кафедра компьютерных и информационных технологий

курс лекций

В настоящем курсе рассматриваются вопросы организации баз данных и знаний. Это важная тема, без основательного знакомства с которой, в наше время, невозможно стать квалифицированным специалистом в сфере информационных технологий.

Основное назначение данного курса – систематическое введение в идеи и методы, используемые в современных системах управления базами данных. В курсе не рассматривается какая-либо одна популярная СУБД; излагаемый материал в равной степени относится к любой современной системе. Как показывает опыт, без знания основ теории баз данных трудно на серьезном уровне работать с конкретными системами, как бы хорошо они не были документированы.

Содержание

ЛЕКЦИЯ 1. Понятие СУБД. Функции СУБД 7

1.1 Введение 7

1.2 Понятие БД и СУБД 7

1.3 Уровни абстракции в СУБД. Функции абстрактных данных 9

1.4 Представления 10

1.5 Функции СУБД 11

1.6 Экспертные системы и базы знаний 11

ЛЕКЦИЯ 2. Модели БД 13

2.1 Обзор ранних (дореляционных) СУБД 13

2.2 Системы, основанные на инвертированных списках 13

2.3 Иерархическая модель 14

2.4 Сетевая модель 16

2.5 Основные достоинства и недостатки ранних СУБД 17

ЛЕКЦИЯ 3. Реляционная модель и ее характеристики. Целостность в реляционной модели 18

3.1 Представление информации в реляционных БД 18

3.2 Домены 19

3.3 Отношения. Свойства и виды отношений 20

3.4 Целостность реляционных данных 21

3.5 Потенциальные и первичные ключи 22

3.6 Внешние ключи 22

3.7 Ссылочная целостность 23

3.8 Значения NULL и поддержка ссылочной целостности 24

ЛЕКЦИЯ 4. Реляционная алгебра 25

4.1 Понятие реляционной алгебры 25

4.2 Замкнутость в реляционной алгебре 25

4.3 Традиционные операции над множествами 25

4.4 Свойства основных операций реляционной алгебры 27

4.5 Специальные реляционные операции 28

ЛЕКЦИЯ 5. Вопросы проектирования БД 34

5.1 Понятие проектирования БД 34

5.2 Функциональные зависимости 35

5.3 Тривиальные и нетривиальные зависимости 36

5.4 Замыкание множества зависимостей и правила вывода Армстронга 36

5.5 Неприводимое множество зависимостей 38

5.6 Нормальные формы – основные понятия 38

5.7 Декомпозиция без потерь и функциональные зависимости 39

5.8 Диаграммы функциональных зависимостей 40

ЛЕКЦИЯ 6. Проектирование БД. Нормальные формы отношений 42

6.1 Первая нормальная форма. Возможные недостатки отношения в 1НФ 42

6.2 Вторая нормальная форма. Возможные недостатки отношения во 2НФ 44

6.3 Третья нормальная форма. Возможные недостатки отношения в 3НФ 45

6.4 Нормальная форма Бойса-Кодда 46

ЛЕКЦИЯ 7. Проектирование БД. Нормальные формы отношений (продолжение) 49

7.1 Многозначные зависимости 49

7.2 Четвертая нормальная форма 51

7.3 Зависимости соединения 51

7.4 Пятая нормальная форма 53

7.5 Итоговая схема процедуры нормализации 53

ЛЕКЦИЯ 8. Проектирование БД методом сущность-связь. ER-диаграммы 55

8.1 Возникновение семантического моделирования 55

8.2 Основные понятия метода 55

8.3 Диаграммы ER-экземпляров и ER-типа 56

8.4 Правила формирования отношений 59

8.5 Методология IDEF1 (самостоятельное изучение) 62

ЛЕКЦИЯ 9. Язык SQL 66

9.1 История создания и развития SQL 66

9.2 Основные понятия SQL 66

9.3 Запросы на чтение данных. Оператор SELECT 71

9.4 Многотабличные запросы на чтение (объединения). 75

ЛЕКЦИЯ 10. Язык SQL (продолжение) 77

10.1 Объединения и стандарт SQL2 77

10.2 Итоговые запросы на чтение. Агрегатные функции 80

10.3 Запросы с группировкой (предложение GROUP BY) 80

10.4 Вложенные запросы 82

ЛЕКЦИЯ 11. Язык SQL. (продолжение) 86

11.1 Внесение изменений в базу данных. 86

11.2 Удаление существующих данных (Оператор DELETE) 87

11.3 Обновление существующих данных (Оператор UPDATE) 87

11.4 Определение структуры данных в SQL 88

11.5 Понятие представления. 91

11.6 Представления в SQL. 92

11.7 Системный каталог (самостоятельное изучение) 93

ЛЕКЦИЯ 12. Обеспечение безопасности БД 99

12.1 Общие положения 99

12.2 Методы обеспечения безопасности 100

12.3 Избирательное управление доступом 101

12.4 Обязательное управление доступом 102

12.5 Шифрование данных 102

12.6 Контрольный след выполняемых операций 102

12.7 Поддержка мер обеспечения безопасности в языке SQL 103

12.8 Директивы GRANT и REVOKE 103

12.9 Представления и безопасность 105

ЛЕКЦИЯ 13. Физическая организация БД: структуры хранения и методы доступа 106

13.1 Доступ к базе данных 106

13.2 Кластеризация 108

13.3 Индексирование 108

13.4 Структуры типа Б-дерева 111

13.5 Хеширование 114

ЛЕКЦИЯ 14. Оптимизация запросов 116

14.1 Оптимизация в реляционных СУБД. 116

14.2 Пример оптимизации реляционного выражения 116

14.3 Обзор процесса оптимизации 117

14.4 Преобразование выражений 119

ЛЕКЦИЯ 15. Восстановление после сбоев 123

15.1 Понятие восстановления системы 123

15.2 Транзакции 123

15.3 Алгоритм восстановления после сбоя системы 125

15.4 Параллелизм. Проблемы параллелизма 127

15.5 Понятие блокировки 129

15.6 Решение проблем параллелизма 130

15.7 Тупиковые ситуации 132

15.8 Способность к упорядочению 133

15.9 Уровни изоляции транзакции 134

15.10 Поддержка в языке SQL 135

ЛЕКЦИЯ 16. Технологии СУБД 136

16.1 Распределенные базы данных 136

16.2 Принципы функционирования распределенной БД 136

16.3 Системы типа клиент/сервер 139

16.4 Серверы баз данных 139

ЛЕКЦИЯ 17. Современные постреляционные модели БД 141

17.1 Системы управления базами данных следующего поколения 141

17.2 Ориентация на расширенную реляционную модель 141

17.3 Объектно-ориентированные СУБД 143

# Понятие СУБД. Функции СУБД

1.1 Введение

1.2 Понятие БД и СУБД

1.3 Уровни абстракции в СУБД. Функции абстрактных данных

1.4 Представления

1.5 Функции СУБД

1.6 Экспертные системы и базы знаний

## Введение

Исторически сложившееся развитие вычислительных систем обусловило необходимость хранения в электронном (машиночитаемом) виде все большего количества информации. Одновременно с совершенствованием и дальнейшим развитием вычислительных систем росли объемы информации, подлежащей обработке и хранению. Сложности, возникшие при решении на практике задач структурированного хранения и эффективной обработки возрастающих объемов информации, стимулировали исследования в соответствующих областях. Задачи хранения и обработки данных были формализованы. Была создана теоретическая база для решения задач такого класса, результатом реализации на практике которой стали системы, предназначенные для организации обработки, хранения и предоставления доступа к информации. Позже такие системы стали называть системами баз данных.

Одновременно с развитием систем баз данных, происходило интенсивное развитие средств вычислительной техники, используемой для работы с большими объемами информации. Вычислительная мощность и, особенно, объемы запоминающих устройств первых вычислительных систем были недостаточны для хранения и обработки информации в объемах, необходимых на практике.

По мере развития систем баз данных, менялись принципы организации данных в них: первоначально данные представлялись на основе иерархической, а в последствии сетевой модели. В конце 1970-х – начале 1980-х годов начали появляться первые реляционные продукты. В настоящее время системы баз данных на основе реляционной модели занимают лидирующее положение, несмотря на заявления многих исследователей о скором переходе к объектно-ориентированным системам. В настоящее время объектно-ориентированные системы, тем не менее, развиваются, хотя темпы их развития и сдерживаются медленным принятием соответствующих стандартов. Кроме того, многие коммерческие реляционные системы приобретают объектно-ориентированные черты. На основании этого, можно предположить, что в будущем объектно-ориентированные системы будут постепенно вытеснять реляционные.

В настоящее время ведутся исследования в следующих направлениях:

1. дедуктивные системы;
2. экспертные системы;
3. расширяемые системы;
4. объектно-ориентированные системы.

## Понятие БД и СУБД

Система баз данных – это компьютеризированная система основная задача которой – хранение информации и предоставление доступа к ней по требованию.

Система баз данных включает в себя (рис. 1.1):

1. данные, непосредственно сохраняемые в базе данных;
2. аппаратное обеспечение;
3. программное обеспечение;
4. пользователей:
	1. прикладные программисты;
	2. конечные пользователи;
	3. администраторы баз данных.

рис. 1.1 Система баз данных.

Прикладные программисты

Конечные пользователи

Система управления базами данных

База данных

### Данные.

Данные в базе данных являются интегрированными и, как правило, общими. Понятие интегрированных данных подразумевает возможность представления базы данных как объединение нескольких отдельных файлов данных, полностью или частично не перекрывающихся. Понятие общие подразумевает возможность использования отдельных областей данных, в базе данных несколькими различными пользователями.

### Аппаратное обеспечение.

К аппаратному обеспечению системы относятся накопители для хранения информации, вместе с устройствами ввода-вывода, контролерами устройств и т.д.; вычислительная техника, используемая для поддержки работы ПО системы.

### Программное обеспечение.

Программное обеспечение является промежуточным слоем между собственно физической базой данных и пользователями системы и называется диспетчером базы данных или системой управления базами данных, СУБД (DBMS). Все запросы пользователей обрабатываются СУБД.

Таким образом, СУБД – это специализированное программное обеспечение, предоставляющее пользователю базы данных возможность работать с ней, не вникая в детали хранения информации на уровне программного обеспечения.

### Пользователи.

Прикладные программисты – отвечают за написание прикладных программ, использующих базу данных.

Конечные пользователи – работают с базой данных непосредственно, через рабочую станцию или терминал. Конечный пользователь может получить доступ к базе данных используя соответствующее прикладное ПО.

Администраторы базы данных – технические специалисты, осуществляющие создание БД, технический контроль работы СУБД и др. операции. Администраторы базы данных отвечают за реализацию решений администратора данных. Администратор данных решает, какие данные необходимо хранить в БД, обеспечивает поддержание порядка при обслуживании и использовании хранимых в БД данных.

Функции администратора базы данных:

1. определение концептуальной схемы. Администратор БД определяет какие именно данные необходимо сохранять в БД. Этот процесс обычно называют логическим (или концептуальным) проектированием БД. После определения содержимого БД на абстрактном уровне, администратор БД создает соответствующую концептуальную схему, с помощью концептуального ЯОД.
2. Определение внутренней схемы. Администратор БД решает, как данные должны быть представлены в хранимой БД. Этот процесс называют физическим проектированием. После завершения физического проектирования администратор БД с помощью внутреннего ЯОД должен создать соответствующую структуру хранения, а также определить отображение между внутренней и концептуальной схемой.
3. Взаимодействие с пользователями. Администратор БД обеспечивает пользователей необходимыми им данными. Для этого администратор БД должен написать (или оказать пользователям помощь в написании) необходимых внешних схем. Кроме этого необходимо определить отображение между внешней и концептуальной схемами.
4. Определение правил безопасности и целостности.
5. Определение процедур резервного копирования и восстановления.
6. Управление производительностью и реагирование на изменяющиеся требования.

База данных состоит из некоторого набора постоянных данных, которые используются прикладными системами какого-либо предприятия. Под словом "постоянные" подразумеваются данные, которые отличаются от других, более изменчивых данных, таких, как промежуточные данные и вообще все транзитные данные. "Постоянные" данные на самом деле могут недолго оставаться таковыми, поскольку данные в БД должны отражать об изменчивых объектах реального мира и отношениях между ними.

Использование баз данных для хранения информации позволяет организовать централизованное управление данными, что обеспечивает следующие преимущества:

1. возможность сокращения избыточности;
2. возможность устранения (до некоторой степени) противоречивости;
3. возможность общего доступа к данным;
4. возможность соблюдения стандартов;
5. возможность введения ограничений для обеспечения безопасности
6. возможность обеспечения целостности данных;
7. возможность сбалансировать противоречивые требования;
8. возможность обеспечения независимости данных. Поскольку программное обеспечение отделяется от данных, хранимых БД, изменения, вносимые в структуру БД, в большинстве случаев не приводят к необходимости внесения радикальных изменений в программное обеспечение.

## Уровни абстракции в СУБД. Функции абстрактных данных

Существует 3 уровня архитектуры СУБД (рис. 1.2):

1. Внутренний уровень - наиболее близкий к физическому хранению. Он связан со способами хранения информации на физических устройствах хранения;
2. Внешний уровень - наиболее близкий к пользователям. Он связан со способами представления данных для отдельных пользователей;
3. Концептуальный уровень - является промежуточным между двумя первыми. Этот уровень связан с обобщенными представлениями пользователей, в отличие от внешнего уровня, связанного с индивидуальными представлениями пользователей.

## Представления

Соответственно трем уровням архитектуры выделяют три уровня абстракции данных в СУБД.

### Внешний уровень – внешнее представление

Внешний уровень - индивидуальный уровень пользователя. Пользователь может быть как прикладным программистом, так и конечным пользователем с любым уровнем профессиональной подготовки. Каждый пользователь имеет свой язык общения с СУБД. Для программиста - это какой-либо язык программирования, для пользователя - язык запросов или язык, основанный на формах и меню. Любой из этих языков включает подъязык данных, т.е. множество операторов всего языка, связанное только с объектами и операциями баз данных. Т.о. подъязык данных встроен в базовый язык пользователя, который также обеспечивает на связанные с БД возможности.

Представление отдельного пользователя о БД на внешнем уровне архитектуры называют внешним представлением. Т.о. внешнее представление - это содержимое БД, каким его видит отдельный пользователь. Например, сотрудник отдела кадров видит БД как набор записей о сотрудниках плюс набор записей о подразделениях. В общем случае внешнее представление состоит из множества экземпляров каждого типа внешней записи, которые не обязательно совпадают с хранимыми записями. Подъязык данных пользователя определен в терминах внешних записей. Каждое внешнее представление определяется средствами внешней схемы, которая, в основном, состоит из определений каждого типа записей во внешнем представлении.

### Концептуальный уровень – концептуальное представление

Концептуальное представление - это представление всей информации БД в несколько более абстрактной форме по сравнению с физическим способом хранения данных. Концептуальное представление представляет данные такими, какими они есть на самом деле, а не такими, какими их вынужден видеть пользователь в рамках определенного языка. Концептуальное представление состоит из множества экземпляров каждого типа концептуальной записи, хотя в некоторых системах в способы концептуального представления данных могут быть другими - например, в виде объектов и связей между ними. Концептуальное представление определяется средствами концептуальной схемы, которая состоит из определений каждого типа концептуальных записей. При определении концептуальной схемы используется концептуальный язык определения данных, определения которого относятся только к содержанию информации. Концептуальное представление, т.о. обеспечивает независимость данных от способа их хранения.

### Внутренний уровень – внутреннее представление

Внутреннее представление - это представление нижнего уровня всей БД. Оно состоит из множества экземпляров каждого типа внутренней записи. Внутренняя запись соответствует хранимой записи. Внутреннее представление не связано с физическим уровнем и в нем не рассматриваются физические записи. Внутреннее представление предполагает существование бесконечного линейного адресного пространства. Подробности отображения этого пространства на физические устройства хранения не включены в общую архитектуру из-за сильной зависимости от системы.

Внутреннее представление описывается с помощью внутренней схемы, которая описывается с помощью внутреннего языка определения данных.

Между тремя уровнями представлений имеются два уровня отображений. Отображения концептуального уровня на внутренний и внешнего уровня на концептуальный. Отображения сохраняют независимость данных случае внесения в структуру БД изменений.

## Функции СУБД

1. Определение данных. СУБД должна допускать определения данных (внешние схемы, концептуальную и внутреннюю схемы, соответствующие отображения). Для этого СУБД включает в себя языковый процессор для различных языков определений данных.
2. Обработка данных. СУБД должна обрабатывать запросы пользователя на выборку, а также модификацию данных. Для этого СУБД включает в себя компоненты процессора языка обработки данных.
3. Безопасность и целостность данных. СУБД должна контролировать запросы и пресекать попытки нарушения правил безопасности и целостности.
4. Восстановление данных и дублирование. СУБД должна обеспечить восстановление данных после сбоев.
5. Словарь данных. СУБД должна обеспечить функцию словаря данных. Сам словарь можно считать системной базой данных, которая содержит данные о данных пользовательской БД, т.е. содержит определения других объектов системы. Словарь интегрирован в определяемую им БД и, поэтому, содержит описание самого себя.
6. Производительность. СУБД должна выполнять свои функции с максимальной производительностью.

## Экспертные системы и базы знаний

В последнее время появилась необходимость хранения и использования слабоструктурированных данных, каковыми являются, например, человеческие знания в экспертных системах.

Экспертная система – система искусственного интеллекта, включающая знания об определенной слабо структурированной и трудно формализуемой узкой предметной области и способная предлагать и объяснять пользователю разумные решения. Экспертная система состоит из базы знаний, механизма логического вывода и подсистемы объяснений.

База знаний – семантическая модель, описывающая предметную область и позволяющая отвечать на такие вопросы из этой предметной области, ответы на которые в явном виде не присутствуют в базе. База знаний является основным компонентом интеллектуальных и экспертных систем.

Для хранения баз знаний в современных экспертных системах используются либо промышленные СУБД и специализированное промежуточное ПО, либо специализированное ПО.

В настоящем курсе основное внимание уделяется проектированию БД и организации хранения в них структурированной информации. Проектирование и создание баз знаний будет подробно рассматриваться в других курсах, связанных с изучением интеллектуальных систем.

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 36–75.

# Модели БД

2.1 Обзор ранних (дореляционных) СУБД

2.2 Системы, основанные на инвертированных списках

2.3 Иерархическая модель

2.4 Сетевая модель

2.5 Основные достоинства и недостатки ранних СУБД

## Обзор ранних (дореляционных) СУБД

Рассмотрим системы, исторически предшествовавшие реляционным, что необходимо для правильного понимания причин повсеместного перехода к реляционным системам. Кроме того, внутренняя организация реляционных систем во многом основана на использовании методов ранних систем. И, наконец, некоторое знание в области ранних систем будет полезно для понимания путей развития постреляционных СУБД.

Ограничимся рассмотрением только общих подходов к организации трех типов ранних систем, а именно, систем, основанных на инвертированных списках, иерархических и сетевых систем управления базами данных. Не будем касаться особенностей каких-либо конкретных систем, поскольку это привело бы к изложению многих технических деталей. Детали можно найти в соответствующей литературе.

Рассмотрим некоторые наиболее общие характеристики ранних систем:

1. Эти системы активно использовались в течение многих лет, дольше, чем используется многие из реляционных СУБД. На самом деле некоторые из ранних систем используются даже в наше время, накоплены громадные базы данных, и одной из актуальных проблем информационных систем является использование этих систем совместно с современными системами.
2. Все ранние системы не основывались на каких-либо абстрактных моделях. Понятие модели данных фактически вошло в обиход специалистов в области БД только вместе с реляционным подходом. Абстрактные представления ранних систем появились позже на основе анализа и выявления общих признаков у различных конкретных систем.
3. В ранних системах доступ к БД производился на уровне записей. Пользователи этих систем осуществляли явную навигацию в БД, используя языки программирования, расширенные функциями СУБД. Интерактивный доступ к БД поддерживался только путем создания соответствующих прикладных программ с собственным интерфейсом.
4. Навигационная природа ранних систем и доступ к данным на уровне записей заставляли пользователя самого производить всю оптимизацию доступа к БД, без какой-либо поддержки системы.
5. После появления реляционных систем большинство ранних систем было оснащено "реляционными" интерфейсами. Однако в большинстве случаев это не сделало их по‑настоящему реляционными системами, поскольку оставалась возможность манипулировать данными в естественном для них режиме.

## Системы, основанные на инвертированных списках

К числу наиболее известных и типичных представителей таких систем относятся Datacom/DB компании Applied Data Research, Inc. (ADR), ориентированная на использование на машинах основного класса фирмы IBM, и Adabas компании Software AG.

Организация доступа к данным на основе инвертированных списков используется практически во всех современных реляционных СУБД, но в этих системах пользователи не имеют непосредственного доступа к инвертированным спискам (индексам).

### **Структуры данных**

В базе данных, организованной с помощью инвертированных списков хранимые таблицы и пути доступа к ним видны пользователям. При этом:

1. Строки таблиц упорядочены системой в некоторой физической последовательности.
2. Физическая упорядоченность строк всех таблиц может определяться и для всей БД (так делается, например, в Datacom/DB).
3. Для каждой таблицы можно определить произвольное число ключей поиска, для которых строятся индексы. Эти индексы автоматически поддерживаются системой, но явно видны пользователям.

### **Манипулирование данными**

Поддерживаются два класса операторов:

1. Операторы, устанавливающие адрес записи, среди которых:
	1. прямые поисковые операторы (например, найти первую запись таблицы по некоторому пути доступа);
	2. операторы, находящие запись в терминах относительной позиции от предыдущей записи по некоторому пути доступа.
2. Операторы над адресуемыми записями

Типичный набор операторов:

1. Найти первую запись таблицы T в физическом порядке;
2. Найти первую запись таблицы T с заданным значением ключа поиска K;
3. Найти запись, следующую за записью с заданным адресом в заданном пути доступа;
4. Найти следующую запись таблицы T в порядке пути поиска с заданным значением K;
5. Найти первую запись таблицы T в порядке ключа поиска K cо значением ключевого поля, большим заданного значения K;
6. Выбрать запись с указанным адресом;
7. Обновить запись с указанным адресом;
8. Удалить запись с указанным адресом;
9. Включить запись в указанную таблицу; операция генерирует адрес записи.

### **Ограничения целостности**

Общие правила определения целостности БД отсутствуют. В некоторых системах поддерживаются ограничения уникальности значений некоторых полей, но в основном все возлагается на прикладную программу.

## Иерархическая модель

Типичным представителем (наиболее известным и распространенным) является Information Management System (IMS) фирмы IBM. Первая версия появилась в 1968 г. До сих пор поддерживается много баз данных, что создает существенные проблемы с переходом как на новую технологию БД, так и на новую технику.

### **Иерархические структуры данных**

Иерархическая БД состоит из упорядоченного набора деревьев; более точно, из упорядоченного набора нескольких экземпляров одного типа дерева.

Тип дерева (рис. 2.1) состоит из одного "корневого" типа записи и упорядоченного набора из нуля или более типов поддеревьев (каждое из которых является некоторым типом дерева). Тип дерева в целом представляет собой иерархически организованный набор типов записи.

рис. 2.1 Пример типа дерева (схемы иерархической БД)

**Группа**

|  |  |  |
| --- | --- | --- |
| Гр\_номер | Гр\_название | Гр\_год\_поступления |

**Куратор**

|  |  |  |
| --- | --- | --- |
| К\_номер | К\_фамилия | К\_телефон |

**Студенты**

|  |  |  |
| --- | --- | --- |
| Ст\_номер | Ст\_фамилия | Ст\_город |

Здесь (рис. 2.1) Группа является предком для Куратора и Студенты, а Куратор и Студенты – потомки Группа. Между типами записи поддерживаются связи.

База данных с такой схемой могла бы выглядеть следующим образом (рис. 2.2):

рис. 2.2 Один экземпляр дерева.

**Группа**

|  |  |  |
| --- | --- | --- |
| 1 | А-98-51 | 1998 |

**Куратор**

|  |  |  |
| --- | --- | --- |
| 2 | Иванов | 6-39-45 |

**Студенты**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | Иванов | Желтые Воды |
| 3 | Петров | Кривой Рог |  |

Все экземпляры данного типа потомка с общим экземпляром типа предка называются близнецами. Для БД определен полный порядок обхода – сверху-вниз, слева-направо.

В IMS использовалась оригинальная и нестандартная терминология: "сегмент" вместо "запись", а под "записью БД" понималось все дерево сегментов.

### **Манипулирование данными**

Примерами типичных операторов манипулирования иерархически организованными данными могут быть следующие:

1. Найти указанное дерево БД (например, отдел 310);
2. Перейти от одного дерева к другому;
3. Перейти от одной записи к другой внутри дерева (например, от отдела - к первому сотруднику);
4. Перейти от одной записи к другой в порядке обхода иерархии;
5. Вставить новую запись в указанную позицию;
6. Удалить текущую запись.

### **Ограничения целостности**

Автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: никакой потомок не может существовать без своего родителя. Заметим, что аналогичное поддержание целостности по ссылкам между записями, не входящими в одну иерархию, не поддерживается (примером такой "внешней" ссылки может быть содержимое поля Каф\_Номер в экземпляре типа записи Куратор).

В иерархических системах поддерживалась некоторая форма представлений БД на основе ограничения иерархии. Примером представления приведенной выше БД может быть иерархия, изображенная на рис. 2.3.

рис. 2.3 Пример представления иерархической БД.

**Группа**

|  |  |  |
| --- | --- | --- |
| Гр\_номер | Гр\_название | Гр\_год\_поступления |

**Куратор**

|  |  |  |
| --- | --- | --- |
| К\_номер | К\_фамилия | К\_телефон |

## Сетевая модель

Типичным представителем является Integrated Database Management System (IDMS) компании Cullinet Software, Inc., предназначенная для использования на машинах основного класса фирмы IBM под управлением большинства операционных систем. Архитектура системы основана на предложениях Data Base Task Group (DBTG) Комитета по языкам программирования Conference on Data Systems Languages (CODASYL), организации, ответственной за определение языка программирования Кобол. Отчет DBTG был опубликован в 1971г., а в 70-х годах появилось несколько систем, среди которых IDMS.

### **Сетевые структуры данных**

Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков.

Сетевая БД состоит из набора экземпляров каждого типа записи и набора экземпляров каждого типа связи (рис. 2.4).

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи L с типом записи предка P и типом записи потомка C должны выполняться следующие два условия:

1. Каждый экземпляр типа P является предком только в одном экземпляре L;
2. Каждый экземпляр C является потомком не более, чем в одном экземпляре L.

На формирование типов связи не накладываются особые ограничения; возможны, например, следующие ситуации:

1. Тип записи потомка в одном типе связи L1 может быть типом записи предка в другом типе связи L2 (как в иерархии).
2. Данный тип записи P может быть типом записи предка в любом числе типов связи.
3. Данный тип записи P может быть типом записи потомка в любом числе типов связи.
4. Может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; и если L1 и L2 – два типа связи с одним и тем же типом записи предка P и одним и тем же типом записи потомка C, то правила, по которым образуется родство, в разных связях могут различаться.
5. Типы записи X и Y могут быть предком и потомком в одной связи и потомком и предком – в другой.
6. Предок и потомок могут быть одного типа записи.

рис. 2.4 Простой пример сетевой схемы БД.

Группа

Студенты

Куратор

Обучаются в группе

Состоит из студентов

Курируется куратором

### **Манипулирование данными**

Примерный набор операций может быть следующим:

1. Найти конкретную запись в наборе однотипных записей (инженера Сидорова);
2. Перейти от предка к первому потомку по некоторой связи (к первому сотруднику отдела 310);
3. Перейти к следующему потомку в некоторой связи (от Сидорова к Иванову);
4. Перейти от потомка к предку по некоторой связи (найти отдел Сидорова);
5. Создать новую запись;
6. Уничтожить запись;
7. Модифицировать запись;
8. Включить в связь;
9. Исключить из связи;
10. Переставить в другую связь и т.д.

### Ограничения целостности

В принципе их поддержание не требуется, но иногда требуется целостности по ссылкам (как в иерархической модели).

## Основные достоинства и недостатки ранних СУБД

Сильные места ранних СУБД:

1. Развитые средства управления данными во внешней памяти на низком уровне;
2. Возможность построения вручную эффективных прикладных систем;
3. Возможность экономии памяти за счет разделения подобъектов (в сетевых системах).

Недостатки:

1. Слишком сложно пользоваться;
2. Фактически необходимы знания о физической организации;
3. Прикладные системы зависят от этой организации;
4. Их логика перегружена деталями организации доступа к БД.

**Литература:**

1. Сергей Кузнецов, “Основы современных баз данных”. Центр Информационных Технологий, http://www.citforum.ru/database/osbd/contents.shtml

# Реляционная модель и ее характеристики. Целостность в реляционной модели

3.1 Представление информации в реляционных БД

3.2 Домены

3.3 Отношения. Свойства и виды отношений

3.4 Целостность реляционных данных

3.5 Потенциальные и первичные ключи

3.6 Внешние ключи

3.7 Ссылочная целостность

3.8 Значения NULL и поддержка ссылочной целостности

## Представление информации в реляционных БД

Реляционный подход является наиболее распространенным в настоящее время, хотя наряду с общепризнанными достоинствами обладает и рядом недостатков. К числу достоинств реляционного подхода можно отнести:

1. наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными;
2. наличие простого и в то же время мощного математического аппарата, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации баз данных;
3. возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

Однако реляционные системы далеко не сразу получили широкое распространение. В то время, как основные теоретические результаты в этой области были получены еще в 70-х, и тогда же появились первые прототипы реляционных СУБД, долгое время считалось невозможным добиться эффективной реализации таких систем. Однако отмеченные выше преимущества и постепенное накопление методов и алгоритмов организации реляционных баз данных и управления ими привели к тому, что уже в середине 80-х годов реляционные системы практически вытеснили с мирового рынка ранние СУБД.

В настоящее время основным предметом критики реляционных СУБД является не их недостаточная эффективность, а следующие недостатки:

1. присущая этим системам некоторая ограниченность (прямое следствие простоты) при использовании в так называемых нетрадиционных областях (наиболее распространенными примерами являются системы автоматизации проектирования), в которых требуются предельно сложные структуры данных.
2. невозможность адекватного отражения семантики предметной области. Другими словами, возможности представления знаний о семантической специфике предметной области в реляционных системах очень ограничены. Современные исследования в области постреляционных систем главным образом посвящены именно устранению этих недостатков.

В реляционной модели рассматриваются три аспекта данных:

1. структура данных (объекты данных);
2. целостность данных;
3. обработка данных (операторы).

Рассмотрим специальную терминологию, применяемую в рамках аспекта "структура данных" (рис. 3.1).

рис. 3.1 Отношение.

SNo

Name

City

|  |  |  |
| --- | --- | --- |
| SNo | Name | City |
| 1 | Иванов | Желтые Воды |
| 2 | Петров | Пятихатки |
| 3 | Сидоров | Кривой Рог |
| 4 | Петренко | Киев |
| 5 | Стрельцов | Львов |

Киев

Львов

и др.

Кортежи

Домены

Атрибуты

Степень

Отношение

Первичный ключ

Кардинальное число

## Домены

Домен является наименьшей семантической единицей данных, которая предполагается отдельным значением данных (таким как номер студента, фамилия студента и т.д.). Такие значения называют скалярами. Скалярные значения представляют собой наименьшую семантическую единицу данных в том смысле, что они являются атомарными: в реляционной модели у них отсутствует внутренняя структура. Следует обратить внимание, что отсутствие внутренней структуры при рассмотрении в реляционной модели вовсе не значит, что внутренняя структура отсутствует вообще. Например, название города имеет внутреннюю структуру (оно состоит из последовательности букв) однако, разложив название по буквам мы потеряем значение. Значение станет понятным лишь в том случае, если буквы сложены вместе и в правильной последовательности.

Таким образом, домен – именованное множество скалярных значений одного типа. Например, домен городов это множество всех возможных названий городов. Домены являются общими совокупностями значений из которых берутся реальные значения атрибутов.

Следует обратить внимание, что обычно в любой момент времени в домене будут значения, не являющиеся значением ни одного из атрибутов, соответствующих этому домену.

Основное значение доменов в том, что домены ограничивают сравнения. Сравнение будет иметь смысл для атрибутов, основанных на одном и том же домене. Например, можно сравнивать числовой код студента и оценку, полученную студентом на экзамене - и то и другое - целые числа, однако такое сравнение будет лишено смысла.

Домены, прежде всего, имеют концептуальную природу. Они могут быть или не быть явно сохранены в базе данных как реальные наборы значений (фактически, в большинстве случаев они не сохраняются), но они должны быть, по крайней мере, определены в рамках определений базы данных. Тогда каждое определение атрибута должно включать ссылку на соответствующий домен, таким образом, системе будет известно, какие атрибуты можно сравнивать, а какие - нет.

## Отношения. Свойства и виды отношений

Вокруг понятия "отношение" сложилась некоторая двусмысленность из-за отсутствия четкого разграничения между переменными отношений и значениями отношений. Переменная отношения – это обычная переменная, такая же, как и в языках программирования, т.е. именованный объект, значение которого может изменяться со временем. А значение этой переменной в любой момент будет значением отношения.

Отношение R, определенное на множестве доменов D1, D2, …, Dn (не обязательно различных), содержит две части – заголовок и тело:

1. заголовок содержит фиксированное множество атрибутов или, точнее, пар <имя‑атрибута : имя‑домена>:
2. {<A1:D1>, <A2:D2>, …, <An:Dn>},
причем каждый атрибут Aj соответствует одному и только одному из лежащих в основе доменов Dj (j=1,2, …, n). Все имена атрибутов A1, A2, …, An разные.
3. Тело состоит из множества кортежей. Каждый кортеж, в свою очередь содержит множество пар <имя‑атрибута : значение‑атрибута>:
{<A1:Vi1>, <A2:Vi2>, …, <An:Vin>},
(i=1, 2, …, m, где m - количество кортежей в этом множестве). В каждом таком кортеже есть одна такая пара <имя‑атрибута : значение‑атрибута>, т.е. <Aj:Vij>, для каждого атрибута Aj в заголовке. Для любой такой пары <Aj:Vij> Vij является значением из уникального домена Dj, который связан с атрибутом Aj.

Значения m и n называются соответственно кардинальным числом и степенью отношения R.

### Свойства отношений

1. В отношении отсутствуют одинаковые кортежи.

Это свойство следует из того факта - что тело отношения – это математическое множество (кортежей), а множества в математике по-определению не содержат одинаковых элементов. Это свойство служит иллюстрацией различия между отношением и таблицей т.к. таблица, в общем случае может содержать одинаковые строки.

Важным следствием того, что не существует одинаковых строк является то, что всегда существует первичный ключ. Так как кортежи уникальны, по крайней мере комбинация всех кортежей будет обладать свойством уникальности, а значит, при необходимости, может служить первичным ключом.

1. Кортежи не упорядочены сверху вниз.

Это свойство также следует из того, что тело отношения – это математическое множество, а простые множества в математике не упорядочены. Второе свойство также служит иллюстрацией того факта, что отношение и таблица – не одно и тоже так как строки таблицы упорядочены сверху вниз, в то время, как кортежи отношения – нет.

1. атрибуты не упорядочены слева на право.

И это свойство следует из того, что заголовок отношения определен как множество атрибутов. Аналогично второму свойству, можно заметить отличия между таблицей и отношением – в таблице столбцы упорядочены слева на право.

1. все значения атрибутов атомарные.

Это свойство является следствием того, что все лежащие в основе домены содержат только атомарные значения. Отношение, удовлетворяющее этому условию, называется нормализованным, или представленном в первой нормальной форме. Это означает, что с точки зрения реляционной модели все отношения нормализованы.

### Виды отношений

Определим некоторые виды отношений, встречающиеся в реляционных системах.

1. Именованное отношение – это переменная отношения, определенная в СУБД посредством операторов создания отношений.
2. Базовым отношением называется именованное отношение, которое не является производным (т.е. базовое отношения является автономным).
3. Производным отношением называется отношение, определенное (посредством реляционного выражения) через другие именованные выражения и, в конечном счете, через базовые отношения.
4. Выражаемое отношение – отношение, которое можно получить из набора именованных отношений посредством некоторого реляционного выражения. Множество всех выражаемых отношений – это в точности множество всех базовых отношений и всех производных отношений.
5. Представление – это именованное производное отношение. Представления, как и базовые отношения являются переменными отношений. Представления виртуальны – они представлены в системе исключительно через определение в терминах других именованных отношений.
6. Снимки – это именованные производные отношения, в отличии от представлений являются реальными и представлены в системе не только в виде определений в терминах других именованных отношений, но и своими данными.
7. Результатом запроса называется неименованное производное отношение, служащее результатом некоторого определенного запроса.
8. Промежуточным результатом называется неименованное производное отношение, являющееся результатом некоторого реляционного выражения, вложенного в другое, большее выражение.
9. Хранимое отношение – отношение, которое поддерживается непосредственно в физической памяти.

Каждое отношение в реляционной модели имеет некоторую интерпретацию, причем пользователи должны ее знать для эффективного использования БД.

Например: студент с номером SNo имеет фамилию SurName и проживает в городе City. При этом нет двух студентов с одинаковыми номерами.

Формально, подобное утверждение называют предикатом, или функцией значения истинности. В последнем примере – функцией трех аргументов. Подстановка значений этих аргументов эквивалентна вызову функции и приводит к получению выражения, называемого высказыванием, которое может быть либо истинным либо ложным.

Предикат данного отношения составляет критерий возможности обновления для этого отношения. Для того, чтобы система могла определить допустимость обновления данного отношения, ей должен быть известен предикат этого отношения. СУБД, чтобы определить допустимость обновления отношения использует определенные для данного отношения правила целостности.

## Целостность реляционных данных

Большинство БД подчиняются множеству правил целостности. В любой момент времени любая база данных содержит некую определенную конфигурацию значений данных, и предполагается, что эта конфигурация отображает действительность – т.е. является моделью части реального мира. Просто определенная конфигурация значений не имеет смысла, если значения в этой конфигурации не представляют определенного состояния реального мира. Исходя из сказанного выше, определение базы данных нуждается в расширении, включающем правила целостности, назначение которых в том, чтобы информировать СУБД о разного рода ограничениях реального мира. В реляционной модели есть два общих особых правил целостности. Эти правила относятся к потенциальным (и первичным) ключам и ко внешним ключам.

## Потенциальные и первичные ключи

Пусть R – некоторое отношение. Тогда потенциальный ключ, скажем, K для R – это подмножество множества атрибутов R, обладающее следующими свойствами:

1. Свойство уникальности – нет двух разных кортежей в отношении R с одинаковым значением K.
2. Свойство не избыточности – никакое из подмножеств K не обладает свойством уникальности.

Следует обратить внимание, что данное выше определение потенциального ключа относится к самому отношению (т.е. к значениям отношения), а не к переменным отношения.

При рассмотрении потенциальных ключей необходимо заметить следующее:

1. На практике большинство отношений имеют только один потенциальный ключ, хотя в общем случае их может быть несколько.
2. Потенциальные ключи определены как множества атрибутов. Потенциальный ключ, состоящий из нескольких атрибутов называется составным, состоящий из одного атрибута – простым.
3. Понятие не избыточности. Если определить "потенциальный ключ" не являющийся не избыточным, системе не будет известно об этом и она не сможет обеспечить должным образом соответствующее ограничение целостности. Например: в отношении с данными о студентах можно определить избыточный потенциальный ключ, состоящий из уникального кода студента StNo и названия города, в котором он проживает City. В таком случае система не сможет соблюдать ограничение, обеспечивающее уникальность номера студента в глобальном смысле, вместо этого будет выполняться более слабое ограничение, обеспечивающее уникальность номера студента в пределах города.

Причина важности потенциальных ключей заключается в том, что они обеспечивают основной механизм адресации на уровне кортежей в реляционной системе. Единственный способ точно указать на какой-либо кортеж – это указать точное значение потенциального ключа.

Отношение может иметь более одного потенциального ключа. В этом случае в реляционной модели, один из них выбирается в качестве первичного в базовом отношении, а остальные потенциальные ключи, если они есть будут называться альтернативными ключами.

Если множество потенциальных ключей содержит более одного элемента, выбор первичного ключа, в общем случае, осуществляется произвольно.

## Внешние ключи

Пусть R2 – базовое отношение. Тогда внешний ключ – FK в отношении R2 – это подмножество множества атрибутов R2 такое, что:

1. существует базовое отношение R1 (R1 и R2 не обязательно различны) с потенциальным ключом CK.
2. каждое значение FK в текущем значении R2 всегда совпадает со значением CK некоторого кортежа в текущем значении R1.

Следует отметить, что:

1. Внешние ключи как и потенциальные определены как множества атрибутов.
2. Данный внешний ключ будет составным (т.е. будет состоять из более чем одного атрибута) тогда и только тогда, когда соответствующий потенциальный ключ также будет составным. Он будет простым тогда и только тогда, когда соответствующий потенциальный ключ также будет простым.
3. Каждый атрибут, входящий в данный внешний ключ, должен быть определен на том же домене, что и соответствующий атрибут соответствующего потенциального ключа.
4. Для внешнего ключа не требуется, чтобы он был компонентом первичного ключа или какого-либо потенциального ключа в содержащем его отношении.
5. Значение внешнего ключа представлено ссылкой к кортежу, содержащему соответствующее значение потенциального ключа (ссылочный кортеж или целевой кортеж). Проблема обеспечения того, что БД не включает никаких неверных значений внешних ключей, известна как проблема ссылочной целостности. Ограничение, по которому значения данного внешнего ключа должны быть адекватны значениям соответствующего потенциального ключа называют ссылочным ограничением. Отношение, содержащее внешний ключ называют ссылающимся отношением, а отношение которое содержит соответствующий потенциальный ключ – ссылочным или целевым отношением.
6. Связи, существующие в базе данных отображают с помощью ссылочных диаграмм. Groups ← Students → Cities
7. Отношение может быть и ссылочным и ссылающимся одновременно Students → Groups → Department
8. В определении внешних ключей сказано, что R1 и R2 не обязательно различны. Т.е. отношение может ссылаться само на себя. Такие отношения иногда называют самоссылающимися.
9. Самоссылающиеся отношения представляют собой частный случай ситуации, когда могут возникнуть ссылочные циклы, которые можно отобразить на ссылочной диаграмме следующим образом: Rn → R(n-1) → R(n-2) → … → R2 → R1 →Rn

## Ссылочная целостность

База данных не должна содержать несогласованных значений внешних ключей. Несогласованное значение внешнего ключа – это такое значение внешнего ключа, для которого не существует отвечающего ему значения соответствующего потенциального ключа в соответствующем целевом отношении.

Понятия внешний ключ и ссылочная целостность определены в терминах друг друга.

### Правила внешних ключей

Сформулированное ранее правило целостности, выражено исключительно в терминах состояний базы данных. Любое состояние базы данных, не удовлетворяющее этому правилу некорректно. Для того, чтобы избежать некорректных состояний для каждого внешнего ключа необходимо установить правила, на основании которых СУБД будет действовать при попытке удалить объект ссылки внешнего ключа или обновить потенциальный ключ, на который ссылается внешний ключ. Для каждого из этих случаев можно предусмотреть по меньшей мере две возможности.

1. При попытке удалить объект ссылки внешнего ключа:
	1. Ограничить – приостановить операцию удаления, до момента, когда не будет существовать ссылающихся объектов.
	2. Каскадировать – каскадировать операцию удаления, удалив соответствующие ссылающиеся объекты.
2. При попытке обновить потенциальный ключ, на который ссылается внешний ключ:
	1. Ограничить – приостановить операцию обновления, до момента, когда не будет существовать ссылающихся объектов.
	2. Каскадировать – каскадировать операцию обновления, обновив значение внешнего ключа в соответствующих ссылающихся объектах.

При каскадных обновлениях удавлениях и обновлениях следует учесть возможность наличия ссылочных циклов, которые могут привести к усложнению процедуры модификации БД.

## Значения NULL и поддержка ссылочной целостности

Значения NULL используются для обозначения факта отсутствия информации. Например: дата рождения человека может быть неизвестна. При этом следует учесть, что значения NULL отличаются от числового значения 0 или символьных пробелов. Значение NULL вообще не является реальным значением. Для данного атрибута может быть разрешено или запрещено содержать значения NULL.

Возможность присутствия в отношении значений NULL приводит к необходимости формирования правила целостности объектов. Целостность объектов – ни один элемент первичного ключа не может содержать значения NULL.

Правило ссылочной целостности также должно быть расширено с учетом возможности присутствия значений NULL.

Возможность присутствия значений NULL приводит к возникновению ряда трудноразрешимых проблем и осуждается некоторыми исследователями (например, К. Дж. Дейтом в книге [1]).

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 79–134.

# Реляционная алгебра

4.1 Понятие реляционной алгебры

4.2 Замкнутость в реляционной алгебре

4.3 Традиционные операции над множествами

4.4 Свойства основных операций реляционной алгебры

4.5 Специальные реляционные операции

## Понятие реляционной алгебры

Основным компонентом той части реляционной модели, которая касается операторов, является так называемая реляционная алгебра, которая в основном состоит из набора операторов, использующих отношения в качестве операндов и возвращающих отношения в качестве результата.

Реляционная алгебра, определенная Коддом в, состоит из восьми операторов, составляющих две группы, по четыре оператора в каждой:

1. Традиционные операции над множествами: объединение, пересечение, вычитание и декартово произведение (модифицированные с учетом того, что их операндами являются отношения, а не произвольные множества).
2. Специальные реляционные операции: выборка, проекция, соединение и деление.

## Замкнутость в реляционной алгебре

Результат каждой операции над отношением (или реляционной операции) также является отношением. Это реляционное свойство называется свойством замкнутости. Поскольку результат любой операции имеет тот же тип, что и исходные объекты (отношения), то результат одной операции может использоваться в качестве исходных данных для другой. Таким образом, имеется возможность, например, взять или проекцию от объединения, или соединение от двух выборок, или объединение соединения и пересечения и т.д.

Другими словами, можно записывать вложенные выражения, т.е. выражения, в которых операнды сами представлены выражениями вместо простых имен отношений.

Если рассматривать замкнутость более строго, каждая реляционная операция должна быть определена таким образом, чтобы выдавать результат с надлежащим заголовком (т.е. с соответствующим набором необходимых имен атрибутов). Причина такого требования к результирующим отношениям заключается в необходимости иметь возможность обращаться к именам атрибутов в последующих операциях, например в дальнейших операциях, расположенных на более глубоких уровнях вложенного выражения. Другими словами, необходим такой набор правил наследования имен атрибутов, встроенный в алгебру, чтобы можно было предсказывать имена атрибутов на выходе произвольной реляционной операции, зная имена атрибутов на входе этой операции.

## Традиционные операции над множествами

### Объединение

Объединение в реляционной алгебре не полностью совпадает с математическим объединением, вернее, это особая форма объединения, в которой требуется, чтобы два исходных отношения были совместимо по типу.

Будем говорить, что два отношения совместимы по типу, если у них идентичные заголовки, а точнее,

1. если каждое из них имеет одно и то же множество имен атрибутов (следовательно, заметьте, они заведомо должны иметь одну и ту же степень);
2. если соответствующие атрибуты (т.е. атрибуты с теми же самыми именами в двух отношениях) определены на одном и том же домене.

Операции объединения, пересечения и вычитания требуют от операндов совместимости по типу.

Объединением двух совместимых по типу отношений А и В (A UNION B) называется отношение с тем же заголовком, как и в отношениях А и В, и с телом, состоящим из множества всех кортежей, принадлежащих А или В или обоим отношениям.

Пример операции объединения отношений приведен на рис. 4.1 – рис. 4.2.

|  |  |  |
| --- | --- | --- |
| A |  | B |
| CityNo | CityName | RgNo |  | CityNo | CityName | RgNo |
| 1 | Желтые Воды | 1 |  | 2 | Кривой Рог | 1 |
| 2 | Кривой Рог | 1 |  | 3 | Пятихатки | 1 |
| 3 | Пятихатки | 1 |  | 4 | Львов | 2 |

рис. 4.1 Исходные отношения

|  |
| --- |
| A UNION B |
| CityNo | CityName | RgNo |
| 1 | Желтые Воды | 1 |
| 2 | Кривой Рог | 1 |
| 3 | Пятихатки | 1 |
| 4 | Львов | 2 |

рис. 4.2 Результат объединения отношений A и B.

### Пересечение

Пересечением двух совместимых по типу отношений А и В (A INTERSECT B) называется отношение с тем же заголовком, как и в отношениях А и В, и с телом, состоящим из множества всех кортежей, принадлежащих одновременно обоим отношениям A и B.

Пример операции пересечения отношений приведен на рис. 4.1 и рис. 4.3.

|  |
| --- |
| A INTERSECT B |
| CityNo | CityName | RgNo |
| 2 | Кривой Рог | 1 |
| 3 | Пятихатки | 1 |

рис. 4.3 Результат операции пересечения отношений A и B.

### Вычитание

Вычитанием двух совместимых по типу отношений А и В (A MINUS B) называется отношение с тем же заголовком, как и в отношениях А и В, и с телом, состоящим из множества всех кортежей, принадлежащих отношению A и не принадлежащих отношению B.

Пример операции вычитания отношений приведен на рис. 4.1 и рис. 4.4.

|  |  |  |
| --- | --- | --- |
| A MINUS B |  | B MINUS A |
| CityNo | CityName | RgNo |  | CityNo | CityName | RgNo |
| 1 | Желтые Воды | 1 |  | 4 | Львов | 2 |

рис. 4.4 Результат операции вычитания отношений A минус B и B минус A.

### Произведение

В математике декартово произведение (или для краткости произведение) двух множеств является множеством всех таких упорядоченных пар элементов, что первый элемент в каждой паре берется из первого множества, а второй элемент в каждой паре берется из второго множества. Следовательно, декартово произведение двух отношений, должно быть множеством упорядоченных пар кортежей. Но опять-таки необходимо сохранить свойство замкнутости; иначе говоря, результат должен содержать кортежи, а не упорядоченные пары кортежей.

Декартово произведение двух отношений А и В (A TIMES B), где А и В не имеют общих имен атрибутов, определяется как отношение с заголовком, который представляет собой сцепление двух заголовков исходных отношений А и В, и телом, состоящим из множества всех кортежей t, таких, что t представляет собой сцепление кортежа a, принадлежащего отношению А, и кортежа b, принадлежащего отношению В. Кардинальное число результата равняется произведению кардинальных чисел исходных отношений А и В, а степень равняется сумме их степеней. Пример операции декартова произведения представлена на рис. 4.5

|  |  |  |
| --- | --- | --- |
| A |  | B |
| CityNo | CityName | A\_RgNo |  | B\_RgNo | RgName |
| 1 | Желтые Воды | 1 |  | 1 | Днепропетровская |
| 2 | Кривой Рог | 1 |  | 2 | Львовская |
| 3 | Пятихатки | 1 |  |  |  |

|  |
| --- |
| A TIMES B |
| CityNo | CityName | A\_RgNo | B\_RgNo | RgName |
| 1 | Желтые Воды | 1 | 1 | Днепропетровская |
| 1 | Желтые Воды | 1 | 2 | Львовская |
| 2 | Кривой Рог | 1 | 1 | Днепропетровская |
| 2 | Кривой Рог | 1 | 2 | Львовская |
| 3 | Пятихатки | 1 | 1 | Днепропетровская |
| 3 | Пятихатки | 1 | 2 | Львовская |

рис. 4.5 Результат операции декартово произведение отношений A и B.

Явное использование операции декартова произведения требуется только для очень сложных запросов. Эта операция включена в реляционную алгебру главным образом по концептуальным соображениям. Декартово произведение требуется как промежуточный шаг при определении операции Θ-соединения которая используется довольно часто.

## Свойства основных операций реляционной алгебры

Операции объединения, пересечения и декартова произведения (но не вычитания) обладают свойствами ассоциативности и коммутативности.

Пусть А, В и С – произвольные реляционные выражения (дающие совместимые по типу отношения). Тогда операция объединения:

(A UNION В) UNION С

Эквивалентна операции:

А UNION (В UNION С) (свойство ассоциативности), а .операция объединения:

А UNION B эквивалентна операции:

В UNION A (свойство коммутативности). Аналогично свойства ассоциативности и коммутативности определяются для остальных операций.

Свойство ассоциативности позволяет записывать последовательные операторы объединения (пересечения и декартова произведения) без использования круглых скобок; таким образом, выражение из предыдущего примера можно однозначно упростить:

A UNION В UNION С.

## Специальные реляционные операции

### Выборка

Выборка – это сокращенное название Θ-выборки, где Θ обозначает любой скалярный оператор сравнения (=, ≠, >, ≥, ≤, <). Θ-выборкой из отношения A по атрибутам Х и Y (в этом порядке)

A WHERE X Θ Y называется отношение, имеющее тот же заголовок, что и отношение А, и тело, содержащее множество всех кортежей отношения А, для которых проверка условия X Θ Y дает значение истина. Атрибуты X и Y должны быть определены на одном и том же домене, а оператор должен иметь смысл для этого домена.

На рис. 4.6приведен пример операции выборки.

|  |
| --- |
| A |
| CityNo | CityName | RgNo |
| 1 | Желтые Воды | 1 |
| 2 | Кривой Рог | 1 |
| 3 | Пятихатки | 1 |
| 4 | Львов | 2 |

|  |
| --- |
| A WHERE RgNo = 1 |
| CityNo | CityName | RgNo |
| 1 | Желтые Воды | 1 |
| 2 | Кривой Рог | 1 |
| 3 | Пятихатки | 1 |

рис. 4.6 Исходное отношение A и результат операции выборки кортежей из отношения A по условию RGNo = 1.

### Проекция

Проекцией отношения А по атрибутам X, Y,..., Z, где каждый из атрибутов принадлежит отношению А

A [ X, Y, …, Z ] называется отношение с заголовком {X, Y,..., Z} и телом, содержащим множество всех кортежей {Х:х, Y:y,..., Z:z}, таких, для которых в отношении А значение атрибута Х равно х, атрибута Y равно y, ..., атрибута Z равно z.

Таким образом, с помощью оператора проекции получено "вертикальное" подмножество данного отношения, т.е. подмножество, получаемое исключением всех атрибутов, не указанных в списке атрибутов, и последующим исключением дублирующих кортежей (рис. 4.7).

A

|  |  |  |
| --- | --- | --- |
| CityNo | CityName | RgNo |
| 1 | Желтые Воды | 1 |
| 2 | Кривой Рог | 1 |
| 3 | Пятихатки | 1 |
| 4 | Львов | 2 |

A [CityName]

|  |
| --- |
| CityName |
| Желтые Воды |
| Кривой Рог |
| Пятихатки |
| Львов |

рис. 4.7 Исходное отношение A и результат операции проекции отношения A по атрибуту CityName.

Никакой атрибут не может быть указан в списке атрибутов более одного раза. Синтаксис позволяет опустить список атрибутов совсем (вместе с квадратными скобками). Действие такой операции эквивалентно указанию списка всех атрибутов исходного отношения, т.е. такая операция представляет собой тождественную проекцию. Другими словами, имя отношения является допустимым реляционным выражением. Проекция вида R[ ], т.е. такая, в которой список атрибутов не пропущен, но пустой, тоже допустима. Она представляет собой "нулевую" проекцию.

### Естественное соединение

Операция соединения имеет несколько разновидностей. Однако наиболее важным, без сомнения, является естественное соединение, причем настолько, что для обозначения исключительно естественного соединения почти постоянно используется общий термин "соединение".

Пусть отношения А и В имеют заголовки {Xl, X2, ..., Xm, Y1, Y2, ..., Yn} и {Yl, Y2, ..., Yn, Zl, Z2, ..., Zp} соответственно; т.е. атрибуты Yl, Y2, ..., Yn (и только они) ‑ общие для двух отношений; Х1, Х2, ... ,Хm – остальные атрибуты отношения A; Zl, Z2, ..., Zp ‑ остальные атрибуты отношения В. Предположим также, что соответствующие атрибуты (т.е. атрибуты с одинаковыми именами) определены на одном и том же домене. Рассматривать выражения {X1, Х2, ..., Хm}, {Y1, Y2, ..., Yn} и {Zl, Z2, ..., Zp} как три составных атрибута X, Y и Z соответственно. Тогда естественным соединением отношений А и В (A JOIN B) называется отношение с заголовком {X, Y, Z} и телом, содержащим множество всех кортежей {Х:х, Y:y, Z:z}, таких, для которых в отношении А значение атрибута X равно х, а атрибута Y равно у, и в отношении В значение атрибута Y равно у, а атрибута Z равно z.

Пример операции естественного соединения приведен на рис. 4.8.

|  |  |  |
| --- | --- | --- |
| A |  | B |
| CityNo | CityName | RgNo |  | RgNo | RgName |
| 1 | Желтые Воды | 1 |  | 1 | Днепропетровская |
| 2 | Кривой Рог | 1 |  | 2 | Львовская |
| 3 | Пятихатки | 1 |  |  |  |

|  |
| --- |
| A JOIN B |
| CityNo | CityName | A\_RgNo | RgName |
| 1 | Желтые Воды | 1 | Днепропетровская |
| 2 | Кривой Рог | 1 | Днепропетровская |
| 3 | Пятихатки | 1 | Днепропетровская |

рис. 4.8 Исходные отношения A и B и результат операции естественного соединения.

Соединение обладает свойствами ассоциативности и коммутативности. Отсюда следует, что выражения:

(A JOIN В) JOIN С и

A JOIN (В JOIN С) могут быть однозначно упрощены к следующему:

A JOIN В JOIN С

Кроме того, выражения:

A JOIN Ви

В JOIN A эквивалентны.

### Θ-соединение

Операция Θ-соединения предназначается для тех случаев (сравнительно редких, но, тем не менее, встречающихся), когда нам нужно соединить вместе два отношения на основе некоторых условий, отличных от эквивалентности. Пусть отношения А и В не имеют общих имен атрибутов (как и в рассмотренной выше операции декартова произведения) и Θ определяется так же, как и в операции выборки. Тогда Θ-соединением отношения А по атрибуту Х с отношением В по атрибуту Y называется результат вычисления выражения

(A TIMES В) WHERE X Θ Y

Θ-соединение, таким образом, это отношение с тем же заголовком, что и при декартовом произведении отношений A и B, и с телом, содержащим множество кортежей, принадлежащих этому декартову произведению и вычисление условия XΘY дает значение истина для этого кортежа. Атрибуты Х и У должны быть определены на одном и том же домене, а операция должна иметь смысл для этого домена.

### Деление

Пусть отношения А и В имеют заголовки:

{X1, X2,..., Xm, Y1, Y2, ..., Yn} и {Y1, Y2, ..., Yn} соответственно, т.е. атрибуты Y1, Y2,..., Yn — общие для двух отношений, и отношение A имеет дополнительные атрибуты X1, Х2, ... ,Хm, а отношение В не имеет дополнительных атрибутов. (Отношения А и В представляют соответственно делимое и делитель.) Предположим также, что соответствующие атрибуты (т.е. атрибуты с одинаковыми именами) определены на одном и том же домене. Пусть теперь выражения {X1, Х2, ..., Хm} и {Y1, Y2, ..., Yn} обозначают два составных атрибута Х и Y соответственно. Тогда делением отношений А на В (A DIVIDEBY B) называется отношение с заголовком {X} и телом, содержащим множество всех кортежей {X:x}, таких что существует кортеж {Х:х, Y:y}, который принадлежит отношению A для всех кортежей {Y:y}, принадлежащих отношению В. Нестрого это можно сформулировать так: результат содержит такие X-значения из отношения А, для которых соответствующие Y-значения (из А) включают все Y-значения из отношения В.

Пример операции деления приведен на рис. 4.9. Отношение M является проекцией отношения Marks, а отношение S – проекцией отношения Subjects. Результат операции деления M DIVIDE BY S фактически содержит номера студентов, которые сдавали дисциплины с номерами 1 и 5.

|  |  |  |
| --- | --- | --- |
| M |  | S |
| StNo | SubjNo | SubjNo |
| 1 | 1 | 1 |
| 1 | 5 | 5 |
| 2 | 1 |  |
| 2 | 5 |
| 3 | 1 |
| 3 | 5 |
| 4 | 1 |
| 5 | 1 |
|  |
| M DIVIDE BY S |
| StNo |  |
| 1 |
| 2 |
| 3 |

рис. 4.9. Пример операции деления.

Восемь операторов Кодда не представляют минимального набора операторов, так как не все из них примитивны, их можно определить в терминах других операторов. Например, естественное соединение – это проекция выборки произведения. Фактически три операции из этого набора, а именно соединение, пересечение и деление, можно определить через остальные пять. Операции выборки, проекции, произведения, объединения и вычитания можно рассматривать как примитивные в том смысле, что ни одна из них не выражается через другие. Поэтому минимальный набор будет состоять из этих пяти примитивных операций. Однако на практике другие три операции (в особенности соединение) настолько часто используются, что имеет смысл обеспечить их непосредственную поддержку, несмотря на то что они не примитивны.

Многие авторы предлагали новые алгебраические операторы после определения Коддом первоначальных восьми. Рассмотрим два таких оператора – EXTEND (расширение) и SUMMARIZE (подведение итогов), которые удачно дополняют основной набор восьми операторов.

### Операция расширения

С помощью операции расширения из определенного отношения создается новое отношение (по крайней мере концептуально), которое похоже на начальное, но содержит дополнительный атрибут, значения которого получены посредством некоторых скалярных вычислений. На рис. 4.10 показаны исходное отношение и результат операции расширения:

EXTEND GROUPS ADD (2002-EnterYear) AS COURCE

|  |  |  |
| --- | --- | --- |
| GROUPS |  | Результат операции расширения |
| GrNo | EnterYear | GrName |  | GrNo | EnterYear | GrName | Cource |
| 1 | 1998 | А–98–51 |  | 1 | 1998 | А–98–51 | 2 |
| 2 | 1999 | Б–99–51 |  | 2 | 1999 | Б–99–51 | 1 |
| 3 | 1998 | Б–98–51 |  | 3 | 1998 | Б–98–51 | 2 |

рис. 4.10 Пример выполнения операции расширения.

### Операция подведения итогов

Пусть А1,А2,... ,An – отдельные атрибуты отношения А. Результатом операции подведения итогов

SUMMARIZE A BY (A1, A2, … An) ADD exp AS Z (которая является выражением, а не командой или оператором) будет отношение с заголовком {А1, А2, ..., An, Z} и с телом, содержащим все такие кортежи, которые являются кортежами проекции отношения А по атрибутам Al, A2, ..., An, расширенного значением для нового атрибута Z. Новое значение Z подсчитывается вычислением итогового выражения ехр по всем кортежам отношения А, которые имеют те же самые значения для атрибутов А1, А2, ..., Аn, что и кортеж t. Список атрибутов А1, А2, ..., Аn не должен включать атрибут с именем Z, а выражение ехр не должно ссылаться на атрибут Z. Кардинальное число результата равно кардинальному числу проекции отношения А по атрибутам Al, A2, ..., An, а степень результата равна степени такой проекции плюс единица.

### Операторы обновления

Реляционная модель (точнее, ее часть, связанная с операторами) кроме реляционной алгебры может включать также операции реляционного присвоения. Такие операции имеют следующий синтаксис:

TARGET := SOURCE где source и target— реляционные выражения, представляющие совместимые по типу отношения. Вычисленное значение source присваивается отношению target, заменяя его старое значение.

В реляционных системах также существуют операции вставки INSERT, удаления DELETE и модификации UPDATE.

Оператор вставки имеет следующий вид:

INSERT source INTO target где source и target – это реляционные выражения, представляющие совместимые по типу отношения (на практике отношение target является просто именованным отношением). Значение отношения source вычисляется, и все кортежи результата вставляются в отношение target.

Оператор обновления имеет следующий вид:

UPDATE target attribute1:=scalar\_expression, attribute2:=scalar\_expression, …, attributeN:=scalar\_expression

где target – реляционное выражение, а каждый атрибут attribute принадлежит отношению, которое является результатом вычисления указанного выражения. Все кортежи в результирующем отношении обновляются в соответствии с указанными операторами attribute2:=scalar‑expression

На практике выражение target часто будет просто ограничивающим условием для некоторого именованного отношения.

Оператор удаления имеет следующий вид:

DELETE target

где target – реляционное выражение; все кортежи в результирующем отношении удаляются.

Как и в случае с оператором обновления, выражение target часто будет просто ограничивающим условием для некоторого именованного отношения.

### Реляционные сравнения

Реляционное сравнение имеет следующий вид:

Expression Θ expression где expression –это выражения реляционной алгебры, представляющие совместимые по типу отношения, а Θ – один из следующих операторов сравнения:

= (равно)

≠ (не равно)

≤ (подмножество)

< (собственное подмножество)

≥ (надмножество)

> (собственное надмножество).

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 135–171.

# Вопросы проектирования БД

5.1 Понятие проектирования БД

5.2 Функциональные зависимости

5.3 Тривиальные и нетривиальные зависимости

5.4 Замыкание множества зависимостей и правила вывода Армстронга

5.5 Неприводимое множество зависимостей

5.6 Нормальные формы – основные понятия

5.7 Декомпозиция без потерь и функциональные зависимости

5.8 Диаграммы функциональных зависимостей

## Понятие проектирования БД

В этой лекции речь пойдет о проектировании реляционной базы данных. В общем проблема формулируется следующим образом: как в некоторой базе данных для заданного набора данных выбрать подходящую логическую структуру? Иначе говоря, нужно решить вопрос, какие базовые отношения и с какими атрибутами следует задать.

Следует отметить следующие особенности.

1. Следует заметить, что речь здесь пойдет о логическом, а не физическом макете.
2. Физический макет может рассматриваться как отдельная сопутствующая часть. Иначе говоря, для "правильного" составления макета базы данных следует прежде всего создать логический (т.е. реляционный) макет, а затем в виде отдельного шага отобразить этот логический макет на некоторые физические структуры, поддерживаемые СУБД.
3. Физический макет по определению является специфическим для данной СУБД. Логический макет, наоборот, совершенно независим от СУБД, и для его реализации могут быть использованы строгие теоретические принципы.

К сожалению, на практике часто случается так, что реализация макета на физическом уровне может оказывать существенное обратное влияние на логический макет. Иначе говоря, в таком случае для достижения компромисса следует выполнить несколько циклов проектирования "логический макет – физический макет".

Следует также подчеркнуть, что проектирование базы данных скорее искусство, чем просто наука. Конечно, существуют некоторые научные принципы составления таких макетов, однако при проектировании базы данных возникает множество других проблем, которые не всегда можно охватить этими принципами. В результате теоретики и практики в области создания баз данных разработали большое число методологий проектирования. Среди них есть как достаточно точные и строгие, так и не очень, однако все они специализированные и предназначены для решения именно той проблемы, которая считалась неразрешимой к моменту создания данной конкретной методики. Иными словами, они были задуманы для поиска такого логического макета, который был бы, бесспорно, лучшим в данной ситуации.

Необходимо отметить некоторые допущения, используемые в дальнейшем изложении материала:

1. Проектирование базы данных — это не единственное условие получения правильной организации структуры данных, помимо этого ключевым условием является также целостность данных.
2. Далее в макет рассматривается независимо от приложения. Иначе говоря, интерес представляют сами данные, а не то, как они используются. Независимость от приложения желательна потому, что обычно в момент проектирования базы данных не известны все возможные способы использования данных. Таким образом, необходимо, чтобы макет был стабильным, т.е. оставался работоспособным даже при возникновении в приложении новых (т.е. неизвестных на момент создания исходного макета) требований к данным.

Следуя этим допущениям нужно создать концептуальную схему, т.е. абстрактный логический макет, не зависящий от аппаратного обеспечения, операционной системы, СУБД, языка программирования, пользователя и т.д.

## Функциональные зависимости

Для демонстрации основных идей данного раздела используется несколько измененная версия отношения Students из учебной БД, которое в дополнение к обычным атрибутам StNo, GrNo, StName, CityNo будет содержать также атрибут RgNo, представляющий город соответствующего поставщика. Во избежание путаницы далее это измененное отношение будет называться SR. В виде таблицы оно представлено на рис. 5.1.

|  |
| --- |
| SR |
| StNo | GrNo | StName | CityNo | RgNo |
| 1 | 1 | Иванов | 3 | 1 |
| 2 | 1 | Петров | 3 | 1 |
| 3 | 1 | Сидоров | 3 | 1 |
| 4 | 2 | Стрельцов | 1 | 1 |
| 5 | 2 | Кузнецов | 4 | 2 |

рис. 5.1 Данные отношения SR.

Следует четко различать:

1. значение этого отношения (т.е. значение переменной отношения) в определенный момент времени;
2. набор всех возможных значений, которые данное отношение (переменная) может принимать в различные моменты времени.

При рассмотрении переменных отношения, например базовых отношений, интерес представляют не столько функциональные зависимости для определенного в некоторый момент времени значения, сколько функциональные зависимости, выполняющиеся для всех возможных значений данной переменной.

Ниже приведено определение концепции функциональной зависимости для второго пункта.

Пусть R является переменной отношения, а X и Y – произвольными подмножествами множества атрибутов отношения R. Тогда Y функционально зависимо от X, что в символическом виде записывается как

X→Y
(и читается либо как "X функционально определяет Y ", либо как "X стрелка Y "), тогда и только тогда, когда для любого допустимого значения отношения R каждое значение Х связано в точности с одним значением Y.

Иначе говоря, для любого допустимого значения отношения R, когда бы два кортежа отношения R ни совпадали по значению X, они также совпадают и по значению Y. Далее термин "функциональная зависимость" будет использоваться в последнем безотносительном ко времени смысле (за исключением особо оговоренных случаев).

Например, в случае отношения SR функциональная зависимость

{StNo}→{GrNo}

выполняется для всех возможных значений SR, поскольку в любой момент времени данному студенту соответствует в точности одна группа; таким образом, любые два кортежа отношения SR в один и тот же момент времени и с одним и тем же номером студента должны соответствовать одной и той же группе. Практически утверждение, что данная функциональная зависимость выполняется "всегда" (т.е. для всех возможных значений SR), является ограничением целостности для отношения SR, поскольку при этом накладываются определенные ограничения на все допустимые значения.

Ниже перечислено несколько безотносительных ко времени функциональных зависимостей для переменной отношения SR

{StNo}→{GrNo}

{StNo}→{StName}

{StNo}→{CityNo}

{StNo}→{RgNo}

{StNo}→{GrNo, StName}

{StNo}→{GrNo, StName, CityNo, RgNo}

{StNo, GrNo}→ {StName}

и другие.

Левая и правая стороны символической записи функциональной зависимости иногда называются детерминантом и зависимой частью соответственно. Как говорится в определении, детерминант и зависимая часть являются множествами атрибутов. Когда множество содержит только один атрибут, он называется одноэлементным множеством, скобки опускаются и символическая запись принимает вид:

StNo→GrNo

Обратите внимание, в частности, на функциональные зависимости, которые выполняются для таблицы на рис. 5.1, но не выполняются "всегда":

GrNo→CityNo

Следует отметить, что если X является потенциальным ключом отношения R, например X является первичным ключом, то все атрибуты Y отношения R должны быть обязательно функционально зависимы от Х (это следует из определения потенциального ключа). В обычном отношении студентов Students, например, необходимо, чтобы выполнялась зависимость

StNo→{GrNo, StName, CityNo}

Фактически, если отношение R удовлетворяет функциональной зависимости A → B и A не является потенциальным ключом, то R будет характеризоваться некоторой избыточностью. Например, в случае отношения SR сведения о том, что каждый данный город находится в данной области, будут повторяться много раз (это хорошо видно на рис. 5.1).

На практике важно сократить множество ФЗ до компактных размеров, поскольку, функциональные зависимости являются ограничениями целостности, поэтому при каждом обновлении данных в СУБД все они должны быть проверены.

## Тривиальные и нетривиальные зависимости

Очевидным способом сокращения размера множества ФЗ было бы исключение тривиальных зависимостей, т.е. таких, которые не могут не выполняться. В качестве примера приведем тривиальную ФЗ для отношения SR:

{StNo, GrNo} →  {StNo}

Фактически ФЗ тривиальна тогда и только тогда, когда правая часть символической записи данной зависимости является подмножеством (не обязательно собственным подмножеством) левой части.

## Замыкание множества зависимостей и правила вывода Армстронга

Некоторые функциональные зависимости обозначают другие функциональные зависимости. Например, функциональная зависимость:

{StNo}→{GrNo, StName}
подразумевает следующие функциональные зависимости:

{StNo}→{GrNo}

{StNo}→{StName}

Множество всех ФЗ, которые задаются данным множеством функциональных зависимостей S, называется замыканием S и обозначается символом S+.

Поскольку функциональные зависимости являются ограничениями целостности, которые должны быть проверены СУБД, желательно для заданного множества ФЗ S найти такое множество ФЗ которое было бы гораздо меньшего размера, чем множество S, причем каждая ФЗ множества S могла бы быть заменена функциональной зависимостью множества T. Для решения этой задачи следует найти способ вычисления S+ на основе S.

Первой попыткой решить эту проблему была статья Армстронга (Armstrong), в которой представлен набор правил вывода функциональных зависимостей на основе заданных (эти правила также называются аксиомами Армстронга).

Правила вывода Армстронга. Пусть в перечисленных ниже правилах А, В, С и D – произвольные подмножества множества атрибутов заданного отношения R, а символическая запись АВ означает объединение А и В.

1. Рефлексивность: если В является подмножеством А, то А→В.
2. Дополнение: если А→В, то АС→ВС.
3. Транзитивность: если А→В и В→С, то А→С.

Каждое из этих правил может быть непосредственно доказано на основе определения функциональной зависимости (первое из них — это всего лишь определение тривиальной зависимости). Более того, эти правила являются полными в том смысле, что для заданного множества функциональных зависимостей S минимальный набор ФЗ, которые подразумевают все зависимости S, может быть выведен из S на основе этих правил. Они также являются исчерпывающими, поскольку никакие дополнительные ФЗ не могут быть выведены (т.е. ФЗ, которые не подразумеваются зависимостями множества S). Иначе говоря, эти правила могут быть использованы для получения замыкания S+.

Из трех описанных выше правил для упрощения задачи практического вычисления замыкания S можно вывести несколько других правил. (Пусть D – это другое произвольное подмножество множества атрибутов R.).

1. Самоопределение: А→А.
2. Декомпозиция: если А→ВС, то А→В и А→C.
3. Объединение: если A→В и А→С, то А→ВС.
4. Композиция: если А→В и С→D, то AC→BD.
5. Если А→В и С→D, то А ∪ (С – В) →BD (где символ "∪" обозначает объединение множеств, а символ "–" – их разность).Это правило называют также теоремой всеобщего объединения.

Например. Пусть дано некоторое отношение R с атрибутами А, В, С, D, Е, F и следующими ФЗ:

А→{ВС}

В→Е

{CD}→{EF}

Далее символами, записанными подряд, например ВС, будем обозначать множество, состоящее из атрибутов В и С, а не объединение В и С.

Этому примеру можно придать более конкретный смысл, а именно: А – номер сотрудника, В – номер отдела, С – номер руководителя (начальника) данного сотрудника, D – номер проекта, возглавляемого данным руководителем (уникальный для каждого отдельно взятого руководителя), Е – название отдела, F – доля времени, уделяемая данным руководителем заданному проекту.

Показать, что зависимость AD → F выполняется для отношения R и таким образом принадлежит замыканию данного множества ФЗ.

1. А→ВС (дано);
2. А→С (из 1 согласно декомпозиции);
3. AD→CD (из 2 согласно дополнению);
4. CD→EF (дано);
5. AD→EF (из 3 и 4 согласно транзитивности);
6. AD→F (из 5 согласно декомпозиции).

## Неприводимое множество зависимостей

Пусть S1 и S2 являются двумя множествами ФЗ. Если любая ФЗ, которая является зависимостью множества S1, также является зависимостью множества S2, т.е. если S1+ является подмножеством S2+ то S2 называется покрытием для S1. Это значит, что если накладываемые в СУБД ограничения представлены зависимостями множества S2, то в этой СУБД также наложены ограничения на основе зависимостей множества S1.

Далее, если S2 является покрытием для S1, а S1 – покрытием для S2, т.е. если S1+=S2+ , то S1 и S2 эквивалентны. Ясно, что если S1 и S2 эквивалентны и наложенные в СУБД ограничения представлены зависимостями множества S2, то эти ограничения также могут быть представлены зависимостями множества S1, верно также и обратное утверждение.

Множество ФЗ называется неприводимым тогда и только тогда, когда выполняются перечисленные ниже свойства.

1. Правая часть (зависимая часть) каждой ФЗ множества S содержит только один атрибут (т.е. является одноэлементным множеством).
2. Левая часть (детерминант) каждой ФЗ множества S является неприводимой, т.е. ни один атрибут не может быть опущен из детерминанта без изменения замыкания S+ (без конвертирования множества S в некоторое множество, не эквивалентное множеству S). В таком случае ФЗ является неприводимой слева
3. Ни одна функциональная зависимость в S не может быть опущена из S без изменения замыкания S+ (т.е. без конвертирования множества S в некоторое множество, не эквивалентное множеству S).

Множество зависимостей t, которое неприводимо и эквивалентно некоторому другому множеству зависимостей S, называется неприводимым покрытием множества S. Таким образом, с тем же успехом в системе вместо исходного множества зависимостей S может быть использовано его неприводимое покрытие t. Однако следует отметить, что для заданного множества ФЗ не всегда существует уникальное неприводимое покрытие.

## Нормальные формы – основные понятия

Процесс дальнейшей нормализации, который далее будет упоминаться просто как нормализация, основывается на концепции нормальных форм. Говорят, что отношение находится в некоторой нормальной форме, если удовлетворяет заданному набору условий. Например, отношение находится в первой нормальной форме, или сокращенно в 1 НФ, тогда и только тогда, когда оно содержит только скалярные значения.

Отсюда следует, что каждое нормализованное отношение находится в первой нормальной форме. Иначе говоря, термины "нормализованное" и "1НФ" означают одно и то же. Однако следует отметить, что термин "нормализованное" часто используется для обозначения нормальной формы более высокого уровня, хотя такое обозначение не очень корректно.

рис. 5.2 Нормальные формы отношений.

Пространство отношений (нормализованных и ненормализованных)

Отношения в 1НФ (нормализованные отношения)

Отношения во 2НФ

Отношения в 3НФ

Отношения в НФБК

Отношения в 4 НФ

Отношения в 5 НФ

На рис. 5.2 показано несколько нормальных форм, которые определены к настоящему времени.

Все нормализованные отношения находятся в 1НФ. Некоторые отношения 1НФ находятся также в 2НФ и некоторые отношения 2НФ находятся также в ЗНФ. Мотивом для введения дополнительных определений было то, что вторая нормальная форма "более желательна", чем первая, а третья, в свою очередь, "более желательна", чем вторая. Таким образом, при проектировании базы данных вообще следует использовать отношения не только в первой или во второй, но также и в третьей форме.

Процедуру нормализации можно охарактеризовать как последовательное приведение данного набора отношений к некоторой более желательной форме. Эта процедура обратима, т.е. всегда можно использовать ее результат (например, множество отношений, находящихся в ЗНФ) для обратного преобразования (в исходное отношение, находящееся в 2НФ). Возможность выполнения обратного преобразования является весьма важной характеристикой, поскольку означает, что в процессе нормализации информация не утрачивается

## Декомпозиция без потерь и функциональные зависимости

Как упоминалось ранее, процедура нормализации включает разбиение, или декомпозицию данного отношения на другие отношения, причем декомпозиция должна быть обратимой, т.е. выполняться без потерь информации. Иначе говоря, интерес представляет только те операции, которые выполняются без потерь информации. Вопрос о том, происходит ли утрата информации при декомпозиции, тесно связан с концепцией функциональной зависимости.

Рассмотрим отношение Students из учебной базы данных, с атрибутами {StNo, GrNo, StName, CityNo} (рис. 5.3).

|  |
| --- |
| Students |
| StNo | GrNo | StName | CityNo |
| 1 | 1 | Иванов | 1 |
| 2 | 1 | Петров | 3 |

рис. 5.3 Отношение Students.

Ниже приведены две возможные декомпозиции этого отношения (рис. 5.4).

|  |  |  |
| --- | --- | --- |
| 1. SGN |  | SC |
| StNo | GrNo | StName |  | StNo | CityNo |
| 1 | 1 | Иванов |  | 1 | 1 |
| 2 | 1 | Петров |  | 2 | 3 |

|  |  |  |
| --- | --- | --- |
| 2. SGN |  | GC |
| StNo | GrNo | StName |  | GrNo | CityNo |
| 1 | 1 | Иванов |  | 1 | 1 |
| 2 | 1 | Петров |  | 1 | 3 |

рис. 5.4 Возможные декомпозиции отношения Students.

В первом случае информация не утрачивается, поскольку отношения SGN и SC все еще содержат информацию о том, что Иванов живет в городе с кодом 1, Петров – 2. Соединение этих отношений позволяет восстановить исходное отношение Students, иначе говоря первая декомпозиция является декомпозицией без потерь.

Во втором случае информация о городе, в котором проживает студент утрачивается, поскольку студенты, учащиеся в группе с кодом 1 живут в разных городах и зная код группы невозможно однозначно определить код города в котором проживает студент.

Следует отметить, что процесс, который до сих пор назывался “декомпозицией”, на самом деле называется проецированием, т.е. каждое из показанных выше отношений SGN, SC и GC – в действительности являются проекциями исходного отношения Students. Таким образом оператор декомпозиции в процедуре нормализации фактически является оператором проецирования.

Исходное отношение при этом равно соединению его проекций. Для выполнения декомпозиции отношения без потерь необходимо знать, какие условия должны быть соблюдены для того, чтобы при обратном соединении гарантировать получение исходного отношения. Ответ на этот вопрос содержится в теореме Хеза.

Теорема Хеза. Пусть R{A, B, C} является отношением, где A, B, C – атрибуты этого отношения. Если R удовлетворяет зависимости A→B, то отношение R равно соединению его проекций {A, B} и {A, C}.

## Диаграммы функциональных зависимостей

Некоторое неприводимое множество зависимостей отношения R можно представить в виде диаграммы функциональных зависимостей (диаграммы ФЗ).

На рис. 5.5 и рис. 5.6 показаны диаграммы ФЗ для некоторых отношений из учебной БД.

рис. 5.5 Диаграмма ФЗ для таблицы Students.

GrNo

StName

CityNo

StNo

GrNo

StName

CityNo

StNo

GrNo

StName

CityNo

StNo

рис. 5.6 Диаграмма ФЗ для таблицы Marks.

StNo

SubjNo

DocNo

Mark

Как видно из рис. 5.5 и рис. 5.6 каждая стрелка начинается с потенциального ключа (на самом деле с первичного ключа) соответствующего отношения. По определению стрелки должны начинаться с каждого потенциального ключа, поскольку одному значению такого ключа всегда соответствует, по крайней мере, еще одно какое-то значение. Некоторые стрелки следовало бы исключить ввиду того, что очи вызывают определенные трудности, но стрелки, начинающиеся с потенциальных ключей, никогда не могут быть исключены.

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 259–276.

# Проектирование БД. Нормальные формы отношений

6.1 Первая нормальная форма. Возможные недостатки отношения в 1НФ

6.2 Вторая нормальная форма. Возможные недостатки отношения во 2НФ

6.3 Третья нормальная форма. Возможные недостатки отношения в 3НФ

6.4 Нормальная форма Бойса-Кодда

## Первая нормальная форма. Возможные недостатки отношения в 1НФ

Для простоты изложения предполагается, что каждое отношение имеет в точности один потенциальный ключ, который является первичным ключом. Это допущение подтверждают предлагаемые здесь доказательства. Далее в этой главе будет рассмотрен случай, когда отношение имеет два или более потенциальных ключа.

Отношение находится в первой нормальной форме тогда и только тогда, когда все используемые домены содержат только скалярные значения.

В этом определении всего лишь утверждается, что любое нормализованное отношение находится в 1НФ. Однако отношение, которое находится только в 1 НФ (т.е. не находится ни во второй, ни в третьей нормальной форме) обладает структурой, по некоторым причинам не совсем желательной. Для иллюстрации этого факта допустим, что информация о студентах и оценках содержится не в 2-х отношениях Students и Marks а в одном, назовем его SM.

SM{StNo, CityNo, GrNo, SubjNo, DocNo, Mark}

PRIMARY KEY (StNo, GrNo, SubjNo, DocNo).

Диаграмма функциональных зависимостей этого отношения будет выглядеть как показано на рис. 6.1.

рис. 6.1 Диаграмма функциональных зависимостей отношения SM

StNo

SubjNo

DocNo

Mark

GrNo

StName

CityNo

Обратите внимание, что диаграммы ФЗ отношения SM “сложнее”, чем диаграммы ФЗ отношений Students и Marks, из которых оно образовано. В диаграммах ФЗ отношений Students и Marks все стрелки начинаются только от первичных ключей, тогда как в диаграмме ФЗ отношения SM появляются дополнительные стрелки. Ниже приведена таблица данных для отношения SM (рис. 6.2).

|  |
| --- |
| SM |
| StNo | StName | GrNo | CityNo | SubjNo | DocNo | Mark |
| 1 | Иванов | 1 | 1 | 1 | 127 | 5 |
| 1 | Иванов | 1 | 1 | 5 | 128 | 4 |
| 2 | Петров | 1 | 3 | 1 | 127 | 3 |

рис. 6.2 Данные отношения SM.

Несмотря на то, что отношение SM, как и Students и Marks находится в 1й НФ, оно очевидно обладает избыточностью, поскольку, например, в каждом кортеже для студента Иванова указан его номер “1”, код его группы – “1” и код города, в котором он проживает – “1”. Аналогичная ситуация с другими студентами.

Избыточность в отношении SM приводит к разным аномалиям обновления, получившим такое название по историческим причинам, т.е. к трудностям при выполнении операций обновления типа INSERT (вставка), DELETE (удаление) и UPDATE (обновление). Для начала рассмотрим избыточность типа студент—код города студента, соответствующую функциональной зависимости StNo →CityNo, и перечисленные ниже проблемы с операциями обновления.

Операция вставки (INSERT). Нельзя вставить данные о студенте, проживающем в некотором городе, не указывая хотя бы одну, полученную этим студентом, оценку. Действительно, в таблице SM не показан студент Сидоров из г. Пятихатки потому, что до тех пор, пока этот студент не получит оценку по какому либо предмету, для него не задано значение первичного ключа.

Операция удаления (DELETE). Если удалить единственный кортеж отношения SM для некоторого студента, будет удалена не только информация о соответствующей оценке, но и информация о студенте и городе, в котором он проживает. Например, если в отношении SM удалить кортеж со значением Петров атрибута StName, будет утрачена вся информация об этом студенте.

Замечание. В действительности проблема заключается в том, что в отношении SM содержится очень много совместной информации, поэтому при удалении некоторого кортежа приходится удалять слишком иного другой информации. А точнее, отношение SM содержит информацию о студентах и об оценках. Таким образом, удаление информации об оценке вызывает также удаление информации о студенте. Для решения этой проблемы нужно разделить информацию на несколько частей, т.е. разместить информацию о студентах в одном отношении, а об оценках – в другом. Таким образом, неформально процедуру нормализации можно охарактеризовать как процедуру разбиения логически несвязанной информации по отдельным отношениям.

Операция модификации (UPDATE). Фамилия студента и код города, в котором он проживает повторяется в отношении SM множество раз, и это приводит к возникновению проблем при обновлении. Если студент меняет фамилию или переезжает в другой город, то возникает проблема, связанная либо с поиском в отношении SM всех кортежей, в которых присутствует информация об этом студенте, либо с получением несовместимого результата (в одном кортеже городом проживания студента будет один город, а в другом кортеже, городом проживания этого студента, будет другой город).

Для решения проблемы избыточности, которая характерна для отношения SM достаточно заменить его двумя другими:

Students{StNo, GrNo, StName, CityNo}

и

Marks{StNo, SubjNo, DocNo, Mark}

Важно отметить, что переработанная таким образом структура позволяет преодолеть все перечисленные ранее проблемы, связанные с операциями обновления.

Операция вставки (INSERT). Теперь с помощью вставки соответствующего кортежа в отношение Students можно включить информацию о студенте и городе, в котором он проживает, даже если он в настоящий момент не получил не одной оценки.

Операция удаления (DELETE). Теперь можно исключить информацию об оценке, удаляя соответствующий кортеж из отношения Marks, при этом информация о студенте и городе, в котором он проживает, не утрачивается.

Операция модификации (UPDATE). В переработанной структуре фамилия студента и информация о городе, в котором он проживает, появляется всего один раз, поскольку существует только один кортеж для данного студента в отношении Students (атрибут StNo является первичным ключом для такого отношения). Иначе говоря, избыточность данных StNo-StName-StCity устранена. Благодаря этому теперь можно один раз изменить в соответствующем кортеже отношения Students название города для какого-либо студента.

## Вторая нормальная форма. Возможные недостатки отношения во 2НФ

Определим 2НФ при условии, что существует только один потенциальный ключ, который является первичным ключом.

Отношение находится во второй нормальной форме тогда и только тогда, когда оно находится в первой нормальной форме и каждый неключевой атрибут неприводимо зависим от первичного ключа.

Оба отношения, Students и Marks находятся во второй нормальной форме с первичными ключами StNp и {StNo, SubjNo, DocNo} соответственно, а отношение SM не находится в ней. Всякое отношение, которое находится в 1НФ и не находится в 2НФ, всегда можно свести к эквивалентному набору отношений, находящихся в 2НФ.

Рассмотрим другой пример. Предположим, информация о коде города, названии города и области, в которой этот город расположен находятся в одной таблице CNR{CityNo, CityName, RgNo, RgName} (рис. 6.3).

|  |
| --- |
| CNR |
| CityNo | CityName | RgNo | RgName |
| 1 | Желтые Воды | 1 | Днепропетровская |
| 2 | Кривой Рог | 1 | Днепропетровская |
| 3 | Пятихатки | 1 | Днепропетровская |
| 4 | Львов | 2 | Львовская |

рис. 6.3 Данные отношения CNR.

Диаграмма ФЗ отношения CNR выглядит следующим образом – рис. 6.4.

рис. 6.4 Функциональные зависимости в отношении CNR.

CityNo

CityName

RgNo

RgName

Как видно из рис. 6.3, это диаграмма ФЗ “сложнее” диаграмм ФЗ отношений Cities и Regions. Несмотря на то, что отношение CNR находится во 2НФ, оно обладает некоторой избыточностью, связанной с наличием транзитивной ФЗ между атрибутами CityNo и RgName. Транзитивная зависимость приводит к следующим аномалиям обновления.

Операция вставки (INSERT). Нельзя включить данные о некоторой области, например, нельзя указать, что существует Львовская область, до тех пор пока не появиться запись о городе, находящемся в данной области, – например о Львове.

Операция удаления (DELETE). При удалении из отношения CNR последнего кортежа для некоторого города будет удалена не только информация о данном городе, но также информация о том, в какой области этот город находился. Например, при удалении из отношения CNR кортежа для города Львов будет утрачена информация о Львовской области.

Замечание. Вновь причиной этих неприятностей является совместная информация: отношение CNR содержит информацию о городах вместе с информацией об областях. Для разрешения этой ситуации следует поступить так, как и раньше, т.е. ''разобрать" всю эту информацию и перенести в одно отношение сведения об областях, а в другое – сведения о городах.

Операция модификации (UPDATE). В отношении CNR код и название области для каждого города повторяется несколько раз (поэтому оно характеризуется некоторой избыточностью). Таким образом, при изменении кода области возникнет либо проблема необходимости поиска в отношении CNR всех кортежей для этой области (для внесения соответствующих изменений), либо проблема получения несовместимого результата.

Для решения этих проблем необходимо заменить отношение CNR двумя проекциями:

Cities{CityNo, CityName, RgNo}

Regions{RgNo, RgName}

Переработанная таким образом структура отношений позволит преодолеть все описанные проблемы с операциями обновления.

## Третья нормальная форма. Возможные недостатки отношения в 3НФ

Отношение находится в третьей нормальной форме тогда и только тогда, когда оно находится во второй нормальной форме и каждый неключевой атрибут нетранзитивно зависит от первичного ключа. (Под "нетранзитивной зависимостью" подразумевается отсутствие какой-либо взаимной зависимости в изложенном выше смысле.)

Отношения Cities и Regions находятся в третьей нормальной форме. Таким образом вторым этапом нормализации является создание проекций для исключения транзитивных зависимостей.

### Сохранение зависимости

В процессе приведения отношений часто возникают ситуации, когда данное отношение может быть подвергнуто операции декомпозиции разными способами. Рассмотрим снова приведенное выше отношение CNR с функциональными зависимостями CityNo→CityName, CityNo→RgNo, CityNo→RgNаме, RgNo→RgName и, следовательно, транзитивной зависимостью CityNo→RgName (на рис. 6.5 транзитивная зависимость показана пунктирной стрелкой).

рис. 6.5 Функциональные зависимости в отношении CNR

CityNo

CityName

RgNo

RgName

Выше отмечалось, что аномалии обновления, которые сопровождают отношение CNR, можно преодолеть с помощью декомпозиции с заменой этого отношения двумя проекциями в ЗНФ.

Cities{CityNo, CityName, RgNo} и Regions{RgNo, RgName}

Назовем эту декомпозицию просто "декомпозицией №1", имея в виду, что для нее существует альтернативная "декомпозиция №2":

Cities{CityNo, CityName, RgNo} и Regions{CityNo, RgName}

При этом обе проекции Cities одинаковы как для №1, так и для №2. Декомпозиция №2 происходит также без потери информации, а обе ее проекции находятся в ЗНФ. Однако по некоторым причинам декомпозиция №2 менее желательна, чем декомпозиция №1. Например, после выполнения декомпозиции №2 все еще невозможно вставить информацию о том, что некоторая область имеет определенный код, без указания города, который находится в этой области.

Рассмотрим этот пример подробнее. Прежде всего заметим, что зависимости проекций в декомпозиции №1 отмечены сплошными стрелками, тогда как одна, из зависимостей проекций декомпозиции №1 отмечена пунктирной стрелкой. В декомпозиции №1 две проекции независимы друг от друга в следующем смысле: обновления в каждой из проекций могут быть выполнены совершенно независимо друг от друга. (Конечно, за исключением ограничения целостности для Cities и Regions) Если такое обновление допустимо только в контексте данной проекции, т.е. не нарушается уникальность первичного ключа для этой проекции, то соединение этих двух проекций после обновления всегда будет равносильно отношению CNR (т.е. при соединении не будут нарушены ограничения, наложенные на ФЗ в отношении CNR). В декомпозиции №2, наоборот, обновление любой из двух проекций должно тщательно фиксироваться, чтобы гарантировать отсутствие нарушения зависимости RgNo→RgName (если два города находятся в одной и той же области, они должны иметь одинаковый код области). Иначе говоря, обе проекции декомпозиции №2 не являются независимыми одна от другой.

Основная проблема заключается в том, что в декомпозиции №2 функциональная зависимость RgNo→RgName становится ограничением между отношениями. (Следует отметить, что во многих современных программных продуктах это ограничение должно поддерживаться с помощью процедурной обработки.) В декомпозиции №1, наоборот, транзитивная зависимость SityNo→RgName является ограничением между отношениями, которое автоматически выполняется при задействовании двух ограничений внутри отношений: CityNo→RgNo и RgNo→RgName. Привести в действие эти ограничения достаточно просто за счет соответствующих ограничений, наложенных на уникальность первичных ключей.

Концепция независимых проекций, таким образом, обеспечивает критерий выбора одной из нескольких возможных декомпозиции. Декомпозиция с независимыми проекциями в приведенном выше общем смысле предпочтительнее той, в которой проекции зависимы. Риссанен (Rissanen) показал, что проекции R1 и R2 отношения R независимы в упомянутом выше смысле тогда и только тогда, когда:

1. каждая ФЗ в отношении R является логическим следствием функциональных зависимостей в проекциях R1 и R2;
2. общие атрибуты проекций R1 и R2 образуют потенциальный ключ, по крайней мере, для одной из них.

Отношение, которое не может быть подвергнуто декомпозиции с получением независимых проекций, называется атомарным. Однако это не значит, что любое неатомарное отношение может быть разбито на атомарные компоненты. Идея нормализации с декомпозицией на независимые проекции называется декомпозицией с сохранением зависимости.

## Нормальная форма Бойса-Кодда

В этом разделе опускается упрощающее допущение о том, что каждое отношение имеет только один потенциальный ключ (а именно первичный ключ), и рассматривается более общий случай. Оригинальное определение Кодда для ЗНФ не совсем подходит для отношений с перечисленными ниже условиями.

1. Отношение имеет два (или более) потенциальных ключа.
2. Два потенциальных ключа являются сложными.
3. Они перекрываются (т.е. имеют, по крайней мере, один общий атрибут).

Поэтому оригинальное определение ЗНФ было впоследствии заменено более строгим определением Бойса-Кодда (Boyce/Codd), для которого было принято отдельное название – нормальная форма Бойса-Кодда, НФБК. (На самом деле строгое определение "третьей" нормальной формы, эквивалентное определению нормальной формы Бойса-Кодда, было впервые дано Хезом (Heath) в 1971 году, и этой форме следовало бы дать название "нормальная форма Хеза".)

Замечание. Комбинация условий 1, 2 и 3 не часто встречается на практике, и для отношения без этих условий ЗНФ и НФБК эквивалентны.

Отношение находится в нормальной форме Бойса-Кодда тогда и только тогда, когда каждая нетривиальная и неприводимая слева ФЗ обладает потенциальным ключом в качестве детерминанта.

Менее формальное определение имеет другую формулировку: отношение находится в нормальной форме Бойса-Кодда тогда и только тогда, когда детерминанты являются потенциальными ключами.

Иначе говоря, на диаграмме ФЗ стрелки будут начинаться только с потенциальных ключей. Согласно данному определению никакие другие стрелки не допускаются.

Примером отношения, которое находится в НФБК может служить отношение Students, в которое добавлен атрибут IdCode – идентификационный код.

Students {StNo, IdCode, GrNo, StName, CityNo}

рис. 6.6 Диаграмма ФЗ расширенного отношения, Students, находящегося в НФБК.

StNo

IdCode

StName

GrNo

CityNo

В этом отношении детерминанты являются потенциальными ключами, а все стрелки начинаются с потенциальных ключей. Рассмотрим отношение, не находящееся в НФБК.

Предположим, что информация об идентификационных кодах студентов хранится в отношении Marks. Назовем модифицированное отношение MI {StNo, IdCode, SubjNo, DocNo, Mark} (рис. 6.7).

|  |
| --- |
| MI |
| StNo | IdCode | SubjNo | DocNo | Mark |
| 1 | 2895764537 | 1 | 127 | 5 |
| 1 | 2895764537 | 5 | 128 | 4 |
| 2 | 3094769520 | 1 | 127 | 3 |
| 2 | 3094769520 | 5 | 128 | 3 |
| 3 | 2984267527 | 1 | 127 | 5 |

рис. 6.7 Данные отношения MI.

В этом отношении присутствуют 2 потенциальных ключа {StNo, SubjNo, DocNo} и {IdCode, SubjNo, DocNo}. Отношение находится в 3-й НФ, но не находится в НФБК, так как содержит два детерминанта, которые не являются потенциальными ключами этого отношения (StNo и IdCode детерминанты, поскольку они определяют друг друга). Как видно, в отношении MI присутствует доля избыточности, которая имелась и в ранее рассмотренных отношениях (SM и CNR), поэтому оно характеризуется такими же аномалиями обновления. Для решения этой проблемы отношение MI следует разбить на две проекции:

SI {StNo, IdCode} и Marks {StNo, SubjNo, DocNo, Mark}

или другим способом

SI {StNo, IdCode} и Marks {IdCode, SubjNo, DocNo, Mark}

Т.о. присутствуют две, в одинаковой мере допустимые декомпозиции, причем все проекции отношения MI находятся в НФБК. Исходя из соображений здравого смысла первая декомпозиция лучше, поскольку в учебной БД для идентификации студента используется его код StNo.

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 279–301.

# Проектирование БД. Нормальные формы отношений (продолжение)

7.1 Многозначные зависимости

7.2 Четвертая нормальная форма

7.3 Зависимости соединения

7.4 Пятая нормальная форма

7.5 Итоговая схема процедуры нормализации

## Многозначные зависимости

Пусть дано ненормализованное отношение UCTX (т.е. отношение, которое не находится в 1НФ), содержащее информацию о курсах обучения, преподавателях и учебниках. Каждый кортеж такого отношения состоит из названия курса (Course), a также групп имен преподавателей (Teachers) и названий учебников (Texts) – на рис. 7.1 показаны два таких кортежа. Под этим подразумевается, что каждый курс может преподаваться любым преподавателем соответствующей группы с использованием всех указанных учебников. Предположим, что для заданного курса может существовать любое количество соответствующих преподавателей и соответствующих учебников. Более того, допустим, хотя это и не совсем реалистичное допущение, что преподаватели и рекомендуемые учебники совершенно независимы друг от друга. Это значит, что независимо от того, кто преподает данный курс, всегда используется один и тот же набор учебников. Наконец, допустим, что определенный преподаватель или определенный учебник могут быть связан с любым количеством курсов.

|  |
| --- |
| UCTX |
| COURSE | TEACHERS | TEXTS |
| Физика | проф. Ивановпроф. Петров | основы механикиоптика |
| Математика | проф. Иванов | основы механикидискретная математикатригонометрия |

рис. 7.1 Ненормализованное отношения UCTX

Преобразуем это отношение в эквивалентное нормализованное отношение. Следует заметить, что для рассматриваемых данных функциональные зависимости не заданы (за исключением тривиальных зависимостей типа Course → Course). Поэтому высказанные в предыдущей главе идеи не позволяют создать никакой формальной основы для выполнения декомпозиции данного отношения на проекции.

|  |
| --- |
| CTX |
| COURSE | TEACHER | TEXT |
| Физика | проф. Иванов | основы механики |
| Физика | проф. Иванов | оптика |
| Физика | проф. Петров | основы механики |
| Физика | проф. Петров  | оптика |
| Математика | проф. Иванов | основы механики |
| Математика | проф. Иванов | дискретная математика |
| Математика | проф. Иванов | тригонометрия |

рис. 7.2 Таблица нормализованного отношения CTX.

В простейшей формулировке нормализованное отношение CTX означает, что кортеж {Course:c, Teacher:t, Техт:x} появляется в данном отношении тогда и только тогда, когда курс c читается преподавателем t с использованием учебника x. Тогда, принимая во внимание допустимость существования для данного отношения всех возможных комбинаций преподавателей вместе с учебниками, можно утверждать, что для отношения CTX верно следующее ограничение: если присутствуют оба кортежа (c,tl,xl) и (c,t2,x2), тогда присутствуют также оба кортежа (c,tl,x2) и (c,t2,xl)

Очевидно, что отношение CTX характеризуется значительной избыточностью и приводит к возникновению аномалий обновления. Например, для добавления информации о том, что курс физики может читаться новым преподавателем, необходимо создать два новых кортежа, по одному для каждого учебника. Тем не менее, отношение CTX находится в НФБК, поскольку является "полностью ключевым".

Можно заметить, что ситуация может быть исправлена к лучшему, если заменить отношение СТХ его проекциями {Course, Teacher} и {Course, Text}, показанными на рис. 7.3. Обе проекции являются "полностью ключевыми" и находятся в НФБК; более того, отношение СТХ может быть восстановлено с помощью обратного соединения проекций СТ и СХ и потому данная композиция выполняется без потерь. Однако только в 1971 году эти интуитивные идеи были сформулированы Фейгином (Fagin) в строгом теоретическом виде с помощью понятия многозначных зависимостей.

|  |  |  |
| --- | --- | --- |
| CT |  | СХ |
| COURSE | TEACHER |  | COURSE | TEXT |
| физика | проф. Иванов | физика | основы механики |
| физика | проф. Петров | физика | оптика |
| математика | проф. Иванов | математика | основы механики |
|  |  | математика | дискретная математика |
|  |  | математика | тригонометрия |

рис. 7.3 Таблицы проекций СТ и СХ

Возвращаясь к рассматриваемому примеру с действительно корректной и желательной декомпозицией, показанной на рис. 7.3, следует, однако, отметить, что такая декомпозиция не может быть выполнена на основе функциональных зависимостей, поскольку они не существуют в данном отношении (кроме тривиальных зависимостей). Однако ее можно осуществить на основе нового типа зависимости, а именно упомянутой выше многозначной зависимости. Многозначные зависимости можно считать обобщением функциональных зависимостей в том смысле, что каждая функциональная зависимость является многозначной (однако обратное утверждение не верно, поскольку существуют многозначные зависимости, которые не являются функциональными). В отношении СТХ есть две многозначные зависимости:

Course—>>Teacher

Course—>>Text

Обратите внимание на двойную стрелку, которая в многозначной зависимости A—>>B означает, что "B многозначно зависит от A" или "A многозначно определяет B".

Пусть A, B и C являются произвольными подмножествами множества атрибутов отношения R. Тогда B многозначно зависит от A, что символически выражается записью

А—>>В

тогда и только тогда, когда множество значений B, соответствующее заданной паре (значение A, значение C) отношения R, зависит только от A, но не зависит от C.

Для данного отношения R{A, B, C} многозначная зависимость A—>>B выполняется тогда и только тогда, когда также выполняется многозначная зависимость A —>> C. Таким образом, многозначные зависимости всегда образуют связанные пары и потому их обычно представляют вместе в символическом виде:

А—>>В|С.

Для рассматриваемого примера такая запись будет иметь следующий вид:

Course—>>Teacher|Text

Возвращаясь к исходной задаче с отношением СТХ, теперь можно отметить, что описанная ранее проблема с отношением типа СТХ возникает из-за того, что оно содержит многозначные зависимости, которые не являются функциональными. (Следует отметить совсем неочевидный факт, что именно наличие таких МЗ требует вставлять два кортежа, когда необходимо добавить данные еще об одном преподавателе физики.) Проекции СТ и СХ не содержат многозначных зависимостей, а потому они действительно представляют собой некоторое усовершенствование исходной структуры. Поэтому было бы желательно заменить отношение СТХ двумя этими проекциями. Это можно сделать, исходя из теоремы Фейгина, которая приведена ниже.

Теорема Фейгина (эта теорема является более строгой версией теоремы Хеза). Пусть А, В и С являются множествами атрибутов отношения R{A, В, С}. Отношение R будет равно соединению его проекций {А, В} и {А, С} тогда и только тогда, когда для отношения R выполняется многозначная зависимость А—>>В|С.

## Четвертая нормальная форма

Отношение R находится в четвертой нормальной форме (4НФ) тогда и только тогда, когда существуют такие подмножества А и В атрибутов отношения R, что выполняется (нетривиальная) многозначная зависимость А —>> В. Тогда все атрибуты отношения R также функционально зависят от атрибута A.

## Зависимости соединения

До сих пор предполагалось, что единственной операцией в процессе декомпозиции является замена данного отношения (при декомпозиции без потерь) двумя его проекциями. Это допущение успешно выполнялось вплоть до определения 4НФ. Однако существуют отношения, для которых нельзя выполнить декомпозицию без потерь на две проекции, но которые можно подвергнуть декомпозиции без потерь на три или более проекции.

На рисунке представлен пример конкретного набора данных, соответствующих некоторому моменту времени. Однако, если данное отношение удовлетворяет некоторому не зависящему от времени ограничению, то 3-декомпозируемость отношения TSG может быть более фундаментальным и не зависящим от времени свойством, т.е. свойством, которое удовлетворяется для всех допустимых значений данного отношения. Для того чтобы понять, каким должно быть такое отношение, прежде всего отметим, что утверждение "отношение TSG равно соединению трех проекций TS, SG и TG" эквивалентно следующему утверждению:

Если пара (t1,s1) находится в отношении TS и пара (s1,g1) находится в отношении SG и пара (t1,g1) находится в отношении TG то тройка (t1,s1,g1) находится в отношении TSG.

|  |
| --- |
| TSG |
| TEACHER | SUBJECT | GROUP |
| Иванов | Математика | А-98-51 |
| Иванов | Физика | Б-00-51 |
| Петров | Математика | А-99-51 |
| Петров | Физика | А-98-51 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TS |  |  | SG |  |  | TG |  |
| TEACHER | SUBJECT |  | SUBJECT | GROUP |  | TEACHER | GROUP |
| Иванов | Физика |  | Математика | А-99-51 |  | Иванов | А-98-51 |
| Иванов | Математика |  | Математика | А-98-51 |  | Иванов | Б-00-51 |
| Петров | Физика |  | Физика | А-98-51 |  | Петров | А-99-51 |
| Петров | Математика |  | Физика | Б-00-51 |  | Петров | А-98-51 |
|  | ⎣Соединение по Subject⎦↓ |  |  |  |
|  | TEACHER | SUBJECT | GROUP |  |  |  |
| Иванов | Физика | А-98-51 |  |  |  |
| Иванов | Физика | Б-00-51 |  |  |  |
| Иванов | Математика | А-99-51 |  |  |  |
| Иванов | Математика | А-98-51 |  |  |  |
| Петров | Физика | А-98-51 |  |  |  |
| Петров | Физика | Б-00-51 |  |  |  |
|  | Петров | Математика | А-99-51 |  |  |  |
|  | Петров | Математика | А-98-51 |  |  |  |
|  |  | ⎣Соединение по комбинации Teacher и Group⎦↓ |  |
|  |  |  | Исходное TSG |  |  |

рис. 7.4 Отношение TSG является соединением трех бинарных проекций.

Исходя из этих заключений можно сказать, что пара (t1,s1) присутствует в отношении TS тогда и только тогда, когда тройка (t1, s1, g2) присутствует в отношении TSG для некоторого значения g2. Тогда приведенное выше утверждение можно переписать в виде ограничения, накладываемого на отношение SPJ:

Если (t1,s1,g2), (t2,s1,g1), (t1,s2,g1) находятся в отношении TSG то (t1,s1,g1) также находится в отношении TSG.

Если это утверждение выполняется всегда, т.е. для всех допустимых значений отношения TSG, то тем самым будет получено независящее от времени (хотя и несколько странное) ограничение для данного отношения. Обратите внимание на циклическую структуру этого ограничения. Отношение будет n-декомпозируемым для n>2 тогда и только тогда, когда оно удовлетворяет некоторому циклическому ограничению.

Циклическое ограничение с практической точки зрения обозначает, что, например, если:

1. Петров преподает математику;
2. математика преподается в А-98-51;
3. Петров преподает в А-98-51

то:

1. Петров преподает математику в А-98-51.

Обратите внимание, что из взятых вместе условий (1), (2) и (3) не следует (4).

Пусть R является отношением, а А, В,..., Z— произвольными подмножествами множества атрибутов отношения R. Отношение R удовлетворяет зависимости соединения

\* (A, B, ..., Z)

тогда и только тогда, когда оно равносильно соединению своих проекций с подмножествами атрибутов А, В, ..., Z.

Отсюда ясно, что отношение TSG с зависимостью соединения \*(TS, SG, TG) может быть 3-декомпозируемым. Однако следует ли выполнять такую декомпозицию? По всей видимости, да, так как отношение TSG характеризуется многочисленными аномалиями обновления, которые можно устранить с помощью 3-декомпозиции. Пример был приведен при определении циклического ограничения, из-за наличия которого, в отношении TSG должен присутствовать следующий кортеж (рис. 7.5)

|  |  |  |
| --- | --- | --- |
| TEACHER | SUBJECT | GROUP |
| Петров | Математика | А-98-51 |

рис. 7.5 Дополнительный кортеж.

Также теорема Фейгина может быть сформулирована следующим образом: отношение R{A, В, С} удовлетворяет зависимости соединения \*(АВ, АС) тогда и только тогда, когда оно удовлетворяет многозначной зависимости А —>> В | С.

Эту теорему можно использовать в качестве определения многозначной зависимости, отсюда следует, что многозначная зависимость является частным случаем зависимости соединения. Более того, из определения зависимости соединения следует, что из всех возможных форм это наиболее общая форма зависимости.

Возвращаясь к рассматриваемому примеру, можно обнаружить следующую проблему: отношение TSG содержит зависимость соединения, которая не является ни многозначной, ни функциональной зависимостью. Можно также заметить, что рекомендуется декомпозировать такое отношение на меньшие компоненты, а именно на проекции, заданные зависимостью соединения. Такой процесс декомпозиции может повторяться до тех пор, пока все результирующие отношения не будут находиться в пятой нормальной форме.

## Пятая нормальная форма

Отношение R находится в пятой нормальной форме (5НФ), которая также называется проекционно-соединительной нормальной формой, тогда и только тогда, когда каждая зависимость соединения в отношении R подразумевается потенциальными ключами отношения R.

Отношение TSG не находится в 5НФ. Оно удовлетворяет некоторой зависимости соединения, а именно ЗД-ограничению, которое, конечно, не подразумевается его единственным потенциальным ключом. Наоборот, после 3-декомпозиции проекции TS, SG и GT находятся в 5НФ, поскольку для них вовсе нет зависимостей соединения.

### Зависимости соединения, подразумеваемой потенциальными ключами

Рассмотрим простой пример, в котором дано отношение с данными студентов Students с потенциальным ключом StNo. Такое отношение удовлетворяет нескольким зависимостям соединения, например зависимости

\* ( (StNo, GrNo, StName), (StNo, CityNo) ).

Это значит, что отношение Students равносильно соединению его проекций с атрибутами {StNo, GrNo, StName} и {StNo, CityNo}, а потому может быть подвергнуто декомпозиции без потерь на указанные проекции. (Заметьте, что его не следует, а можно подвергнуть декомпозиции.) Существование этой зависимости соединения следует (или подразумевается) из того, что StNo является потенциальным ключом (в действительности это следует из теоремы Хеза).

В заключение заметим, что, как следует из определения 5НФ, она является окончательной нормальной формой по отношению к проекции и соединению. Таким образом, гарантируется, что отношение в пятой нормальной форме не содержит аномалий, которые могут быть исключены разбиением на проекции.

## Итоговая схема процедуры нормализации

Пусть дано отношение R, которое находится в 1НФ (или может быть приведено к такой форме после выравнивания исходной ненормализованной структуры), вместе с некоторыми ограничениями (функциональными зависимостями, многозначными зависимостями и зависимостями соединения). Тогда основная идея этой технологии состоит в систематическом приведении отношения R к набору меньших отношений, который в некотором заданном смысле эквивалентен отношению R, но более предпочтителен. Каждый этап процесса приведения состоит из разбиения на проекции отношений, полученных на предыдущем этапе, таким образом, чтобы проекции находились в нормальной форме более высокого порядка, чем первоначальное отношение.

Из приведенных выше правил можно выделить некоторые особенности.

1. Прежде всего, процесс разбиения на проекции на каждом этапе должен быть выполнен без потерь и с сохранением зависимости (там, где это возможно).
2. Необходимо подчеркнуть тот факт, что могут существовать соображения, по которым нормализацию не следует выполнять полностью.

Пятая нормальная форма является окончательной в том смысле, что дальнейшее устранение аномалий невозможно путем разбиения исходного отношения на проекции. Существуют нормальные формы более высоких порядков, однако они крайне редко встречаются на практике и в данном курсе не рассматриваются.

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 309–328.

# Проектирование БД методом сущность-связь. ER-диаграммы

8.1 Возникновение семантического моделирования

8.2 Основные понятия метода

8.3 Диаграммы ER-экземпляров и ER-типа

8.4 Правила формирования отношений

8.5 Методология IDEF1 (самостоятельное изучение)

## Возникновение семантического моделирования

Широкое распространение реляционных СУБД и их использование в самых разнообразных приложениях показывает, что реляционная модель данных достаточна для моделирования предметных областей. Однако проектирование реляционной базы данных в терминах отношений на основе механизма нормализации часто представляет собой очень сложный и неудобный для проектировщика процесс. Потребности проектировщиков баз данных в более удобных и мощных средствах моделирования предметной области вызвали к жизни направление семантических моделей данных. В этой лекции рассматривается одна из популярных семантических моделей данных – модель "сущность–связь".

Метод сущность-связь называют также методом "ER-диаграмм": во-первых, ER –аббревиатура от слов Essence (сущность) и Relation (связь), во-вторых, метод основан на использовании диаграмм, называемых соответственно диаграммами ER-экземпляров и диаграммами ER-типа.

## Основные понятия метода

Основными понятиями метода сущность-связь являются следующие:

1. сущность – представляет собой объект, информация о котором хранится в БД. Экземпляры сущности отличаются друг от друга и однозначно идентифицируются. Названиями сущностей являются, как правило, существительные, например: ПРЕПОДАВАТЕЛЬ, ДИСЦИПЛИНА, ГРУППА.
2. Атрибут сущности – представляет собой свойство сущности. Это понятие аналогично понятию атрибута в отношении. Так, атрибутами сущности ПРЕПОДАВАТЕЛЬ может быть его Фамилия, Должность, Стаж (преподавательский) и т. д.
3. Ключ сущности – атрибут или набор атрибутов, используемый для идентификации экземпляра сущности. Как видно из определения, понятие ключа сущности аналогично понятию ключа отношения.;
4. Связь между сущностями. Связь двух или более сущностей - предполагает зависимость между атрибутами этих сущностей. Название связи обычно представляется глаголом. Примерами связей между сущностями являются следующие- ПРЕПОДАВАТЕЛЬ ВДЕТ ДИСЦИПЛИНУ (Иванов ВЕДЕТ "Организацию БД и знаний"), ПРЕПОДАВАТЕЛЬ ПРЕПОДАЕТ В ГРУППЕ (Иванов ПРЕПОДАЕТ В 256 группе);
5. Степень связи – является характеристикой связи между сущностями, которая может быть следующих видов: 1:1, 1:М, М:1, М:М.;
6. Класс принадлежности (КП) экземпляров сущности. КП сущности может быть: обязательным и необязательным. Класс принадлежности сущности является обязательным, если все экземпляры этой сущности обязательно участвуют в рассматриваемой связи, в противном случае класс принадлежности сущности является необязательным.
7. Диаграммы ER-экземпляров;
8. Диаграммы ER-типа.

Приведенные определения сущности и связи не полностью формализованы, но приемлемы для практики. Следует иметь в виду, что в результате проектирования могут быть получены несколько вариантов одной и той же БД. Так, два разных проектировщика, рассматривая одну и ту же проблему с разных точек зрения, могут получить различные наборы сущностей и связей. При этом оба варианта могут быть рабочими, а выбор лучшего из них будет результатом личных предпочтений.

## Диаграммы ER-экземпляров и ER-типа

С целью повышения наглядности и удобства проектирования для представления сущностей, экземпляров сущностей и связей между ними используются следующие графические средства:

1. диаграммы ER-экзрмпляров,
2. диаграммы ER-типа, или ER-диаграммы.

На рисунке рис. 8.1 приведена диаграмма ER-экземпляров для сущностей ПРЕПОДАВАТЕЛЬ и ДИСЦИПЛИНА со связью ВЕДЕТ.

рис. 8.1 Диаграмма ER-экземпляров.

|  |  |  |
| --- | --- | --- |
| ПРЕПОДАВАТЕЛЬ | ВЕДЕТ | ДИСЦИПЛИНА |
|  |  |  |
| Иванов |  | Средства СУБД |
| Петров |  | C++ |
| Сидоров |  | Паскаль |
| Егоров |  | Алгол |
| Козлов |  | Фортран |

Диаграмма ER-экземпляров показывает, какую конкретно дисциплину (СУБД, C++ и т.д.) ведет каждый из преподавателей. На рис. 8.2 представлена диаграмма ER-типа, соответствующая рассмотренной диаграмме ER-экземпляров.

рис. 8.2 Диаграмма ER-типа.

Преподаватель

Дисциплина

Ведет

На начальном этапе проектирования БД выделяются атрибуты, составляющие ключи сущностей.

На основе анализа диаграмм ER-типа формируются отношения проектируемой БД. При этом учитывается степень связи сущностей и класс их принадлежности, которые, в свою очередь, определяются на основе анализа диаграмм ER-экземпляров соответствующих сущностей.

Варьируя классом принадлежности сущностей для каждого из названных типов связи, можно получить несколько вариантов диаграмм ER-типа. Рассмотрим примеры некоторых из них.

### Связи типа 1:1 и необязательный класс принадлежности

В приведенной на рис. 8.2 диаграмме степень связи между сущностями 1:1, а класс принадлежности обеих сущностей необязательный. Действительно, из рисунка видно следующее:

1. каждый преподаватель ведет не более одной дисциплины, а каждая дисциплина ведется не более чем одним преподавателем (степень связи 1:1);
2. некоторые преподаватели не ведут ни одной дисциплины и имеются дисциплины, которые не ведет ни один из преподавателей (класс принадлежности обеих сущностей необязательный).

### Связи типа 1:1 и обязательный класс принадлежности

На рисунке приведены диаграммы, у которых степень связи между сущностями 1:1, а класс принадлежности обеих сущностей обязательный.

рис. 8.3 Диаграмма ER-экземпляров для связи 1:1 и обязательным КП обеих сущностей.

|  |  |  |
| --- | --- | --- |
| ПРЕПОДАВАТЕЛЬ | ВЕДЕТ | ДИСЦИПЛИНА |
|  |  |  |
| Иванов |  | Средства СУБД |
| Петров |  | C++ |
| Сидоров |  | Паскаль |
| Егоров |  | Алгол |
| Козлов |  | Фортран |

рис. 8.4 Диаграмма ER-типа для связи 1:1 и обязательным КП обеих сущностей.

Преподаватель

Дисциплина

Ведет

1

1

В этом случае каждый преподаватель ведет одну дисциплину и каждая дисциплина ведется одним преподавателем.

Возможны два промежуточных варианта с необязательным классом принадлежности одной из сущностей.

Диаграммы ER-типа графически изображаются следующим образом:

1. обязательное участие в связи экземпляров сущности отмечается блоком с точкой внутри, смежным с блоком этой сущности (рис. 8.4).
2. необязательное участие экземпляров сущности в связи – дополнительный блок к блоку сущности не пристраивается, а точка размещается на линии связи (рис. 8.2).
3. символы на линии связи указывают на степень связи.
4. под каждым блоком, соответствующим некоторой сущности, указывается ее ключ, выделяемый подчеркиванием. Многоточие за ключевыми атрибутами означает, что возможны другие атрибуты сущности, но ни один из них не может быть частью ее ключа. Эти атрибуты выявляются после формирования отношений.

На практике степень связи и класс принадлежности сущностей при проектировании БД определяется спецификой предметной области. Рассмотрим примеры вариантов со степенью связи 1:М или М:1.

Связь типа 1:М – каждый преподаватель может вести несколько дисциплин, но каждая дисциплина ведется одним преподавателем,

Связи типа М:1 – каждый преподаватель может вести одну дисциплину, но каждую дисциплину могут вести несколько преподавателей.

Примеры с типом связи 1:М или М:1 могут иметь ряд вариантов, отличающихся классом принадлежности одной или обеих сущностей. Обозначим обязательный класс принадлежности символом "О", а необязательный - символом "Н", тогда варианты для связи типа 1:М условно можно представить как: О–О, О–Н, Н–О, Н–Н. Для связи типа М:1 также имеются 4 аналогичных варианта.

### Связи типа 1:М вариант Н-О

Каждый преподаватель может вести несколько дисциплин ИЛИ ни одной, но каждая дисциплина ведется одним преподавателем (рис. 8.5, рис. 8.6).

рис. 8.5 Диаграмма ER-экземпляров для связи типа 1:М варианта Н-О

|  |  |  |
| --- | --- | --- |
| ПРЕПОДАВАТЕЛЬ | ВЕДЕТ | ДИСЦИПЛИНА |
|  |  |  |
|  |  | СУБД |
| Иванов |  | ПЛ/1 |
| Петров |  | Паскаль |
| Сидоров |  |  Алгол |
| Егоров |  |  Фортран |
| Козлов |  | C++ |
|  |  | Java |

рис. 8.6. Диаграмма ER-типа для связи типа 1:М варианта Н-О

Преподаватель

Дисциплина

Ведет

1

М

По аналогии легко составить диаграммы и для остальных вариантов.

Связи типа М:М – каждый преподаватель может вести несколько дисциплин, а каждая дисциплина может вестись несколькими преподавателями. Как и в случае других типов связей, для связи типа М:М возможны 4 варианта, отличающиеся классом принадлежности сущностей.

### Связи типа М:М и вариант класса принадлежности О-Н

Допустим, что каждый преподаватель ведет не менее одной дисциплины, а дисциплина может вестись более чем одним преподавателем, есть и такие дисциплины, которые никто не ведет. Соответствующие этому случаю диаграммы приведены на рисунке рис. 8.7.

рис. 8.7 Диаграмма ER-экземпляров для связи типа М:М и вариант класса принадлежности О-Н.

|  |  |  |
| --- | --- | --- |
| ПРЕПОДАВАТЕЛЬ | ВЕДЕТ | ДИСЦИПЛИНА |
|  |  |  |
|  |  | Средства СУБД |
| Иванов |  | ПЛ/1 |
| Петров |  | Паскаль |
| Сидоров |  | Алгол |
| Егоров |  | Фортран |
| Козлов |  | C++ |
|  |  | Java |

рис. 8.8 Диаграмма ER-типов для связи типа М : М и варианта О-Н.

Преподаватель

Дисциплина

Ведет

М

М

Выявление сущностей и связей между ними, а также формирование на их основе диаграмм ER-типа выполняется на начальных этапах метода сущность-связь. Рассмотрим этапы реализации метода.

Процесс проектирования базы данных является итерационным – допускающим возврат к предыдущим этапам для пересмотра ранее принятых решений и включает следующие этапы:

1. выделение сущностей и связей между ними;
2. построение диаграмм er-типа с учетом всех сущностей и их связей;
3. формирование на основе построенных ранее диаграмм er-типа набора предварительных отношений с указанием предполагаемого первичного ключа для каждого отношения;
4. добавление не ключевых атрибутов в отношения;
5. приведение предварительных отношений к нормальной форме Бойса-Кодда, например, с помощью метода нормальных форм;
6. пересмотр er-диаграмм в следующих случаях;
	1. некоторые отношения не приводятся к нормальной форме Бойса-Кодда;
	2. некоторым атрибутам не находится логически обоснованных, мест в предварительных отношениях.

После преобразования ER-диаграмм осуществляется повторное выполнение предыдущих этапов проектирования (возврат к этапу 1).

Одним из узловых этапов проектирования является этап формирования отношений. Рассмотрим процесс формирования предварительных отношений, составляющих первичный вариант схемы БД.

В рассмотренных выше примерах связь ВЕДЕТ всегда соединяет две сущности и поэтому является бинарной. Сформулированные ниже правила формирования отношений из диаграмм ER-типа распространяются именно на бинарные связи. Поэтому, когда речь идет о связях, слово "бинарные" далее опускается.

## Правила формирования отношений

Правила формирования отношений основываются на учете следующего:

1. степени связи между сущностями (1:1, 1:М, М:1, М:М);
2. класса принадлежности экземпляров сущностей (обязательный и необязательный).

Рассмотрим формулировки шести правил формирования отношений на основе диаграмм ER-типа.

### Степень связи 1:1, класс принадлежности обеих сущностей обязательный

Если степень бинарной связи 1:1 и класс принадлежности обеих сущностей обязательный, то формируется одно отношение. Первичным ключом этого отношения может быть ключ любой из двух сущностей.

На рис. 8.9 приведены диаграмма ER-типа и отношение, сформированное по правилу 8.4.1 на ее основе.

рис. 8.9 Диаграмма и отношения для правила 8.4.1

C1

C2

1

1

R1

K1 или K2,…

K1,…

K2,…

На рис. 8.9 используются следующие обозначения:

Cl, C2 – сущности 1 и 2;

Kl, K2 – ключи первой и второй сущности соответственно;

Rl – отношение 1, сформированное на основе первой и второй сущностей;

Kl, K2,... означает, что ключом сформированного отношения может быть либо К1, либо К2.

### Степень связи 1:1, класс принадлежности одной сущности обязательный, а второй – необязательный

Если степень связи 1:1 и класс принадлежности одной сущности обязательный, а второй – необязательный, то под каждую из сущностей формируется по отношению с первичными ключами, являющимися ключами соответствующих сущностей. Далее к отношению, сущность которого имеет обязательный КП, добавляется в качестве атрибута ключ сущности с необязательным КП.

На рис. 8.10 приведены диаграмма ER-типа и отношения, сформированные по правилу 8.4.2 на ее основе.

рис. 8.10 Диаграмма и отношения для правила 8.4.2

C1

C2

1

1

K1,…

K2,…

R1

K1, K2

R2

K2,…

### Степень связи 1:1, класс принадлежности обеих сущностей – необязательный

Если степень связи 1:1 и класс принадлежности обеих сущностей является необязательным, то необходимо использовать три отношения. Два отношения соответствуют связываемым сущностям, ключи которых являются первичными в этих отношениях. Третье отношение является связным между первыми двумя, поэтому его ключ объединяет ключевые атрибуты связываемых отношений.

рис. 8.11 Диаграмма и отношения для правила 8.4.3

C1

C2

1

1

K1,…

K2,…

R1

K1,…

R2

R1\_R2

K1, K2

K2,…

На рис. 8.11 приведены диаграмма ER-типа и отношения, сформированные по правилу 8.4.3 на ее основе.

Сформулируем аналогичные два правила для вариантов, степень связи между сущностями которых 1:М. Если две сущности С1 и С2 связаны как 1:М, сущность С1 будем называть односвязной (1-связной), а сущность С2-многосвязной (М-связной). Определяющим фактором при формировании отношений, связанных этим видом связи, является класс принадлежности М-связной сущности. Так, если класс принадлежности М-связной сущности обязательный, то в результате применения правила получим два отношения, если необязательный – три отношения. Класс принадлежности односвязной сущности не влияет на результат.

### Степень связи между сущностями 1:М (или М:1), класс принадлежности М-связной сущности обязательный

Если степень связи между сущностями 1:М (или М:1) и класс принадлежности М-связной сущности обязательный, то достаточно формирование двух отношений (по одному на каждую из сущностей). При этом первичными ключами этих отношений являются ключи их сущностей. Кроме того, ключ 1-связной сущности добавляется как атрибут (внешний ключ) в отношение, соответствующее М-связной сущности.

На рис. 8.12 приведены диаграмма ER-типа и отношения, сформированные по правилу 8.4.4.

рис. 8.12 Диаграмма и отношения для правила 8.4.4.

C1

C2

1

М

K1,…

K2,…

K1,…

R1

R2

K1,K2,…

### Степень связи 1:М (М:1)и класс принадлежности М-связной сущности – необязательный

Если степень связи 1:М (М:1)и класс принадлежности М-связной сущности является необязательным, то необходимо формирование трех отношений (рис. 8.13).

R1

K1,…

R2

R1\_R2

K1, K2

K2,…

C1

C2

1

М

K1,…

K2,…

рис. 8.13 Диаграмма и отношение для правила 8.4.5

Два отношения соответствуют связываемым сущностям, ключи которых являются первичными в этих отношениях. Третье отношение является связным между первыми двумя (его ключ объединяет ключевые атрибуты связываемых отношений).

При наличии связи М:М между двумя сущностями необходимо три отношения независимо от класса принадлежности любой из сущностей. Использование одного или двух отношений в этом случае не избавляет от пустых полей или избыточно дублируемых данных.

### Степень связи М:М, независимо от класса принадлежности сущностей

Если степень связи М:М, то независимо от класса принадлежности сущностей формируются три отношения Два отношения соответствуют связываемым сущностям и их ключи являются первичными ключами этих отношений. Третье отношение является связным между первыми двумя, а его ключ объединяет ключевые атрибуты связываемых отношений.

На рис. 8.14 приведены диаграмма ER-типа и отношения, сформированные по правилу 8.4.6. В конспекте показан вариант с классом принадлежности сущностей Н-Н, хотя, согласно правилу 8.4.6, он может быть произвольным.

рис. 8.14. Диаграмма и отношения для правила 8.4.6.

R1

K1,…

R2

R1\_R2

K1, K2

K2,…

C1

C2

М

М

K1,…

K2,…

Аналогичные результаты получаются и для трех других вариантов, различающихся классами принадлежности их сущностей.

## Методология IDEF1 (самостоятельное изучение)

Метод IDEF1, разработанный Т. Рэмей (T. Ramey), также основан на подходе П. Чена и позволяет построить модель данных, эквивалентную реляционной модели в третьей нормальной форме. В настоящее время на основе совершенствования методологии IDEF1 создана ее новая версия - методология IDEF1X. IDEF1X разработана с учетом таких требований, как простота изучения и возможность автоматизации. IDEF1X-диаграммы используются рядом распространенных CASE-средств (таких, как, ERwin, Design/IDEF).

Сущность в методологии IDEF1X является независимой от идентификаторов или просто независимой, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями (рис. 8.15). Сущность называется зависимой от идентификаторов или просто зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности (рис. 8.16).

рис. 8.15. Независимые от идентификатора сущности.

Имя сущности/Номер сущности

Студент/44

рис. 8.16. Зависимые от идентификатора сущности.

Имя сущности/Номер сущности

Оценка/56

Каждой сущности присваивается уникальное имя и номер, разделяемые косой чертой "/" и помещаемые над блоком.

Связь может дополнительно определяться с помощью указания степени или мощности (количества экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности-родителя). В IDEF1X могут быть выражены следующие мощности связей:

1. каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка;
2. каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка;
3. каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка;
4. каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка.

Если экземпляр сущности-потомка однозначно определяется своей связью с сущностью-родителем, то связь называется идентифицирующей, в противном случае – неидентифицирующей.

Связь изображается линией, проводимой между сущностью-родителем и сущностью-потомком с точкой на конце линии у сущности-потомка. Мощность связи обозначается как показано на рис. 8.17 (мощность по умолчанию – N).

Ноль или один

Z

Один или более

P

Ноль, один или более

N

рис. 8.17. Мощность связи.

Идентифицирующая связь между сущностью-родителем и сущностью-потомком изображается сплошной линией (рис. 8.18). Сущность-потомок в идентифицирующей связи является зависимой от идентификатора сущностью. Сущность-родитель в идентифицирующей связи может быть как независимой, так и зависимой от идентификатора сущностью (это определяется ее связями с другими сущностями).

рис. 8.18. Идентифицирующая связь.

Ключевой-атрибут-A

Сущность-A/1

Ключевой-атрибут-A (FK)

Ключевой-атрибут-B

Сущность-B/2

Сущность-родитель

Сущность-потомок

Имя связи

Пунктирная линия изображает неидентифицирующую связь (рис. 8.19). Сущность-потомок в неидентифицирующей связи будет независимой от идентификатора, если она не является также сущностью-потомком в какой-либо идентифицирующей связи.

рис. 8.19. Неидентифицирующая связь.

Ключевой-атрибут-A

Сущность-A/1

**Атрибут-**A (FK)

Ключевой-атрибут-B

Сущность-B/2

Сущность-родитель

Сущность-потомок

Имя связи

Атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой (рис. 8.20).

рис. 8.20. Атрибуты и первичные ключи.

имя-атрибута

[имя-атрибута]

...

[имя-атрибута]

[имя-атрибута]

[имя-атрибута]

...

**Атрибуты первичного ключа**

**Имя\_сущности/номер\_сущности**

Сущности могут иметь также внешние ключи (Foreign Key), которые могут использоваться в качестве части или целого первичного ключа или неключевого атрибута. Внешний ключ изображается с помощью помещения внутрь блока сущности имен атрибутов, после которых следуют буквы FK в скобках (рис. 8.21).

рис. 8.21. Примеры внешних ключей.

Код\_Студента

**Код\_группы (**FK)

**Студент/44**

Код\_Студента (FK)

Код\_дисциплины (FK)

Код\_документа

**Оценка/44**

**Литература:**

1. Базы данных: Учебник для высших учебных заведений /Под ред. проф. А.Д. Хомоненко. –Спб.: КОРОНА принт, 2000. –416с. Стр. 147–161.
2. Сергей Кузнецов, “Основы современных баз данных”. Центр Информационных Технологий, http://www.citforum.ru/database/osbd/contents.shtml

# Язык SQL

9.1 История создания и развития SQL

9.2 Основные понятия SQL

9.3 Запросы на чтение данных. Оператор SELECT

9.4 Многотабличные запросы на чтение (объединения).

## История создания и развития SQL

Язык для взаимодействия с БД SQL появился в середине 70-х и был разработан в рамках проекта экспериментальной реляционной СУБД System R. Исходное название языка SEQUEL (Structered English Query Language) только частично отражает суть этого языка. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционной БД, но на самом деле уже являлся полным языком БД, содержащим помимо операторов формулирования запросов и манипулирования БД средства определения и манипулирования схемой БД; определения ограничений целостности и триггеров; представлений БД; возможности определения структур физического уровня, поддерживающих эффективное выполнение запросов; авторизации доступа к отношениям и их полям; точек сохранения транзакции и откатов. Таким образом, SQL стал достаточно мощным языком для взаимодействия с СУБД. На сегодняшний день SQL является единственным стандартным языком запросов. Язык SQL обладает следующими достоинствами:

1. независимость от конкретных СУБД. Если при создании БД не использовались нестандартные возможности языка SQL предоставляемые некоторой СУБД, то такую БД можно без изменений перенести на СУБД другого производителя. К сожалению большинство БД используют особенности СУБД, на которой работают, что затрудняет их перенос на другую СУБД без изменений;
2. реляционная основа. Реляционная модель имеет солидный теоретический фундамент. Язык SQL основан на реляционной модели и является единственным языком для реляционных БД;
3. SQL обладает высокоуровневой структурой, напоминающей английский язык.
4. SQL позволяет создавать различные представления данных для различных пользователей;
5. SQL является полноценным языком для работы с БД;
6. стандарты языка SQL. Официальный стандарт языка SQL опубликован ANSI и ISO в 1989 году и значительно расширен в 1992 году.

## Основные понятия SQL

### Операторы

В SQL используется приблизительно тридцать операторов, каждый из которых "просит" СУБД выполнить определенное действие, например, прочитать данные, создать таблицу или добавить в таблицу новые данные. Все операторы SQL имеют одинаковую структуру, которая показана на рис. 9.1.

рис. 9.1 Структура оператора SQL.

Глагол

Имя таблицы

SELECT \* FROM Students

WHERE StNo = 2

Имя столбца

Константа

Ключевые слова

Предложения

Каждый оператор SQL начинается с глагола, т.е. ключевого слова, описывающего действие, выполняемое оператором. Типичными глаголами являются SELECT (выбрать), CREATE (создать), INSERT (добавить), DELETE (удалить), COMMIT(завершить). После глагола идет одно или несколько предложений. Предложение описывает данные, с которыми работает оператор, или содержит уточняющую информацию о действии, выполняемом оператором. Каждое предложение также начинается с ключевого слова, такого как WHERE (где), FROM (откуда), INTO (куда) и HAVING (имеющий). Одни предложения в операторе являются обязательным, а другие – нет. Конкретная структура и содержимое предложения могут изменяться. Многие предложения содержат имена таблиц или столбцов; некоторые из них могут содержать дополнительные ключевые слова, константы и выражения.

В стандарте ANSI/ISO определены ключевые слова, которые применяются в качестве глаголов и в предложениях операторов. В соответствии со стандартом, эти ключевые слова нельзя использовать для именования объектов базы данных, таких как таблицы, столбцы и пользователи

### Имена.

У каждого объекта в базе данных есть уникальное имя. Имена используются в операторах SQL и указывают, над каким объектом базы данных оператор должен выполнить действие. В стандарте ANSI/ISO определено, что имена имеются у таблиц, столбцов и пользователей. Во многих реализациях SQL поддерживаются также дополнительные именованные объекты, такие как хранимые процедуры, именованные отношения "первичный ключ – внешний ключ" и формы для ввода данных.

В соответствии со стандартом ANSI/ISO, в SQL имена должны содержать от 1 до 18 символов, начинаться с буквы и не содержать пробелы или специальные символы пунктуации. В стандарте SQL2 максимальное число символов в имени увеличено до 128.

Полное имя таблицы состоит из имени владельца таблицы и собственно ее имени, разделенных точкой (.). Например, полное имя таблицы Students, владельцем которой является пользователь по имени Admin, имеет следующий вид:

Admin.Students

Если в операторе задается имя столбца, SQL сам определяет, в какой из указанных в этом же операторе таблиц содержится данный столбец. Однако если в оператор требуется включить два столбца из различных таблиц, но с одинаковыми именами, необходимо указать полные имена столбцов, которые однозначно определяют их местонахождение. Полное имя столбца состоит из имени таблицы, содержащей столбец, и имени столбца (простого имени), разделенных точкой (.). Например, полное имя столбца StName из таблицы Students имеет следующий вид:

Students.StName

### Типы данных в SQL

В стандарте ANSI/ISO определены типы данных, которые можно использовать для представления информации в реляционной базе данных. Типы данных, имеющиеся в стандарте SQL1, составляют лишь минимальный набор и поддерживаются во всех коммерческих СУБД. В табл. 9.1 перечислены типы данных, определенные в стандартах SQL1 и SQL2.

табл. 9.1 Типы данных в SQL.

| Тип данных | Описание |
| --- | --- |
| CHAR (длина) | Строки символов постоянной длины |
| CHARACTER (длина) |
| VARCHAR(длина) | Строки символов переменной длины\* |
| CHAR VARYING(длина) |
| CHARACTER VARYING (длина) |
| NСНАР(длина) | Строки локализованных символов постоянной длины\* |
| NATIONAL CHAR(длина) |
| NATIONAL CHARACTER(длина) |
| NCHAR VARYING(длина) | Строки локализованных символов переменной длины\* |
| NATIONAL CHAR VARYING(длина) |
| NATIONAL CHARACTER VARYING(длина) |
| INTEGER | Целые числа |
| INT |
| SMALLINT | Маленькие целые числа |
| BIT(длина) | Строки битов постоянной длины\* |
| BIT VARYNG(длина) | Строки битов переменной длины\* |
| NUMERIC(точность, степень) | Масштабируемые целые (десятичные) числа |
| DECIMAL(точность, степень) |
| DEC(точность, степень) |
| FLOAT(точность) | Числа с плавающей запятой |
| REAL | Числа с плавающей запятой низкой точности |
| DOUBLE PRECISION | Числа с плавающей запятой высокой точности |
| DATE | Календарная дата\* |
| TIME(точность) | Время |
| TIME STAMP(точность) | Дата и время\* |
| INTERVAL | Временной интервал\* |

В SQL1 используются следующие типы данных:

1. Строки символов постоянной длины. В столбцах, имеющих этот тип данных, обычно хранятся имена людей и компаний, адреса, описания и т д.
2. Целые числа. В столбцах, имеющих этот тип данных, обычно хранятся данные о счетах, количествах, возрастах и т.д. Целочисленные столбцы часто используются также для хранения идентификаторов, таких как идентификатор клиента, служащего или заказа.
3. Масштабируемые целые числа. В столбцах данного типа хранятся числа,) имеющие дробную часть, которые необходимо вычислять точно, например курсы валют и проценты. Кроме того, в таких столбцах часто хранятся) денежные величины.
4. Числа с плавающей запятой. Столбцы этого типа используются для хранения величин, которые можно вычислять приблизительно, например веса и расстояния. Числа с плавающей запятой могут представлять больший диапазон значений, чем десятичные числа, однако при вычислениях могут возникать погрешности округления.

В большинстве коммерческих СУБД помимо типов данных, определенных в стандарте SQL1, имеется множество дополнительных типов данных, большинство из которых вошло в стандарт SQL2. Ниже перечислены наиболее важные из них:

1. Строки символов переменной длины. Почти во всех СУБД поддерживается тип данных VARCHAR, позволяющий хранить строки символов, длина которых изменяется в некотором диапазоне В стандарте SQL1 были определены строки постоянной длины, которые справа дополняются пробелами.
2. Денежные величины. Во многих СУБД поддерживается тип данных MONEY или CURRENCY, который обычно хранится в виде десятичного числа или числа с плавающей запятой. Наличие отдельного типа данных для представления денежных величин позволяет правильно форматировать их при выводе на экран.
3. Дата и время. Поддержка значений даты/времени также широко распространена в различных СУБД, хотя способы ее реализации довольно сильно отличаются друг от друга. Как правило, над значениями этого типа данных можно выполнять различные операции. В стандарт SQL2 входит определение типов данных DATE, TIME, TIMESTAMP и INTERVAL, включая поддержку часовых поясов и возможность указания точности, представления времени (например, десятые или сотые доли секунды).
4. Булевы данные. Некоторые СУБД явным образом поддерживают логические значения (TRUE или FALSE).

### Константы

В стандарте ANSI/ISO определен формат числовых и строковых констант, или литералов, которые представляют конкретные значения данных. Этот формат используется в большинстве реализации SQL.

Числовые константы. Целые и десятичные константы (известные также под названием точных числовых литералов) в операторах SQL представляются в виде обычных. десятичных чисел с необязательным знаком плюс (+) или минус (-) перед ними:

21 -3752000,00 +497500,8778

Константы с плавающей запятой (известные также под названием приблизительных числовых литералов) определяются с помощью символа Е и имеют такой же формат, как и в большинстве языков программирования. Ниже приведены примеры констант с плавающей запятой:

1.5Е3 -3.14159Е1 2.5Е-7 0.783926Е21

Символ Е читается как "умножить на десять в степени", так что первая константа представляет число "1,5 умножить на десять в степени 3", или 1500.

Строковые константы. В соответствии со стандартом ANSI/ISO, строковые константы в SQL должны быть заключены в одинарные кавычки, как показано в следующих примерах:

Jones, John J.' 'New York' 'Western'

Если необходимо включить в строковую константу одинарную кавычку, вместо нее следует написать две одинарные кавычки.

Константы даты и времени. В реляционных СУБД календарные даты, время и интервалы времени представляются в виде строковых констант. Форматы этих констант в различных СУБД отличаются друг от друга. Кроме того, способы записи времени и даты изменяются в зависимости от страны.

Символьные константы. Кроме пользовательских констант, в SQL существуют специальные символьные константы, возвращающие значения, хранимые в самой СУБД.

В стандарт SQL2 вошли наиболее полезные символьные константы из различных реализации SQL, в частности константы CURRENT\_DATE, CURRENT\_TIME, CURRENT\_TIMESTAMP, а также USER, SESSION\_USER и SYSTEM\_USER (обратите внимание на знак подчеркивания!).

Выражения в SQL используются для выполнения операций над значениями, считанными из базы данных или используемыми для поиска в базе. данных. Например, в следующем запросе вычисляется процентное соотнесение объема и плана продаж для каждого офиса:

SELECT CITY, TARGET, SALES, (SALES/TARGET) \* 100

FROM OFFICES

В соответствии со стандартом ANSI/ISO, в выражениях можно использовать четыре арифметические операции: сложение (X + Y), вычитание (X – Y), умножение (X \* Y) и деление (X / Y). Для формирования сложных выражений можно использовать скобки.

Отсутствующие данные (значения NULL). Поскольку база данных представляет собой модель реального мира, отдельные элементы данных в ней неминуемо будут отсутствовать или подходить не для всех сущностей. Для указания на такие данные используеBӦся специальная константа – NULL.

Встроенные функции

В стандарт SQL2 вошли наиболее полезные функции из различных реализации SQL. Эти функции перечислены в таблце.

табл. 9.2 Встроенные функции SQL.

| Функция | Возвращается |
| --- | --- |
| ВIT LENGTH(строка) | количество битов в строке |
| САSТ(значениеAS тип данных) | значение, преобразованное тип данных (например, дата преобразованная в строку) |
| CHAR\_LENGTH(строка) | длина строки символов |
| CONVERT(строка USING функция) | строка, преобразованная в соответствии с указанной функцией |
| CURRENT\_DATE | текущая дата |
| CURRENT\_TIME(точность) | текущее время с указанной точностью |
| CURRENT\_IMESTAMP (точность) | текущие дата и время с указанной точностью |
| EXTRACT(часть FROM значение) | указанная часть (DAY, HOUR и т.д.) из значения типа DATETIME |
| LOWER(строка) | строка, преобразованная к нижнему регистру |
| OCTETLENGTH(строка) | число байтов в строке символов |
| POSITION(первая строкаIN вторая строка) | позиция, с которой начинается вхождение первой строки во вторую строку |
| SUBSTRING(строка FROMn FOR длина) | часть строки, начинающаяся с n-го символа и имеющая указанную длину |
| TRANSLATE(строка USING функция) | строка, преобразованная с помощью указанной функции |
| TRIM(BOTH символFROM строка) | строка, в которой удалены первые и последние указанные символы |
| TRIM(LEADING символFROM строка) | строка, в которой удалены первые указанные символы |
| TRIM(TRAILING символFROM строка) | строка, в которой удалены последние указанные символы |
| UPPER(строка) | строка, преобразованная к верхнему регистру |

## Запросы на чтение данных. Оператор SELECT

Оператор SELECT применяется для построения выборок данных и имеет следующий синтаксис:

SELECT [ALL | DISTINCT] возвращаемый\_столбец, … | \*

FROM спецификатор\_таблицы, …

WHERE условие\_поиска

GROUP BY имя\_столбца, …

HAVING условие\_поиска

ORDER BY спецификатор\_сортировки, …

### Предложение SELECT

В предложении SELECT, с которого начинаются все операторы SELECT, необходимо указать элементы данных, которые будут возвращены в результате запроса Эти элементы задаются в виде списка возвращаемых столбцов, разделенных запятыми Для каждого элемента из этого списка в таблице результатов запроса будет создан один столбец Столбцы в таблице результатов будут расположены в том же порядке, что и элементы списка возвращаемых столбцов Возвращаемый столбец может представлять собой:

1. имя столбца, идентифицирующее один из столбцов, содержащихся в таблицах, которые перечислены в предложении FROM Когда в качестве возвращаемого столбца указывается имя столбца таблицы базы данных SQL просто берет значение этого столбца для каждой из строк таблицу и помещает его в соответствующую строку таблицы результатов запроса;
2. константа, показывающая, что в каждой строке результатов запроса должно содержаться одно и то же значение,
3. выражение, показывающее, что SQL должен вычислять значение, помещаемое в результаты запроса, по формуле, определенной в выражении.

### Предложение FROM

Предложение FROM состоит из ключевого слова FROM, за которым следует список спецификаторов таблиц, разделенных запятыми Каждый спецификатор таблицы идентифицирует таблицу, содержащую данные, которые считывает запрос Такие таблицы называются исходными таблицами запроса (и оператора SELECT), поскольку все данные, содержащиеся в таблице результатов запроса, берутся из них.

Результатом SQL-запроса на чтение всегда является таблица, содержащая данные и ничем не отличающаяся от таблиц базы данных

Кроме столбцов, значения которых считываются непосредственно из базы данных, SQL-запрос на чтение может содержать вычисляемые столбцы, значения которых определяются на основании значений, хранящихся в базе данных. Чтобы получить вычисляемый столбец, в списке возвращаемых столбцов необходимо указать выражение.

Иногда требуется получить содержимое всех столбцов таблицы. На практике такая ситуация может возникнуть, когда вы впервые сталкиваетесь с новой базой данных и необходимо быстро получить представление о ее структуре и хранимых в ней данных. С учетом этого в SQL разрешается использовать вместо списка .возвращаемых столбцов символ звездочки (\*), который означает, что требуется прочитать все столбцы:

Показать все данные, содержащиеся в таблице Students.

SELECT \*

FROM Students

Повторяющиеся строки из таблицы результатов запроса можно удалить, если в операторе SELECT перед списком возвращаемых столбцов указать ключевое слово DISTINCT.

### Отбор строк (предложение WHERE)

SQL-запросы, считывающие из таблицы все строки, полезны при просмотре базы данных и создании отчетов, однако редко применяются для чего-нибудь еще Обычно требуется выбрать из таблицы несколько строк и включить в результаты запроса только их Чтобы указать, какие строки требуется отобрать, следует использовать предложение WHERE.

Предложение WHERE состоит из ключевого слова WHERE, за которым следует условие поиска, определяющее, какие именно строки требуется прочитать.

Если условие поиска имеет значение TRUE, строка будет включена в результаты запроса.

Если условие поиска имеет значение FALSE или NULL, то строка исключается из результатов запроса.

### Условия поиска

В SQL используется множество условий поиска, позволяющих эффективно и естественным образом создавать различные типы запросов. Ниже рассматриваются пять основных условий поиска (в стандарте ANSI/ISO они называются предикатами).

Сравнение (=, <>, <, <=, >, >=). Наиболее распространенным условием поиска в SQL является сравнение. При сравнении SQL вычисляет и сравнивает значения двух выражений для каждой строки данных. Выражения могут быть как очень простыми, например содержать одно имя столбца или константу, так и более сложными – арифметическими выражениями.

Ниже приведен синтаксис оператора сравнения.

Выражение1 = | <> | < | <= | > | >= Выражение2

Проверка на принадлежность диапазону значений (BETWEEN). Другой формой условия поиска является проверка на принадлежности диапазону значений (ключевое слово BETWEEN). При этом проверяется, находится ли значение данных между двумя определенными значениями.

Проверяемое\_выражение [NOT] BETWEEN нижнее\_выражение AND верхнее\_выражение

При проверке на принадлежность диапазону верхний и нижний пределы считаются частью диапазона, поэтому в результаты запроса.

Если выражение, определяющее нижний предел диапазона, имеет значение NULL, то проверка BETWEEN возвращает значение FALSE тогда, когда проверяемое значение больше верхнего предела диапазона, и значение NULL в противном случае.|

Если выражение, определяющее верхний предел диапазона, имеет значение NULL, то проверка BETWEEN возвращает значение FALSE, когда проверяемое значение меньше нижнего предела, и значение NULL противном случае.

Поверка на членство в множестве (IN). Еще одним распространенным условием поиска является проверка на членство в множестве (IN). В этом случае проверяется, соответствует ли значение данных какому-либо значению из заданного списка.

проверяемое\_выражение [NOT] IN (константа, …)

Например: вывести список фамилий студентов, которые учатся в группах с кодами 1, 3, 5 и 10.

SELECT StName

FROM Students

WHERE GrNo IN (1, 3, 5, 10)

Проверка на соответствие шаблону (LIKE). Для чтения строк, в которых содержимое некоторого текстового столбца совпадает с заданным текстом, можно использовать простое сравнение, однако иногда необходимо осуществить сравнение на основе фрагмента строки.

Поверка на соответствие шаблону (ключевое слово LIKE), позволяет определить, соответствует ли значение данных в столбце некоторому шаблону. Шаблон представляет собой строку, в которую может входить один или более подстановочных знаков. Эти знаки интерпретируются особым образом.

имя\_столбца [NOT] LIKE шаблон [ESCAPE символ\_пропуска]

Подстановочные знаки. Подстановочный знак % совпадает с любой последовательностью из нуля или более символов. Например, следующий запрос выведет информацию о студентах, чья фамилия начинается с "Иван":

SELECT \*

FROM StudentsWHERE StName LIKE 'Иван%'

Подстановочный знак "\_" (символ подчеркивания) совпадает с любым отдельным символом. Например, если вы уверены, что имя студентки- либо "Наталья", либо "Наталия", то можете воспользоваться следующим запросом:

SELECT \*

FROM Students WHERE StName LIKE 'Натал\_я'

Подстановочные знаки можно помещать в любое место строки шаблона, и в одной строке может содержаться несколько подстановочных знаков.

Символы пропуска. При проверке строк на соответствие шаблону может оказаться, что подстановочные знаки входят в строку символов в качестве литералов. Например, нельзя проверить, содержится ли знак процента в строке, просто включив его в шаблон, поскольку SQL будет считать этот знак подстановочным. Как правило, это не вызывает серьезных проблем, поскольку подстановочные знаки довольно редко встречаются в именах, названиях товаров и других текстовых данных, которые обычно хранятся в базе данных.

В стандарте ANSI/ISO определен способ проверки наличия в строках литералов, использующихся в качестве подстановочных знаков. Для этом применяются символы пропуска. Когда в шаблоне встречается такой символ то символ, следующий непосредственно за ним, считается не подстановочным знаком, а литералом. (Происходит пропуск символа.) Непосредственно за символом пропуска может следовать либо один из двух подстановочных знаков, либо сам символ пропуска, поскольку он тоже приобретает в шаблоне особое значение.

Символ пропуска определяется в виде строки, состоящей из одного символа, и предложения ESCAPE. Ниже приведен пример использования знака доллара ($) в качестве символа пропуска:

SELECT \*

FROM DataTable WHERE Text LIKE '%менее 50$% %' ESCAPE $

Проверка на равенство значению NULL (IS NULL). Значения NULL обеспечивают возможность применения трехзначной логики в условиях поиска. Для любой заданной строки результат применения условия поиска может быть TRUE, FALSE или NULL (в случае, когда в одном из столбцов содержится значение NULL). Иногда бывает необходимо явно проверять значения столбцов на равенство NULL и непосредственно обрабатывать их. Для этого в SQL имеется специальная проверка на равенство значению NULL (IS NULL).

IS [NOT] NULL имя\_ столбца

Составные условия поиска (AND, OR и NOT). Простые условия поиска, описанные в предыдущих параграфах, после применения к некоторой строке возвращают значения TRUE, FALSE или NULL. С помощью правил логики эти простые условия можно объединять в более сложные. Условия поиска, объединенные с помощью ключевых слов AND, OR и NOT, сами могут быть составными.

WHERE [NOT] условие\_поска [AND | OR] [NOT] условие\_поска …

### Сортировка результатов запроса (предложение ORDER BY).

Строки результатов запроса, как и строки таблицы базы данных, не имеют определенного порядка. Включив в оператор SELECT предложение ORDER BY, можно отсортировать результаты запроса. Это предложение состоит из ключевых слов ORDER BY, за которыми следует список имен столбцов, разделенных запятыми.

ORDER BY имя\_столбца [ASC | DESC], …

В предложении ORDER BY можно выбрать возрастающий или убывающий порядок сортировки. По умолчанию, данные сортируются в порядке возрастания. Чтобы сортировать их по убыванию, следует включить в предложение сортировки ключевое слово DESC.

Например: вывести список фамилий студентов учащихся в группе с кодом 1 в обратном алфавитном порядке.

SELECT StName

FROM Students ORDER BY DESC StName

## **Многотабличные запросы на чтение (объединения).**

На практике многие запросы считывают данные сразу из нескольких таблиц базы данных.

### Запросы с использованием отношения предок/потомок.

Среди многотабличных запросов наиболее распространены запросы к двум таблицам, связанным с помощью отношения предок/потомок. Запрос о студентах, учащихся в группе с некоторым названием является примером такого запроса. У каждого студента (потомка) есть соответствующая ему группа (предок), и каждая группа (предок) может иметь много студентов (потомков). Пары строк, из которых формируются результаты запроса, связаны отношением предок/потомок.

Например: вывести список фамилий студентов с названием группы, в которой он учится

SELECT StName, GrName

FROM Students, Groups

WHERE Students.GrNo = Groups.GrNo

### Прочие объединения таблиц по равенству

Огромное множество многотабличных запросов основано на отношениях предок/потомок, но в SQL не требуется, чтобы связанные столбцы представляли собой пару "внешний ключ – первичный ключ". Любые два столбца из двух таблиц могут быть связанными столбцами, если только они имеют сравнимые типы данных.

Объединение таблиц по неравенству. Термин "объединение" применяется к любому запросу, который объединяет данные из двух таблиц базы данных путем сравнения значений в двух столбцах этих таблиц. Самыми распространенными являются объединения, созданные на основе равенства связанных столбцов (объединения по равенству). Кроме того, SQL позволяет объединять таблицы с помощью других операций сравнения. Например, получить все коды дисциплин по которым студентом с кодом 1 была получена оценка большая, чем по дисциплине с кодом 1.

SELECT Marks1.SubjNo

FROM Marks AS Marks1, Marks AS Marks2

WHERE Marks1.Mark > Marks2.Mark

AND Marks2.SubjNo = 1 AND Marks1.StNo = 1

Следует признать, что данный пример имеет несколько искусственный характер и является иллюстрацией того, почему столь мало распространены объединения по неравенству. Однако они могут оказаться полезными в приложениях, предназначенных для поддержки принятия решений, и в других приложениях, исследующих более сложные взаимосвязи в базе данных.

Полные имена столбцов. В учебной базе данных имеется несколько случаев, когда две таблицы содержат столбцы с одинаковыми именами. Например, столбцы с именем GrNo имеются в таблицах Groups и Students.

Чтобы исключить разночтения, при указании столбцов необходимо использовать их полные имена. Полное имя столбца содержит имя столбца и имя таблицы, в которой он находится. Полные имена двух столбцов GrNo в учебной базе данных будут такими:

Groups.GrNo Students.GrNo

В операторе SELECT вместо простых имен столбцов всегда можно использовать полные имена. Таблица, заданная в полном имени столбца, должна, конечно, соответствовать одной из таблиц, заданных в предложении FROM.

Чтение всех столбцов. Как уже говорилось ранее, оператор SELECT \* используется для чтения всех столбцов таблицы, указанной в предложении FROM. В многотабличном запросе звездочка означает выбор всех столбцов из всех таблиц, указанных в предложении FROM.

Самообъединения. Некоторые многотабличные запросы используют отношения, существующие внутри одной из таблиц. Предположим, например, что требуется вывести список имен всех преподавателей и их руководителей. Каждому преподавателю соответствует одна строка в таблице Teachers, а столбец ChiefNo содержит идентификатор преподавателя, являющегося руководителем. Столбцу ChiefNo следовало бы быть внешним ключом для таблицы, в которой хранятся данные о руководителях. И он им, фактически, является – это внешний ключ для самой таблицы Teachers.

Для объединения таблицы с самой собой в SQL применяется именно такой подход: использование "виртуальной копии". Вместо того чтобы на самом деле сделать копию таблицы, SQL позволяет вам сослаться на нее, используя другое имя, которое называется псевдонимом таблицы.

Например: вывести список всех преподавателей и их руководителей.

SELECT Teachers. TName, Chiefs.TName

FROM Teachers, Teachers Chiefs

WHERE Teachers. ChiefNo = Chiefs.TNo

Псевдонимы таблиц. Как уже было сказано в предыдущем параграфе, псевдонимы таблиц необходимы в запросах, включающих самообъединения. Однако псевдоним можно использовать в любом запросе (например, если запрос касается таблицы другого пользователя или если имя таблицы очень длинное и использовать его в полных именах столбцов утомительно).

Внешнее объединение таблиц. Операция объединения в SQL соединяет информацию из двух таблиц, формируя пары связанных строк из этих двух таблиц. Объединенную таблицу образуют пары тех строк из различных таблиц, у которых в связанных столбцах содержатся одинаковые значения. Если строка одной из таблиц не имеет пары, то такой вид объединения, называемый внутренним объединением, может привести к неожиданным результатам (потере некоторых данных в результате запроса). Для создания объединений таблиц, имеющих неодинаковые значения в столбцах, на основе которых осуществляется связь, применяют внешнее объединение таблиц наиболее полно представленное в стандарте SQL2.

**Литература:**

1. Джеймс Р. Грофф, Пол Н. Вайнберг. SQL: полное руководство: пер.с англ. –К.: Издательская группа BHV, 2000.–608с. Стр. 31–39,69–166.

# Язык SQL (продолжение)

10.1 Объединения и стандарт SQL2

10.2 Итоговые запросы на чтение. Агрегатные функции

10.3 Запросы с группировкой (предложение GROUP BY)

10.4 Вложенные запросы

## Объединения и стандарт SQL2

В стандарте SQL2 был определен совершенно новый метод поддержки внешних объединений, который не опирался ни на одну популярную СУБД. В спецификации стандарта SQL2 поддержка внешних объединений осуществлялась в предложении FROM с тщательно разработанным синтаксисом, позволявшим пользователю точно определить, как исходные таблицы должны быть объединены в запросе. Расширенное предложение FROM поддерживает также операцию UNION над таблицами и допускает сложные комбинации запросов на объединение операторов SELECT и объединений таблиц.

### Внутренние объединения в стандарте SQL2

Две объединяемые таблицы соединяются явно посредством операции JOIN, а условие поиска, описывающее объединение, находится теперь в предложении ON внутри предложения FROM В условии поиска, следующем за ключевым словом ON, могут быть заданы любые критерии сравнения строк двух объединяемых таблиц.

Например: вывести список фамилий студентов, и названия групп, к которых они учаться.

SELECT StName, GrName

FROM Students INNER JOIN Groups

ON Students.GrNo = Groups.GrNo

(В этих простых двухтабличных объединениях все содержимое предложения WHERE просто перешло в предложение ON, и предложение ON не добавляет ничего нового в язык SQL.

Стандарт SQL2 допускает еще один вариант запроса на простое внутреннее объединение таблиц Students и Groups. Так как связанные столбцы этих таблиц имеют одинаковые имена и сравниваются на предмет равенства (что делается довольно часто), то можно использовать альтернативную форму предложения ON, в которой задается список имен связанных столбцов:

SELECT StName, GrName

FROM Students INNER JOIN Groups USING (GrNo)

Ниже приведен синтаксис оператора JOIN:

1. естественное соединение.

FROM спецификация\_таблиц,…,

таблица1

NATURAL {INNER|FULL[OUTER]|LEFT[OUTER]|RIGHT[OUTER]}JOIN

таблица2 …

1. соединение с использованием выражения.

FROM спецификация\_таблицы,…,

таблица1

{INNER|[OUTER] FULL|[OUTER]LEFT|[OUTER]RIGHT} JOIN

таблица2

ON условие | USING(список\_столбцов),…

1. объединение или декартово произведение.

FROM спецификация\_таблицы,…,

таблица1 {UNION | CROSS JOIN} таблица2 ,…

Объединение двух таблиц, в котором связанные столбцы имеют идентичные имена, называется естественным объединением, так как обычно это действительно самый "естественный" способ объединения двух таблиц. Запрос на выборку пар фамилия студента/название группы, в которой он учиться, можно выразить как естественное объединение следующим образом:

SELECT StName, GrName

FROM Students NATURAL INNER JOIN Groups

### Внешние объединения в стандарте SQL2

Стандарт SQL2 обеспечивает полную поддержку внешних объединений, расширяя языковые конструкции, используемые для внутренних объединений. Например, для построения таблицы подчиненности преподавателей можно применить следующий запрос:

SELECT Chief.TName, SubOrdinate.TName

FROM Teachers AS Chief FULL OUTER JOIN Teachers AS SubOrdinate

ON Chief.TNo = SubOrdinate.TChiefNo

Результат такого запроса (данные из Приложения А) приведен на рис. 10.1.

|  |  |
| --- | --- |
| Chief.TName | SubOrdinate.TName |
| NULL | Иванов |
| Иванов | Петров |
| Петров | Стрельцов |
| Петров | Сидоров |
| Сидоров | NULL |
| Стрельцов | NULL |

рис. 10.1 Результатом такого запроса на внешнее объединение.

Таблица результатов запроса будет содержать по одной строке для каждой связанной пары начальник/подчиненный, а также по одной строке для каждой несвязанной записи для начальника или подчиненного, расширенной значениями NULL в столбцах другой таблицы.

Ключевое слово OUTER, так же как и ключевое слово INNER, в стандарте SQL2 не является обязательным. Поэтому предыдущий запрос можно, было бы переписать следующим образом:

SELECT Chief.TName, SubOrdinate.TName

FROM Teachers AS Chief FULL JOIN Teachers AS SubOrdinate

ON Chief.TNo = SubOrdinate.TChiefNo

По слову FULL СУБД сама определяет, что запрашивается внешнее объединение.

Вполне естественно, что в стандарте SQL2 левое и правое внешние объединения обозначаются словами LEFT и RIGHT вместо слова FULL. Вот вариант того же запроса, определяющий левое внешнее объединение:

SELECT Chief.TName, SubOrdinate.TName

FROM Teachers AS Chief LEFT OUTER JOIN Teachers AS SubOrdinate

ON Chief.TNo = SubOrdinate.TChiefNo

В результате такого запроса (данные из Приложения А.) будет получено следующее отношение (рис. 10.2).

|  |  |
| --- | --- |
| Chief.TName | SubOrdinate.TName |
| Иванов | Петров |
| Петров | Стрельцов |
| Петров | Сидоров |
| Сидоров | NULL |
| Стрельцов | NULL |

рис. 10.2 Результатом такого запроса на внешнее объединение

### Перекрестные объединения и запросы на объединение в SQL2

Расширенное предложение FROM в стандарте SQL2 поддерживает также два других способа соединения данных из двух таблиц – декартово произведение и запросы на объединение. Строго говоря, ни один из них не является операцией "объединения", но они поддерживаются в стандарте SQL2 с помощью тех же самых предложений, что и внутренние и внешние объединения. Вот запрос, создающий декартово произведение таблиц Students и Groups:

SELECT \*

FROM Students CROSS JOIN Groups

### Многотабличные объединения в стандарте SQL2

Одно из крупных преимуществ расширенного предложения FROM заключается в том, что оно дает единый стандарт для определения как внутренних и внешних объединений, так и произведений и запросов на объединение. Другим, даже еще более важным преимуществом этого предложения является то, что оно обеспечивает очень ясную и четкую спецификацию объединений трех и четырех таблиц, а также произведений и запросов на объединение. Для построения этих сложных объединений любые выражения описанные ранее, могут быть заключены в круглые скобки. Результирующее выражение, в свою очередь, можно использовать для создания других выражений объединения, как если бы оно было простой таблицей. Точно так же, как SQL позволяет с помощью круглых скобок комбинировать различные арифметические операции (+, –, \* и /) и строить сложные выражения, стандарт SQL2 дает возможность создавать сложные выражения для объединений.

## Итоговые запросы на чтение. Агрегатные функции

Для подведения итогов по информации, содержащейся в базе данных, в SQL предусмотрены агрегатные (статистические) функции. Агрегатная функция принимает в качестве аргумента какой-либо столбец данных целиком, а возвращает одно значение, которое определенным образом подытоживает этот столбец. Например, агрегатная функция AVG() принимает в качестве аргумента столбец чисел и вычисляет их среднее значение.

В SQL имеется шесть агрегатных функций, которые позволяют получать различные виды итоговой информации. Ниже описан синтаксис этих функций:

1. функция SUM() вычисляет сумму всех значений, содержащихся в столбце:

SUM(выражение | [DISTINCT] имя\_столбца)

1. функция AVG() вычисляет среднее всех значений, содержащихся в столбце:

AVG(выражение | [DISTINCT] имя\_столбца)

1. функция MIN() находит наименьшее среди всех значений, содержащихся в столбце:

MIN(выражение | имя\_столбца)

1. функция МАХ() находит наибольшее среди всех значений, содержащихся в столбце:

MAX(выражение | имя\_столбца)

1. функция COUNT() подсчитывает количество значений, содержащихся в столбце:

COUNT([DISTINCT] имя\_столбца)

1. функция COUNT(\*) подсчитывает количество строк в таблице результатов запроса:

COUNT(\*)

### Агрегатные функции и значения NULL

В стандарте ANSI/ISO также определены следующие точные правила обработки значений NULL в агрегатных функциях:

1. если какие-либо из значений, содержащихся в столбце, равны NULL, при вычислении результата функции они исключаются;
2. если все значения в столбце равны NULL, то функции SUM(), AVG(), MIN() и MAX () возвращают значение NULL; функция COUNT () возвращает ноль;
3. если в столбце нет значений (т.е. столбец пустой), то функции SUM() , AVG(), MIN() и МАХ() возвращают значение NULL; функция COUNT() возвращает ноль;
4. функция COUNT(\*) подсчитывает количество строк и не зависит от наличия или отсутствия в столбце значений NULL; если строк в таблице нет, эта функция возвращает ноль.

## Запросы с группировкой (предложение GROUP BY)

Итоговые запросы, о которых до сих пор шла речь в настоящей главе рассчитывают итоговые результаты на основании всех записей в таблице. Однако необходимость в таких вычислениях возникает достаточно редко, чаще бывает необходимо получать промежуточные итоги на основании групп записей в таблице. Эту возможность предоставляет предложение GROUP BY оператора SELECT.

Запрос, включающий в себя предложение GROUP BY, называется запросом с группировкой, поскольку он объединяет строки исходных таблиц в группы и для каждой группы строк генерирует одну строку таблицы результатов запроса. Столбцы, указанные в предложении GROUP BY, называются столбцами группировки, поскольку именно они определяют, по какому признаку строки делятся на группы. Например, получить список фамилий студентов и их средних оценок.

SELECT StName, AVG(Mark)

FROM Marks INNER JOIN Students USING(StNo)

GROUP BY StName

### Несколько столбцов группировки

SQL позволяет группировать результаты запроса на основании двух или более столбцов. Например, получить список фамилий студентов и их средних оценок за каждый семестр.

SELECT StName, Semester, AVG(Mark)

FROM Marks INNER JOIN Students USING(StNo)

GROUP BY StName, Semester

### Ограничения на запросы с группировкой

На запросы, в которых используется группировка, накладываются дополнительные ограничения. Столбцы с группировкой должны представлять собой реальные столбцы таблиц, перечисленных в предложении FROM. Нельзя группировать строки на основании значения вычисляемого выражения.

Кроме того, существуют ограничения на элементы списка возвращаемых столбцов. Все элементы этого списка должны иметь одно значение для каждой группы строк. Это означает, что возвращаемым столбцом может быть:

1. константа;
2. агрегатная функция, возвращающая одно значение для всех строк, входящих в группу;
3. столбец группировки, который, по определению, имеет одно и то же значение во всех строках группы;
4. выражение, включающее в себя перечисленные выше элементы.

На практике в список возвращаемых столбцов запроса с группировкой всегда входят столбец группировки и агрегатная функция. Если последняя не указана, значит, запрос можно более просто выразить с помощью ключевого слова DISTINCT без использования предложения GROUP BY. И наоборот, если не включить в результаты запроса столбец группировки, вы не сможете определить, к какой группе относится каждая строка результатов.

### Значения NULL в столбцах группировки

В стандарте ANSI определено, что два значения NULL в предложении GROUP BY равны.

Строки, имеющие значение NULL в одинаковых столбцах группировки и идентичные значения во всех остальных столбцах группировки, помещаются в одну группу.

### Условия поиска групп (предложение HAVING)

Точно так же, как предложение WHERE используется для отбора отдельных строк, участвующих в запросе, предложение HAVING можно применить для отбора групп строк. Его формат соответствует формату предложения WHERE. Предложение HAVING состоит из ключевого слова HAVING, за которым следует условие поиска. Таким образом, данное предложение определяет условие поиска для групп.

### Ограничения на условия поиска групп

Предложение HAVING используется для того, чтобы включать и исключать группы строк из результатов запроса, поэтому на используемое в нем условие поиска наложены такие же ограничения, как и на элементы списка возвращаемых столбцов.

### Предложение HAVING без GROUP BY

Предложение HAVING почти всегда используется в сочетании с предложением GROUP BY, однако синтаксис оператора SELECT не требует этого. Если предложение HAVING используется без предложения GROUP BY, SQL рассматривает полные результаты запроса как одну группу. Другими словами, агрегатные функции, содержащиеся в предложении HAVING, применяются к одной и только одной группе, и эта группа состоит из всех строк. На практике предложение HAVING очень редко используется без соответствующего предложения GROUP BY.

## Вложенные запросы

Вложенным (или подчиненным) запросом называется запрос, содержащийся в предложении WHERE или HAVING другого оператора SQL. Вложенные запросы позволяют естественным образом обрабатывать запросы, выраженные через результаты других запросов.

Чаще всего вложенные запросы указываются в предложении WHERE оператора SQL. Когда вложенный запрос содержится в данном предложении, он участвует в процессе отбора строк.

Например, вывести список фамилий студентов, средний балл которых выше 4,5:

SELECT StName

FROM Students

WHERE (SELECT AVG(Mark)

 FROM Marks

 WHERE Marks.StNo = Students.StNo) > 4.5

Вложенный запрос всегда заключается в круглые скобки, но по-прежнему сохраняет знакомую структуру оператора SELECT, содержащего предложение FROM и необязательные предложения WHERE, GROUP BY и HAVING. Структура этих предложений во вложенном запросе идентична их структуре в оператора SELECT; во вложенном запросе эти предложения выполняют свои обычные функции. Однако между вложенным запросом и оператором SELECT имеется ряд отличий:

1. Таблица результатов вложенного запроса всегда состоит из одного столбца. Это означает, что в предложении SELECT вложенного запроса всегда указывается один возвращаемый столбец.
2. Во вложенный запрос не может входить предложение ORDER BY. Результаты вложенного запроса используются только внутри главного запроса и для пользователя остаются невидимыми, поэтому нет смысла их сортировать.
3. Вложенный запрос не может быть запросом на объединение нескольких различных операторов SELECT; допускается использование только одного оператора SELECT.
4. Имена столбцов во вложенном запросе могут являться ссылками на столбцы таблиц главного запроса. Это так называемые внешние ссылки (ссылка на Students.StNo в предыдущем примере). Внешние ссылки необязательно должны присутствовать во вложенном запросе.

### Условия поиска во вложенном запросе

Вложенный запрос всегда является частью условия поиска в предложении WHERE или HAVING. Ранее были рассмотрены простые условия поиска, которые могут использоваться в этих предложениях. Кроме того, в SQL имеются следующие условия поиска во вложенном запросе.

Сравнение с результатом вложенного запроса (=, <>, <, <=, >, >=). Сравнивает значение выражения с одним значением, возвращенным вложенным запросом. Эта проверка напоминает простое сравнение.

проверяемое\_выражение  | = | <> | < | <= | > | >= | вложенный\_запрос

Сравнение с результатом вложенного запроса является модифицированной формой простого сравнения. Значение выражения сравнивается со значением, возвращенным вложенным запросом, и если условие сравнения выполняется, то проверка возвращает значение TRUE. Эта проверка используется для сравнения значения из проверяемой строки с одним значением, возвращенным вложенным запросом, как показано первом примере.

Проверка на принадлежность результатам вложенного запроса (IN). Проверка на принадлежность результатам вложенного запроса (ключевое слово IN) является видоизмененной формой простой проверки на членство в множестве.

проверяемое\_выражение [NOT] IN вложенный\_запрос

Одно значение сравнивается со столбцом данных, возвращенных вложенным запросом, и если это значение равно одному из значений в столбце, проверка возвращает TRUE. Данная проверка используется, когда необходимо сравнить значение из проверяемой строки с множеством значений, возвращенных вложенным запросом. Например, вывести всю информацию о студентах, учащихся в группах сназваниями, начинающимися на букву ″А″:

SELECT \*

FROM Students

WHERE GrNo IN (SELECT GrNo FROM Groups WHERE GrName LIKE ‘А%’)

Проверка на существование (EXISTS). В результате проверки на существование (ключевое слово EXISTS) можно выяснить, содержится ли в таблице результатов вложенного запроса хотя бы одна строка. Аналогичной простой проверки не существует. Проверка на существование используется только с вложенными запросами.

[NOT] EXISTS вложенный\_запрос

Например, вывести фамилии студентов, которые в 1-м семестре сдали хотябы одну дисциплину:

SELECT StName

FROM Students

WHERE EXISTS (SELECT \*

 FROM Marks

 WHERE Marks.StNo = Students.StNo AND Semester = 1)

Многократное сравнение (ANY и ALL). В проверке IN выясняется, не равно ли некоторое значение одному из значений, содержащихся в столбце результатов вложенного запроса. В SQL имеется две разновидности многократного сравнения – ANY и ALL, расширяющие предыдущую проверку до уровня других операторов сравнения. В обеих проверках некоторое значение сравнивается со столбцом данных, возвращенным вложенным запросом.

проверяемое\_выражение  | = | <> | < | <= | > | >= |  ANY | ALL вложенный\_запрос

Проверка ANY. В проверке ANY используется один из шести операторов сравнения (=, <>, <, <=, >, >=) для того, чтобы сравнить одно проверяемое значение со столбцом данных, возвращенным вложенным запросом. Проверяемое значение поочередно сравнивается с каждым значением, содержащимся в столбце. Если любое из этих сравнений дает результат TRUE, то проверка ANY возвращает значение TRUE.

Например, вывести список фамилий студентов, получивших в первом семестре хотя бы одну отличную оценку.

SELECT StName

FROM Students

WHERE 5 = ANY (SELECT Mark

 FROM Marks

 WHERE Marks.StNo = Students.StNo AND Semester = 1)

Если вложенный запрос в проверке ANY не создает ни одной строки результата или если результаты содержат значения NULL, то в различных СУБД проверка ANY может выполняться по-разному. В стандарте ANSI/ISO для языка SQL содержатся подробные правила, определяющие результаты проверки ANY, когда проверяемое значение сравнивается со столбцом результатов вложенного запроса:

1. если вложенный запрос возвращает пустой столбец результатов, то проверка ANY имеет значение FALSE (в результате выполнения вложенного запроса не получено ни одного значения, для которого выполнялось бы условие сравнения).
2. если операция сравнения имеет значение TRUE хотя бы для одного значения в столбце, то проверка ANY возвращает значение TRUE (имеется некоторое значение, полученное вложенным запросом, для которого условие сравнения выполняется).
3. если операция сравнения имеет значение FALSE для всех значений в столбце, то проверка ANY возвращает значение FALSE (можно утверждать, что ни для одного значения, возвращенного вложенным запросом, условие сравнения не выполняется);
4. если операция сравнения не имеет значение TRUE ни для одного значения в столбце, но в нем имеется одно или несколько значений NULL то проверка ANY возвращает результат NULL. (В этой ситуации невозможно но с определенностью утверждать, существует ли полученное вложенным запросом значение, для которого выполняется условие сравнения; может быть, существует, а может и нет – все зависит от "настоящих" значений неизвестных данных.)

Проверка ALL. В проверке ALL, как и в проверке ANY, используется один из шести операторов (=, <>, <, <=, >, >=) для сравнения одного проверяемого значения со столбцом данных, возвращенным вложенным запросом. Проверяемое значение поочередно сравнивается с каждым значением, содержащимся в столбце. Если все сравнения дают результат TRUE, то проверка ALL возвращает значение TRUE.

Например, вывести список фамилий студентов, получивших в первом семестре только удовлетворительные оценки.

SELECT StName

FROM Students

WHERE 3 = ALL (SELECT Mark

 FROM Marks

 WHERE Marks.StNo = Students.StNo AND Semester = 1)

Если вложенный запрос в проверке ALL не возвращает ни одной строки или если результаты запроса содержат значения NULL, то в различных СУБД проверка ALL может выполняться по-разному. В стандарте ANSI/ISO для языка SQL содержатся подробные правила, определяющие результаты проверки ALL, когда проверяемое значение сравнивается со столбцом результатов вложенного запроса:

1. если вложенный запрос возвращает пустой столбец результатов, то проверка ALL имеет значение TRUE. Считается, что условие сравнения выполняется, даже если результаты вложенного запроса отсутствуют.
2. если операция сравнения дает результат TRUE для каждого значения в столбце, то проверка ALL возвращает значение TRUE. Условие сравнения выполняется для каждого значения, полученного вложенным запросом.
3. если операция сравнения дает результат FALSE для какого-нибудь значения в столбце, то проверка ALL возвращает значение FALSE. В этом, случае можно утверждать, что условие поиска выполняется не для каждого значения, полученного вложенным запросом.
4. если операция сравнения не дает результат FALSE ни для одного значения в столбце, но для одного или нескольких значений дает результат NULL, то проверка ALL возвращает значение NULL. В этой ситуации нельзя с определенностью утверждать, для всех ли значений, полученных вложенным запросом, справедливо условие сравнения; может быть, для всех, а может и нет – все зависит от "настоящих" значений неизвестных данных.

### Вложенные запросы и объединения

При чтении данной главы вы, возможно, заметили, что многие запросы, записанные с применением вложенных запросов, можно также записать в виде многотабличных запросов. Такое случается довольно часто, и SQL позволяет записать запрос любым способом.

### Уровни вложенности запросов

Все рассмотренные до сих пор запросы были "двухуровневыми" и состояли из главного и вложенного запросов. Точно так же, как внутри главной запроса может находится вложенный запрос, внутри вложенного запроси может находиться еще один вложенный запрос.

### Вложенные запросы в предложении HAVING

Хотя вложенные запросы чаще всего применяются в предложении WHERE их можно использовать и в предложении HAVING главного запроса. Когда вложенный запрос содержится в предложении HAVING, он участвует в отбор группы строк.

**Литература:**

1. Джеймс Р. Грофф, Пол Н. Вайнберг. SQL: полное руководство: пер.с англ. –К.: Издательская группа BHV, 2000.–608с. Стр. 169–217.

# Язык SQL. (продолжение)

11.1 Внесение изменений в базу данных.

11.2 Удаление существующих данных (Оператор DELETE)

11.3 Обновление существующих данных (Оператор UPDATE)

11.4 Определение структуры данных в SQL

11.5 Понятие представления.

11.6 Представления в SQL.

11.7 Системный каталог (самостоятельное изучение)

## Внесение изменений в базу данных.

SQL представляет собой полноценный язык, предназначенный для работы с данными и позволяющий не только извлекать информацию из базы данных с помощью запросов на чтение, но и изменять содержащуюся в ней информацию с помощью запросов на добавление, удаление и обновление.

По сравнению с оператором SELECT, с помощью которого выполняются запросы на чтение, операторы SQL, изменяющие содержимое базы данных, являются более простыми. Однако при изменении содержимого базы данных к СУБД предъявляется ряд дополнительных требований. При внесении изменений СУБД должна сохранять целостность данных и разрешать ввод в базу данных только допустимых значений, а также обеспечивать непротиворечивость базы данных даже в случае системной ошибки. Помимо этого, СУБД должна обеспечивать возможность одновременного изменения базы данных несколькими пользователями таким образом, чтобы они не мешали друг другу.

### Добавление новых данных (оператор INSERT).

Однострочный оператор INSERT, синтаксис которого описан ниже, добавляет в таблицу новую строку. В предложении INTO указывается таблица, в которую добавляется новая строка (целевая таблица), а в предложении VALUES содержатся значения данных для новой строки. Список столбцов определяет, какие значения в какой столбец заносятся.

INSERT INTO имя\_таблицы (имя\_столбца,…) VALUES (константа | NULL,…)

Ниже приведен пример оператора INSERT, который добавляет информацию о новой группе ″К-99-51″ в учебную базу данных:

INSERT INTO Groups(GrNo, EnterYear, GrName)

VALUES(6, 1999, 'К-99-51')

Добавление значений NULL. При добавлении в таблицу новой строки всем столбцам, имена которых отсутствуют в списке столбцов оператора INSERT, автоматически присваивается значение NULL.

Добавление всех столбцов. Для удобства в SQL разрешается не включать список столбцов в оператор INSERT. Если список столбцов опущен, он генерируется автоматически и в нем слева направо перечисляются все столбцы таблицы. При выполнении оператора SELECT \* генерируется такой же список столбцов. Пользуясь этой сокращенной формой записи, оператор INSERT из предыдущего примера можно переписать таким образом:

INSERT INTO Groups

VALUES(6, 1999, 'К-99-51')

Если список столбцов опущен, то в списке значений необходимо явно указывать значение NULL. Кроме того, последовательность значений данных должна в точности соответствовать порядку столбцов в таблице.

Многострочный оператор INSERT, добавляет в целевую таблицу несколько строк. В этой разновидности оператора INSERT значения данных для новых строк явно не задаются. Источником новых строк служит запрос на чтение, содержащийся внутри оператора INSERT.

INSERT INTO имя\_таблицы (имя\_столбца,…) запрос

## Удаление существующих данных (Оператор DELETE)

Наименьшей единицей информации, которую можно удалить из реляционной базы данных, является одна строка.

Оператор DELETE, синтаксис которого изображен ниже, удаляет выбранные строки данных из одной таблицы.

DELETE FROM имя\_таблицы [WHERE условие\_поиска]

В предложении FROM указывается таблица, содержащая строки, которые требуется удалить. В предложении WHERE указываются строки, которые должны быть удалены.

Например, удалить из учебной базы данных студента с кодом 3.

DELETE FROM Students

WHERE StNo = 3

Хотя предложение WHERE в операторе DELETE является необязательным, оно присутствует почти всегда. Если же оно отсутствует, то удаляются все строки целевой таблицы.

Иногда отбор строк необходимо производить, опираясь на данные из других таблиц. Для этого можно использовать вложенные запросы.

## Обновление существующих данных (Оператор UPDATE)

Оператор UPDATE, обновляет значения одного или нескольких столбцов в выбранных строках одной таблицы.

UPDATE имя\_таблицы

SET имя\_столбца = выражение,...

[WHERE условие\_поиска]

В операторе указывается целевая таблица, которая должна быть модифицирована, при этом пользователь должен иметь разрешение на обновление таблицы и каждого конкретного столбца. Предложение WHERE отбирает строки таблицы, подлежащие обновлению. В предложении SET указывается, какие столбцы должны быть обновлены, и для них задаются новые значения.

Например, Изменить фамилию студента ″Петрова″ с кодом 2 на фамилию ″Петренко″:

UPDATE Students

SET StName = 'Петренко'

WHERE StNo = 2

Условия поиска, которые могут быть заданы в предложении WHERE оператора UPDATE, в точности соответствуют условиям поиска, доступным в операторах SELECT и DELETE.

Как и оператор DELETE, оператор UPDATE может одновременно обновить несколько строк, соответствующих условию поиска.

Предложение WHERE в операторе UPDATE является необязательным. Если оно опущено, то обновляются все строки целевой таблицы.

В операторе UPDATE, так же как и в операторе DELETE, вложенные запросы могут играть важную роль, поскольку они дают возможность отбирать строки для обновления, опираясь на информацию из других таблиц.

## Определение структуры данных в SQL

В реляционной базе данных наиболее важным элементом ее структуры является таблица. В многопользовательской базе данных основные таблицы, как правило, создает администратор, а пользователи просто обращаются и ним в процессе работы. При работе с базой данных часто оказывается удобным создавать собственные таблицы и хранить в них собственные данные, а также данные, прочитанные из других таблиц. Такие таблицы могут быть временными и существовать только в течение одного интерактивного SQL‑сеанса, но могут сохраняться и в течение нескольких недель или даже месяцев.

### Создание таблицы (оператор CREATE TABLE)

Оператор CREATE TABLE, синтаксис которого изображен ниже, определяет новую таблицу. Различные предложения оператора задают элементы определений таблицы. следует также помнить, что не все параметры стандарта SQL2 присутствуют во всех СУБД.

После выполнения оператора CREATE TABLE появляется новая таблица, которой присваивается имя, указанное в операторе. Имя таблицы должно быть идентификатором, допустимым в SQL, и не должно конфликтовать с именами существующих таблиц.

CREATE TABLE имя\_таблицы(

определение столбцов

имя\_столбца тип\_данных [DEFAULT значение] [NOT NULL],…

определение первичного ключа

PRIMARY KEY(имя\_столбца,…),

определение внешних ключей

FOREIGN KEY имя\_ограничения (имя\_столбца,…)

REFERENCE имя\_таблицы

[ON DELETE CASCADE | SET NULL | SET DEFAULT | NO ACTION]

[ON UPDATE CASCADE | SET NULL | SET DEFAULT | NO ACTION],

условие уникальности данных

UNIQUE(имя\_столбца,…),

условие проверки

CHECK(условие\_поиска)

)

Определения столбцов. Столбцы новой таблицы задаются в операторе CREATE TABLE. Определения столбцов представляют собой заключенный в скобки список, элементы которого отделены друг от друга запятыми. Порядок следования определений столбцов в списке соответствует расположению столбцов в таблице. Каждое такое определение содержит следующую информацию:

1. Имя столбца, которое используется для ссылки на столбец в оператора SQL. Каждый столбец в таблице должен иметь уникальное имя, но' разных таблицах имена столбцов могут совпадать.
2. Тип данных столбца, показывающий, данные какого вида хранятся в столбце.
3. Указание на то, обязательно ли столбец должен содержать данные. Предложение NOT NULL предотвращает занесение в столбец значений NULL, в противном случае значения NULL допускаются.
4. Значение по умолчанию для столбца СУБД, которое заносится в столбец в том случае, если в операторе INSERT для таблицы не определено значение данного столбца.

Определения первичного и внешнего ключей. Кроме определений столбцов таблицы, в операторе CREATE TABLE указывается информация о первичном ключе таблицы и о ее связях с другими таблицами базы данных Эта информация содержится в предложениях PRIMARY KEY и FOREIGN KEY.

В предложении PRIMARY KEY задается столбец или столбцы, которые образуют первичный ключ таблицы. Этот столбец (или комбинация столбцов) служит в качестве уникального идентификатора строк таблицы. СУБД автоматически следит за тем, чтобы первичный ключ каждой строки таблицы имел уникальное значение. Кроме того, в определениях столбцов первичного ключа должно быть указано, что они не могут содержать значения NULL .

В предложении FOREIGN KEY задается внешний ключ таблицы и определяется связь, которую он создает для нее с другой таблицей (таблицей-предком). В нем указываются:

1. столбец или столбцы создаваемой таблицы, которые образуют внешний ключ;
2. таблица, связь с которой создает внешний ключ. Это таблица-предок;
3. необязательное имя для этого отношения; оно не используется в операторах SQL, но может появляться в сообщениях об ошибках и потребуется в дальнейшем, если будет необходимо удалить внешний ключ;
4. как СУБД должна обращаться со значениями NULL в одном или нескольких столбцах внешнего ключа при связывании его со строками таблицы-предка;
5. необязательное правило удаления для данного отношения (CASCADE, SET NULL, SET DEFAULT или NO ACTION), которое определяет действие, предпринимаемое при удалении строки-предка;
6. необязательное правило обновления для данного отношения, которое определяет действие, предпринимаемое при обновлении первичного ключа в строке-предке;
7. необязательное условие проверки, которое ограничивает данные в таб лице так, чтобы они отвечали определенному условию поиска.

Например, создать таблицу Students.

CREATE TABLE Students (

StNo INT NOT NULL,

GrNo INT NOT NULL,

StName CHAR(30) NOT NULL,

CityNo INT,

PRIMARY KEY (StNo),

FOREIGN KEY Students\_Groups(GrNo)

REFERENCES Groups,

FOREIGN KEY Students\_Cities(CityNo)

REFERENCES Cities,

CHECK(CHAR\_LENGTH(StName)>10)

)

### Удаление таблицы (оператор DROP TABLE)

Таблицы можно удалить из базы данных посредством оператора DROP TABLE.

DROP TABLE имя\_таблицы CASCADE | RESTRICT

Оператор содержит имя удаляемой таблицы. Обычно пользователь удаляет одну из своих собственных таблиц и указывает в операторе неполное. имя таблицы. Имея соответствующее разрешение, можно также удалите таблицу другого пользователя, но в этом случае необходимо указывать полное имя таблицы.

Стандарт SQL2 требует, чтобы оператор DROP TABLE включал в себя либо параметр CASCADE, либо RESTRICT, которые определяют, как влияет удаление таблицы на другие объекты базы данных. Если задан параметр RESTRICT и в базе данных имеются объекты, которые содержат ссылку на удаляемую таблицу, то выполнение оператора DROP TABLE закончится неуспешно. В большинстве коммерческих СУБД допускается применение оператора DROP TABLE без каких-либо параметров.

### Изменение определения таблицы (оператор ALTER TABLE)

В процессе работы с таблицей иногда возникает необходимость добавить в таблицу некоторую информацию. Для изменения структуры таблицы используется оператор ALTER TABLE, с помощью которого возможно:

1. добавить в таблицу определение столбца;
2. изменить значение по умолчанию для какого-либо столбца;
3. добавить или удалить первичный ключ таблицы;
4. добавить или удалить новый внешний ключ таблицы;
5. добавить или удалить условие уникальности;
6. добавить или удалить условие проверки.

ALTER TABLE имя\_таблицы

 ADD определение\_столбца

 ALTER имя\_столбца SET DEFAULT значение | DROP DEFAULT

 DROP имя\_столбца CASCADE | RESTRICT

 ADD определение\_первичного\_ключа

 ADD определение\_внешнегого\_ключа

 ADD условие\_уникальности\_данных

 ADD условие\_проверки

 DROP CONSTRAINT имя\_ограничения CASCADE | RESTRICT

### Определения доменов

Для содержимого отдельного столбца таблицы в реляционной базе данных существует ограничение: значения столбца во всех строках таблицы должны быть одного и того же типа. Например, все названия городов в столбце CityName таблицы Sities являются символьными строками переменной длины.

В стандарте SQL2 формально определение домена реализовано как часть определения базы данных. Согласно этому стандарту, домен является именованной совокупностью значений данных и широко используется в определении базы данных как дополнительный тип данных. Домен создается посредством оператора CREATE DOMAIN на основе базовых типов данных. После создания домена на него можно ссылаться внутри определения таблицы как на тип данных.

### Индексы (операторы CREATE/DROP INDEX)

Одним из структурных элементов физической памяти, присутствующих в большинстве реляционных СУБД, является индекс – средство обеспечивающее быстрый доступ к строкам таблицы на основе значения одного или нескольких столбцов.

Для создания и удаления индексов применяются операторы CREATE INDEX и DROP INDEX, синтаксис которых описан ниже.

Следует также заметить, что СУБД автоматически создает индексы для первичных ключей.

CREATE [UNIQUE] INDEX имя\_индекса ON имя\_таблицы (имя\_столбца [ASC | DESC],…)

DROP INDEX имя\_индекса

## Понятие представления.

Когда СУБД встречает в операторе SQL ссылку на представление, она отыскивает его определение, сохраненное в базе данных. Затем СУБД преобразует пользовательский запрос, ссылающийся на представление, в эквивалентный запрос к исходным таблицам представления, и выполняет этот запрос. Таким образом, СУБД создает иллюзию существования представления в виде отдельной таблицы и в то же время сохраняет целостность исходных таблиц.

### Преимущества представлений

Применение представлений в базах данных различных типов может оказаться полезным в самых разнообразных ситуациях. В базах данных на персональных компьютерах представления применяются для удобства и позволяют упрощать запросы к базе данных. В промышленных базах данных представления играют главную роль в создании собственной структуры базы данных для каждого пользователя и обеспечении ее безопасности. Основные преимущества представлений перечислены ниже.

1. Безопасность. Каждому пользователю можно разрешить доступ к небольшому числу представлений, содержащих только ту информацию, которую ему позволено знать. Таким образом, можно осуществить ограничение доступа пользователей к хранимой информации.
2. Простота запросов. С помощью представления можно прочитать данные из нескольких таблиц и представить их как одну таблицу, превращая тем самым запрос ко многим таблицам в однотабличный запрос к представлению.
3. Простота структуры. С помощью представлений для каждого пользователя можно создать собственную "структуру" базы данных, определив ее как множество доступных пользователю виртуальных таблиц.
4. Защита от изменений. Представление может возвращать непротиворечивый и неизменный образ структуры базы данных, даже если исходные таблицы разделяются, реструктуируются или переименовываются.
5. Целостность данных. Если доступ к данным или ввод данных осуществляется с помощью представления, СУБД может автоматически проверять, выполняются ли определенные условия целостности.

### Недостатки представлений

Наряду с перечисленными выше преимуществами, представления обладают и двумя существенными недостатками;

1. Производительность. Представление создает лишь видимость существования соответствующей таблицы, и СУБД приходится преобразовывать запрос к представлению в запрос к исходным таблицам. Если представление отображает многотабличный запрос, то простой запрос к представлению становится сложным объединением и на его выполнение может потребоваться много времени.
2. Ограничения на обновление. Когда пользователь пытается обновить строки представления, СУБД должна установить их соответствие строкам исходных таблиц, а также обновить последние. Это возможно только для простых представлений; сложные представления обновлять нельзя, они доступны только для чтения.

Указанные недостатки означают, что не стоит без разбора применять представления вместо исходных таблиц. В каждом конкретном случае необходимо учитывать перечисленные преимущества и недостатки представлений.

## Представления в SQL.

Оператор CREATE VIEW, синтаксис которого изображен ниже, используется для создания представлений. В нем указываются имя представления и запрос, лежащий в его основе. Для успешного создания представления необходимо иметь разрешение на доступ ко всем таблицам, входящим в запрос.

CREATE VIEW имя\_представления (имя\_столбца,…) AS запрос

При необходимости в операторе CREATE VIEW можно задать имя для каждого столбца создаваемого представления. Если указывается список имен столбцов, то он должен содержать столько элементов, сколько столбцов содержится в запросе. Обратите внимание на то, что задаются только имена. Например, создать представление, содержащее фамилии студентов группы A-98-51.

CREATE VIEW GroupA98 (Name) AS

SELECT StName

FROM Students INNER JOIN Groups ON Students.GrNo = Groups.GrNo

WHERE Groups.GrName = 'A-98-51'

### Обновление представлений и стандарт ANSI/ISO

В стандарте SQL1 точно указано, для каких представлений в базе данных должна существовать возможность обновления. Согласно стандарту, представление можно обновлять в том случае, если определяющий его запрос соответствует следующим требованиям:

1. Должен отсутствовать предикат DISTINCT, т.е. повторяющиеся строки не должны исключаться из таблицы результатов запроса.
2. В предложении FROM должна быть задана только одна таблица, которую можно обновлять; т.е. у представления должна быть одна исходная таблица, а пользователь должен иметь соответствующие права доступа к ней. Если исходная таблица сама является представлением, то оно также должно удовлетворять этим условиям.
3. Каждое имя в списке возвращаемых столбцов должно быть ссылкой на простой столбец; в этом списке не должны содержаться выражения, вычисляемые столбцы или агрегатные функции.
4. Предложение WHERE не должно содержать вложенный запрос; в нем могут присутствовать только простые условия поиска.
5. В запросе не должно содержаться предложение GROUP BY или HAVING.

Правила, установленные в стандарте SQL1, являются очень жесткими. Существует множество представлений, которые теоретически можно обновлять, хотя они не соответствуют этим правилам. Кроме того, существуют представления, над которыми можно производить не все операции обновления, а также представления, в которых можно обновлять не все столбцы. В большинстве коммерческих СУБД правила обновления представлений значительно менее строги, чем в стандарте SQL1.

### Удаление представления (оператор DROP VIEW)

В стандарте SQL2 было формально закреплено использование оператора DROP VIEW для удаления представлений. В нем также детализированы правила удаления представлений, на основе которых были созданы другие представления.

DROP VIEW имя\_представления CASCADE | RESTRICT

## Системный каталог (самостоятельное изучение)

### Понятие системный каталог

Системным каталогом называется совокупность специальных таблиц базы данных. Их создает, сопровождает и владеет ими сама СУБД. Эти системные таблицы содержат информацию, которая описывает структуру базы данных. Таблицы системного каталога создаются автоматически при создании базы данных. Обычно они объединяются под специальным "системным идентификатором пользователя" с таким именем, как SYSTEM, SYSTEM, MASTER или DBA.

При обработке операторов SQL СУБД постоянно обращается к данным системного каталога. Например, чтобы обработать двухтабличный оператор SELECT, СУБД должна:

1. проверить, существуют ли две указанные таблицы;
2. убедиться, что пользователь имеет разрешение на доступ к ним;
3. проверить, существуют ли столбцы, на которые имеются ссылки в данном запросе;
4. установить, к каким таблицам относятся неполные имена столбцов;
5. определить тип данных каждого столбца.

Если бы системные таблицы служили только для удовлетворения внутренних потребностей СУБД, то для пользователей базы данных они не представляли бы практически никакого интереса. Однако системные таблицы, как правило, доступны также и для пользователей – запросы к системным каталогам разрешены почти во всех СУБД для персональных компьютеров и больших ЭВМ. С помощью запросов к системным каталогам вы можете получить информацию о структуре базы данных, даже если никогда раньше с ней не работали.

Пользователи могут только считывать информацию из системного каталога. СУБД запрещает пользователям модифицировать системные таблицы непосредственно, так как это может нарушить целостность базы данных. СУБД сама вставляет, удаляет и обновляет строки системных таблиц во время модификации структуры базы данных. Изменения в системных таблицах происходят также в качестве побочного результата выполнения таких операторов DDL, как CREATE, ALTER, DROP, GRANT И REVOKE.

### Системный каталог и стандарт ANSI/ISO

В стандарте SQL1 ничего не говорится о структуре и содержании системного каталога. Стандарт фактически не требует даже наличия самого системного каталога. Однако во всех основных реляционных СУБД в той или иной форме он создается. Структура каталога и содержащиеся в нем таблицы значительно отличаются друг от друга в разных СУБД.

В связи с растущим значением инструментальных программ общего назначения, предназначенных для работы с базами данных и требующих доступа к системному каталогу, в стандарт SQL2 включена спецификация набора представлений, обеспечивающая стандартизированный доступ к информации, которая обычно содержится в системном каталоге. СУБД, соответствующая стандарту SQL2, должна поддерживать эти представления, (обозначаемые все вместе как INFORMATION\_SCHEMA (информационная схема).

### Содержимое системного каталога

Каждая таблица системного каталога содержит информацию об отдельном структурном элементе базы данных. В состав почти всех коммерческих реляционных СУБД входят, с небольшими различиями, системные таблицы, каждая из которых описывает один из следующих пяти элементов:

1. Таблицы. В каталоге описывается каждая таблица базы данных: указывается ее имя, владелец, число содержащихся в ней столбцов, их размер и т.д.
2. Столбцы. В каталоге описывается каждый столбец базы данных: приводится имя столбца, имя таблицы, которой он принадлежит, тип данных столбца, его размер, разрешены ли значения null и т.д.
3. Пользователи. В каталоге описывается каждый зарегистрированный пользователь базы данных: указываются имя пользователя, его пароль в зашифрованном виде и другие данные.
4. Представления. В каталоге описывается каждое представление, имеющееся в базе данных: указываются его имя, имя владельца, запрос, являющийся определением представления, и т.д.
5. Привилегии. В каталоге описывается каждый набор привилегий, предоставленных в базе данных: приводятся имена тех, кто предоставил привилегии, и тех, кому они предоставлены, указываются сами привилегии, объект, на которые они распространяются, и т.д.

### Информационная схема в стандарте SQL2

В стандарте SQL2 не определена форма системного каталога, которую должны поддерживать реляционные СУБД. Поскольку в то время, когда принимался стандарт SQL2, уже существовал широкий разброс характеристик коммерческих СУБД различных типов и огромные различия в их системных каталогах, было невозможно достичь согласия по вопросу стандартной спецификации системного каталога. Вместо этого авторы стандарта дали определение "идеализированного" системного каталога, который поставщики СУБД могли бы применять при разработке "с нуля" СУБД, соответствующих стандарту SQL2. Таблицы этого идеализированного системного каталога (который в стандарте называется схема определений) приведены в табл. 11.1.

табл. 11.1 Идеализированный системный каталог, описанный в стандарте SQL2

| Системная таблица | Содержимое |
| --- | --- |
| USERS | Одна строка для каждого идентификатора пользователя в каталоге |
| SCHEMATA  | Одна строка для каждой информационной схемы в каталоге |
| DATA\_TYPE\_DESCRIPTOR | Одна строка для каждого домена или столбца, имеющего какой-то тип данных |
| DOMAINS | Одна строка для каждого домена |
| DOMAIN\_CONSTRAINTS | Одна строка для каждого ограничительного условия, наложенного на домен |
| TABLES | Одна строка для каждой таблицы или представления  |
| VIEWS | Одна строка для каждого представления |
| COLUMNS | Одна строка для каждого столбца в каждом определении таблицы или представления |
| VIEW\_TABLE\_USAGE | Одна строка для каждой таблицы, на которую имеется ссылка в каком-либо определении представления (если определением представления является многотабличный запрос, каждая таблица будет представлена отдельной строкой) |
| VIEW\_COLUMN\_USAGE | Одна строка для каждого столбца, на который имеется ссылка в каком-либо представлении |
| TABLE\_CONSTRAINTS | Одна строка для каждого ограничительного условия, заданного в каком-либо определении таблицы |
| KEY\_COLUMN\_USAGE | Одна строка для каждого столбца, на который наложено условие уникальности и который присутствует в определении первичного или внешнего ключа (если в определении ключа или условия уникальности указано несколько столбцов, то это определение будет представлено несколькими строками) |
| REFERENTIAL\_CONSTRAINTS | Одна строка для каждого определения внешнего ключа, присутствующего в определении таблицы |
| CHECK\_CONSTRAINTS | Одна строка для каждого условия проверки, заданного в определении таблицы |
| CHECK\_TABLE\_USAGE | Одна строка для каждой таблицы, на которую имеется ссылка в условии проверки, ограничительном условии для домена или утверждении |
| CHECK\_COLUMN\_USAGE | Одна строка для каждого столбца, на который имеется ссылка в условии проверки, ограничительном условии для домена или утверждении |
| ASSERTIONS | Одна строка для каждого заданного утверждения |
| TABLE\_PRIVILEGES | Одна строка для каждой привилегии, предоставленной на какую-либо таблицу |
| COLUMN\_PRIVILEGES | Одна строка для каждой привилегии, предоставленной на какой-либо столбец |
| USAGE\_PRIVILEGES | Одна строка для каждой привилегии, предоставленной на какой-либо домен, набор символов и т.п. |
| CHARACTER\_SETS | Одна строка для каждого заданного набора символов |
| COLLATIONS | Одна строка для каждой заданной последовательности сравнения |
| TRANSLATIONS | Одна строка для каждого заданного преобразования |
| SQL\_LANGUAGES | Одна строка для каждого языка (например, COBOL, С и т.д.), поддерживаемого СУБД данного типа |

Стандарт SQL2 не требует, чтобы СУБД поддерживали таблицы системного каталога, приведенные в табл. 11.1 или какие-либо иные. Вместо этого в стандарте SQL2 определен ряд представлений, основанных на этих системных таблицах. Данные представления содержат те объекты базы данных, которые должны быть доступны для рядового пользователя. (Эти представления системного каталога называются в стандарте информационной схемой). Для того чтобы СУБД соответствовала стандарту SQL2, она должна поддерживать эти представления. Такой подход дает пользователю стандартный способ получения информации о доступных ему объектах базы данных с помощью стандартных запросов к представлениям системного каталога.

На практике коммерческие реляционные СУБД поддерживают стандартные представления каталога путем создания соответствующих представлений на основе таблиц своих собственных системных каталогов. Информация в системных каталогах большинства СУБД достаточно близка к требуемой в стандарте, поэтому определения стандартных представлений каталога, создаваемых в этих СУБД, будут относительно простыми.

Представления системного каталога, требуемые стандартом SQL2, приведены в табл. 11.2. В ней дается краткое описание информации, которая содержится в каждом представлении. В стандарте определены также три домена, которые .используются представлениями системного каталога и являются доступными для пользователей. Эти домены приведены в табл. 11.3.

табл. 11.2 Представления системного каталога, установленные стандартом SQL2

| Представление в системном каталоге | Содержимое |
| --- | --- |
| INFORMATION\_SСНЕМА\_CATALOG\_NAME | Одна строка с именем базы данных для каждого пользователя ("каталога" по терминологии стандарта SQL2), описываемого данной информационной схемой |
| SCHEMATA | Одна строка для каждой информационной схемы в базе данных, принадлежащей текущему пользователю; содержит имя схемы, набор символов по умолчанию и т.д.  |
| DOMAINS | Одна строка для каждого домена, доступного текущему пользователю; содержит имя домена, базовый тип данных, набор символов, максимальную длину, степень, точность и т.д.  |
| DOMAIN\_CONSTRAINTS | Одна строка для каждого ограничительного условия домена; содержит имя условия и его характеристики  |
| TABLES | Одна строка для каждой таблицы или представления, доступных пользователю; содержит имя и признак того, идет ли речь о таблице или представлении  |
| VIEWS | Одна строка для каждого представления, доступного пользователю; содержит имя, информацию о режиме контроля и возможности обновления.  |
| COLUMNS | Одна строка для каждого столбца, доступного пользователю; содержит имя столбца, имя таблицы или представления, которые содержат данный столбец, тип содержащихся в нем данных, степень, точность, набор символов и т.д. |
| TABLE\_PRIVILEGES | Одна строка для .каждой привилегии на таблицу, предоставленной пользователю или предоставленной им другому пользователю; содержит имя таблицы, тип привилегии, указание на то, кто предоставил привилегию, кому она предоставлена и имеет ли пользователь право предоставления этой привилегии  |
| COLUMN\_PRIVILEGES | Одна строка для каждой привилегии на столбец, предоставленной пользователю или предоставленной им другому пользователю; содержит имя таблицы и столбца, тип привилегии, указание на то, кто предоставил привилегию, кому она предоставлена и имеет ли пользователь право предоставления этой привилегии  |
| USAGE\_PRIVILEGES | Одна строка для каждой привилегии, предоставленной пользователю или пользователем на какой-либо домен, набор символов и т.п.  |
| TABLE\_CONSTRAINTS | Одна строка на каждое ограничительное условие (первичный ключ, внешний ключ, условие уникальности или условие проверки), заданное для таблицы, которой владеет пользователь; содержит имя условия и таблицы, тип условия и его характеристики  |
| REFERENTIAL\_CONSTRAINTS | Одна строка для каждого ссылочного ограничения (определения внешнего ключа) на таблицу, которой владеет пользователь; содержит имя ограничения, имя таблицы-потомка и имя таблицы-предка |
| CHECK\_\_CONSTRAINTS | Одна строка на каждое условие проверки для таблицы, которой владеет пользователь |
| KEY\_COLUMN\_USAGE | Одна строка для каждого столбца первичного или внешнего ключа, на который (столбец) наложено ), условие уникальности и который входит в таблицу, принадлежащую пользователю; строка содержит имя таблицы, имя столбца и позицию столбца в ключе |
| ASSERTIONS | Одна строка для каждого утверждения, которым владеет пользователь; содержит имя утверждения и его характеристики |
| CHARACTER\_SETS | Одна строка для каждого определения набора символов, доступного пользователю |
| COLLATIONS | Одна строка для каждого определения последовательности сравнения, доступного пользователю |
| TRANSLATIONS | Одна строка для каждого определения преобразования, доступного пользователю |
| VIEW\_TABLE\_USAGE | Одна строка для каждой таблицы, на которую имеется ссылка в определениях представлений, принадлежащих пользователю; строка содержит имя таблицы |
| VIEW\_COLUMN\_USAGE | Одна строка для каждого столбца, на который имеется ссылка в представлениях, принадлежащих пользователю; строка содержит имя столбца и таблицы, в которую входит столбец |
| CONSTRAINT\_TABLE\_USAGE | Одна строка для каждой таблицы, на которую имеется ссылка в условии проверки, условии уникальности, утверждении и определении внешнего ключа, принадлежащих пользователю |
| CONSTRAINT\_COLUMN\_USAGE | Одна строка для каждого столбца, на который имеется ссылка в условии проверки, условии уникальности, утверждении и определении внешнего ключа, принадлежащих пользователю |
| SQL\_LANGUAGES | Одна строка для каждого языка (например, COBOL, С и т.д.), поддерживаемого СУБД данного типа; в строке указывается уровень соответствия языка стандарту SQL2, тип поддерживаемого диалекта SQL и т.д. |

табл. 11.3 Домены, определенные в стандарте SQL2

| Системный домен | Область значений домена |
| --- | --- |
| SQL\_IDENTIFIER | Домен всех символьных строк переменной длины, которые являются допустимыми идентификаторами SQL согласно стандарту SQL2. Любое значение, взятое из этого (домена, является допустимым именем таблицы, именем столбца и т.д. |
| CHARACTER\_DATA | Домен всех символьных строк переменной длины, имеющих длину от нуля до максимального значения, поддерживаемого данной СУБД. Значение, взятое из этого домена, является допустимой символьной строкой. |
| CARDINAL\_NUMBER | Домен всех неотрицательных чисел от нуля до максимального целого числа, с которым может работать данная СУБД. Значение, взятое из этого домена, является нулем или допустимым положительным числом. |

Вот примеры нескольких запросов, используемых для извлечения информации о структуре базы данных из представлений системного каталога, определенных в стандарте SQL2:

1. Вывести имена всех таблиц и представлений пользователя, работающего в настоящий момент с базой данных.

SELECT TABLE\_NAME

FROM TABLES

1. Вывести имя, позицию и тип данных для каждого столбца во всех представлениях.

SELECT TABLE\_NAME, С.COLUMN\_NAME, ORDINAL\_POSITION, DATAJTYPE

FROM COLUMNS

WHERE (COLUMNS.TABLE\_NAME IN (SELECT TABLE\_NAME FROM VIEWS))

1. Определить, сколько столбцов имеется в таблице STUDENTS.

SELECT COUNT(\*)

FROM COLUMNS

WHERE (TABLE\_NAME = 'STUDENTS')

**Литература:**

1. Джеймс Р. Грофф, Пол Н. Вайнберг. SQL: полное руководство: пер.с англ. –К.: Издательская группа BHV, 2000.–608с. Стр. 295–346.

# Обеспечение безопасности БД

12.1 Общие положения

12.2 Методы обеспечения безопасности

12.3 Избирательное управление доступом

12.4 Обязательное управление доступом

12.5 Шифрование данных

12.6 Контрольный след выполняемых операций

12.7 Поддержка мер обеспечения безопасности в языке SQL

12.8 Директивы GRANT и REVOKE

12.9 Представления и безопасность

## Общие положения

Термины безопасность и целостность в контексте обсуждения баз данных часто используется совместно, хотя на самом деле, это совершенно разные понятия. Термин безопасность относится к защите данных от несанкционированного доступа, изменения или разрушения данных, а целостность – к точности или истинности данных. По-другому их можно описать следующим образом:

1. под безопасностью подразумевается, что пользователям разрешается выполнять некоторые действия;
2. под целостностью подразумевается, что эти действия выполняются корректно.

Между ними есть, конечно, некоторое сходство, поскольку как при обеспечении безопасности, так и при обеспечении целостности система вынуждена проверить, не нарушают ли выполняемые пользователем действия некоторых правил. Эти правила должны быть заданы (обычно администратором базы данных) на некотором удобном для этого языке и сохранены в системном каталоге. Причем в обоих случаях СУБД должна каким-то образом отслеживать все действия пользователя и проверять их соответствие заданным правилам.

Среди многочисленных аспектов проблемы безопасности необходимо отметить следующие:

1. Правовые, общественные и этические аспекты (имеет ли право некоторое лицо получить запрашиваемую информацию, например об оценках студента).
2. Физические условия (например, закрыт ли данный компьютер или терминальная комната или защищен каким-либо другим образом).
3. Организационные вопросы (например, как в рамках предприятия, обладающего некой системой, организован доступ к данным).
4. Вопросы реализации управления (например, если используется метод доступа по паролю, то как организована реализация управления и как часто меняются пароли).
5. Аппаратное обеспечение (обеспечиваются ли меры безопасности на аппаратном уровне, например, с помощью защитных ключей или привилегированного режима управления).
6. Безопасность операционной системы (например, затирает ли базовая операционная система содержание структуры хранения и файлов с данными при прекращении работы с ними).
7. И наконец, некоторые вопросы, касающиеся непосредственно самой системы управления базами данных (например, существует ли для базы данных некоторая концепция предоставления прав владения данными).

В настоящей лекции рассматриваются вопросы, касающиеся последнего пункта этого перечня.

## Методы обеспечения безопасности

В современных СУБД поддерживается один из двух широко распространенных подходов к вопросу обеспечения безопасности данных, а именно избирательный подход или обязательный подход. В обоих подходах единицей данных или "объектом данных", для которых должна быть создана система безопасности, может быть как вся база данных целиком или какой-либо набор отношений, так и некоторое значение данных для заданного атрибута внутри некоторого кортежа в определенном отношении. Эти подходы отличаются следующими свойствами:

1. В случае избирательного управления некий пользователь обладает различными правами (привилегиями или полномочиями) при работе с разными объектами. Более того, разные пользователи обычно обладают и разными правами доступа к одному и тому же объекту. Поэтому избирательные схемы характеризуются значительной гибкостью.
2. В случае обязательного управления, наоборот, каждому объекту данных присваивается некоторый классификационный уровень, а каждый пользователь обладает некоторым уровнем допуска. Следовательно, при таком подходе доступом к опре­деленному объекту данных обладают только пользователи с соответствующим уровнем допуска. Поэтому обязательные схемы достаточно жестки и статичны.

Независимо от того, какие схемы используются – избирательные или обязательные, все решения относительно допуска пользователей к выполнению тех или иных операций принимаются на стратегическом, а не техническом уровне. Поэтому они находятся за пределами досягаемости самой СУБД, и все, что может в такой ситуации сделать СУБД, – это только привести в действие уже принятые ранее решения. Исходя из этого, можно отметить следующее:

Во-первых. Результаты стратегических решений должны быть известны системе (т.е. выполнены на основе утверждений, заданных с помощью некоторого подходящего языка) и сохраняться в ней (путем сохранения их в каталоге в виде правил безопасности, которые также называются полномочиями).

Во-вторых. Очевидно, должны быть некоторые средства регулирования запросов доступа по отношению к соответствующим правилам безопасности. (Здесь под "запросом, доступа" подразумевается комбинация запрашиваемой операции, запрашиваемого, объекта и запрашивающего пользователя.) Такая проверка выполняется подсистемой безопасности СУБД, которая также называется подсистемой полномочий.

В-третьих. Для того чтобы разобраться, какие правила безопасности к каким запросам доступа применяются, в системе должны быть предусмотрены способы опознания источника этого запроса, т.е. опознания запрашивающего пользователя. Поэтому в момент входа в систему от пользователя обычно требуется ввести не только его идентификатор (например, имя или должность), но также и пароль (чтобы подтвердить свои права на заявленные ранее идентификационные данные). Обычно предполагается, что пароль известен только системе и некоторым лицам с особыми правами.

В отношении последнего пункта стоит заметить, что разные пользователи могут обладать одним и тем же идентификатором некоторой группы. Таким образом, в системе могут поддерживаться группы пользователей и обеспечиваться одинаковые права доступа для пользователей одной группы, например для всех лиц из расчетного отдела. Кроме того, операции добавления отдельных пользователей в группу или их удаления из нее могут выполняться независимо от операции задания привилегий для этой группы. Обратите внимание, однако, что местом хранения информации о принадлежности к группе также является системный каталог (или, возможно, база данных).

Перечисленные выше методы управления доступом на самом деле являются частью более общей классификации уровней безопасности. Прежде всего в этих документах определяется четыре класса безопасности (security classes) – D, С, В и А. Среди них класс D наименее безопасный, класс С – более безопасный, чем класс D, и т.д. Класс D обеспечивает минимальную защиту, класс С – избирательную, класс В – обязательную, а класс А – проверенную защиту.

Избирательная защита. Класс С делится на два подкласса – С1 и С2 (где подкласс С1 менее безопасен, чем подкласс С2), которые поддерживают избирательное управление доступом в том смысле, что управление доступом осуществляется по усмотрению владельца данных.

Согласно требованиям класса С1 необходимо разделение данных и пользователя, т.е. наряду с поддержкой концепции взаимного доступа к данным здесь возможно также организовать раздельное использование данных пользователями.

Согласно требованиям класса С2 необходимо дополнительно организовать учет на основе процедур входа в систему, аудита и изоляции ресурсов.

Обязательная защита. Класс В содержит требования к методам обязательного управления доступом и делится на три подкласса – В1, В2 и В3 (где В1 является наименее, а В3 – наиболее безопасным подклассом).

Согласно требованиям класса В1 необходимо обеспечить "отмеченную защиту" (это значит, что каждый объект данных должен содержать отметку о его уровне классификации, например: секретно, для служебного пользования и т.д.), а также неформальное сообщение о действующей стратегии безопасности.

Согласно требованиям класса В2 необходимо дополнительно обеспечить формальное утверждение о действующей стратегии безопасности, а также обнаружить и исключить плохо защищенные каналы передачи информации.

Согласно требованиям класса В3 необходимо дополнительно обеспечить поддержку аудита и восстановления данных, а также назначение администратора режима безопасности.

Проверенная защита. Класс А является наиболее безопасным и согласно его требованиям необходимо математическое доказательство того, что данный метод обеспечения безопасности совместимый и адекватен заданной стратегии безопасности.

Хотя некоторые коммерческие СУБД обеспечивают обязательную защиту на уровне класса В1, обычно они обеспечивают избирательное управление на уровне класса С2.

## Избирательное управление доступом

Избирательное управление доступом поддерживается многими СУБД. Избирательное управление доступом поддерживается в языке SQL.

В общем случае система безопасности таких СУБД базируется на трех компонентах:

1. Пользователи. СУБД выполняет любое действия с БД от имени какого-то пользователя. Каждому пользователю присваивается идентификатор – короткое имя, однозначно определяющее пользователя в СУБД. Для подтверждения того, что пользователь может работать с введенным идентификатором используется пароль. Таким образом, с помощью идентификатора и пароля производится идентификация и аутентификация пользователя. Большинство коммерческих СУБД позволяет объединять пользователей с одинаковыми привилегиями в группы – это позволяет упростить процесс администрирования.
2. Объекты БД. По стандарту SQL2 защищаемыми объектами в БД являются таблицы, представления, домены и определенные пользователем наборы символов. Большинство коммерческих СУБД расширяет список объектов, добавляя в него хранимые процедуры и др. объекты.
3. Привилегии. Привилегии показывают набор действий, которые возможно производить над тем или иным объектом. Например пользователь имеет привилегию для просмотра таблицы.

## Обязательное управление доступом

Методы обязательного управления доступом применяются к базам данных, в которых данные имеют достаточно статичную или жесткую структуру, свойственную, например, правительственным или военным организациям. Как уже отмечалось, основная идея заключается в том, что каждый объект данных имеет некоторый уровень классификации, например: секретно, совершенно секретно, для служебного пользования и т.д., а каждый пользователь имеет уровень допуска с такими же градациями, что и в уровне классификации. Предполагается, что эти уровни образуют строгий иерархический порядок, например: совершенно секретно → секретно → для служебного пользования и т.д. Тогда на основе этих сведений можно сформулировать два очень простых правила безопасности:

1. пользователь имеет доступ к объекту, только если его уровень допуска больше или равен уровню классификации объекта.
2. пользователь может модифицировать объекту, только если его уровень допуска равен уровню классификации объекта.

Правило 1 достаточно очевидно, а правило 2 требует дополнительных разъяснений. Прежде всего следует отметить, что по-другому второе правило можно сформулировать так: любая информация, записанная некоторым пользователем, автоматически приобретает уровень, равный уровню классификации этого пользователя. Такое правило необходимо, например, для того, чтобы предотвратить запись секретных данных, выполняемую пользователем с уровнем допуска "секретно", в файл с меньшим уровнем классификации, что нарушает всю систему секретности.

В последнее время методы обязательного управления доступом получили широкое распространение. Требования к такому управлению доступом изложены в двух документах, которые неформально называются "оранжевой" книгой (Orange Book) и "розовой" книгой (Lavender Book). В "оранжевой" книге перечислен набор требований к безопасности для некой "надежной вычислительной базы" (Trusted Computing Base), а в "розовой" книге дается интерпретация этих требований для систем управления базами данных.

## Шифрование данных

До сих пор в этой главе подразумевалось, что предполагаемый нелегальный пользователь пытается незаконно проникнуть в базу данных с помощью обычных средств доступа, имеющихся в системе. Теперь следует рассмотреть случай, когда такой пользователь пытается проникнуть в базу данных, минуя систему, т.е. физически перемещая часть базы данных или подключаясь к коммуникационному каналу. Наиболее эффективным методом борьбы с такими угрозами является шифрование данных, т.е. хранение и передача особо важных данных в зашифрованном виде.

Для обсуждения основных концепций кодирования данных следует ввести некоторые новые понятия. Исходные (незакодированные) данные называются открытым текстом. Открытый текст шифруется с помощью специального алгоритма шифрования. В качестве входных данных для такого алгоритма выступают открытый текст и ключ шифрования, а в качестве выходных – зашифрованная форма открытого текста, которая называется зашифрованным текстом. Если детали алгоритма шифрования могут быть опубликованы или, по крайней мере, могут не утаиваться, то ключ шифрования обязательно хранится в секрете. Именно зашифрованный текст, который непонятен тем, кто не обладает ключом шифрования, хранится в базе данных и передается по коммуникационному каналу.

## Контрольный след выполняемых операций

Важно понимать, что не бывает неуязвимых систем безопасности, поскольку настойчивый потенциальный нарушитель всегда сможет найти способ преодоления всех систем контроля, особенно если за это будет предложено достаточно высокое вознаграждение. Поэтому при работе с очень важными данными или при выполнении критических операций возникает необходимость регистрации контрольного следа выполняемых операций. Если, например, противоречивость данных приводит к подозрению, что совершено несанкционированное вмешательство в базу данных, то контрольный след должен быть использован для прояснения ситуации и подтверждения того, что все процессы находятся под контролем. Если это не так, то контрольный след поможет, по крайней мере, обнаружить нарушителя.

Для сохранения контрольного следа обычно используется особый файл, в котором система автоматически записывает все выполненные пользователями операции при работе с обычной базой данных. Типичная запись в файле контрольного следа может содержать такую информацию:

1. запрос (исходный текст запроса);
2. терминал, с которого была вызвана операция;
3. пользователь, задавший операцию;
4. дата и время запуска операции;
5. вовлеченные в процесс исполнения операции базовые отношения, кортежи и атрибуты;
6. старые значения;
7. новые значения.

Как уже упоминалось ранее, даже констатация факта, что в данной системе поддерживается контрольное слежение, в некоторых случаях весьма существенна для предотвращения несанкционированного проникновения в систему.

## Поддержка мер обеспечения безопасности в языке SQL

В действующем стандарте языка SQL предусматривается поддержка только избирательного управления доступом. Она основана на двух более или менее независимых частях SQL. Одна из них называется механизмом представлений, который (как говорилось выше) может быть использован для скрытия очень важных данных от несанкционированных пользователей. Другая называется подсистемой полномочий и наделяет одних пользователей правом избирательно и динамично задавать различные полномочия другим пользователям, а также отбирать такие полномочия в случае необходимости.

## Директивы GRANT и REVOKE

Механизм представлений языка SQL позволяет различными способами разделить базу данных на части таким образом, чтобы некоторая информация была скрыта от пользователей, которые не имеют прав для доступа к ней. Однако этот режим задается не с помощью параметров операций, на основе которых санкционированные пользователи выполняют те или иные действия с заданной частью данных. Эта функция (как было показано выше) выполняется с помощью директивы GRANT.

Обратите внимание, что создателю любого объекта автоматически предоставляются все привилегии в отношении этого объекта.

Стандарт SQL1 определяет следующие привилегии для таблиц:

1. SELECT – позволяет считывать данные из таблицы или представления;
2. INSERT – позволяет вставлять новые записи в таблицу или представление;
3. UPDATE – позволяет модифицировать записи из таблицы или представления;
4. DELETE – позволяет удалять записи из таблицы или представления.

Стандарт SQL2 расширил список привилегий для таблиц и представлений:

1. INSERT для отдельных столбцов, подобно привилегии UPDATE;
2. REFERENCES – для поддержки внешнего ключа.

Помимо перечисленных выше добавлена привилегия USAGE – для других объектов базы данных.

Кроме того, большинство коммерческих СУБД поддерживает дополнительные привилегии, например:

1. ALTER – позволяет модифицировать структуру таблиц (DB2, Oracle);
2. EXECUTE – позволяет выполнять хранимые процедуры.

Создатель объекта также получает право предоставить привилегии доступа какому-нибудь другому пользователю с помощью оператора GRANT. Ниже приводится синтаксис утверждения GRANT:

GRANT {SELECT|INSERT|DELETE|(UPDATE столбец, …)}, …

ON таблица ТО {пользователь | PUBLIC} [WITH GRANT OPTION]

Привилегии вставки (INSERT) и обновления (UPDATE) (но не привилегии выбора SELECT, что весьма странно) могут задаваться для специально заданных столбцов.

Если задана директива WITH GRANT OPTION, это значит, что указанные пользователи наделены особыми полномочиями для заданного объекта – правом предоставления полномочий. Это, в свою очередь, означает, что для работы с данным объектом они могут наделять полномочиями других пользователей

Например: предоставить пользователю Ivanov полномочия для осуществления выборки и модификации фамилий в таблице Students с правом предоставления полномочий.

GRANT SELECT, UPDATE StName

ON Students ТО Ivanov WITH GRANT OPTION

Если пользователь А наделяет некоторыми полномочиями другого пользователя В, то впоследствии он может отменить эти полномочия для пользователя В. Отмена полномочий выполняется с помощью директивы REVOKE с приведенным ниже синтаксисом.

REVOKE {{SELECT | INSERT | DELETE | UPDATE},…|ALL PRIVILEGES}

ON таблица,… FROM {пользователь | PUBLIC},… {CASCADE | RESTRICT}

Поскольку пользователь, с которого снимается привилегия, мог предоставить ее другому пользователю (если обладал правом предоставления полномочий), возможно возникновение ситуации покинутых привилегий. Основное предназначение параметров RESTRICT и CASCADE заключается в предотвращении ситуаций с возникновением покинутых привилегий. Благодаря заданию параметра RESTRICT не разрешается выполнять операцию отмены привилегии, если она приводит к появлению покинутой привилегии. Параметр CASCADE указывает на последовательную отмену всех привилегий, производных от данной.

Например: снять с пользователя Ivanov полномочия для осуществления модификации фамилий в таблице Students. Также снять эту привилегию со всех пользователей, которым она была предоставлена Ивановым.

REVOKE UPDATE

ON Students FROM Ivanov CASCADE

При удалении домена, таблицы, столбца или представления автоматически будут удалены также и все привилегии в отношении этих объектов со стороны всех пользователей.

## Представления и безопасность

Создавая представления, и давая пользователям разрешение на доступ к нему, а не к исходной таблице, можно тем самым ограничить доступ пользователя, разрешив его только к заданным столбцам или записям. Таким образом, представления позволяют осуществить полный контроль над тем, какие данные доступны тому или иному пользователю.

Продемонстрируем использование представлений для обеспечения безопасности с помощью описанных на языке SQL примеров.

1. Создание представления для доступа к данным группы А–98–51.

CREATE VIEW GroupA98 AS

SELECT \*

FROM Students INNER JOIN Groups ON Students.GrNo = Groups.GrNo

WHERE Groups.GrName = 'A-98-51'

1. Предоставление полномочий пользователю Ivanov.

GRANT SELECT, INSERT, DELETE, UPDATE

ON GroupA98 ТО Ivanov

Теперь пользователь Ivanov может просматривать и модифицировать только данные группы А–98–51.

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 394 – 410.
2. Джеймс Р. Грофф, Пол Н. Вайнберг. SQL: полное руководство: пер.с англ. –К.: Издательская группа BHV, 2000.–608с. Стр. 329–368.

# Физическая организация БД: структуры хранения и методы доступа

13.1 Доступ к базе данных

13.2 Кластеризация

13.3 Индексирование

13.4 Структуры типа Б-дерева

13.5 Хеширование

## Доступ к базе данных

В этой лекции рассматриваются технологии физического хранения данных хранящимся на диске и осуществления доступа к ним. (Далее под термином диск будут подразумеваться все устройства хранения данных прямого доступа, т.е. прежде всего традиционные жесткие диски с подвижными магнитными головками, внешние запоминающие устройства большой емкости, оптические диски и др.)

Внимание специалистов к структурам хранения и методам доступа вызвано очень медленной внешней памятью. Основным способом повышения производительности является минимизация числа дисковых операций ввода-вывода данных.

Как упоминалось ранее, любое упорядочение (расположение) данных на диске называется структурой хранения. Можно организовать самые разные структуры хранения, обладающие различной производительностью и оптимальные для различных способов использования. Однако не существует идеальной структуры хранения, которая была бы оптимальна для любых задач. Исходя из этого, можно заключить, что совершенная СУБД должна содержать несколько разных структур хранения для различных частей системы. Кроме того, следует также предусмотреть возможность изменения структуры хранения по мере изменения требований к производительности системы.

Работа СУБД построена следующим образом и включает три основных этапа (рис. 13.1).

рис. 13.1 Схема взаимодействия СУБД, диспетчера файлов и диспетчера дисков.

СУБД

Диспетчер файлов

Диспетчер дисков

Хранимая БД

Возвращение хранимой записи

Запрос хранимой записи

Запрос хранимой страницы

Дисковая операция ввода/вывода

Возвращение хранимой страницы

Чтение данных с диска

1. Сначала в СУБД определяется искомая запись, а затем для ее извлечения запрашивается диспетчер файлов.
2. Диспетчер файлов определяет страницу, на которой находится искомая запись, а затем для извлечения этой страницы запрашивает диспетчер дисков.
3. 3. Наконец, диспетчер дисков определяет физическое положение искомой страницы на диске и посылает соответствующий запрос на ввод-вывод данных.

Диспетчер дисков. Диспетчер дисков является компонентом используемой операционной системы, с помощью которого выполняются все дисковые операции ввода-вывода (в некоторых операционных системах он называется компонентом базовой системы ввода-вывода).

Для выполнения этих операций необходимо знать значения физических адресов на диске. Например, если диспетчер файлов запрашивает некоторую страницу диспетчеру дисков необходимо знать, где конкретно находится страница на физическом диске. Однако "пользователю" диспетчера дисков, т.е. диспетчеру файлов, не обязательно знать физические адреса. Вместо этого диспетчеру файлов достаточно рассматривать диск как набор страниц фиксированного размера (т.е. набор "ячеек" памяти одинакового размера) с уникальным идентификационным номером набора страниц. Каждая страница, в свою очередь, обладает уникальным внутри данного набора идентификационным номером страницы, причем наборы не имеют общих страниц. Соответствие физических адресов на диске и номеров страниц достигается с помощью диспетчера дисков. Главным (и не единственным) преимуществом такой организации является изоляция программного кода, зависящего от конкретного устройства диска, внутри одного из компонентов системы, а именно внутри диспетчера дисков. В таком случае все компоненты высокого уровня, в частности диспетчера файлов, могут быть аппаратно независимыми.

Диспетчер файлов. При работе с диском как набором хранимых файлов, диспетчер файлов использует все имеющиеся средства диспетчера дисков согласно определенному в СУБД способу. При этом каждый набор страниц может содержать один или несколько хранимых файлов.

Каждый хранимый файл имеет имя (file name) или идентификационный номер (file ID), уникальные в данном наборе страниц. А каждая хранимая запись, в свою очередь, обладает идентификационным номером записи (record number или record ID), уникальным, по крайней мере, в пределах данного хранимого файла.

## Кластеризация

Нельзя завершить этот краткий обзор без упоминания технологии кластеризации данных. В ее основе лежит принцип как можно более близкого физического размещения на диске логически связанных между собой и часто используемых данных. Физическая кластеризация данных – чрезвычайно важное условие высокой производительности, что можно продемонстрировать следующим примером. Допустим, что наиболее часто используется хранимая запись r1 страницы p1, для работы с которой также требуется вызывать хранимую запись r2 страницы p2. Тогда возможно возникновение следующих ситуаций:

1. Если страницы р1 и р2 совпадают, то для доступа к записи r2 не потребуется выполнять еще одну физическую операцию ввода-вывода, поскольку нужная страница уже будет находиться в оперативной памяти.
2. Если страницы р1 и р2 не совпадают, но физически размещаются достаточно близко, например смежные страницы, то для доступа к записи r2 потребуется выполнить еще одну физическую операцию ввода-вывода (если, конечно, страница p2 еще не находится в оперативной памяти). Однако, поскольку головка чтения/записи уже будет находиться в непосредственной близости от нужного положения, время поиска будет очень малым. А если страницы р1 и р2 находятся на одном цилиндре, время поиска вообще будет равно нулю.

Внутрифайловую и межфайловую кластеризацию СУБД может осуществлять, размещая логически связанные записи на одной странице (если это возможно) или на смежных страницах (в противном случае).

Кластеризация внутри СУБД возможна только в том случае, если администратор базы данных организует ее. В совершенных СУБД часто предусмотрено задание нескольких различных типов кластеризации данных из разных файлов.

## Индексирование

Рассмотрим в качестве примера таблицу с данными о студентах, а также часто используемый и потому очень важный запрос типа "Найти всех студентов учащихся в группе X", где X – некий параметр. При таких условиях администратор базы данных может выбрать способ сохранения данных, схематически показанный на рис. 13.2. Он основан на двух хранимых файлах: файле с данными о студентах и файле с данными о группах; файлы могут размещаться в различных наборах страниц. Предполагается, что в файле групп используется упорядочение по алфавитному перечню их названий, т.е. по ключевому полю GrName (название группы) с указателями на соответствующие записи в файле поставщиков.

рис. 13.2 Индексирование файла поставщиков по полю CITY файла городов.

|  |  |  |
| --- | --- | --- |
| Файл студентов (данные) |  | Файл групп (индекс) |
| StNo | GrName | StName | City |  |  |  |
| 1 | А–98–51 | Иванов | Желтые Воды |  |  | А–98–51 |
| 4 | Б–99–51 | Стрельцов | Львов |  |  | А–98–51 |
| 2 | А–98–51 | Петров | Пятихатки |  |  | А–98–51 |
| 5 | Б–99–51 | Кузнецов | Львов |  |  | Б–99–51 |
| 3 | А–98–51 | Сидоров | Пятихатки |  |  | Б–99–51 |

Для поиска всех студентов из группы Б-99-51 можно применить следующую стратегию: найти в файле групп группу Б-99-51, а затем согласно указателям извлечь все соответствующие записи из файла студентов.

Такая стратегия будет более эффективной по сравнению с поиском в файле с данными студентов, поскольку, СУБД известна физическая последовательность записей в файле групп (поиск будет прекращен после извлечения следующей за Б-98-51 названия группы в алфавитном порядке). Кроме того, даже если придется просмотреть файл групп полностью, для такого поиска потребуется гораздо меньше операций ввода-вывода, поскольку физический размер файла групп меньше, чем размер файла с данными студентов из-за меньшего размера записей.

В рассматриваемом примере файл групп называется индексным файлом или индексом по отношению к файлу студентов, и наоборот, файл студентов индексирован (называется индексированным файлом) по отношению к файлу групп.

Индексный файл – это хранимый файл особого типа, в котором каждая запись состоит из двух значений, а именно данных и указателя. Данные соответствуют некоторому полю (индексному полю) из индексированного файла, а указатель служит для связывания с соответствующей записью индексированного файла. Индексное поле также называется индексным ключом (index key).

Индекс можно сравнить с предметным указателем обычной книги, который состоит из списка слов с "указателями" (номерами страниц) для упрощения поиска связанной с этими словами информации из "индексированного файла" (т.е. из содержимого книги).

Основным преимуществом использования индексов является значительное ускорение процесса выборки или извлечения данных, а основным недостатком – замедление процесса обновления данных, поскольку при каждом добавлении новой записи в индексированный файл потребуется также добавить новый индекс в индексный файл.

Хранимый файл может иметь несколько индексов, которые могут как раздельно, так и совместно использоваться для более эффективного доступа к записям о поставщиках.

Индексы часто называют инвертированными списками. Дело в том, что если файл студентов (см. рис. 13.2) имеет традиционную структуру списка набора значений полей для каждой записи, то индекс содержит список набора записей для каждого значения индексированного поля.

Индекс можно также создать на основе комбинации двух или более полей. Например, на рис. 13.3 показана схема индексирования файла студентов на основе комбинации полей GrName и City. При такой организации в СУБД можно выполнить запрос типа "Найти студентов учащихся в группе Б-98-51 проживающих в г. Кривой Рог" на основе однократного просмотра с помощью одного индекса.

рис. 13.3 Индексирование файла поставщиков на основе комбинации полей GrName и City

|  |  |  |
| --- | --- | --- |
| Файл студентов (данные) |  | индекс GrName/City |
| StNo | GrName | StName | City |  |  |  |
| 1 | А–98–51 | Иванов | Желтые Воды |  |  | А–98–51/Желтые Воды |
| 4 | Б–99–51 | Стрельцов | Львов |  |  | А–98–51/Пятихатки |
| 2 | А–98–51 | Петров | Пятихатки |  |  | А–98–51/Пятихатки |
| 5 | Б–99–51 | Кузнецов | Львов |  |  | Б–99–51/Львов |
| 3 | А–98–51 | Сидоров | Пятихатки |  |  | Б–99–51/Львов |

Обратите внимание, что комбинированный индекс GrName/City может также служить индексом по одному полю GrName, поскольку все записи в комбинированном индексе расположены последовательно.

### Плотное и неплотное индексирование

Основной целью использования индекса является ускорение процесса извлечения данных, точнее, уменьшение числа дисковых операций ввода-вывода, необходимых для извлечения требуемой записи. В основном это достигается благодаря использованию указателей. Хотя до сих пор предполагалось, что в этом качестве используются указатели записей, на самом деле для этого достаточно было бы указателей страниц (т.е. номеров страниц). Конечно, для последующего поиска записи внутри данной страницы придется осуществить еще одну операцию извлечения записи, однако теперь она будет выполняться в оперативной памяти и для этого не придется увеличивать число дисковых операций ввода-вывода.

Эту идею можно развить дальше, если вспомнить, что данные в каждом хранимом файле находятся в единой "физической" последовательности на основе комбинации последовательности хранимых записей внутри каждой страницы и последовательности страниц внутри каждого набора страниц. Предположим, что физическая последовательность файла студентов соответствует логической последовательности, заданной на основе некоторого поля, например номера студента. Иначе говоря, в этом файле выполнена кластеризация по данному полю. Допустим, что по этому же полю осуществляется индексирование; тогда нет необходимости в данном индексе хранить указатели для каждой записи индексируемого файла (в данном случае для файла студентов). Все, что требуется, – это указатель для каждой страницы, состоящий из максимального номера студента для данной страницы и соответствующего номера страницы. Схематически такая структура показана на

рис. 13.4, где для простоты предполагается, что на каждой странице может размещаться максимум две записи.

рис. 13.4 Рис. А. 12 Пример использования неплотного индекса.

|  |  |
| --- | --- |
| Файл c данными о студентах | индекс StNo |
|  | StNo | GrName | StName | City |  |  |  |
| страницаp-1 | 1 | А–98–51 | Иванов | Желтые Воды |  |  |  |
| 2 | А–98–51 | Петров | Пятихатки |  |  |  |
| страницаp | 3 | А–98–51 | Сидоров | Пятихатки |  |  |  |
| 4 | Б–99–51 | Стрельцов | Львов |  |  |  |
| страницаp+1 | 5 | Б–99–51 | Кузнецов | Львов |  |  |  |
| 6 | … | … | … |  |  |  |

В качестве примера рассмотрим процесс извлечения записи с номером 3 с помощью такого индекса. Сначала в СУБД проводится поиск индекса для записи с номером, большим или равным 3. При этом будет найдено поле с номером 4, которое содержит указатель на страницу p. Страница p извлекается, помещается в оперативную память и просматривается для поиска заданной хранимой записи (которая в данном примере будет найдена очень быстро).

Индекс с описанной структурой называется неплотным (или разряженным), поскольку в нем не содержатся указатели на все записи индексированного файла. Схематически пример такого индекса показан на

рис. 13.4. (Все описанные выше индексы, наоборот, называются плотными.) Одним из преимуществ неплотных индексов является их малый размер по сравнению с плотными индексами, так как они содержат меньшее число записей. Это часто позволяет просматривать содержимое базы данных с большей скоростью. Однако с помощью одного только неплотного индекса нельзя выполнить проверку наличия некоторого значения.

Следует отметить, что в данном хранимом файле может быть по крайней мере один неплотный индекс, который организуется на основе (уникальной) физической последовательности, заданной в файле. А все другие индексы обязательно должны быть плотными.

## Структуры типа Б-дерева

Одним из наиболее важных и распространенных индексов является структура типа Б-дерева (B-tree).

Причина необходимости создания структуры типа Б-дерева заключается в желании избежать обязательного просмотра всего содержимого индексированного файла согласно его физической последовательности. Дело в том, что если индексированный файл имеет большой размер, то и его индекс также очень велик. Поэтому последовательный просмотр даже одного только индекса требует больших затрат времени. Разрешить эту проблему можно тем же способом, что и раньше: рассмотреть индексный файл как обычный хранимый файл и создать для него еще один индекс. Эту операцию можно осуществлять повторно нужное количество раз (обычно она применяется трижды, поскольку создание большого количества иерархических уровней индексирования требуется для очень больших файлов). При этом индекс на каждом из уровней будет неплотным по отношению к нижнему индексируемому уровню (он обязательно должен быть неплотным, иначе такая структура бессмысленна, так как уровень n содержал бы такое же количество записей, что и уровень n+1, а для просмотра потребовалось бы такое же длительное время).

Структура типа Б-дерева является частным случаем индекса древовидного типа и впервые описана в статье Байера (Вауег) и Мак-Крайта (McCreight) в 1972 году. С тех пор Байером и другими исследователями было предложено множество вариантов реализации этой идеи. В результате бинарные индексы различных типов стали широко использоваться во всех современных СУБД.

В варианте Кнута индекс состоит из двух частей:

1. Набор последовательностей включает одноуровневый индекс для реальных данных, который обычно является плотным, но может быть и неплотным, если в индексированном файле проведена кластеризация на основе индекса
2. Набор индексов, в свою очередь, обеспечивает быстрый непосредственный доступ к набору последовательностей (а значит, и к данным). По сути, набор индексов является древовидным индексным файлом для набора последовательностей или, строго говоря, индексом со структурой Б-дерева. Комбинация набора индексов и набора последовательностей называется структурой типа Б-плюс-дерева (B-plus tree или B-tree). На рис. 13.5 показан простой пример такой структуры.

Числа 6, 8, 12, ... 97, 99 являются значениями индексированного поля F. Корневой элемент содержит два значения поля F (50 и 82) и три указателя (номера страниц). Данные со значением поля F, равным или меньшим 50, могут быть найдены с помощью левого указателя; данные со значением поля F, большим 50 и равным или меньшим 82, – с помощью среднего указателя; наконец, данные со значением поля F, большим 82, – с помощью правого указателя. Другие элементы набора индексов следует интерпретировать подобным образом. Обратите внимание, что благодаря переходу на второй уровень по левому указателю в дальнейшем поиск по правому указателю будет осуществляться ко всем записям со значением поля F, большим 32 и равным или меньшим 50.

Вообще, Б-дерево порядка п содержит не менее п и не более 2п записей с данными в каждом из элементов структуры (для каждых k записей требуется также k+1 указателей). Кроме того, ни одна из записей не может использоваться двумя разными элементами.

Одним из недостатков иерархических структур является несбалансированность их работы после удаления или вставки некоторых элементов. Дело в том, что в результате таких изменений структуры элементы с реальными данными могут оказаться на разных уровнях и на разных расстояниях от корневого элемента. Поскольку во время поиска при каждом посещении элементов структуры происходит обращение к диску, общая продолжительность поиска в несбалансированной древовидной структуре может оказаться совершенно непредсказуемой.

Преимуществом структуры типа Б-дерева является возможность сбалансированной вставки или удаления значений. (Вот почему для английского написания такого индекса, "B-tree", иногда употребляют вместо символа "В" эпитет от "сбалансированный" (balanced).) Ниже приводится краткий алгоритм вставки нового значения V в структуру типа Б-дерева порядка п. Он рассчитан на вставку значения только лишь в набор индексов, но может быть достаточно просто расширен для вставки записи с данными в набор последовательностей.

1. На самом низком уровне набора индексов следует найти элемент (допустим, что это элемент N), с которым логически связано вставляемое значение V. Если элемент N содержит свободное пространство, то значение V вставляется в него и на этом процесс завершается.
2. В противном случае (если свободного пространства нет, т.е. придется создать еще один уровень) элемент N (допустим, что он содержит 2n индексных записей) разделяется на два элемента – N1 и N2. Обозначим символом 5 множество из 2n+1 значений, в котором 2n исходных значений и одно новое значение V. Тогда n первых значений этой логической (уже упорядоченной) последовательности необходимо поместить в элемент N1, n последних – в элемент N2, а среднее между ними значение W– в родительский элемент Р на более высоком структурном уровне. Впоследствии, при осуществлении поиска значения U и достижении элемента P, поиск будет перенаправлен в сторону элемента N1, если V<W, либо в сторону элемента N2, если U> W.
3. Далее этот процесс следует повторить для вставки среднего значения W в родительский элемент Р на более высоком структурном уровне.

В худшем случае процесс разделения элементов структуры может продлиться вплоть до корневого элемента всей структуры с образованием нового иерархического уровня.

Для удаления некоторого значения следует применить аналогичный алгоритм, но только в обратном порядке. А для изменения значения его можно удалить, а затем вставить новое.

рис. 13.5 Пример структуры типа Б-дерева

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 50 |  | 82 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Набор индексов |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | 12 |  | 32 |  |  |  |  |  |  |  |  |  | 58 |  | 70 |  |  |  |  |  |  |  |  |  | 89 |  | 94 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Набор последовательностей(с указателями на реальные данные) |
| 6 | 8 | 12 |  | 15 | 18 | 32 |  | 35 | 40 | 50 |  | 51 | 52 | 58 |  | 60 | 62 | 70 |  | 71 | 78 | 82 |  | 83 | 85 | 89 |  | 91 | 93 | 94 |  | 96 | 97 | 99 |  |
| ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ |  | ↓ | ↓ | ↓ |  |

## Хеширование

Хешированием, хеш-адресацией или хеш-индексированием называется технология быстрого прямого доступа к хранимой записи на основе заданного значения некоторого поля. При этом не обязательно, чтобы поле было ключевым.

Ниже перечислены основные черты этой технологии:

1. каждая хранимая запись базы данных размещается по адресу, который вычисляется с помощью специальной хеш-функции на основе значения некоторого поля данной записи, т.е. хеш-поля, или хеш-ключа. Вычисленный адрес называется хеш-адресом.
2. для сохранения записи в СУБД сначала вычисляется хеш-адрес новой записи, а затем диспетчер файлов помещает эту запись по вычисленному адресу.
3. Для извлечения нужной записи по заданному значению хеш-поля в СУБД сначала вычисляется хеш-адрес, а затем диспетчеру файлов посылается запрос для извлечения записи по вычисленному адресу.

В качестве простой иллюстрации предположим, что у нас есть записи с данными о студентах с кодами 100, 200, 300, 400, 500, а в качестве хеш-функции h используется следующая: h = StNo mod 13, где h – хеш-адрес, StNo – код студента.

Это простейший пример общего класса хеш-функций типа деление/остаток. (В качестве делителя следует выбирать простое натуральное число). В этом примере хеш-адресами для заданных записей будут 9, 5, 1, 10 и 6 соответственно. Схематически взаимное расположение записей на страницах показано на рис. 13.6.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 |  | 1 | Иванов | … | … | 2 |  | 3 |  | 4 |  |
|  | 300 |  |  |  |
| 5 | Петров | … | … | 6 | Сидоров | … | … | 7 |  | 8 |  | 9 | Стрельцов | … | … |
| 200 | 500 |  |  | 100 |
| 10 | Кузнецов | … | … | 11 |  | 12 |  |  |
| 400 |  |  |

рис. 13.6 Пример использования хеширования.

Недостатком хеширования является возможность возникновения коллизий, т.е. ситуаций, когда две или более различных записи имеют одинаковые адреса. Допустим, что файл студентов из предыдущего примера (с записями 100, 200 и т.д.) содержит также запись с номером 1400. При использовании хеш-функции h = StNo mod 13 возникнет коллизия (по адресу 9) с записью 100.

Для разрешения таких коллизий можно использовать значения хеш-функции в качестве адреса не какой-либо записи с данными, а точки привязки. Точка привязки – это начальный адрес цепочки указателей (цепочки коллизий), связывающей вместе все записи или все страницы с записями, которые вызывают коллизии по адресу. Внутри цепочки коллизий записи также могут быть упорядочены согласно хеш-полю для упрощения последующего поиска.

Один из недостатков описанного выше способа хеширования – возрастание числа коллизий с увеличением размера хранимого файла. Это, в свою очередь, может привести к значительному увеличению среднего времени доступа, поскольку все больше и больше времени придется тратить на поиск записей в наборах конфликтующих записей. Однако этот недостаток можно устранить, если реорганизовать файл, т.е. выгрузить данный файл и загрузить его вновь, используя новую хеш-функцию.

Существует ряд модификаций алгоритма хеширования, например, расширяемое хеширование и др., применяемые для оптимизации операций обновления и поиска в БД.

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 674–696.

# Оптимизация запросов

14.1 Оптимизация в реляционных СУБД.

14.2 Пример оптимизации реляционного выражения

14.3 Обзор процесса оптимизации

14.4 Преобразование выражений

## Оптимизация в реляционных СУБД.

Для реляционных систем оптимизация является как проблемой, так и возможностью повышения производительности. Проблема оптимизации состоит в том, что некоторые системы для достижения определенного уровня производительности требуют оптимизации. Оптимизация позволяет улучшить работу системы, так как одной из сильных сторон реляционного подхода является то, что первое применение оптимизации к реляционному выражению переводит это выражение на более эффективный семантический уровень. Общее назначение оптимизатора состоит в выборе эффективной стратегии для вычисления данного реляционного выражения.

Преимущество автоматической оптимизации заключается в том, что пользователь может не задумываться над наилучшим способом выражения своих запросов (т.е. над тем, как сформулировать запрос, чтобы система выполнила его с максимально возможной производительностью). Но это далеко не все. Существует реальная возможность, что оптимизатор сформулирует запрос лучше, чем программист-пользователь. Для подобного утверждения есть ряд причин. Ниже приведены только некоторые из них:

1. Хороший оптимизатор – обратите внимание на слово "хороший" – обладает достаточным количеством информации, которой пользователь может не иметь. Точнее, оптимизатор должен обладать некоторыми статистическими данными, такими как кардинальное число каждого базового отношения, количество различающихся значений для каждого атрибута в отношении, количество вхождений каждого значения в атрибутах и т.п. Благодаря наличию этих данных оптимизатор способен более точно оценивать эффективность любой стратегии реализации конкретного запроса. Поэтому оптимизатор сможет выбрать наилучшую стратегию реализации запроса.
2. Если с течением времени статистика базы данных значительно изменится (например, база данных будет физически реорганизована), то для реализации запроса может потребоваться совсем иная стратегия, чем до реорганизации. Другими словами, может понадобиться повторная оптимизация, или реоптимизация. В реляционных системах процесс реоптимизации достаточно тривиален – это просто повторная обработка исходного запроса системным оптимизатором. С другой стороны, в нереляционных системах реоптимизация требует переписывания программы, и, возможно, невыполнима вообще.
3. Оптимизатор – это программа, поэтому он более "настойчив" по сравнению с человеком. Оптимизатор вполне способен рассматривать буквально сотни различных стратегий реализации данного запроса, в то время как программист вряд ли изучает более трех-четырех стратегий (по крайней мере, достаточно глубоко).
4. В оптимизатор встроены знания и опыт "лучших из лучших" программистов, что делает эти знания и опыт доступными для всех. Естественно, в противном случае широкому кругу пользователей будет предоставлен явно недостаточный набор дешевых и неэффективных возможностей.

## Пример оптимизации реляционного выражения

Начнем изложение с простого примера, дающего представление о результатах, которые можно получить с помощью оптимизации. Рассмотрим запрос "Получить список фамилий студентов, учащихся в группе А-98-51". Алгебраическая запись этого запроса такова:

((Students JOIN Groups) WHERE GrName = 'А-98-51') [StName]

Предположим, что база данных содержит информацию о 100 группах и 10000 студентов, только 30 из которых обучаются в группе А-98-51. В таком случае, если система будет вычислять выражение прямо (т.е. вообще без оптимизации), то последовательность выполняемых действий будет выглядеть так:

1. Соединение отношений Students и Groups (по атрибуту GrNo). На этом этапе считывается информация о 10000 студентов и 10000 раз считывается информация о 100 группах (один раз для каждого студента). После этого создается промежуточный результат, состоящий из 10000 соединенных кортежей.
2. Выборка кортежей с данными только о группе А-98-51 из результата, полученного на этапе 1. На этом этапе создается новое отношение, которое состоит из 30 кортежей.
3. Проекция результата, полученного на этапе 2, по атрибуту StName. На этом этапе создается требуемый результат, состоящий из 30 кортежей.

Показанная ниже процедура эквивалентна описанной в том смысле, что обязательно создаст тот же конечный результат, но более эффективным способом:

1. Выборка кортежей с данными только о группе А-98-51 из отношения Groups. На этом этапе выполняется чтение 100 кортежей и создается результат, состоящий только из 1 кортежа.
2. Соединение результата, полученного на этапе 1, с отношением Students (по атрибуту GrNo). На этом этапе выполняется считывание данных о 10000 студентов и 10000 раз считывается информация о группе А-98-51, полученная на 1 этапе. Результат содержит 30 кортежей.
3. Проецирование результата, полученного на этапе 2, по атрибуту StName (аналогично этапу 3 предыдущей последовательности действий). Требуемый результат содержит 30 кортежей.

Первая из показанных процедур выполняет в общем 1010000 операций ввода-вывода кортежа, в то время как вторая процедура выполняет только 20000 операции ввода-вывода. Следовательно, если принять "количество операции ввода-вывода кортежа" в качестве меры производительности, то вторая процедура в 50 раз эффективнее первой. (На практике мерой производительности служит количество операций ввода-вывода страницы, а не одного кортежа, но для данного примера эту поправку можно игнорировать.)

## Обзор процесса оптимизации

### **Стадия 1. Преобразование запроса во внутреннюю форму**

На этой стадии выполняется преобразование запроса в некоторое внутреннее представление, более удобное для машинных манипуляций. Это полностью исключает из рассмотрения конструкции внешнего уровня (такие как "игра слов" конкретного синтаксиса рассматриваемого языка запросов) и готовит почву для последующих стадий оптимизации.

Обычно внутреннее представление запросов является определенной модификацией абстрактного синтаксического дерева, или дерева запроса.

Например, на рисунке показано дерево рассматриваемого выше в этой главе запроса ("Получить список фамилий студентов, учащихся в группе А-98-51").

рис. 14.1. Дерево запроса "Получить список фамилий студентов, учащихся в группеА-98-51"

Окончательный результат

Проекция по атрибуту StName

Выборка кортежей, в которых GrName =‘А-98-51’

Соединение по аттрибуту GrNo

Groups

Students

### **Стадия 2. Преобразование в каноническую форму**

На этой стадии оптимизатор выполняет несколько операций оптимизации, которые "гарантированно являются хорошими" независимо от реальных данных, хранящихся в базе данных, и путей доступа к ним. Суть в том, что все запросы (за исключением простейших) реляционные языки обычно позволяют выразить несколькими разными (по крайней мере, внешне) способами.

Замечание о канонической форме. Понятие канонической формы употребляется, во многих разделах математики и связанных с ней дисциплин. Каноническая форма может быть определена следующим образом. Пусть Q – множество объектов (запросов), и пусть существует понятие об эквивалентности этих объектов (а именно: запросы q1 и q2 эквивалентны тогда и только тогда, когда дают идентичные результаты) Говорят, что подмножество C множества Q является подмножеством канонических форм для запросов из Q в смысле определенной выше эквивалентности тогда и только тогда, когда каждому объекту q из Q соответствует только один объект c из C. Тогда говорят, что объект с является канонической формой объекта q. Все "интересующие" свойства, которыми обладает объект q, также присущи и объекту с. Поэтому, чтобы доказать различные "интересующие" результаты, достаточно изучить менее мощное множество объектов C, а не более мощное множество Q.

Чтобы преобразовать результаты стадии 1 в некоторую эквивалентную, но более эффективную форму, оптимизатор использует определенные и хорошо известные правила преобразования, или законы.

### Стадия 3. Выбор потенциальных низкоуровневых процедур

После преобразования внутренней формы запроса в **более** подходящую (каноническую) форму оптимизатор должен решить, как выполнять запрос, представленный в канонической форме. На этой стадии принимается во внимание наличие индексов и других путей доступа, распределение хранимых значений данных, физическая кластеризация хранимых данных и т.п. Заметьте, что на стадиях 1 и 2 этим вопросам совсем не уделялось внимания

Для каждой низкоуровневой операции оптимизатор обладает набором низкоуровневых процедур реализации.

Замечание. С каждой процедурой также связана стоимостная формула, которая указывает "стоимость" выполнения процедуры (т.е. уровень требуемых затрат на ее выполнение). Обычно стоимость вычисляется в контексте операций ввода-вывода с диска, но некоторые системы учитывают также время использования процессора и другие факторы. Эти стоимостные формулы используются на стадии 4.

Следовательно, далее с помощью информации из каталога о состоянии базы данных (существующие индексы, кардинальные числа отношений и т.п.) и данных о зависимостях, описанных выше, оптимизатор выберет одну или несколько процедур-кандидатов для каждой низкоуровневой операции в запросе. Этот процесс обычно называют выбором пути доступа.

### Стадия 4. Генерация планов вычисления запроса и выбор плана с наименьшей стоимостью

На последней стадии процесса оптимизации конструируются потенциальные планы запросов, после чего следует выбор лучшего (т.е. наименее дорогого) плана выполнения запроса. Каждый план выполнения строится как комбинация набора процедур реализации, при этом каждой низкоуровневой операции в запросе соответствует одна процедура.

Для выбора плана с наименьшей стоимостью необходим метод привязки стоимости к данному плану. В основном стоимость плана – это просто сумма стоимостей отдельных процедур, которые использованы для его выполнения. Таким образом, работа оптимизатора сводится к вычислению стоимостных формул для каждой такой процедуры. Проблема состоит в том, что стоимость выполнения процедуры зависит от размера отношения (или отношений), которое выбранная процедура обрабатывает.

## Преобразование выражений

### Выборки и проекции

1. Последовательность выборок данного отношения может быть преобразована в одну (объединенную операцией AND) выборку этого отношения. Например, выражение

(A WHERE выборка\_1) WHERE выборка\_2

эквивалентно выражению

A WHERE выборка\_1 AND выборка\_2

1. В последовательности проекций данного отношения можно игнорировать все проекции, кроме последней. Таким образом, выражение

(А [проекция\_1]) [проекция\_2]

эквивалентно выражению

А [Проекция\_2]

Конечно, чтобы первое выражение имело смысл, каждый атрибут, используемый в проекции\_2, должен присутствовать и в проекции\_1.

1. Выборку проекции можно трансформировать в проекцию выборки. Например, выражение

(А [проекция]) WHERE выборка

эквивалентно выражению

(A WHERE выборка) [проекция]

Заметьте, что в основном всегда полезно выполнять операцию выборки перед операцией проекции, так как выборка приведет к уменьшению размера входных данных для операции проекции и, следовательно, к уменьшению количества данных, которые нужно сортировать для исключения дублирующихся записей в процессе вычисления проекции.

### Распределительный закон

Говорят, что унарный оператор распределяется по бинарной операции О, если для всех А и В выполняется условие

F (А О В) ≡ f (А) О f (В).

В реляционной алгебре операция выборки распределяется по операциям объединения, пересечения и вычитания. Операция выборки также распределяется по oneрации соединения, но только тогда, когда условие выборки состоит (в самом сложном случае) из объединенных операцией AND двух отдельных условий выборки – по одному для каждого операнда операции соединения. Для рассматриваемого выше в этой главе примера сформулированное условие соблюдено (условие выборки очень простое и относится лишь к одному операнду), и можно использовать распределительный закон для замены рассматриваемого в примере выражения его более эффективным эквивалентом. Чистый эффект этого закона состоит в том, что можно выполнять "раннюю выборку". Выполнение ранней выборки почти всегда себя оправдывает, так как приводит к значительному уменьшению количества кортежей, которые нужно рассматривать в следующей операции. Кроме того, ранняя выборка может привести к уменьшению количества кортежей и на выходе следующей операции.

Далее приведено несколько более специфических примеров распределительного закона, на этот раз с операцией проекции. Во-первых, операция проекции распределяется по операциям объединения и пересечения (но не по операции вычитания). Во-вторых, эта операция также распределяется по операции соединения, но только в том случае, если в проекцию включены все атрибуты соединения. Точнее, выражение

(A JOIN В) [проекция]

эквивалентно выражению

(А [А\_проекция]) JOIN (В [В\_проекция])

тогда и только тогда, когда множество использованных в проекции атрибутов равняется объединению множеств атрибутов в А\_проекции и В\_проекции и включает атрибуты, по которым выполнено соединение. Этот закон можно использовать для выполнения ранних "проекций", которые обычно себя оправдывают по тем же причинам, что и операции выборки.

### Коммутативность и ассоциативность

Законы коммутативности и ассоциативности – это еще два общих правила преобразования. Говорят, что бинарная операция О является коммутативной, если для всех А и В истинно равенство

А О В ≡ В О А

Например, в обычной арифметике операции умножения и сложения являются коммутативными, а операции деления и вычитания – нет. В реляционной алгебре коммутативными являются операции объединения, пересечения и соединения, а операции вычитания и деления таковыми не являются.

Перейдем к ассоциативности. Принято считать, что бинарная операция О является ассоциативной, если для всех А, В и С истинно равенство

А О (В О С) ≡ (А О В) О С.

Например, в обычной арифметике произведение и сложение – ассоциативные операции, деление и вычитание – нет. В реляционной алгебре ассоциативными являются операции объединения, пересечения и соединения, а операции вычитания и деления таковыми не являются. Так, например, если в запросе используется соединение трех отношений, А, В и С, то из законов коммутативности и ассоциативности

### Идемпотентность

Еще одним важным правилом является закон идемпотентности. Идемпотентной называют такую бинарную операцию О, для которой для всех А выполняется равенство

A О А = А.

Можно ожидать, что свойство идемпотентности также может быть полезным в процессе трансформации выражений. В реляционной алгебре операции объединения, пересечения и соединения являются идемпотентными, а операции деления и вычитания – нет.

### Вычисляемые скалярные выражения

Предметом применения законов трансформации являются не только реляционные выражения. Например, уже было показано, что некоторые законы трансформации применимы и к арифметическим выражениям. Ниже приведен пример. Выражение

А \* В + А \* С

можно трансформировать в выражение

А \* (В + С)

вследствие того, что операция умножения "\*" распределяется по операции сложения "+". Оптимизатор реляционных выражений должен обладать информацией о подобных преобразованиях, так как он учитывает вычисляемые скалярные выражения в контексте операций EXTEND и SUMMARIZE.

Говорят, что бинарная операция О распределяется по бинарной операции О, если для всех А, В и С истинно равенство

A Ú (B О C) = (A Ú B) O ( A Ú C )

(для приведенного выше арифметического примера замените Ú на "\*", а О на "+").

### Условия

Перейдем к обсуждению условий или выражений, результатами которых могут быть истина или ложь. Предположим, что А и В – атрибуты двух различных отношений, тогда условие

А>В AND В>3

(которое может быть частью запроса) абсолютно эквивалентно выражению

А > В AND В > 3 AND A > 3

и потому может быть преобразовано в это выражение.

Данная эквивалентность базируется на том, что операция ">" является транзитивной. Заметьте, что выполнение подобного преобразования весьма полезно, так как позволяет системе создать дополнительную выборку (с помощью условия "А > З") перед выполнением соединения "больше чем", требуемого условием "А > В".

Замечание. Этот прием реализован в различных коммерческих продуктах, включая систему DB2, в которой его называют транзитивным замыканием предикатов. А вот другой пример. Условие

А > В OR (С = D AND Е < F)

можно преобразовать в условие

(A > B OR С = D) AND (А > В OR Е < F)

вследствие того, что операция OR распределяется по операции AND. Этот пример демонстрирует другой общий закон: "Любое условие может быть преобразовано в эквивалентное условие, называемое конъюнктивной нормальной формой (КНФ)". КНФ-выражение имеет вид:

C1 AND C2 AND … AND Cn,

где С1, C2, ..., Cn – условия (называемые частичная конъюнкция), в которых не используется операция AND. Преимущество КНФ состоит в том, что КНФ-выражение истинно, только если истинны все его частичные конъюнкции. Аналогично, КНФ‑выражение ложно, если ложь является результатом хотя бы одной частичной конъюнкции. Так как операция AND коммутативна (A AND В равно В AND А), то оптимизатор может вычислять отдельные частичные конъюнкции в любом порядке, в частности по возрастанию сложности (сначала простые). И как только найдена частичная конъюнкция, результатом которой является ложь, весь процесс вычисления КНФ-выражения можно останавливать.

Более того, в среде параллельных вычислений возможно параллельное вычисление всех частичных конъюнкций. Опять же, как только найдена первая частичная конъюнкция, результатом которой является ложь, весь процесс вычисления КНФ-выражения можно останавливать.

### Семантические преобразования

Рассмотрим следующее выражение:

(Students JOIN Groups) [StName]

Данное соединение относится к соединениям типа внешниий-к-согласованному-потенциалъному-ключу. В этом соединении внешнему ключу в отношении Students ставится в соответствие потенциальный ключ отношения Groups. Следовательно, кортеж в отношении Students связан с определенным кортежем в отношении Groups. Таким образом, из каждого кортежа в отношении Students в общий результат поступает только значение атрибута StName. Другими словами, соединение можно не выполнять! Рассматриваемое выражение можно заменить выражением

Students [StName]

Преобразование, корректное в силу определенного условия целостности, называют семантическим преобразованием, а оптимизацию, полученную в результате подобных преобразований, – семантической оптимизацией. Семантическую оптимизацию можно определить как процесс преобразования запроса в другой, качественно отличный запрос, который, тем не менее, дает результат, идентичный результату исходного запроса, благодаря тому что данные удовлетворяют определенному условию целостности.

Важно понимать, что в принципе любое условие целостности может быть использовано для семантической оптимизации (если это условие не отсрочено и в данный момент действует на базу данных).

### Статистики базы данных

На стадиях 3 и 4 общего процесса оптимизации (они называются стадиями "выбора пути доступа") используются так называемые статистики базы данных, которые хранятся в каталоге.

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 474–516.

# Восстановление после сбоев

15.1 Понятие восстановления системы

15.2 Транзакции

15.3 Алгоритм восстановления после сбоя системы

15.4 Параллелизм. Проблемы параллелизма

15.5 Понятие блокировки

15.6 Решение проблем параллелизма

15.7 Тупиковые ситуации

15.8 Способность к упорядочению

15.9 Уровни изоляции транзакции

15.10 Поддержка в языке SQL

## Понятие восстановления системы

Восстановление в системе управления базами данных, означает в первую очередь восстановление самой базы данных, т.е. возвращение базы данных в правильное состояние, если какой-либо сбой сделал текущее состояние неправильным или подозрительным. Основной принцип, на котором строится такое восстановление, – это избыточность. Избыточность организуется на физическом уровне. Такая избыточность будет скрыта от пользователя, а следовательно, не видна на логическом уровне. Другими словами, если любая часть информации, содержащаяся в базе данных, может быть реконструирована из другой хранимой в системе избыточной информации, значит, база данных восстанавливаема.

## Транзакции

### Понятие транзакции

Транзакция – это логическая единица работы. Например. Предположим сначала, что отношение Students (отношение студентов) включает дополнительный атрибут AvgMark, представляющий собой средний балл студента, по результатам сдачи текущей сессии. Значение AvgMark для любой определенной детали предполагается равным среднему арифметическому всех значений Mark из таблицы Marks для всех оценок полученных в текущем семестре.

В приведенном примере предполагается, что речь идет об одиночной, атомарной операции. На самом деле добавление новой оценки в таблицу Marks – это выполнение двух обновлений в базе данных (под обновлениями здесь, конечно, понимаются операции insert, delete, а также сами по себе операции update). Более того, в базе данных между этими двумя обновлениями временно нарушается требование, что значение AvgMark для студента 1 равно среднему арифметическому всех значений поля Mark для студента 1 в текущем семестре. Таким образом, логическая единица работы (т.е. транзакция) – не просто одиночная операция системы баз данных, а скорее согласование нескольких таких операций. В общем, это преобразование одного согласованного состояния базы данных в другое, причем в промежуточных точках база данных находится в несогласованном состоянии.

Из этого следует, что недопустимо, чтобы одно из обновлений было выполнено, а другое нет, так как база данных останется в несогласованном состоянии. В идеальном случае должны быть выполнены оба обновления. Однако нельзя обеспечить стопроцентную гарантию, что так и будет. Не исключена вероятность того, что, система, например, будет разрушена между двумя обновлениями, или же на втором обновлении произойдет арифметическое переполнение и т.п. Система, поддерживающая транзакции, гарантирует, что если во время выполнения неких обновлений произошла ошибка (по любой причине), то все эти обновления будут аннулированы. Таким образом, транзакция или выполняется полностью, или полностью отменяется (как будто она вообще не выполнялась).

Системный компонент, обеспечивающий атомарность (или ее подобие), называется администратором транзакций (или диспетчером транзакций), а ключами к его выполнению служат операторы COMMIT TRANSACTION и ROLLBACK TRANSACTION.

Оператор COMMIT TRANSACTION (для краткости commit) сигнализирует об успешном окончании транзакции. Он сообщает администратору транзакций, что логическая единица работы завершена успешно, база данных вновь находится (или будет находиться) в согласованном состоянии, а все обновления, выполненные логической единицей работы, теперь могут быть зафиксированы, т.е. стать постоянными.

Оператор ROLLBACK TRANSACTION (для краткости ROLLBACK) сигнализирует о неудачном окончании транзакции. Он сообщает администратору транзакций, что произошла какая-то ошибка, база данных находится в несогласованном состоянии и все обновления могут быть отменены, т.е. аннулированы.

Для отмены обновлений система поддерживает файл регистрации, или журнал, на диске, где записываются детали всех операций обновления, в частности новое и старое значения модифицированного объекта. Таким образом, при необходимости отмены некоторого обновления система может использовать соответствующий файл регистрации для возвращения объекта в первоначальное состояние.

Еще один важный момент. Система должна гарантировать, что индивидуальные операторы сами по себе атомарные (т.е. выполняются полностью или не выполняются совсем). Это особенно важно для реляционных систем, в которых операторы многоуровневые и обычно оперируют множеством кортежей одновременно; такой оператор просто не может быть нарушен посреди операции и привести систему в несогласованное состояние. Другими словами, если произошла ошибка во время работы такого оператора, база данных должна остаться полностью неизмененной. Более того, это должно быть справедливо даже в том случае, когда действия оператора являются причиной дополнительной, например каскадной, операции.

### Восстановление транзакции.

Транзакция начинается с успешного выполнения оператора BEGIN TRANSACTION) и заканчивается успешным выполнением либо оператора COMMIT, либо ROLLBACK. Оператор COMMIT устанавливает так называемую точку фиксации (которая в коммерческих продуктах также называется точкой синхронизации (syncpoint). Точка фиксации соответствует концу логической единицы работы и, следовательно, точке, в которой база данных находится (или будет находиться) в состоянии согласованности. В противовес этому, выполнение оператора ROLLBACK вновь возвращает базу данных в состояние, в котором она была во время операции BEGIN TRANSACTION, т.е. в предыдущую точку фиксации.

Случаи установки точки фиксации:

1. Все обновления, совершенные программой с тех пор, как установлена предыдущая точка фиксации, выполнены, т.е. стали постоянными. Во время выполнения все такие обновления могут расцениваться только как пробные (в том смысле, что они могут быть не выполнены, например прокручены назад). Гарантируется, что однажды зафиксированное обновление так и останется зафиксированным (это и есть определение понятия "зафиксировано").
2. Все позиционирование базы данных утеряно, и все блокировки кортежей реализованы. Позиционирование базы данных здесь означает, что в любое конкретное время программа обычно адресована определенным кортежам. Эта адресуемость в точке фиксации теряется.

Следовательно, система может выполнить откат транзакции как явно – например по команде ПО с которым работает пользователь, так и неявно – для любой программы, которая по какой-либо причине не достигла запланированного завершения операций, входящих в транзакцию.

Из этого видно, что транзакции — это не только логические единицы работы, но также и единицы восстановления при неудачном выполнении операций. При успешном завершении транзакции система гарантирует, что обновления постоянно установлены в базе данных, даже если система потерпит крах в следующий момент. Возможно, что в системе произойдет сбой после успешного выполнения COMMIT, но перед тем, как, обновления будут физически записаны в базу данных (они все еще могут оставаться в буфере оперативной памяти и таким образом могут быть утеряны в момент сбоя системы). Даже если подобное случилось, процедура перезагрузки системы все равно должна устанавливать эти обновления в базу данных, исследуя соответствующие записи в файле регистрации. Из этого следует, что файл регистрации должен быть физически записан перед завершением операции COMMIT. Это важное правило ведения файла регистрации известно как протокол предварительной записи в журнал (т.е. запись об операции осуществляется перед ее выполнением). Таким образом, процедура перезагрузки сможет восстановить любые успешно завершенные транзакции, хотя их обновления не были записаны физически до аварийного отказа системы. Следовательно, как указывалось ранее, транзакция действительно является единицей восстановления.

### Свойства АСИД.

Из предыдущих разделов следует, что транзакции обладают четырьмя важными свойствами: атомарность, согласованность, изоляция и долговечность (назовем это свойствами АСИД).

1. Атомарность. Транзакции атомарны (выполняется все или ничего).
2. Согласованность. Транзакции защищают базу данных согласованно. Это означает, что транзакции переводят одно согласованное состояние базы данных в другое без обязательной поддержки согласованности во всех промежуточных точках.
3. Изоляция. Транзакции отделены одна от другой. Это означает, что, если даже будет запущено множество конкурирующих друг с другом транзакций, любое обновление определенной транзакции будет скрыто от остальных до тех пор, пока эта транзакция выполняется. Другими словами, для любых двух отдаленных транзакций Т1 и Т2 справедливо следующее утверждение: Т1 сможет увидеть обновление Т2 только после выполнения Т2, а Т2 сможет увидеть обновление Т1 только после выполнения Т1.
4. Долговечность. Когда транзакция выполнена, ее обновления сохраняются, даже если в следующий момент произойдет сбой системы.

## Алгоритм восстановления после сбоя системы

Система должна быть готова к восстановлению не только после небольших локальных нарушений, таких как невыполнение операции в пределах определенной транзакции, но также и после глобальных нарушений типа сбоев в питании вычислительного устройства и др. Местное нарушение по определению поражает только транзакцию, в которой оно собственно и произошло. Глобальное нарушение поражает сразу все транзакции и, следовательно, приводит к значительным для системы последствиям.

Существует два вида глобальных нарушений:

1. Отказы системы (например, сбои в питании), поражающие все выполняющиеся в данный момент транзакции, но физически не нарушающие базу данных в целом. Такие нарушения в системе также называют аварийным отказом программного обеспечения.
2. Отказы носителей (например, поломка головок дискового накопителя), которые могут представлять угрозу для базы данных или для какой-либо ее части и поражать. по крайней мере, те транзакции, которые используют эту часть базы данных. Отказы носителей также называют аварийным отказом аппаратуры.

### Восстановление после отказов системы

Критической точкой в отказе системы является потеря содержимого оперативной памяти (в частности, рабочих буферов базы данных). Поскольку точное состояние какой-либо выполняющейся в момент нарушения транзакции не известно, транзакция может не завершиться успешно и, таким образом, будет отменена при перезагрузке системы.

Более того, возможно, потребуется повторно выполнить определенную успешно завершившуюся до аварийного отказа транзакцию при перезагрузке системы, если не были физически выполнены обновления этой транзакции.

Для определения во время перезагрузки, какую транзакцию отменить, а какую выполнить повторно система в некотором предписанном интервале (когда в журнале накапливается определенное число записей) автоматически принимает контрольную точку. Принятие контрольной точки включает физическую запись содержимого рабочих буферов базы данных непосредственно в базу данных и специальную физическую запись контрольной точки, которая предоставляет список всех осуществляемых в данный момент транзакций. На рис. 15.1 рассматривается пять возможных вариантов выполнения транзакций до аварийного сбоя системы.

рис. 15.1 Варианты выполнения пяти транзакций.

tc

tf

T1

T2

T3

T4

Контрольная точка

(время tc)

Отказ системы

(время tf)

Транзакции

Время

T5

Пояснения к рис. 15.1:

1. Отказ системы произошел в момент времени tf.
2. Близлежащая к моменту времени tf контрольная точка была принята в момент времени tc.
3. Транзакция Т1 успешно завершена до момента времени tc.
4. Транзакция Т2 начата до момента времени tc и успешно завершена после момента времени tc, но до момента времени tf.
5. Транзакция ТЗ также начата до момента времени tc, но не завершена к моменту времени tf
6. Транзакция Т4 начата после момента времени tc и успешно завершена до момента времени tf.
7. Транзакция Т5 также начата после момента времени tc, но не завершена к моменту времени tf.

Очевидно, что при перезагрузке системы транзакции типа ТЗ и Т5 должны быть отменены, а транзакции типа Т2 и Т4 – выполнены повторно. Тем не менее заметьте, что транзакции типа Т1 вообще не включаются в процесс перезагрузки, так как обновления попали в базу данных еще до момента времени tc (т.е. зафиксированы еще до принятия контрольной точки). Отметьте также, что транзакции, завершившиеся неудачно (в том числе отмененные) перед моментом времени tf, вообще не будут вовлечены в процесс перезагрузки.

## Параллелизм. Проблемы параллелизма

Термин параллелизм означает возможность одновременной обработки в СУБД многих транзакций с доступом к одним и тем же данным, причем в одно и то же время. В такой системе для корректной обработки параллельных транзакций без возникновения конфликтных ситуаций необходимо использовать некоторый метод управления параллелизмом.

Каждый метод управления параллелизмом предназначен для решения некоторой конкретной задачи. Тем не менее, при обработке правильно составленных транзакций возникают ситуации, которые могут привести к получению неправильного результата из-за взаимных помех среди некоторых транзакций. (Обратите внимание, что вносящая помеху транзакция сама по себе может быть правильной. Неправильный конечный результат возникает по причине бесконтрольного чередования операций из двух правильных транзакций). Основные проблемы, возникающие при параллельной обработке транзакций следующие:

1. проблема потери результатов обновления;
2. проблема незафиксированной зависимости;
3. проблема несовместимого анализа.

### Проблема потери результатов обновления

Рассмотрим ситуацию, показанную на рис. 15.2, в такой интерпретации: транзакция A извлекает некоторый кортеж p в момент времени t1; транзакция B извлекает некоторый кортеж p в момент времени t2; транзакция A обновляет некоторый кортеж p (на основе значений, полученных в момент времени t1) в момент времени t3; транзакция B обновляет тот же кортеж р (на основе значений, полученных в момент времени t2, которые имеют те же значения, что и в момент времени t1) в момент времени t4. Однако результат операции обновления, выполненной транзакцией A, будет утерян, поскольку в момент времени t4 она не будет учтена и потому будет "отменена" операцией обновления, выполненной транзакцией B.

|  |  |  |
| --- | --- | --- |
| Транзакция A | Время | Транзакция B |
| Извлечение кортежа р | t1 | – |
| – | t2 | Извлечение кортежа р |
| Обновление кортежа р | t3 | – |
| – | t4 | Обновление кортежа р |

рис. 15.2. Потеря в момент времени t4 результатов обновления, выполненного транзакцией A.

### Проблема незафиксированной зависимости

Проблема незафиксированной зависимости появляется, если с помощью некоторой транзакции осуществляется извлечение (или, что еще хуже, обновление) некоторого кортежа, который в данный момент обновляется другой транзакцией, но это обновление еще не закончено. Таким образом, если обновление не завершено, существует некоторая вероятность того, что оно не будет завершено никогда. (Более того, в подобном случае может быть выполнен возврат к предыдущему состоянию кортежа с отменой выполнения транзакции.) B таком случае, в первой транзакции будут принимать участие данные, которых больше не существует. Эта ситуация показана на рис. 15.3, рис. 15.4.

В первом примере (рис. 15.3) транзакция A в момент времени t2 встречается с невыполненным обновлением (оно также называется невыполненным изменением). Затем это обновление отменяется в момент времени t3. Таким образом, транзакция A выполняется на основе фальшивого предположения, что кортеж р имеет некоторое значение в момент времени t2, тогда как на самом деле он имеет некоторое значение, существовавшее еще в момент времени t1. В итоге после выполнения транзакции A будет получен неверный результат. Кроме того, обратите внимание, что отмена выполнения транзакции B может произойти не по вине транзакции B, а, например, в результате краха системы. (К этому времени выполнение транзакции A может быть уже завершено, а потому крушение системы не приведет к отмене выполнения транзакции A.)

|  |  |  |
| --- | --- | --- |
| Транзакция A | Время | Транзакция B |
| – | t1 | Обновление кортежа р |
| Извлечение кортежа р | t2 | – |
| – | t3 | Отмена выполнения транзакции |

рис. 15.3. Транзакция A становится зависимой от невыполненного изменения в момент времени t2.

|  |  |  |
| --- | --- | --- |
| Транзакция A | Время | Транзакция B |
| – | t1 | Обновление кортежа р |
| Обновление кортежа р | t2 | – |
| – | t3 | Отмена выполнения транзакции |

рис. 15.4. Транзакция A обновляет невыполненное изменение в момент времени t2, и результаты этого обновления утрачиваются в момент времени t3.

Второй пример, приведенный на рис. 15.4, иллюстрирует другой случай. Не только транзакция A становится зависимой от изменения, не выполненного в момент времени t2, но также в момент времени t3 фактически утрачивается результат обновления, поскольку отмена выполнения транзакции B в момент времени t3 приводит к восстановлению кортежа р к исходному значению в момент времени t1. Это еще один вариант проблемы потери результатов обновления.

### Проблема несовместимого анализа

На рис. 15.5 показаны транзакции A и B, которые выполняются для кортежей со счетами (табл. 9.1). При этом транзакция A суммирует балансы, транзакция B производит перевод суммы 10 со счета 3 на счет 1. Полученный в итоге транзакции A результат 110, очевидно, неверен, и если он будет записан в базе данных, то в ней может возникнуть проблема несовместимости. В таком случае говорят, что транзакция A встретилась с несовместимым состоянием и на его основе был выполнен несовместимый анализ. Обратите внимание на следующее различие между этим примером и предыдущим: здесь не идет речь о зависимости транзакции A от транзакции B, так как транзакция B выполнила все обновления до того, как транзакция A извлекла СЧЕТ 3.

табл. 15.1 Остатки на счетах до выполнения транзакций.

|  |  |  |  |
| --- | --- | --- | --- |
| Счет | СЧЕТ 1 | СЧЕТ 2 | СЧЕТ 3 |
| Остаток | 40 | 50 | 30 |

|  |  |  |
| --- | --- | --- |
| Транзакция A | Время | Транзакция B |
| Извлечение кортежа СЧЕТ 1:СУММА = 40 | t1 | – |
| Извлечение кортежа СЧЕТ 1:СУММА = 90 | t2 | – |
| – | t3 | Извлечение кортежа СЧЕТ 3: |
| – | t4 | Обновление кортежа СЧЕТ 3:30 → 20 |
| – | t5 | Извлечение кортежа СЧЕТ 1: |
| – | t6 | Обновление кортежа СЧЕТ 1:40 → 50 |
| – | t7 | Завершение выполнения транзакции |
| Извлечение кортежа СЧЕТ 3:СУММА = 110 (а не 120) | t8 | – |

рис. 15.5. Транзакция A выполнила несовместимый анализ.

## Понятие блокировки

Описанные выше проблемы могут быть разрешены с помощью методики управления параллельным выполнением процессов под названием блокировка. Ее основная идея очень проста: в случае, когда для выполнения некоторой транзакции необходимо, чтобы некоторый объект (обычно это кортеж базы данных) не изменялся непредсказуемо и без ведома этой транзакции (как это обычно бывает), такой объект блокируется. Таким образом, эффект блокировки состоит в том, чтобы "заблокировать доступ к этому объекту со стороны других транзакций", а значит, предотвратить непредсказуемое изменение этого объекта. Следовательно, первая транзакция в состоянии выполнить всю необходимую обработку с учетом того, что обрабатываемый объект остается в стабильном состоянии настолько долго, насколько это нужно.

Предположим, что в системе поддерживается два типа блокировок: блокировка без взаимного доступа (монопольная блокировка), называемая Х-блокировкой (X locks – exclusive locks), и блокировка с взаимным доступом, называемая S-блокировкой (S locks - Shared locks). Замечание. Х- и S-блокировки иногда называют блокировками записи и чтения соответственно. Предположим, что Х- и S-блокировки единственно возможные, хотя в коммерческих системах существуют блокировки других типов. Кроме того, допустим, что в кортежи являются единственным типом "блокируемого объекта", хотя опять же н в коммерческих системах могут блокироваться и другие объекты. Ниже показано функционирование механизма блокировок.

1. Если транзакция A блокирует кортеж р без возможности взаимного доступа (Х‑блокировка), то запрос другой транзакции B с блокировкой этого кортежа p будет отменен.
2. Если транзакция A блокирует кортеж р с возможностью взаимного доступа (S‑блокировка), то:
	1. запрос со стороны некоторой транзакции B на Х‑блокировку кортежа будет отвергнут;
	2. запрос со стороны некоторой транзакции B на S‑блокировку кортежа р будет принят (т.е. транзакция B также будет блокировать кортеж р с помощью S‑блокировки).

Эти правила можно наглядно представить в виде матрицы совместимости, показанной на рис. 15.6, и интерпретировать ее следующим образом. Рассмотрим некоторый кортеж р и предположим, что транзакция A блокирую кортеж р различными типами блокировки (это обозначено соответствующими символами S и X, а отсутствие блокировки — прочерком). Предположим также, что некоторая транзакция B запрашивает блокировку кортежа р, что обозначено в первом слева столбце матрицы на рис. 15.6 (для полноты картины в таблице также приведен случай "отсутствия блокировки"). В других ячейках матрицы символ N обозначает конфликтную ситуацию (запрос со стороны транзакции B не может быть удовлетворен, и сама эта транзакция переходит в состояние ожидания), a Y – полную совместимость (запрос со стороны транзакции B удовлетворен). Очевидно, что эта матрица является симметричной.

|  |  |  |  |
| --- | --- | --- | --- |
|  | X | S | – |
| X | N | N | Y |
| S | N | Y | Y |
| – | Y | Y | Y |

рис. 15.6. Матрица совместимости для Х- и S-блокировки.

Введем протокол доступа к данным, который на основе введения только что описанных Х- и S-блокировки позволяет избежать возникновения проблем параллелизма.

1. Транзакция, предназначенная для извлечения кортежа, прежде всего должна наложить S‑блокировку на этот кортеж.
2. Транзакция, предназначенная для обновления кортежа, прежде всего должна наложить Х–блокировку на этот кортеж. Иначе говоря, если, например, для последовательности действий типа извлечение/обновление для кортежа уже задана S-блокировка, то ее необходимо заменить Х‑блокировкой. Блокировки в транзакциях обычно задаются неявным образом: например, запрос на "извлечение кортежа" является неявным запросом с S-блокировкой, а запрос на "обновление кортежа" – неявным запросом с Х‑блокировкой соответствующего кортежа. При этом под термином "обновление" (как и ранее) подразумеваются помимо самих операций обновления также операции вставки и удаления.
3. Если запрашиваемая блокировка со стороны транзакции B отвергается из-за конфликта с некоторой другой блокировкой со стороны транзакции A, то транзакция B переходит в состояние ожидания. Причем транзакция B будет находиться в состоянии ожидания до тех пор, пока не будет снята блокировка, заданная транзакцией A. В системе обязательно должны быть предусмотрены способы устранения бесконечно долгого состояния ожидания транзакции B.
4. Х–блокировки сохраняются вплоть до конца выполнения транзакции (до операции "завершение выполнения" или "отмена выполнения"). S‑блокировки также обычно сохраняются вплоть до этого момента.

## Решение проблем параллелизма

Рассмотрим решение проблем параллелизма с помощью механизма блокировок.

### Проблема потери результатов обновления.

На рис. 15.7 приведена измененная версия процесса, показанного на рис. 15.2, с учетом применения протокола блокировки для чередующихся операций. Операция обновления для транзакции A в момент времени t3 не будет выполнена, поскольку она является неявным запросом с заданием Х-блокировки для кортежа р, а этот запрос вступает в конфликт с S-блокировкой, уже заданной транзакцией B. Таким образом, транзакция A переходит в состояние ожидания. По аналогичным причинам транзакция B переходит в состояние ожидания в момент времени t4.Обновления теперь не утрачиваются, однако возникает новая проблема – бесконечное ожидание или тупиковая ситуация. Способы решения этой проблемы рассматриваются ниже.

|  |  |  |
| --- | --- | --- |
| Транзакция A | Время | Транзакция B |
| Извлечение кортежа р(задание S-блокировки для p) | t1 | – |
| – | t2 | Извлечение кортежа р(задание S-блокировки для p) |
| Обновление кортежа р(задание X-блокировки для p) | t3 | – |
| Ожидание | t4 | Обновление кортежа р(задание X-блокировки для p) |
| Ожидание |  | Ожидание |

рис. 15.7. Хотя обновления не утрачиваются, но в момент времени t4 возникает тупиковая ситуация.

### Проблема незафиксированной зависимости.

На рис. 15.8, рис. 15.9 приведены в измененном виде примеры, показанные ранее на рис. 15.3 и рис. 15.4 соответственно. Они демонстрируют чередующееся выполнение операций согласно описанному выше протоколу блокировки. Операция для транзакции A в момент времени t2 (извлечение на рис. 15.8 и обновление на рис. 15.9) не будет выполнена. Дело в том, что она является неявным запросом с заданием блокировки для кортежа р, а этот запрос вступает в конфликт с Х-блокировкой, уже заданной транзакцией B. Таким образом, транзакция A переходит в состояние ожидания до тех пор, пока не будет прекращено выполнение транзакции B (до операции окончания или отмены выполнения транзакции B). Тогда заданная транзакцией B блокировка будет снята и транзакция A может быть выполнена. Причем транзакция A будет иметь дело с некоторым фиксированным значением (либо существовавшим до выполнения транзакции B при отмене ее выполнения, либо полученным после выполнения транзакции B). В любом случае транзакция A больше не зависит от незафиксированного обновления.

|  |  |  |
| --- | --- | --- |
| Транзакция A | Время | Транзакция B |
| – | t1 | Обновление кортежа р(задание X-блокировки для p) |
| Извлечение кортежа р(задание S-блокировки для p) | t2 | – |
| Ожидание | t3 | Отмена выполнения транзакции(снятие X-блокировки для p) |
| Итог: Извлечение кортежа р(задание S-блокировки для p) | t4 |  |

рис. 15.8. Транзакция A предохраняется от выполнения операций с незафиксированным изменением в момент времени t2.

|  |  |  |
| --- | --- | --- |
| Транзакция A | Время | Транзакция B |
| – | t1 | Обновление кортежа р(задание X-блокировки для p) |
| Обновление кортежа р(задание X-блокировки для p) | t2 | – |
| Ожидание | t3 | Отмена выполнения транзакции(снятие X-блокировки для p) |
| Итог: Обновление кортежа р(задание X-блокировки для p) | t4 |  |

рис. 15.9. Транзакция A предохраняется от выполнения операций с незафиксированным изменением в момент времени t2.

### Проблема несовместимого анализа

На рис. 15.10 приведена измененная версия отношения (рис. 15.5) с перечислением чередующихся транзакций согласно протоколу блокировки. Операция обновления для транзакции B в момент времени t6 не будет выполнена. Дело в том, что она является неявным запросом с заданием Х-блокировки для кортежа СЧЕТ 1, а этот запрос вступает в конфликт с S-блокировкой, уже заданной транзакцией A. Таким образом, транзакция B переходит в состояние ожидания. Точно так же операция извлечения для транзакции A в момент времени t7 не будет выполнена. Дело в том, что она является неявным запросом с заданием S-блокировки для кортежа СЧЕТ 3, а этот запрос вступает в конфликт с Х‑блокировкой, уже заданной транзакцией B. Таким образом, транзакция A переходит в состояние ожидания. Следовательно, блокировка хотя и помогает решить одну проблему (а именно проблему несовместимого анализа), но приводит к необходимости решения другой проблемы (а именно проблемы возникновения тупиковой ситуации).

|  |  |  |
| --- | --- | --- |
| СЧЕТ 140 | СЧЕТ 250 | СЧЕТ 330 |
| Транзакция A | Время | Транзакция B |
| Извлечение кортежа СЧЕТ 1:(задание S-блокировки для СЧЕТ 1)СУММА = 40 | t1 | – |
| Извлечение кортежа СЧЕТ 1:(задание S-блокировки для СЧЕТ 2)СУММА = 90 | t2 | – |
| – | t3 | Извлечение кортежа СЧЕТ 3:(задание S-блокировки для СЧЕТ 3) |
| – | t4 | Обновление кортежа СЧЕТ 3:(задание X-блокировки для СЧЕТ 3)30 → 20 |
| – | t5 | Извлечение кортежа СЧЕТ 1:(задание S-блокировки для СЧЕТ 1) |
| – | t6 | Обновление кортежа СЧЕТ 1:(задание X-блокировки для СЧЕТ 1)40 → 50 |
| – | t7 | Ожидание |
| Извлечение кортежа СЧЕТ 3:(задание S-блокировки для СЧЕТ 3) | t8 | Ожидание |
| Ожидание |  | Ожидание |

рис. 15.10. Проблема несовместимого анализа разрешается, но в момент времени t7 возникает тупиковая ситуация.

## Тупиковые ситуации

Как было показано выше, блокировку можно использовать для разрешения трех основных проблем, возникающих при параллельной обработке кортежей. К сожалению, использование блокировок приводит к возникновению другой проблемы – тупиковой ситуации. На рис. 15.11 показан обобщенный пример этой проблемы, в котором p1 и p2 представляют любые блокируемые объекты, необязательно кортежи базы данных, а выражения типа "блокировка ... без взаимного доступа" представляют любые операции с наложением блокировки без взаимного доступа, заданные как явно, так и неявно.

|  |  |  |
| --- | --- | --- |
| Транзакция A | Время | Транзакция B |
| Блокировка р1 без взаимного доступа | t1 | – |
| – | t2 | Блокировка р2 без взаимного доступа |
| Блокировка р2 без взаимного доступа | t3 | – |
| Ожидание | t4 | Блокировка р1 без взаимного доступа |
| Ожидание |  | Ожидание |

рис. 15.11. Пример тупиковой ситуации.

Тупиковая ситуация возникает тогда, когда две или более транзакции одновременно находятся в состоянии ожидания, причем для продолжения работы каждая из транзакций ожидает прекращения выполнения другой транзакции.

Для обнаружения тупиковой ситуации следует обнаружить цикл в диаграмме состояний ожидания, т.е. в перечне "транзакций, которые ожидают окончания выполнения других транзакций" Поиск выхода из тупиковой ситуации состоит в выборе одной из заблокированных транзакций в качестве жертвы и отмене ее выполнения. Таким образом, с нее снимается блокировка, а выполнение другой транзакции может быть возобновлено.

На практике не все системы в состоянии обнаружить тупиковую ситуацию. Например, в некоторых из них используется хронометраж выполнения транзакций, и сообщение о возникновении тупиковой ситуации поступает, если транзакция не выполняется за некоторое предписанное заранее время.

Следует обратить внимание на то, что транзакция-жертва признается "некорректной" и отменяется "не из-за собственной некорректности". В некоторых системах предусмотрен автоматический перезапуск транзакции с самого начала при условии, что обстоятельства, которые привели к тупиковой ситуации, не повторятся вновь. A в других системах в программу, связанную с данной транзакцией, просто посылается сообщение о "вызвавшей тупиковую ситуацию транзакции-жертве" для обработки этой ситуации в самой программе С точки зрения программирования приложений предпочтительнее первый из этих подходов. Но несмотря на это, всегда рекомендуется решать данную проблему с точки зрения пользователя.

## Способность к упорядочению

Чередующееся выполнение заданного множества транзакций будет верным, если оно упорядочено, т.е. при его выполнении будет получен такой же результат, как и при последовательное выполнении тех же транзакций. Обосновать это утверждение помогут следующие замечания:

1. Отдельные транзакции считаются верными, если при их выполнении база данных переходит из одного непротиворечивого состояния в другое непротиворечивое состояние.
2. Выполнение транзакций одна за другой в любом последовательном порядке также является верным. При этом под выражением "любой последовательный порядок" подразумевается, что используются независимые друг от друга транзакции.
3. Чередующееся выполнение транзакций, следовательно, является верным, если оно эквивалентно некоторому последовательному выполнению, т.е. если оно подлежит упорядочению.

Возвращаясь к приведенным выше примерам (рис. 15.2 – рис. 15.5), можно отметить, что проблема в каждом случае заключалась в том, что чередующееся выполнение транзакций не было упорядочено, т.е. не было эквивалентно выполнению либо сначала транзакции A, а затем транзакции B, либо сначала транзакции B, а затем транзакции A.

Для заданного набора транзакций любой порядок их выполнения (чередующийся или какой-либо другой) называется графиком запуска. Выполнение транзакций по одной без их чередования называется последовательным графиком запуска, а непоследовательное выполнение транзакций – чередующимся графиком запуска или непоследовательным графиком запуска. Два графика называются эквивалентными, если при их выполнении будет получен одинаковый результат, независимо от исходного состояния базы данных. Таким образом, график запуска является верным (т.е. допускающим возможность упорядочения), если он эквивалентен некоторому последовательному графику запуска.

При выполнении двух различных последовательных графиков запуска, содержащих одинаковый набор транзакций, можно получить совершенно различные результаты. Поэтому выполнение двух различных чередующихся графиков запуска с одинаковыми транзакциями может также привести к различным результатам, которые могут быть восприняты как верные.

Теорема двухфазной блокировки (не имеет отношения к протоколу двухфазной фиксации), которая может быть сформулирована следующим образом:

Если все транзакции подчиняются "протоколу двухфазной блокировки", то для всех возможных чередующихся графиков запуска существует возможность упорядочения.

При этом протокол двухфазной блокировки, в свою очередь, формулируется следующим образом.

1. Перед выполнением каких-либо операций с некоторым объектом (например, с кортежем базы данных) транзакция должна заблокировать этот кортеж.
2. После снятия блокировки транзакция не должна накладывать никаких других блокировок.

Таким образом, транзакция, которая подчиняется этому протоколу, характеризуется двумя фазами: фазой наложения блокировки и фазой снятия блокировки.

Характеристика упорядочения может быть выражена следующим образом. Если A и B являются любыми двумя транзакциями некоторого графика запуска, допускающего возможность упорядочения, то либо A логически предшествует B, либо B логически предшествует A, т.е. либо B использует результаты выполнения транзакции A, либо A использует результаты выполнения транзакции B. (Если транзакция A приводит к обновлению кортежей р, q, ... r и транзакция B использует эти кортежи в качестве входных данных, то используются либо все обновленные с помощью A кортежи, либо полностью не обновленные кортежи до выполнения транзакции A, но никак не их смесь.) Наоборот, график запуска является неверным и не подлежит упорядочению, если результат выполнения транзакций не соответствует либо сначала выполнению транзакции A, а затем транзакции B, либо сначала выполнению транзакции B, а затем транзакции A.

В настоящее время с целью понижения требований к ресурсам и, следовательно, повышения производительности и пропускной способности в реальных системах обычно предусмотрено использование не двухфазных транзакций, а транзакций с "ранним снятием блокировки" (еще до выполнения операции прекращения транзакции) и наложением нескольких блокировок. Однако следует понимать, что использование таких транзакций сопряжено с большим риском. Действительно, при использовании недвухфазной транзакции A предполагается, что в данной системе не существует никакой другой чередующейся с ней транзакции B (в противном случае в системе возможно получение ошибочных результатов).

## Уровни изоляции транзакции

Термин уровень изоляции, грубо говоря, используется для описания степени вмешательства параллельных транзакций в работу некоторой заданной транзакции. Но при обеспечении возможности упорядочения не допускается никакого вмешательства, иначе говоря, уровень изоляции должен быть максимальным. Однако, как уже отмечалось, в реальных системах по различным причинам обычно допускаются транзакции, которые работают на уровне изоляции ниже максимального.

Уровень изоляции обычно рассматривается как некоторое свойство транзакции. В реальных СУБД может быть реализовано различное количество уровней изоляции.

Кроме того помимо кортежей могут блокироваться другие единицы данных, например целое отношение, база данных или (пример противоположного характера) некоторое значение атрибута внутри заданного кортежа.

## Поддержка в языке SQL

SQL поддерживает операции COMMIT и ROLLBACK для фиксации и отката транзакции соответственно.

Специальный оператор SET TRANSACTION используется для определения некоторых характеристик транзакции, которую нужно будет инициировать, такие, как режим доступа и уровень изоляции.

В стандарте языка SQL не предусмотрена поддержка явным образом возможности блокировки (фактически, блокировка в нем вообще не упоминается). Блокировки накладываются неявно, при выполнении операторов SQL.

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 354–392.

# Технологии СУБД

16.1 Распределенные базы данных

16.2 Принципы функционирования распределенной БД

16.3 Системы типа клиент/сервер

16.4 Серверы баз данных

## Распределенные базы данных

### Предварительные замечания.

Системы дистрибутивных баз данных состоят из набора узлов, связанных вместе коммуникационной сетью, в которой:

1. каждый узел обладает своими собственными системами баз данных;
2. узлы работают согласованно, поэтому пользователь может получить доступ к данным на любом узле сети, как будто все данные находятся на его собственном узле.

Из этого следует, что так называемая "распределенная база данных" на самом деле является типом виртуального объекта, части которого физически сохраняются в ряду отдаленных "реальных" баз данных на удаленных узлах (фактически, это логическая единица всех этих реальных баз данных).

Каждый узел обладает своими собственными базами данных, собственными локальными пользователями, собственной СУБД и программным обеспечением для управления транзакциями (включая собственное программное обеспечение для блокирования, регистрации, восстановления и т.д.), а также своим собственным локальным диспетчером передачи данных. В частности, пользователь может на собственном локальном узле выполнять операции с данными так, как будто этот узел вовсе не является частью распределенной системы.

## Принципы функционирования распределенной БД

Теперь, после краткого введения, можно привести формулировку фундаментального принципа распределенной базы данных: для пользователя распределенная система должна выглядеть точно так же, как нераспределенная система.

1. Изложенный фундаментальный принцип приводит к следующему набору правил и целе9ф:Локальная автономия;Независимость от центрального узла;Непрерывное функционирование;Независимость от расположения
2. Независимость от фрагментации;
3. Независимость от репликации;
4. Обработка распределенных запросов;
5. Управление распределенными транзакциями;
6. Независимость от аппаратного обеспечения;
7. Независимость от операционной системы;
8. Независимость от сети;
9. Независимость от СУБД.

### Локальная автономия

В распределенной системе узлы следует делать автономными. Локальная автономия означает, что операции на данном узле управляются этим узлом, т.е. функционирование любого узла X не зависит от успешного выполнения некоторых операций на каком-то другом узле Y (в противном случае может возникнуть крайне нежелательная ситуация, а именно: выход из строя узла Y может привести к невозможности исполнения операций на узле X, даже если с узлом X ничего не случилось). Из принципа локальной автономии также следует, что владение и управление данными осуществляется локально вместе с локальным ведением учета.

### Независимость от центрального узла

Под локальной автономией подразумевается, что все узлы должны рассматриваться как равные. Следовательно, не должно существовать никакой зависимости и от центрального "основного" узла с некоторым централизованным обслуживанием, например централизованной обработкой запросов, централизованным управлением транзакциями или централизованным присвоением имен.

Зависимость от центрального узла нежелательна по крайней мере по двум причинам. Во-первых, центральный узел может быть "узким" местом всей системы, а во-вторых, более важно то, что система в таком случае становится уязвимой, т.е. при повреждении центрального узла может выйти из строя вся система.

### Непрерывное функционирование

Одним из основных преимуществ распределенных систем является то, что они обеспечивают более высокую надежность и доступность.

1. Надежность (вероятность того, что система исправна и работает в любой заданный момент) повышается благодаря работе распределенных систем не по принципу "все или ничего", а в постоянном режиме; т.е. работа системы продолжается, хотя и на более низком уровне, даже в случае неисправности некоторого отдельного компонента, например отдельного узла.
2. Доступность (вероятность того, что система исправна и работает в течение некоторого промежутка времени) повышается частично по той же причине, а частично благодаря возможности репликации данных (подробнее это описывается ниже).

### Независимость от расположения

Основная идея независимости от расположения (которая также называется прозрачностью расположения) достаточно проста: пользователям не следует знать, в каком физическом месте хранятся данные, наоборот, с логической точки зрения пользователям следовало бы обеспечить такой режим, при котором создается впечатление, что все данные хранятся на их собственном локальном узле.

### Независимость от фрагментации

В системе поддерживается фрагментация данных, если некое хранимое отношение в целях физического хранения можно разделить на части, или "фрагменты". Фрагментация желательна для повышения производительности системы, поскольку данные лучше хранить в том месте, где они наиболее часто используются. При такой организации многие операции будут чисто локальными, что снизит нагрузку на сеть.

Существует два основных типа фрагментации – горизонтальная и вертикальная, которые связаны с реляционными операциями выборки и проекции соответственно. Иначе говоря, фрагментом может быть любое произвольное подчиненное отношение, которое можно вывести из исходного отношения с помощью операций выборки и проекции. При этом следует учесть приведенные ниже допущения.

Предполагается без утраты общности, что все фрагменты данного отношения независимы, т.е. ни один из фрагментов не может быть выведен из других фрагментов либо иметь выборку или проекцию, которая может быть выведена из других фрагментов. (Если есть необходимость сохранить одну и ту же информацию в нескольких разных местах, для этого следует использовать механизм репликации системы.).Проекции не должны допускать потерю информации.

### Независимость от репликации

В системе поддерживается независимость от репликации, если заданное хранимое отношение или заданный фрагмент могут быть представлены несколькими различными копиями, или репликами, хранимыми на нескольких различных узлах.

### Обработка распределенных запросов.

Вопрос оптимизации более важен для распределенной, нежели для централизованной системы. Основная причина заключается в том, что для выполнения охватывающего несколько узлов запроса существует довольно много способов перемещения данных по сети. В таком случае чрезвычайно важно найти наиболее эффективную стратегию.

### Управление распределенными транзакциями.

Существует два основных аспекта управления обработкой транзакций, а именно: управление восстановлением и управление параллелизмом, каждому из которых в распределенных системах должно уделяться повышенное внимание.

### Независимость от аппаратного обеспечения.

Подразумевает возможность работы узлов системы на разном аппаратном обеспечении.

### Независимость от операционной системы.

Подразумевает возможность работы узлов системы под управлением различных операционных систем.

### Независимость от сети.

Подразумевает возможность работы узлов системы в гетерогенных сетях, с использованием различного сетевого оборудования

### Независимость от СУБД.

Эта цель подразумевает использование несколько менее точной формулировки предположения о строгой однородности. В новой форме это предположение означает, что все экземпляры СУБД на различных узлах поддерживают один и тот же интерфейс, хотя они не обязательно должны быть копиями одного и того же программного обеспечения.

### Распространение обновления

Как указывалось выше, основной проблемой репликации данных является то, что обновление любого логического объекта должно распространяться на все хранимые копии этого объекта. Трудности возникают из-за того, что некоторый узел, содержащий данный объект, может быть недоступен (например, из-за краха системы или данного узла) именно в момент обновления.

Общая схема устранения этой проблемы (и не единственно возможная в этом случае), называемая схемой первичной копии, будет описана далее.

1. Одна копия каждого реплицируемого объекта называется первичной копией, а все остальные – вторичными.
2. Первичные копии различных объектов находятся на различных узлах (таким образом, эта схема является распределенной).
3. Операции обновления считаются завершенными, если обновлены все первичные копии. В таком случае в некоторый момент времени узел, содержащий такую копию, несет ответственность за распространение операции обновления на вторичные копии.

## Системы типа клиент/сервер

Системы клиент/сервер могут рассматриваться как особый случай распределенных систем. Точнее, система клиент/сервер является распределенной системой, в которой одни узлы являются клиентами, а другие – серверами,, все данные хранятся на серверах, все приложения исполняются клиентами, а "места их соединения не скрыты от пользователя" (т.е. не достигается цель независимости от расположения).

Термин "клиент/сервер" относится преимущественно к архитектуре или логике распределения ответственности, поэтому клиент – это приложение, т.е. внешний интерфейс, а сервер – СУБД, т.е. внутренний интерфейс для непосредственной работы с базами данных. Таким образом, система может быть четко разделена на две части с использованием двух разных типов компьютеров. Эта последняя возможность настолько притягательна, что термин "клиент/сервер" стал применяться почти исключительно в случаях, когда клиент и сервер действительно находятся на разных компьютерах.

Реальное распространение архитектуры "клиент-сервер" стало возможным благодаря развитию и широкому внедрению в практику концепции открытых систем.

Основным смыслом подхода открытых систем является упрощение комплексирования вычислительных систем за счет международной и национальной стандартизации аппаратных и программных интерфейсов. Главной побудительной причиной развития концепции открытых систем явились повсеместный переход к использованию локальных компьютерных сетей и те проблемы комплексирования аппаратно-программных средств, которые вызвал этот переход. В связи с бурным развитием технологий глобальных коммуникаций открытые системы приобретают еще большее значение и масштабность.

Ключевой фразой открытых систем, направленной в сторону пользователей, является независимость от конкретного поставщика. Ориентируясь на продукцию компаний, придерживающихся стандартов открытых систем, потребитель, который приобретает любой продукт такой компании, не попадает к ней в рабство. Он может продолжить наращивание мощности своей системы путем приобретения продуктов любой другой компании, соблюдающей стандарты.

## Серверы баз данных

Обычно для обозначения всей СУБД, основанной на архитектуре "клиент-сервер", включая и серверную, и клиентскую части используют термин "сервер баз данных". Такие системы предназначены для хранения и обеспечения доступа к базам данных.

Доступ к базе данных от прикладной программы или пользователя производится путем обращения к клиентской части системы. В качестве основного интерфейса между клиентской и серверной частями выступает язык баз данных SQL.

В типичном на сегодняшний день случае на стороне клиента СУБД работает только такое программное обеспечение, которое не имеет непосредственного доступа к базам данных, а обращается для этого к серверу с использованием языка SQL.

Следует отметить и другие варианты этой основной темы.

1. Несколько клиентов могут использовать один и тот же сервер (действительно, это довольно обычная ситуация).
2. Один клиент может осуществить доступ к нескольким серверам. Эта ситуация может быть реализована двумя путями.
	1. Клиент в заданный момент времени может осуществить доступ только к одному серверу, т.е. каждый отдельный запрос к базе данных должен быть направлен только к одному серверу и внутри одного запроса нельзя комбинировать данные, размещенные на двух или более серверах. Кроме того, пользователь должен знать, какие данные и на каких серверах хранятся.
	2. Клиент в заданный момент времени может осуществить доступ к нескольким серверам, т.е. в отдельном запросе к базе данных могут комбинироваться данные, размещенные на двух или более серверах. Это значит, что данные для пользователя выглядят так, как если бы они действительно были размещены на одном сервере. В таком случае пользователю не нужно знать, какие данные и на каких серверах хранятся.

Второй способ, по сути, является формулировкой истинной распределенной системы, смысл которой не соответствует широко распространенному смыслу термина "клиент/сервер".

**Литература:**

1. Дейт К.Дж. Введение в системы баз данных. –Пер. с англ. –6-е изд. –К. Диалектика, 1998. Стр. 564–590.

# Современные постреляционные модели БД

17.1 Системы управления базами данных следующего поколения

17.2 Ориентация на расширенную реляционную модель

17.3 Объектно-ориентированные СУБД

## Системы управления базами данных следующего поколения

В этом разделе очень кратко рассматриваются основные направления исследований и разработок в области так называемых постреляционных систем, т.е. систем, относящихся к следующему поколению (хотя термин "next-generation DBMS" зарезервирован для некоторого подкласса современных систем).

Хотя отнесение СУБД к тому или иному классу в настоящее время может быть выполнено только условно (например, иногда объектно-ориентированную СУБД O2 относят к системам следующего поколения), можно отметить три направления в области СУБД следующего поколения. Чтобы не изобретать названий, будем обозначать их именами наиболее характерных СУБД.

1. Направление Postgres. Основная характеристика: максимальное следование (насколько это возможно с учетом новых требований) известным принципам организации СУБД (если не считать коренной переделки системы управления внешней памятью).
2. Направление Exodus/Genesis. Основная характеристика: создание собственно не системы, а генератора систем, наиболее полно соответствующих потребностям приложений. Решение достигается путем создания наборов модулей со стандартизованными интерфейсами, причем идея распространяется вплоть до самых базисовых слоев системы.
3. Направление Starburst. Основная характеристика: достижение расширяемости системы и ее приспосабливаемости к нуждам конкретных приложений путем использования стандартного механизма управления правилами. По сути дела, система представляет собой некоторый интерпретатор системы правил и набор модулей-действий, вызываемых в соответствии с этими правилами. Можно изменять наборы правил (существует специальный язык задания правил) или изменять действия, подставляя другие модули с тем же интерфейсом.

В целом можно сказать, что СУБД следующего поколения - это прямые наследники реляционных систем.

## Ориентация на расширенную реляционную модель

Одним из основных положений реляционной модели данных является требование нормализации отношений: поля кортежей могут содержать лишь атомарные значения. Для традиционных приложений реляционных СУБД – банковских систем, систем резервирования и т.д. – это вовсе не ограничение, а даже преимущество, позволяющее проектировать экономные по памяти БД с предельно понятной структурой.

Однако с появлением эффективных реляционных СУБД их стали пытаться использовать и в менее традиционных прикладных системах – САПР, системах искусственного интеллекта и т.д. Такие системы обычно оперируют сложно структурированными объектами, для реконструкции которых из плоских таблиц реляционной БД приходится выполнять запросы, почти всегда требующие соединения отношений.

Приведение исходного табличного представления предметной области к "плоскому" виду является обязательным первым шагом в процессе проектирования реляционной базы данных на основе принципов нормализации. С другой стороны, абсолютно очевидно, что такое "уплощение" таблиц хотя и является необходимым условием получения неизбыточной и "правильной" схемы реляционной базы данных, в дальнейшем потенциально вызывает выполнение многочисленных соединений, наличие которых может свести на нет все преимущества "хорошей" схемы базы данных.

В "ненормализованных" реляционных моделях данных допускается хранение в качестве элемента кортежа кортежей (записей), массивов (регулярных индексированных множеств данных), регулярных множеств элементарных данных, а также отношений. При этом такая вложенность может быть, по существу, неограниченной. К настоящему времени фактически полностью сформировано теоретическое основание реляционных баз данных с отказом от нормализации.

### Абстрактные типы данных

Одной из наиболее известных СУБД третьего поколения является система Postgres, (создатель этой системы М.Стоунбрекер).Одно свойство системы Postgres сближает ее со свойствами объектно-ориентированных СУБД. В Postgres допускается хранение в полях отношений данных абстрактных, определяемых пользователями типов.

### Генерация систем баз данных, ориентированных на приложения

Идея очень проста: никогда не станет возможно создать универсальную систему управления базами данных, которая будет достаточна и не избыточна для применения в любом приложении. Например, если посмотреть на использование универсальных коммерческих СУБД, то можно легко увидеть, что по крайней мере в 90% случаев применяется не более чем 30% возможностей системы. Тем не менее, приложение несет всю тяжесть поддерживающей его СУБД, рассчитанной на использование в наиболее общих случаях.

Поэтому очень заманчиво производить не законченные универсальные СУБД, а нечто вроде компиляторов (сompiler compiler), позволяющих собрать систему баз данных, ориентированную на конкретное приложение (или класс приложений). Существуют как минимум два экспериментальных прототипа таких систем – Genesis и Exodus.

### Поддержка исторической информации и темпоральных запросов

Обычные БД хранят мгновенный снимок модели предметной области. Любое изменение в момент времени t некоторого объекта приводит к недоступности состояния этого объекта в предыдущий момент времени. Самое интересное, что на самом деле в большинстве развитых СУБД предыдущее состояние объекта сохраняется в журнале изменений, но возможности доступа со стороны пользователя нет.

Конечно, можно явно ввести в хранимые отношения явный временной атрибут и поддерживать его значения на уровне приложений. Более того, в большинстве случаев так и поступают. Недаром в стандарте SQL появились специальные типы данных date и time. Но в таком подходе имеются несколько недостатков: СУБД не знает семантики временного поля отношения и не может контролировать корректность его значений; появляется дополнительная избыточность хранения (предыдущее состояние объекта данных хранится и в основной БД, и в журнале изменений); языки запросов реляционных СУБД не приспособлены для работы со временем.

Существует отдельное направление исследований и разработок в области темпоральных БД. В этой области исследуются вопросы моделирования данных, языки запросов, организация данных во внешней памяти и т.д. Основной тезис темпоральных систем состоит в том, что для любого объекта данных, созданного в момент времени t1 и уничтоженного в момент времени t2, в БД сохраняются (и доступны пользователям) все его состояния во временном интервале [t1,t2].

## Объектно-ориентированные СУБД

Направление объектно-ориентированных баз данных (ООБД) возникло сравнительно давно. Публикации появлялись уже в середине 1980-х. Однако наиболее активно это направление развивается в последние годы. С каждым годом увеличивается число публикаций и реализованных коммерческих и экспериментальных систем.

В компьютерных технологиях сегодня отчетливо просматривается стремление с минимальными потерями перенести в виртуальный мир объекты мира реального. Объектно-ориентированная СУБД – именно то средство, которое обеспечивает запись объектов в базу данных "как есть". Данное обстоятельство стало решающим аргументом в пользу выбора ООСУБД для переноса семантики объектов и процессов реального мира в сферу информационных систем.

Использование объектного подхода к проектированию систем поднимает роль ООСУБД как средства для наиболее естественного хранения и манипулирования создаваемыми объектами. Несмотря на наличие многих теоретических проблем, ключевой из которых, безусловно, является сложность строгой формализации объектной модели данных, многие эксперты полагают, что за этими системами будущее.

Единого мнения по поводу того, как конкретно следует организовывать ООСУБД, нет. Тем не менее, можно указать ряд непременных свойств, которым они должны удовлетворять. Эти свойства продекларированы в "Манифесте систем объектно-ориентированных баз данных", а впоследствии закреплены в документах ODMG, организации, объединяющей ведущих производителей ООСУБД, ставящей своей целью выработать стандарты, соблюдение которых обеспечивало бы переносимость приложений. Используемая в статье терминология отражает требования стандарта ODMG 2.0, однако при описании примеров, взятых из различных коммерческих ООСУБД, авторы в первую очередь опирались на документацию соответствующих производителей.

### Модель данных ООБД

В соответствии со стандартом ODMG 2.0 объектная модель

Базовыми примитивами являются объекты и литералы. Каждый объект имеет уникальный идентификатор, литерал не имеет идентификатора.

1. Объекты и литералы различаются по типу. Все элементы одного типа имеют одинаковый диапазон изменения состояния (множество свойств) и одинаковое поведение (множество определенных операций). Объект, на который можно установить ссылку, называется экземпляром; он хранит определенный набор данных.
2. Состояние объекта определяется набором значений, реализуемых множеством свойств. Этими свойствами могут быть атрибуты объекта или связи между объектом и одним или несколькими другими объектами.
3. Поведение объекта определяется набором операций, которые могут быть выполнены над объектом или самим объектом. Операции могут иметь список входных и выходных параметров строго определенного типа. Каждая операция может также возвращать типизированный результат.
4. База данных хранит объекты, позволяя совместно использовать их различным пользователям и приложениям. База данных основана на схеме данных, определяемой языком определения данных, и содержит экземпляры типов, определенных схемой.

рис. 17.1. Основные элементы ООСУБД.

Объект

Объект

…

Объект

Тип

Объект

Поведение

Состояние

Операция

Свойства

Атрибуты

Связи

Каждый тип имеет внешнюю спецификацию и одну или несколько реализации. Спецификация определяет внешние характеристики типа: пользователю для работы с объектом предоставляется набор операций и набор атрибутов объекта, при помощи которых можно работать с реальными экземплярами. Реализация определяет внутреннее содержание объектов, например операции.

Тип также является объектом. Поддерживается иерархия супертипов и подтипов, реализуя стандартный механизм объектно-ориентированного программирования – наследование.

ООСУБД обслуживает множество баз данных, каждая из которых содержит определенное множество типов. В базах данных могут содержаться объекты соответствующего типа из этого множества. Тип имеет набор свойств, а объект характеризуется состоянием в зависимости от значения каждого свойства. Операции, определяющие поведение типа, едины для всех объектов одного типа. Свойство едино для всего типа, а все объекты типа также имеют одинаковый набор свойств. Значение свойства относится к конкретному объекту.

### Идентификатор объекта

Каждый объект в базе данных уникален. Существует несколько подходов для идентификации объекта. Самый простой – присвоить ему уникальный номер (OID – object identificator) в базе и никогда больше не повторять этот номер, даже если пре базах могут оказаться объекты одного класса, а уникальность номеров соблюдается только в пределах одной базы. Преимущество подхода – в простоте извлечения объектов нужного класса: объекты одного класса будут иметь идентификатор, имеющий общую часть. Идеальный же вариант – использование OID, состоящего из трех частей: номер базы, номер класса, номер объекта. Однако и при этом остается вопрос о том, как обеспечить уникальность номеров баз и классов на глобальном уровне – при использовании ООСУБД на различных платформах, в разных городах и странах.

### Новые типы данных

Одним из принципиальных отличий объектных баз данных от реляционных является возможность создания и использования новых типов данных. Концептуально объект характеризуется поведением и состоянием. Определение типа заключается в определении поведения, т.е. операций, которые могут быть выполнены объектом или над состоянием объекта – набором атрибутов определенных типов (атрибут может иметь любой объявленный в базе тип). Важная особенность ООСУБД состоит в том, что создание нового типа не требует модификации ядра базы и основано на принципах объектно-ориентированного программирования: инкапсуляции, наследовании, перегрузке операций и позднем связывании.

табл. 17.1. Различие атрибутов типа и объекта

|  |  |  |
| --- | --- | --- |
| Атрибуты типа | Значение атрибутов типа | Значение атрибутов объекта |
| Название подразделения | Подразделение учета входящей корреспонденции | Нет |
| Название входящего документа | Нет | Постановление правительства № 357 |
| Запрос на уточнение данных о продукте |
| Поздравление с праздником |

Как правило, в ООСУБД для объектов, которые предполагается хранить в базе, (постоянные объекты), требуется, чтобы их предком был конкретный базовый тип, определяющий все основные операции взаимодействия с сервером баз данных.

Поэтому для создания своего типа необходимо унаследовать свойства любого имеющегося типа, наиболее подходящего по своему поведению и состоянию к типу, который требуется получить, расширить недостающие операции и атрибуты и переопределить, по необходимости, уже имеющиеся.

Пример на рис. 17.2 иллюстрирует возможность наращивания типа "Человек", который может быть "Мужчиной", "Женщиной", "Взрослым" или "Ребенком". Соответственно возможны попарные пересечения этих типов. Каждый из этих типов может иметь свой набор свойств и операций. Любой объект типа "Мужчина", "Женщина", "Взрослый" или "Ребенок" является объектом типа "Человек". Аналогично объект подтипа "Мужчина Взрослый", полученного наследованием типов "Мужчина" и "Взрослый" является человеком, мужского пола, взрослым и, соответственно, может пользоваться свойствами и операциями всех своих супертипов.

рис. 17.2. Пример наследования типов.

Человек

Мужчина

Мужчина Взрослый

Женщина

Женщина Взрослая

Взрослый

Мужчина Ребенок

Ребенок

Женщина Ребенок

Функционирование базы основано на схеме данных. Как уже отмечалось в определении объектной модели, любой тип является объектом, следовательно, схемы данных являются уровнем интерпретации специфических служебных объектов, использующих свойства этих объектов как схему для создания новых типов. Схема данных может быть как первичной, для создания классов, которые собирается использовать программист, так и вторичной, выделяемой из созданных на языке программирования (скажем, на C++) классов и загружаемой в базу. Язык ODL разработан ODMG как универсальный язык описания объектов и не претендует на то, чтобы называться полноценным языком программирования. Для целей разработки этой же организацией предусмотрены элементы расширения классических объектных языков C++, Smalltalk, Java, позволяющих описать структуру объектов, их связи и типы связей.

В объектных базах данных также различают два вида операций и атрибутов, распространяющих свое действие только на конкретный экземпляр или на весь тип.

В примере из табл. 17.1 определен тип "Подразделение учета входящей корреспонденции", атрибут "Название подразделения". Все объекты входящей корреспонденции данного подразделения могут относиться только к этому подразделению и, для того, чтобы не засорять базу данных лишней информацией, название подразделения хранится в коде пользовательских программ, а не в объектах, хранимых в базе. Это справедливо в том случае, если название подразделения не меняется.

Для атрибутов объектов, значение которых может меняться, такой механизм не подходит: атрибут "Название входящего документа" в различных экземплярах может иметь различные значения. Аналогичный механизм применим и для операций: пользователь может определять операции для конкретного экземпляра. Базы данных, которые могут хранить код операции в экземпляре типа и способны его выполнить, называются активными объектными базами.

### Оптимизация ядра СУБД

Ядро ООСУБД оптимизировано для операций с объектами. Естественными операциями для него являются кэширование объектов, ведение версий объектов, разделение прав доступа к конкретным объектам. Ядро объектно-реляционной СУБД остается реляционным, а "объектность" реализуется в виде специальной надстройки. Как следствие, ООСУБД свойственно более высокое быстродействие на операциях, требующих доступа и получения данных, упакованных в объекты, по сравнению с реляционными СУБД, для которых необходимость выборки связных данных ведет к выполнению дополнительных внутренних операций.

### Язык СУБД и запросы

Общепризнанны две группы вариантов языков запросов. Первая объединяет языки, унаследованные от SQL и представляют собой разновидность OQL (Object Query Language), языка, стандартизованного ODMG. Хотя он существенно отличается от SQL, некоторое подобие явно прослеживается. Некоторые функции принципиально не могут быть реализованы для объектных баз данных только средствами SQL (например, функции модификации структуры данных). Это обусловлено, прежде всего, возможной потерей связи с объектами в прикладных программах. Объектно-реляционные СУБД используют различные варианты ограниченных объектных расширений SQL.

Вторая, сравнительно новая (применяется с 1998 года) группа языков запросов базируется на XML. Собирательное название языков этой группы — XML QL (или XQL). Они могут применяться в качестве языков запросов в объектных и объектно-реляционных базах данных. Использование в чисто реляционных базах не целесообразно, поскольку языком предусматривается объектная структура запроса.

### Транзакции в ООСУБД

В соответствии со стандартом ODMG 2.0 транзакции представляют логический блок, гарантирующий атомарность (atomicity), целостность (consistency), изолированность (isolation) и долговечность (durability).

Короткие транзакции характеризуются малым временем выполнения; они могут существовать только в рамках сеанса работы с ООСУБД. Это наиболее простой вид транзакций, реализованный во всех современных СУБД. Все объекты подлежащие изменениям блокируются, а после принятия транзакции разблокируются, изменения же записываются в базу данных.

Длинные транзакции предназначены для увеличения производительности при групповой работе. Реализовано это следующим образом. Например, в Versant можно создавать персональные и групповые базы. Для снижения вычислительной нагрузки на центральный сервер при одновременной работе большого числа пользователей в Versant предлагает пользователям, постоянно работающим с определенными объектами, возможность организации персональной базы данных. Пользователи работают со своей базой, а объекты из нее синхронизируются с групповой базой данных. Пользователь, начав длинную транзакцию, тем самым отмечает объекты, с которыми предстоит работать в групповой базе данных (операция "поставить на контроль" – check out). Эти объекты копируются в его персональную базу, а в групповой базе блокируются, причем блокировать их можно как на запись, так и на чтение. В групповой базе создается объект, содержащий все данные о длинных транзакциях. В случае повреждения групповой базы или физического отключения сервера групповой базы пользователь сможет продолжать работу с объектами в своей персональной базе, а после восстановления групповой базы – синхронизировать объекты. Перед завершением длинной транзакции пользователь должен поместить все измененные объекты обратно в основную базу (операция "зарегистрировать" – check in). После этого объекты копируются в основную базу, а блокировка снимается. В случае аварийного завершения длинной транзакции все изменения будут потеряны.

Вложенные транзакции по принципу функционирования аналогичны коротким. В процессе выполнения одной транзакции формируются другие. Если в текущем сеансе работает один процесс, то создается стек, а если несколько процессов – дерево транзакций.

### Блокировки в ООСУБД

Назначение блокировок – гарантировать монопольность использования объекта конкретным пользователем с целью предотвращения одновременного изменения данных. В соответствии с терминологией, принятой в системе Versant, существуют короткие, продолжительные и оптимистические блокировки.

Короткие блокировки (short lock) предназначены для обеспечения последовательного доступа к данных при многопользовательском режиме работы. Они автоматически выполняются во время выполнения коротких транзакций.

Продолжительные блокировки (persistent lock) обеспечивают блокирование объектов на продолжительное время – часы, дни, недели. Применяются совместно с длинными транзакциями. При этом объект может быть заблокирован несколькими способами: с исключением снятия другим процессом (hard lock); с возможностью снятия другим процессом (soft lock); по конкретным операциям.

### Ведение версий

В "Манифесте объектно-ориентированных баз данных", поддержка множественных версий объектов отнесена к необязательным характеристикам ООСУБД. Однако большинство современных коммерческих ООСУБД поддерживает версионность; более того, именно она рассматривается многими авторами как отличительная черта ООСУБД (это не совсем так, поскольку и в других СУБД, используя специфические приемы, можно поддержать версионность: но, безусловно, в ООСУБД реализация такого механизма будет гораздо проще).

Важность версионности для ООСУБД обусловлена их историческими корнями: считается, что версионность появилась для решения задач автоматизированного проектирования, выделившись постепенно в самостоятельный класс систем. Для САПР характерна задача сохранения многих версий одного и того же проекта. Впрочем, поддержка версионности может оказаться полезной и для других приложений. Достаточно указать на делопроизводство, где также необходима поддержка многих версий (редакций) документа, при этом высока вероятность появления запросов типа: "Найти редакцию проекта контракта по состоянию на 1.12.2000". Кроме того, версионность способствует повышению надежности информационной системы в целом: пользователи модифицируют не сами объекты, а их версии, а окончательные изменения происходят на сервере лишь после выполнения специальных процедур. Это уменьшает вероятность порчи информации из-за ошибок оператора или вследствие каких-то умышленных действий.

В качестве примера вновь рассмотрим ООСУБД Versant. Для всех объектов возможно сохранение всех версий их изменения. Создается граф происхождения версий, поддерживающий ряд свойств.

1. Доступ к любой ранее сохраненной версии. Благодаря этому свойству возможно извлечение из базы данных, например, случайно удаленных данных.
2. Установка версии по умолчанию. Это свойство объясняет причину возникновения ветвей в графе происхождения версий, так как любая порожденная версия создаст ветвь от установленной текущей, не последней версии.
3. Удаление версии. Вполне логичным является необходимость удаления некоторых версий (например, промежуточных или случайно порожденных) из графа происхождения версий.

### Физические хранилища

Один из ключевых моментов функционирования любой СУБД — хранилище данных. Обычно база данных объединяет несколько хранилищ, каждое из которых ассоциируется с одним или несколькими файлами. Хранятся как метаданные (класс, атрибут, определения операций), так и пользовательские данные. Выделяют три типа хранилищ:

1. системное хранилище (system store) используется для хранения системы классов, создается на этапе установки Jasmine и содержит информацию о семействах классов и пользовательских хранилищ. Возможно определение пользовательских хранилищ для помещения в них описаний пользовательских семейств классов.
2. пользовательское хранилище (user store) служит для хранения пользовательских объектов, например Persons, Projects или Locations. Пользовательские хранилища могут располагаться внутри системного.
3. рабочее хранилище (work store) предназначено для поддержки временной информации в ходе сеанса работы, например результатов поисковых запросов.

Другой пример – организация хранилищ в ООСУБД Versant, все базы данных в которой делятся на групповые и персональные. База данных состоит из нескольких физических разделов:

1. системный раздел (system volume), который создается автоматически при создании базы данных;
2. раздел данных (data volume), который может быть добавлен для увеличения доступного размера базы;
3. раздел логического протокола (logical log volume) и раздел физического протокола (physical log volume) предназначены для хранения служебной информации о транзакциях, блокировках для обеспечения возможности отката транзакций и восстановления базы.

**Литература:**

1. Аркадий Андреев, Дмитрий Березкин, Роман Самарев, "Внутренний мир объектно-ориентированных СУБД" \\ Открытые системы , март 2001, стр 47-57.

Приложение А. Структура учебной базы данных. (БД деканата ВУЗа).

**Students**

|  |  |  |  |
| --- | --- | --- | --- |
| StNo | GrNo | StName | CityNo |
| 1 | 1 | Иванов | 1 |
| 2 | 1 | Петров | 3 |
| 3 | 1 | Сидоров | 3 |
| 4 | 2 | Стрельцов | 4 |
| 5 | 2 | Кузнецов | 4 |

**Teachers**

|  |  |  |  |
| --- | --- | --- | --- |
| TNo | TName | TPosition | TChiefNo |
| 1 | Иванов | Ректор | NULL |
| 2 | Петров | Зав. кафедрой | 1 |
| 3 | Сидоров | Преподаватель | 2 |
| 4 | Стрельцов | Лаборант | 2 |

**Groups**

|  |  |  |
| --- | --- | --- |
| GrNo | EnterYear | GrName |
| 1 | 1998 | А–98–51 |
| 2 | 1999 | Б–99–51 |
| 3 | 1998 | Б–98–51 |

**Subjects**

|  |  |
| --- | --- |
| SubjNo | SubjName |
| 1 | Высшая математика |
| 2 | Физика |
| 3 | История Украины |
| 4 | Основы информатики |
| 5 | Программирование |

**WorkPlan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GrNo | SubjNo | Semester | TNo | Hours |
| 1 | 1 | 1 | 3 | 2 |
| 2 | 1 | 1 | 3 | 4 |
| 1 | 5 | 1 | 2 | 2 |

**Marks**

|  |  |  |  |
| --- | --- | --- | --- |
| StNo | SubjNo | DocNo | Mark |
| 1 | 1 | 127 | 5 |
| 1 | 5 | 128 | 4 |
| 2 | 1 | 127 | 3 |
| 2 | 5 | 128 | 3 |
| 3 | 1 | 127 | 5 |
| 3 | 5 | 128 | 3 |
| 4 | 1 | 129 | 4 |
| 5 | 1 | 129 | 5 |

**Cities**

|  |  |  |
| --- | --- | --- |
| CityNo | CityName | RgNo |
| 1 | Желтые Воды | 1 |
| 2 | Кривой Рог | 1 |
| 3 | Пятихатки | 1 |
| 4 | Львов | 2 |

**Regions**

|  |  |
| --- | --- |
| RgNo | RgName |
| 1 | Днепропетровская |
| 2 | Львовская |

Приложение Б.Описание структуры учебной базы данных.

|  |  |
| --- | --- |
| Отношение "Students" содержит информацию об обучающихся студентах.StNo – уникальный, в БД, код студентаGrNo – код группы, в которой обучается студентStName – фамилия студентаStCity – Код города, в котором проживет студентОтношение "Cities" содержит информацию о городахCityNo – код городаCityName – название городаRgNo – код области, в которой расположен городОтношение "Regions" содержит информацию об областяхRgNo – код областиRgName – название областиОтношение "Teachers" содержит информацию о сотрудниках и преподавателяхTNo - уникальный, в БД, код сотрудника (преподавателя)TName – фамилия преподавателяTPosition – должность, занимаемая преподавателемTChiefNo – код сотрудника, являющегося руководителем для данного преподавателя | Отношение "Groups" содержит информацию о группахGrNo – код группыEnterYear – год образования группыGrName – название группыОтношение "WorkPlan" содержит информацию о рабочем плане. Рабочий план определяет, какие дисциплины, какой группе, кем, в каком объеме и когда читаются.GrNo – код группыSubjNo – код дисциплиныSemester - семестрTNo – код преподавателяHours – количество часовОтношение "Subjects" содержит информацию о предметахSubjNo – код предметаSubjName – название предметаОтношение "Marks" содержит информацию о полученных студентами оценкахStNo – код студентаSubjNo – код дисциплиныDocNo – номер документа (экз. ведомость, разрешение на пересдачу и т.д.)Mark - оценка |