

Содержание

| | |
|--|-----------|
| 1 Основные понятия БД. Архитектура СБД | 2 |
| 1.1 База данных (БД) | 2 |
| 1.2 Система БД (СБД) | 2 |
| 1.3 Основные составляющие СБД | 2 |
| 1.4 Реляционные БД | 2 |
| 1.5 Дореляционные и постреляционные БД | 3 |
| 1.6 Три уровня архитектуры (внутренний, внешний, концептуальный) | 5 |
| 2 Реляционная модель БД | 7 |
| 2.1 Таблица, столбец, строка, поле | 7 |
| 2.2 Домены, отношения, атрибуты, кортежи | 7 |
| 2.3 Степень, каринальное число отношения | 11 |
| 2.4 Свойства отношений | 11 |
| 2.5 Виды отношений | 12 |
| 3 Целостность данных | 12 |
| 3.1 Ограничения (правила) целостности | 12 |
| 3.2 Потенциальные ключи, первичный ключ | 14 |
| 3.3 Внешние ключи | 15 |
| 3.4 Ссылочная целостность | 16 |
| 3.5 NULL-значения | 18 |
| 4 Проектирование БД | 20 |
| 4.1 Функциональные зависимости | 20 |
| 4.2 Нормальные формы | 20 |
| 4.3 Алгоритм нормализации | 21 |
| 4.4 Модель сущность (связь) | 23 |
| 5 Организация параллельной работы пользователей | 23 |
| 5.1 Транзакция. Свойства транзакции | 23 |
| 5.2 Двухфазная фиксация | 24 |
| 5.3 Проблемы параллелизма | 24 |
| 5.4 Блокировки | 28 |
| 5.5 Тупики | 28 |
| 5.6 Уровни изолированности транзакций | 28 |
| 5.7 Привилегии пользователей | 28 |
| 5.8 Решение проблем параллелизма | 28 |
| 5.9 Механизм выделения версий | 28 |
| 6 Дополнительная информация | 28 |
| 6.1 Области БД | 28 |
| 6.2 Вид конфликтов данных | 31 |

1 Основные понятия БД. Архитектура СБД

1.1 База данных (БД)

Определение:

База данных (БД) — набор постоянных данных, которые используются прикладными системами для какого-либо предприятия.

1.2 Система БД (СБД)

Определение:

Система баз данных (СБД) — это, по сути, не что иное, как компьютеризированная система хранения записей. Саму же базу данных можно рассматривать как подобие электронной картотеки, то есть хранилище для некоторого набора занесённых в компьютер файлов данных (где файл — абстрактный набор данных).

1.3 Основные составляющие СБД

1. данные;
2. аппаратное обеспечение;
3. программное обеспечение;
4. пользователи.

или

1. СУБД;
2. прикладное программное обеспечение базы данных и операционной системы;
3. технические средства, обеспечивающие обслуживание пользователей.

1.4 Реляционные БД

Определение реляционной БД:

Реляционной базой данных называется набор отношений.

1.5 Дореляционные и постреляционные БД

Файлы и файловые системы:

Первые БД были созданы на основе файловых систем. Для каждой прикладной программы предоставлялся свой набор данных, оформленный в виде файла со своей структурой.

Такие системы имели ряд недостатков:

1. ФС не знает конкретной структуры файла. Поэтому при малейших изменениях в структуре файлов приходилось менять программы, работающие с ними;
2. В файловой системе для каждого файла имеется информация о пользователе — владельце файла и определённые доступные для других пользователей. Из-за этого было очень трудно выстраивать ФС, так как часто файл созданные одной программой не доступен для другой;
3. Файл в файловой системе был недоступен для нескольких пользователей одновременно. Его можно было модифицировать одновременно только 1 человеку, что делало совместную работу невозможной.

Иерархические БД:

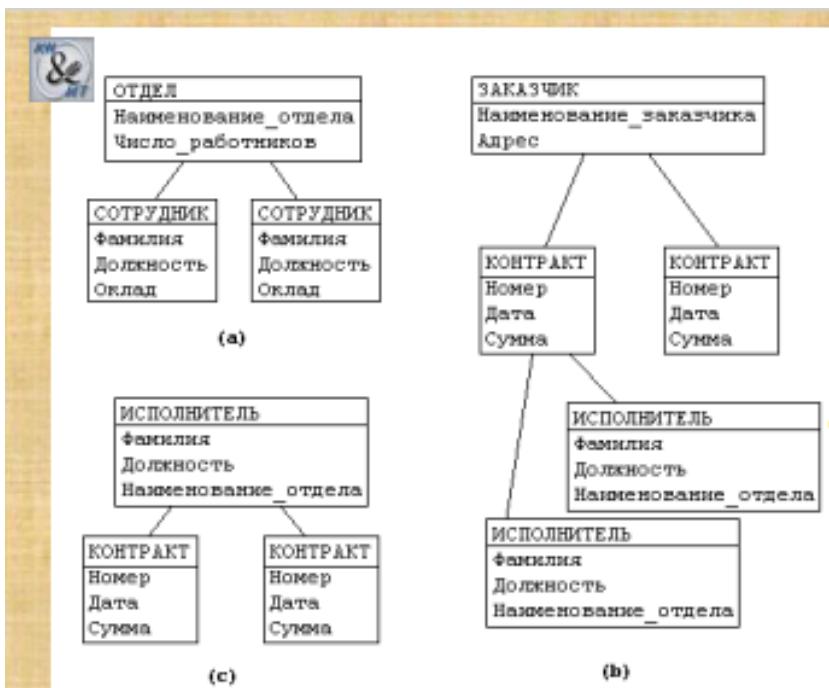


Иерархические БД

Рассмотрим следующую модель данных предприятия: предприятие состоит из отделов, в которых работают сотрудники. В каждом отделе может работать несколько сотрудников, но сотрудник не может работать более чем в одном отделе.

Поэтому, для информационной системы управления персоналом необходимо создать структуру, состоящую из родительской записи **ОТДЕЛ** (**НАИМЕНОВАНИЕ_ОТДЕЛА**, **ЧИСЛО_РАБОТНИКОВ**) и дочерней записи **СОТРУДНИК** (**ФАМИЛИЯ**, **ДОЛЖНОСТЬ**, **ОКЛАД**).

Для автоматизации учета контрактов с заказчиками необходимо создание еще одной иерархической структуры: заказчик - контракты с ним - сотрудники, задействованные в работе над контрактом. Это дерево будет включать записи
ЗАКАЗЧИК(**НАИМЕНОВАНИЕ_ЗАКАЗЧИКА**, **АДРЕС**),
КОНТРАКТ(**НОМЕР**, **ДАТА**, **СУММА**), **ИСПОЛНИТЕЛЬ** (**ФАМИЛИЯ**, **ДОЛЖНОСТЬ**, **НАИМЕНОВАНИЕ_ОТДЕЛА**).



Недостатки иерархических БД:

1. Частично дублируется информация между записями **СОТРУДНИК** и **ИСПОЛНИТЕЛЬ**, причем в иерархической модели данных не предусмотрена поддержка соответствия между парными записями.

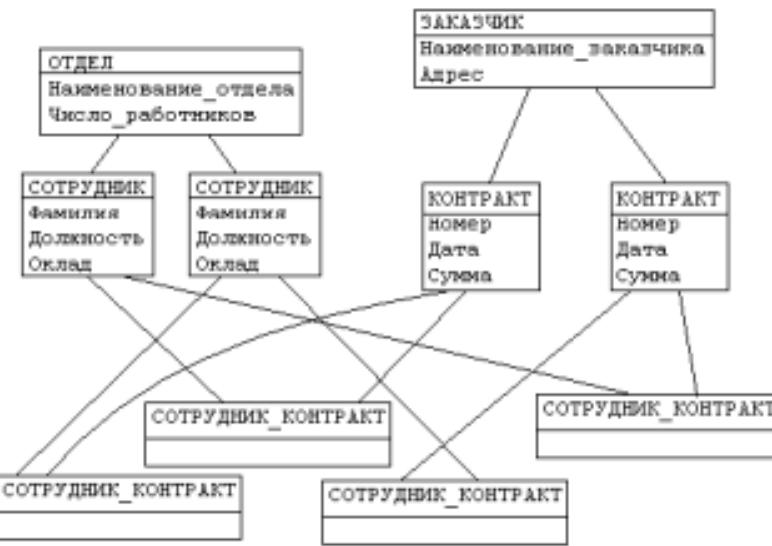
2. Иерархическая модель реализует отношение между исходной и дочерней записью по схеме **1:N**. Допустим, что исполнитель может принимать участие более чем в одном контракте (связь типа **M:N**). В этом случае в базу данных необходимо ввести еще одно групповое отношение, в котором **ИСПОЛНИТЕЛЬ** будет являться исходной записью, а **КОНТРАКТ** - дочерней (рис. (c)). Таким образом, опять дублируется информация.

7

Сетевые БД:



Сетевые БД



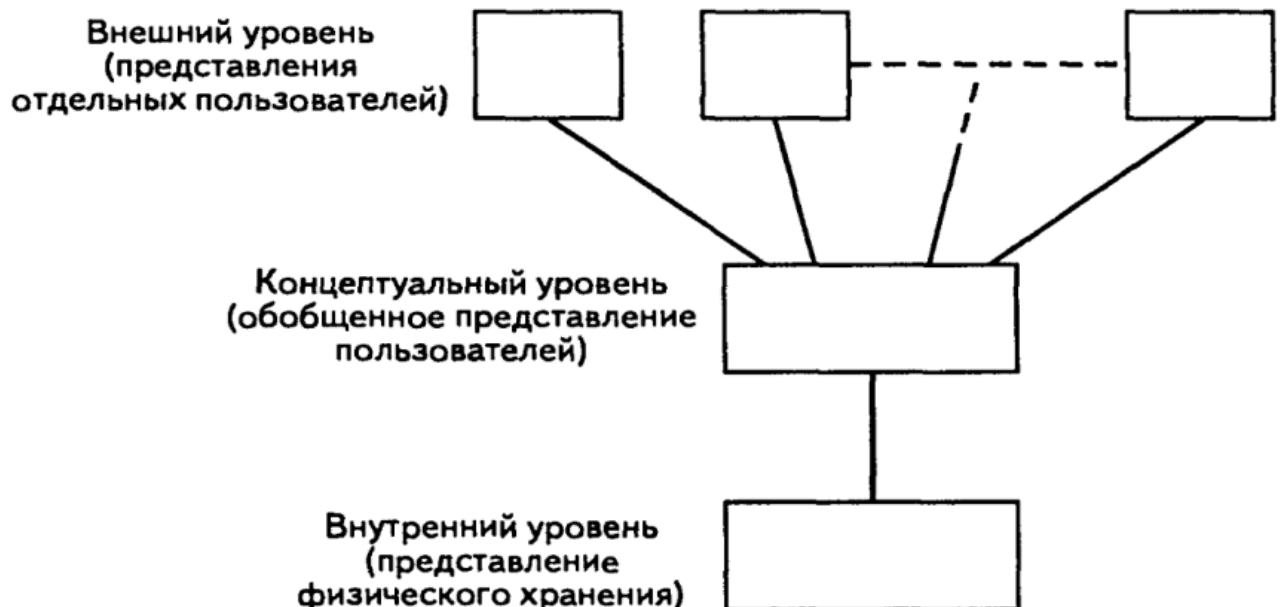
Иерархическая структура преобразовывается в сетевую следующим образом:

- 1) деревья (а) и (б) заменяются одной сетевой структурой, в которой запись СОТРУДНИК входит в обе группы;
- 2) для отображения типа M:N вводится запись СОТРУДНИК_КОНТРАКТ, которая не имеет полей и служит только для связи записей КОНТРАКТ и СОТРУДНИК.

«Если Вы хотите запутать базу данных – сделайте ее сетевой» - разработчик БД из IBM.

8

1.6 Три уровня архитектуры (внутренний, внешний, концептуальный)



Определение:

Внутренний уровень — уровень наиболее близкий к физическому хранению

данных, т.е. связанный с со способами сохранения информации на физических устройствах хранения. Внутренний уровень — собственно данные, расположенные в файлах или в страничных структурах, расположенных на внешних носителях информации.

Определение:

Концептуальный уровень — уровень, на котором база данных представлена в наиболее общем виде, который объединяет данные, используемые всеми приложениями, работающими с данной базой данных. Фактически концептуальный уровень отражает обобщенную модель предметной области (объектов реального мира), для которой создавалась база данных. Как любая модель, концептуальная модель отражает только существенные, с точки зрения обработки, особенности объектов реального мира.

Определение:

Внешний уровень — самый верхний уровень, где каждое представление имеет свое "видение" данных. Этот уровень определяет точку зрения на БД отдельных приложений. Каждое приложение видит и обрабатывает только те данные, которые необходимы именно этому приложению. Например, система распределения работ использует сведения о квалификации сотрудника, но ее не интересуют сведения об окладе, домашнем адресе и телефоне сотрудника, и наоборот, именно эти сведения используются в подсистеме отдела кадров.

Размышления:



| Внешний (PL/I) | Внешний (COBOL) |
|---|---|
| DCL 1 EMPP, 2 EMP# CHAR(6), 3 SAL FIXED BIN(31); | 01 EMPC. 02 EMPNO PIC X(6). 02 DEPTNO PIC X(4). |
| Концептуальный | |
| EMPLOYEE EMPLOYEE_NUMBER | CHARACTER (6) CHARACTER (4) |
| DEPARTMENT_NUMBER | |
| SALARY | NUMERIC (5) |
| Внутренний | |
| STORED_EMP | BYTES=20 |
| PREFIX | TYPE=BYTE(6), OFFSET=0 |
| EMP# | TYPE=BYTE(6), INDEX=EMPK, OFFSET=6, |
| DEPT# | TYPE=BYTE(4), OFFSET=12 |
| PAY | TYPE=FULLWORD, OFFSET=16 |

Трехуровневая архитектура БД позволяет обеспечить логическую (между 1 и 2 ур.) и физическую (между 2 и 3 ур.) независимость при работе с данными.

Логическая независимость предполагает возможность изменения одного приложения без корректировки других приложений, работающих с этой же БД.

Физическая независимость предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с данной БД.

Выделение концептуального уровня позволило разработать аппарат централизованного управления БД.

11

2 Реляционная модель БД

2.1 Таблица, столбец, строка, поле

2.2 Домены, отношения, атрибуты, кортежи

Определение домена:

Домен — допустимое потенциальное множество значений для отдельно взятого типа.

Дополнительная информация:

1.2. Домен

Понятие домена более специфично для баз данных, хотя и имеет некоторые аналогии с подтипами в некоторых языках программирования.

Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа. Например, в число значений домена "Имена" могут входить только те строки, которые могут изображать имя.

Данные считаются сравнимыми только в том случае, когда они относятся к одному домену.

Домены более точно можно определить как именованное множество скалярных значений одного типа. Эти скалярные значения называют **скалярами**. По сути, это наименьшая семантическая (смысловая) единица данных. У скаляров нет внутренней структуры, т.е. они не разложимы в данной реляционной модели.

Например, если имеется атрибут (свойство объекта) «ФИО», он предусматривает скаляры, содержащие фамилию, имя и отчество. Конечно, эти скаляры можно еще разбить на буквы, но тогда будет утрачен нужный смысл. То есть для данной модели наименьшими семантическими единицами данных будут именно фамилия, имя и отчество.

Из доменов, как уже говорилось, берутся значения атрибутов. На практике домены часто не описывают, а задают типом, форматом и другими свойствами данных. Каждый атрибут должен быть определен на **единственном домене**.

Основное назначение доменов — **ограничение сравнения** различных по смыслу атрибутов.

Например: Если для атрибутов №ЗачетнойКнижки отношения Студенты и №Кабинета для отношения Кабинеты домены заданы следующим образом:

№ зачетной книжки = {100000, 100001, 100002, ... 999999}

№ кабинета = {1, 2, 3, ... 999},

то система выдаст ошибку на запрос типа: «Вывести всех студентов, № зачетной книжки которых совпадает с № кабинета». Если же домены не определены, а определен только целый тип данных для атрибутов №ЗачетнойКнижки и №Кабинета, то подобный запрос выполнится, хотя не будет иметь смысла.

Еще одно возможное применение доменов – использование их в специальных **запросах**. Например, «Какие отношения в БД включают атрибуты, определенные на домене «№ зачетной книжки»?». В системе, поддерживающей домены, такой запрос будет иметь смысл и результатом его будет список отношений, где используется № зачетной книжки (это могут быть отношения Студенты, Занятия, Успеваемость, ...). А в системе, где домены не определены, реализовать такого рода запрос гораздо сложнее – если через имена атрибутов, то они могут не совпадать (имена атрибутов, содержащих № зачетной книжки могут варьироваться: № зачетки, № зачетной книжки и т.п.), а если через тип – то получится много лишних отношений, т.к. немало атрибутов может иметь целый тип данных.

Определение отношения, атрибута и кортежа:

Еще одно возможное применение доменов – использование их в специальных **запросах**. Например, «Какие отношения в БД включают атрибуты, определенные на домене «№ зачетной книжки»?». В системе, поддерживающей домены, такой запрос будет иметь смысл и результатом его будет список отношений, где используется № зачетной книжки (это могут быть отношения Студенты, Занятия, Успеваемость, ...). А в системе, где домены не определены, реализовать такого рода запрос гораздо сложнее – если через имена атрибутов, то они могут не совпадать (имена атрибутов, содержащих № зачетной книжки могут варьироваться: № зачетки, № зачетной книжки и т.п.), а если через тип – то получится много лишних отношений, т.к. немало атрибутов может иметь целый тип данных.

Что такое отношение в реляционной модели

У отношения нет физического представления. Реляционная база данных — это попытка отразить реляционную модель, а не ее точная копия.

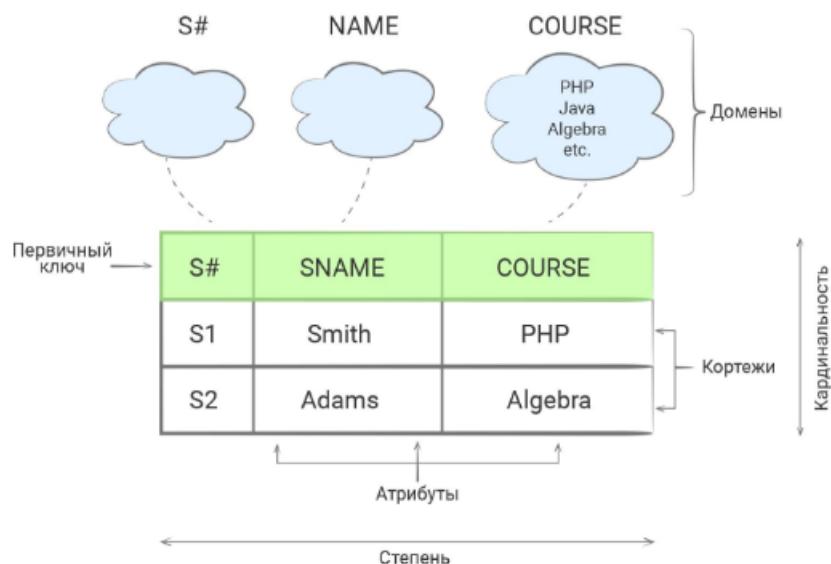
Реляционная модель опирается на раздел математики теорию множеств, которую нужно знать хотя бы немного. Множество — это совокупность произвольных элементов, которые объединены по некоторому признаку. Например, множество натуральных чисел или множество учеников одного класса. Изучить этот раздел подробнее вы можете в нашем курсе [«Теория множеств»](#).

Из любого множества можно выделить **подмножество** — множество элементов, все элементы которого входят в исходное множество. Подмножество — это часть множества. Например, множество натуральных чисел — это подмножество по отношению к множеству целых чисел, которое в свою очередь — подмножество рациональных чисел. Из этого следует, что натуральные числа — это подмножество рациональных чисел.

Еще одно важное понятие — **кортеж**. Это упорядоченный набор данных фиксированной длины. Элементами кортежа может быть все что угодно. Это математический способ представить некоторый набор связанных данных.

На основе этих понятий можно сформулировать более понятное определение отношения. Отношение — это множество кортежей, которые называются **телом отношения**. В нем каждый кортеж соответствует схеме.

Схема — это заголовок отношения. Она описывает общую структуру кортежей, количество элементов внутри них и их типы. Каждый такой элемент называется **атрибутом**:



Отношение визуально похоже на таблицу, но у него есть существенные отличия:

- Нет двух одинаковых элементов (кортежей)
- Порядок кортежей не определен
- Порядок атрибутов в заголовке не определен

По этой причине таблица непригодна для точного описания отношения. В любой таблице порядок столбцов (атрибуты схемы) и строк (кортеж) строго определен. При этом на практике мы оперируем таблицами и не можем игнорировать их.

По Батраевой:

1.3. Кортеж, отношение

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, \dots, D_n (не обязательно различных), содержит две части: заголовок и тело.

Заголовок отношения содержит фиксированное количество атрибутов отношения:

$(\langle A_1:D_1 \rangle, \dots, \langle A_n:D_n \rangle)$

Тело отношения содержит множество кортежей отношения. Каждый **кортеж отношения** представляет собой множество пар вида *<Имя_атрибута : Значение_атрибута>*:

$(\langle A_1:V_{11} \rangle, \dots, \langle A_n:V_{n1} \rangle)$

таких что значение V_{i1} атрибута A_i принадлежит домену D_i

Отношение обычно записывается в виде:

$R(\langle A_1:D_1 \rangle, \dots, \langle A_n:D_n \rangle)$, или короче $R(A_1, \dots, A_n)$, или просто R .

2.3 Степень, каринальное число отношения

Определение степени отношения:

Число атрибутов в отношении называют степенью отношения.

Каринальное число отношения:

Количество кортежей в отношении называется мощностью или каринальным числом отношения.

2.4 Свойства отношений

1. В отношении нет одинаковых кортежей. Тело отношения есть множество кортежей и, как всякое множество, не может содержать неразличимые

элементы. 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. 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. Одно и то же отношение может быть изображено разными таблицами, в которых столбцы идут в различном порядке;
4. Все значения атрибутов атомарны. Это четвертое отличие отношений от таблиц — в ячейки таблиц можно поместить что угодно — массивы, структуры, и даже другие таблицы.

2.5 Виды отношений

Существует два вида отношений: дочернее и родительское.

Определение дочернего отношения:

Отношение, входящее в связь со стороны "много" ("Поставки"), называется дочерним отношением.

Определение родительского отношения:

Отношение, входящее в связь со стороны "один" ("Поставщики"), называют родительским отношением.

3 Целостность данных

3.1 Ограничения (правила) целостности

Правило целостности сущностей (NULL значения в потенциальных ключах):

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

Правило целостности внешних ключей:

Так как внешние ключи фактически служат ссылками на кортежи в другом (или в том же самом) отношении, то эти ссылки не должны указывать на несуществующие объекты.

Формально:

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

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

Опр. 8. Ограничения целостности домена представляют собой ограничения, накладываемые только на допустимые значения домена. Ограничения домена не проверяются.

Например, ограничение домена «Возраст_военнослужащего» это условие «Возраст не менее 18 и не более 60».

Ограничения атрибута

Опр. 9. Ограничение целостности атрибута представляют собой ограничения, накладываемые на допустимые значения атрибута вследствие того, что атрибут основан на каком-либо домене. Ограничения атрибута проверяются.

89

Ограничения кортежа

Опр. 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.

Атрибут "Возраст_военнослужащего" в таблице "Спецгруппа", может иметь дополнительное ограничение "Возраст_военнослужащего не менее 25 и не более 45", помимо того, что этот атрибут уже имеет ограничение, определяемое доменом - "Возраст_военнослужащего" не менее 18 и не более 60".

Ограничения отношения

Опр. 11. **Ограничения целостности отношения** это ограничения, накладываемые только на допустимые значения *отдельного* отношения, и не являющиеся ограничением целостности кортежа (т.е. для его проверки не требуется информации о других отношениях)

Ограничение целостности сущности, задаваемое потенциальным ключом отношения, является ограничением отношения. 90

Ограничения базы данных

Опр. 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).

Ограничение целостности ссылок, задаваемое внешним ключом отношения, является ограничением базы данных.

3.2 Потенциальные ключи, первичный ключ

Определение потенциального (и чуть-чуть первичного) ключа:

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 R 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 называется **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.

Замечание. Понятие потенциального ключа является **семантическим** понятием и отражает некоторый смысл (трактовку) понятий из конкретной предметной области.

3.3 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.

Определение внешнего ключа:

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, как правило, не обладает *свойством уникальности*. Это, собственно, и дает тип отношения 1:N.

Замечание. Хотя каждое значение внешнего ключа обязано совпадать со значениями потенциального ключа в некотором кортеже родительского отношения, то обратное, вообще говоря, неверно.

Например, могут существовать поставщики, не поставляющие никаких деталей.

Замечание. Для внешнего ключа не требуется, чтобы он был компонентом некоторого потенциального ключа.

3.4 Ссыльная целостность

Операции ведущие к нарушению ссылочной целостности при изменении родительского отношения:

1. 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;
2. 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;
3. 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.

Операции ведущие к нарушению ссылочной целостности при изменении дочернего отношения:

1. 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;
2. 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;
3. Удаление кортежа в дочернем отношении. При удалении кортежа в дочернем отношении ссылочная целостность не нарушается.

Вывод:

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

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

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

Изменение начинается в родительском отношении и каскадно выполняется в дочернем отношении. Необходимо учитывать, 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.

Стратегии поддержания ссылочной целостности реализуются в основном двумя способами:

- на уровне определения данных

```
alter table POSTAVKI add foreign key (fkidTovar) references  
TOVAR(idTovar) on delete cascade
```

- с использованием триггеров

```
create trigger trTovar for Tovar before update as  
declare variable n;  
begin  
select count(*) from Postavki where old.idTovar = Postavki.fkidTovar into n;  
if n>0 then delete from Postavki where old.idTovar=Postavki.fkidTovar ;  
end
```

3.5 NULL-значения

Достаточно 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 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.

Дополнительная информация:

2.2. 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 есть null, если один или оба аргумента есть null. Таким образом, определение истинности логических выражений базируется на *трехзначной логике (three-valued logic, 3VL)*, в которой кроме значений Т - ИСТИНА и F - ЛОЖЬ, введено значение U - НЕИЗВЕСТНО. Логическое значение U - это то же самое, что и null-значение.

| AND | F | T | U | OR | F | T | U | NOT | |
|-----|---|---|---|----|---|---|---|-----|---|
| F | F | F | F | F | F | T | U | F | T |
| T | F | T | U | T | T | T | T | T | F |
| U | F | U | U | U | U | T | U | U | U |

Имеется несколько парадоксальных следствий применения трехзначной логики.

Парадокс 1. Null-значение не равно самому себе. Действительно, выражение $null = null$ дает значение не ИСТИНА, а НЕИЗВЕСТНО. Значит выражение не обязательно ИСТИНА!

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

Парадокс 3. $a \text{ or } (\text{not } a)$ не обязательно ИСТИНА. Значит, в трехзначной логике не работает принцип исключенного третьего (любое высказывание либо истинно, либо ложно).

4 Проектирование БД

4.1 Функциональные зависимости

Определение функциональной зависимости:

Определение 1. Пусть R - отношение. Множество атрибутов Y **функционально зависимо** от множества атрибутов X (X **функционально определяет** Y) т. и т.т., к. 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 в любом состоянии отношения). Символически функциональная зависимость (Φ_3) записывается $X \rightarrow Y$. Множество атрибутов X называется **дeterminантой функциональной зависимости**, а множество атрибутов Y называется **зависимой частью**.

4.2 Нормальные формы

Определение первой нормальной формы:

Первая нормальная форма — это и есть отношение. То есть отношения ни в первой нормальной форме не существует.

Можно сказать, что первая нормальная форма это отношение, атрибуты которого содержат только атомарные значения.

Определение второй нормальной формы:

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

Если потенциальный ключ отношения является простым, то отношение автоматически находится в 2НФ.

Отношения, полученные в результате декомпозиции автоматически находятся во второй нормальной форме.

Определение третьей нормальной формы:

Определение 3. Атрибуты называются **взаимно независимыми**, если ни один из них не является функционально зависимым от другого.

Определение 4. Отношение находится в **третьей нормальной форме (3НФ)** тогда и только тогда, когда отношение находится в 2НФ и все неключевые атрибуты взаимно независимы.

4.3 Алгоритм нормализации

Алгоритм нормализации (приведение к 3НФ)

Шаг 1 (Приведение к 1НФ). На первом шаге задается одно или несколько отношений, отображающих понятия предметной области. По модели предметной области выписываются обнаруженные функциональные зависимости. Все отношения автоматически находятся в 1НФ.

Шаг 2 (Приведение к 2НФ). 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 $R(K1, K2, A1, \dots, An, B1, \dots, Bn)$.

Ключ $\{K1, K2\}$ - сложный.

Функциональные зависимости:

$\{K1, K2\} \rightarrow (A1, \dots, An, B1, \dots, Bn)$ - зависимость всех атрибутов от ключа отношения.

$\{K1\} \rightarrow (A1, \dots, An)$ - зависимость некоторых атрибутов от части сложного ключа.

59

Декомпозированные отношения:

$R1(K1, K2, B1, \dots, Bn)$ - остаток от исходного отношения. Ключ $\{K1, K2\}$.

$R2(K1, A1, \dots, An)$ - атрибуты, вынесенные из исходного отношения вместе с частью сложного ключа. Ключ $K1$.

Шаг 3 (Приведение к ЗНФ). Если в некоторых отношениях обнаружена зависимость некоторых неключевых атрибутов от других неключевых атрибутов, то проводим декомпозицию этих отношений следующим образом: те неключевые атрибуты, которые зависят от других неключевых атрибутов выносятся в отдельное отношение. В новом отношении ключом становится детерминант функциональной зависимости:

Исходное отношение $R(K, A1, \dots, An, B1, \dots, Bn)$.

Ключ K .

Функциональные зависимости:

$K \rightarrow (A1, \dots, An, B1, \dots, Bn)$ - зависимость всех атрибутов от ключа отношения.

$\{A1, \dots, An\} \rightarrow (B1, \dots, Bn)$ - зависимость некоторых неключевых атрибутов от других неключевых атрибутов.

60

Декомпозированные отношения:

$R1(K, A1, \dots, An)$ - остаток от исходного отношения. Ключ K .

$R2(A1, \dots, An, B1, \dots, Bn)$ - атрибуты, вынесенные из исходного отношения вместе с детерминантом функциональной зависимости. Ключ $(A1, \dots, An)$.

Замечание. Обычно при проектировании логической модели данных разработчики сразу строят отношения в ЗНФ, не следуя алгоритму нормализации. Несмотря на это алгоритм важен и для практики:

- алгоритм показывает, какие проблемы возникают при разработке слабо нормализованных отношений.

- во многих случаях модель предметной области не бывает правильно разработана с первого шага. Забыли о чем-то упомянуть эксперты в предметной области, между разработчиками и экспертами возможно недопонимание, изменились правила и требования к предметной области. В результате могут появиться новые зависимости. В этом случае и необходимо использовать алгоритм нормализации, чтобы проверить, что отношения остались в ЗНФ.

61

4.4 Модель сущность (связь)

5 Организация параллельной работы пользователей

5.1 Транзакция. Свойства транзакции

Опр. 1. **Транзакция** - это последовательность операторов манипулирования данными, выполняющаяся как единое целое (все или ничего) и переводящая БД из одного целостного состояния в другое целостное состояние.

Транзакция обладает четырьмя важными свойствами, известными как **свойства АСИД**:

(А) Атомарность. Транзакция выполняется как атомарная операция - либо выполняется вся транзакция целиком, либо она целиком не выполняется.

(С) Согласованность. Транзакция переводит БД из одного согласованного (целостного) состояния в другое согласованное (целостное) состояние. Внутри транзакции согласованность БД 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 сохраняться в БД, даже если в следующий момент произойдет сбой системы.

80

5.2 Двухфазная фиксация

5.3 Проблемы параллелизма

Проблемы параллельной работы транзакций

Различают три основные проблемы параллелизма:

- *Проблема потери результатов обновления*
- *Проблема незафиксированной зависимости (чтение "грязных" данных, неаккуратное считывание).*
- *Проблема несовместимого анализа*

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 |
|-------------------------------|-------|----------------------------|
| Чтение $P=P_0$ | t1 | |
| | t2 | Чтение $P=P_0$ |
| Запись $P_1 \rightarrow P$ | t3 | |
| | t4 | Запись $P_2 \rightarrow P$ |
| Фиксация | t5 | |
| | t6 | Фиксация |
| Потеря результатов обновления | | |

Результат. 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 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 |
|-------------------------------------|-------|--------------------------------------|
| | t1 | Чтение $P=P_0$ |
| | t2 | Запись $P_1 \rightarrow P$ |
| Чтение $P=P_1$ | t3 | |
| Работа с прочитанными данными P_1 | t4 | |
| | t5 | Откат транзакции $P_0 \rightarrow P$ |
| Фиксация | t6 | |
| Работа с «грязными» данными | | |

Результат. Транзакция А в своей работе использовала данные, которых нет в БД. Более того, А использовала данные, которых нет, и не было в БД! Действительно, после отката транзакции В, должна восстановиться ситуация, как если бы транзакция В вообще никогда не выполнялась. Т.о., результаты работы транзакции А некорректны. q7

Проблема несовместимого анализа

Проблема несовместимого анализа включает несколько различных вариантов:

- **Неповторяемое считывание.**
- **Фиктивные элементы (фантомы).**
- **Собственно несовместимый анализ.**

Неповторяющееся 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 |
|-----------------------------------|-------|----------------------------|
| Чтение $P=P_0$ | t1 | |
| | t2 | Чтение $P=P_0$ |
| | t3 | Запись $P_1 \rightarrow P$ |
| | t4 | Фиксация |
| Повторное чтение $P=P_1$ | t5 | |
| Фиксация | t6 | |
| <i>Неповторяющееся считывание</i> | | |

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.

Результат. Транзакция А работает с данными, которые, с точки зрения транзакции А, самопроизвольно изменяются.

Фиктивные элементы (фантомы)

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 U (отобрано n строк) | t1 | |
| | t2 | Вставка новой строки удовлетворяющей условию U |
| | t3 | Фиксация |
| Выборка строк , удовлетворяющих условию U (отобрано $n+1$ строка) | t4 | |
| Фиксация | t5 | |
| <i>Появились строки, которых раньше не было</i> | | |

Результат. Транзакция А ничего не знает о существовании транзакции В, и, т.к. сама она не меняет ничего в БД, то ожидает, что после повторного отбора будут отобраны те же самые строки.

100

Собственно несовместимый анализ

В смеси присутствуют две транзакции – длинная и короткая.

Длинная выполняет некоторый анализ по всей таблице, например, подсчитывает общую сумму денег на счетах клиентов банка для главбуха. Пусть на всех счетах находятся 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 $P_1=100$ и $SUM=100$ | t1 | |
| | t2 | Снятие денег с P_3 . $P_3:100 \rightarrow 50$ |
| | t3 | Помещение денег на P_1 . $P_1:100 \rightarrow 150$ |
| | t4 | Фиксация |
| Чтение счета $P_2=100$ и $SUM=200$ | t5 | |
| Чтение счета $P_3=50$ и $SUM=250$ | t6 | |
| Фиксация | t7 | |
| Сумма по счетам 250, а д. б. 300 | | |

Результат. 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.

5.4 Блокировки

5.5 Тупики

5.6 Уровни изолированности транзакций

5.7 Привилегии пользователей

5.8 Решение проблем параллелизма

5.9 Механизм выделения версий

6 Дополнительная информация

6.1 Области БД

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.

Модель предметной области. Модель предметной области - это наши знания о предметной области. Знания могут быть как в виде неформальных знаний в мозгу эксперта, так и выражены формально при помощи каких-либо средств (текстовые описания предметной области, наборы должностных инструкций, и т.п.)

Описания предметной области обычно выполняются при помощи методик описания предметной области. Из наиболее известных можно назвать методику структурного анализа SADT и основанную на нем IDEF0, диаграммы потоков данных Гейна-Сарсона, методику объектно-ориентированного анализа UML, и др.

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

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 – «актер», «фильм», «режиссер», "зарплата". Примеры взаимосвязей между понятиями - «актер снимается в нескольких фильмах», «режиссер снимает фильм», «в одном фильме снимается несколько актеров». Примеры ограничений - "возраст режиссера не менее 18 лет".

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но без привязки к конкретной СУБД.

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.

Физическая модель данных. На еще более низком уровне находится физическая модель данных. Физическая модель данных описывает данные средствами конкретной СУБД. Будем считать, что физическая модель данных реализована средствами именно реляционной СУБД.

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 решения, принятые на уровне логического моделирования определяют некоторые границы, в пределах которых можно развивать физическую модель данных и 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.⁴⁶

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.

6.2 Вид конфликтов данных

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" данных.

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.

103