенностью наших бухгалтерий является то, что все расчеты должны вестись с определенной точностью, а именно - до копеек. Возникает проблема округления, а это еще более усложняет формулы для расчетов цен. Простой пример - вычисление НДС содержит операцию деления, следовательно может приводить к бесконечным дробям типа 15,519999... Такую дробь необходимо округлить до 15.52. Если продается одна единица товара, то это не страшно, но если продается несколько единиц товара, то сумму НДС на все количество можно считать по разным формулам:

- 1. S4 = N\* ROUND(P2\*NDS/100) СНАЧАЛА округлить при вычислении НДС на единицу товара, а ПОТОМ умножить на все количество, или
- 2. S4 = ROUND(N\*P2\*NDS/100) CHAЧАЛА умножить на все количество, а ПОТОМ округлить до требуемого знака.

<u>Лирическое отступление</u>. Автор, как математик по образованию (к.ф.-м.н.), считает, что верной, безусловно, является первая формула. Действительно, вычисляя по первой формуле, мы получим *одну и ту же сумму* НДС независимо от того, продали мы одну партию товара, содержащую 50 единиц, или продали 50 партий по одной единице в каждой. При вычислениях по второй формуле сумма НДС в партии, состоящий из 50 единиц товара *отличается* от суммы НДС по 50 партиям по одной единице товара в каждой. Разработав несколько складских программ, автор получал разные ответы на этот вопрос в разных бухгалтерия, разные ответы на этот вопрос в одной бухгалтерии в разное время, и разные ответы на этот вопрос в разных инспекциях. В конечном итоге, автор пришел к грустному выводу, что для того, чтобы стать бухгалтером, его способностей и образования недостаточно.

<u>Другие решения</u>. Можно попытаться использовать и другие решения, для облегчения разработки и сопровождения в данном случае. Например, можно хранить в базовом отношении только базовые атрибуты, а для работы бухгалтерии использовать заранее подготовленные представления (динамические отношения, задаваемые оператором SQL). Тогда логика расчетов будет храниться в одном SQL-операторе, определяющем это представление. Другим вариантом может быть сохранение формул в виде хранимых процедур и функций базы данных.

<u>Проверка ограничения</u>. К моменту проверки ограничения кортежа должны быть проверены ограничения целостности атрибутов, входящих в этот кортеж.

Ограничение кортежа является немедленно проверяемым ограничением. Действительно, ограничение кортежа не зависит ни от каких других объектов базы данных, кроме атрибутов, входящих в состав кортежа. Поэтому никакие изменения в других объектах не могут повлиять на истинность ограничения.

#### Ограничения отношения

Определение 11. Ограничения целостности отношения представляют ограничения, накладываемые только на допустимые значения отдельного отношения, и не являющиеся ограничением целостности кортежа. Требование, что ограничение относится к отдельному отношению, означает, что для его проверки не требуется информации о других отношениях (в том числе не требуется ссылок по внешнему ключу на кортежи этого же отношения).

<u>Пример 9</u>. Ограничение целостности сущности (см. гл. 3), задаваемое потенциальным ключом отношения, является ограничением отношения, т.к. для его проверки необходимо иметь информацию обо всех кортежах отношения (более точно, обо всех занятых в данный момент значениях потенциального ключа).

<u>Пример 10</u>. Ограничение целостности, определяемые наличием функциональных, многозначных зависимостей и зависимостей соединения, являются ограничениями отношения.

<u>Пример 11</u>. Предположим, что в отношении PERSON (см. пример 1) задано следующее ограничение - в каждом отделе должно быть не менее двух сотрудников. Это ограничение можно сформулировать так - количество строк с одинаковым значением Dept\_Id должно быть не меньше 2.

<u>Замечание</u>. Для того чтобы ввести в действие (объявить) это ограничение, необходимо, чтобы в отношение уже были вставлены некоторые кортежи.

<u>Пример 12</u>. Ограничение целостности, определяемое требованием, что некоторая таблица должна быть не пуста, являются ограничениями отношения.

<u>Проверка ограничения</u>. К моменту проверки ограничения отношения должны быть проверены ограничения целостности кортежей этого отношения.

Ограничение отношения может быть как немедленно проверяемым ограничением, так и ограничением с отложенной проверкой.

Ограничение отношения, являющееся ограничением потенциального ключа (пример 9) является немедленно проверяемым ограничением.

Ограничение, определенное наличием функциональной зависимости атрибутов также является немедленно проверяемым ограничением.

Ограничения же, определенные многозначной зависимостью или зависимостью соединения являются ограничениями с отложенной проверкой. Действительно, эти ограничения требуют, чтобы кортежи вставлялись и удалялись *целыми группами* (см. гл. 7). Это невозможно сделать, если выполнять проверку после каждой одиночной вставки или удаления кортежа.

Ограничение в примере 11 кажется немедленно проверяемым. Действительно, можно сразу после вставки или удаления кортежа проверить, выполняется ли ограничение, и, если оно не выполняется, то откатить операцию. Но, однако, в этом случае, невозможно вставить ни один новый кортеж для нового отдела. В новый отдел необходимо вставить сразу не менее двух сотрудников. Таким образом, это ограничение с отложенной проверкой.

Ограничение из примера 12 имеет смысл проверять только при удалении кортежей из отношения. Это ограничение может быть как немедленно проверяемым, так и отложенным.

#### Ограничения базы данных

Определение 12. **Ограничения целостности базы данных** представляют ограничения, накладываемые на значения двух или более связанных между собой отношений (в том числе отношение может быть связано само с собой).

<u>Пример 13</u>. Ограничение целостности ссылок (см. гл. 3), задаваемое внешним ключом отношения, является ограничением базы данных.

**Пример 14**. Ограничение на таблицы DEPART и PERSON из примера 1 является отношением базы данных, т.к. оно связывает данные, размещенные в различных таблицах.

<u>Проверка ограничения</u>. К моменту проверки ограничения базы данных должны быть проверены ограничения целостности отношений.

Ограничение базы данных может быть как немедленно проверяемым ограничением, так и ограничением с отложенной проверкой.

Ограничение отношения, являющееся ограничением внешнего ключа может быть как немедленно проверяемым ограничением, так и отложенным ограничением. Действительно, в простейшем случае, если кортеж X отношения A должен ссылаться на кортеж Y отношения B, то проверку ограничения ссылочной целостности можно производить cpasy после любой из операций вставки, обновления или удаления в любом из отношений A или B. В более сложном случае, предположим, что кортеж X отношения A должен ссылаться на кортеж X отношения и последовательности можно провести только посл завершения транзакции, состоящей из последовательности операций:

- 1. вставки кортежа X в отношение Aс нулевой ссылкой на отношение B,
- 2. вставки кортежа Y отношение B со ссылкой на кортеж X отношения A,
- 3. исправления ссылки в кортеже X с NULL на ссылку на кортеж Y.

Ограничение, приведенное в примере 1, может быть только ограничением с отложенной проверкой.

# Реализация декларативных ограничений целостности средствами SQL

# Общие принципы реализации ограничений средствами SQL

Стандарт SQL не предусматривает процедурных ограничений целостности, реализуемых при помощи триггеров и хранимых процедур. В стандарте SQL 92 отсутствует понятие "триггер", хотя триггеры имеются во всех промышленных СУБД SQL-типа. Таким образом, реализация ограничений средствами конкретной СУБД обладает большей гибкостью, нежели с использованием исключительно стандартных средств SQL.

Стандарт SOL позволяет задавать декларативные ограничения следующими способами:

- Как ограничения домена.
- Как ограничения, входящие в определение таблицы.
- Как ограничения, хранящиеся в базе данных в виде независимых утверждений (assertion).

Допускаются как немедленно проверяемые, так и ограничения с отложенной проверкой. Режим проверки отложенных ограничений можно в любой момент изменить так, чтобы ограничение проверялось:

- 1. После исполнения каждого оператора, изменяющего содержимое таблицы, к которой относится данное ограничение.
- 2. При завершении *каждой транзакции*, включающей операторы, изменяющие содержимое таблиц, к которым относятся данное ограничение.
- 3. В любой промежуточный момент, если пользователь инициирует проверку.

При определении ограничения указывается тип проверки ограничения - является ли это ограничение *неоткладываемым* (*NOT DEFERRED*) или может быть *откладываемым* (*DEFERRED*). Во втором случае можно задать процедуру по умолчанию: *проверять* немедленно или проверять по завершении транзакции. Таким образом, можно определить

потенциально откладываемое ограничение, которое по умолчанию проверяется немедленно. В любой момент режим проверки такого ограничения можно изменить на отложенный и наоборот. Режим проверки может быть изменен для одного ограничения или сразу для всех потенциально откладываемых ограничений. Если ограничение определено как неоткладываемое, то тип такого ограничения изменить нельзя и ограничение всегда проверяется немедленно.

Элементы процедурности все же присутствуют в стандарте SQL в виде так называемых *действий, исполняемых по ссылке* (referential triggered actions). Эти действия определяют, что будет происходить при изменении значения родительского ключа, на который ссылается некоторый внешний ключ. Эти действия можно задавать независимо для операций обновления (ON UPDATE) или для операций удаления (ON DELETE) записей в родительском отношении. Стандартом SQL определяется 4 типа действий, исполняемых по ссылке:

- *CASCADE*. Изменения значения родительского ключа автоматически приводят к таким же изменениям связанного с ним значения внешнего ключа. Удаление кортежа в родительском отношении приводит к удалению связанных с ним кортежей в дочернем отношении.
- *SET NULL*. Все внешние ключи, которые ссылаются на обновленный или удаленный родительский ключ получают значения NULL.
- **SET DEFAULT**. Все внешние ключи, которые ссылаются на обновленный или удаленный родительский ключ получают значения, принятые по умолчанию для этих ключей.
- *NO ACTION*. Значения внешнего ключа не изменяются. Если операция приводит к нарушению ссылочной целостности (появляются "висящие" ссылки), то такая операция не выполняется.

Как видно, действия, исполняемые по ссылке, фактически являются встроенными в СУБД триггерами. Действия типа CASCADE, SET NULL и SET DEFAULT являются компенсирующими операциями, вызывающимися при попытке нарушить ссылочную пелостность.

#### Синтаксис ограничений стандарта SQL

Понятие ограничения используется во многих операторах определения данных (DDL).

```
Ограничение check::=

CHECK Предикат

Ограничения таблицы ::=

[CONSTRAINT Имя ограничения]

{
{PRIMARY KEY (Имя столбца.,..)}

| {UNIQUE (Имя столбца.,..)}

| {FOREIGN KEY (Имя столбца.,..) REFERENCES Имя таблицы [(Имя столбца.,..)]

[Ссылочная спецификация]}

| { Ограничение check }

}

[Атрибуты ограничения]
```

```
Ограничения столбца::=

[CONSTRAINT Имя ограничения]

{
NOT NULL}

| {PRIMARY KEY}

| {UNIQUE}

| {REFERENCES Имя таблицы [(Имя столбца)] [Ссылочная спецификация]}

| {Ограничение check }

}

[Атрибуты ограничения]

Ссылочная спецификация::=

[MATCH {FULL | PARTIAL }]

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

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

Атрибуты ограничения::=

{DEFERRABLE [INITIALLY DEFERRED | INITIALLY IMMEDIATE]}

| {NOT DEFERRABLE}
```

*Ограничение типа СНЕСК*. Ограничение типа СНЕСК содержит предикат, могущий принимать значения TRUE, FALSE и UNKNOWN (NULL). Примеры предикатов различного вида приведены в главе 5. Ограничение типа СНЕСК может быть использовано как часть описания домена, таблицы, столбца таблицы или отдельного ограничения целостности - ASSERTION. Ограничение считается нарушенным, если предикат ограничения принимает значение FALSE.

# <u>Пример 15</u>. Пример ограничения типа СНЕСК:

CHECK (Salespeaple.Salary IS NOT NULL) OR (Salespeaple.Commission IS NOT NULL)

Данное ограничение утверждает, что каждый продавец должен иметь либо ненулевую зарплату, либо ненулевые комиссионные.

# **Пример 16**. Еще пример ограничения типа СНЕСК:

CHECK EXIST(SELECT \* FROM Salespeaple)

Данное ограничение утверждает, что список продавцов не может быть пустым.

*Ограничения таблицы* и *ограничения столбца*. Ограничения таблицы и ограничения столбца таблицы входят как часть описания соответственно таблицы или столбца таблицы. Ограничение таблицы может относиться к нескольким столбцам таблицы. Ограничение столбца относится только к одному столбцу таблицы. Любое ограничение столбца можно описать как ограничение таблицы, но не наоборот.

Ограничения таблицы или столбца могут иметь наименования, при помощи которого в дальнейшем можно отменять это ограничение или менять время его проверки.

*Ограничение PRIMARY KEY*. Ограничение PRIMARY KEY для таблицы или столбца означает, что группа из одного или нескольких столбцов образуют потенциальный ключ таблицы. Это означает, что комбинация значений в PRIMARY KEY должна быть

уникальной для каждой строки таблицы. Дублированные значения или значения, содержащие NULL, будут отвергнуты. Для одной таблицы может быть определено единственное ограничение PRIMARY KEY. В терминах стандарта SQL это называется первичным ключом таблицы.

**Ограничение UNIQUE**. Ограничение UNIQUE для таблицы или столбца означает, что группа из одного или нескольких столбцов образуют потенциальный ключ таблицы, в котором допускаются значения NULL. Это означает, что две строки, содержащие одинаковые и не равные NULL-значения, считаются нарушающими уникальность и не допускаются. Две строки, содержащие NULL-значения *считаются различными* и допускаются. Для одной таблицы может быть определено несколько ограничений UNIQUE.

<u>Замечание</u>. С точки зрения реляционной модели данных (см. главу 3, замечание к правилам целостности сущностей и внешних ключей), ограничение типа UNIQUE не определяет потенциальный ключ, т.к. потенциальный ключ не должен содержать NULL-значений.

Ограничения FOREIGN KEY и REFERENCES. Ограничение FOREIGN KEY... REFERENCES... для таблицы и ограничение REFERENCES... для столбца определяют внешний ключ таблицы. Ограничение REFERENCES... для столбца определяет простой внешний ключ, т.е. ключ, состоящий из одной колонки. Ограничение FOREIGN KEY... REFERENCES... для таблицы может определять как простой, так и сложный внешний ключ, т.е. ключ, состоящий из нескольких колонок таблицы. Столбец или группа столбцов таблицы, на которую ссылается внешний ключ, должна иметь ограничения PRIMARY КЕҮ или UNIQUE. Столбцы, на которые ссылается внешний ключ, должны иметь тот же тип данных, что и столбцы, входящие в состав внешнего ключа. Таблица может иметь ссылку на себя. Ограничение внешнего ключа нарушается, если значения, присутствующие во внешнем ключе, не совпадают со значениями соответствующего ключа родительской таблицы ни для одной строки из родительской таблицы. Операции, приводящие к нарушению ограничения внешнего ключа, отвергаются. Как должны совпадать значения внешнего ключа и ключа родительской таблицы, а также, какие действия необходимо выполнить при изменениях ключей в родительской таблице, описаны ниже в ссылочной спецификации.

*Ограничение NOT NULL*. Ограничение NOT NULL столбца не допускает появления в столбце NULL-значений.

**Ссылочная спецификация**. Ссылочная спецификация определяет характеристики внешнего ключа таблицы.

Предложение MATCH {FULL | PARTIAL}. Предложение MATCH FULL требует полного совпадения значений внешнего и первичного ключей. Предложение MATCH PARTIAL допускает частичное совпадение значений внешнего и первичного ключей. Предложение MATCH может быть также пропущенным. Для случая MATCH PARTIAL в дочерней таблице могут появиться строки, имеющие значения внешнего ключа, неуникально совпадающие со значениями родительского ключа. Т.е. одна строка дочерней таблицы может иметь неуникальные ссылки не несколько строк родительской таблицы. Это очень сильно отличается от реляционной модели данных, и это отличие провоцируется допущением NULL-значений. Чтобы рассмотреть различные варианты совпадений внешнего и родительского ключей, рассмотрим следующий пример.

# **Пример 17**. Пусть имеется две таблицы:

X	Y	
1	Aa	
1	Bb	
2	Cc	
2	Dd	
3	Ee	
3	Ff	

# Таблица 4 таблица А (Родительская)

Z	X	Y	
1	1	Aa	
2	1	Null	
3	Null	Cc	
4	Null	Null	
5	4	Gg	

Таблица 5 Таблица В (Дочерняя)

Таблица A имеет первичный ключ (X, Y). Таблица B имеет первичный ключ Z, и внешний ключ (X, Y), ссылающийся на первичный ключ таблицы A. Различные варианты совпадения строк дочерней таблицы B со строками родительской таблицы A приведены ниже:

	Колонки X и Y таблицы В допускают null-значения	Колонки X и Y таблицы В не допускают null-значений
матсн	1 строка - допустима, совпадает с 1 строкой таблицы А. 2 строка - допустима, не совпадает ни с чем. 3 строка - допустима, не совпадает ни с чем. 4 строка - допустима, не совпадает ни с чем. 5 строка - не допустима.	1 строка - допустима, совпадает с 1 строкой таблицы А. 2 строка - не допустима. 3 строка - не допустима. 4 строка - не допустима. 5 строка - не допустима.
MATCH FULL	1 строка - допустима, совпадает с 1 строкой таблицы А. 2 строка - не допустима. 3 строка - не допустима. 4 строка - допустима, не совпадает ни с чем. 5 строка - не допустима.	1 строка - допустима, совпадает с 1 строкой таблицы А. 2 строка - не допустима. 3 строка - не допустима. 4 строка - не допустима. 5 строка - не допустима.

MATCH	1 строка - допустима, совпадает с 1	1 строка - допустима, совпадает
PARTIAL	строкой таблицы А.	с 1 строкой таблицы А.
	2 строка - допустима, неуникально	2 строка - не допустима.
	совпадает с 1 и 2 строками таблицы А.	3 строка - не допустима.
	3 строка - допустима, уникально	4 строка - не допустима.
	совпадает с 3 строкой таблицы А.	5 строка - не допустима.
	4 строка - допустима, не совпадает ни с	
	чем.	
	5 строка - не допустима.	

Предложение MATCH игнорируется, если все столбцы внешнего ключа имеют ограничения NOT NULL.

**Предложения ON UPDATE и ON DELETE**. Предложения ON UPDATE и ON DELETE определяют действия, исполняемые по ссылке. Действия, исполняемые по ссылке, в основном описаны выше в этой главе. Сложности в понимании того, как выполняются эти действия, возникают если установлено MATCH PARTIAL и колонки, входящие в состав внешнего ключа, допускают NULL-значения. Подробно эти действия с учетом возможных сложностей описаны в [9].

**Атрибуты ограничения**. Атрибуты ограничения определяют, в какой момент проверяются ограничения. Ограничение может быть определено как NOT DEFERRABLE (неоткладываемое) или DEFERRABLE (откладываемое). Если атрибуты ограничения не указаны, то по умолчанию принимается NOT DEFERRABLE.

Если ограничение определено как NOT DEFERRABLE (*неоткладываемое*), то ограничение всегда проверяется *сразу после выполнения каждого оператора* INSERT, UPDATE или DELETE, которые могут привести к нарушению ограничения.

Если ограничение определено как DEFERRABLE (*откладываемое*), то ограничение может иметь два режима проверки - немедленно после выполнения операции или в конце транзакции. Режим проверки может быть изменен в любой момент внутри транзакции командой SET CONSTRAINTS. При определении ограничения можно указать начальный режим проверки INITIALLY DEFERRED (начально отложенное) или INITIALLY IMMEDIATE (начально немедленно проверяемое).

#### Синтаксис операторов SQL, использующих ограничения

Стандарт SQL описывает следующие операторы, в которых может быть использованы ограничения:

- CREATE DOMAIN создать домен
- ALTER DOMAIN изменить домен
- DROP DOMAIN удалить домен
- CREATE TABLE создать таблицу
- ALTER TABLE изменить таблицу
- DROP TABLE удалить таблицу
- CREATE ASSERTION создать утверждение
- DROP ASSERTION удалить утверждение
- COMMIT WORK зафиксировать транзакцию
- SET CONSTRAINTS установить момент проверки ограничений

## CREATE DOMAIN Имя домена [AS] Тип данных

#### [DEFAULT Значение по умолчанию]

[Имя ограничения] Ограничение check [Атрибуты ограничения]

Этот оператор задает домен, на основе которого можно определять колонки таблиц. Т.к. имя колонки, которая будет основана на этом домене заранее неизвестно, то в ограничении CHECK домена для ссылки на значение этого домена используется ключевое слово VALUE. В конкретной таблице СУБД заменит слово VALUE на имя колонки таблицы.

<u>Пример 18</u>. Приведенный ниже оператор создает домен Salary на основе целочисленного типа данных, причем значения из этого домена не могут принимать неположительные значения (но могут принимать значение NULL!). По умолчанию это ограничение проверяется немедленно, но может быть и отложенным:

CREATE DOMAIN Salary AS integer CHECK (VALUE > 0) DEFERRABLE INITIALLY IMMEDIATE

```
ALTER DOMAIN Имя домена {SET DEFAULT Значение по умолчанию} | {DROP DEFAULT} | {ADD [Имя ограничения] Ограничение check [Атрибуты ограничения]} | {DROP CONSTRAINT Имя ограничения}
```

Этот оператор изменяет имеющийся домен. Стандарт запрещает вносить несколько изменений одной командой ALTER DOMAIN. Т.е. если требуется удалить ограничение CHECK и добавить значение по умолчанию, то придется выполнить два оператора ALTER DOMAIN.

#### **DROP DOMAIN** Имя домена **CASCADE** | **RESTRICT**

Этот оператор уничтожает имеющийся домен. Если указана опция RESTRICT, то домен не уничтожается, если имеются ссылки на него из столбцов таблиц. Если указана опция CASCADE, то происходят следующие действия:

- Тип данных домена передается столбцам, основанным на этом домене.
- Если столбец не имеет значения по умолчанию, а для домена значение по умолчанию определено, то оно становится значением по умолчанию для столбца.
- Все ограничения домена становятся ограничениями столбца.

# **CREATE TABLE** Имя таблицы ({Определение столбца | [Ограничение таблицы]}.,..) Определение столбца::= Имя столбца {Имя домена | Тип данных [Размер]} [Ограничение столбца...]

[DEFAULT Значение по умолчанию]

Этот оператор (синтаксис приведен не полностью - пропущены опции создания временных таблиц) создает таблицу базы данных. В таблице обязано быть не менее

одного определения столбца. В таблице может быть определено несколько ограничений (в том числе и ни одного).

Каждый столбец должен иметь имя и быть определен на некотором типе данных или на некотором домене. Ограничения домена становятся ограничениями столбца. Кроме того, столбец может иметь дополнительные ограничения. Если домен имеет значение по умолчанию и в определении столбца определено значение по умолчанию, то значение для столбца перекрывает значение для домена.

#### Пример 19.

```
CREATE TABLE Salespeaple
(Salespeaple_Id Id_Nums PRIMARY KEY,
Fam CHAR(20) NOT NULL,
Im CHAR(15),
BirthDate DATE,
Salary Salary_Domain DEFAULT 1000,
City_Id INTEGER REFERENCES City ON UPDATE CASCADE ON DELETE RESTRICT,
District_Id INTEGER,
CONSTRAINT AltKey UNIQUE(Fam, Im, BirthDate),
CHECK (City_Id IS NOT NULL OR District_Id IS NOT NULL),
FOREIN KEY District_Id REFERENCES District ON UPDATE CASCADE ON DELETE
RESTRICT)
```

Этот оператор создает таблицу Salespeaple с колонками (Salespeaple\_Id, Fam, Im, BirthDate, Salary, City\_Id, District\_Id) и следующими ограничениями:

- Колонка Salespeaple\_Id наследует все ограничения домена Id\_Nums. Кроме того, эта колонка образует первичный ключ таблицы (следовательно, не допускает NULL-значений).
- Колонка Fam не допускает NULL-значений.
- Колонка Salary наследует все ограничения домена Salary\_Domain. Кроме того, эта колонка имеет значения по умолчанию 1000.
- Колонка City\_Id является внешним ключом, ссылающимся на первичный ключ таблицы City. При изменении первичного ключа в таблице City соответствующие значения внешнего ключа в таблице Salespeaple будут каскадно изменены. При удалении строки из таблицы City будет выполняться проверка, нет ли ссылок на удаляемую строку из таблицы Salespeaple. Если такие ссылки найдутся, то операция удаления в таблице City будет отвергнута.
- Колонка District\_Id также является внешним ключом, ссылающимся на первичный ключ таблицы District. Этот внешний ключ, в отличие от предыдущего, определен как ограничение таблицы. Действия, определенные по ссылке аналогичны предыдущим.
- Колонки (Fam, Im, BirthDate) образуют альтернативный ключ таблицы. Это ограничение имеет наименование AltKey.
- Колонки City\_Id и District\_Id не могут одновременно принимать NULL-значения (хотя каждая из них по отдельности допускает использование NULL-значений).

```
ALTER TABLE Имя таблицы
{ADD [COLUMN] Определение столбца}
| {ALTER [COLUMN] Имя столбца {SET DEFAULT Значение по умолчанию | DROP DEFAULT}}
| {DROP [COLUMN] Имя столбца RESTRICT | CASCADE}
```

# | {ADD Ограничение таблицы} | {DROP CONSTRAINT Имя ограничения RESTRICT | CASCADE}

Этот оператор позволяет изменять имеющуюся таблицу. В таблице можно удалять или добавлять колонки и/или ограничения. Кроме того, для колонок можно менять значение по умолчанию.

# DROP TABLE Имя таблицы CASCADE | RESTRICT

Этот оператор позволяет удалять имеющуюся таблицу. Вместе с таблицей удаляются и ограничения, определенные для этой таблицы.

Если указан параметр RESTRICT, то таблица удаляется только если нет никаких ссылок на эту таблицу в других ограничениях или представлениях.

Если указан параметр CASCADE, то удаляются также и все объекты, ссылающиеся на эту таблицу.

#### CREATE ASSERTION Имя утверждения

Ограничение check [Атрибуты ограничения]

Этот оператор позволяет создавать утверждения - т.е. ограничения, не являющиеся частью определения домена, колонки или таблицы.

Предикат СНЕСК, входящий в определение утверждения, может быть достаточно сложным и содержать ссылки на несколько таблиц.

#### Пример 20.

CREATE ASSERTION Check\_Pay CHECK (Salespeaple.Salary IS NOT NULL) OR (Salespeaple.Commission IS NOT NULL) DEFERRABLE INITIALLY IMMEDIATE

#### DROP ASSERTION Имя утверждения

Этот оператор позволяет удалять имеющееся утверждение.

#### **COMMIT WORK**

Этот оператор фиксирует транзакцию. При этом проверяются все отложенные до конца транзакции ограничения. Если одно из ограничений не выполняется, то транзакция откатывается.

# **SET CONSTRAINT** {*Имя ограничения.,..* | **ALL**} {**DEFERRED** | **IMMEDIATE**}

Этот оператор позволяет во время выполнения транзакции менять момент проверки всех (ALL) или некоторых ограничений. Этот оператор действует только на ограничения, определенные как DEFERRABLE (потенциально откладываемые). Если ограничение А находилось в состоянии IMMEDIATE (немедленно проверяемое), то оператор SET CONSTRAINT A DEFERRED переводит его в состояние DEFERRED (с отложенной проверкой) и тогда все операции, потенциально могущие нарушить это ограничение,

будут выполняться без проверки. Проверка будет произведена в конце транзакции или в момент полачи команды SET CONSTRAINT A IMMEDIATE.

#### Выволы

*Транзакция* - это неделимая, с точки зрения воздействия на СУБД, последовательность операций манипулирования данными, выполняющаяся по принципу "*все или ничего*", и переводящая базу данных из одного *целостного состояния* в другое целостное состояние.

Транзакция обладает четырьмя важными свойствами, известными как свойства АСИД:

- *(А) Атомарность*.
- (С) Согласованность.
- (И) Изоляция.
- *(Д) Долговечность*.

База данных находится в *согласованном состоянии*, если для этого состояния выполнены все *ограничения целостности*.

*Ограничение целостности* - это некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния базы данных.

Ограничения целостности классифицируются несколькими способами:

- По способам реализации.
- По времени проверки.
- По области действия.

По способам реализации различают:

- *Декларативную* поддержку ограничений целостности средствами языка определения данных (DDL).
- *Процедурную* поддержку ограничений целостности посредством триггеров и хранимых процедур.

По времени проверки ограничения делятся на:

- Немедленно проверяемые ограничения.
- Ограничения с отложенной проверкой.

По области действия ограничения делятся на:

- Ограничения домена.
- Ограничения атрибута.
- Ограничения кортежа.
- Ограничения отношения.
- Ограничения базы данных.

Стандарт языка SQL поддерживает только декларативные ограничения целостности, реализуемые как:

- Ограничения домена.
- Ограничения, входящие в определение таблицы.
- Ограничения, хранящиеся в базе данных в виде независимых утверждений (assertion).

Проверка ограничений допускается как после выполнения каждого оператора, могущего нарушить ограничение, так и в конце транзакции. Во время выполнения транзакции можно изменить режим проверки ограничения.

# Глава 10. Транзакции и параллелизм

В данной главе изучаются возможности параллельного выполнения транзакций несколькими пользователями, т.е. свойство (И) - изолированность транзакций.

Современные СУБД являются многопользовательскими системами, т.е. допускают параллельную одновременную работу большого количества пользователей. При этом пользователи не должны мешать друг другу. Т.к. логической единицей работы для пользователя является транзакция, то работа СУБД должна быть организована так, чтобы у пользователя складывалось впечатление, что их транзакции выполняются независимо от транзакций других пользователей.

Простейший и очевидный способ обеспечить такую иллюзию у пользователя состоит в том, чтобы все поступающие транзакции выстраивать в единую очередь и выполнять строго по очереди. Такой способ не годится по очевидным причинам - теряется преимущество параллельной работы. Таким образом, транзакции необходимо выполнять одновременно, но так, чтобы результат был бы такой же, как если бы транзакции выполнялись по очереди. Трудность состоит в том, что если не предпринимать никаких специальных мер, то данные измененные одним пользователем могут быть изменены транзакцией другого пользователя раньше, чем закончится транзакция первого пользователя. В результате, в конце транзакции первый пользователь увидит не результаты своей работы, а неизвестно что.

Одним из способов (не единственным) обеспечить независимую параллельную работу нескольких транзакций является метод блокировок.

#### Работа транзакций в смеси

Транзакция рассматривается как последовательность элементарных атомарных операций. Атомарность отдельной элементарной операции состоит в том, что СУБД гарантирует, что, с точки зрения пользователя, будут выполнены два условия:

- 1. Эта операция будет выполнена целиком или не выполнена вовсе (атомарность все или ничего).
- 2. Во время выполнения этой операции не выполняются никакие другие операции других транзакций (строгая очередность элементарных операций).

Например, элементарными операциями транзакции будут считывание страницы данных с диска или запись страницы данных на диск (страница данных - это минимальная единица для дисковых операций СУБД). Условие 2 на самом деле является именно логическим условием, т.к. реально система может выполнять несколько различных элементарных операций в один и тот же момент. Например, данные могут храниться на нескольких

физически различных дисках и операции чтения-записи на эти диски могут выполняться одновременно.

Элементарные операции различных транзакций могут выполняться в произвольной очередности (конечно, внутри каждой транзакции последовательность элементарных операций этой транзакции является строго определенной). Например, если есть несколько транзакций, состоящих из последовательности операций элементарных:

$$T = \{T_1, T_2, T_3, \dots, T_n\},\$$

$$Q = \{Q_1, Q_2, Q_3, \dots, Q_m\},\$$

$$S = \{S_1, S_2, S_3, \dots, S_l\}$$

то реальная последовательность, в которой СУБД выполняет эти транзакции может быть, например, такой:

$$\{T_1, Q_1, T_2, S_1, T_3, S_2, S_3, Q_2, \dots\}$$

*Определение 1*. Набор из нескольких транзакций, элементарные операции которых чередуются друг с другом, называется *смесью транзакций*.

*Определение* 2. Последовательность, в которой выполняются элементарные операции заданного набора транзакций, называется *графиком запуска* набора транзакций.

<u>Замечание</u>. Очевидно, что для заданного набора транзакций может быть несколько (вообще говоря, достаточно много) различных графиков запуска.

Обеспечение изолированности пользователей, таким образом, сводится к выбору подходящего (в каком-то смысле правильного) графика запуска транзакций. Одновременно с этим график запуска должен быть оптимальным в некотором смысле, например, давать минимальное среднее время выполнения транзакций каждым пользователем.

Далее мы уточним понятие "правильного" графика и сделаем некоторые замечания об оптимальности.

#### Проблемы параллельной работы транзакций

Каким образом транзакции различных пользователей могут мешать друг другу? Различают три основные проблемы параллелизма:

- Проблема потери результатов обновления.
- Проблема незафиксированной зависимости (чтение "грязных" данных, неаккуратное считывание).
- Проблема несовместимого анализа.

Рассмотрим подробно эти проблемы.

Рассмотрим две транзакции, А и В, запускающиеся в соответствии с некоторыми графиками. Пусть транзакции работают с некоторыми объектами базы данных, например со строками таблицы. Операцию чтение строки  $^P$  будем обозначать  $^{P=P_0}$ , где  $^{P_0}$  прочитанное значение. Операцию записи значения  $^{P_1}$ в строку  $^P$  будем обозначать  $^{P_1} \rightarrow P$ 

# Проблема потери результатов обновления

Две транзакции по очереди записывают некоторые данные в одну и ту же строку и фиксируют изменения.

Транзакция А	Время	Транзакция В
$_{ m Чтение} \ P = P_0$	$t_1$	
	$t_2$	$_{ m Trehue} P = P_0$
$3$ апись $P_1 \rightarrow P$	t <sub>3</sub>	
	t <sub>4</sub>	$3$ апись $P_2 \rightarrow P$
Фиксация транзакции	$t_5$	
	<i>t</i> <sub>6</sub>	Фиксация транзакции
Потеря результата обновления		

<u>Результат</u>. После окончания обеих транзакций, строка  $^{p}$  содержит значение  $^{P_{2}}$ , занесенное более поздней транзакцией В. Транзакция А ничего не знает о существовании транзакции В, и естественно ожидает, что в строке  $^{p}$  содержится значение  $^{P_{1}}$ . Таким образом, транзакция А потеряла результаты своей работы.

# Проблема незафиксированной зависимости (чтение "грязных" данных, неаккуратное считывание)

Транзакция В изменяет данные в строке. После этого транзакция А читает измененные данные и работает с ними. Транзакция В откатывается и восстанавливает старые данные.

Транзакция А	Время	Транзакция В
	$t_1$	$_{ m Uтениe} P = P_0$
	$t_2$	Запись $P_1 \rightarrow P$
$_{ m Uтениe} P = P_1$	t <sub>3</sub>	
Работа с прочитанными данными $P_1$	t <sub>4</sub>	
	t <sub>5</sub>	Откат транзакции $P_0 \rightarrow P$
Фиксация транзакции	$t_6$	
Работа с ''грязными'' данными		

С чем же работала транзакция А?

<u>Результат</u>. Транзакция А в своей работе использовала данные, которых нет в базе данных. Более того, транзакция А использовала данные, которых нет, и *не было* в базе данных! Действительно, после отката транзакции В, должна восстановиться ситуация, как если бы транзакция В вообще *никогда не выполнялась*. Таким образом, результаты работы транзакции А некорректны, т.к. она работала с данными, отсутствовавшими в базе данных.

#### Проблема несовместимого анализа

Проблема несовместимого анализа включает несколько различных вариантов:

- Неповторяемое считывание.
- Фиктивные элементы (фантомы).
- Собственно несовместимый анализ.

#### Неповторяемое считывание

Транзакция А дважды читает одну и ту же строку. Между этими чтениями вклинивается транзакция В, которая изменяет значения в строке.

Транзакция А	Время	Транзакция В
$_{ m Trehue} P = P_0$	$t_1$	
	$t_2$	$_{ m Чтение} P = P_0$
	t <sub>3</sub>	$3$ апись $P_1 \rightarrow P$
	t <sub>4</sub>	Фиксация транзакции
Повторное чтение $P = P_1$	t <sub>5</sub>	
Фиксация транзакции	<i>t</i> <sub>6</sub>	
Неповторяемое считывание		

Транзакция А ничего не знает о существовании транзакции В, и, т.к. сама она не меняет значение в строке, то ожидает, что после повторного чтения значение будет тем же самым.

<u>Результат</u>. Транзакция А работает с данными, которые, с точки зрения транзакции А, самопроизвольно изменяются.

#### Фиктивные элементы (фантомы)

Эффект фиктивных элементов несколько отличается от предыдущих транзакций тем, что здесь за один шаг выполняется достаточно много операций - чтение одновременно нескольких строк, удовлетворяющих некоторому условию.

Транзакция А дважды выполняет выборку строк с одним и тем же условием. Между выборками вклинивается транзакция В, которая добавляет новую строку, удовлетворяющую условию отбора.

Транзакция А	Время	Транзакция В
Выборка строк, удовлетворяющих условию а. (Отобрано п строк)	$t_1$	
	$t_2$	Вставка новой строки, удовлетворяющей условию $\alpha$ .
	t <sub>3</sub>	Фиксация транзакции
Выборка строк, удовлетворяющих условию а. (Отобрано n+1 строк)	£4	
Фиксация транзакции	$t_5$	
Появились строки, которых раньше не было		

Транзакция А ничего не знает о существовании транзакции В, и, т.к. сама она не меняет ничего в базе данных, то ожидает, что после повторного отбора будут отобраны те же самые строки.

<u>Результат</u>. Транзакция А в двух одинаковых выборках строк получила разные результаты.

#### Собственно несовместимый анализ

Эффект собственно несовместимого анализа также отличается от предыдущих примеров тем, что в смеси присутствуют две транзакции - одна длинная, другая короткая.

Длинная транзакция выполняет некоторый анализ по всей таблице, например, подсчитывает общую сумму денег на счетах клиентов банка для главного бухгалтера. Пусть на всех счетах находятся одинаковые суммы, например, по \$100. Короткая транзакция в этот момент выполняет перевод \$50 с одного счета на другой так, что общая сумма по всем счетам не меняется.

Транзакция А	Время	Транзакция В
Чтение счета $P_1 = 100_{\rm H}$ суммирование. $SUM = 100$	t <sub>1</sub>	
	$t_2$	Снятие денег со счета $P_3$ . $P_3:100 \rightarrow 50$
	t <sub>3</sub>	Помещение денег на счет $P_1$ . $P_1:100 { ightarrow} 150$
	t <sub>4</sub>	Фиксация транзакции

Чтение счета $P_2 = 100$ и суммирование. $SUM = 200$	t <sub>5</sub>	
Чтение счета $P_3 = 50$ и суммирование. $SUM = 250$	t <sub>6</sub>	
Фиксация транзакции	t <sub>7</sub>	
Сумма \$250 по всем счетам неправильная - должно быть \$300		

<u>Результат</u>. Хотя транзакция В все сделала правильно - деньги переведены без потери, но в результате транзакция А подсчитала неверную общую сумму.

Т.к. транзакции по переводу денег идут обычно непрерывно, то в данной ситуации следует ожидать, что главный бухгалтер *никогда* не узнает, сколько же денег в банке.

#### Конфликты между транзакциями

Итак, анализ проблем параллелизма показывает, что если не предпринимать специальных мер, то при работе в смеси нарушается свойство (И) транзакций - изолированность. Транзакции реально мешают друг другу получать правильные результаты.

Однако не всякие транзакции мешают друг другу. Очевидно, что транзакции не мешают друг другу, если они обращаются к *разным данным* или выполняются в *разное время*.

*Определение 3.* Транзакции называются *конкурирующими*, если они пересекаются по времени и обращаются к одним и тем же данным.

В результате конкуренции за данными между транзакциями возникают *конфликты доступа* к данным. Различают следующие виды конфликтов:

- *W-W (Запись Запись)*. Первая транзакция изменила объект и не закончилась. Вторая транзакция пытается изменить этот объект. Результат потеря обновления.
- **R-W** (**Чтение Запись**). Первая транзакция прочитала объект и не закончилась. Вторая транзакция пытается изменить этот объект. Результат несовместимый анализ (неповторяемое считывание).
- **W-R** (Запись Чтение). Первая транзакция изменила объект и не закончилась. Вторая транзакция пытается прочитать этот объект. Результат чтение "грязных" ланных.

Конфликты типа R-R (Чтение - Чтение) отсутствуют, т.к. данные при чтении не изменяются.

Другие проблемы параллелизма (фантомы и собственно несовместимый анализ) являются более сложными, т.к. принципиальное отличие их в том, что они не могут возникать при работе с одним объектом. Для возникновения этих проблем требуется, чтобы транзакции работали с целыми *наборами* данных.

*Определение 4*. График запуска набора транзакций называется *последовательным*, если транзакции выполняются строго по очереди, т.е. элементарные операции транзакций не чередуются друг с другом.

*Определение 5*. Если график запуска набора транзакций содержит чередующиеся элементарные операции транзакций, то такой график называется **чередующимся**.

При выполнении последовательного графика гарантируется, что транзакции выполняются правильно, т.е. при последовательном графике транзакции не "чувствуют" присутствия друг друга.

Определение 6. Два графика называются эквивалентными, если при их выполнении будет получен один и тот же результат, независимо от начального состояния базы данных.

*Определение 7.* График запуска транзакции называется *верным* (*сериализуемым*), если он эквивалентен какому-либо последовательному графику.

Замечание. При выполнении двух различных последовательных (а, следовательно, верных) графиков, содержащих один и тот же набор транзакций, могут быть получены различные результаты. Действительно, пусть транзакция А заключается в действии "Сложить X с 1", а транзакция В - "Удвоить X". Тогда последовательный график {A, B} даст результат 2(X+1), а последовательный график {B, A} даст результат 2X+1. Таким образом, может существовать несколько верных графиков запусков транзакций, приводящих к разным результатам при одном и том же начальном состоянии базы данных.

Задача обеспечения изолированной работы пользователей не сводится просто к нахождению правильных (сериальных) графиков запусков транзакций. Если бы этого было достаточно, то лучшим был бы простейший способ сериализации - ставить транзакции в общую очередь по мере их поступления и выполнять строго последовательно. Таким способом автоматически будет получен правильный (сериальный) график. Проблема в том, что этот график будет неоптимальным с точки зрения общей производительности системы. Получается ситуация, в которой борются противоположные силы - с одной стороны, стремление обеспечить сериальность за счет ухудшения общей эффективности работы, с другой стороны, стремление улучшить общую эффективность за счет ухудшения сериальности.

Один крайний случай (выполнение транзакций по очереди) мы рассмотрели. Рассмотрим другой крайний случай - попытаемся достичь оптимального графика - т.е. графика с максимальной эффективностью выполнения транзакций. Для этого сначала нужно уточнить понятие "оптимальность". С каждым возможным графиком запуска транзакций мы можем связать значение некоей стоимостной функции. В качестве стоимостной функции можно взять, например, суммарное время выполнения всех транзакций в наборе. Время выполнения одной транзакции считается от момента, когда транзакция возникла и до момента, когда транзакция выполнила свою последнюю элементарную операцию. Это время складывается из следующих компонентов:

- 1. Время ожидания начала транзакции то время, которое проходит от момента, когда транзакция возникла до момента, когда началась реально выполняться ее первая элементарная операция.
- 2. Сумма времен выполнения элементарных операций транзакции.
- 3. Сумма времен всех элементарных операций других транзакций, вклинившихся между элементарными операциями транзакции.

Оптимальным будет график, дающий минимум стоимостной функции. Очевидно, оптимальность графика запуска зависит от выбора стоимостной функции, т.е. график,

оптимальный с точки зрения одних критериев (например, с точки зрения приведенной функции стоимости) не будет оптимальным с точки зрения других критериев (например, с точки зрения достижения максимально быстрого начала выполнения каждой транзакции).

Рассмотрим следующую гипотетическую ситуацию. Предположим, что нам *заранее* на некоторый промежуток времени наперед известно, какие транзакции в какие моменты поступят, т.е. *заранее известна вся будущая смесь транзакций и моменты поступят*, каждой транзакции:

Транзакция 
$$T^1 = \{T_1^1, T_2^1, T_3^1, \dots, T_{n_i}^1\}$$
 поступит в момент  $t_1$ .

Транзакция 
$$T^2 = \{T_1^2, T_2^2, T_3^2, ..., T_{n_2}^2\}$$
 поступит в момент  $t_2$ .

. . .

Транзакция 
$$T^m = \{T_1^m, T_2^m, T_3^m, \dots, T_{n_m}^m\}$$
 поступит в момент  $t_m$ .

В этом случае, т.к. набор всех транзакций заранее известен, теоретически можно перебрать все возможные варианты графиков запусков (их конечное число, хотя и очень большое), и выбрать из них те графики, которые, во-первых, правильные, а во-вторых, оптимальны по выбранному критерию. В этом случае оптимальный график запуска транзакций достижим.

В реальной ситуации, однако, неизвестно не только какие транзакции будут поступать в какие моменты времени, но и неизвестна длительность периода времени, охватывающего набор транзакций. Реально, система может непрерывно работать несколько дней или месяцев и в этом случае набором транзакций будет набор всех транзакций за этот период. С другой стороны, прекращение работы сервера может произойти в любой момент либо по команде администратора системы, либо в результате сбоя. Необходимо, следовательно, чтобы система работала так, чтобы к любому моменту времени набор выполненных и выполняющихся в этот момент транзакций был бы правильным и не слишком далек от оптимального.

Т.к. транзакции не мешают друг другу, если они обращаются к разным данным или выполняются в разное время, то имеется два способа разрешить конкуренцию между поступающими в произвольные моменты транзакциями:

- 1. "Притормаживать" некоторые из поступающих транзакций настолько, насколько это необходимо для обеспечения правильности смеси транзакций в каждый момент времени (т.е. обеспечить, чтобы конкурирующие транзакции выполнялись в разное время).
- 2. Предоставить конкурирующим транзакциям "разные" экземпляры данных (т.е. обеспечить, чтобы конкурирующие транзакции работали *с разными версиями данными*).

Первый метод - "притормаживание" транзакций - реализуется путем использованием блокировок различных видов или метода временных меток.

Второй метод - предоставление разных версий данных - реализуется путем использованием данных из журнала транзакций.

#### Блокировки

Основная идея блокировок заключается в том, что если для выполнения некоторой транзакции необходимо, чтобы некоторый объект не изменялся без ведома этой транзакции, то этот объект должен быть заблокирован, т.е. доступ к этому объекту со стороны других транзакций ограничивается на время выполнения транзакции, вызвавшей блокировку.

Различают два типа блокировок:

- *Монопольные блокировки (X-блокировки, X-locks* eXclusive locks) блокировки без взаимного доступа (блокировка записи).
- *Разделяемые блокировки* (*S-блокировки*, *S-locks* Shared locks) блокировки с взаимным доступом (блокировка чтения).

Если транзакция А блокирует объект при помощи Х-блокировки, то всякий доступ к этому объекту со стороны других транзакций отвергается.

Если транзакция A блокирует объект при помощи S-блокировки, то

- запросы со стороны других транзакций на X-блокировку этого объекта будут отвергнуты,
- запросы со стороны других транзакций на S-блокировку этого объекта будут приняты.

Правила взаимного доступа к заблокированным объектам можно представить в виде следующей *матрицы совместимости блокировок*. Если транзакция А наложила блокировку на некоторый объект, а транзакция В после этого пытается наложить блокировку на этот же объект, то успешность блокирования транзакцией В объекта описывается таблицей:

	Транзакция В пытается наложить блокировку:	
Транзакция А наложила блокировку:	S-блокировку	Х-блокировку
S-блокировку	Да	НЕТ (Конфликт R-W)
Х-блокировку	НЕТ (Конфликт W-R)	НЕТ (Конфликт W-W)

Таблица 1 Матрица совместимости S- и X-блокировок

Три случая, когда транзакция В не может блокировать объект, соответствуют трем видам конфликтов между транзакциями.

Доступ к объектам базы данных на чтение и запись должен осуществляться в соответствии со следующим *протоколом доступа к данным*:

1. Прежде чем прочитать объект, транзакция должна наложить на этот объект S-блокировку.

- 2. Прежде чем обновить объект, транзакция должна наложить на этот объект X- блокировку. Если транзакция уже заблокировала объект S-блокировкой (для чтения), то перед обновлением объекта S-блокировка должна быть заменена X- блокировкой.
- 3. Если блокировка объекта транзакцией В отвергается оттого, что объект уже заблокирован транзакцией А, то транзакция В переходит в *состояние ожидания*. Транзакция В будет находиться в состоянии ожидания до тех пор, пока транзакция А не снимет блокировку объекта.
- 4. Х-блокировки, наложенные транзакцией А, сохраняются до конца транзакции А.

# Решение проблем параллелизма при помощи блокировок

Рассмотрим, как будут себя вести транзакции, вступающие в конфликт при доступе к данным, если они подчиняются протоколу доступа к данным.

# Проблема потери результатов обновления

Две транзакции по очереди записывают некоторые данные в одну и ту же строку и фиксируют изменения.

Транзакция А	Время	Транзакция В
S-блокировка Р - успешна	$t_1$	
$_{ m Чтение} \ P = P_0$	$t_2$	
	t <sub>3</sub>	S-блокировка $P$ - успешна
	t <sub>4</sub>	$_{ m Trehue}$ $P$ = $P_0$
X-блокировка $P$ - отвергается	$t_5$	
Ожидание	<i>t</i> <sub>6</sub>	X-блокировка $P$ - отвергается
Ожидание	t <sub>7</sub>	Ожидание
Ожидание		Ожидание

Обе транзакции успешно накладывают S-блокировки и читают объект  $^{P}$ . Транзакция A пытается наложить X-блокирокировку для обновления объекта  $^{P}$ . Блокировка отвергается, т.к. объект  $^{P}$  уже S-заблокирован транзакцией B. Транзакция A переходит в состояние ожидания до тех пор, пока транзакция B не освободит объект. Транзакция B, в свою очередь, пытается наложить X-блокировку для обновления объекта  $^{P}$ . Блокировка отвергается, т.к. объект  $^{P}$  уже S-заблокирован транзакцией A. Транзакция B переходит в состояние ожидания до тех пор, пока транзакция A не освободит объект.

<u>Результат</u>. Обе транзакции ожидают друг друга и не могут продолжаться. Возникла ситуация *тупика*.

Проблема незафиксированной зависимости (чтение "грязных" данных, неаккуратное считывание)

Транзакция В изменяет данные в строке. После этого транзакция А читает измененные данные и работает с ними. Транзакция В откатывается и восстанавливает старые данные.

Транзакция А	Время	Транзакция В
	$t_1$	S-блокировка P - успешна
	$t_2$	$_{ m Uтениe} P = P_0$
	t <sub>3</sub>	Х-блокировка Р - успешна
	t <sub>4</sub>	$3$ апись $P_1 \rightarrow P$
S-блокировка Р - отвергается	$t_5$	
Ожидание	t <sub>6</sub>	Откат транзакции $P_0 \rightarrow P$
		(Блокировка снимается)
S-блокировка Р - успешна	t <sub>7</sub>	
$_{ m Чтение} \ P = P_0$	<i>t</i> <sub>8</sub>	
Работа с прочитанными данными $P_0$	t <sub>9</sub>	
	t <sub>10</sub>	
Фиксация транзакции	t <sub>11</sub>	
Все правильно		

<u>Результат</u>. Транзакция А притормозилась до окончания (отката) транзакции В. После этого транзакция А продолжила работу в обычном режиме и работала с правильными данными. Конфликт разрешен за счет некоторого увеличения времени работы транзакции А (потрачено время на ожидание снятия блокировки транзакцией В).

# Проблема несовместимого анализа

# Неповторяемое считывание

Транзакция А дважды читает одну и ту же строку. Между этими чтениями вклинивается транзакция В, которая изменяет значения в строке.

Транзакция А	Время	Транзакция В
S-блокировка $P$ - успешна	$t_1$	
$_{ m Чтение} \ P = P_0$	$t_2$	
	t <sub>3</sub>	Х-блокировка Р - отвергается
	t <sub>4</sub>	Ожидание
Повторное чтение $P = P_0$	t <sub>5</sub>	Ожидание

Фиксация транзакции (Блокировка снимается)	<i>t</i> <sub>6</sub>	Ожидание
	$t_7$	X-блокировка <sup>р</sup> - успешна
	t <sub>8</sub>	$3$ апись $P_1 \rightarrow P$
	t <sub>9</sub>	Фиксация транзакции
		(Блокировка снимается)
Все правильно		

<u>Результат</u>. Транзакция В притормозилась до окончания транзакции А. В результате транзакция А дважды читает одни и те же данные правильно. После окончания транзакции А, транзакция В продолжила работу в обычном режиме.

# Фиктивные элементы (фантомы)

Транзакция A дважды выполняет выборку строк с одним и тем же условием. Между выборками вклинивается транзакция B, которая добавляет новую строку, удовлетворяющую условию отбора.

Транзакция А	Время	Транзакция В
S-блокировка строк, удовлетворяющих условию $\alpha$ . (Заблокировано n строк)	$t_1$	
Выборка строк, удовлетворяющих условию $\alpha$ . (Отобрано п строк)	<i>t</i> <sub>2</sub>	
	t <sub>3</sub>	Вставка новой строки, удовлетворяющей условию α.
	<i>t</i> <sub>4</sub>	Фиксация транзакции
S-блокировка строк, удовлетворяющих условию $\alpha$ . (Заблокировано $n+1$ строка)	t <sub>5</sub>	
Выборка строк, удовлетворяющих условию $\alpha$ . (Отобрано n+1 строк)	<i>t</i> <sub>6</sub>	
Фиксация транзакции	t <sub>7</sub>	
Появились строки, которых раньше не было		

<u>Результат</u>. Блокировка на уровне строк не решила проблему появления фиктивных элементов.

# Собственно несовместимый анализ

Длинная транзакция выполняет некоторый анализ по всей таблице, например, подсчитывает общую сумму денег на счетах клиентов банка для главного бухгалтера. Пусть на всех счетах находятся одинаковые суммы, например, по \$100. Короткая транзакция в этот момент выполняет перевод \$50 с одного счета на другой так, что общая сумма по всем счетам не меняется.

Транзакция А	Время	Транзакция В
S-блокировка счета $P_1$ - успешна	$t_1$	
Чтение счета $P_1 = 100$ и суммирование. $SUM = 100$	$t_2$	
	t <sub>3</sub>	$X$ -блокировка счета $\frac{P_3}{3}$ - успешна
	£4	Снятие денег со счета $P_3$ . $P_3:100 \rightarrow 50$
	t <sub>5</sub>	$X$ -блокировка счета $P_1$ - отвергается
	$t_6$	Ожидание
S-блокировка счета $\frac{P_2}{2}$ - успешна	$t_7$	Ожидание
Чтение счета $P_2 = 100$ и суммирование. $SUM = 200$	t <sub>8</sub>	Ожидание
S-блокировка счета $P_3$ - отвергается	t <sub>9</sub>	Ожидание
Ожидание	t <sub>10</sub>	Ожидание
Ожидание		Ожидание

<u>Результат</u>. Обе транзакции ожидают друг друга и не могут продолжаться. Возникла ситуация *тупика*.

#### Разрешение тупиковых ситуаций

Итак, при использовании протокола доступа к данным с использованием блокировок часть проблем разрешилось (не все), но возникла новая проблема - тупики:

- Проблема потери результатов обновления возник тупик.
- Проблема незафиксированной зависимости (чтение "грязных" данных, неаккуратное считывание) *проблема разрешилась*.
- Неповторяемое считывание проблема разрешилась.
- Появление фиктивных элементов проблема не разрешилась.
- Проблема несовместимого анализа возник тупик.

Общий вид *mynuкa* (dead locks) следующий:

Транзакция А	Время	Транзакция В
<u>.</u>	•	<u>.</u>

Блокировка объекта $P_1$ - успешна	$t_1$	
	$t_2$	Блокировка объекта $P_2$ -успешна
Блокировка объекта $\frac{P_2}{P_2}$ - конфликтует с блокировкой, наложенной транзакцией В	t <sub>3</sub>	
Ожидание	<i>t</i> <sub>4</sub>	Блокировка объекта $P_1$ - конфликтует с блокировкой, наложенной транзакцией $A$
Ожидание	$t_5$	Ожидание
Ожидание		Ожидание

Как видно, ситуация тупика может возникать при наличии не менее двух транзакций, каждая из которых выполняет не менее двух операций. На самом деле в тупике может участвовать много транзакций, ожидающих друг друга.

Т.к. нормального выхода из тупиковой ситуации нет, то такую ситуацию необходимо распознавать и устранять. Методом разрешения тупиковой ситуации является откат одной из транзакций (транзакции-жертвы) так, чтобы другие транзакции продолжили свою работу. После разрешения тупика, транзакцию, выбранную в качестве жертвы можно повторить заново.

Можно представить два принципиальных подхода к обнаружению тупиковой ситуации и выбору транзакции-жертвы:

- 1. СУБД не следит за возникновением тупиков. Транзакции сами принимают решение, быть ли им жертвой.
- 2. За возникновением тупиковой ситуации следит сама СУБД, она же принимает решение, какой транзакцией пожертвовать.

Первый подход характерен для так называемых настольных СУБД (FoxPro и т.п.). Этот метод является более простым и не требует дополнительных ресурсов системы. Для транзакций задается время ожидания (или число попыток), в течение которого транзакция пытается установить нужную блокировку. Если за указанное время (или после указанного числа попыток) блокировка не завершается успешно, то транзакция откатывается (или генерируется ошибочная ситуация). За простоту этого метода приходится платить тем, что транзакции-жертвы выбираются, вообще говоря, случайным образом. В результате из-за одной простой транзакции может откатиться очень дорогая транзакция, на выполнение которой уже потрачено много времени и ресурсов системы.

Второй способ характерен для промышленных СУБД (ORACLE, MS SQL Server и т.п.). В этом случае система сама следит за возникновением ситуации тупика, путем построения (или постоянного поддержания) графа ожидания транзакций. *Граф ожидания транзакций* - это ориентированный двудольный граф, в котором существует два типа вершин - вершины, соответствующие транзакциям, и вершины, соответствующие объектам захвата. Ситуация тупика возникает, если в графе ожидания транзакций имеется хотя бы один цикл. Одну из транзакций, попавших в цикл, необходимо откатить, причем, система сама может выбрать эту транзакцию в соответствии с некоторыми стоимостными соображениями (например, самую короткую, или с минимальным приоритетом и т.п.).

#### Преднамеренные блокировки

Как видно из анализа поведения транзакций, при использовании протокола доступа к данным не решается проблема фантомов. Это происходит оттого, что были рассмотрены только блокировки на уровне строк. Можно рассматривать блокировки и других объектов базы данных:

- Блокировка самой базы данных.
- Блокировка файлов базы данных.
- Блокировка таблиц базы данных.
- Блокировка страниц (Единиц обмена с диском, обычно 2-16 Кб. На одной странице содержится несколько строк одной или нескольких таблиц).
- Блокировка отдельных строк таблиц.
- Блокировка отдельных полей.

Кроме того, можно блокировать индексы, заголовки таблиц или другие объекты.

Чем крупнее объект блокировки, тем меньше возможностей для параллельной работы. Достоинством блокировок крупных объектов является уменьшение накладных расходов системы и решение проблем, не решаемых с использованием блокировок менее крупных объектов. Например, использование монопольной блокировки на уровне таблицы, очевидно, решает проблему фантомов.

Современные СУБД, как правило, поддерживают минимальный уровень блокировки на уровне строк или страниц. (В старых версиях настольной СУБД Paradox поддерживалась блокировка на уровне отдельных полей.).

При использовании блокировок объектов разной величины возникает проблема обнаружения уже наложенных блокировок. Если транзакция А пытается заблокировать таблицу, то необходимо иметь информацию, не наложены ли уже блокировки на уровне строк этой таблицы, несовместимые с блокировкой таблицы. Для решения этой проблемы используется протокол преднамеренных блокировок, являющийся расширением протокола доступа к данным. Суть этого протокола в том, что перед тем, как наложить блокировку на объект (например, на строку таблицы), необходимо наложить специальную преднамеренную блокировку (блокировку намерения) на объекты, в состав которых входит блокируемый объект - на таблицу, содержащую строку, на файл, содержащий таблицу, на базу данных, содержащую файл. Тогда наличие преднамеренной блокировки таблицы будет свидетельствовать о наличии блокировки строк таблицы и для другой транзакции, пытающейся блокировать целую таблицу не нужно проверять наличие блокировок отдельных строк. Более точно, вводятся следующие новые типы блокировок:

- Преднамеренная блокировка с возможностью взаимного доступа (IS-блокировка Intent Shared lock). Накладывается на некоторый составной объект Т и означает намерение блокировать некоторый входящий в Т объект в режиме S-блокировки. Например, при намерении читать строки из таблицы Т, эта таблица должна быть заблокирована в режиме IS (до этого в таком же режиме должен быть заблокирован файл).
- *Преднамеренная блокировка без взаимного доступа (IX-блокировка* Intent eXclusive lock). Накладывается на некоторый составной объект T и означает намерение блокировать некоторый входящий в T объект в режиме X-блокировки. Например, при намерении удалять или модифицировать строки из таблицы T эта

- таблица должна быть заблокирована в режиме IX (до этого в таком же режиме должен быть заблокирован файл).
- Преднамеренная блокировка как с возможностью взаимного доступа, так и без него (SIX-блокировка Shared Intent eXclusive lock). Накладывается на некоторый составной объект Т и означает разделяемую блокировку всего этого объекта с намерением впоследствии блокировать какие-либо входящие в него объекты в режиме X-блокировок. Например, если выполняется длинная операция просмотра таблицы с возможностью удаления некоторых просматриваемых строк, то можно заблокировать эту таблицу в режиме SIX (до этого захватить файл в режиме IS).

IS, IX и SIX-блокировки должны накладываться на сложные объекты базы данных (таблицы, файлы). Кроме того, на сложные объекты могут накладываться и блокировки типов S и X. Для сложных объектов (например, для таблицы базы данных) таблица совместимости блокировок имеет следующий вид:

	Транза	Транзакция В пытается наложить на таблицу блокировку:			
Транзакция А наложила на таблицу блокировку:	IS	S	IX	SIX	X
IS	Да	Да	Да	Да	Нет
S	Да	Да	Нет	Нет	Нет
IX	Да	Нет	Да	Нет	Нет
SIX	Да	Нет	Нет	Нет	Нет
X	Нет	Нет	Нет	Нет	Нет

Таблица 2 Расширенная таблица совместимости блокировок

Более точная формулировка протокола преднамеренных блокировок для доступа к данным выглядит следующим образом:

- 1. При задании X-блокировки для сложного объекта *неявным* образом задается X-блокировка для всех дочерних объектов этого объекта.
- 2. При задании S- или SIX-блокировки для сложного объекта *неявным* образом задается S-блокировка для всех дочерних объектов этого объекта.
- 3. Прежде чем транзакция наложит S- или IS-блокировку на заданный объект, она должна задать IS-блокировку (или более сильную) по крайней мере для одного родительского объекта этого объекта.
- 4. Прежде чем транзакция наложит X-, IX- или SIX-блокировку на заданный объект, она должна задать IX-блокировку (или более сильную) для всех родительских объектов этого объекта.
- 5. Прежде чем для данной транзакции будет отменена блокировка для данного объекта, должны быть отменены все блокировки для дочерних объектов этого объекта.

Понятие относительной силы блокировок можно описать при помощи следующей диаграммы приоритета (сверху - более сильные блокировки, снизу - более слабые):

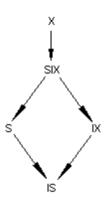


Таблица 3 Диаграмма приоритета блокировок

Замечание. Протокол преднамеренных блокировок не определяет однозначно, какие блокировки должны быть наложены на родительский объект при блокировании дочернего объекта. Например, при намерении задать S-блокировку строки таблицы, на таблицу, включающую эту строку, можно наложить любую из блокировок типа IS, S, IX, SIX, X. При намерении задать X-блокировку строки, на таблицу можно наложить любую из блокировок типа IX, SIX, X.

Посмотрим, как разрешается проблема фиктивных элементов (фантомов) с использованием протокола преднамеренных блокировок для доступа к данным.

Транзакция A дважды выполняет выборку строк с одним и тем же условием. Между выборками вклинивается транзакция B, которая добавляет новую строку, удовлетворяющую условию отбора.

Транзакция В перед попыткой вставить новую строку должна наложить на таблицу IX-блокировку, или более сильную (SIX или X). Тогда транзакция А, для предотвращения возможного конфликта, должна наложить такую блокировку на таблицу, которая не позволила бы транзакции В наложить IX-блокировку. По таблице совместимости блокировок определяем, что транзакция А должна наложить на таблицу S, или SIX, или X-блокировку. (Блокировки IS недостаточно, т.к. эта блокировка позволяет транзакции В наложить IX-блокировку для последующей вставки строк).

Транзакция А	Время	Транзакция В
S-блокировка таблицы (с целью потом блокировать строки) - успешна	$t_1$	
S-блокировка строк, удовлетворяющих условию а. (Заблокировано п строк)	$t_2$	
Выборка строк, удовлетворяющих условию а. (Отобрано п строк)	<i>t</i> <sub>3</sub>	
	£4	IX-блокировка таблицы (с целью потом вставлять строки) - отвергается из-за конфликта с S-блокировкой, наложенной транзакцией A

	t <sub>5</sub>	Ожидание
	t <sub>6</sub>	Ожидание
S-блокировка строк, удовлетворяющих условию $\alpha$ . (Заблокировано п строк)	t7	Ожидание
Выборка строк, удовлетворяющих условию а. (Отобрано п строк)	t <sub>8</sub>	Ожидание
Фиксация транзакции - блокировки снимаются	t <sub>9</sub>	Ожидание
	t <sub>10</sub>	IX-блокировка таблицы (с целью потом вставлять строки) - успешна
	t <sub>11</sub>	Вставка новой строки, удовлетворяющей условию а.
	t <sub>12</sub>	Фиксация транзакции
Транзакция А дважды читает один и тот же набор строк Все правильно		

<u>Результат</u>. Проблема фиктивных элементов (фантомов) решается, если транзакция А использует преднамеренную S-блокировку или более сильную.

Замечание. Т.к. транзакция А собирается только *читать* строки таблицы, то *минимально необходимым* условием в соответствии с протоколом преднамеренных блокировок является преднамеренная IS-блокировка таблицы. Однако этот тип блокировки не предотвращает появление фантомов. Таким образом, транзакцию А можно запускать с *разными уровнями изолированности* - предотвращая или допуская появление фантомов. Причем, оба способа запуска *соответствуют* протоколу преднамеренных блокировок для доступа к данным.

# Предикатные блокировки

Другим способом блокирования является блокировка не объектов базы данных, а условий, которым могут удовлетворять объекты. Такие блокировки называются *предикатными блокировками*.

Поскольку любая операция над реляционной базой данных задается некоторым условием (т.е. в ней указывается не конкретный набор объектов базы данных, над которыми нужно выполнить операцию, а условие, которому должны удовлетворять объекты этого набора), то удобным способом было бы S или X-блокирование именно этого условия. Однако при попытке использовать этот метод в реальной СУБД возникает трудность определения совместимости различных условий. Действительно, в языке SQL допускаются условия с подзапросами и другими сложными предикатами. Проблема совместимости сравнительно легко решается для случая простых условий, имеющих вид:

 ${Имя атрибута {= | <> | > | >= | < | <=} Значение}$ 

```
[\{OR \mid AND\} \{ \text{Имя атрибута } \{= | <> | > | >= | < | <= \} \}  Значение\}....]
```

Проблема фиктивных элементов (фантомов) легко решается с использованием предикатных блокировок, т.к. вторая транзакция не может вставить новые строки, удовлетворяющие уже заблокированному условию.

Заметим, что блокировка всей таблицы в каком-либо режиме фактически есть предикатная блокировка, т.к. каждая таблица имеет предикат, определяющий какие строки содержатся в таблице и блокировка таблицы есть блокировка предиката этой таблицы.

#### Метод временных меток

Альтернативный метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций и не требующий построения графа ожидания транзакций основан на использовании *временных меток*.

Основная идея метода состоит в следующем: если транзакция A началась раньше транзакции B, то система обеспечивает такой режим выполнения, как если бы A была целиком выполнена до начала B.

Для этого каждой транзакции Т предписывается временная метка t, соответствующая времени начала Т. При выполнении операции над объектом r базы данных транзакция Т помечает его своей временной меткой и типом операции (чтение или изменение).

Перед выполнением операции над объектом r транзакция B выполняет следующие действия:

- Проверяет, не закончилась ли транзакция A, пометившая этот объект. Если A закончилась, B помечает объект r своей временной меткой и выполняет операцию.
- Если транзакция A не завершилась, то B проверяет конфликтность операций. Если операции неконфликтны, при объекте г остается или проставляется временная метка с меньшим значением (более ранняя), и транзакция В выполняет свою операцию.
- Если операции В и А конфликтуют, то если t(A) > t(B) (т.е. транзакция А является более "молодой", чем В), то транзакция А откатывается и, получив новую временную метку, начинается заново. Транзакция В продолжает работу.
- Если же t(A) < t(B) (А "старше" В), то транзакция В откатывается и, получив новую временную метку, начинается заново. Транзакция А продолжает работу.

В итоге система обеспечивает такую работу, при которой при возникновении конфликтов всегда откатывается более молодая транзакция (начавшаяся позже).

Очевидным недостатком метода временных меток является то, что может откатиться более дорогая транзакция, начавшаяся позже более дешевой.

К другим недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования блокировок. Это связано с тем, что конфликтность транзакций определяется более грубо.

#### Механизм выделения версий данных

Использование блокировок гарантирует сериальность планов выполнения смеси транзакций за счет общего замедления работы - конфликтующие транзакции ожидают, когда транзакция, первой заблокировавшая некоторый объект, не освободит его. Без блокировок не обойтись, если все транзакции изменяют данные. Но если в смеси транзакций присутствуют как транзакции, изменяющие данные, так и только читающие данные, можно применить альтернативный механизм обеспечения сериальности, свободный от недостатков метода блокировок. Этот метод состоит в том, что транзакциям, читающим данные, предоставляется как бы "своя" версия данных, имевшаяся в момент начала читающей транзакции. При этом транзакция не накладывает блокировок на читаемые данные, и, поэтому, не блокирует другие транзакции, изменяющие данные. Такой механизм называется механизм выделения версий и заключается в использовании журнала транзакций для генерации разных версий данных.

Журнал транзакций предназначен для выполнения операции отката при неуспешном выполнении транзакции или для восстановления данных после сбоя системы. Журнал транзакций содержит старые копии данных, измененных транзакциями.

Кратко суть метода состоит в следующем:

- Для каждой транзакции (или запроса) запоминается текущий системный номер (*SCN System Current Number*). Чем позже начата транзакция, тем больше ее SCN.
- При записи страниц данных на диск фиксируется SCN транзакции, производящей эту запись. Этот SCN становится текущим системным номером страницы данных.
- Транзакции, только читающие данные не блокируют ничего в базе данных.
- Если транзакция A читает страницу данных, то SCN транзакции A сравнивается с SCN читаемой страницы данных.
- Если SCN страницы данных меньше или равен SCN транзакции A, то транзакция A читает эту страницу.
- Если SCN страницы данных больше SCN транзакции A, то это означает, что некоторая транзакция B, начавшаяся позже транзакции A, успела изменить или сейчас изменяет данные страницы. В этом случае транзакция A просматривает журнал транзакция назад в поиске первой записи об изменении нужной страницы данных с SCN меньшим, чем SCN транзакции A. Найдя такую запись, транзакция A использует старый вариант данных страницы.

Рассмотрим, как решается проблема несовместного анализа с использованием механизма выделения версий.

Длинная транзакция выполняет некоторый анализ по всей таблице, например, подсчитывает общую сумму денег на счетах клиентов банка для главного бухгалтера. Пусть на всех счетах находятся одинаковые суммы, например, по \$100. Короткая транзакция в этот момент выполняет перевод \$50 с одного счета на другой так, что общая сумма по всем счетам не меняется.

Транзакция А	Время	Транзакция В
Проверка SCN счета $P_1$ - SCN транзакции больше	$t_1$	
SCN счета.		
Чтение счета $P_1 = 100$ без наложения блокировки и		
суммирование.		
<i>SUM</i> = 100		

	$t_2$	X-блокировка счета $P_3$ - успешна
	t <sub>3</sub>	Снятие денег со счета $P_3$ . $P_3:100 \rightarrow 50$
	t <sub>4</sub>	X-блокировка счета $P_1$ - успешна
	t <sub>5</sub>	Помещение денег на счет $P_1$ . $P_1:100 \rightarrow 150$
	t <sub>6</sub>	Фиксация транзакции (Снятие блокировок)
Проверка SCN счета $\frac{P_2}{S}$ - SCN транзакции больше SCN счета.  Чтение счета $\frac{P_2}{S}$ = $100$ без наложения блокировка и суммирование. $\frac{SUM}{S}$ = 200	t <sub>7</sub>	
Проверка SCN счета $P_3$ - SCN транзакции <b>МЕНЬШЕ</b> SCN счета.  Чтение старого варианта счета $P_3 = 100$ и суммирование. $SUM = 300$	t <sub>8</sub>	
Фиксация транзакции	t <sub>9</sub>	
Сумма на счетах посчитана правильно.		

<u>Результат</u>. Транзакция A, начавшаяся первой не тормозит конкурирующую транзакцию B. При обнаружении конфликта (чтение транзакцией A измененного счета 3), транзакции A предоставляется своя версия данных, которая была на момент начала транзакции A.

#### Теорема Есварана о сериализуемости

Концепция способности к упорядочению была впервые предложена Есвараном [50].

В этой работе был предложен протокол двухфазной блокировки:

- 1. Перед выполнение каких-либо операций с некоторым объектом, транзакция должна заблокировать этот объект.
- 2. После снятия блокировки, транзакция не должна накладывать никаких других блокировок.

Транзакции, используемые в этом протоколе, не различаются по типам и считаются монопольными. Описанные выше протоколы доступа к данным с использованием S- и X- блокировок и протокол преднамеренных блокировок являются модификациями протокола двухфазной блокировки для случая, когда блокировки имеют различные типы.

Есвараном сформулирована следующая теорема:

<u>Теорема Есварана</u>. Если все транзакции в смеси подчиняются протоколу двухфазной блокировки, то для всех чередующихся графиков запуска существует возможность упорядочения.

Протокол называется двухфазным, потому что он характеризуется двумя фазами:

- *1 фаза нарастание блокировок*. Во время этой фазы накладываются блокировки, и производится работа с заблокированными объектами.
- *2 фаза снятие блокировок*. Во время этой фазы блокировки только снимаются. Работа с ранее заблокированными данными может продолжаться.

Работа транзакции может выглядеть приблизительно, как на рисунке:

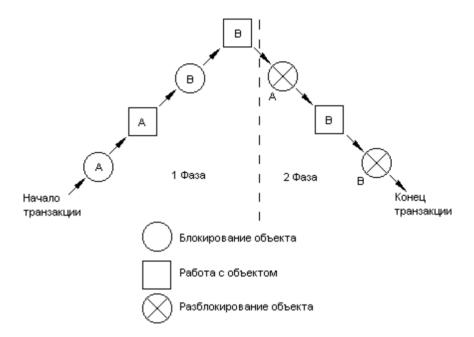


Рисунок 1 Работа транзакции по протоколу двухфазной блокировки

На следующем рисунке показан пример транзакции, не подчиняющийся протоколу двухфазной блокировки:

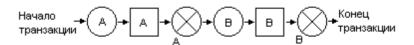


Рисунок 2 Транзакция, не подчиняющаяся протоколу двухфазной блокировки

На практике, как правило, вторая фаза сводится к одной операции завершения транзакции (или отката транзакции) с одновременным снятием всех блокировок.

Если некоторая транзакция А не подчиняется протоколу двухфазной блокировки (и, следовательно, состоит не менее чем из двух операция блокирования и разблокирования),

то всегда можно построить другую транзакцию В, которая при чередующемся выполнении вместе с А приводит к графику, не подлежащему упорядочению и неверному.

# Реализация изолированности транзакций средствами SQL

### Уровни изоляции

Стандарт SQL не предусматривает понятие блокировок для реализации сериализуемости смеси транзакций. Вместо этого вводится понятие уровней изоляции. Этот подход обеспечивает необходимые требования к изолированности транзакций, оставляя возможность производителям различных СУБД реализовывать эти требования своими способами (в частности, с использованием блокировок или выделением версий данных).

Стандарт SQL предусматривает 4 уровня изоляции:

- READ UNCOMMITTED уровень незавершенного считывания.
- READ COMMITTED уровень завершенного считывания.
- REPEATABLE READ уровень повторяемого считывания.
- SERIALIZABLE уровень способности к упорядочению.

Если все транзакции выполняются на уровне способности к упорядочению (принятом по умолчанию), то чередующееся выполнение любого множества параллельных транзакций может быть упорядочено. Если некоторые транзакции выполняются на более низких уровнях, то имеется множество способов нарушить способность к упорядочению. В стандарте SQL выделены три особых случая нарушения способности к упорядочению, фактически именно те, которые были описаны выше как проблемы параллелизма:

- Неаккуратное считывание ("Грязное" чтение, незафиксированная зависимость).
- Неповторяемое считывание (Частный случай несовместного анализа).
- Фантомы (Фиктивные элементы частный случай несовместного анализа).

Потеря результатов обновления стандартом SQL не допускается, т.е. на самом низком уровне изолированности транзакции должны работать так, чтобы не допустить потери результатов обновления.

Различные уровни изоляции определяются по возможности или исключению этих особых случаев нарушения способности к упорядочению. Эти определения описываются следующей таблицей:

Уровень изоляции	Неаккуратное считывание	Неповторяемое считывание	Фантомы
READ UNCOMMITTED	Да	Да	Да
READ COMMITTED	Нет	Да	Да
REPEATABLE READ	Нет	Нет	Да
SERIALIZABLE	Нет	Нет	Нет

Таблица 4 Уровни изоляции стандарта SQL

Синтаксис операторов SQL, определяющих уровни изоляции

Уровень изоляции транзакции задается следующим оператором:

```
SET TRANSACTION {ISOLATION LEVEL

{READ UNCOMMITTED

| READ COMMITTED

| REPEATABLE READ

| SERIALIZABLE }

| {READ ONLY | READ WRITE } } ....
```

Этот оператор определяет режим выполнения *следующей* транзакции, т.е. этот оператор не влияет на изменение режима той транзакции, в которой он подается. Обычно, выполнение оператора SET TRANSACTION выделяется как отдельная транзакция:

```
... (предыдущая транзакция выполняется со своим уровнем изоляции)

COMMIT;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

COMMIT;

... (следующая транзакция выполняется с уровнем изоляции REPEATABLE READ)
```

Если задано предложение ISOLATION LEVEL, то за ним должно следовать один из параметров, определяющих уровень изоляции.

Кроме того, можно задать признаки READ ONLY или READ WRITE. Если указан признак READ ONLY, то предполагается, что транзакция будет только читать данные. При попытке записи для такой транзакции будет сгенерирована ошибка. Признак READ ONLY введен для того, чтобы дать производителям СУБД возможность уменьшать количество блокировок путем использования других методов сериализации (например, метод выделения версий).

Oператор SET TRANSACTION должен удовлетворять следующим условиям:

- Если предложение ISOLATION LEVEL отсутствует, то по умолчанию принимается уровень SERIALIZABLE.
- Если задан признак READ WRITE, то параметр ISOLATION LEVEL не может принимать значение READ UNCOMMITTED.
- Если параметр ISOLATION LEVEL определен как READ UNCOMMITTED, то транзакция становится по умолчанию READ ONLY. В противном случае по умолчанию транзакция считается как READ WRITE.

### Выводы

Современные многопользовательские системы допускают одновременную работу большого числа пользователей. При этом если не предпринимать специальных мер, транзакции будут мешать друг другу. Этот эффект известен как проблемы параллелизма.

Имеются три основные проблемы параллелизма:

- Проблема потери результатов обновления.
- Проблема незафиксированной зависимости (чтение "грязных" данных, неаккуратное считывание).
- Проблема несовместимого анализа.

График запуска набора транзакций называется *последовательным*, если транзакции выполняются строго по очереди. Если график запуска набора транзакций содержит чередующиеся элементарные операции транзакций, то такой график называется *чередующимся*. Два графика называются *эквивалентными*, если при их выполнении будет получен один и тот же результат, *независимо от начального состояния базы данных*. График запуска транзакции называется *верным* (*сериализуемым*), если он эквивалентен какому-либо последовательному графику.

Решение проблем параллелизма состоит в нахождении такой стратегии запуска транзакций, чтобы обеспечить сериализуемость графика запуска и не слишком уменьшить степень параллельности.

Одним из методов обеспечения сериальности графика запуска является протокол доступа к данным при помощи блокировок. В простейшем случае различают *S-блокировки* (разделяемые) и *X-блокировки* (монопольные). Протокол доступа к данным имеет вид:

- 1. Прежде чем прочитать объект, транзакция должна наложить на этот объект S-блокировку.
- 2. Прежде чем обновить объект, транзакция должна наложить на этот объект X- блокировку. Если транзакция уже заблокировала объект S-блокировкой (для чтения), то перед обновлением объекта S-блокировка должна быть заменена X- блокировкой.
- 3. Если блокировка объекта транзакцией В отвергается оттого, что объект уже заблокирован транзакцией А, то транзакция В переходит в *состояние ожидания*. Транзакция В будет находиться в состоянии ожидания до тех пор, пока транзакция А не снимет блокировку объекта.
- 4. Х-блокировки, наложенные транзакцией А, сохраняются до конца транзакции А.

Если все транзакции в смеси подчиняются протоколу доступа к данным, то проблемы параллелизма решаются (почти все, кроме "фантомов"), но появляются тупики. Состояние *mynuкa* (*dead locks*) характеризуется тем, что две или более транзакции пытаются заблокировать одни и те же объекты, и бесконечно долго ожидают друг друга.

Для разрушения тупиков система периодически или постоянно поддерживает графа ожидания транзакций. Наличие циклов в графе ожидания свидетельствует о возникновении тупика. Для разрушения тупика одна из транзакций (наиболее дешевая с точки зрения системы) выбирается в качестве жертвы и откатывается.

Для решения проблемы "фантомов", а также для уменьшения накладных расходов, вызываемых большим количеством блокировок, применяются более сложные методы. Одним из таких методов являются *преднамеренные блокировки*, блокирующие объекты разной величины - строки, страницы, таблицы, файлы и др.

Альтернативным является метод предикатных блокировок

Имеются также методы сериализации, не использующие блокировки. Это *метод временных меток* и *механизм выделения версий*. Механизм выделения версий удобен

тем, что он позволяет одновременно, и читать, и изменять одни и те же данные разными транзакциями. Это увеличивает степень параллельности, и общую производительность системы.

Стандарт SQL *не предусматривает* понятие блокировок. Вместо этого вводится понятие *уровней изоляции*. Стандарт предусматривает 4 уровня изоляции:

- **READ UNCOMMITTED** уровень незавершенного считывания.
- READ COMMITTED уровень завершенного считывания.
- REPEATABLE READ уровень повторяемого считывания.
- SERIALIZABLE уровень способности к упорядочению.

Транзакции могут запускаться с различными уровнями изоляции.

# Глава 11. Транзакции и восстановление данных

В данной главе изучаются возможности восстановления данных после сбоев системы, т.е. свойство (Д) - долговечность транзакций.

Главное требование долговечности данных транзакций состоит в том, что данные зафиксированных транзакций должны сохраняться в системе, даже если в следующий момент произойдет сбой системы. Казалось бы, самый простой способ обеспечить такую гарантию - это во время каждой операции сразу записывать все изменения на дисковые носители. Такой способ не является удовлетворительным, т.к. имеется существенное различие в скорости работы с оперативной и с внешней памятью. Единственный способ достичь приемлемой скорости работы состоит в буферизации страниц базы данных в оперативной памяти. Это означает, что данные попадают во внешнюю долговременную память не сразу после внесения изменений, а через некоторое (достаточно большое) время. Тем не менее, что-что во внешней памяти должно оставаться, т.к. иначе неоткуда получить информацию для восстановления.

Требование атомарности транзакций утверждает, что не законченные или откатившиеся транзакции не должны оставлять следов в базе данных. Это означает, что данные должны храниться в базе данных с избыточностью, позволяющей иметь информацию, по которой восстанавливается состояние базы данных на момент начала неудачной транзакции. Такую избыточность обычно обеспечивает журнал транзакций. Журнал транзакций содержит детали всех операций модификации данных в базе данных, в частности, старое и новое значение модифицированного объекта, системный номер транзакции, модифицировавшей объект и другая информация.

#### Виды восстановления данных

Восстановление базы данных может производиться в следующих случаях:

- *Индивидуальный откат транзакции*. Откат индивидуальной транзакции может быть инициирован либо самой транзакцией путем подачи команды ROLLBACK, либо системой. СУБД может инициировать откат транзакции в случае возникновения какой-либо ошибки в работе транзакции (например, деление на нуль) или если эта транзакция выбрана в качестве жертвы при разрешении тупика.
- *Мягкий сбой системы* (*аварийный отказ программного обеспечения*). Мягкий сбой характеризуется утратой оперативной памяти системы. При этом поражаются все выполняющиеся в момент сбоя транзакции, теряется содержимое всех буферов

- базы данных. Данные, хранящиеся на диске, остаются неповрежденными. Мягкий сбой может произойти, например, в результате аварийного отключения электрического питания или в результате неустранимого сбоя процессора.
- Жесткий сбой системы (аварийный отказ аппаратуры). Жесткий сбой характеризуется повреждением внешних носителей памяти. Жесткий сбой может произойти, например, в результате поломки головок дисковых накопителей.

Во всех трех случаях основой восстановления является избыточность данных, обеспечиваемая журналом транзакций.

Как и страницы базы данных, данные из журнала транзакций не записываются сразу на диск, а предварительно буферизируются в оперативной памяти. Таким образом, система поддерживает два вида буферов - буферы страниц базы данных и буферы журнала транзакций.

Страницы базы данных, содержимое которых в буфере (в оперативной памяти) отличается от содержимого на диске, называются "грязными" (dirty) страницами. Система постоянно поддерживает список "грязных" страниц - dirty-список. Запись "грязных" страниц из буфера на диск называется выталкиванием страниц во внешнюю память. Очевидно, необходимо предусмотреть такие правила выталкивания буферов базы данных и буферов журнала транзакций, которые обеспечивали бы два требования:

- 1. Максимальную скорость выполнения транзакций. Для этого необходимо выталкивать страницы как можно реже. В идеале, если оперативная память была бы бесконечной, и сбои никогда бы не происходили, наилучшим выходом была бы загрузка всей базы данных в оперативную память, работа с данными только в оперативной памяти, и запись измененных страниц на диск только в момент завершения работы всей системы.
- 2. Гарантию, что при возникновении сбоя (любого типа), данные завершенных транзакций можно было бы восстановить, а данные незавершенных транзакций бесследно удалить, т.е. обеспечение восстановления последнего согласованного состояния базы данных. Для этого что-то выталкивать на диск все-таки необходимо, даже если мы обладали бы бесконечной оперативной памятью.

Таким образом, имеется две причины для периодического выталкивания страниц во внешнюю память - недостаток оперативной памяти и возможность сбоев.

Основным принципом согласованной политики выталкивания буфера журнала и буферов страниц базы данных является то, что запись об изменении объекта базы данных должна попадать во внешнюю память журнала раньше, чем измененный объект оказывается во внешней памяти базы данных. Соответствующий протокол журнализации (и управления буферизацией) называется Write Ahead Log (WAL) - "пиши сначала в журнал", и состоит в том, что если требуется вытолкнуть во внешнюю память измененный объект базы данных, то перед этим нужно гарантировать выталкивание во внешнюю память журнала записи о его изменении. Это означает, что если во внешней памяти базы данных содержится объект, к которому применена некоторая команда модификации, то во внешней памяти журнала транзакций содержится запись об этой операции. Обратное неверно - если во внешней памяти журнала содержится запись о некотором изменении объекта, то во внешней памяти базы данных может и не быть самого измененного объекта.

Дополнительное условие на выталкивание буферов накладывается тем требованием, что каждая успешно завершившаяся транзакция должна быть реально зафиксирована во внешней памяти. Какой бы сбой не произошел, система должна быть в состоянии восстановить состояние базы данных, содержащее результаты всех зафиксированных к моменту сбоя транзакций.

Третьим условием выталкивания буферов является ограниченность объемов буферов базы данных и журнала транзакций. Периодически или при наступлении определенного события (например, количество страниц в dirty-списке превысило определенный порог, или количество свободных страниц в буфере уменьшилось и достигло критического значения) система принимает так называемую контрольную точку. Принятие контрольной точки включает выталкивание во внешнюю память содержимого буферов базы данных и специальную физическую запись контрольной точки, которая представляет собой список всех осуществляемых в данный момент транзакций.

Оказывается, что минимальным требованием, гарантирующим возможность восстановления последнего согласованного состояния базы данных, является выталкивание при фиксации транзакции во внешнюю память журнала всех записей об изменении базы данных этой транзакцией. При этом последней записью в журнал, производимой от имени данной транзакции, является специальная запись о конце этой транзакции.

### Индивидуальный откат транзакции

Для того чтобы можно было выполнить по журналу транзакций индивидуальный откат транзакции, все записи в журнале от данной транзакции связываются в обратный список. Началом списка для не закончившихся транзакций является запись о последнем изменении базы данных, произведенном данной транзакцией. Для закончившихся транзакций (индивидуальные откаты которых уже невозможны) началом списка является запись о конце транзакции, которая обязательно вытолкнута во внешнюю память журнала. Концом списка всегда служит первая запись об изменении базы данных, произведенном данной транзакцией. В каждой записи имеется уникальный системный номер транзакции, чтобы можно было восстановить прямой список записей об изменениях базы данных данной транзакцией.

Индивидуальный откат транзакции выполняется следующим образом:

- Просматривается список записей, сделанных данной транзакцией в журнале транзакций (от последнего изменения к первому изменению).
- Выбирается очередная запись из списка данной транзакции.
- Выполняется противоположная по смыслу операция: вместо операции INSERT выполняется соответствующая операция DELETE, вместо операции DELETE выполняется INSERT, и вместо прямой операции UPDATE обратная операция UPDATE, восстанавливающая предыдущее состояние объекта базы данных.
- Любая из этих обратных операций также журнализируются. Это необходимо делать, потому что во время выполнения индивидуального отката может произойти мягкий сбой, при восстановлении после которого потребуется откатить такую транзакцию, для которой не полностью выполнен индивидуальный откат.
- При успешном завершении отката в журнал заносится запись о конце транзакции.

#### Восстановление после мягкого сбоя

Несмотря на протокол WAL, после мягкого сбоя не все физические страницы базы данных содержат измененные данные, т.к. не все "грязные" страницы базы данных были вытолкнуты во внешнюю память.

Последний момент, когда гарантированно были вытолкнуты "грязные" страницы - это момент принятия последней контрольной точки. Имеется 5 вариантов состояния транзакций по отношению к моменту последней контрольной точки и к моменту сбоя:

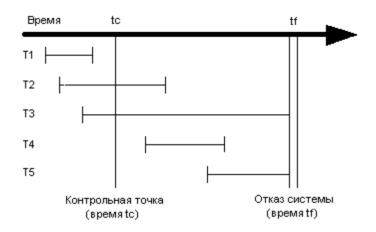


Рисунок 1 Пять вариантов транзакций

Последняя контрольная точка принималась в момент tc. Мягкий сбой системы произошел в момент tf. Транзакции T1-T5 характеризуются следующими свойствами:

- *T1* транзакция успешно завершена до принятия контрольной точки. Все данные этой транзакции сохранены в долговременной памяти как записи журнала, так и страницы данных, измененные этой транзакцией. Для транзакции T1 никаких операций по восстановлению не требуется.
- *T2* транзакция начата до принятия контрольной точки и успешно завершена после контрольной точки, но до наступления сбоя. Записи журнала транзакций, относящиеся к этой транзакции вытолкнуты во внешнюю память. Страницы данных, измененные этой транзакцией, только частично вытолкнуты во внешнюю память. Для данной транзакции необходимо повторить заново те операции, которые были выполнены после принятия контрольной точки.
- ТЗ транзакция начата до принятия контрольной точки и не завершена в результате сбоя. Такую транзакцию необходимо откатить. Проблема, однако, в том, что часть страниц данных, измененных этой транзакцией, уже содержится во внешней памяти это те страницы, которые были обновлены до принятия контрольной точки. Следов изменений, внесенных после контрольной точки в базе данных нет. Записи журнала транзакций, сделанные до принятия контрольной точки, вытолкнуты во внешнюю память, те записи журнала, которые были сделаны после контрольной точки, отсутствуют во внешней памяти журнала.
- *Т4* транзакция начата после принятия контрольной точки и успешно завершена до сбоя системы. Записи журнала транзакций, относящиеся к этой транзакции вытолкнуты во внешнюю память журнала. Изменения в базе данных, внесенные этой транзакцией, полностью отсутствуют во внешней памяти базы данных. Такую транзакцию необходимо повторить целиком.
- T5 транзакция начата после принятия контрольной точки и не завершена в результате сбоя. Никаких следов этой транзакции нет ни во внешней памяти

журнала транзакций, ни во внешней памяти базы данных. Для такой транзакции никаких действий предпринимать не нужно, ее как бы и не было вовсе.

Восстановление системы после мягкого сбоя осуществляется как *часть процедуры перезагрузки* системы. При перезагрузке системы транзакции Т2 и Т4 необходимо частично или полностью повторить, транзакцию Т3 - частично откатить, для транзакций Т1 и Т5 никаких действий предпринимать не нужно. При перезагрузке система выполняет следующие действия:

- Создается два списка транзакций UNDO (отменить) и REDO (повторить). В список UNDO заносятся все транзакции из последней записи контрольной точки (т.е. все транзакции, выполнявшиеся в момент принятия контрольной точки). Список REDO остается пустым. В нашем случае будет: UNDO = {T2, T3}, REDO = {}.
- Начиная с записи контрольной точки просматривается вперед журнал транзакций.
- Если в журнале транзакций обнаруживается запись о начале транзакции, то эта транзакция добавляется в список UNDO. В нашем случае будет: UNDO = {T2, T3, T4}, REDO = { }. Заметим, что следов транзакции Т5 в журнале транзакций нет.
- Если в файле регистрации обнаруживается запись COMMIT об окончании транзакции, то эта транзакция добавляется в список REDO. В нашем случае будет: UNDO = {T2, T3, T4}, REDO = {T2, T4}. Заметим, что записи о конце этих транзакций *имеются* во внешней памяти журнала транзакций в соответствии с минимальным требованием выталкивания записей журнала при фиксации транзакции.
- Когда достигается конец журнала транзакций, оба списка анализируются. При этом из списка UNDO удаляются те транзакции, которые попали в список REDO. В нашем случае будет: UNDO = {T3}, REDO = {T2, T4}.
- После этого система просматривает журнал транзакций назад, начиная с момента контрольной точки и откатывая все транзакции из списка UNDO. В нашем случае будут откатываться те операции транзакции Т3, которые были выполнены до принятия контрольной точки.
- Окончательно, система просматривает журнал транзакций вперед, начиная с момента контрольной точки, и повторно выполняет все операции транзакций из списка REDO. В нашем случае, система выполнит повторно все операции транзакции Т4 и те операции транзакции Т2, которые были выполнены после принятия контрольной точки.

## Восстановление после жесткого сбоя

При жестком сбое база данных на диске нарушается физически. Основой восстановления в этом случае является журнал транзакций и *архивная копия базы данных*. Архивная копия базы данных должна создаваться периодически, а именно с учетом скорости наполнения журнала транзакций.

Восстановление начинается с обратного копирования базы данных из архивной копии. Затем выполняется просмотр журнала транзакций для выявления всех транзакций, которые закончились успешно до наступления сбоя. (Транзакции, закончившиеся откатом до наступления сбоя, можно не рассматривать). После этого по журналу транзакций в прямом направлении повторяются все успешно законченные транзакции. При этом нет необходимости отката транзакций, прерванных в результате сбоя, т.к. изменения, внесенными этими транзакциями, отсутствуют после восстановления базы данных из резервной копии.

Наиболее плохим случаем является ситуация, когда разрушены физически и база данных, и журнал транзакций. В этом случае единственное, что можно сделать - это восстановить состояние базы данных на момент последнего резервного копирования. Для того чтобы не допустить возникновение такой ситуации, базу данных и журнал транзакций обычно располагают на физически разных дисках, управляемых физически разными контроллерами.

## Восстановление данных и стандарт SQL

Стандарт языка SQL не содержит требований к восстановимости данных, оставляя эти вопросы на усмотрение разработчиков СУБД.

#### Выводы

Главное требование долговечности данных транзакций состоит в том, что данные зафиксированных транзакций должны сохраняться в системе, даже если в следующий момент произойдет сбой системы. Избыточность хранения данных, позволяющую восстановить систему после сбоя обычно обеспечивает журнал транзакций.

Восстановление базы данных может производиться в следующих случаях:

- Индивидуальный откат транзакции.
- Мягкий сбой системы (аварийный отказ программного обеспечения).
- Жесткий сбой системы (аварийный отказ аппаратуры).

Страницы базы данных и журнала транзакций не записываются сразу на диск, а предварительно буферизируются в оперативной памяти. Страницы базы данных, содержимое которых в буфере отличается от содержимого на диске, называются "грязными" (dirty) страницами. Запись "грязных" страниц из буфера на диск называется выталкиванием страниц во внешнюю память.

Основным принципом согласованной политики выталкивания буфера журнала и буферов страниц базы данных является протокол журнализации  $\mathit{Write\ Ahead\ Log\ (WAL)}$  - " $\mathit{nuuuu\ chaчaлa\ b\ журнал}$ ".

Минимальным требованием, гарантирующим возможность восстановления последнего согласованного состояния базы данных, является выталкивание при фиксации транзакции во внешнюю память журнала всех записей об изменении базы данных этой транзакцией.

Индивидуальный откат транзакции выполняется при помощи журнала транзакций.

Восстановление системы после мягкого сбоя осуществляется как часть процедуры перезагрузки системы. При перезагрузке системы транзакции проходят процедуру идентификации для выявления завершившихся и прерванных в результате сбоя транзакций. Транзакции, успешно завершившиеся до наступления сбоя, и данные о которых отсутствуют в базе данных, повторяются заново. Транзакции, не успевшие завершиться к моменту сбоя, и данные о которых имеются в базе данных, откатываются.

Восстановление системы после жесткого сбоя выполняется при помощи архивной копии базы данных и журнала транзакций.

# Список литературы

- 1. Атре Ш. Структурный подход к организации баз данных. М.: Финансы и статистика, 1983. 320 с.
- 2. Беренсон X., Бернштейн Ф., Грэй Д., Мелтон Д., О"Нил Э., О"Нил П. Критика уровней изолированности в стандарте ANSI SQL //СУБД. 1996. №2. С.45-60.
- 3. Бойко В.В., Савинков В.М. Проектирование баз данных информационных систем. М.: Финансы и стати-стика, 1989. 351 с.
- 4. Боуман Д, Эмерсон С., Дарновски М. Практическое руководство по SQL. Киев: Диалектика, 1997.
- 5. Васкевич Д. Стратегии клиент/сервер. Киев: Диалектика, 1997.
- 6. Гилуа М.М. Множественная модель данных в информационных системах. М.: Наука, 1992.
- 7. Голосов А.О. Аномалии в реляционных базах данных //СУБД. 1986. №3. С.23-28.
- 8. Грабер М. Введение в SQL. М.: Лори, 1996. 379 с.
- 9. Грабер М. Справочное руководство по SQL. М.: Лори, 1997. 291 с.
- 10. Дейт К. Руководство по реляционной СУБД DB2. М.: Финансы и статистика, 1988. 320 с.
- 11. Дейт К. Введение в системы баз данных //6-издание. Киев: Диалектика, 1998. 784 с.
- 12. Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ. М.: Мир, 1991. 252 с.
- 13. Диго С.М. Проектирование и использование баз данных. М.: Финансы и статистика, 1995. 208 с.
- 14. Злуф М.М. Query-by-Example: язык баз данных //СУБД. 1996. №3. С.149-160.
- 15. Кириллов В.В. Структуризованный язык запросов (SQL). СПб.: ИТМО, 1994. 80 с.
- 16. Кузнецов С.Д. Введение в системы управления базами данных //СУБД. 1995. №1,2,3,4, 1996. №1,2,3,4,5.
- 17. Кузнецов С.Д. Стандарты языка реляционных баз данных SQL: краткий обзор //СУБД. 1996. №2. С.6-36.
- 18. Кузнецов С.Д. Операционные системы для управления базами данных //СУБД. 1996. №3. С.95-102.
- 19. Кузнецов С.Д. Дубликаты, неопределенные значения, первичные и возможные ключи и другие экзотиче-ские прелести языка SQL //СУБД. 1997. №3. С.77-80.
- 20. Кузнецов С.Д. Неопределенная информация и трехзначная логика //СУБД. 1997. №5. С.65-67.
- 21. Ладыженский Г.М. Системы управления базами данных коротко о главном //СУБД. 1995. №1,2,3,4.
- 22. Мартин Д. Планирование развития автоматизированных систем. М.: Финансы и статистика, 1984. 196 с.
- 23. Мейер М. Теория реляционных баз данных. М.: Мир. 1987. 608 с.
- 24. Оззу М.Т., Валдуриз П. Распределенные и параллельные системы баз данных //СУБД. 1996. №4. С.4-26.
- 25. Озкарахан Э. Машины баз данных и управление базами данных. М.: Мир, 1989.
- 26. Пржиялковский В. В. Абстракции в проектировании БД //СУБД. 1998. №1. С.90-97.
- 27. Прохоров A, Определение оптимальной структуры базы данных //Informix magazine. Русское издание. 1998. Апрель.
- 28. Нагао М., Катаяма Т., Уэмура С. Структуры и базы данных. М.: Мир, 1986. 197 с.

- 29. Тиори Т., Фрай Д. Проектирование структур баз данных. В 2 кн., М.: Мир, 1985. Кн. 1. - 287 с.: Кн. 2. - 320 с.
- 30. Ульман Д. Основы систем баз данных. М.: Финансы и статистика, 1983. 334 с.
- 31. Ульман Д. Базы данных на Паскале. М.: Машиностроение, 1990. 386 с.
- 32. Хаббард Д. Автоматизированное проектирование баз данных. М.: Мир, 1984. 294 с.
- 33. Цаленко М.Ш. Моделирование семантики в базах данных. М.: Наука, 1988.
- 34. Цикритизис Д., Лоховски Ф. Модели данных. М.: Финансы и статистика, 1985. 344 с.
- 35. Чамберлин Д.Д., Астрахан М.М., Эсваран К.П., Грифитс П.П., Лори Р.А., Мел Д.В., Райшер П., Вейд Б.В. SEQUEL 2: унифицированный подход к определению, манипулированию и контролю данных //СУБД. 1996. №1. С.144-159.
- 36. Чаудхари С. Методы оптимизации запросов в реляционных системах //СУБД. 1998. №3. С.22-36.
- 37. Чен П. Модель "сущность-связь" шаг к единому представлению о данных //СУБД. 1995. №3. С.137-158.
- 38. ANSI X3.135-1992, American National Standart for Information Systems Database Language SQL, November, 1992.
- 39. Astrahan M.M., System R: A Relational Approach to Data Base Management //ACM Transactions on Data Base Systems. 1976. V1, 97, June.
- 40. Boyce R.F., Chamberlin D.D., King W.F., Hammer M.M. Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage //Communications ACM. 1975. V.18, November. P.621.
- 41. Chamberlin D.D., Raymond F.B. SEQUEL: A Structured English Query Language. //Proc. ACM-SIGMOD. 1974. Workshop, Ann Arbor, Michigan, May.
- 42. Chamberlin D.D., Gray J.N., Traiger L.L. Views, Authorization and Locking in a Relational Data Base System //Proceedings of AFIPS National Computer Conference, Anaheim, CA, May. 1975.
- 43. Codd E.F. Relation Model of Data for Large Shared Data Banks //Comm. ACM. 1970. V.13, №.6. Р.377-383. (Имеется перевод: Кодд Е.Ф. Реляционная модель данных для больших совместно используемых банков данных //СУБД. 1995. №1. C.145-160.)
- 44. Codd E.F. Normalized Data Base Structure: A Brief Tutorial //Proc. of 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control.- N.-Y.: ACM. 1971. P.1-17.
- 45. Codd E.F. A data base sublanguage founded on the relational calculus //Proc. ACM-SIGFIDET/ 1971. Workshop, San Diego, Calif., Nov. P.35-68.
- 46. Codd E.F. Further Normalization of the Data base Relational Model //Data Base Systems.- N.J.: Prentice-Hall, 1972. P.33-64.
- 47. Codd E.F. Recent investigations in relational data base systems //Proc. IFIP Congress. 1974. North-Holland Pub. Co., Amsterdam. P.1017-1021.
- 48. Codd E.F. Extending the Database Relation Model to Capture More Meaning. //ACM Transaction on Database Systems. 1979.- V.4, №4. Р.397-434. (Имеется перевод: Кодд Э.Ф. Расширение реляционной модели для лучщего отражения семантики //СУБД. 1996. №5-6. С.163-192.)
- 49. Eswaran K.P. Chamberlin D.D. Functional specifications of a subsystem for data base integrity //Proc. Very Large Data Base Conf., Framingham, Mass., Sept. 1975. P.48-68.
- 50. Eswaran K.P., Gray J.N., Lorie R.A., Traiger I.L. The Notions of Consistency and Predicate Locks in a Data Base System //CACM. 1976. V.19, №11.
- 51. Fagin R.A. Normal Form for Relational Databases That is Based on Domains and Key //ACM Transactions on Database Systems. 1981. V.6, №3. P.387-415.

- 52. Fagin R. Multivalued Dependencies and New Normal Form for Relational Databases //ACM TODS. 1977. V.2, №3.
- 53. Gray J., Lorie R., Putzolu G., Traiger I. Granularity of Locks and Degrees of Consistency in a Shared Data Base //in Readings in Database Systems, Second Edition, Chapter 3, Michael Stonebraker, Ed., Morgan Kaufmann. 1994.
- 54. Heath I.J. Unacceptable File Operations in Relational Database //Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access, and Control. San Diego, Calif. 1971.
- 55. Held G.D., Stonebraker M.R., Wong E. INGRES: A Relational Data Base System //Proceedings of AFIPS National Computer Conference, Anaheim, CA, May. 1975.
- 56. Meiton J., Simon A.R. Understanding The New SQL: A Comlete Guide //Morgan Kaufmann. 1993.
- 57. Reisner P., Boyce R.F., Chamberlin D.D. Human Factors Evaluation of Two Data Base Query Languages: SQUARE and SEQUEL //Proceedings of AFIPS National ComputerConference, Anaheim, CA, May. 1975.
- 58. Smith J.M., Smith D.C. Database Abstractions: Aggregation and Generalization. //ACM Transactions on Database Systems. 1977. V.2, №2, June.- Р.105-133. (Имеется перевод: Смит Д.М., Смит Д.К. Абстракции баз дан-ных: Агрегация и обобщение //СУБД. 1996. №2. С.141-160.)
- 59. Zloof M.M. Query By Example //Proceedings of AFIPS National Computer Conference, Anaheim, CA, May. 1975.