Кафедра прикладной математики и кибернетики ПетрГУ

Конспект лекций по дисциплине «Базы данных» («Управление данными»)

составитель: Л. В. Щёголева

Петрозаводск 2015

Содержание

Список литературы

1. Крёнке Д. Теория и практика построения баз данных. - СПб: Питер, 2003. - 800с.
2. Карпова Т.С. Базы данных: модели, разработка, реализация. СПб: Питер, 2001. – 304с.
3. Кляйн К., Кляйн Д., Хант Б. SQL. Справочник. - СПб.:Символ-Плюс, 2010. - 656с. (есть в электронной библиотеке ПетрГУ)
4. Хансен Г., Хансен Дж. Базы данных: разработка и управление. Бином, 2000. - 704с.
5. Дейт К. Дж. Введение в системы баз данных. К., М., СПб.: Издательский дом «Вильямс», 2000. – 848с.
6. Michael Kifer, Arthur Bernstein, Philip M. Lewis Database Systems: an Application-Oriented Approach. – Introductionary version, Second Edition. Pearson, Addison Wesley 2005. P. 666.
7. Хансен Г., Хансен Дж. Базы данных: разработка и управление. Бином, 2000. – 704 с.
8. Тиори Т., Фрай Дж. Проектирование структур баз данных. Т.1,2, - М.: Мир,1985. -287с., -320с.
9. Диго С.М. Проектирование и использование баз данных: Учебник для вузов. - М.: Финансы и статистика, 1995. – 208 с.
10. Ульман Дж. Основы систем баз данных. М: Финансы и статистика, 1983. – 334с.
11. Тиори Т., Фрай Дж. Проектирование структур баз данных. Т.1,2, М.: Мир,1985. – 287с., –320с.
12. Цикритзис Д., Лоховски Ф. Модели данных. М: Финансы и статистика, 1985. – 343с.
13. Мейер Д. Теория реляционных баз данных. М: Мир, 1987. – 608с.

Глава 1. Введение

§ 1 Основные определения

Что такое база данных?
Для чего нужна база данных?

База данных – совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными (database system).

Данные – информация, фиксированная в определенной форме, пригодной для последующей обработки, хранения и передачи (от латинского слова datum, означающего в переводе факт).

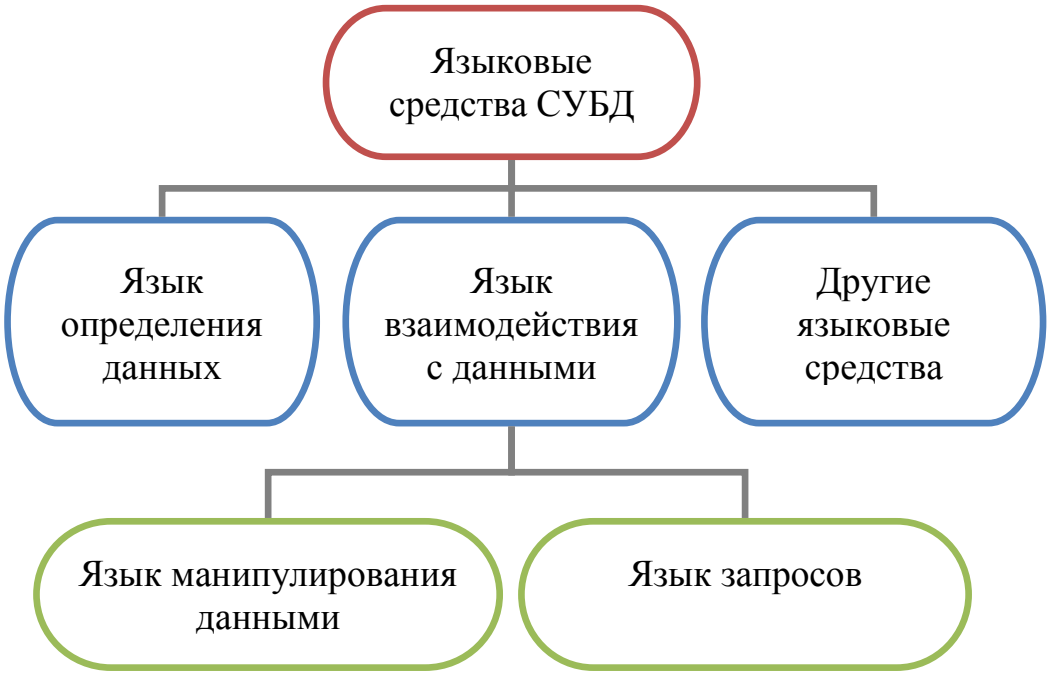
Для работы с данными было разработано специальное программное обеспечение:

Система управления базами данных (СУБД) – комплекс программ и языковых средств, предназначенных для создания, ведения и использования баз данных.

СУБД обеспечивает выполнение стандартных операций с данными в БД:

- ввод данных,
- редактирование данных,
- удаление данных,
- поиск данных, выполнение запросов.

Другая обработка данных возлагается на прикладные программы. Использование специального программного обеспечения для работы с данными предоставляет некоторые преимущества.
Основная функция СУБД – предоставление пользователю базы данных возможности работать с данными, не вникая в детали на уровне аппаратного обеспечения.



В базе данных обычно хранится информация из некоторой сферы деятельности, например, данные по отгрузке готовых изделий со склада, или по успеваемости студентов факультета, или по оплате телефонных разговоров и т. п. Эту сферу деятельности мы будем называть **предметной областью**. Таким образом, база данных будет содержать информацию из предметной области, которую необходимо хранить и обрабатывать с помощью компьютера.

Информация, хранящаяся в базе данных, описывает текущее состояние предметной области, например, количество денег в настоящий момент на каждом счете клиентов. Когда изменяется ситуация в реальном мире, например, были добавлены деньги на счет, соответствующее изменение информации должно произойти и в базе данных. Все действия над данными выполняются с помощью транзакции.

Транзакция – последовательность операций, производимых над базой данных, и переводящих базу данных из одного согласованного состояния в другое согласованное состояние.

Свойства современных СУБД

Доступность – в любой момент времени можно выполнить операции над базой данных, например, система резервирования авиа билетов.

Надежность – быстрое восстановление после отказов (ошибок) и сохранение данных (благодаря механизму транзакций).

Параллельность – параллельное выполнение операций над базой данных (благодаря механизму транзакций).

Мощность – приемлемое время выполнения действий над базой данных (благодаря механизму индексации записей).

Масштабируемость – наращивание мощности аппаратуры и ПО.

Безопасность – обеспечение защиты от несанкционированного доступа к данным и выполнения операций над данными.

Поддержка решений

В обычной базе данных выполняются простые запросы: Сколько штук товара было продано вчера?

Для принятия стратегических решений, рассчитанных на продолжительный срок действия, планирования необходимо выполнить более сложные запросы: В течение зимних месяцев последних пяти лет, какой процент покупателей супермаркетов, расположенных на Древлянке, покупали крекеры и суп одновременно? Для того чтобы разместить эти товары рядом на полке. Такой запрос не требует мгновенного ответа, а может занять некоторое время, в отличие от текущих запросов к базе данных.

В базе данных обычно содержится текущая информация, необходимая для работы. Архивы информации собираются в специальные **хранилища данных** (**data warehouse**), которые обновляются через заданные интервалы времени, например, ежедневно или ежемесячно, или каждую декаду месяца и т. д.

Для принятия решений используется специальная технология OLAP (online analytic processing).

Запрос может быть более неопределенным: есть ли какие-нибудь интересные комбинации товаров, приобретаемых покупателями? Выполнение таких запросов называется **data mining**. Это уже получение знаний, а не получение данных. Это относится к области искусственного интеллекта.

Одним из ответов на запрос было установлено: Ранним вечером большой процент мужчин, которые одновременно покупают пеленки и пиво, являются папами и собираются провести вечер дома вместе с маленьким ребенком.

Классификация СУБД

1. По языкам общения:
   - открытые (системы с базовым языком – имеется язык программирования, расширенный операциями ЯМД.) (для обращения к БД используется язык программирования с операторами ЯМД, требует знания логической структуры БД);
   - замкнутые (имеются специальные языки запросов для общения с пользователями, возможность взаимодействия массового, неподготовленного пользователя с БД в режиме диалога.) (имеют самостоятельный язык общения пользователей с БД)
   - смешанные.
2. По выполняемым функциям:
   - информационные;
   - операционные.
3. По сфере применения:
   - универсальные
   - специализированные (проблемно-ориентированные).
4. По экономико-организационным признакам:
   - бесплатные и платные;
   - самоокупаемые (бесприбыльные) и прибыльные;
   - государственные и негосударственные
   - общедоступные и с ограниченным кругом пользователей.
5. По способу организации обработки данных:
   - локальные
   - распределенные.
6. По поддерживаемой модели:
   - иерархические
   - сетевые
   - реляционные
   - объектные

- смешанные
- мультимодельные
- географические ИС (ГИС)

Преимущества использования СУБД:

- Возможность централизации управления данными.
- Сокращение избыточности данных и их многократное использование.
- Обеспечение секретности и защита данных от несанкционированного доступа.
- Обеспечение целостности данных.
- Возможность доступа к данным в реальном времени.
- Простые средства общения с БД.
- Возможность обработки непредсказуемых запросов.
- Простота расширения и корректировки БД.
- Возможность оптимального использования ресурсов ЭВМ.
- Возможность обеспечения независимости данных.

Недостатки СУБД:

- Сложность разработки и проектирования.
- Потеря эффективности для отдельных приложений.
- Значительная часть ресурсов расходуется на поддержку самой СУБД.
- Большая чувствительность к сбоям.

В целом, положительные аспекты использования СУБД преобладают над недостатками. Однако свой нынешний вид СУБД приобрели не сразу.

§ 2 История развития баз данных

Первые компьютеры использовались только для выполнения сложных расчетов, позднее, когда появились более совершенные запоминающие устройства, их стали использовать для накопления и обработки данных.

Конец 50-х начало 60-х – организации и предприятия накапливают и хранят данные в виде файлов или наборов данных. Файл – набор записей одного типа. Запись – последовательность элементов данных. Прикладные программы реализуют все функции работы с данными, записи на запоминающие устройства, обработку запросов.

Понятие БД появилось в конце 60-х годов.

I этап: Простые файлы данных (начало 60-х гг.)

Об организации данных приходилось заботиться при написании прикладных программ, и делалось это элементарным способом т.е. данные обычно организовывались в виде простых последовательных файлов на магнитной ленте. Независимость данных отсутствовала. Для того чтобы обновить файл, нужно было записать новый. Старый файл сохранялся и назывался отец.

Особенности:

- Файл – последовательный файл на магнитной ленте.
- Физическая и логическая структуры данных идентичны.
- Программист определяет физическое расположение данных и включает формирование физической структуры в прикладные программы.
- Файл создается и используется для одной прикладной программы (высокая степень избыточности).
- Обработка в пакетном режиме без доступа в реальном времени.
- Хранятся несколько копий одного и того же файла, т.к. предшествующие поколения данных сразу не уничтожаются.
- Типовое программное обеспечение выполняет только операции ввода/вывода.
- При изменении структуры данных или запоминающих устройств требовалось переписывать все программы.

Недостатки файловых систем:

- Зависимость от аппаратного обеспечения.
- Затраты на получение информации превосходят ценность самой информации. Чтобы выполнить один запрос, необходимо написать программу.

- Избыточность данных: изменение и добавление данных сопровождалось созданием нового файла, старый файл сохранялся.
- Дублирование информации: каждая прикладная программа имела свои файла с данными ⇒
- Противоречивость данных: одна прикладная программа внесла изменения, а другая нет.
- Дублирование прикладными программами одних и тех же функций работы с данными, нерациональное использование вычислительных ресурсов, отсутствие стандартного интерфейса с пользователем.
- Отсутствие универсальных средств описания данных.
- Сильная чувствительность к изменениям в предметной области или структуре файлов.
- Способ организации данных и способ доступа к ним встроены в прикладную программу.

II этап: Методы доступа к файлу (конец 60-х гг.)

Второй этап характеризуется изменением, как природы файлов, так и устройств, на которых они запоминались. Предпринимается попытка оградить прикладного программиста от влияния изменений в аппаратуре. Программное обеспечение допускает возможность изменения физического расположения данных без изменения при этом их логического представления при условии, что содержимое записей или основная структура файлов не изменяется.

Особенности:

- Логическая и физическая структуры файла различаются между собой, но взаимосвязь между ними достаточно простая.
- Запоминающие устройства можно менять без изменения прикладных программ.
- Возможен последовательный или произвольный доступ к записям (но не к полям).
- Типовое программное обеспечение обработки данных представляет собой методы доступа, но не управления данными.
- Обработка пакетная, оперативная или в реальном времени.
- Возможен последовательный или произвольный доступ к записям.

III этап: Первые системы управления БД (начало 70-х гг.)

По мере развития средств обработки данных становилось ясно, что прикладные программы желательно сделать независимыми не только от изменений в аппаратных средствах хранения файлов и от увеличения размеров файлов, но также и от добавления к хранимым данным новых полей и новых взаимосвязей. Встала задача объединения различных приложений, работающих с одними данными. Программное обеспечение БД должно располагать средствами отображения файловой структуры прикладного программиста в такую физическую структуру данных, которая запоминается на реальном носителе и наоборот.

Особенности:

- Физическая структура данных независима от прикладных программ.
- Различные логические файлы могут быть получены из одних и тех же физических данных.
- Элементы данных являются общими для различных приложений. Доступ к одним и тем же данным может осуществляться различными приложениями по различным путям, отвечающим требованиям этих приложений.
- Данные адресуются на уровне полей или групп.

IV этап: Современные СУБД

Довольно скоро стало очевидным, что необходим дополнительный уровень независимости данных. Общая логическая структура данных, как правило, слоная, и по мере роста БД она неизбежно изменяется. Поэтому важно обеспечить возможность изменения общей логической структуры без изменения при этом использующих ее многочисленных прикладных программ. Поэтому требуется два уровня независимости данных.

Особенности:

- Программные средства обеспечивают логическую и физическую независимость данных: **Логическая независимость данных** означает, что общая логическая структура данных может быть изменена без изменения прикладных программ (изменение, конечно, не должно заключаться в удалении из БД таких элементов, которые используются прикладными программами). **Физическая независимость данных** означает, что физическое расположение и организация данных могут изменяться, не вызывая при этом изменений ни общей логической структуры данных, ни прикладных программ.
- Минимальная избыточность.

- Отсутствие противоречий.
- Программное обеспечение выполняет все основные действия по вводу, выводу, хранению и поиску информации.
- Появляется возможность получения ответов на незапланированные запросы.
- Программное обеспечение предусматривает использование языковых средств (языка описания данных, языка программиста, языка пользователя и т.п.).
- Лучшее обеспечение сохранности и секретности.

Результаты развития концепций БД

- Методология проектирования БД, организации данных, модели данных.
- Программное обеспечение, реализующее универсальные функции обработки и хранения данных
- Языковые средства

Хронология развития принципов баз данных

1879	Готтлоб Фредж (Gottlob Frege) основал современную математическую логику
1937	Алан Тьюринг сформулировал принципы машины Тьюринга
1941	Альфред Тарски (Alfred Tarski) применил теорию моделей для вычисления бинарных отношений
1962	Чарльз Бачман (Charles Bachman) разработал Интегрированную систему управления данными (IDMS – Integrated Data Management System) для сетевых моделей баз данных
1966	IBM разработала (ISAM – Indexed Sequential Access Method)
1970--1971	Эдгар Кодд представил реляционную модель данных на основе реляционной алгебры и реляционных вычислений
1971	Рудольф Байер и Эдвард Макгрей (Rudolf Bayer and Edward McCreight) опубликовали статью о B-деревьях как мощный механизм индексации
1972	Бойс-Кодд (Boyce-Codd) ввели нормальные формы
1973	Чарльз Бачман (Charles Bachman) получил премию Тьюринга за исследования в области сетевой модели баз данных
1974	Теория функциональных зависимостей и нормализации была сформулирована
1976	Eswaran, Gray, Lorie, and Traiger определили уровни изолированности, сериализуемость и двухфазную фиксацию, которые во второй половине 70-х были реализованы во многих СУБД
1976	Peter Chen создал модель сущность-связь
1977	Akifumi Makinouchi описал nested relational model, a precursor of the object-relational model
1979	Home H. Gallaire and Jack Minker представили базу данных с логическим выводом ныне известные как дедуктивные базы данных
1979	Fagin, Nievergelt, Pippenger and Strong define Extensible Hashing
1981	Кодд получил премию Тьюринга за вклад в теорию баз данных
1985	Появление активных баз данных
1988	Дедуктивные и объектно-ориентированные базы данных объединены в единую модель
1985-	
-1993	Развивается технология объектно-ориентированных и объектно-реляционных баз данных
1995	Появление кубов OLAP
1995	Развиваются полуструктурированные модели данных
1998	Jim Gray получил премию Тьюринга за вклад в развитие концепций баз данных и транзакций

Хронология развития приложений баз данных

1956	IBM изобрела первый жесткий диск
1959	G.E. выпустила 32 компьютера (ERMA) для Bank of America внеся первый вклад в компьютеризацию банковской системы
1963	Американские авиалинии ввели систему резервирования билетов (SABRE)
1968	IBM выпускает IMS (Information Management System) – первая коммерческая СУБД
1971	CODASYL публикует отчет Data Base Task Group для сетевой модели
1974	появление языка запросов QUEL для СУБД Ingres
1975	IBM выпускает СУБД System R – экспериментальная реляционная СУБД с языком запросов SEQUEL, который позднее стал называться SQL (система позднее стала DB2)
1975	IBM предлагает графический язык запросов QBE
1976	IBM создает язык SQL
1977	Relational Software Inc., которая в дальнейшем стала Oracle основывает компанию и является первой компанией, создающей реляционную СУБД на основе модели IBM System R и языка

	SQL
1983	IBM создает СУБД DB2
1986	Ingres создает первую распределенную СУБД IngresStar
1986	GemStone создает первую объектно-ориентированную СУБД
1986	The American National Standards Institute (ANSI) публикует стандарт SQL 1.0
1986	LDL создается логический язык баз данных, реализованный в MCC Corporation
1987	RAID массив
1991	WWW
1992	создается технология Open Database Connectivity (ODBC) для взаимодействия с различными СУБД
1993	ODMG 1.0 опубликован для объектно-ориентированных СУБД
1998	UML
1998	XML
1999	SQL3
2003	SQL-2003

Глава 2. Уровни представления информации и этапы проектирования базы данных

Вряд ли в компьютере требуется хранить всю информацию о реальном мире. Скорее всего, перед Вами встанет более узкая задача, например, с помощью компьютера организовать учет отпуска готовых изделий со склада, или успеваемости студентов факультета, или оплаты телефонных разговоров. Это всего лишь часть реального мира, которую мы будем называть предметной областью.

Предметная область – часть реального мира, требующая автоматизированной обработки.

В результате развития концепций баз данных было выделено три уровня представления данных из предметной области.

Инфологический – на этом уровне информация о предметной области представляется вне зависимости от того, какие программные и технические средства будут использованы для хранения и обработки этой информации. На этом уровне предметная область описывается в терминах классов объектов и их взаимосвязей, которые являются понятными конечным пользователям и людям, работающим в предметной области.

Даталогический (концептуальный) – на этом уровне информация представляется в виде данных и логических связей между данными вне зависимости от того, что представляют собой данные и какие технические средства будут использованы для хранения данных, но с учетом программных средств.

Физический – на этом уровне определяется как и где на физическом носителе будут храниться данные.

Внешний – параллельный уровень представления информации с точки зрения одного пользователя, включающий инфологическое и даталогическое представления.

Этапы проектирования базы данных:

- Построение инфологической модели
- Построение внешних моделей
- Построение реляционной/объектной модели
- Физическое проектирование

Глава 3. Инфологическая модель предметной области

§ 1 Описание предметной области

База данных содержит информацию не о предприятии, а для предприятия – информацию, обработку которой необходимо автоматизировать для эффективного функционирования предприятия.

Обследование предметной области.

1. Краткое описание основных функций и структуры предприятия
   - Основные функции: Цель работы предприятия
   - Результаты деятельности: продукция (объемы, номенклатура), услуги (виды, клиенты)
   - Структура управления (схема подчинения, подразделения, службы, отделы)
   - Взаимодействие подразделений
   - Описание каждого подразделения
   - Другие общие сведения
2. Анализ требований
   - Производственные процессы:
     a) объекты
     b) действия
     c) информация
   - Пользователи (рабочие места):
     a) информационные потребности пользователей
     b) операции – обработка информации: вид, результаты, сроки
     c) запросы (поиск, печать)
     d) входные и выходные формы
   - Информационные потоки
     a) какая информация
     b) от кого и кому передается
     c) с какой периодичностью
     d) как и кем обрабатывается
     e) собираемая статистика

§ 2 Инфологическая модель

Модель предметной области (инфологическая модель) – это описание предметной области, выполненное без ориентации на используемые в дальнейшем программные и технические средства.

Цель инфологического проектирования заключается в представлении семантики (т. е. смысла) предметной области. Эта модель должна быть понятна заказчику, который не является специалистом в области баз данных.

Для описания предметной области наиболее часто используется модель «сущность–связь», предложенная П. Ченом в 1976 году (или ее модификации). Сокращенно такую модель называют ER-моделью (ERD) от английского названия «Entity–Relationship» («Сущность–связь»). Диаграмма модели имеет лексикографическую структуру, т. е. включает в себя текст и элементы графики. Из названия модели понятно, что основными ее структурными элементами будут объекты и связи между ними. Рассмотрим каждый из структурных элементов.

§ 2.1 *Объекты и классы объектов*

Предметная область состоит из объектов, в качестве которых могут выступать люди, материальные предметы, производственные отделы, бумажные документы, операции, явления, абстрактные идеи. Например, объектами могут быть: студент Иванов И.И., группа 22301, экономический факультет, стипендия студента Иванова И.И., зачетная книжка Иванова И.И., и т.п.

Объект – это предмет или идея, который может быть четко идентифицирован.

Представить в модели каждый объект предметной области не представляется возможным, это уже будет не модель, а сам реальный мир. Модель предполагает некоторое обобщение и сведение всего к общей структуре. Поэтому, среди всех конкретных объектов необходимо выделить их общие признаки, характеристики и по ним объединить объекты в **классы**. Так Петрова, Иванова и Колпакова можно объединить в класс Студент, а Карандаш, Пенал, Бумага – в класс Товар.

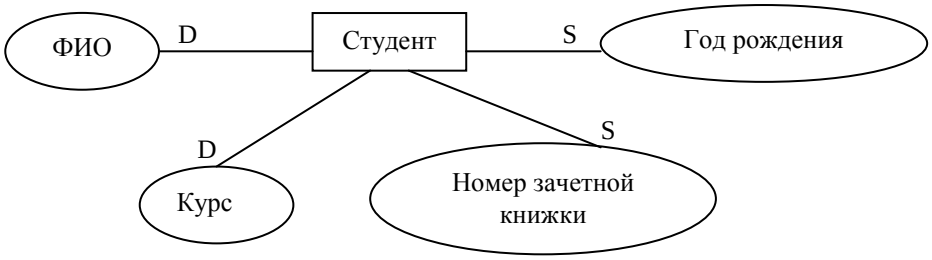
Каждый класс определяется набором атрибутов, т.е. **свойств**, которыми обладает каждый объект, принадлежащий этому классу. Например, класс Студент может иметь следующие свойства: ФИО, Год рождения, Курс обучения, Номер зачетной книжки, а класс Товар: Наименование, Дата изготовления, Цена.

Свойства могут носить статический или динамический характер, что означает, что значение свойства не может измениться или может измениться с течением времени. Например, значение свойства Год рождения для класса Студент не может измениться, а значение свойства Курс обучения того же класса будет изменяться каждый учебный год, если, конечно, это добросовестный студент.

Сведения, относящиеся к классам объектов и взаимосвязям между ними необходимо описать на некотором формальном языке. Одним из таких языков описания модели предметной области являются ER-диаграммы (Модель "Объект-отношение", Чен, 1976 год (Chen P.P.-S. The Entity-Relationship Model)). Диаграмма имеет лексикографическую структуру, т.е. включает в себя текст и элементы графики.

В терминах этой структуры классы объектов обозначаются прямоугольниками, а свойства – овалами. Внутри прямоугольника записывается название класса, внутри овала – название свойства. Изменчивость свойств на схеме обозначается латинскими буквами S и D. Буква S означает статический характер свойства, буква D – динамический.

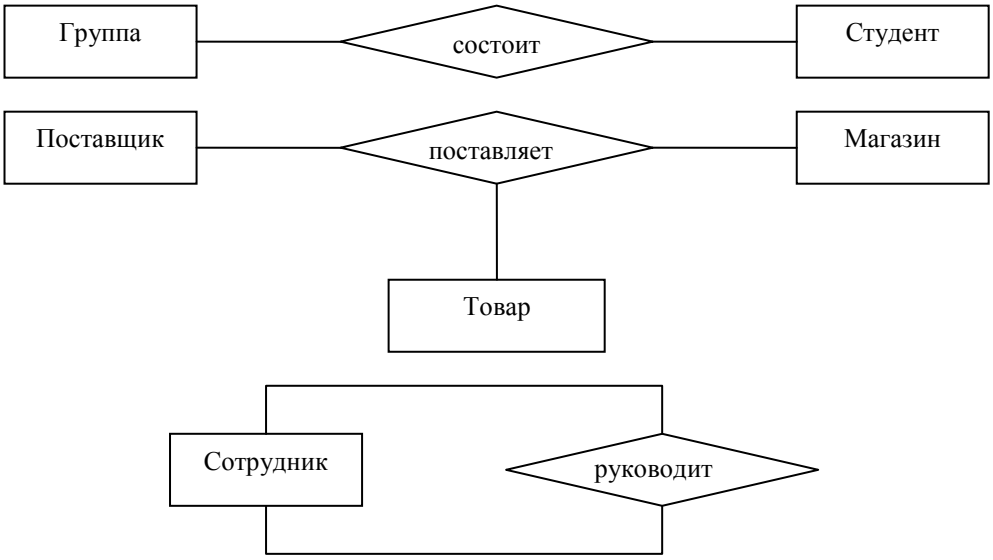
На рисунке ниже представлен класс Студент, имеющий четыре свойства, два из которых являются статическими (Год рождения и Номер зачетной книжки), а два других – динамическими (ФИО и Курс).



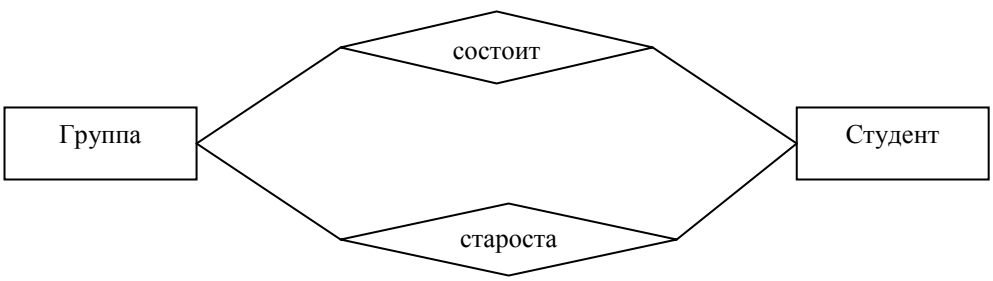
§ 2.2 *Связи между классами объектов*

Между классами объектов могут существовать некоторые отношения, называемые связями. Например, между классами Студент и Группа существует связь: студенты распределены по группам или можно сказать, что группы состоят из студентов.

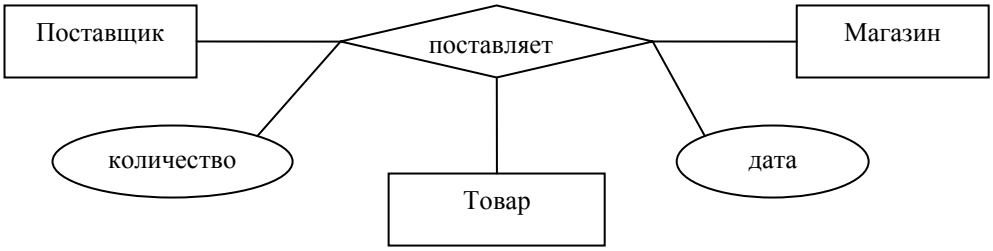
Связи могут быть бинарными, т.е. между двумя классами объектов, или между бóльшим количеством классов. Можно определить и циклические связи, т.е. связи между объектами одного класса.



Между двумя классами может существовать одновременно несколько связей. Например, между классами Студент и Группа можно образовать связь «распределение студентов по группам» и связь «староста группы».



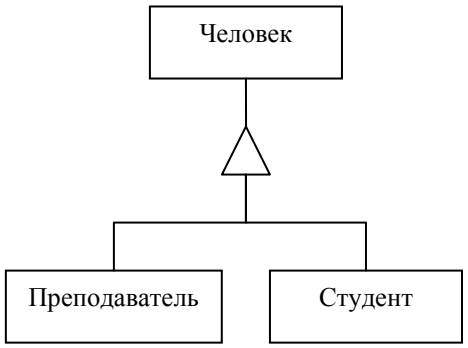
Связи, как и классы, могут иметь свойства. Например, между двумя классами Товар и Поставщик существует связь, которую назовем Поставка, т.е. поставщик поставляет товар. Эта связь будет иметь два свойства – дата поставки и количество. Понятно, что и дата, и количество не могут быть свойствами ни класса Товар, ни класса Поставщик. Это характеристики связи между двумя объектами (классами).



Классы участвуют в связях, играя при этом разные роли. Например, в связи «руководит» между двумя объектами одного класса Служащий. Один класс играет роль руководителя, а второй – подчиненного. А например для связи Состоит между классами Группа и Студент роли одинаковые.

В терминах ER-диаграмм связь изображается в виде ромба, внутри которого записывается название связи. Классы объектов, участвующие в этой связи соединяются с ромбом стрелками. Стрелки могут быть подписаны ролями классов.

Иерархические связи изображаются треугольником и называются «IsA». Например, Студент и Преподаватель являются Человеком. Такие связи могут иметь дополнительные ограничения: покрытие – когда объединение множеств объектов (реализаций) подклассов равно множеству объектов (реализаций) суперкласса, например, человек – суперкласс, рабочий и студент – подклассы, все студенты и рабочие являются людьми, и нет людей, не являющихся ни рабочим, ни студентом, но могут быть люди, являющиеся и студентом, и рабочим. Второе ограничение – несвязность, когда объекты подклассов одного суперкласса не связаны между собой, например, подклассы первокурсники и второкурсники могут быть потомками суперкласса студент, и они не связаны друг с другом.



Связи «Целое-часть» бывают двух типов: часть может существовать самостоятельно (неэксклюзивная), а может существовать только вместе с целым (эксклюзивная). Неэксклюзивная связь не имеет специального обозначения, используется обычный ромб. Для обозначения эксклюзивной связи используется ромб с двойной рамкой и прямоугольник с двойной рамкой для класса, являющегося частью.

Свойства связи изображаются, так же как и свойства класса и соединяются с ромбом, обозначающим связь.

§ 2.3 *Типы связей в предметной области*

§ 2.3.1 *Характеристика однозначности для бинарных связей*

1. Связь 1:1, **«один-к-одному»**. Эта связь означает, что каждому объекту из первого класса соответствует ровно один объект из второго класса и, наоборот, каждому объекту из второго класса соответствует ровно один объект из первого класса. В отношение вступают два объекта из разных классов.

Например, между классами: Факультет и Декан. Факультет может иметь только одного декана, а декан может быть деканом только одного факультета. На схеме такая связь изображается двумя одинарными стрелками.

2. Связь 1:M, **«один-ко-многим»** (или M:1, «многие-к-одному»). Эта связь означает, что одному объекту из первого класса соответствует несколько объектов второго класса, но каждому объекту второго класса соответствует только один объект первого класса. В такое отношение вступают несколько объектов – один объект из первого класса, остальные из второго, и если какой-то объект из второго класса уже входит в отношение с каким-то объектом из первого класса, то он уже не может входить в такое же отношение с другим объектом первого класса.

   Такой характер носит связь между классами: Группа и Студент. В группе может быть несколько студентов, но каждый студент может быть только в одной группе. Или отношение между классами Факультет и Кафедра. Факультет может иметь несколько кафедр, но каждая кафедра принадлежит только одному факультету.

   На схеме связь изображается одной одинарной и одной двойной стрелками. Двойная стрелка направлена к тому классу, объекты которого могут входить в отношение в любом количестве.

3. Связь M:N, **«многие-ко-многим»**. Эта связь означает, что одному объекту первого класса соответствует несколько объектов второго класса, и каждому объекту второго класса соответствует несколько объектов первого класса. В отношение входит несколько объектов от одного и от другого класса.

   Например, связь между классами Преподаватель и Предмет. Преподаватель может читать несколько предметов, и каждый предмет могут читать несколько преподавателей.

   На схеме такая связь изображается двумя двойными стрелками.

§ 2.3.2 *Характеристика полноты для бинарных связей*

Характеристика полноты отражает зависимость класса от связи, в которую он входит.

1. Необязательная по отношению к классу связь означает, что существование объектов класса не зависит от наличия связи.
2. Обязательная по отношению к классу связь означает, что существование объектов класса зависит от наличия связи.

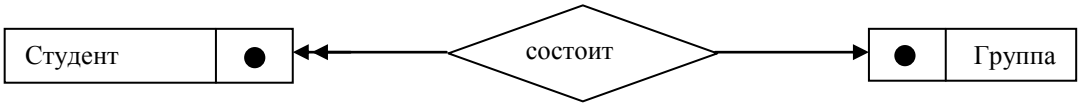
На схеме обязательная по отношению к классу связь обозначается квадратом с точкой внутри, примыкающим к прямоугольнику, представляющему класс, перед входящей в него стрелкой связи.

Например:

1. Связь между классами Факультет и Компьютерный класс. В университете есть факультеты, не имеющие компьютерных классов, и есть компьютерные классы, не приписанные ни к одному из факультетов, т. е. и компьютерные классы, и факультеты существуют вне зависимости от того, относится ли компьютерный класс к факультету и имеет ли факультет хотя бы один компьютерный класс. Такая связь является необязательной и по отношению к классу Факультет, и по отношению к классу Компьютерный класс.

2. Связь между классами Служащий и Должность. Каждый сотрудник в организации должен иметь какую-либо должность, т. е. нет сотрудника без должности. Это означает, что не существует объекта класса Служащий, не связанного с каким-либо объектом класса Должность. Но должность может быть и вакантной, т. е. быть не связанной ни с одним объектом класса Служащий. Такая связь является обязательной по отношению к классу Служащий и необязательной по отношению к классу Должность.

3. Связь между классами Студент и Группа. Не может быть студента, не принадлежащего ни к одной группе, так же как не может быть группы без студентов. Такая связь является обязательной и по отношению к классу Студент, и по отношению к классу Группа.



§ 2.3.3 *Характеристика изменчивости связи*

1. **Статическая** связь – это связь, не изменяющаяся во времени. Например, между классами Договор и Сотрудник, заключивший договор. Если договор заключен определенным сотрудником, то эта связь уже никогда не изменится, другой сотрудник не сможет занять место того, кто заключил договор.
2. **Динамическая** связь – это связь, которая может измениться с течением времени. Например, связь между классами Сотрудник и Должность носит динамический характер. Сотрудник может перейти на другую должность или уволиться, тогда на эту же должность будет назначен другой сотрудник.

Общая схема, содержащая все классы объектов, их свойства, связи между классами и свойства этих связей, называется концептуальной моделью предметной области.

После того, как построена концептуальная модель предметной области, можно переходить к построению модели данных.

§ 2.4 *Качество инфологической модели данных*

В этом параграфе изложены некоторые принципы проверки качества и полноты информационной модели (источник - Richard Barker, Case Method: Entity Relationship Modelling, Addison-Wesley, 1990)

§ 2.4.1 *Качество класса*

Основной гарантией качества класса является ответ на вопрос, действительно ли объект является классом, то есть важным объектом или явлением, информация о котором должна храниться в базе данных.

Список проверочных вопросов для класса:

- Отражает ли имя класса суть данного объекта?
- Нет ли пересечения с другими классами?
- Имеются ли хотя бы два свойства?
- *Всего свойств не более восьми?
- Есть ли синонимы/омонимы данного класса?
- Есть ли уникальный идентификатор?
- Имеется ли хотя бы одна связь?
- Существует ли хотя бы одна функция по созданию, поиску, корректировке, удалению, архивированию и использованию значений свойств класса?
- Ведется ли история изменений состояний класса?
- Нет ли такого же класса в другой базе данных, возможно, под другим именем?
- Не имеет ли класс слишком общий смысл?
- Достаточен ли уровень обобщения, воплощенный в классе?

§ 2.4.2 *Качество подкласса*

- Отсутствуют ли пересечения с другими подклассами?
- Имеет ли подкласс какие-нибудь свойства и/или связи?
- Имеют ли они все свои собственные уникальные идентификаторы или наследуют один на всех от суперкласса?
- Имеется ли исчерпывающий набор подклассов?
- Не является ли подкласс примером вхождения класса?
- Знаете ли вы какие-нибудь свойства, связи и условия, отличающие данный подкласс от других подклассов?

§ 2.4.3 *Качество свойства*

Следует выяснить, а действительно ли это свойство, то есть, описывает ли оно тем или иным образом данный класс.

Список проверочных вопросов для свойства:

- Является ли наименование свойства существительным единственного числа, отражающим суть обозначаемой характеристики?
- Не включает ли в себя наименование свойства имя класса (этого быть не должно)?
- Имеет ли свойство только одно значение в каждый момент времени?
- Отсутствуют ли повторяющиеся значения (или группы)?
- Описаны ли формат, длина, допустимые значения, алгоритм получения и т.п.?
- Не может ли это свойство быть пропущенным классом, который пригодился бы для другой прикладной системы (уже существующей или предполагаемой)?
- Не может ли свойство быть пропущенной связью?
- Нет ли где-нибудь ссылки на свойство как на "особенность проекта", которая при переходе на прикладной уровень должна исчезнуть?
- Есть ли необходимость в истории изменений?
- Зависит ли его значение только от данного класса?
- Если значение свойства является обязательным, всегда ли оно известно?
- Есть ли необходимость в создании домена для этого и ему подобных свойств?
- Зависит ли его значение только от какой-то части уникального идентификатора?
- Зависит ли его значение от значений некоторых свойств, не включенных в уникальный идентификатор?

§ 2.4.4 *Качество связи*

Нужно выяснить, отражают ли связи действительно важные отношения, наблюдаемые между классами.

Список проверочных вопросов для связи:

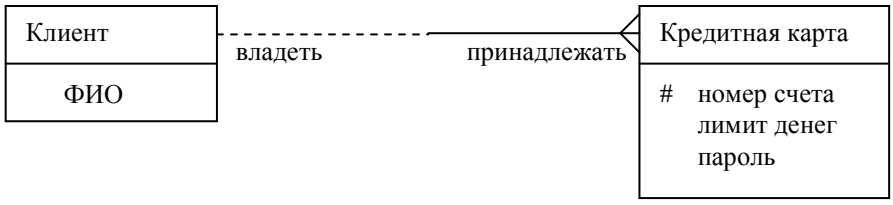
- Имеется ли ее описание для каждой участвующей стороны, точно ли оно отражает содержание связи и вписывается ли в принятый синтаксис?
- Участвуют ли в ней только две стороны?
- *Не является ли связь переносимой?
- Заданы ли степень связи и обязательность для каждой стороны?
- Допустима ли конструкция связи?
- Не относится ли конструкция связи к редко используемым?
- Не является ли она избыточной?
- Не изменяется ли она с течением времени?
- Если связь обязательная, всегда ли она отражает отношение к классу, представляющему противоположную сторону?

Для исключающей связи:

- Все ли концы связей, покрываемые исключающей дугой, имеют один и тот же тип обязательности?
- Все ли из них относятся к одному и тому же классу?
- Обычно дуги пересекают разветвляющиеся концы – что вы можете сказать о данном случае?
- Связь может покрываться только одной дугой. Так ли это?
- Все ли концы связей, покрываемые дугой, входят в уникальный идентификатор?

§ 2.5 *Другие ER модели предметной области*

Нотация Баркера.



Имя класса и список его свойств записывается в прямоугольнике. Знак # ставится перед ключом. Все связи бинарные, обязательные отмечаются сплошной линией, необязательные – пунктирной. Читается связь отдельно для каждого конца: каждый клиент может владеть одной или несколькими кредитными картами; каждая кредитная карта должна принадлежать ровно одному клиенту. Однозначность связи обозначается одной линией, многозначность – вилкой.

Обязательность связи обозначается перпендикулярной чертой (хотябы один объект), необязательность – кружком (может быть ноль объектов).

§ 2.6 *Построение инфологической модели*

1. Выделить основные классы объектов и их свойства
2. Определить связи между классами, их тип и свойства
3. Определить ограничения предметной области: на значения данных, на связи, другие

Обратить внимание на:

1. Область возможных значений данных и максимальное число экземпляров класса (объектов)
2. Как часто меняются данные
3. Старение данных и необходимость ведения архива
4. Отчетные документы
5. Оперативность получения первичных данных

Глава 4. Реляционная модель данных

§ 1 Модель данных

Модель данных определяет правила, в соответствии с которыми структурируются данные. Однако, для интерпретации семантики данных только структурных спецификаций недостаточно, также должны быть специфицированы операции над данными и ограничения целостности.

Модель данных – это тройка <S, O, C>, где S – **структура данных** (объекты данных, правила порождения), O – множество **операций**, которые переводят базу данных из одного допустимого состояния в другое (правила вывода), C – **ограничения целостности**, определяющим множество непротиворечивых состояний базы данных (целостность – точность и корректность).

В качестве структуры данных используются сети, деревья, бинарные отношения, n-арные отношения, объекты.

Операции делятся на два типа: первые позволяют получить некоторые утверждения о предметной области (поиск и обработка информации), вторые – позволяют поддерживать адекватность модели реальному миру (включение, удаление, модификация).

Ограничения целостности делятся на ограничения на структуру данных, на операции и ограничения предметной области.

Со структурой данных и ограничениями целостности тесно связан язык определения данных (ЯОД), операции над данными реализованы в языке манипулирования данными (ЯМД).

Реляционная модель данных – это абстрактная теория данных, основанная на некоторых положениях математики (в основном теории множеств и логике предикатов). Принципы реляционной модели были изначально заложены в 1969–70-х годах доктором Е. Ф. Коддом (E. F. Codd), в то время исследователем, работавшим в корпорации IBM. В конце 1968 года Кодд, математик по образованию, впервые осознал, что математические дисциплины можно использовать, чтобы привнести в область управления базами данных строгие принципы и точность; именно таких качеств и не доставало этой области в то время.

§ 2 Структура

Доменом называется множество, однородных элементов. Домены являются произвольными непустыми конечными или счетными множествами. Например, множество названий дней недели, множество целых чисел, состоящих не более чем из 10 цифр, множество символьных строк длиной 20 символов и т.п.

Атрибутом называется именованный домен, представляющий свойство объекта или связи. Разные атрибуты могут иметь одинаковые домены.

Атрибут	Домен
Фамилия	строка символов длиной не более 30 символов
Название города	строка символов длиной не более 25 символов
Номер квартиры	натуральное число
Кол-во ед. товара	натуральное число
Стипендия	вещественное число с двумя знаками после запятой
Форма детали	круг, квадрат, шестиугольник

Схемой отношения R называется конечное множество имен атрибутов $\mathbf{R(A_1,A_2,\ldots, A_n)}$. Каждому атрибуту с именем $\mathbf{A_i}$ соответствует домен $\mathbf{D_i}$, i=1,…,n. Введем обозначение: $\mathbf{dom(A_i) = D_i}$. Количество атрибутов в схеме – **n** – называется **степенью** отношения

Пусть $\mathbf{D} = D_1 \times D_2 \times \ldots \times D_n$ декартово произведение всех доменов схемы отношения **D**=dom(R).

Отношение r со схемой **R** – это конечное подмножество в **D**: $\mathbf{r(R)=\{t_1,t_2,\ldots,t_p\}}$, где **p** – количество элементов, входящих в отношение **r(R) – кардинальное число**. Каждый элемент отношения – **t** – называется **кортежем** – совокупность связанных элементов данных из соответствующих доменов отношения. Кортеж состоит из **элементов данных - $\mathbf{t(A_i)}$** - значений кортежа **t** на атрибуте $\mathbf{A_i}$. Каждый элемент данных должен принадлежать домену соответствующего ему атрибута, т.е. удовлетворять следующему ограничению: $\mathbf{t(A_i) \in D_i}$, i=1,…,n.

Определим схему отношения **Студент** с атрибутами *Фамилия*, *Год_рождения*, *Год_поступления*, *Номер_билета*, *Курс_обучения*, *Стипендия*:

Студент(*Фамилия, Год_рождения, Год_поступления, Номер_билета, Курс_обучения, Стипендия*).

В качестве доменов для атрибутов отношения **Студент** могут выступать следующие множества:

dom(*Фамилия*) = строка символов длиной 20;
dom(*Год_рождения*) = dom(*Год_поступления*) = множество четырехзначных натуральных чисел;
dom(*Курс_обучения*) = {1, 2, 3, 4, 5, 6};
dom(*Стипендия*) = вещественное число с двумя знаками после запятой.

Кортежами, удовлетворяющими схеме, могут быть:

t_1 = (Петров, 1978, 1994, 123456, 5, 100.50)
t_2 = (Иванов, 1979, 1995, 122453, 4, 0.00)

Тогда элементами данных будут:

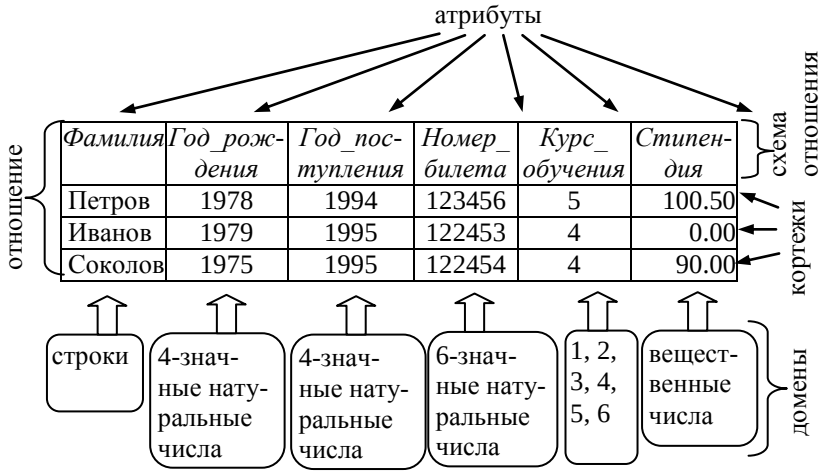
t_1(*Год_рождения*) = 1978,
t_2(*Стипендия*) = 0.00.

Отношения удобно представлять в виде таблицы. В верхней строке таблицы записываются названия атрибутов. Каждая следующая строка представляет собой один кортеж (или, по-другому, запись). На пересечении столбца и строки таблицы находится конкретное значение – элемент данных. Все данные, находящиеся в одном столбце должны быть одного типа, соответствующего домену атрибута, заголовок которого находится в первой строке таблицы.

Студент

Фамилия	*Год_рождения*	*Год_поступления*	*Номер_билета*	*Курс_обучения*	*Стипендия*
Петров	1978	1994	123456	5	100.50
Иванов	1979	1995	122453	4	0.00
Соколов	1975	1995	122454	4	90.00

Таблица как раз и представляет собой реляционное отношение. Столбцы таблицы соответствуют атрибутам отношения, шапка таблицы (верхняя строка) – схеме отношения. Каждая следующая строка представляет собой один кортеж (или, по-другому, запись). На пересечении столбца и строки таблицы находится конкретное значение – данное. Все данные, находящиеся в одном столбце, должны быть одного типа, соответствующего домену атрибута, заголовок которого находится в первой строке таблицы.



Ключом отношения **r(R)** называется подмножество $\mathbf{K} \subseteq \mathbf{R}$, такое, что для любых различных кортежей $\mathbf{t_1}$ и $\mathbf{t_2}$ из **r** выполняется $\mathbf{t_1(K) \neq t_2(K)}$ (свойство уникальности), и ни одно собственное подмножество $\mathbf{K' \subset K}$ не обладает этим свойством (свойство неизбыточности).

Это означает, что ключ – это минимальный (неизбыточный) набор атрибутов, уникальным образом идентифицирующих кортеж, т.е. набор элементов данных, соответствующих атрибутам, входящим в ключ, не повторяется, не существует двух кортежей, для которых эти наборы совпадают. Среди всех возможных кортежей отношения (среди всех возможных, а не только среди тех, из которых состоит отношение в данный момент) нет двух различных кортежей с одинаковыми значениями атрибутов, входящих в ключ.

Причина такой важности ключей заключается в том, что они обеспечивают основной механизм адресации на уровне кортежей. Единственный гарантируемый способ точно указать на какой-нибудь кортеж – это указать значение ключа.

Например, атрибут {*Год_поступления*} не может быть ключом, так как значения этого атрибута повторяются (второй и третий кортеж). Атрибуты {*Год_поступления, Курс_обучения*} также не являются

ключом, так как набор данных {1995, 4} повторяется в двух кортежах. Ключом для схемы **Студент** может быть один атрибут {*Номер_билета*}, так как каждый студент имеет студенческий билет с уникальным номером.

В определении ключа рассматриваются не только кортежи, входящие в данное отношение, но все возможные кортежи для данной схемы и данных доменов.

Ключ, состоящий из одного атрибута, называется **простым**. Ключ, состоящий более чем из одного атрибута, называется **составным**.

Схема отношения может иметь несколько ключей. Они называются **потенциальными** ключами. Один из них выбирается в качестве основного и называется **первичным**, остальные ключи называются **вторичными** (возможными, альтернативными). В схеме отношения имена атрибутов, входящих в первичный ключ, подчеркиваются. Атрибуты, входящие в первичный ключ, называются **первичными атрибутами** (**ключевыми**), остальные атрибуты называются **непервичными** (**неключевыми**).

Студент(*Фамилия, Год_рождения, Год_поступления, Номер_билета, Курс_обучения, Стипендия*).

Введем понятие внешнего ключа.

Пусть **r(R)** – отношение со схемой **R**, **FK** – набор атрибутов из схемы **R** (**FK** $\subset$ **R**). Пусть **s(S)** – отношение со схемой **S**, **CK** – ключ отношения **s** (**CK** $\subset$ **S**). Набор атрибутов **FK** называется **внешним ключом** отношения **r(R)**, ссылающимся на отношение **s**, если каждое значение атрибутов **FK** в отношении **r** всегда совпадает со значениями атрибутов **CK** некоторого кортежа в отношении **s**.

Рассмотрим отношение **Соревнования**:

Соревнования(*Название_вида*, *Номер_участника*, *Номер_билета*).

Это отношение будет содержать информацию о том, в каких видах и под какими номерами студенты участвуют в соревнованиях. В каждом виде соревнований может участвовать несколько студентов, каждый из которых имеет уникальный номер участника в пределах данного вида соревнований, любой студент может участвовать в нескольких видах соревнований, поэтому ключом отношения будет совокупность атрибутов {*Название_вида, Номер_участника*}.

Соревнования

Название_вида	*Номер_участника*	*Номер_билета*
Бег на 100 м	1	122453
Бег на 100 м	2	123456
Прыжки в высоту	1	122454
Прыжки в высоту	2	122453

Значениями атрибута *Номер_билета* будут номера зачетных книжек студентов, информация о которых содержится в отношении **Студент**. Напомним, что ключом отношения **Студент** является атрибут *Номер_билета*. Тогда в отношении **Соревнования** атрибут *Номер_билета* будет внешним ключом, ссылающимся на ключ отношения **Студент**. Это означает, что значения атрибута *Номер_билета* отношения **Соревнования** выбираются из значений атрибута *Номер_билета* отношения **Студент**.

Значения внешнего ключа могут повторяться. Для отношения **Соревнования** это означает, что один и тот же студент может участвовать в разных видах соревнований.

Атрибут, являющийся внешним ключом, может иметь произвольное название, необязательно совпадающее с названием первичного ключа отношения, на которое ссылается внешний ключ.

Другой пример.

Протокол_Соревнования(*Номер_билета, Номер_попытки, Результат*).

Это отношение будет содержать информацию о результатах одного вида соревнований, в котором допускается несколько попыток, например, прыжки в длину. Ключом отношения будет совокупность атрибутов {*Номер_билета, Номер_попытки*}.

Протокол_Соревнования

Номер_билета	*Номер_попытки*	*Результат*
122453	1	2,25
123453	2	2,17
122453	3	2,21
122454	1	2,22

Значениями атрибута *Номер_билета* будут номера зачетных книжек студентов, информация о которых содержится в отношении **Студент**. Тогда в отношении **Протокол_Соревнования** атрибут *Номер_билета* будет внешним ключом, ссылающимся на ключ отношения **Студент**.

В этом примере атрибут *Номер_билета* является и внешним ключом, и входит в состав первичного ключа.

Другой пример. Два отношения:

Предмет(Шифр_*предмета, Название_предмета*)

Экзамен(*Номер_билета, Шифр_предмета, Оценка*)

Отношение **Экзамен** будет содержать информацию о результатах сдачи студентами экзаменов по всем предметам. Ключом отношения будет совокупность атрибутов {*Номер_билета, Шифр_предмета*}.

Предмет

Шифр_предмета	*Название_предмета*
101	Алгебра
103	Геометрия
203	Информатика

Экзамен

Номер_билета	*Шифр_предмета*	*Оценка*
122453	101	5
123453	103	4
122453	203	5
122454	101	5

Значениями атрибута *Номер_билета* будут номера зачетных книжек студентов, информация о которых содержится в отношении **Студент**. Тогда в отношении **Экзамен** атрибут *Номер_билета* будет внешним ключом, ссылающимся на ключ отношения **Студент**. Значениями атрибута *Шифр_предмета* будут номера предметов, информация о которых содержится в отношении **Предмет**. Тогда в отношении **Экзамен** атрибут *Шифр_предмета* будет внешним ключом, ссылающимся на ключ отношения **Предмет**.

В этом примере в отношении **Экзамен** первичный ключ является составным, и каждый атрибут ключа является внешним ключом.

§ 3 Операции

Операции реляционной модели данных можно условно разделить на две группы: операции обновления отношений и операции реляционной алгебры.

Операции обновления предназначены для добавления, удаления и изменения кортежей отношения.

Операции реляционной алгебры используют в качестве операндов реляционные отношения, и результатом операции также является реляционное отношение. Именно поэтому они образуют алгебру. Основу этой группы операций составляют 8 операторов, предложенных Коддом, это так называемая «начальная» алгебра.

Группу реляционных операторов можно разбить еще на две группы:

1. Традиционные операции над множествами: объединение, пересечение, вычитание и декартово произведение.
2. Специальные реляционные операции: селекция (выборка), проекция, соединение и деление.

§ 3.1 *Операции обновления отношений*

Операции обновления – это операции, выполняемые над одним кортежем отношения.

Добавление

Цель операции - добавить один кортеж с данными $\mathbf{<d_1,\ldots,d_n>}$ в отношение **r** со схемой $\mathbf{(A_1,A_2,\ldots,A_n)}$.

$$ADD(r;\ A_1 = d_1,\ldots,\ A_n=d_n)$$

Когда фиксирован порядок имен атрибутов, допустима более короткая запись:

$$ADD(r;\ d_1,\ldots,\ d_n)$$

Операция может быть не выполнена по следующим причинам:

1. Добавляемый кортеж не соответствует схеме отношения r.
2. Некоторые значения кортежа не принадлежат соответствующим доменам.
3. Добавляемый кортеж совпадает по ключу с кортежем, уже находящимся в отношении.

Примеры для отношения **Студент**:

a). ADD(Студент; Дубов, 1980, 1996, 134742, 2, 100.00)

b). ADD(Студент; Фамилия = Марков, Имя = Сергей, Отчество = Петрович, Курс = 2 курс) – не допускается по причине 1.

c). ADD(Студент; Колобков, 1976, 1996, 122454, 8, 90.00) - не допускается по причинам 2 и 3. Ключом отношения **Студент** является атрибут {*Номер_билета*}. Отношение уже содержит кортеж с номером билета 122454 у студента Соколова (причина 3), и значением атрибута *Курс_обучения* не может быть число 8 (причина 2).

Удаление

Цель операции – удаление одного кортежа с данными <d_1,…d_n> из отношения r.

DEL(r; A_1=d_1,…, A_n=d_n)

В действительности, нет необходимости перечислять значения всех атрибутов удаляемого кортежа. Чтобы однозначно определить кортеж, который надо удалить, достаточно определить значение первичного ключа:

DEL(r; <ключ> = <d>),

<ключ> - содержит только те атрибуты, которые входят в ключ,

<d> - содержит только те значения удаляемого кортежа, которые соответствуют ключевым атрибутам.

Операция может быть не выполнена по следующим причинам:

1. Удаляемый кортеж не соответствует схеме отношения r.
2. Удаляемого кортежа нет в отношении r.

Примеры:

a). DEL(Студент; 122454)

b). DEL(Студент; 555556) – не будет выполнена, так как такого кортежа в отношении **Студент** нет.

Изменение

Цель операции – заменить значения атрибутов C_1,…,C_p кортежа <d_1,…,d_n> отношения r значениями e_1,…,e_p.

CH(r; A_1=d_1,…, A_n=d_n; C_1=e_1,…, C_p=e_p)

Аналогично операции удаления в операции изменения для определения кортежа, значения атрибутов которого требуется изменить, не обязательно перечислять все атрибуты и их значения, достаточно перечислить только ключевые атрибуты. Структура команды в этом случае будет такой:

CH(r; <ключ> = <d>; C_1=e_1,…, C_p=e_p)

Операцию изменения можно заменить двумя последовательно выполненными операциями удаления и добавления. Ограничения на применение операции изменения такие же, как и у операций добавления и удаления.

Примеры:

a). CH(Студент; 122453; *Стипендия* = 70.00)

b). CH(Студент; 122453; *Средний_балл* = 4.5) – не будет выполнена, так как атрибут *Средний_балл* не входит в схему **Студент**.

§ 3.2 *Операции над множествами*

Два отношения с одной и той же схемой могут быть рассмотрены как два подмножества одного и того же универсума - множества всех возможных кортежей с этой схемой. К таким двум отношениям могут быть применены булевы операции.

Пересечение

В результате пересечения двух отношений с одной и той же схемой получается новое отношение с той же схемой, содержащее только те кортежи, которые были и в первом отношении, и во втором отношении. Если таковых не оказалось, то результатом будет пустое отношение, т.е. отношение, не содержащее ни одного кортежа.

Объединение

В результате объединения двух отношений с одной и той же схемой получается новое отношение с той же схемой, содержащее все кортежи из первого отношения и все кортежи из второго отношения. Если оба отношения содержали абсолютно совпадающие кортежи, то такие кортежи входят в новое отношение только один раз.

Разность

При выполнении вычитания из одного отношения другого отношения с той же схемой получается новое отношение с той же схемой, содержащее все кортежи из первого отношения, которых нет во втором отношении. Если оба отношения не содержат одинаковых кортежей, то новое отношение будет полностью совпадать с первым отношением.

Для примера булевых операций возьмем два отношения с одинаковыми схемами **Баскетбол**(*ФИО, Факультет, Курс*) и **Самбо**(*ФИО, Факультет, Курс*). Эти отношения содержат информацию о студентах, занимающихся в секции баскетбола и карате.

Баскетбол

ФИО	*Факультет*	*Курс*
Иванов И.И.	матфак	2
Петров А.П.	физтех	2
Яшин В.В.	матфак	3

Самбо

ФИО	*Факультет*	*Курс*
Бочкин К.А.	филфак	1
Жуков Е.А.	матфак	3
Иванов И.И.	истфак	2
Макаров В.С.	матфак	1
Петров А.П.	физтех	2

В результате операции объединения отношений **Баскетбол** и **Самбо** (**Баскетбол** ∪ **Самбо**) получим информацию о всех студентах, занимающихся спортом, это отношение назовем **Спорт1**. В результате пересечения отношений (**Баскетбол** ∩ **Самбо**) получим информацию о студентах, занимающихся в обеих секциях – отношение **Спорт2**. В результате операции вычитания из отношения **Баскетбол** отношения **Самбо** (**Баскетбол – Самбо**) получим информацию о студентах играющих в баскетбол, но при этом, не занимающихся борьбой – отношение **Спорт3**, а при вычитании из отношения **Самбо** отношения **Баскетбол** (**Самбо – Баскетбол**) получим информацию о студентах, занимающихся борьбой, но при этом, не играющих в баскетбол – отношение **Спорт4.**

Отметим, что и в отношении **Баскетбол**, и в отношении **Самбо** есть кортеж со значением атрибута *ФИО*, равным "Иванов И.И.", и значением атрибута *Курс*, равным "2". Но эти два кортежа различаются значением атрибута *Факультет*, и, следовательно, считаются различными. Поэтому в объединении отношений есть два кортежа со значением атрибута *ФИО*, равным "Иванов И.И.", а в пересечение отношений эти кортежи не попали.

Спорт1

ФИО	*Факультет*	*Курс*
Бочкин К.А.	филфак	1
Жуков Е.А.	матфак	3
Иванов И.И.	истфак	2
Иванов И.И.	матфак	2
Макаров В.С.	матфак	1
Петров А.П.	физтех	2
Яшин В.В.	матфак	3

Спорт2

ФИО	*Факультет*	*Курс*
Петров А.П.	физтех	2

Спорт3

ФИО	*Факультет*	*Курс*
Иванов И.И.	матфак	2
Яшин В.В.	матфак	3

Спорт4

ФИО	*Факультет*	*Курс*
Бочкин К.А.	филфак	1
Жуков Е.А.	матфак	3
Иванов И.И.	истфак	2
Макаров В.С.	матфак	1

Дополнение

Пусть **dom(R)** – множество всех кортежей над атрибутами **R** и их доменами. Дополнение отношения **r**: **r'(R) = dom(R) – r**. Однако, если какой-либо атрибут в **R** имеет бесконечный домен, то дополнение **r'** также будет бесконечным и не будет отношением по определению. Например, дополнение к отношению студент будет бесконечным, так как бесконечны все домены, кроме домена атрибута *Курс_обучения*. Модифицированная версия дополнения называется активное дополнение.

Активное дополнение

Активное дополнение отношения **r**: **r'' = adom(R, r) – r**, где **adom(R, r)** – множество всех кортежей над атрибутами из **R** и их активными доменами относительно **r**. **Активным доменом** атрибута A_i относительно отношения r называется множество:

$$\mathbf{adom}(A_i, r) = \{ d \in D_i \mid \exists t \in r : t(A_i) = d \}.$$

Рассмотрим активное дополнение для отношения **Баскетбол**. Активные домены для каждого атрибута состоят из множеств:

adom(*ФИО,* **Баскетбол**) = {Иванов И.И., Петров А.П., Яшин В.В.};

adom(*Факультет,* **Баскетбол**) = {матфак, физтех};

adom(*Курс,* **Баскетбол**) = {2, 3};

Активный домен относительно отношения **Баскетбол** будет состоять из 3·2·2 = 12 элементов, три из которых входят в отношение **Баскетбол**, значит остальные попадут в его активное дополнение **Баскетбол''**.

Баскетбол''

ФИО	*Факультет*	*Курс*
Иванов И.И.	матфак	3
Иванов И.И.	физтех	2
Иванов И.И.	физтех	3
Петров А.П.	физтех	3
Петров А.П.	матфак	2
Петров А.П.	матфак	3
Яшин В.В.	матфак	2
Яшин В.В.	физтех	2
Яшин В.В.	физтех	3

Активные дополнения могут использованы, например, для такого запроса: каждый студент должен сдать зачет по всем предметам, информация о том, кто уже сдал зачет заносится в отношение **Зачет**(*ФИО, Предмет*), требуется узнать, кто из студентов еще не сдал зачет и, по каким предметам, об этом как раз и сообщит активное дополнение. Но, если какой-то студент еще не сдал ни одного зачета или по какому-то предмету ни один студент еще не сдал зачет, то информации об этом студенте и этом предмете в активном дополнении не будет.

Активное дополнение может быть использовано для сжатия памяти, когда активное дополнение имеет меньше кортежей, чем само отношение.

§ 3.3 *Операции реляционной алгебры*

Селекция

Пусть **r** отношение со схемой **R**, **A** – атрибут, принадлежащий схеме **R**, **a** – значение из домена атрибута **A** (**a ∈ dom(A)**). Результатом выполнения операции селекции $\sigma_{A=a}(r)$ является новое отношение с той же схемой, которое представляет собой подмножество кортежей отношения **r**, каждый из которых имеет значение **a** на атрибуте **A**.

$$\sigma_{A=a}(r) = \{ t \in r \mid t(A) = a \}$$

Операция селекции – это операция выбора только тех кортежей, которые имеют заданное значение на заданном атрибуте. Пример: найдем тех студентов, которые учатся на 4 курсе. Для этого необходимо выполнить операцию селекции для отношения **Студент**, при этом условием отбора будет: *Курс_обучения* = 4. В результате получим новое отношение **Студент3** с двумя кортежами:

$$\sigma_{\text{Курс\_обучения} = 4}(\text{Студент}) = \text{Студент3}$$

Студент3

Фамилия	*Год_рождения*	*Год_поступления*	*Номер_билета*	*Курс_обучения*	*Стипендия*
Иванов	1979	1995	122453	4	0.00
Соколов	1975	1995	122454	4	90.00

(В условии отбора операции селекции кроме сравнения на равенство можно использовать и другие операции сравнения. Например, если требуется найти всех студентов, родившихся не ранее 1979 года, то в качестве условия можно записать *Год_рождения*>1978, тогда будут выбраны кортежи, соответствующие студентам с годом рождения 1979, 1980, 1981 и т.д.

Также можно составить и более сложное условие с использованием логических операций and, or, not для одного или нескольких атрибутов. Например, (*Год_рождения* > 1978)and(*Год_рождения* <= 1980)and(*Стипендия* = 100.50).)

Свойства селекции:

1. Пусть, **r(R)**, **A ∈R**, **a ∈ dom(A)**, **B ∈ R**, **b ∈ dom(B)**, тогда
   $\sigma_{A=a}(\sigma_{B=b}(r)) = \sigma_{B=b}(\sigma_{A=a}(r)) = \sigma_{(A=a)\text{ and }(B=b)}(r)$ (коммутативность).
2. Пусть, **r(R)**, **s(R)**, **A ∈R**, **a ∈ dom(A)**, тогда
   $\sigma_{A=a}(r \ \gamma \ s) = (\sigma_{A=a}(r)) \ \gamma \ (\sigma_{A=a}(s))$, где **γ** – булева операция: ∩, −, ∪ (дистрибутивность относительно бинарных булевых операций).

Проекция

Пусть **r** отношение со схемой **R**, **A** – атрибут, принадлежащий схеме **R**. Результатом выполнения операции проекции отношения **r** на атрибут **A** – $\pi_A(r)$ является новое отношение, схема которого будет состоять только из одного атрибута **A**:

$$\pi_A(r) = \{ t(A) \mid t \in r \} \text{ или } \pi_A(r) = \{ d \in dom(A) \mid \exists t \in r: t(A) = d \}.$$

Кортежами этого отношения будут отличные друг от друга значения атрибута **A** из исходного отношения **r**. Если представить себе исходное отношение **r** в виде таблицы, то проекция получается вычеркиванием всех столбцов кроме столбца, соответствующего атрибуту **A**, и исключением из оставшегося столбца повторяющихся строк.

Например, требуется узнать все года рождения студентов. Для этого достаточно выполнить операцию проекции отношения **Студент** на атрибут *Год_рождения*: $\pi_{\text{Год\_рождения}}$**(Студент)**. Результатом операции будет новое отношение **Студент1** с одним атрибутом и тремя кортежами.

Студент1

Год_рождения
1978
1979
1975

Студент2

Год_поступления	*Курс_обучения*
1994	5
1995	4

Операцию проекции можно выполнять и для нескольких атрибутов. Например, требуется информация о том, в каких годах поступали студенты и на каких курсах они сейчас учатся. Эту информацию можно получить, выполнив операцию проекции отношения **Студент** на атрибуты *Год_поступления* и *Курс_обучения*: $\pi_{\text{Год\_поступления, Курс\_обучения}}$**(Студент)**. Результатом операции будет отношение **Студент2** с двумя атрибутами и двумя кортежами. (Последний кортеж отношения **Студент2** полностью совпадал с предпоследним кортежем, и, поэтому, был удален).

Свойства проекции:

1. Пусть, $r(R)$, $Y \subseteq X \subseteq R$, тогда $\pi_Y(\pi_X(r)) = \pi_Y(r)$
2. Пусть, $r(R)$, $X \subseteq R$, $A \in X$, $a \in dom(A)$, тогда $\pi_X(\sigma_{A=a}(r)) = \sigma_{A=a}(\pi_X(r))$

Соединение (естественное соединение)

Эта операция предназначена для комбинирования двух отношений с разными схемами. Пусть **r** отношение со схемой **R**, **s** отношение со схемой **S** и **T** – множество атрибутов, входящих и в схему **R**, и в схему **S** ($T = R \cap S$). Результатом выполнения операции соединения $r(R) \bowtie s(S)$ является новое отношение **q**, схема которого состоит из всех атрибутов схемы **R** и всех атрибутов схемы **S**, при этом одинаковые атрибуты не дублируются. Каждый кортеж отношения **q** является комбинацией кортежа из отношения **r** и кортежа из отношения **s** с равными значениями атрибутов, входящих во множество **T**:

$$r(R) \bowtie s(S) = q(R \cup S) = \{ t \in R \cup S \mid \exists t_r \in r, \ t_s \in s, \ t_r = t(R), \ t_s = t(S), \ t_r(R \cap S) = t_s(R \cap S) \}.$$

Для примера рассмотрим еще одно отношение **Материальная помощь** с атрибутами *Курс_обучения* и *Сумма*.

Материальная помощь

Курс_обучения	*Сумма*
1	100.00
2	120.00
4	140.00
5	130.00

Требуется для каждого студента определить сумму материальной помощи. Для этого выполним соединение двух отношений **Студент** и **Материальная помощь**. Общим атрибутом, входящим и в схему отношения **Студент**, и в схему отношения **Материальная помощь**, является атрибут *Курс_обучения*. По этому атрибуту и будет выполнено соединение. Новое отношение, которое получится в результате операции соединения, будет иметь в своей схеме семь атрибутов и три кортежа.

Студент4

Фамилия	*Год_рождения*	*Год_поступления*	*Номер_билета*	*Курс_обучения*	*Стипендия*	*Сумма*
Петров	1978	1994	123456	5	100.00	130.00
Иванов	1979	1995	122453	4	0.00	140.00
Соколов	1975	1995	122454	4	90.00	140.00

Если бы отношение **Материальная помощь** состояло бы из следующих кортежей:

Материальная помощь

Курс_обучения	*Сумма*
4	100.00
4	120.00

т.е. помощь выдается только четвертому курсу и, начисления производятся дважды с разными суммами, то в результате операции соединения получилось бы отношение **Студент5** с четырьмя кортежами. Первый и второй кортежи отношения **Студент** образовали по два кортежа отношения **Студент5**, а третий кортеж ни одного, так как значения атрибута *Курс_обучения* этого кортежа не равны ни одному значению атрибута *Курс_обучения* отношения **Материальная помощь**.

Студент5

Фамилия	*Год_рождения*	*Год_поступления*	*Номер_билета*	*Курс_обучения*	*Стипендия*	*Сумма*
Иванов	1979	1995	122453	4	0.00	100.00
Иванов	1979	1995	122453	4	0.00	120.00
Соколов	1975	1995	122454	4	90.00	100.00
Соколов	1975	1995	122454	4	90.00	120.00

Если бы отношение **Материальная помощь** состояло бы из следующих кортежей:

Материальная помощь

Курс_обучения	*Стипендия*	*Сумма*
4	0.00	100.00
5	0.00	120.00

т.е. помощь выдается только четвертому и пятому курсу и, только тем студентам, которые не получают стипендию, то в результате операции соединения получилось бы отношение **Студент6** с одним кортежем. В этом случае требуется совпадение значений одновременно по двум атрибутам *Курс_обучения* и *Стипендия*, что возможно только для второго кортежа отношения **Студент** и первого кортежа отношения **Материальная помощь**.

Студент6

Фамилия	*Год_рождения*	*Год_поступления*	*Номер_билета*	*Курс_обучения*	*Стипендия*	*Сумма*
Иванов	1979	1995	122453	4	0.00	100.00

Свойства соединения:

1. Пусть, **r(R)**, **q(R)**, **s(S)**, тогда $(r \cup q) \bowtie s = (r \bowtie s) \cup (q \bowtie s)$.
2. Пусть, **r(R)**, **s(S)**, $A \in R$, $a \in dom(A)$, тогда $\sigma_{A=a}(r \bowtie s) = \sigma_{A=a}(r) \bowtie s$.
3. Пусть, **r(R)**, $A \in R$, $a \in dom(A)$, $s(A) = \{<a>\}$, тогда $r \bowtie s = \sigma_{A=a}(r)$.
4. Если $R \cap S = \varnothing$, то $r \bowtie s = r \times s$ (декартово произведение кортежей).

Эквисоединение

Эта операция предназначена для соединения двух отношений с абсолютно разными схемами. Пусть **r** – отношение со схемой **R**, **s** – отношение со схемой **S**, $R \cap S = \varnothing$, т. е. схемы не содержат одинаковых атрибутов. Пусть **A** – атрибут схемы **R**, а **B** – атрибут схемы **S** ($A \in R$, $B \in S$). Эти атрибуты определены на одинаковых доменах **dom(A)=dom(B)**. Результатом выполнения операции эквисоединения **r(R) [A=B] s(S)** является новое отношение, схема которого состоит из всех атрибутов схемы **R** и всех атрибутов схемы **S**. Каждый кортеж этого отношения является комбинацией кортежа из отношения **r** и кортежа из отношения **s** с равными значениями атрибутов **A** и **B**:

$$r(R)\,[A=B]\,s(S) = \{\, t \in R \cup S \mid \exists\, t_r \in r,\ \exists\, t_s \in s :\ t_r = t(R),\ t_s = t(S),\ t_r(A) = t_s(B) \,\}.$$

Операция эквисоединения используется вместо операции соединения, которую также называют естественным соединением, когда атрибуты, по которым необходимо выполнить соединение, имеют разные имена. Например, пусть, отношение **Материальная помощь** содержит атрибуты *Курс* (вместо *Курс_обучения*) и *Сумма*. В этом случае операцию соединения выполнить нельзя, так как пересечение схем отношений **Студент** и **Материальная помощь** пусто. Но атрибуты *Курс* и *Курс_обучения* по смыслу содержат одинаковую информацию и формально их домены совпадают, поэтому вместо операции соединения выполняется операция эквисоединения **Студент** [*Курс_обучения* = *Курс*] **Материальная помощь.**

Материальная помощь

Курс	*Сумма*
1	100.00
2	120.00
4	140.00
5	130.00

Студент [*Курс_обучения* = *Курс*] **Материальная помощь**

Фамилия	*Год_рож-дения*	*Год_пос-тупления*	*Номер_билета*	*Курс_обучения*	*Сти-пендия*	*Курс*	*Сумма*
Петров	1978	1994	123456	5	100.00	5	130.00
Иванов	1979	1995	122453	4	0.00	4	140.00
Соколов	1975	1995	122454	4	90.00	4	140.00

Отношения могут быть соединены по нескольким атрибутам с одинаковыми доменами.

Пусть, **r(R)** и **s(S)** два отношения с различными именами атрибутов, т.е. $\mathbf{R \cap S = \varnothing}$, и $\mathbf{A_i \in R}$, $\mathbf{B_i \in S}$ и $\mathbf{dom(A_i)=dom(B_i)}$ для i=1,…,m. Эквисоединением **r** и **s** по атрибутам $\mathbf{A_1, A_2, \ldots, A_m}$ и $\mathbf{B_1, B_2, \ldots, B_m}$ называется отношение:

$$\mathbf{r\,[A_1=B_1, \ldots, A_m=B_m]\,s = \{\, t \mid \exists\, t_r \in r, \exists\, t_s \in s,\ t(R)=t_r,\ t(S)=t_s,\ t(A_i)=t(B_i),\ \ i=1,\ldots,m \,\}}$$

Эквисоединение содержит все столбцы из схемы **R** и все столбцы из схемы **S** в отличие от естественного соединения.

Если **m**=0, то эквисоединение превращается в декартово произведение.

Деление

Пусть, **r** – отношение со схемой **R**, **s** – отношение со схемой **S** и $\mathbf{S \subseteq R}$, тогда **r**, разделенное на **s**, – это отношение, содержащее такие значения атрибутов **R–S** кортежей из отношения **r**, для которых соответствующие значения атрибутов **S** включают все значения из отношения **s**:

$$\mathbf{r \div s = \{\, t \mid \text{для } \forall\, t_s \in s \ \ \exists\, t_r \in r \ :\ t_r(R - S) = t \text{ и } t_r(S) = t_s \ \}.}$$

В качестве примера рассмотрим отношение **Зачет**(*ФИО, Название*), содержащее информацию о студентах, сдавших зачеты по различным дисциплинам, и отношение **Дисциплины**(*Название*), содержащее некоторый список названий дисциплин. Тогда операция деления отношения **Зачет** на отношение **Дисциплины** будет содержать фамилии только тех студентов, которые получили зачеты по всем дисциплинам, перечисленным в отношении **Дисциплины**.

Зачет

ФИО	*Название*
Иванов И.И.	Алгебра
Петров В.В.	Алгебра
Иванов И.И.	История
Дубов С.С.	Физика
Петров В.В.	Физика
Иванов И.И.	Физика
Дубов С.С.	Топология

Предметы

Название
Алгебра
Физика

Зачет ÷ Предметы

ФИО
Иванов И.И.
Петров В.В.

Только два студента сдали оба зачета и по Алгебре, и по Физике.

Свойства деления:

1. Пусть, **r(R)**, **s(S)**, $\mathbf{S \subseteq R}$, тогда $\mathbf{r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}((\pi_{(R-S)}(r) \bowtie s) - r)}$.

Сравнение на доменах

Часто домены упорядочены, и в этих случаях сравнения <, ≤, >, ≥ также имеют смысл, как = и ≠. Для общего рассмотрения таких сравнений вводится множество $\mathbf{\Theta}$ – символов бинарных отношений над элементами одного домена (**a θ b**) или парой доменов (**A θ B**) . Два атрибута, **A** и **B**, **θ-сравнимы**, если знаку $\mathbf{\theta \in \Theta}$ сопоставлено бинарное отношение в **dom(A)×dom(B)**.

Расширение операции селекции (выбора)

Пусть отношение **r(R)**, $\mathbf{A \in R}$, $\mathbf{a \in dom(A)}$, тогда

$$\mathbf{\sigma_{A\,\theta\,a}(r) = \{\, t \in r \mid t(A)\ \theta\ a \,\}.}$$

Пусть отношение **r(R)**, $\mathbf{A \in R}$, $\mathbf{B \in R}$, тогда

$$\mathbf{\sigma_{A\,\theta\,B}(r) = \{\, t \in r \mid t(A)\ \theta\ t(B) \,\}.}$$

Пусть, операция сравнения "<<" означает "предшествует хотя бы на два часа", тогда можно выполнить следующую операцию:

$$\sigma_{\text{отправление} \,<<\, \text{прибытие}}\ (\mathbf{Рейс\_А\_Б})$$

θ – соединение

Можно выполнять операцию соединения не только для равенства значений атрибутов, но и для любой другой операции сравнения на доменах **θ∈Θ**.

Пусть, **r(R)** и **s(S)** два отношения, для которых **R∩S=∅** и **A∈R**, **B∈S**, **A** и **B** **θ**–сравнимы, тогда **θ-соединение** – это отношение:

$$r\ [\ A\ \theta\ B\ ]\ s = \{\ t \mid \exists\ t_r \in r,\ \exists\ t_s \in s\ :\ t_r(A)\ \theta\ t_s(B),\ \ t(R) = t_r,\ \ t(S) = t_s\ \}$$

Рейс_А_Б

НомерА	*Отправление*	*Прибытие*
60	9.40	11.45
91	12.50	14.47
112	16.05	18.15
306	20.30	22.25
420	21.15	23.11

Рейс_Б_С

НомерБ	*Время_отпр*	*Время_приб*
11	8.30	9.52
60	12.25	13.43
156	16.20	17.40
158	19.10	20.35

Требуется узнать какими парами рейсов можно попасть из пункта А в пункт С. Для этого необходимо выполнить операцию соединения двух отношений – **Рейс_А_Б** и **Рейс_Б_С** по атрибутам *Прибытие* и *Время_отпр* с операцией сравнения "<" на доменах этих атрибутов.

$$\pi_{НомерА,\ НомерБ}\ (\ \textbf{Рейс\_А\_Б}\ [\ Прибытие < Время\_отпр\ ]\ \textbf{Рейс\_Б\_С}\ ) = \textbf{Транзит\_А\_С}$$

Транзит_А_С

НомерА	*НомерБ*
60	60
60	156
60	158
91	156
91	158
112	158

Эквисоединение – это частный случай **θ**–соединения – равенство.

Можно выполнять **θ**–соединение по нескольким парам атрибутов с различными операциями сравнения. Например, $\mathbf{r\ [\ A_1 < B_1,\ \ A_2 = B_2,\ \ A_3 \geq B_3\ ]\ s}$.

§ 4 Ограничения целостности

Отношения реляционной модели данных должны удовлетворять следующим правилам:

1. Запрещается дублирование кортежей в отношении. Это означает, что каждое отношение имеет, по крайней мере, один ключ, состоящий из всех атрибутов.
2. Порядок кортежей в отношении не определен. Нет ни первого, ни второго, ни последнего кортежа. Все кортежи равнозначны. Однако при реализации в конкретной СУБД может быть задан физический порядок следования кортежей. А также с помощью индексов может быть задан порядок обработки кортежей.
3. Порядок атрибутов в отношении не определен. Каждый атрибут в отношении имеет уникальное имя, по которому его можно идентифицировать. Однако при реализации в конкретной СУБД атрибуты могут быть упорядочены.
4. Два правила целостности:
   a). Целостность по сущностям. Не допускаются неопределенные значения ключевых атрибутов. Это означает, что каждый кортеж обязательно должен содержать значения в атрибутах, входящих в ключ. Соблюдение этого правила должно отслеживаться при выполнении операций обновления отношения (добавление, удаление, изменение).
   b). Целостность по ссылкам. Значения внешних ключей должны либо соответствовать значениям первичных ключей, либо быть неопределенными.

Последнее правило требует пояснения. Вернемся к отношению **Соревнования** из раздела 3.1. Согласно правилу целостности по ссылкам, значения атрибута *Номер_билета* отношения **Соревнования** должны либо совпадать с каким-нибудь значением атрибута *Номер_билета* отношения **Студент**, либо быть неопределенными. Это означает, что участником соревнований может быть только человек, являющийся студентом, т.е. информация о нем должна быть в отношении **Студент**. Если участник соревнований пока неизвестен, тогда это поле остается пустым. В этом поле не должно быть значения, которого нет в отношении **Студент.**

Кроме ограничений самой реляционной модели можно определить ограничения целостности, вытекающие из предметной области, которые делятся на два типа.

Например:

1. Структурные ограничения:
   a). «Зарплата должна быть положительным числом», «Оценка принимает значения 2,3,4,5» – эти ограничения оформляются в виде доменов.
   b). «Номера студенческих билетов не повторяются» – это ограничение выполняется за счет определения атрибута *Номер_билета* в качестве ключа.
   c). «В соревнованиях участвуют только студенты» – это ограничение выражено с помощью внешнего ключа.
2. Семантические ограничения:
   a). «Дата окончания не должна превышать дату начала семестра» – это ограничение для нескольких атрибутов в отношении, но для одного кортежа. Такое ограничение можно оформить в виде ограничения для таблицы.
   b). «Зарплата подчиненного не должна превышать зарплату начальника» – выполнение этого ограничения требует сравнения нескольких кортежей одного отношения, такое ограничение невозможно оформить средствами языка определения данных.
   c). «Количество кортежей в отношении **Студент** должно быть равно значению атрибута *Количество* соответствующего кортежа в отношении **Группа**» – выполнение этого ограничения охватывает данные, хранящиеся в нескольких таблицах, и может быть оформлено с помощью триггера или ограничения базы данных.
   d). «Состояние человека не может стать «разведен» сразу после состояния «холост» – это динамическое ограничение, которое никак не может быть учтено средствами языка определения данных.

Для семантических ограничений в СУБД создаются специальные механизмы поддержки – триггеры.

§ 5 Ограничения целостности Кодда

12 правил Кодда (Codd’s 12 rules) — 12 правил (на самом деле их 13), которым должна удовлетворять каждая система управления реляционными базами данных.

Предложены английским математиком Эдгаром Коддом (Edgar Codd).

В действительности правила столь строги, что все популярные т. н. «реляционные» СУБД не соответствуют многим критериям.

0. Основное правило (Foundation Rule): Реляционная СУБД должна быть способна полностью управлять базой данных, используя связи между данными.

   Чтобы быть реляционной системой управления базами данных, система должна использовать исключительно свои реляционные возможности для управления базой данных.

1. Явное представление данных (The Information Rule):

   Информация должна быть представлена в виде данных, хранящихся в ячейках. Данные, хранящиеся в ячейках, должны быть атомарны. Порядок строк в реляционной таблице не должен влиять на смысл данных.

2. Гарантированный доступ к данным (Guaranteed Access Rule):

   Доступ к данным должен быть свободен от двусмысленности. К каждому элементу данных должен быть гарантирован доступ с помощью комбинации имени таблицы, первичного ключа строки и имени столбца.

3. Полная обработка неизвестных значений (Systematic Treatment of Null Values):

   Неизвестные значения NULL, отличные от любого известного значения, должны поддерживаться для всех типов данных при выполнении любых операций. Например, для числовых данных неизвестные значения не должны рассматриваться как нули, а для символьных данных — как пустые строки.

4. Доступ к словарю данных в терминах реляционной модели (Active On-Line Catalog Based on the Relational Model):

   Словарь данных должен сохраняться в форме реляционных таблиц, и СУБД должна поддерживать доступ к нему при помощи стандартных языковых средств, тех же самых, которые используются для работы с реляционными таблицами, содержащими пользовательские данные.

5. Полнота подмножества языка (Comprehensive Data Sublanguage Rule):

Система управления реляционными базами данных должна поддерживать хотя бы один реляционный язык, который (а) имеет линейный синтаксис, (б) может использоваться как интерактивно, так и в прикладных программах, (в) поддерживает операции определения данных, определения представлений, манипулирования данными (интерактивные и программные), ограничители целостности, управления доступом и операции управления транзакциями (begin, commit и rollback).

6. Возможность модификации представлений (View Updating Rule):

   Каждое представление должно поддерживать все операции манипулирования данными, которые поддерживают реляционные таблицы: операции выборки, вставки, модификации и удаления данных.

7. Наличие высокоуровневых операций управления данными (High-Level Insert, Update, and Delete):

   Операции вставки, модификации и удаления данных должны поддерживаться не только по отношению к одной строке реляционной таблицы, но по отношению к любому множеству строк.

8. Физическая независимость данных (Physical Data Independence):

   Приложения не должны зависеть от используемых способов хранения данных на носителях, от аппаратного обеспечения компьютеров, на которых находится реляционная база данных.

9. Логическая независимость данных (Logical Data Independence):

   Представление данных в приложении не должно зависеть от структуры реляционных таблиц. Если в процессе нормализации одна реляционная таблица разделяется на две, представление должно обеспечить объединение этих данных, чтобы изменение структуры реляционных таблиц не сказывалось на работе приложений.

10. Независимость контроля целостности (Integrity Independence):

    Вся информация, необходимая для поддержания целостности, должна находиться в словаре данных. Язык для работы с данными должен выполнять проверку входных данных и автоматически поддерживать целостность данных.

11. Дистрибутивная независимость (Distribution Independence):

    База данных может быть распределённой, может находиться на нескольких компьютерах, и это не должно оказывать влияние на приложения. Перенос базы данных на другой компьютер не должен оказывать влияния на приложения.

12. Согласование языковых уровней (The Nonsubversion Rule):

    Если используется низкоуровневый язык доступа к данным, он не должен игнорировать правила безопасности и правила целостности, которые поддерживаются языком более высокого уровня.

Глава 5. Теория нормализации

§ 1 Аномалии схемы отношения

После того как построена схема отношений, являющаяся моделью данных, при ее использовании могут возникнуть некоторые нежелательные эффекты. Например, мы построили отношение **Поставка**(*Товар, Поставщик, Адрес, Цена, Склад, Объем*). Это отношение содержит информацию о наличии различных товаров, поставляемых несколькими поставщиками в некоторый магазин, и размещении их на складах магазина. Для каждого поставщика хранится его адрес, а для каждого склада – его объем. Цена товара зависит от поставщика.

Предположим, что изменился адрес одного из поставщиков. Чтобы обновить информацию в базе данных, необходимо просмотреть все кортежи отношения, и для соответствующего поставщика изменить его адрес. Во-первых, на выполнение этой операции потребуется много времени, если база данных содержит много записей, а во-вторых, может случиться так, что в каком-то кортеже адрес поставщика не будет изменен, тогда база данных будет содержать противоречивую информацию.

Другой недостаток такой схемы: если какой-то поставщик временно прекратил поставки товаров для нашего магазина, и база данных не содержит кортежей с именем этого поставщика, то его адрес теряется. Аналогично, мы не сможем хранить адрес нового поставщика, пока он не доставит хотя бы единицу какого либо товара.

Такая же ситуация и с информацией о складе – если склад опустошается, то теряются данные о его объеме.

Эти проблемы называются **аномалиями обновления.**

Можно также заметить, что такая база данных содержит избыточную информацию: адрес поставщика повторяется для каждого товара им поставляемого, а объем склада повторяется для каждого товара, размещенного на этом складе.

Решением этих проблем было бы разбиение отношения **Поставка** на несколько других отношений: **Склад**(*Номер, Объем*), **Поставщик**(*Название, Адрес*), **Товар**(*Наименование, Поставщик, Цена, Склад*). (Атрибут *Поставщик* будет внешним ключом отношения **Товар**, ссылающимся на отношение **Поставщик**, атрибут *Склад* также будет внешним ключом отношения **Товар,** ссылающимся на отношение **Склад.**)

Эти действия по разбиению одного отношения на несколько отношений называются **декомпозицией.**

Схема, для которой мы выполнили декомпозицию, была очень простой, и мы смогли выполнить операцию разбиения интуитивно. В более сложных случаях одной интуиции будет недостаточно. На помощь приходит теория нормализации реляционных отношений, дающая ответы на три основных вопроса, возникающих при выполнении декомпозиции:

По каким правилам проводить декомпозицию?

Как оценить, хорошая получилась схема или плохая?

Можно ли после обратного соединения получить исходное отношение?

§ 2 Функциональные зависимости

Объединение нескольких атрибутов в одно отношение выполняется не случайным образом. Данные, которые будут храниться в этом отношении взаимосвязаны между собой. Эта взаимосвязь определяется множеством функциональных зависимостей между атрибутами отношения.

Это означает, что значения одного атрибута зависят от значений других атрибутов. Зависимости эти вытекают из ограничений предметной области, т.е. допустимы не любые сочетания значений атрибутов.

Например, в отношении **Поставки** существуют следующие ограничения:

- Каждый поставщик имеет только один адрес,
- Каждый поставщик поставляет товар по определенной цене,
- Товары, поставленные разными поставщиками, могут быть распределены по разным складам,
- Каждый склад имеет свой объем.

Эти ограничения являются зависимостями, которые можно сформулировать следующим образом:

- Адрес функционально зависит от поставщика,
- Цена функционально зависит от товара и поставщика,
- Номер склада функционально зависит от товара и поставщика,
- Объем функционально зависит от номера склада.

Функциональная зависимость имеет место, когда значения кортежа на одном множестве атрибутов однозначно определяют значения кортежа на другом множестве атрибутов (или на одном атрибуте).

Пусть **r** со схемой **R**, **X** и **Y** – подмножества **R**. Отношения **r** удовлетворяет **функциональной зависимости (F-зависимости) X→Y**, если $\pi_Y (\sigma_{X=x} (r))$ имеет не более чем один кортеж для каждого значения $x \in X$.

Функциональную зависимость будем обозначать следующим образом:

- Поставщик → Адрес,
- {Товар, Поставщик}→ Цена,
- {Товар, Поставщик}→ Склад,
- Склад → Объем.

А читаются они так:

- Поставщик функционально определяет адрес,
- Товар и поставщик функционально определяют цену,
- Товар и поставщик функционально определяют склад,
- Склад функционально определяет объем.

На языке функциональных зависимостей ключ для схемы **R** – это подмножество $K \subseteq R$, такое, что $K \rightarrow R$, и ни какое собственное подмножество $K' \subseteq K$ этим свойством не обладает.

Для отношения r(R) существует некоторое семейство F-зависимостей, которым это отношение удовлетворяет. Здесь возникает та же проблема, что и с ключами: одно состояние отношения может удовлетворять F-зависимости, а другое – нет. Требуется выявить семейство F-зависимостей, которому удовлетворяют все допустимые состояния r.

Если известны некоторые функциональные зависимости, то можно вывести остальные с помощью аксиом вывода. **Аксиомы вывода** – это правила, устанавливающие, что если отношение удовлетворяет некоторой функциональной зависимости, то оно удовлетворяет и некоторой другой функциональной зависимости.

F1.	Рефлексивность	X→X		
F2.	Пополнение	X→Y	⇒	XZ→Y
F3.	Аддитивность	X→Y, X→Z	⇒	X→YZ
F4.	Проективность	X→YZ	⇒	X→Y
F5.	Транзитивность	X→Y, Y→Z	⇒	X→Z
F6.	Псевдотранзитивность	X→Y, YZ→W	⇒	XZ→W

С помощью аксиом можно вывести все множество функциональных зависимостей F^+, которые влечет за собой множество исходных функциональных зависимостей F. Это означает, что система аксиом является **полной.**

§ 3 Нормальные формы

Нормальные формы представляют собой ограничения на схему отношения, избавляющие ее от нежелательных свойств, которые были перечислены выше. Прежде чем приводить отношения к нормальной форме следует построить все функциональные зависимости между атрибутами, которые существуют в предметной области.

Схема отношения **R** находится в **первой нормальной форме** (**1НФ**), если значения всех атрибутов являются атомарными (не составными), т.е. значение каждого атрибута не является ни списком, ни множеством значений.

Например, атрибут *ФИО* является составным, состоит из трех данных: фамилии, имени и отчества.

Чтобы привести схему в 1НФ, нужно все составные атрибуты заменить простыми.

Чтобы избавиться от избыточности информации, хранящейся в базе данных, и аномалий обновления, существуют вторая и третья нормальные формы.

Схема отношения **R** находится во **второй нормальной форме** (**2НФ**), если она находится в первой нормальной форме, и каждый непервичный атрибут функционально полно зависит от первичного ключа.

Неполная функциональная зависимость от ключа присутствует в отношении, если какой-либо атрибут, не входящий в ключ, функционально зависит от части атрибутов, входящих в ключ.

Любой непервичный атрибут обязательно функционально зависит от всех первичных атрибутов по определению ключа отношения. А если какой-либо непервичный атрибут, кроме того, функционально зависит не от всех, а от части первичных атрибутов, то это и есть неполная функциональная зависимость.

Например, в отношении **Поставки** первичными атрибутами являются *Товар* и *Поставщик*. Атрибут *Цена* функционально полно зависит от ключа, а атрибут *Адрес* зависит от части ключа, только от атрибута *Поставщик*, это неполная функциональная зависимость. Значит, схема **Поставки** не находится во 2НФ.

Чтобы привести схему, находящуюся в 1НФ, ко 2НФ, нужно разбить схему на несколько схем:

- выполнить проекцию схемы **R** на первичные атрибуты и атрибуты, функционально полно зависящие от ключа, т.е. исключить непервичные атрибуты, которые неполно зависят от ключа,

- для каждой неполной функциональной зависимости выполнить проекцию схемы **R** на атрибуты, входящие в эту зависимость, т.е. оставить часть ключа отношения **R** и атрибуты, функционально зависящие от этой части.

В примере с отношением **Поставки** в результате приведения схемы ко 2НФ получатся два отношения:

- **Поставки1**(Товар, Поставщик, Цена, Склад, Объем)
- **Поставки2**(Поставщик, Адрес)

Однако информация об объеме склада продолжает дублироваться. Для устранения этого недостатка схемы существует понятие третьей нормальной формы.

Схема отношения **R** находится в **третьей нормальной форме (3НФ**), если она находится во второй нормальной форме и в ней отсутствуют транзитивные зависимости непервичных атрибутов от ключа.

Транзитивная зависимость имеет место, если какой-либо непервичный атрибут функционально зависит от другого непервичного атрибута, а тот в свою очередь функционально зависит от ключа.

Схема отношения **Поставки1**(*Товар, Поставщик, Цена, Склад, Объем*) не находится в 3НФ, так как в ней присутствует транзитивная зависимость:

{*Товар, Поставщик*} → *Склад*, *Склад* → *Объем*.

Чтобы привести схему, находящуюся во 2НФ, в 3НФ, нужно:

- выполнить проекцию схемы **R** на первичные атрибуты и атрибуты, транзитивно не зависящие от ключа, т.е. исключить непервичные атрибуты, которые транзитивно зависят от ключа,
- для каждого транзитивно зависимого непервичного атрибута выполнить проекцию схемы **R** на атрибуты, входящие во вторую часть транзитивной зависимости, т.е. оставить только непервичные атрибуты отношения **R**, между которыми имеется функциональная зависимость.

В примере с отношением **Поставки1** в результате приведения схемы к 3НФ получатся два отношения:

- **Поставки11**(Товар, Поставщик, Цена, Склад)
- **Поставки12**(Склад, Объем)

Таким образом, последовательно выполняя разбиение исходной схемы отношения на несколько других схем согласно рассмотренным правилам, получаем схему в 3НФ, свободную от аномалий обновления и дублирования информации, о чем говорилось в начале раздела.

Процесс разбиения схемы отношения на несколько других схем называется **декомпозицией** схемы отношения.

В рассмотренном примере в результате декомпозиции вместо одного отношения **Поставки** мы получили три новых отношения:

- **Поставки11**(Товар, Поставщик, Цена, Склад)
- **Поставки12**(Склад, Объем)
- **Поставки2**(Поставщик, Адрес)

При такой схеме, состоящей из трех связанных внешними ключами отношений, не будет дублирования информации об адресе поставщика и объеме склада, если склад пуст, то объем его останется в базе данных, если поставщик не поставляет товары, то его адрес все равно будет храниться в базе данных.

Отношение находится в **нормальной форме Бойса-Кодда (НФБК)**, если оно находится в 3НФ и в нем отсутствуют зависимости первичных атрибутов от непервичных. Эквивалентное определение требует, чтобы все левые части функциональных зависимостей были потенциальными ключами.

Например, отношение **Лекции**(*Студент, Предмет, Преподаватель*). Каждый студент, изучающий данный предмет, обучается только одним преподавателем, Каждый преподаватель ведет только один предмет, но каждый предмет может вести несколько преподавателей. Из этих ограничений вытекают следующие функциональные зависимости:

{Студент, Предмет} → Преподаватель;

Преподаватель → Предмет.

Лекции

Студент	*Предмет*	*Преподаватель*
Иванов	Алгебра	проф. Белый
Иванов	Физика	проф. Яров
Петров	Алгебра	проф. Белый
Петров	Физика	проф. Серов

Отношение **Лекции** находится в 3НФ. Но оно страдает аномалиями обновления. Если требуется удалить информацию о том, что Петров изучает физику, то утратится информация о том, что профессор Серов преподает физику.

Эти трудности вызваны тем, что существует функциональная зависимость первичного атрибута от непервичного. Решением проблемы было бы разбиение отношения на два: **Лекции1**(*Студент, Преподаватель*) и **Лекции2**(*Преподаватель, Предмет*).

Лекции1

Студент	*Преподаватель*
Иванов	проф. Белый
Иванов	проф. Яров
Петров	проф. Белый
Петров	проф. Серов

Лекции2

Преподаватель	*Предмет*
проф. Белый	Алгебра
проф. Яров	Физика
проф. Серов	Физика

§ 4 Многозначные зависимости

В **R** имеет место **многозначная зависимость X→→Y**, если для любого отношения **r** со схемой **R** и кортежей **t**, **s∈r**: **t[X]=s[X]**, **r** содержит также кортежи **u**, **v**:

1. **u[X]=v[X]=t[X]=s[X]**
2. **u[Y]=t[Y], u[R – X – Y]=s[R – X – Y]**
3. **v[Y]=s[Y], v[R – X – Y]=t[R – X – Y]**

Атрибут **X многозначно определяет** атрибут **Y** в **R** (или **Y** многозначно зависит от **X**), если каждому значению атрибута **X** соответствует множество (возможно пустое) значений атрибута **Y**, никак не связанных с другими атрибутами **R**.

Для наличия в отношении многозначной зависимости необходимо иметь минимум три атрибута.

Правила вывода для многозначных зависимостей, **X,Y,Z ∈ R** :

M1. Дополнение **X→→Y** ⇒ **X→→R – X – Y**

M2. Транзитивность **X→→Y, Y→→Z** ⇒ **X→→Z – Y**

Например, отношение **Преподаватель**(*Номер*, *Имя_ребенка*, *Предмет*, *Должность*) имеет две многозначные зависимости и одну функциональную:

Номер→→Имя,

Номер→→Предмет,

Номер→Должность.

Каждый преподаватель может иметь несколько детей, вести несколько предметов и занимать одну должность, каждый предмет могут вести несколько преподавателей.

Преподаватель

Номер	*Имя*	*Предмет*	*Должность*
111	Ольга	Алгебра	Доцент
111	Иван	Геометрия	Доцент
111	Ольга	Геометрия	Доцент
111	Иван	Алгебра	Доцент
115	Сергей	Алгебра	Профессор

Это отношение, во-первых, содержит избыточную информацию – должность преподавателя повторяется несколько раз. Во-вторых, оно не свободно от аномалий обновления: если у преподавателя появляется еще один ребенок, необходимо добавить в отношение не один кортеж, а столько, сколько предметов ведет этот преподаватель. Аналогично, при добавлении еще одного предмета, требуется добавить столько кортежей, сколько детей имеет преподаватель. А если преподаватель не имеет детей, то информаци о том, какие предметы он ведет, вообще нельзя занести в отношение.

Отношение находится в **четвертой нармальной форме** (**4НФ**), если оно находится в нормальной форма Бойса-Кодда и в нем отсутствуют многозначные зависимости, которые не являются функциональными.

Отношение **Преподаватель** необходимо разбить на три отношения: **Преподаватель1** (*Номер*, *Должность*), **Преподаватель2**(*Номер*, *Имя*), **Преподаватель3**(*Номер*, *Предмет*).

Преподаватель1

Номер	*Должность*
111	Доцент
115	Профессор

Преподаватель2

Номер	*Имя*
111	Ольга
111	Иван

Преподаватель3

Номер	*Предмет*
111	Алгебра
111	Геометрия

115	Сергей	115	Алгебра

§ 5 Свойства декомпозиции

Пусть исходная схема **R** с множеством функциональных зависимостей **F** была приведена в результате декомпозиции к схеме отношений $\mathbf{R_1}$, $\mathbf{R_2}$, … , $\mathbf{R_k}$ с множеством функциональных зависимостей $\mathbf{F_1}$, $\mathbf{F_2}$, … , $\mathbf{F_k}$.

1. Декомпозиция обладает свойством **соединения без потерь**, если любой отношение r со схемой R удовлетворяет соотношению:

$$\mathbf{r = \pi_{R1}(r) \bowtie \pi_{R2}(r) \bowtie \pi_{R3}(r) \bowtie \ldots \bowtie \pi_{Rk}(r)}$$

2. Декомпозиция обладает свойством **сохранения зависимостей**, если из объединения всех зависимостей $\mathbf{F_1}$, $\mathbf{F_2}$, … , $\mathbf{F_k}$ можно вывести все зависимости **F**.

§ 5.1 *Свойство декомпозиции – соединение без потерь*

Пример

Рассмотрим отношение **Студент**(*Номер*, *Город*, *Группа*), которое содержит информацию о студенте, включающую номер зачетной книжки, город, в котором он родился, и номер группы, в которой он обучается. Каждый студент учится лишь в одной группе и родился только в одном городе. Это отражают следующие зависимости: Номер → Город, Номер → Группа. Предложим две декомпозиции:

1. **Студент1**(*Номер*, *Группа*) и **Студент2**(*Номер*, *Город*)
2. **Студент3**(*Номер*, *Группа*) и **Студент4**(*Группа*, *Город*)

Студент

Номер	Город	Группа
1	Тула	101
2	Тула	102
3	Омск	101

Студент1

Номер	Группа
1	101
2	102
3	101

Студент2

Номер	Город
1	Тула
2	Омск
3	Тула

Студент3

Номер	Группа
1	101
2	102
3	101

Студент4

Группа	Город
101	Тула
101	Омск
102	Тула

При выполнении операции естественного соединения для отношений **Студент1** и **Студент2** получается исходное отношение **Студент**.

Студент1 ⋈ Студент2

Номер	Город	Группа
1	Тула	101
2	Тула	102
3	Омск	101

Студент3 ⋈ Студент4

Номер	Город	Группа
1	Тула	101
1	Омск	101
2	Тула	102
3	Тула	101
3	Омск	101

Результат операции естественного соединения для отношений **Студент3** и **Студент4** содержит кроме кортежей исходного отношения **Студент** еще лишние кортежи. Такая ситуация называется «**ловушкой соединения**» – если в результате соединения двух отношений, полученных после декомпозиции, добавляются лишние кортежи или, наоборот, теряются кортежи, которые были в исходном отношении.

Это означает, что вторая декомпозиция не удовлетворяет свойству соединение без потерь. А про первую декомпозицию пока сказать ничего нельзя, так как она прошла тест только на одном примере.

Для проверки выполнения свойства декомпозиции соединение без потерь можно воспользоваться следующей теоремой:

Теорема Пусть R_1, R_2, отношения, полученные в результате декомпозиции отношения R и множества функциональных зависимостей F. Декомпозиция обладает свойством соединения без потерь тогда и только тогда, когда F содержит хотя бы одну из функциональных зависимостей: $\mathbf{R_1 \cap R_2 \rightarrow R_1 - R_2}$ или $\mathbf{R_1 \cap R_2 \rightarrow R_2 - R_1}$.

Для рассмотренного примера:

Первая декомпозиция: R_1=(Номер, Группа) R_2=(Номер, Город).

$R_1 \cap R_2 \rightarrow R_1 - R_2$ == Номер → Группа

$R_1 \cap R_2 \rightarrow R_2 - R_1$ == Номер → Город

Обе функциональные зависимости принадлежат множеству функциональных зависимостей отношения **Студент**. Следовательно, первая декомпозиция обладает свойством соединение без потерь.

Вторая декомпозиция: R_1=(Номер, Группа) R_2=(Группа, Город).

$R_1 \cap R_2 \rightarrow R_1 - R_2$ == Группа → Номер

$R_1 \cap R_2 \rightarrow R_2 - R_1$ == Группа → Город

Ни одна из функциональных зависимостей не принадлежат множеству функциональных зависимостей отношения **Студент**. Следовательно, вторая декомпозиция не обладает свойством соединение без потерь.

Для того, чтобы проверить обладает ли декомпозиция свойством соединения без потерь, можно воспользоваться методом табло.

Пусть отношение R(A, B, C, D) с множеством функциональных зависимостей F={**A→B**, **B→C**, **CD→B**, **C→D**} в результате декомпозиции было разделено на отношения R_1(A, B), R_2(B, D), R_3(A, B, C), R_4(B, C, D)

	A	**B**	**C**	**D**
$\mathbf{R_1}$	a_1	a_2		
$\mathbf{R_2}$		a_2		a_4
$\mathbf{R_3}$	a_1	a_2	a_3	
$\mathbf{R_4}$		a_2	a_3	a_4

Таблица заполняется следующим образом: для каждой функциональной зависимости **X→Y**, если для для атрибутов **X** найдутся строки, где в соответсвующих местах стоят $\mathbf{a_j}$, то пустые клетки этих строк, соответствующие атрибутам **Y** заменяются на $\mathbf{a_j}$.

Если в результате появится строка, полностью заполненная $\mathbf{a_j}$, то данное соединение без потерь.

§ 5.2 *Свойство декомпозиции – сохранение функциональных зависимостей*

Декомпозиция отношения **Лекции**, проведенная при приведении его к НФБК, обладает свойством соединения без потерь, но она не сохраняет функциональные зависимости. Объединение множеств функциональных зависимостей двух отношений декомпозиции будет содержать только одну зависимость: «Преподаватель → Предмет», а зависимость «{Студент, Предмет} → Преподаватель» в результате декомпозиции потеряна.

§ 5.3 *Общие вопросы декомпозиции*

Теорема Всякая схема отношения R с множеством функциональных зависимостей F может быть приведена к декомпозиции в 3НФ с сохранением зависимостей и соединением без потерь.

Теорема Всякая схема отношения R с множеством функциональных зависимостей и многозначных зависимостей может быть приведена к декомпозиции в 4НФ с соединением без потерь.

Проблемы декомпозиции:

1. Временная сложность не ограничивается полиномиальной.
2. Число порожденных схем может оказаться бóльшим, чем это необходимо.

Глава 6. Построение реляционной модели данных

Существует два метода проведения декомпозиции:

1. Объединить все атрибуты в одно отношение, а затем приводить его к 3НФ, первоначально построив множество всех функциональных зависимостей, вытекающих из предметной области.
2. Строить отношения исходя из инфологической модели.

§ 1 Построение реляционной модели данных на основе инфологической модели

От модели предметной области очень легко перейти к реляционной модели данных. Для этого нужно выполнить несколько последовательных действий:

1) Построить схемы отношения для классов объектов.
   a) Для каждого класса объектов сформировать отдельное отношение, атрибутами которого будут свойства этого класса. Например, для объекта Студент можно составить следующую схему отношения:

      Студент(*ФИО, Курс, Год_рождения, Номер*).

   b) Далее, для каждого атрибута определить домен и привести отношение к первой нормальной форме. В данном примере необходимо заменить атрибут *ФИО* тремя атрибутами *Фамилия*, *Имя*, *Отчество*.
   c) Затем, нужно выделить в отношении ключ. В данном случае ключом будет атрибут *Номер*, так как номера зачетных книжек не дублируются.

      Если ключ получается громоздким, то есть включает в себя много атрибутов или атрибут, имеющий в качестве домена длинную строку символов, то удобно ввести искусственный (суррогатный) ключ – новый атрибут с целочисленным доменом. Значения этого домена будут уникальными для каждого кортежа отношения.
2) Построив схемы отношений для каждого класса объектов, можно перейти к построению отношений для связей между объектами.
   a) Для связи схема отношения строится следующим образом: для каждого класса, участвующего в связи, из соответствующей ему схемы отношения ключевые атрибуты копируются в новое отношение; если связь имеет свойства, то они также включаются в схему как атрибуты.
   b) Ключом такого отношения, скорее всего, будут либо ключ одного из классов, входящих в связь (это справедливо для связи «один-ко-многим», ключом будет ключ отношения, соответствующего классу, входящему в связь со стороны «многие»), либо оба ключа вместе (для бинарной связи). Иногда бывает удобно ввести искусственный ключ.

      Например, связь между классами Студент и Группа выльется в следующее отношение:

      Группы(*Студент, Номер_группы*),

      где атрибут *Студент* будет содержать номер зачетной книжки студента, который является ключом схемы отношения **Студент**, а *Номер_группы* – номер группы, в которой состоит студент. В этом отношении ключом будет атрибут *Студент*, так как, согласно предметной области, каждый студент состоит только в одной группе и в этом отношении не будет двух кортежей с одинаковыми номерами зачетных книжек. Подчеркнем здесь тот факт, что в схеме отношения для связи между классами объектов названия атрибутов, соответствующих ключам схем для классов объектов, необязательно должны совпадать с названиями соответствующих первичных атрибутов в схемах для классов объектов.

      Во втором примере для связи между классами Товар и Поставщик схема отношения будет иметь следующий вид:

      Поставки(*Товар, Поставщик, Дата, Количество*).

      Это связь вида «многие-ко-многим», следовательно, в ключ должны входить и атрибут *Товар* и атрибут *Поставщик*. Но один и тот же поставщик может поставлять один и тот же товар несколько раз, следовательно, двух атрибутов для ключа недостаточно. Если в предметной области определено такое ограничение, что в один и тот же день поставщик не может поставить один и тот же товар, то тогда ключом будут три атрибута {*Товар, Поставщик, Дата*}. Если же этого ограничения нет, то в этом случае даже всех атрибутов будет недостаточно для определения ключа и необходимо ввести искусственный ключ (например, *Номер_поставки* или т. п.).
   c) Для иерархической связи – ключ отношения, соответствующего классу родителю, будет ключом отношений, соответствующих классам потомкам.

      Для ограничения несвязность – для всех подклассов и суперкласса можно организовать одно отношение, включающее все атрибуты, плюс один атрибут, выражающий тип подкласса. Например, для связи студент (свойства – ФИО, номер зачетной книжки) – первокурсник, второкурсник (дополнительное свойство – ФИО руководителя курсовой работы), можно создать отношение с атрибутами: ФИО, номер зачетной книжки, ФИО руководителя курсовой работы, Курс –

дополнительный атрибут с доменом {первокурсник, второкурсник}. Для первокурсников атрибут ФИО руководителя будет иметь пустое значение.

d) Схемы отношений, соответствующие связям между классами, будут иметь внешние ключи – атрибуты, соответствующие ключам схем классов, входящих в связь. Например, в отношении **Поставки** атрибут *Товар* будет внешним ключом, ссылающимся на отношение **Товар**, а атрибут *Поставщик* будет внешним ключом, ссылающимся на отношение **Поставщик.**

e) Для атрибутов, соответствующих свойствам связи, необходимо определить домены.

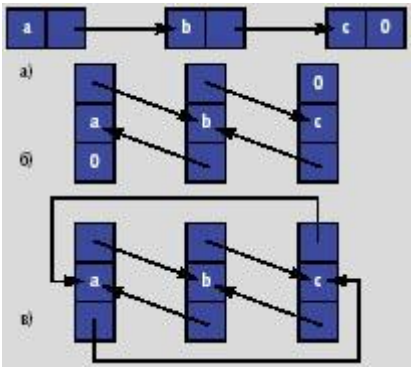
3) Конечно, схема может получиться неоптимальной; например, связь, относящуюся к типу «один-ко-многим», в некоторых случаях бывает удобным не оставлять как отдельное отношение, а включить в схему отношения для класса объекта, который входил в связь со стороны «многие». Однако при перестроении схемы будьте внимательны, могут быть нарушены условия одной из нормальных форм.

4) Проверить, находятся ли все отношения в 3НФ. При правильном выполнении 1–3 шагов алгоритма построения реляционной модели отношения будут находиться в 3НФ, исключением могут быть только составные атрибуты, нарушающие условия 1НФ, но, исправив их, вы получите 3НФ.

Глава 7. Представление сложных структур данных в реляционной базе данных

§ 1 Список

Односвязный список – динамическая структура данных, каждый элемент которой имеет ссылку на следующий элемент. У последнего элемента ссылка на следующий элемент равна нулю.



Для хранения односвязного списка требуется таблица, состоящая из 4 полей:

Номер	*Название*	*Описание*
1	Номер элемента	Уникальный номер элемента, используется для ссылки на него других элементов списка. В качестве типа данных можно использовать «Автоинкрементное поле».
2	Тип элемента	принимает значения: 1 – первый элемент списка; 2 – последний элемент списка; 0 – внутренний элемент списка; 3 – единственный элемент списка.
3	Указатель на следующий элемент	Хранится номер элемента, следующего за текущим.
4	Информационное поле	Одно или несколько полей представляющих собой содержимое списка.

Пример

Номер элемента	Тип элемента	Следующий элемент	Информационное поле
1	1	2	A
2	0	3	S
3	2	null	E

В таблице можно хранить несколько списков, добавив еще одно поле с номером списка.

Двусвязный список отличается от односвязного тем, что содержит ссылку на предыдущий элемент. Нужно ли добавлять еще одно поле в таблицу?

§ 2 Стеки и очереди

Стек и очередь можно представить как односвязный список.

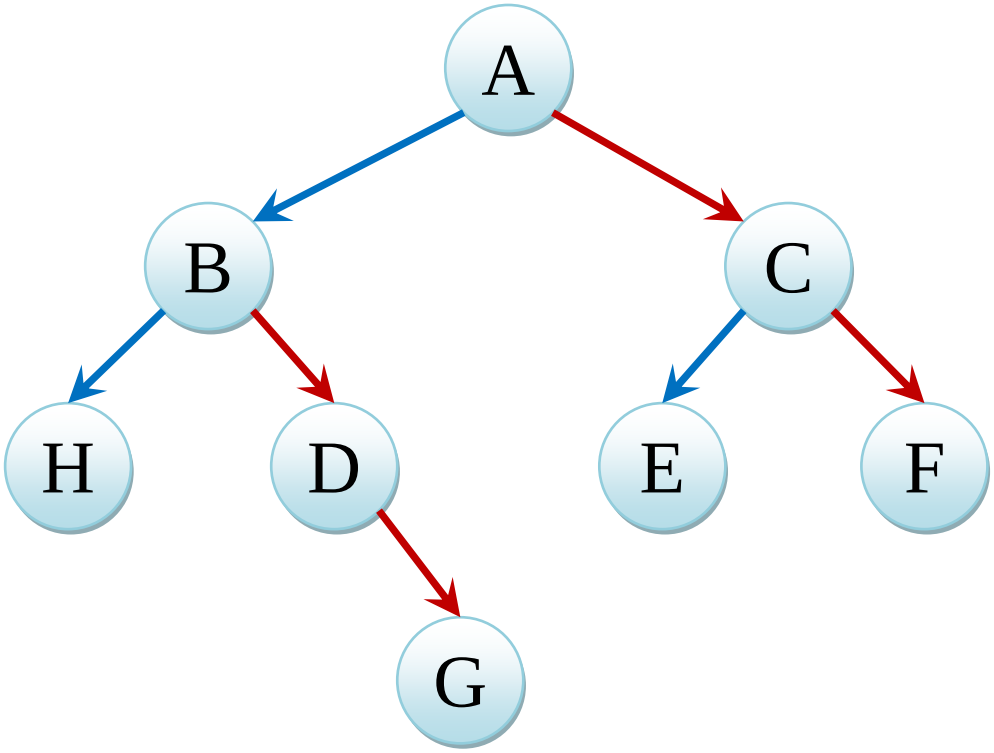
§ 3 Бинарные деревья

В бинарном дереве каждый узел хранит указатель на левого и правого потомка.

Для хранения бинарного дерева требуется таблица, состоящая из 5 полей:

Номер	*Название*	*Описание*
1	Номер вершины	Уникальный номер вершины дерева, используется для ссылки на него других вершин. В качестве типа данных можно использовать «Автоинкрементное поле».
2	Тип вершины	принимает значения: 1 – корневая вершина; 2 – лист; 0 – ветка.
3	Указатель на левого потомка	Хранится номер вершины – левого потомка.
4	Указатель на правого потомка	Хранится номер вершины – правого потомка.
5	Информационное поле	Одно или несколько полей представляющих собой содержимое вершины.

Пример



номер вершины	Тип вершины	Левый потомок	Правый потомок	Информационное поле
1	1	2	3	a
2	0	4	5	b
3	0	7	8	c
4	2	0	0	h
5	0	0	6	d
6	2	0	0	g
7	2	0	0	e
8	2	0	0	f

§ 4 Деревья и графы

Для хранения дерева достаточно ссылки на вершину предка, поэтому потребуется таблица, состоящая из 4 полей:

Номер	*Название*	*Описание*
1	Номер вершины	Уникальный номер вершины дерева, используется для ссылки на него других вершин. В качестве типа данных можно использовать «Автоинкрементное поле».
2	Тип вершины	принимает значения: 1 – корневая вершина; 2 – лист; 0 – ветка. Может быть уровень вершины или номер по порядку для упорядоченного дерева
3	Указатель на предка	Хранится номер вершины – предка (0 – для корневой вершины).
4	Информационное поле	Одно или несколько полей представляющих собой содержимое вершины.

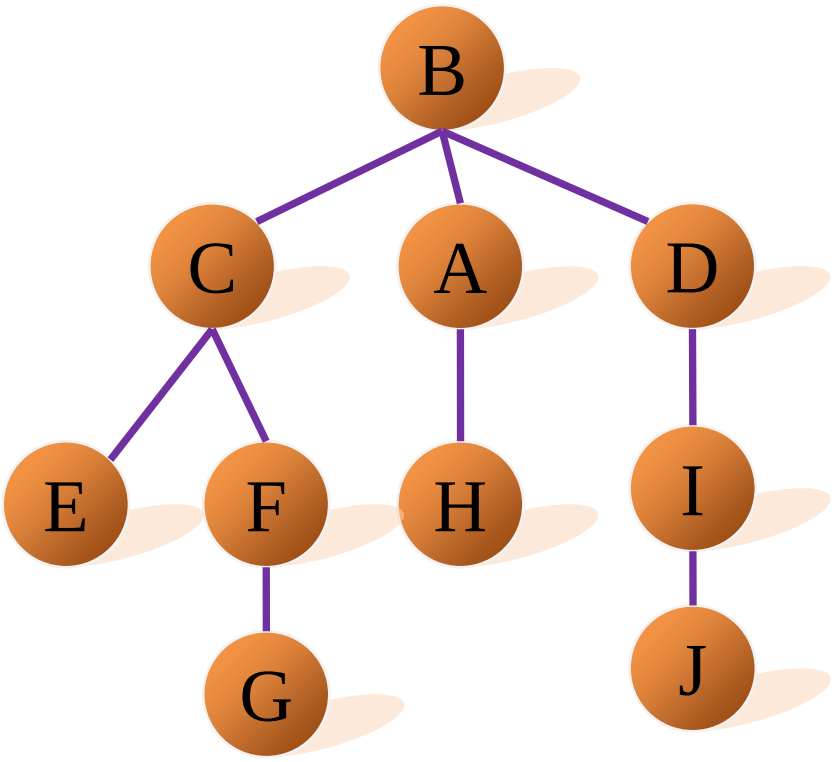
Для представления графа потребуется две таблицы: одна – для вершин, вторая – для дуг.

Номер	*Название*	*Описание*
1	Номер вершины	Уникальный номер вершины графа, используется для ссылки на него других вершин. В качестве типа данных можно использовать «Автоинкрементное поле».
2	Информационное поле	Одно или несколько полей представляющих собой содержимое вершины.

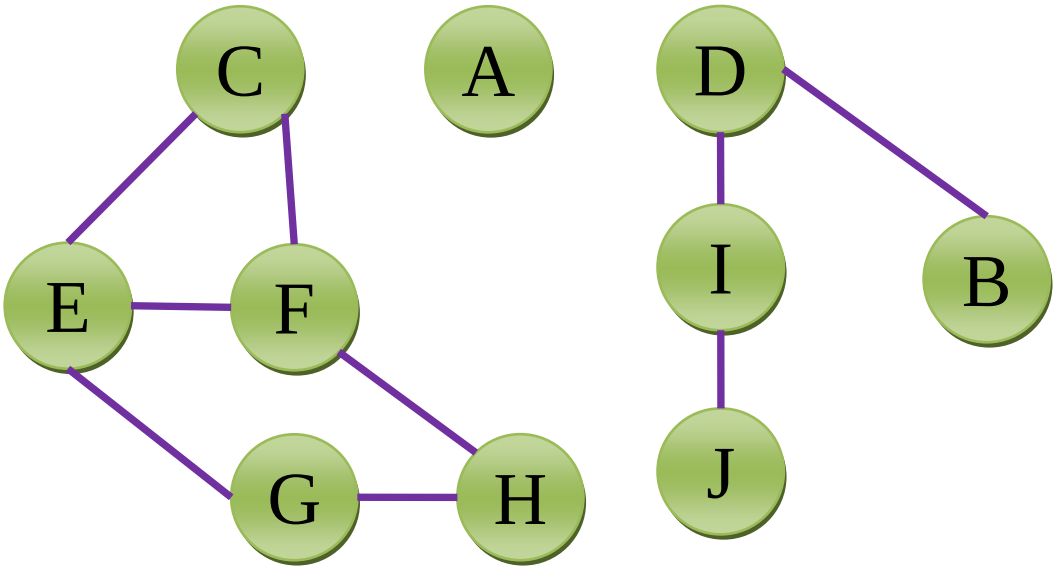
Номер	*Название*	*Описание*
1	Номер дуги	Уникальный номер дуги графа. В качестве типа данных можно использовать «Автоинкрементное поле».
2	Номер начальной вершины	Хранится номер вершины из первой таблицы.
3	Номер конечной вершины	Хранится номер вершины из первой таблицы.
4	Информационное поле	Одно или несколько полей представляющих собой содержимое дуги.

Для дерева можно организовать такую же структуру, как и для графа, состоящую из двух таблиц.

Пример



Номер вершины	Тип вершины	Предок	Информационное поле
1	0	2	A
2	1	null	B
3	0	2	C
4	0	2	D
5	2	3	E
6	0	3	F
7	2	6	G
8	2	1	H
9	0	4	I
10	2	9	J

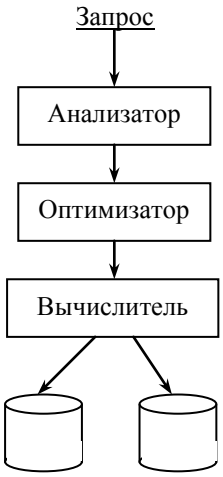


Номер вершины	Информационное поле
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J

Номер дуги	Начальная вершина	Конечная вершина	Информационное поле
1	3	5	null
2	3	6	null
3	5	7	null
4	5	6	null
5	6	8	null
6	8	7	null
7	4	2	null
8	4	9	null
9	9	10	null

Глава 8. Оптимизация запросов реляционной модели данных

§ 1 Общая схема



Оптимизация:
1. Получение эквивалентного, но более простого запроса.
2. Определение порядка вычисления операций реляционной алгебры.
3. Оптимизация алгоритма вычисления каждой операции.

§ 2 Получение эквивалентного, но более простого запроса

Эквивалентность зависит от схемы базы данных. Например, две операции: $\pi_{AB}(R_1 \rhd\lhd R_2)$ и $\pi_{AB}(R_2) \rhd\lhd \pi_{AB}(R_1)$ эквивалентны для схем $R_1(ABC)$ и $R_2(ABD)$, но неэквивалентны для схем $R_1(ABCD)$ и $R_2(ABCE)$.

§ 2.1 Законы, сохраняющие эквивалентность выражений

1. Законы коммутативности
   a). $R_1 \times R_2 \equiv R_2 \times R_1$
   b). $R_1 \rhd\lhd R_2 \equiv R_2 \rhd\lhd R_1$
2. Законы ассоциативности
   a). $(R_1 \times R_2) \times R_3 \equiv R_1 \times (R_2 \times R_3)$
   b). $(R_1 \rhd\lhd R_2) \rhd\lhd R_3 \equiv R_1 \rhd\lhd (R_2 \rhd\lhd R_3)$

   Законы для селекций и проекций
3. Каскад из нескольких проекций может быть скомбинирован в одну проекцию, если $\{A_1, \dots, A_n\} \subset \{B_1, \dots, B_m\}$:
$$\pi_{A1,\dots,An}(\pi_{B1,\dots,Bm}(R)) \equiv \pi_{A1,\dots,An}(R)$$
4. Каскад селекций может быть скомбинирован в одну селекцию, которая проверяет все условия одновременно:
$$\sigma_{F1}(\sigma_{F2}(R)) \equiv \sigma_{F1 \text{ and } F2}(R)$$
5. Перестановка селекции и проекции:
$$\sigma_F(\pi_{A1,\dots,An}(R)) \equiv \pi_{A1,\dots,An}(\sigma_F(R))$$
   В более общем случае, если условие F вовлекает также атрибуты $B_1, \dots, B_m$, которых нет среди $A_1,\dots,A_n$, то:
$$\pi_{A1,\dots,An}(\sigma_F(R)) \equiv \pi_{A1,\dots,An}(\sigma_F(\pi_{A1,\dots,An,\,B1,\dots,Bm}(R)))$$
6. Перестановка селекции с декартовым произведением. Если все упоминаемые в F атрибуты являются атрибутами R_1:
$$\sigma_F(R_1 \times R_2) \equiv \sigma_F(R_1) \times R_2$$
   Если F имеет вид (F_1 and F_2), где F_1 включает только атрибуты R_1, а F_2 – только атрибуты R_2, то:
$$\sigma_F(R_1 \times R_2) \equiv \sigma_{F1}(R_1) \times \sigma_{F2}(R_2)$$
7. Перестановка селекции с объединением:
$$\sigma_F(R_1 \cup R_2) \equiv \sigma_F(R_1) \cup \sigma_F(R_2)$$
8. Перестановка селекции с разностью:
$$\sigma_F(R_1 - R_2) \equiv \sigma_F(R_1) - \sigma_F(R_2)$$
9. Перестановка проекции с декартовым произведением. Пусть, $A_1,\dots, A_k$ – атрибуты R_1 и R_2, из них $B_1,\dots,B_n$ – атрибуты R_1, а $C_1,\dots,C_m$ – атрибуты R_2, тогда:
$$\pi_{A1,\dots,Ak}(R_1 \times R_2) \equiv \pi_{B1,\dots,Bn}(R_1) \times \pi_{C1,\dots,Cm}(R_2)$$

10. Перестановка проекции с объединением:

$$\pi_{A1,\ldots,Ak}(R_1 \cup R_2) \equiv \pi_{A1,\ldots,Ak}(R_1) \cup \pi_{A1,\ldots,Ak}(R_2)$$

§ 2.2 *Общие стратегии оптимизации*

1. Предварительно обрабатывать файл перед выполнением соединения: сортировка и создание индексов.
2. Искать общие подвыражения в выражении.
3. Выполнять операции селекции по возможности раньше
4. Собирать в каскады селекции и проекции. Последовательность таких операций, каждая из которых требует только одного операнда, может выполняться одновременно за один просмотр файла операнда.
5. Комбинировать проекции с предшествующими или последующими двуместными операциями.
6. Комбинировать некоторые селекции с предшествующим декартовым произведением и выполнять вместо них соединения.

§ 2.3 *Алгоритм оптимизации реляционных отношений*

1. Строим дерево разбора, представляющее выражение реляционной алгебры.
2. Представляем каждую селекцию в виде каскада селекций по правилу 4.
3. Перемещаем каждую селекцию, насколько это возможно, вниз в дереве, используя правила 4-8.
4. Перемещаем каждую проекцию, насколько это возможно, вниз в дереве, используя правила 3, 9, 10 и 5.
5. Комбинируем каждый каскад селекций и проекций в одиночную селекцию, одиночную проекцию или селекцию с последующей проекцией, используя правила 3,4,5.
6. Разбиваем внутренние узлы полученного в результате дерева на группы: каждый внутренний узел, представляющий двуместный оператор (произведения, объединения или разности), принадлежит группе вместе с непосредственными его предками, которые помечены унарными операторами (селекции или проекции). Включаем также в группу любую цепь потомков данного узла, помеченных унарными операторами, завершающуюся в листе дерева. Исключением является случай, когда двуместный оператор представляет собой декартово произведение без последующей селекции, которая комбинировалась бы с произведением, образуя эквисоединение.

Рассмотрим базу данных «Библиотека»:
Книги(*Название*, *Автор*, *Издательство*, *НомерКниги*)
Читатели(*ФИО*, *Адрес*, *НомерЧитателя*)
Выдачи(*НомерЧитателя*, *НомерКниги*, *Дата*)
Перечислить названия книг, которые были выданы до первого января 2001 года, и фамилии читателей, взявших эти книги.

$$\pi_{\text{Название, ФИО}}(\sigma_{\text{Дата<01/01/2001}}(\pi_{\text{Название, Автор, Издательство, НомерК, ФИО, Адрес, НомерЧ, Дата}}(\text{Читатели} \rhd\lhd \text{Выдачи} \rhd\lhd \text{Книги}) \equiv \pi_{\text{Название}}(\sigma_{\text{Дата<01/01/2001}}(\pi_{\text{Название, Автор, Издательство, НомерК, ФИО, Адрес, НомерЧ, Дата}}(\sigma_{\text{Читатели.НомерЧ = Выдачи.НомерЧ, Выдачи.НомерК=Книги.НомерК}}(\text{Читатели} \times \text{Выдачи} \times \text{Книги})$$

Строим дерево разбора для этого реляционного выражения.

§ 3 Порядок выполнения операций

Порядок выполнения операций соединения имеет значение.

Пример. Пусть $|R1|=16=2^4$, $|R2|=1024=2^{10}$, тогда количество операций поиска при выполнении соединения $R_1 \rhd\lhd R_2$ будет равно $16 \cdot \log_2 2^{10} = 160$, а при выполнении соединения $R_2 \rhd\lhd R_1 - 2^{10} \cdot \log_2 2^4 = 4096$.

Глава 9. Сетевая и иерархическая модели данных

§ 1 Сетевая модель

§ 1.1 Структура

Сетевая модель данных базируется на табличных и графовых представлениях.

Для представления объектов и связей между ними используется **структурная диаграмма** (диаграмма Бахмана). Помеченным вершинам в диаграмме соответствуют **типы записей**, а дугам – **типы связей.**

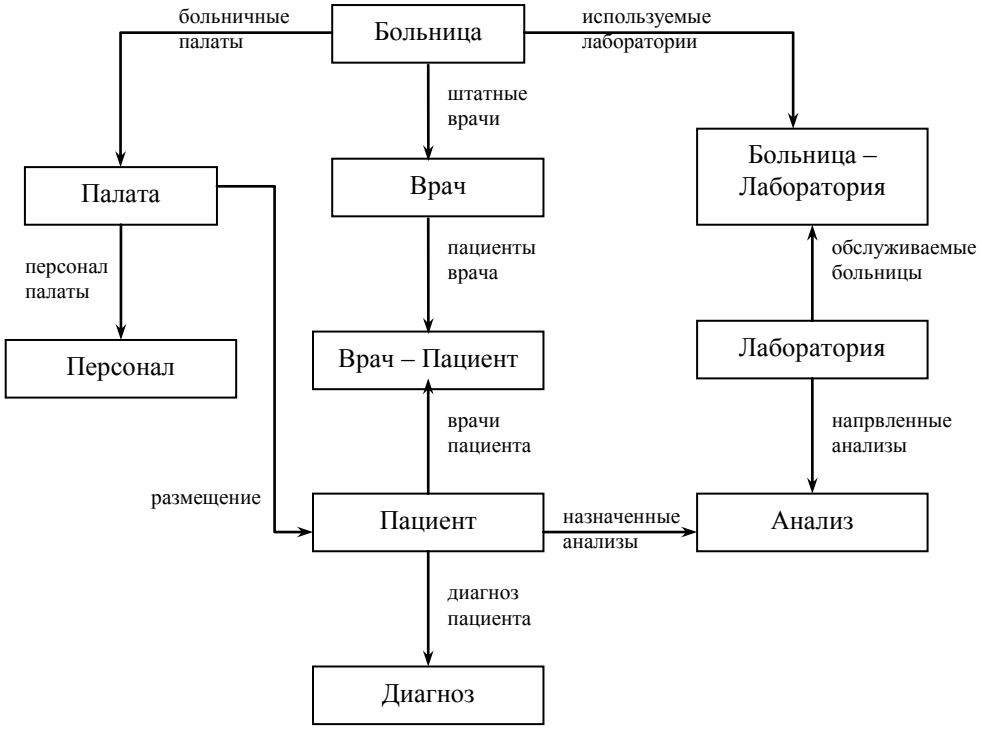
Типы записей – это классы объектов или сущностей.

Типы связей всегда соответствует функциональным связям, поэтому роли двух типов записей, участвующих в связи неодинаковы. Дуга исходит из **типа записи–владельца** и заходит в **тип записи–члена**. Направление дуги противоположно направленности функциональной зависимости.

Расширение структурной диаграммы состоит из записей и наборов.

Расширением типа записи является **таблица**, состоящая из записей. **Запись** состоит из нуля или более единиц данных. **Единица данных** может быть **элементом данных**, определенном на простом домене или **агрегатом** – именованной совокупностью данных, т.е. массивом, в том числе многомерным. Таблицы в сетевой модели рассматриваются как расширенные множества, это означает, что допускаются дубликаты строк, и строки могут быть упорядочены.

Расширение типа набора можно представить как совокупность дуг, соединяющих табличные представления расширений типов записей-владельцев и записей-членов. Подмножество дуг, соединяющих одну запись-владельца с нулем или более записей-членов, называют реализацией (экземпляром) набора, или просто **набором**.

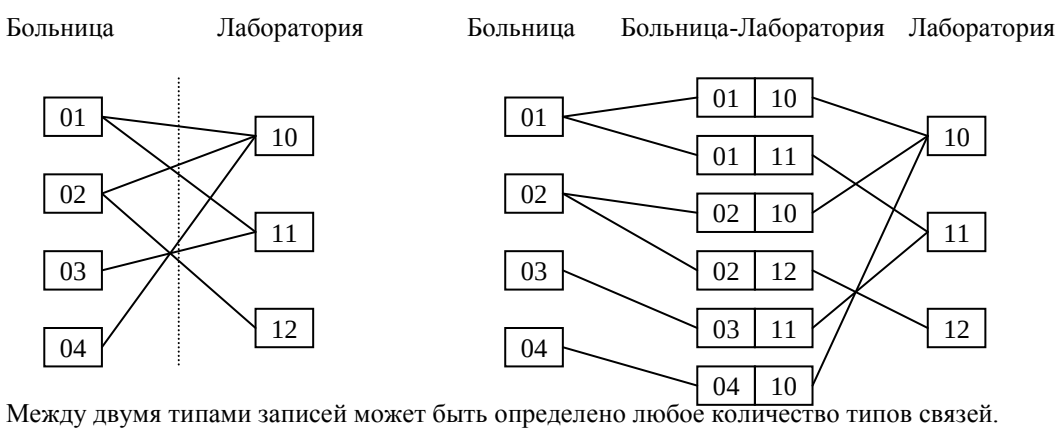


Особый тип набора данных, называемый **сингулярным**, имеет только одну реализацию, а запись-владелец отсутствует (владельцем является СУБД).

§ 1.2 Ограничения целостности

1. Связи должны быть функциональными. Это означает, что запись-член набора может иметь не более одной записи-владельца набора. Вместе с тем, запись может и не входить ни в один экземпляр набора, т.е. не иметь записи-владельца. Запись-владелец набора связана с множеством записей-членов, которое может быть и пустым. Запись не может быть членом более чем одного экземпляра набора каждого типа.
2. Для того чтобы представить связь «многие-ко-многим», необходимо ввести вспомогательный тип записи.

Связь "многие-ко-многим"



3. Между двумя типами записей может быть определено любое количество типов связей.
4. Тип записи может быть одновременно владельцем и членом нескольких типов связей
5. В некоторых моделях не разрешены рекурсивные связи, когда запись-владелец и запись-член набора принадлежат одному типу записей.
6. Тип членства в наборе
   a). **Варианты сохранения в наборе** (удаления из набора)
   - **Фиксированное членство** – как только запись стала членом набора, ее нельзя разъединить с владельцем или перенести в другой набор. Единственный способ исключения записи из набора – это удаление этой записи. (Например, больничная палата не может существовать без больницы и не может быть переведена в другую больницу.)
   - **Обязательное членство** – запись можно перевести из одного набора в другой, но нельзя исключить из набора без удаления самой записи. (Например, если каждый служащий больничного персонала прикреплен к какой-либо палате, то его можно перевести из состава персонала одной палаты в состав персонала другой палаты).
   - **Необязательное членство** – запись может быть исключена из набора в любой момент времени. (Например, пациент может покинуть больничную палату без назначения в другую)

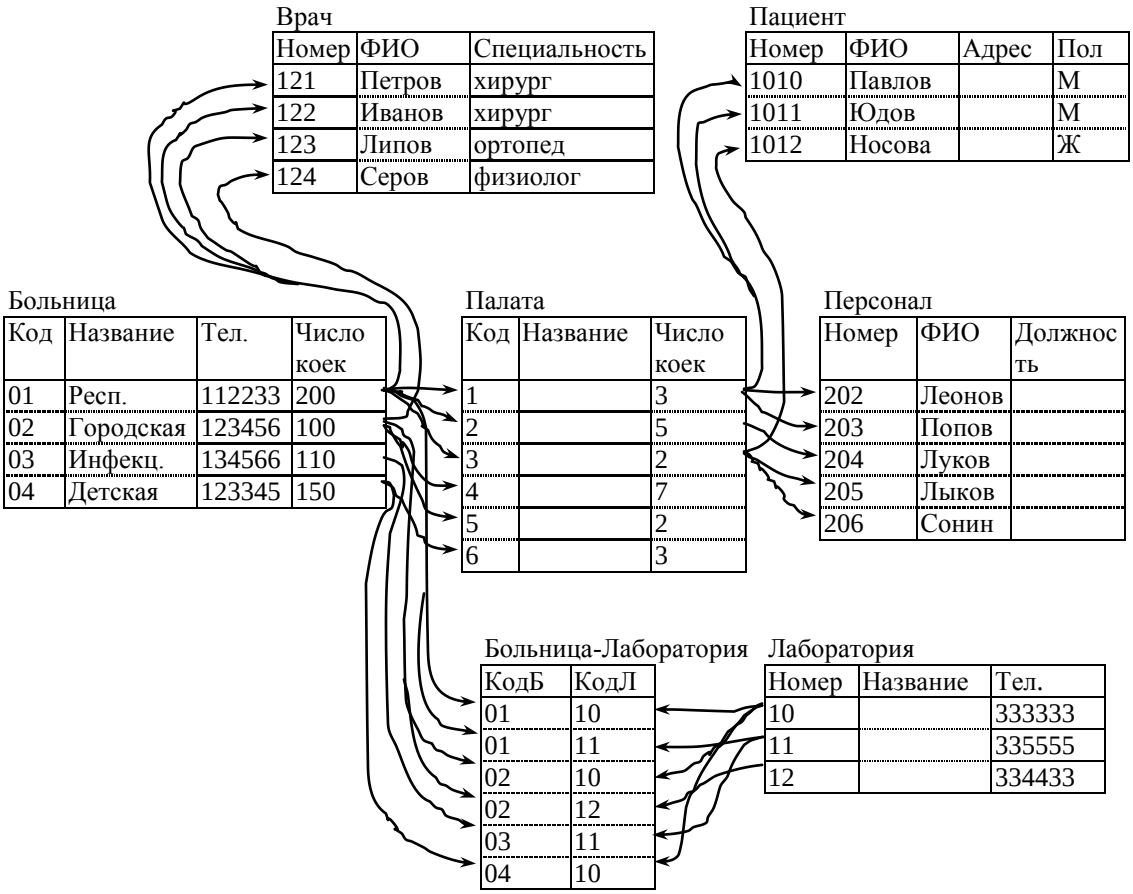
   b). **Варианты включения** в набор
   - **Автоматический тип** членства в наборе – в момент создания новой записи она должна быть включена в набор. Это означает, что, по крайней мере, в начальный момент запись не может существовать вне набора. (Например, диагноз всегда ставится конкретному пациенту.)
   - **Ручной тип** членства в наборе – создание новой записи не ведет к каким-либо действиям по включению ее в набор. (Например, Лаборатория)

Членство в наборе определяется парой: **"вариант включения – вариант удаления"**. Таким образом, возможны шесть комбинаций.

7. Структурное ограничение – устанавливает, что значение элемента данных записи-члена должно быть равно значению элемента данных записи-владельца набора. Это ограничение находит применение при представлении связей «многие-ко-многим».
8. Фиксация порядка в наборе – определяет позицию относительно существующих членов, в которую будет включен новый член.
   a). «первый» – «последний»: порядок относительно записи-владельца
   b). «следующий» – «предыдущий»: порядок относительно текущей записи-члена в наборе
   c). «сортированный»: порядок установлен для типа записи по ключам
   d). «системный-по-умолчанию»: порядок не установлен
9. Таблицы могут иметь ключи.

§ 1.3 *Операции*

1. Добавить, удалить, изменить запись
2. Включить, исключить запись из набора, перевести запись в другой набор
3. Извлечь – выделяет подграф
4. Навигационные операции – установление позиции в базе данных и перемещение по связям



§ 2 Иерархическая модель данных

§ 2.1 *Структура*

Структурная диаграмма иерархической модели данных должна быть упорядоченным деревом. Это означает, что относительное расположение вершин (слева или справа друг от друга) имеет значение. Такая структурная диаграмма называется **деревом определения**.

Вершина дерева соответствует типу сущности и называется **типом записи**. Тип записи состоит из одной или более единиц данных, определенных только на простом домене.

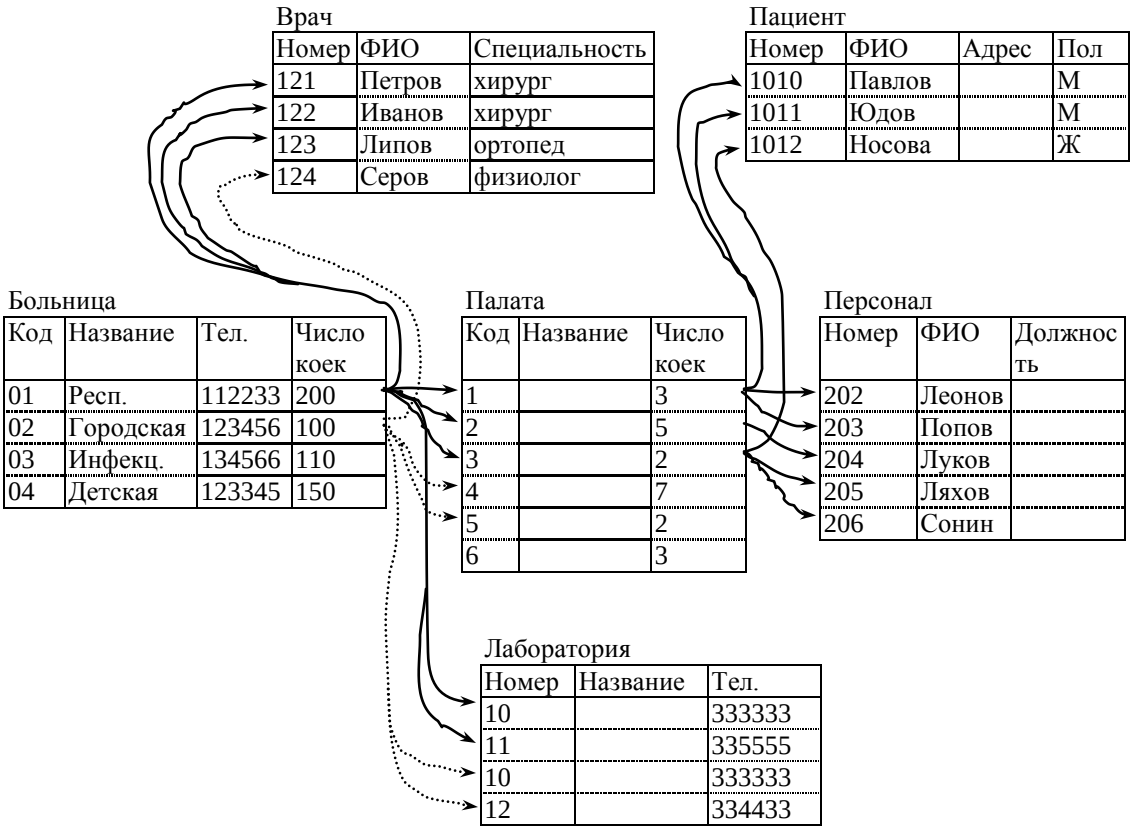
Дуга соответствует функциональной **связи**, называемой **исходный – порожденный** и всегда направлена от корня к листьям дерева. Так как между двумя типами записей может быть не более одной связи, то дуга не помечается. Дуга исходит из типа родительской связи и заходит в тип порожденной записи.

Среди вершин выделяется одна особая вершина, называемая **типом корневой записи**. В эту вершину не заходит ни одна дуга. Все другие вершины дерева определения носят название **типов зависимых записей.**

Иерархический путь в дереве определения – это последовательность типов записей, начинающихся с типа корневой записи, в которой типы записей выступают попеременно в ролях исходного и порожденного. **Уровень типа записи** относительно типа корневой записи определяется как длина пути от корневой записи, выраженная в числе дуг.

Расширением каждого типа записи является таблица (таблица упорядочена, дубликаты допускаются).

Альтернативным способом представления расширения дерева определения является лес или совокупность отдельных деревьев, состоящих из одной корневой записи и всех ее зависимых записей. Каждое такое дерево называется деревом базы данных.



§ 2.2 *Ограничения целостности*

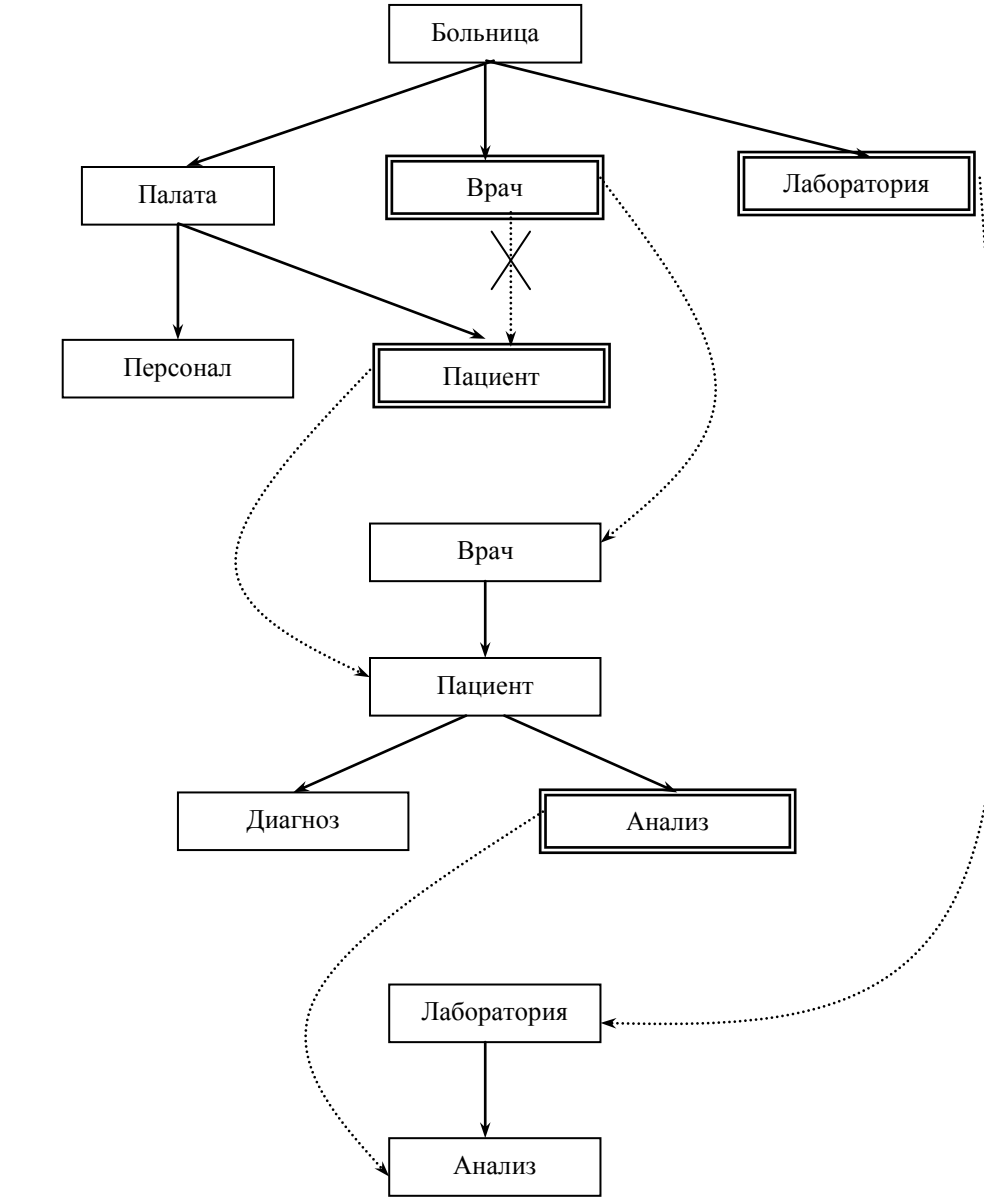
1. Все типы связей должны быть функциональными. Запись может иметь не более одной исходной записи (исключение корневой тип записи).
2. Структура связей должна быть древовидной.
3. Дерево упорядочено.
4. Невозможно представить связи, имеющие тип «многие-ко-многим». Это ограничение можно обойти двумя методами:
   a). Один из типов записей объявляется порожденным другого типа, а связь обеспечивается за счет дублирования записей первого типа.
   b). Два дерева определения. В одном из них первый тип выступает как родительский, а второй как порожденный. В другом дереве наоборот. В этом случае степень дублирования еще больше, но такое представление может оказаться желательным по соображениям эффективности доступа к данным.
5. Каждый тип записи может иметь не более одной исходной записи. Если какой-то тип записи имеет более одного родительского типа записи, то для представления этого необходимо определить несколько деревьев, охватывающих все связи. Непротиворечивость данных поддерживается посредством установления логических связей, которые определяют, что данные в различных деревьях должны быть идентичными. Логически порожденные типы записей содержат только ссылку на экземпляр соответствующей записи в той же или другой физической базе.
6. Каждая запись идентифицируется сцепленным ключом, состоящим из всех ключей записей на пути из корня к данной записи.

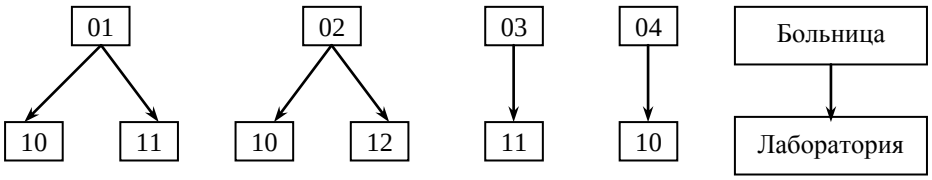
§ 2.3 *Операции*

Операции доступа начинаются от корневой вершины.

Операция извлечения – выделение поддерева или леса поддеревьев.

Навигационные операции: перейти на следующий уровень, перейти на следующую запись, перейти на следующую вершину дерева (слева направо).


Больница
Палата
Врач
Лаборатория
Персонал
Пациент
Врач
Пациент
Диагноз
Анализ
Лаборатория
Анализ


01
10
11
02
10
12
03
11
04
10
Больница
Лаборатория

Глава 10. Объектная модель данных

§ 1 Введение

Направление объектно-ориентированных баз данных (ООБД) возникло сравнительно давно. Публикации появлялись уже в середине 1980-х.

Объектно-ориентированная СУБД – это система, позволяющая создавать, хранить и использовать информацию в форме объектов. Полностью объектно-ориентированная СУБД обеспечивает также объектно-ориентированный интерфейс взаимодействия с пользователем.

В то время как классическая статья Кодда предоставила отчетливую спецификацию систем реляционных баз данных (модель данных и язык запросов), для систем объектно-ориентированных баз данных подобная спецификация отсутствует.

Основные трудности объектно-ориентированного моделирования данных проистекают из того, что такого развитого математического аппарата, на который могла бы опираться общая объектно-ориентированная модель данных, не существует. В большой степени поэтому до сих пор нет базовой объектно-ориентированной модели.

Второй чертой является отсутствие надежного теоретического базиса.

Система объектно-ориентированных баз данных должна удовлетворять двум критериям: она должна быть СУБД и при этом являться объектно-ориентированной системой, т.е. в максимально возможной степени находиться на уровне современных объектно-ориентированных языков программирования. Первый критерий означает пять свойств: стабильность, управление вторичной памятью, параллелизм, восстанавливаемость и средства обеспечения незапланированных запросов. Второй означает восемь свойств: сложные объекты, идентифицируемость объектов, инкапсуляцию, типы или классы, наследование, перекрытие методов совместно с поздним связыванием, расширяемость и вычислительную полноту.

§ 2 Ограничения реляционной технологии

1. Неестественное представление данных со сложной структурой.
2. Ограничение типов данных. Реляционная модель данных не позволяет определить набор операций, связанных с данными определенного пользователем типа (бинарными данными, например, большой текст или рисунок). Операции приходится задавать в конкретном приложении.
3. Сложность представления связей через внешние ключи (для понимания и работы с данными и выполнения запросов).

Пример:

Рассмотрим информацию о человеке, включающую номер паспорта, ФИО, номер телефона и ссылки на детей.

tblPerson

intPassport	txtName
1	Иванов Василий
2	Петров Сергей
3	Иванов Антон
4	Иванова Екатерина
5	Петрова Ольга
6	Серова Ирина

tblPhone

intPassport	intPhoneNumber
1	123
1	234
2	345
2	456
3	567
4	678

tblChild

intPassport	intChildPassport
1	3
1	4
2	5
4	6

Запрос: Получить номера телефонов всех внуков Иванова Василия.

SQL:

```
Select tblPhone.intPhoneNumber
From tblPhone, tblPerson, tblChild as A, tblChild as B
Where (tblPerson.intPassport = A.intPassport) and (A.intChaildPassort = B.intPassport) and
(B.intChildPassport = tblPhone.intPassport) and  (tblPerson.txtName = 'Иванов Василий')
```

4. Усложненный доступ к базе данных. Интерфейс между языком программирования и языком баз данных обычно усложнен, поскольку каждый язык имеет свой набор типов и свою модель вычислений. Организуя обращение к базе данных из прикладной программы, написанной, например, на C++, приходится подвергать данные структурной трансформации при передаче их из/в базу данных.

§ 3 Модель объектной базы данных

Концептуальная модель объектных данных (Conceptual Object Data Model (CODM))

§ 3.1 *Классы и объекты*

База данных состоит из множества классов. Каждый класс состоит из множества объектов, которые называются расширением класса.

Класс включает свойства и методы.

Свойства могут иметь:

простой тип: целое число, строка, вещественное число, логический тип;
ссылка на другой объект по его идентификационному номеру (OID);
кортеж $[A_1: v_1, \dots A_n: v_n]$, где A_i – имена атрибутов, v_i – значения;
множество $\{v_1, \dots v_n\}$, где v_i – значение.

Методы представляют собой процедуры, написанные на внутреннем языке СУБД.

Между классами определено наследование.

Объект представляет собой пару (<OID>, [<значения свойств>]).

Каждый объект имеет уникальный во всей базе данных идентификатор OID (это может быть адрес памяти, где хранится объект), никак не связанный с объектом. Идентификатор присваивается системой в момент создания объекта и никогда не изменяется. Изменение идентификатора означает создание нового объекта. Идентификатор сокрыт от пользователя.

```
(# 32, [   intPassport: 1,
           txtName: “Иванов Василий”,
           intPhoneNumber: {123, 234},
           intChild: {#445, #73}])
```

Пример для SQL:2003

```
Create Type PersonType As (
      Name Char(20),
      Address Row( Number Integer, Street Char(20), City Char(20), ZIP Char(6));
Create Type StudentType Under PersonType As (
      NZK integer,
      Department Char(20),
      Year Integer)
Method award_degree() Returns Boolean;
Create Method award_degree() For StudentType
      Language C
      External name ‘file:/home/admin/award_degree’;

New StudentType()
      .NZK (111111)
      .Department (‘Математический’)
      .Year (1)
      .Name (‘Иванов Иван’)
      .Address (Row (12, ‘пр. Ленина’, ‘Петрозаводск’, ‘185968’))
```

Для каждого класса автоматически создаются два метода: конструктор и деструктор.

Сложные объекты строятся из более простых при помощи конструкторов. Простейшими объектами являются такие объекты, как целые числа, символы, символьные строки произвольной длины, булевские переменные и числа с плавающей точкой (можно было бы добавить другие атомарные типы).

Имеются различные конструкторы сложных объектов: примерами могут служить конструкторы кортежей, множеств, мультимножеств, списков, массивов. Минимальный набор конструкторов, который должна иметь система, – это конструкторы множеств, списков и кортежей.

Множества необходимы, потому что они обеспечивают естественный способ представления наборов объектов реального мира.

Кортежи необходимы, потому что они представляют естественный способ представления свойств сущностей.

Списки или массивы важны потому, что они сохраняют порядок, который имеет место в реальном мире, а также потому, что они присутствуют во многих научных приложениях в виде матриц или временных рядов.

§ 3.2 Связи между объектами предметной области

§ 3.2.1 Родственные связи

Некоторые связи, например, «сотрудник является человеком», можно реализовать, используя аппарат наследования классов. Описав класс «Человек», можно описать класс-потомок «Сотрудник». Однако, создание объекта класса «Сотрудник», не влечет за собой создание объекта класса «Человек», и получается, что фактическая связь между объектами отсутствует по причине несуществования одного из объектов.

§ 3.2.2 Неродственные связи

Рассмотрим предметную область университет.

В предметной области можно выделить классы: Группа, Студент, Преподаватель, Предмет. Определим связь между классами Группа и студент. Эта связь будет иметь тип «один-ко-многим».

Связь «один-ко-многим» реализуется посредством контейнеров (множеств). Они могут устроены четырмя способами.

I способ: В классе, соответствующем объекту предметной области со стороны «один», описывается контейнер, содержащий объекты класса, соответствующего объекту предметной области со стороны «многие» (рис. 1). В объекте «Группа 22203» содержатся два объекта класса «Студент»: «Студент Мухин» и «Студент Комаров». Объекты класса «Студент» могут существовать только внутри класса «Группа», самостоятельно они не существуют.

II способ: В классе, соответствующем объекту предметной области со стороны «один», описывается контейнер, содержащий ссылки на объекты класса, соответствующего объекту предметной области со стороны «многие» (рис. 2). В этом случае объекты класса «Студент» существуют без объектов класса «Группа».

III способ: В классе, соответствующем объекту предметной области со стороны «многие», описана ссылка на объект класса, соответствующего объекту предметной области со стороны «один» (рис. 3).

IV способ: В классе, соответствующем объекту предметной области со стороны «один», описывается контейнер, содержащий ссылки на объекты класса, соответствующего объекту предметной области со стороны «многие». В классе, соответствующем объекту предметной области со стороны «многие», описана ссылка на объект класса, соответствующего объекту предметной области со стороны «один» (рис. 4).

Рис. 1 – Объект Группа содержит множество объектов Студент

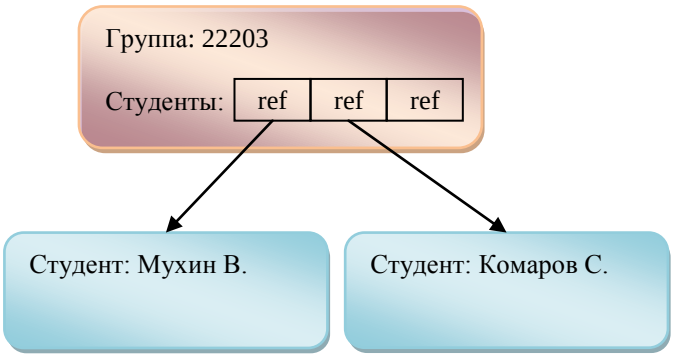


Рис. 2 – Объект Группа содержит множество ссылок на объекты Студент

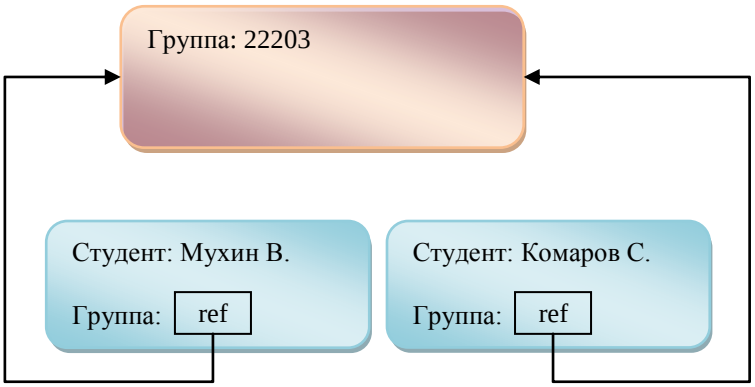


Рис. 3 – Объект Студент содержит ссылку на объект Группа

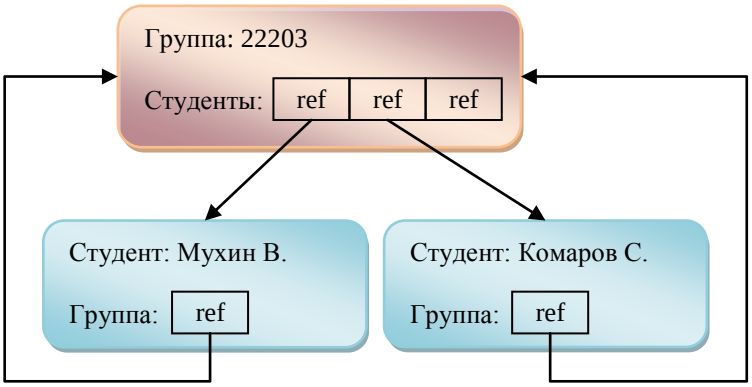


Рис. 4 – Объект Группа содержит множество ссылок на объекты Студент, каждый объект Студент содержит ссылку на объект Группа

Связь «многие-ко-многим» реализуется посредством контейнеров (множеств) в обоих классах (рис. 5). Либо связь представляется отдельным классом (рис. 6). Например, связь между классами «Предмет» и «Преподаватель» реализована с помощью нового класса «Преподавание», в котором находятся ссылка на объект класса «Предмет» и ссылка на объект класса «Преподаватель». Объект класса «Предмет» содержит множество ссылок на объекты класса «Преподавание», объект класса «Преподаватель» содержит множество ссылок на объекты класса «Преподавание».

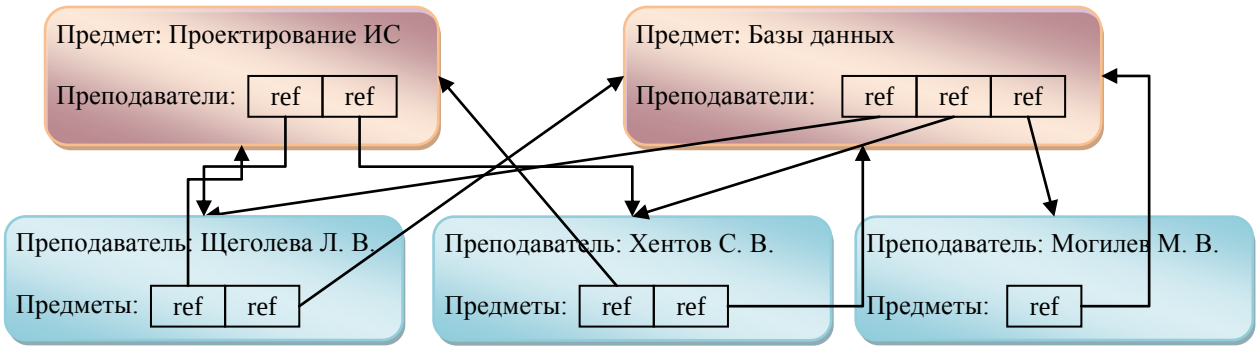


Рис. 5 – Объект Предмет содержит множество ссылок на объекты Преподаватель, объект Преподаватель содержит ссылки на объекты Предмет

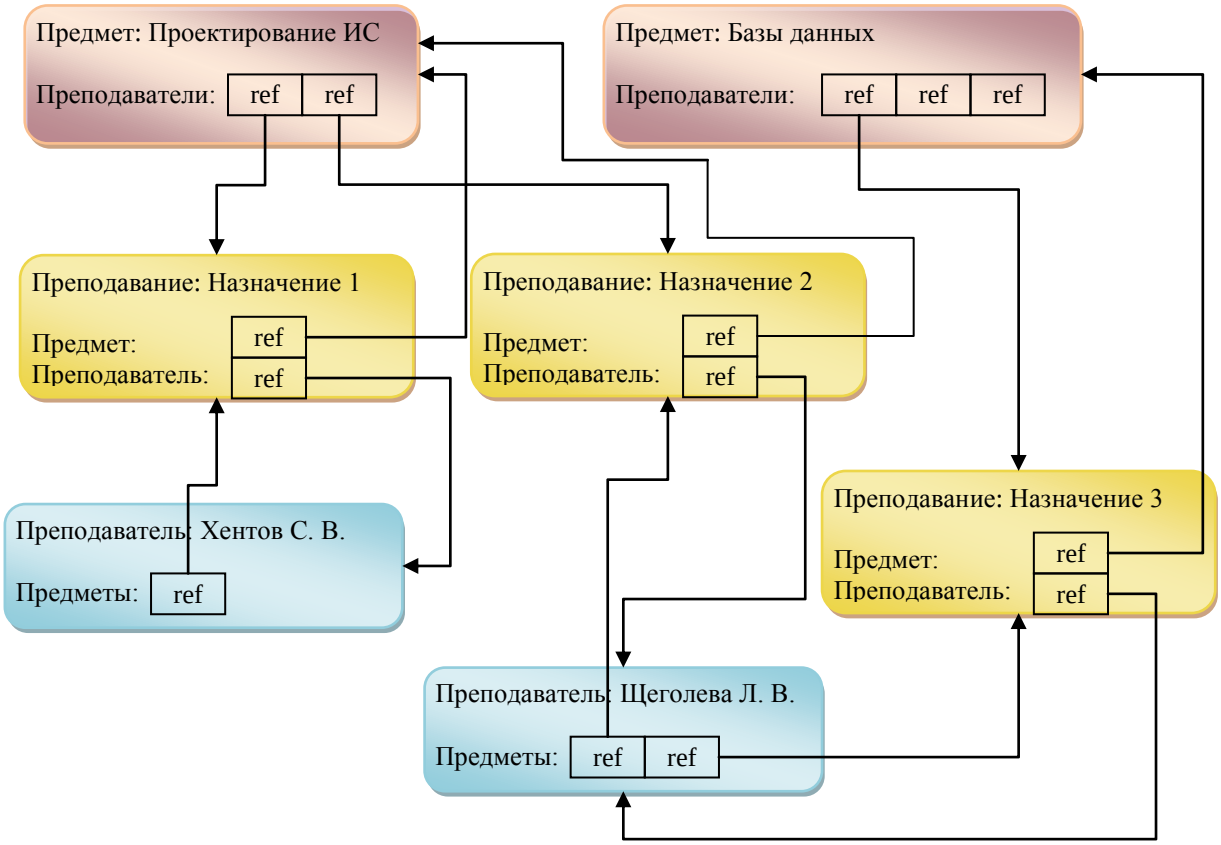


Рис. 6 – Связь «многие-ко-многим», реализованная с помощью нового класса

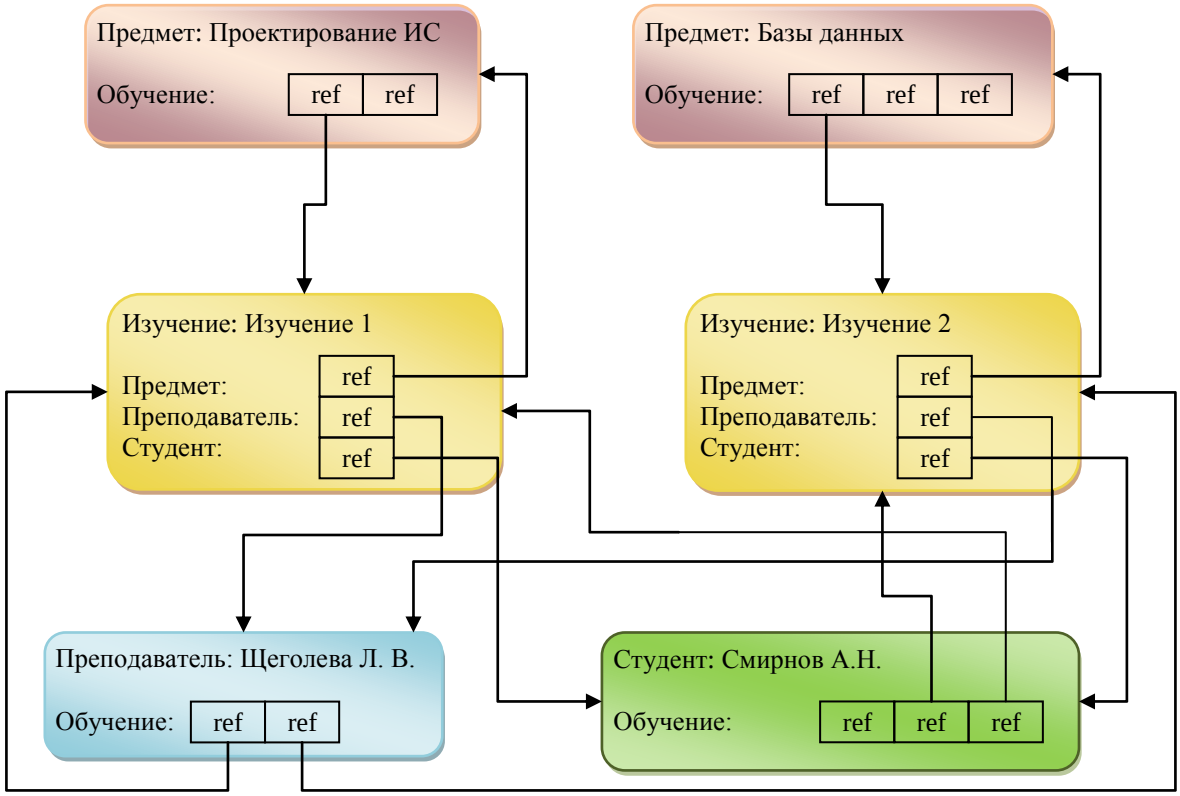


Рис. 7 – Связь между тремя классами

Связь между несколькими классами может быть реализована с помощью нового класса (рис. 7). Например, связь между классами «Предмет», «Преподаватель», «Студент» реализована с помощью класса «Изучение», в котором находятся ссылка на объект класса «Предмет», ссылка на объект класса «Преподаватель» и ссылка на объект класса «Студент». Объект класса «Предмет» содержит множество ссылок на объекты класса «Изучение», объект класса «Преподаватель» содержит множество ссылок на объекты класса «Изучение», объект класса «Студент» содержит множество ссылок на объекты класса «Изучение».

§ 3.3 *Операции*

Все операции оформляются в виде методов класса.

Все операции записаны на внутреннем языке СУБД.

Для каждого класса автоматически создаются два метода: конструктор и деструктор.

Проблемы:

Возникает вопрос, для какого класса определить метод, если операция поиска затрагивает объекты из нескольких классов? Например: Найти все дисциплины Общеобразовательного цикла с объемом часов превышающим 30.

При описании доступа к данным определяется путь доступа – через какие классы?

Оптимизация выполнения операций возлагается на разработчика, а не на СУБД.

Незапланированные запросы невозможны.

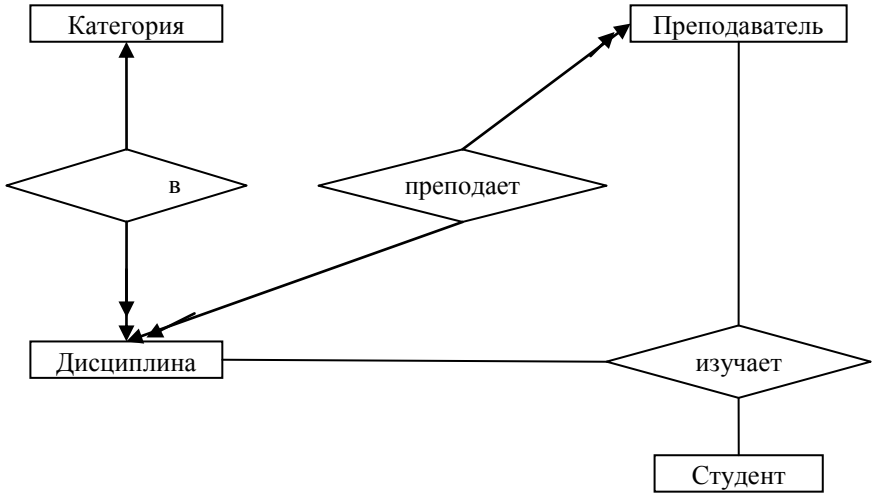
При удалении объектов необходимо каскадом удалять все ссылки на него в других объектах.

§ 3.4 *Ограничения целостности*

Проверка целостности данных выполняется в методах класса и, следовательно, возлагается на разработчика. Тем самым, суть ограничений остается скрытой в методах.

СУБД поддерживает только механизм ссылок на другие объекты.

§ 3.5 *Пример*



Каждый преподаватель может вести несколько дисциплин. Каждую дисциплину может преподавать несколько преподавателей. Студенты изучают дисциплинs у определенного преподавателя. Каждая дисциплина относится к одной из категорий (гуманитарная, общеобразовательная, специальная).

Инфологическая модель предметной области включает связь «один-ко-многим» между классами Категория и Дисциплина, связь «многие-ко-многим» между классами Преподаватель и Дисциплина, связь между тремя классами: Преподаватель, Дисциплина, Студент.

```
Class Type(
        Name Char(20),
        NumberOfSubjects Integer,
        Subjects  { Subject }
);

Class Subject(
        Name Char(40),
        Hour Integer,
        Teachers  { ref (Teacher) }
);
```

Дисциплины могут быть включены в класс Категория (как представлено в примере), а могут быть самостоятельными, тогда для свойства Subjects класса Type необходимо указать ссылку на объект класса Subject, а в классе Subject добавить свойство «Type ref (Type)», или выполнить только одно из двух предложенных изменений, но тогда связь будет односторонней.

```
Class Teacher(
        Name Char(50),
        Post Char(20),
        Year Integer,
        Subjects  { ref (Subject) },
        Students  { ref (Study) }
);

Class Student(
        Name Char(50),
        NumberOfSubjects Integer,
        Subjects  { ref (Study) }
);
```

Для связи между тремя классами в базе данных определяется отдельный объект:

```
Class Study(
        Student  ref (Student),
        Teacher  ref (Teacher),
        Subject  ref (Subject)
);
```

Для класса Subject также можно определить свойство со множеством ссылок на объекты класса Study.

§ 4 ODMG 3.0

Set<t> – неупорядоченная коллекция элементов без дубликатов,
Bag<t> – неупорядоченная коллекция элементов, в которой могут быть дубликаты
List<t> – упорядоченная коллекция элементов
Array<t> – динамическая коллекция элементов, которые могут быть адресованы по позиции
Dictionary<t,v> – неупорядоченная последовательность пар без дубликатов ключей

Атрибуты и связи.
Пример

```
class Professor {
...
relationship set<Course> teaches
inverse Course::is_taught_by;
...
}
```

and

```
class Course {
...
relationship Professor is_taught_by
inverse Professor::teaches;
...
}
```

СУБД является ответственной за поддержку связей. Если объект удаляется, то СУБД разрушает связь.

§ 5 Объектные СУБД

POET (Software Inc)
INGRES Intelligent Database (ASK Group)
ObjectStore (Object Design)
GemStore (Servio)
PostGres язык UniSQL

§ 6 Реализация ООБД в Oracle

§ 6.1 *Введение*

СУБД Oracle является объектно-реляционной. Это означает, что можно создавать типы данных, определенные пользователем (классы), которые можно хранить в таблицах или столбце реляционной таблицы.

§ 6.2 *Описание классов*

Объектные типы – это конструкция базы данных Oracle, управляемая через расширение DDL, которая объявляет структуру данных (атрибуты) и разрешенные операции (методы) над атрибутами.

Объект – это экземпляр объектного типа Oracle8. Объект – это место, где живут актуальные данные. Объекты могут храниться в таблицах, и, в таких случаях, они являются постоянными, либо они могут существовать только временно в переменных PL/SQL.

Атрибут – это структурная часть объекта Oracle, приблизительно похожая на столбец таблицы. Каждый атрибут может принадлежать одному типу данных, либо скалярному, как VARCHAR2 или INTEGER, либо составному, как объявленные пользователем вложенные таблицы или другие (вложенные) объекты. Скалярные атрибуты иногда называют простыми, а составные атрибуты могут упоминаться как сложные.

Метод – это процедура или функция, обычно выполняемая в PL/SQL, которая (как правило) производит операции над атрибутами объекта. Методы для объекта могут быть вызваны только в контексте конкретного объекта этого типа. Методы могут быть созданы и на языке C, тогда они вызываются как «внешние процедуры» Oracle. Существует специальный метод по умолчанию, поддерживаемый Oracle, называемый конструктор, который инициализирует объекты.

Прежде всего, нужно объявить объектный тип:

```
Create Type Project As Object(
project_no Number(2),
title      Varchar(35),
cost       Number(7,2),
Member Function get_per_cent (per_cent In Integer) Return Number);
```

Этот объектный тип имеет три атрибута и один метод. Необходимо также создать «тело объектного типа» для поддержки тела метода.

```
Create Type Body Project As
Member Function get_per_cent (per_cent In Integer) Return Number Is
  a Number;
Begin
  a:= per_cent *cost/100;
  Return a;
End;
End;
```

Объектный тип может также быть типом данных столбца в таблице. В этом случае говорят, что столбец содержит столбцовые объекты (column objects). Различные столбцы в таблице могут быть разных объектных типов.

```
Create Table manager (
surname Varchar2(50),
main_project project,
salary number);


Insert Into manager (surname, main_project) Values (‘Петров’, Project(3, ‘Минотавр’, 620));
Select m.main_project.title From manager m  Where m.surname=’ Петров’; –  returns ‘Минотавр’
Update manager m SET m.main_project.cost = 600;
```

Другой способ – создать объектную таблицу, содержащую строчные объекты (row objects), что означает, что каждая строка в таблице является экземпляром объекта.

```
Create Table projects Of project;
```

Объектная таблица имеет специальный скрытый столбец, который называется идентификатор объекта (object identifier) или OID. Этот идентификатор является уникальным не только среди всех таблиц, но и среди всей базы данных! Поскольку OID доступен для программиста, в других таблицах можно хранить ссылки, указывающие на строку-объект. Такие указатели называются ссылки (REF), они работают подобно внешним ключам.

После создания объектной таблицы можно создать объект (экземпляр объекта), используя «конструктор по умолчанию» (default constructor) – специальный метод, автоматически поддерживаемый Oracle при создании пользовательского типа данных. Этот конструктор имеет то же имя, что и объектный тип, и имеет по одному аргументу на каждый атрибут, объявленный в типе

```
Insert Into projects Values (project (2, 'Как украсть миллион', 0));
```

Объект может иметь тип данных «локальная переменная» (local variable). В этом случае, объявление и инициализация объектной переменной выполняется одним оператором. При инициализации используется автоматически доступный конструктор, который имеет то же имя, что и тип данных.

```
Declare
  my_project project := project (2, 'Как украсть миллион', 1);
  n number := my_project.get_per_cent(10);
```

Сложные объекты.

```
Create Type worker As Object(
nom Number(5),
surname Varchar(35),
salary Number(7, 2));

Create Type project As Object(
project_no Number(2),
title      Varchar(35),
cost       Number(7, 2),
boss       worker);
```

В этом случае объект project будет включать в себя объект worker (встроенный объект). Для того чтобы на объект worker могли ссылаться другие объекты, поле необходимо сделать ссылкой.

```
Create Type project As Object(
project_no Number(2),
title      Varchar(35),
cost       Number(7, 2),
boss       Ref   worker);
```

Пусть отдел работает над несколькими проектами. Определим тип объектная таблица для хранения списка проектов.

```
Create Type project_list As Table of project;
```

Объекты department содержат список проектов, выполняемых в отделе.

```
Create Type department As Object(
department_no Number(2),
name      Varchar(50),
dep_projects  project_list);

Create Table departments Of  department;
```

Операция добавления

```
Insert Into departments
    Values (department (101, 'Конструкторский отдел',
      project_list (project (1, 'Сердолик', 5),
                    project (2, 'Минотавр', 51),
                    project (3, 'Ночной дозор', 23),
                    project (4, 'Как украсть миллион', 0))));

Declare
  new_list project_list :=
    project_list(project (1, 'Сердолик', 5),
                 project (2, 'Минотавр', 51),
                 project (3, 'Ночной дозор', 23),
                 project (4, 'Как украсть миллион', 0));

Update departments Set dep_projects = new_list Where department_no = 1;

Select dep_projects Into new_list From departments Where name = ‘Конструкторский отдел’;
```

§ 7 Реализация ООБД в Cache 5

Хранимые классы наследуются от системного класса %Persistent.
OID – идентификатор экземпляра класса
Элементы класса:

- имя
- ключевые слова
- свойства
- методы
- параметры
- запросы
- индексы

Свойства:

- константы
- ссылки на объекты
- коллекции
  - массивы
  - списки
- отношения

```
class User.Persion Extends (%Persistent)
{
Property Surname As %String [Required];
Property Name As %String;
Property DOB As %Date (Format=4);
Property Age As %Integer [Calculated, SqlComputeCode = {Set {AgeInYear}=$Horolog –
DOB}\365.2425}, SqlComputed, SqlFieldName=AgeInYear];
Method AgeGet() As %Integer
{
Quit $Horolog – ..DOB\365.2425
}
Index SurnameIndex On Surname;
Query Wybor() As %SQLQuery
{
Select %ID, Surname, Name, DOB From Person Order by Surname
}
}
```

$Horolog – системная переменная, содержит дату и время
Методы класса %Persistent:
%Open() – находит объект и создает в памяти копию
%OpenId()
%Save() – обновление в БД
%Delete() – удаляет объект из базы данных
%DeleteId()

```
Set pers=##class(User.Person).%New()
Set pers.Name=”Василий”
Set pers.Surname=”Иванов”
Set sc=pers.%Save()
Write pers.%Id()
Set pers=””

Set rs=##class(%ResultSet).%New(“User.Person:Wybor”)
Set sc=rs.Execute()
Set sc=rs.Next()
Write rs.Data(“Surname”)
Set ID=rs.Data(“Id”)
Set pers=##class(User.Person).%OpenId(ID)
Write pers.Surname
Set pers.Surname=”Петров”
Set sc=pers.%Save()
```

Глава 11. Технологии доступа к данным

§ 1 Страничная организация памяти

Время поиска и доступа к диску равняются приблизительно 10 мс, что на 4–5 порядков выше по сравнению с оперативной памятью. Аналогичная закономерность справедлива для скорости передачи данных с диска и из памяти. Основной целью повышения производительности является минимизация числа дисковых операций ввода-вывода данных.

SQL описывает только информацию, над которой производятся действия, но не описывает, как выполнить эти действия. СУБД сама решает этот вопрос. Техника выполнения команд включает: структуру хранения, индексы и способ доступа.

Не существует идеальной структуры хранения, которая была бы оптимальна для любых задач.

Особенность реляционной модели заключается в том, что результат запроса не зависит от структуры хранения, и пользователю не нужно знать, как организованы данные на диске. Но структура хранения, индексы и способы доступа влияют на скорость выполнения запросов, которая может различаться от долей секунды до нескольких часов.

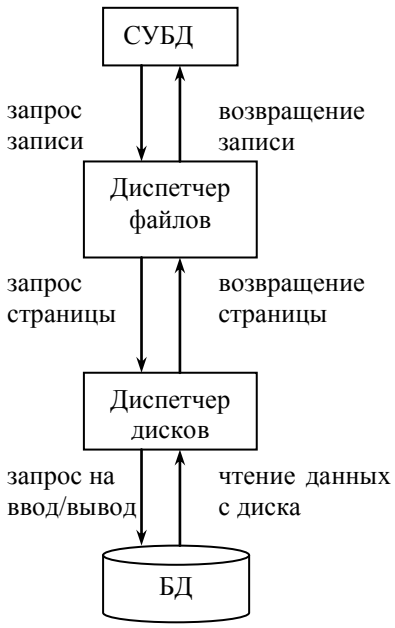
Поэтому в функции администратора входит настройка этих характеристик, которая определяется на основе содержания SQL запросов и частоты выполнения запросов.

Замечания

- Выполнение операций чтения/записи в оперативной памяти происходит на несколько порядков быстрее, чем чтение/запись на диск. Следовательно, необходимо оптимизировать поток данных между диском и оперативной памятью.
- Если записи упорядочены физически, то доступ к ним будет быстрее, так как не нужно тратить время на поиск следующей записи.
- Обмен данными между оперативной памятью (буфером) и диском осуществляется блоками, которые называются страницами, чаще всего страница включает последовательность соседних секторов дорожки. Обычно размер страницы составляет 4096 байт.

§ 1.1 Доступ к базе данных

С точки зрения СУБД база данных выглядит как набор записей, которые могут просматриваться с помощью **диспетчера файлов**. С точки зрения диспетчера файлов база данных выглядит как **набор страниц**, которые могут просматриваться с помощью **диспетчера дисков**. Диспетчер файлов рассматривает диск как набор страниц, каждый набор имеет свой уникальный идентификационный номер. Каждая страница внутри набора обладает своим идентификационным номером страницы, уникальным только внутри набора. Физические адреса наборов страниц известны диспетчеру дисков, и не известны диспетчеру файлов. (Один набор пустых страниц).



Диспетчер дисков выполняет следующие операции:

- создать набор страниц s,
- удалить набор страниц s,
- извлечь (прочитать) страницу p из набора страниц s,
- заменить (записать) страницу p из набора страниц s,
- добавить новую страницу в набор страниц s (перенос страницы из набора пустых страниц),
- удалить страницу p из набора страниц s (возвращение страницы в набор пустых страниц).

Набор страниц может содержать один или несколько файлов (набор однотипных записей). Каждый файл имеет идентификационный номер, уникальный в данном наборе страниц. Каждая запись файла имеет идентификационный номер записи, уникальный в пределах файла.

Диспетчер файлов выполняет следующие операции:

- извлечь (прочитать) запись r из файла f,
- заменить (записать) запись r в файле f,
- добавить новую запись в файл f,
- удалить запись r из файла f,
- создать новый файл,
- удалить файл f.

Доступ к данным осуществляется следующим образом:

1. В СУБД определяется искомая запись и файл, в котором она находится.
2. Диспетчер файлов определяет страницу, на которой находится запись.
3. Диспетчер дисков определяет физическое расположение страницы и считывает ее в оперативную память.

Основной функцией диспетчера дисков является скрытие от диспетчера файлов всех деталей физических дисковых операций ввода-вывода и замена их логическими операциями ввода-вывода страниц.

Каждая страница содержит заголовок страницы с информацией о физическом дисковом адресе страницы, которая логически следует за данной страницей. Страница 0 содержит каталог наборов страниц. На ней перечислены все наборы страниц вместе с указателями на первые страницы каждого из наборов. Этот механизм позволяет быстро находить логически связанные страницы.

Чтобы сохранять логически связанные страницы в физически близких местах диска диспетчер диска обычно размещает или удаляет страницы в наборах не по одной, а целыми группами физически связанных страниц, или блоками (экстентами – непрерывная область на диске).

§ 1.2 *КЭШ*

СУБД записывает страницы с данными в специальный буфер, который называется КЭШ. Когда приложение выполняет транзакцию СУБД сначала записывает требуемые данные в КЭШ, а затем в память (буфер) приложения. Страницы обычно остаются в КЭШ исходя из предположения о высокой вероятности того, что они скоро снова потребуются или другое приложение запросит эти же самые данные. Это позволит не обращаться к диску, на что требуется гораздо больше времени по сравнению с обращением к оперативной памяти. Например, индексы очень часто используются при выполнении разных запросов разными приложениями. В КЭШ хранятся не только данные из таблиц, но так же индексы, хранимые процедуры, планы выполнения SQL запросов и другие данные, необходимые СУБД.

При выполнении обновлений данных измененные данные сначала записываются в КЭШ, а потом переносятся на диск. Измененные страницы получают метку: «грязные», а неизмененные – «чистые». С течением времени КЭШ заполняется полностью, тогда новые страницы записываются поверх старых. В этом случае если это была «чистая» страница, то она просто затирается, а если «грязная», то ее сначала необходимо записать на диск, а потом можно стереть. Выбор станицы, которую следует стереть, производится на основе специального алгоритма СУБД, обычно выбираются страницы, к которым обращение происходило очень давно. Более гибкий алгоритм может оценивать, с какой целью было обращение к странице, и, какова вероятность в ближайшем будущем снова обратиться к этим данным, и выбирает наименее вероятную страницу.

Считается хорошим, если 90% данных, запрашиваемых приложением, читаются из КЭШ, а не с диска. Поэтому размер КЭШ устанавливается большим и может достигать гигабайта.

§ 2 Индексирование

Рассмотрим операцию поиска записи по названию города. Для того чтобы найти все записи с городом «Москва» необходимо просмотреть все записи таблицы. Если бы записи были упорядочены по названию городов, то поиск можно было бы произвести быстрее, используя алгоритм бинарного поиска. Для физического упорядочивания записей в таблице используется кластеризация.

§ 2.1 *Кластерный индекс*

Технология **кластеризации** данных основана на принципе как можно более близкого физического размещения на диске логически связанных между собой и часто используемых данных. Это является важным условие высокой производительности.

Кластеризация может быть:

- **внутрифайловой** – доступ к поставщику по его номеру, тогда желательно упорядочить записи по номерам,
- **межфайловой** – требуется информация о поставках и их поставщиках, тогда желательно расположить запись о поставке и запись о поставщике рядом.

Организацией кластеризации занимается администратор базы данных.

Недостатком кластеризации является тот факт, что ее можно задать только для одного атрибута. Если в запросе указаны условия, касающиеся другого атрибута, то кластеризация данных не поможет. Выходом из этой ситуации является создание индексов!

§ 2.2 *Структура индекса*

Индексный файл (индекс) – это таблица особого типа, в которой каждая запись состоит из двух значений, а именно данных и RID-указателя. Данные формируются из индексируемого поля индексируемой таблицы, а RID указатель содержит физический адрес записи индексируемой таблицы.

Индекс

Город	RID
Москва	●
Псков	●
Псков	●
С.-Пб.	●
С.-Пб.	●

Индексируемая таблица

Номер	Имя	Статус	Город
1	Кузнецов	20	Псков
2	Иванов	10	С.-Пб.
5	Чернов	30	С.-Пб.
7	Букин	30	Псков
10	Тихий	30	Москва

Основным **преимуществом** использования индексов является значительное ускорение процесса выборки или извлечения данных, а основным **недостатком** – замедление процесса обновления данных. При выборе некоторого поля для индексирования необходимо выяснить, что более важно для данной базы данных и СУБД: скорость извлечения данных или скорость обновления?

Индексирование может быть организовано на основе:

- ключевого поля – **первичный** индекс (иногда по первичному индексу проводят кластеризацию и тогда индекс называют главным (main)),
- любого поля – **вторичный** индекс,
- другого ключа – **уникальный** индекс.

Индексы можно использовать двумя способами:

- для последовательного доступа к индексированной таблице, т.е. записи должны быть упорядочены по индексному полю,
- для прямого доступа, т.е. к отдельным записям на основе заданного значения индексного поля.

Можно использовать индекс для проверки на наличие записи в таблице.

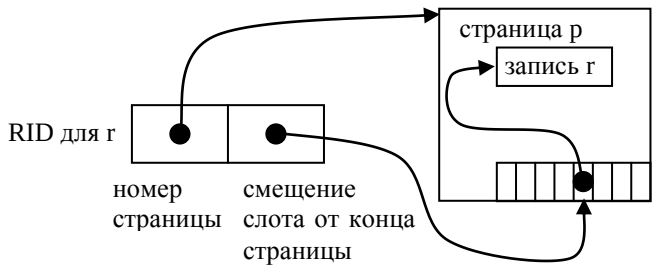
Индекс обычно занимает меньше места, чем индексируемая таблица.

Таблица может иметь несколько индексов, которые могут использоваться раздельно и совместно. Например, если таблица поставщиков имеет индекс по полю город и по полю статус и, требуется найти поставщиков из города Москва со статусом 20, то по первому индексу будут найдены все поставщики из Москвы, по второму – все поставщики со статусом 20, а пересечение этих двух множеств и будет ответом на запрос.

Индексирование можно провести на основе комбинации двух или более полей. Индекс на основе комбинации полей F_1, F_2,…,F_n может использоваться для индексирования по полю F_1, либо для индексирования на основе комбинации F_1, F_2, либо комбинации F_1, F_2, F_3 и т.д.

§ 2.3 *RID указатель*

Каждая запись идентифицируется с помощью **идентификационного номера записи** (record ID – RID). Идентификационный номер записи состоит из двух частей – номера страницы r, на которой данная запись находится, и байта смещения слота от конца страницы p, который содержит байт смещения записи r от начала страницы p. Эта схема является компромиссом между быстротой непосредственной адресации и гибкостью косвенной адресации. Записи внутри страницы могут сдвигаться вверх или вниз без изменения идентификационных номеров записей. Если известен идентификационный номер записей, доступ к данной записи осуществляется очень быстро.



Слот – это 4–байтовое слово, 2 байта соответствуют смещению записи на странице и 2 байта – длина записи.

Особенности:

- Для любого файла можно осуществить последовательный доступ ко всем записям, что означает доступ согласно последовательности записей внутри страницы и последовательности страниц внутри набора страниц (обычно это порядок возрастания идентификационных номеров записей). Такая последовательность называется физической, хотя она не обязательно соответствует физическому расположению данных на диске.
- Запись может содержать дополнительную или сопроводительную информацию. Такая информация располагается перед записью и называется **приставкой записи**. Например,

идентификационный номер файла (если на одной странице расположены записи из нескольких файлов), длину записи (для записей с переменной длиной), флаг удаления, указатель (если записи связаны в цепочку) и др.

§ 2.4 *Неплотный индекс*

Основной целью использования индекса является ускорение процесса извлечения данных, а точнее, уменьшение числа дисковых операций ввода-вывода, необходимых для извлечения требуемой записи. В основном это достигается благодаря использованию указателей записей. На самом деле для индексации достаточно указателей страниц. Конечно, для извлечения записи внутри страницы придется выполнять еще одну операцию поиска, но она уже будет выполняться в оперативной памяти.

Эту идею можно развить дальше. Если в файле поставщиков выполнена кластеризация по полю номер поставщика, т.е. физическая последовательность соответствует логической последовательности, заданной на основе поля номер, и по этому же полю выполняется индексирование. Тогда нет необходимости хранить в индексе указатели для каждой записи, достаточно хранить указатель страницы, состоящий из максимального номера поставщика для данной страницы и соответствующего номера страницы. Для извлечения записи сначала проводится поиск индекса для записи с номером, большим или равным искомому значению, в оперативную память помещается соответствующая страница, которая просматривается уже в оперативной памяти для получения заданной записи.

Если неплотный индекс задан для неуникального поля, то может потребоваться чтение нескольких страниц для получения записей, удовлетворяющих условию (возможен вариант, когда для каждой страницы с данными составляется список значений индексируемого поля, находящихся на данной странице, такой индеек будет больше неплотного индекса, но меньше плотного).

Индекс такой структуры называется **неплотным.**

- Преимуществом таких индексов является малый размер по сравнению с плотными индексами.
- С помощью неплотного индекса нельзя выполнить проверку наличия значения.
- В файле может быть не более одного неплотного индекса, на основе физической последовательности, заданной в файле (кластеризации).

§ 2.5 *Древовидный индекс*

Если файл очень большой, то и его индексный файл будет также большим, и на его просмотр потребуется много времени. Поэтому, чтобы избежать обязательного просмотра всего содержимого индексированного файла согласно его физической последовательности, можно рассмотреть индексный файл как обычный файл и построить для него индексный файл. Эту операцию можно повторить (обычно трижды). Индекс каждого следующего уровня будет неплотным по сравнению с предыдущим. (Технология ISAM – Index-Sequential Access Method)

Пример

Пусть, файл данных содержит 100,000 записей, каждая запись занимает 128 байт. Одна страница составляет 1024 байта (1 килобайт). Тогда, для хранения файла данных потребуется 12,500 страниц. Для получения записи в таком файле может потребоваться 12,500 операций чтения диска. Построим для файла данных индексный файл. Пусть, индексная запись занимает 16 байт. Тогда, для индексного файла потребуется 1563 страницы. В этом случае для доступа к записи потребуется максимум 1563 операции чтения диска. Применив бинарный поиск, потребуется всего $\log_2 1563 + 1 = 12$ операций чтения диска (одна для чтения непосредственно данных по найденному адресу). Если построить неплотный индекс, то количество индексных записей составит 12,500, и тогда потребуется всего 196 страниц для хранения записей неплотного индексного файла. Количество операций чтения диска в этом случае составит $\log_2 196 + 1 = 9$. Можно построить индексный файл для неплотного индекса. он будет содержать всего 4 страницы, а индекс следующего уровня – 1 страницу. В этом случае для доступа к любой записи потребуется ровно 4 операции чтения. Три – для чтения индексных страниц, и одна – для чтения страницы данных.

В древовидном индексе количество обращений к диску для поиска любой записи данных всегда одинаково и равно количеству уровней в построенном дереве, в отличие от обычного индекса, где известна оценка сверху (бинарный поиск).

Применение индексирования усложняет выполнение операций обновления. Требуется вставить новую индексную запись в конкретное место. Для этого на странице, содержащей индексные записи, оставляют свободную область. После определения страницы, на которую должен быть занесен индекс, она копируется в оперативную память, где модифицируется путем вставки в нужное место новой записи и, измененная записывается обратно на диск.

§ 2.6 *Bitmap индексы*

Используется для атрибутов, домены которых содержат небольшое количество значений. Например, пол, курящий, возраст (10, 20, 30, 40, 50, 60, 70, 80). Индекс содержит для каждой записи вектор из битов, количество которых равно количеству значений в домене атрибута. Таким образом, для каждого

значения из домена создается битовый вектор, размерность которого равна количеству записей. В этом векторе, если на i-ом месте стоит 1, то соответствующая запись имеет это значение.

§ 2.7 *Индексы для соединения*

Индекс для соединения содержит пары RID указателей для соответствующих пар записей из двух разных таблиц, для которых выполняется условие на соединение.

§ 2.8 *Хэширование*

Хэширование – технология быстрого прямого доступа к записи на основе заданного значения некоторого поля. В файле может быть только одно хэш–поле.

Недостатки хэширования:

- Практически всегда физическая последовательность записей в хэшированном файле не имеет ничего общего с заданной в нем логической последовательностью.
- Возможность возникновения коллизий – ситуаций, когда две или более различных записей имеют одинаковые адреса. В этом случае значением хэш-функции является адрес точки привязки – начального адреса цепочки указателей, связывающей вместе все записи или страницы с записями, которые вызывают коллизию.

§ 2.9 *Индексирование в InterBase*

```
CREATE [UNIQUE] [{ASC[ENDING] | DESC[ENDING]}] INDEX <имя> ON <таблица> (
<столбец>, ...);
```

Перед тем как выполнить большое количество операций обновления лучше всего отключить индекс, а после завершения блока операций снова включить индекс, это позволит выполнить операции быстрее.

```
ALTER INDEX <имя> {ACTIVE | INACTIVE};
```

Команда удаления индекса из базы данных:

```
DROP INDEX <имя>;
```

§ 2.10 *Индексирование в MS SQL Server*

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX <имя> ON <таблица> (
<столбец> [ ASC | DESC ], ...)
```

Пример

```
create index idx_d_city on d(city)
```

§ 2.11 *Индексирование в MySQL*

В MySQL таблица может иметь до 16 ключей, каждый из которых может иметь до 15 полей. Максимальная поддерживаемая длина ключа 120 байт. Можно увеличить длину ключа, изменяя N_MAX_KEY_LENGTH в файле nisam.h и перекомпилировав пакет.

Ключи могут иметь имена. В случае первичного ключа имя будет всегда PRIMARY. Если имя ключа не задано в процессе создания таблицы, то заданное по умолчанию имя ключа – первое имя столбца с факультативным суффиксом (_2, _3, и т. д.) чтобы сделать это имя уникальным. Имя ключа может использоваться с командой ALTER TABLE, чтобы удалить ключ.

При создании ключа можно факультативно определить, что только первые N символов поля будут использоваться. Например, чтобы создать уникальный ключ на поле, в котором только первые 40 символов уникальны, можно сделать следующее:

```
CREATE TABLE SomeTable (composite CHAR(200), INDEX comp_idx(composite(40)));
```

Хорошая идея – использовать эту опцию на неуникальных полях, поскольку эта мера значительно уменьшит размер индекса, а снижение производительности будет очень небольшим.

Можно иметь один первичный ключ на таблицу. Если поле определено, как поле первичного ключа, то генерируется индекс.

Ключи с несколькими полями следует использовать для оптимизации узкоспецифических запросов. То есть, все поля в предложении WHERE запроса должны появляться в многопольном ключе.

Поскольку MySQL использует B-Tree не нужно объявлять ключи, которые являются префиксами других ключей. Оптимизатор найдет любой пригодный для использования префикс ключа и использует его, чтобы выполнить поиск. Например, если объявлен следующий ключ:

INDEX (first, second, third, fourth)

то неявно созданы следующие ключи:

(first, second, third)

(first, second)

(first)

Ключи должны быть созданы во время создания таблицы или изменения таблицы с использованием команды ALTER TABLE.

Пример

```
Alter Table d Add Index idx_d_city (city)
```

Глава 12. Целостность базы данных

Современные СУБД обладают специальными механизмами управления данными для обеспечения целостности базы данных.

Целостность означает точность и корректность данных, хранящихся в базе данных, которые вытекают из ограничений предметной области и могут быть выражены в виде правил, которые пользователи и прикладные программы не должны нарушать, или событий, которые необходимо отслеживать и при их наступлении выполнить некоторые действия.

Примеры:

- Вес детали должен быть положительным числом (ограничение домена).
- Поставщик имеет один адрес (функциональное ограничение).
- Поставщики из Москвы должны обладать статусом не ниже 30 (ограничение отношения, одной записи).
- Количество деталей некоторого типа должно быть не менее 1000 вне зависимости от поставщика (ограничение отношения, вся таблица).
- Должно быть, по крайней мере, две детали красного цвета.
- Поставщики со статусом меньше 20 не могут поставлять детали в количестве более 500 (ограничение базы данных, вовлекает несколько отношений).

На ранних этапах развития баз данных знания о предметной области включались в прикладные программы. В современных СУБД эта информация хранится вместе с базой данных обычно в системном каталоге. Она используется подсистемой целостности СУБД, которая ответственна за отслеживанием выполняемых пользователем операций и гарантирует выполнение только тех, которые не нарушают известных системе ограничений.

Аспекты целостности необходимо учитывать как при проектировании базы данных, так и во время ее использования. Это означает, что при проектировании необходимо задать не только структуру данных, но и ограничения.

§ 1 Каталог

Каталог (или **словарь данных**) содержит данные о данных (метаданные). Каталог содержит детальную информацию (иногда называемую дескрипторами), касающуюся различных объектов, которые имеют значение для самой системы. Примерами таких объектов могут служить таблицы, индексы, правила и т.д.

Прекрасным свойством реляционных систем является то, что каталог сам состоит из таблиц. В результате пользователь может обращаться к каталогу так же, как к своим данным.

Например, в каталоге обычно есть таблицы с описанием известных системе таблиц (название, количество столбцов, количество записей, …) и столбцов этих таблиц (название, тип данных, размер, индекс, …).

§ 2 Процедуры

Процедура – это программа, написанная на языке манипулирования данными СУБД и хранящаяся вместе с метаданными в базе данных. Называются хранимыми, присоединенными. Процедуры вызываются прикладной программой и исполняются на сервере. Они могут содержать параметры и возвращать значение, в том числе и сообщение об ошибке.

Преимущества:

- централизованный контроль доступа к данным;
- прикладные программы могут вызывать процедуру, что сокращает время написания программ, при модификации процедуры, все вызывающие ее программы получат новый код, оптимизация кода;
- снижает трафик в сети в системах «клиент-сервер» за счет передачи только имени процедуры и ее параметров вместо обмена данными, а процедура выполняется на сервере;
- сокрытие от пользователя многих особенностей конкретного устройства базы данных, что обеспечивает большую независимость данных;
- большая безопасность данных, пользователь может иметь право вызывать процедуру, но не управлять данными, которые вызываются этой процедурой;
- Недостаток: отсутствие стандартов в реализации хранимых процедур.

```
CREATE PROCEDURE <имя процедуры> [ ({ in | out | inout } <параметр> <тип>, ...)]
BEGIN
[DECLARE  <имя переменной> <тип>;  ...]
  <операторы>
END;
```

Вызов процедуры:

```
CALL  <имя процедуры> [ ( <значение параметра> , ...)]
```

§ 2.1 *Процедуры в InterBase*

Создание

```
CREATE PROCEDURE <имя> [ (<параметр> <тип>, ...)]
[RETURNS (<параметр> <тип>, ...)]
AS
[DECLARE VARIABLE <переменная> <тип>;
[DECLARE VARIABLE <переменная> <тип>; ...]
BEGIN
  <операторы>
END
```

Вызов

Чтобы вызвать процедуру нужно выполнить команду:

```
EXECUTE PROCEDURE <имя> [<параметр>, ...] [RETURNS <параметр>, ...];
```

Пример

```
CREATE PROCEDURE count_d (id integer)
RETURNS (n integer)
AS
DECLARE VARIABLE i INTEGER;
BEGIN
SELECT SUM(number)
FROM dp
WHERE id_d=:id
INTO :i
n=i
END
```

Вызов

```
EXECUTE PROCEDURE count_d 2 RETURNS number
```

§ 3 Триггеры

Триггер (правило) присоединяется к таблице или представлению и автоматически вызывается, когда выполняются операции обновления над таблицей (добавление, удаление, изменение). Триггер никогда прямо не вызывается, он срабатывает, когда выполняется операция Insert, Delete, Update.

Преимущества использования триггеров:

- автоматический контроль правильности вводимых значений;
- переносит операции по проверке значений из прикладной программы в СУБД и делает их доступными для всех программ;
- автоматическое уведомление об изменениях в базе данных с вызовом события.
- автоматически преобразует данные при вводе в таблицу

Триггер вызывается для операций:

- INSERT,
- UPDATE,
- DELETE,
- ROLLBACK,
- SIGNAL.

Триггер имеет условие, при истинности которого выполняется триггер.

Существует две стратегии вызова триггера:

- немедленная – триггер вызывается сразу при наступлении события,
- отложенная – триггер вызывается после фиксации транзакции.

При немедленном вызове возможны следующие варианты:

- вызов триггера до выполнения действий над таблицей,

вызов триггера после выполнения действий над таблицей,
вызов триггера вместо выполнения действий над таблицей (не поддерживается стандартом SQL:1999; используется для редактирования представлений).

События рассматриваются на двух уровнях:
выполнение операции над каждой записью таблицы (вызов триггера для каждой записи), в теле триггера обеспечивается доступ к значениям записи до и после выполнения операции;
выполнение операции над таблицей целиком (вызов триггера для всей таблицы один раз; в случае, когда ни одна запись в таблице не изменилась при выполнении операции, триггер все равно вызывается один раз для всей таблицы), в теле триггера обеспечивается доступ к копии таблицы до и после выполнения операции.

Можно определить несколько триггеров на одно и тоже событие, которые будут вызываться по очереди. СУБД либо устанавливает порядок выполнения триггеров, либо выбирает случайным образом. Скорее первое. Существует две стратегии проверки условия:
проверяется условие для триггера, и он выполняется, затем осуществляется переход к следующему триггеру,
проверяются сразу все условия, затем выполняются только те триггеры, для которых условие истинным, даже, если оно стало ложным после выполнения предыдущего триггера в очереди.

В теле триггера могут быть операторы обновления данных в той же самой таблице или в других таблицах, которые повлекут вызовы других триггеров, а те в свою очередь других триггеров и т. д. В этом случае может произойти зацикливание. Эту ситуацию СУБД предсказать не может, поэтому вся ответственность ложится на разработчика базы данных, который должен промоделировать возможные варианты последовательности вызовов триггеров. Можно составить граф, где вершинами будут триггеры, а дуги – это вызовы одного триггера из другого. Если граф не содержит циклов, то зацикливания не будет.

```
CREATE TRIGGER <имя триггера>
 {BEFORE | AFTER} {DELETE | INSERT | UPDATE [ OF <список столбцов>]}
 ON <имя таблицы>
  [REFERENCING [OLD AS <имя переменной>]
               [NEW AS <имя переменной>]
               [OLD TABLE AS <имя переменной>]
               [NEW TABLE AS <имя переменной>] ]
 [FOR EACH { ROW | STATEMENT } ]
 [WHEN (<условие>)]
  <операторы>
```

FOR EACH ROW – возов для каждой записи,
FOR EACH STATEMENT – вызов для всей таблицы
OLD, NEW – переменные для хранения значений записи до и после выполнения операции для режима FOR EACH ROW (OLD – для операций удаления и изменения, NEW – для операций добавления и изменения).
OLD TABLE, NEW TABLE – переменные для хранения записей, подвергшихся изменению в результате выполнения операции (<отношение после выполнения операции> = (<отношение до выполнения операции> – OLD) ∪ NEW).

Пример

```
CREATE TRIGGER RoomCapacityCheck
 BEFORE INSERT ON Transcript
  REFERENCING NEW AS N
 FOR EACH ROW
 WHEN
  (( Select Count(T.StudId) From Transcript T
    Where T.CRSCode = N.CrsCode AND T.Semester = N.Semester)
   >=
  ( Select L.Limit From CrsLimits L
    Where L.CrsCode = N.CrsCode AND L.Semester = N.Semester))
 RollBack
```

Триггер вызывается до выполнения операции добавления одной записи, и если условие для триггера истинно, то вызывается откат (превышен лимит количества курсов, на которые может записаться студент, поэтому запись на новый курс не выполняется – откат).

```
CREATE TRIGGER LimitSalaryRaise
  AFTER UPDATE OF Salary ON Employee
   REFERENCING OLD AS O
             NEW AS N
  FOR EACH ROW
  WHEN
    (N.Salary – O.Salary > 0.5 * O.Salary)
  Update Employee
  Set Salary = 1.05 * O.Salary
  Where Id = O.Id
```

Триггер вызывается после выполнения операции изменения одной записи, и если зарплата оказалась увеличенной более чем на 5%, то это исправляется, и зарплата увеличивается ровно на 5% от предыдущей зарплаты.

```
CREATE TRIGGER RacordNewAverage
  AFTER UPDATE OF Salary ON Employee
  FOR EACH STATEMENT
  Insert into Log Values (Current_date, (Select avg(Salary) From Employee))
```

Триггер вызывается после выполнения операции изменения для всей таблицы в целом, при этом в другую таблицу записывается текущая дата и текущая средняя зарплата, которая рассчитывается по данным из измененной таблицы.

§ 3.1 *Триггеры в InterBase*

```
CREATE TRIGGER <имя> FOR <таблица>
[{ACTIVE | INACTIVE}]
{BEFORE | AFTER} {DELETE | INSERT | UPDATE}
[POSITION <номер>]
AS
[DECLARE VARIABLE <переменная> <тип>;
[DECLARE VARIABLE <переменная> <тип>; ...]
BEGIN
  <операторы>
END

ALTER TRIGGER <имя> {ACTIVE | INACTIVE}

DROP TRIGGER <имя>
```

Пример:

```
CREATE TRIGGER del_dp FOR p
ACTIVE BEFORE DELETE
AS
DECLARE VARIABLE id integer
BEGIN
  id = OLD.id_p
  DELETE from dp WHERE id_p=:id
END

CREATE TRIGGER det1 FOR dp ACTIVE AFTER INSERT
AS
BEGIN
  UPDATE d
  SET TOTQTY = TOTQTY + NEW.number
  WHERE id_d = NEW.id_d
END
```

```
CREATE TRIGGER det2 FOR dp ACTIVE AFTER UPDATE
AS
DECLARE VARIABLE h integer
BEGIN
  EXECUTE PROCEDURE count_d OLD.id_d RETURNS :h
  UPDATE d
  SET TOTQTY = :h
  WHERE id_d = NEW.id_d
END

CREATE GENERATOR D_GEN
CREATE TRIGGER SET_d_NO FOR d
BEFORE INSERT
AS
BEGIN
  NEW.id_d = GEN_ID(D_GEN, 1);
END !!
```

§ 4 События

Иногда требуется выполнить некоторые действия, когда база данных перейдет в определенное состояние, например, когда количество некоторых деталей станет меньше 1000. Для того чтобы вовремя выполнить действия, нужно отследить это событие. Прикладная программа может опрашивать базу данных через определенные промежутки времени, например выполняя следующий SQL–оператор:

```
SELECT SUM(NUMBER) FROM dp WHERE id_d=1
```

В СУБД предусмотрен специальный механизм, называющийся **событием**. События это сообщения, которые посылаются триггером или хранимой процедурой менеджеру событий, объявляющее о выполнении некоторого условия или действий (посылаются только после завершения транзакции). Менеджер событий вносит полученное событие в список событий. Менеджер событий также имеет список приложений, заинтересованных в отслеживании некоторых событий.

Механизм событий состоит из трех частей:

- триггер или процедура посылают сообщение менеджеру событий;
- менеджер событий помещает сообщение в очередь и уведомляет приложения о наступлении события;
- приложение, которое заинтересовано в событии и ждет его наступления.

Для отслеживания событий приложения должны:

- зарегистрироваться у менеджера событий;
- ожидать уведомления о наступлении события;
- идентифицировать, какое событие произошло (если ожидается несколько событий).

Вызов события в InterBase

```
CREATE TRIGGER order FOR dp ACTIVE AFTER DELETE
AS
DECLARE VARIABLE h integer
BEGIN
  EXECUTE PROCEDURE count_d OLD.id_d RETURNS :h
  If (h < 1000) then
    Post_Event(“send_order”);
  End If
END
```

Для получения события приложение должно зарегистрироваться.

```
EVENT INIT <имя регистрации> (<имя события>,…);
EVENT INIT Order (“send_order”);
```

После команды:

```
EVENT WAIT <имя регистрации>
```

приложение переходит в режим ожидания события. Только в этом режиме приложение может получать уведомления. После получения уведомления приложение продолжает работу до следующего оператора WAIT.

События в MS SQL Server

```
CREATE EVENT NOTIFICATION event_notification_name ON { SERVER | DATABASE | QUEUE
queue_name } [ WITH FAN_IN ]FOR { event_type | event_group } [ ,...n ]TO SERVICE
'broker_service' , { 'broker_instance_specifier' | 'current database' }[ ; ]
```

```
CREATE EVENT NOTIFICATION Notify_ALTER_T1
ON DATABASE
FOR ALTER_TABLE
TO SERVICE 'NotifyService',
   '8140a771-3c4b-4479-8ac0-81008ab17984';
```

§ 5 Представления

Отдельным пользователям необходима только часть информации, хранящейся в базе данных. С помощью механизма представлений можно предоставить доступ к части информации.

Представление – это именованная таблица, которая определяется в терминах других таблиц и не существует сама по себе.

После создания представления с ним можно работать как с обычной таблицей (есть исключения). Представления создаются на основе одной или нескольких таблиц и других представлений. Физически в базе данных хранится только определение представления, а сами данные хранятся только в таблицах. При изменении данных в исходных таблицах, изменения мгновенно отражаются и в представлениях. Когда изменения выполняются в представлении, они происходят в исходных таблицах.

Необходимость представлений:

- обеспечение логической независимости данных (пользователи и прикладные программы не зависят от изменений в логической структуре базы данных:
  - добавление нового атрибута (рост базы данных),
  - создание нового базового отношения (рост базы данных),
  - изменение расположения данных (реструктуризация базы данных), например, разделение на отношения;
- предоставление пользователям различных способов просмотра одних и тех же данных – простота использования и обучения (для лучшего понимания предметной области);
- обеспечивает доступ к части информации;
- скрытие данных (защита).

Типы представлений:

- вертикальное – подмножество столбцов таблицы (проекция), например, номер детали и цвет из таблицы детали;
- горизонтальное – подмножество строк таблицы, например, информация о деталях красного цвета;
- сжатие исходного отношения – количество поставляемых деталей каждого вида;
- соединение таблиц – пары поставщик–деталь из одного города;
- на основе других представлений – детали красного цвета из Москвы.

```
CREATE VIEW <имя> [(<новое имя столбца>, ... )]
AS <SELECT ...>
[WITH CHECK OPTION];
```

Примеры:

```
CREATE VIEW first AS SELECT id_d, color from d WITH CHECK OPTION
CREATE VIEW second AS SELECT * FROM d WHERE color="red" WITH CHECK OPTION
CREATE VIEW third (type, totqty) AS SELECT id_d, sum(number) FROM dp GROUP BY id_d
CREATE VIEW forth (pcity, dcity) AS SELECT DISTINCT p.city, d.city FROM d, p, dp WHERE
d.id_d=dp.id_d AND p.id_p=dp.id_p
CREATE VIEW fifth AS SELECT * FROM first WHERE city="Москва" WITH CHECK OPTION
```

Обновление представлений

С представлениями можно выполнять операции вставки и изменения в случае, если

- представление является подмножеством записей одной таблицы или представления;
- все столбцы, не вошедшие в представление, допускают пустые значения;
- оператор SELECT не должен содержать подзапросов, объединений, пересечений, разности, не содержит инструкций DISTINCT, GROUP BY, HAVING, агрегатных функций.

Не удовлетворяющие этим условиям представления предназначены только для чтения.

Инструкция WITH CHECK OPTION позволяет добавлять записи в представлении. Столбцы, которые не входят в представление, получают значение NULL.

§ 6 Снимки

Снимки (snapshots) – это именованное производное отношение. Это запрос, результат которого хранится в базе данных под указанным именем как отношение, которое допускает только чтение и периодически снимок обновляется, т.е. текущие данные аннулируются и запрос выполняется повторно, после чего результат запроса принимается в качестве нового значения снимка.

Многие приложения могут использовать или даже запрашивать данные по состоянию на конкретный момент времени. Например, в эту категорию приложений попадают приложения для отчетности и бухгалтерского учета. Подобные приложения обычно требуют фиксирования состояния данных в нужный момент (например, в конце периода отчетности) и снимки позволяют выполнять такую фиксацию, не влияя на другие транзакции, обновляющие данные.

```
CREATE SNAPSHOT <имя> AS <выражение> REFRESH EVERY <время_обновления>
```

Мгновенный снимок в MS SQL Server 2005

```
CREATE DATABASE <имя снимка> AS SNAPSHOT OF <имя базы данных>

CREATE DATABASE AW_Snapshot_Sample on (name='AdventureWorks_Data',
FILENAME='D:\MSSQL\Data\AW_S.ss')
AS SNAPSHOT OF AdventureWorks

SELECT ContactID, LastName FROM AW_Snapshot_Sample.Person.Contact WHERE ContactID=1
```

В файлах моментальных снимков баз данных не хранится полная копия базы данных на момент создания снимка, а содержатся только копии измененных, с момента создания снимка, страниц базы.

Глава 13. Восстановление

Восстановление в системе управления базами данных означает в первую очередь возвращение базы данных в правильное состояние. Основной принцип, на котором строится восстановление, это избыточность. Но эта избыточность организуется на физическом уровне и на логическом уровне пользователю не видна.

§ 1 Транзакции

Транзакция – последовательность операций, производимых над базой данных и переводящих базу данных из одного непротиворечивого (согласованного) состояния в другое непротиворечивое состояние. Транзакция рассматривается как некое неделимое действие над базой данных, осмысленное с точки зрения пользователя – логическая единица работы.

Разработчик сам определяет, какая последовательность операций является транзакцией, исходя из семантики предметной области. Например, перевод денег с одного счета на другой в банковской системе. Эта операция состоит из двух действий. На первом счету сумма уменьшается, а на втором увеличивается. Выполнение только одной из этих операций приведет к потере денег и целостности базы данных.

Транзакции обладают четырьмя свойствами:

- **Атомарность** – транзакция должна быть выполнена в целом или не выполнена вовсе.
- **Согласованность** – по мере выполнения транзакции база данных переходит из одного согласованного состояния в другое согласованное состояние.
- **Изолированность** – конкурирующие за доступ к базе данных транзакции обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.
- **Долговечность** – если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах.

Возможны два варианта завершения транзакции: фиксация или откат.

Фиксация транзакции – это запись на диск изменений в базе данных, которые были сделаны в процессе выполнения транзакции.

Если в процессе выполнения транзакции произошел сбой, и база данных должна быть возвращена в исходное состояние, то происходит **откат** транзакции, при котором происходит аннулирование всех изменений, которые были сделаны во время выполнения текущей незавершенной транзакции.

Оператор COMMIT фиксирует транзакцию. Он сообщает администратору транзакций, что логическая единица работы завершениа успешно, база данных вновь находится в согласованном состоянии, а все выполненные обновления теперь могут быть зафиксированы, т.е. стать постоянными.

Оператор ROLLBACK выполняет откат транзакции. Он сообщает администратору транзакций, что произошла какая-то ошибка, база данных находится в несогласованном состоянии и все обновления должны быть отменены, т.е. аннулированы.

Дополнительные команды:

- SAVEPOINT. Оператор SAVEPOINT позволяет создать в транзакции «метку», или точку сохранения. В одной транзакции можно выполнять оператор SAVEPOINT несколько раз, устанавливая несколько точек сохранения.
- ROLLBACK TO <точка сохранения>. Этот оператор используется совместно с представленным выше оператором SAVEPOINT. Транзакцию можно откатить до указанной точки сохранения, не отменяя все сделанные до нее изменения.

Завершением транзакции также может быть успешное или неуспешное завершение программы, в первом случае происходит фиксация транзакции, во втором – откат.

Начало транзакции определяют операторы BEGIN TRANSACTION, SET TRANSACTION или любой SQL–оператор.

Оператор SET TRANSACTION позволяет устанавливать атрибуты транзакции, такие как уровень изолированности и то, будет ли она использоваться только для чтения данных или для чтения и записи. Этот оператор также позволяет привязать транзакцию к определенному сегменту отката.

Явные транзакции. По умолчанию, каждая команда выполняется как отдельная транзакция. Пользователь может объединить несколько команд в одну транзакцию, явно указав ее начало и конец.

Неявные транзакции. Не существует оператора начала транзакции. Транзакция начинается с началом сеанса работы с БД. Завершается транзакция при следующих событиях:

- Явно выполненный оператор завершения транзакции – rollback или commit;
- Оператор DDL;
- Завершение сеанса.

После окончания транзакции сразу неявно начинается новая транзакция.

Журнал транзакций

Откат и фиксация транзакций становятся возможными благодаря **журналу транзакций (файл регистрации).**

Однако назначение журнала транзакций гораздо шире. Он позволяет восстанавливать согласованное состояние базы данных после любого рода аппаратных и программных сбоев.

Общая структура журнала может быть представлена в виде последовательного файла, в котором фиксируется каждое изменение базы данных. Каждая запись помечается номером транзакции, к которой она относится. Также в журнале фиксируется начало и конец каждой транзакции.

Существует два способа ведения журнала: **протокол с немедленными обновлениями** и **протокол с отложенными обновлениями.**

В первом случае запись состоит из двух компонентов: первый – это состояние строки таблицы до внесения изменений, второй – после внесения изменений и помечается номером транзакции, к которой относятся выполняемые изменения.

При выполнении любого оператора SQL, который модифицирует базу данных, СУБД автоматически заносит очередную запись в журнал транзакций. Только после внесения записи в журнал транзакция СУБД действительно модифицирует базу данных.

Если выполнен оператор COMMIT, то в журнале делается отметка о завершении текущей транзакции. Если выполнен оператор ROLLBACK или произошел сбой, то СУБД просматривает журнал транзакций и отыскивает записи, отражающие состояние строк до модификации. Используя их, СУБД восстанавливает те строки в таблицах, которые были модифицированы текущей транзакцией – таким образом аннулируются все изменения в базе данных.

Во втором случае запись содержит только вносимые в базу данных изменения. База данных модифицируется только после фиксации транзакции. Если выполнен оператор отката, то саму базу данных восстанавливать не нужно, так изменения были зафиксированы только в журнале, а в базу данных перенесены еще не были.

§ 2 Сбои

Отказ системы (например, сбой в питании), поражающий все выполняющиеся в данный момент транзакции, но физически не нарушающий базу данных в целом.

В некотором интервале система автоматически принимает контрольную точку, включающую запись содержимого рабочих буферов базы данных непосредственно в базу данных и список всех осуществляемых в данный момент транзакций. При загрузке системы после сбоя просматривается журнал транзакций. Завершившиеся после контрольной точки и до сбоя транзакции выполняются повторно (REDO), а незавершенные к моменту сбоя отменяются (UNDO).

Отказ носителя (например, поломка головок дискового накопителя), поражают всю базу данных или ее часть (физическое разрушение).

Восстановление после такого нарушения включает перезапись базы данных с резервной копии и последующее использование журнала транзакций, т.е. повторное выполнение всех транзакций, которые выполнялись с момента создания резервной копии базы данных.

§ 3 Параллелизм

Если с базой данных одновременно работают несколько пользователей, то обработка транзакций должна рассматриваться с новой точки зрения. В этом случае СУБД должна обеспечить корректную параллельную работу пользователей.

§ 3.1 *Проблемы параллелизма*

Основные проблемы, которые возникают при параллельной обработке транзакций:

- потеря результатов обновления;
- незафиксированная зависимость;
- несовместимый анализ.

Проблема потери результатов обновления

Транзакция A	время	Транзакция B
Извлечение кортежа p	t1	–
–	t2	Извлечение кортежа p
Обновление кортежа p	t3	–
–	t4	Обновление кортежа p

Два продавца обслуживают двух клиентов. Каждый продавец узнает, сколько продукции есть на складе, и уменьшает ее количество на размер купленной клиентом партии. Обновления, выполненные первой транзакцией, потеряны.

Проблема незафиксированной зависимости

Транзакция A	время	Транзакция B
Обновление кортежа p	t1	–
–	t2	Извлечение кортежа p
Отмена транзакции	t3	–

Первый продавец уменьшил количество на складе в соответствии с заказом покупателя. Второй продавец выполнил запрос и так как товара не хватает, то второму покупателю было отказано. После этого первый покупатель отказался от заказа и был выполнен откат транзакции. Действия транзакции B были выполнены на основе неверного предположения о состоянии кортежа p.

Может произойти и более ужасный случай.

Транзакция A	время	Транзакция B
Обновление кортежа p	t1	–
–	t2	Извлечение кортежа p
–	t3	Обновление кортежа p
Отмена транзакции	t4	–

Первый продавец уменьшил количество на складе в соответствии с заказом покупателя. Второй продавец продал второму покупателю часть оставшегося товара. После этого первый покупатель отказался от заказа и был выполнен откат транзакции. Утрачивается результат обновления, выполненного транзакцией B.

Проблема несовместимого анализа

Транзакция A	время	Транзакция B
Извлечение кортежа p 20	t1	–
Извлечение кортежа q 30	t2	–
–	t3	Обновление кортежа p ~~20~~ → 10
–	t4	Фиксация транзакции
Расчет sum = 50	t5	

В результате итоговая сумма будет содержать неверную информацию.

§ 3.2 *Механизм блокировки*

Перечисленные проблемы решаются с помощью механизма блокировки. Во время выполнения транзакции объект, с которым она выполняет операции, блокируется, т.е. становится недоступным другим транзакциям.

Транзакция A	Таблица 1	Таблица 2	Таблица 3	Транзакция B
Update t1	Locked A			
			Locked B	Update t3
Select t2		Locked A		
		wait B		Select t2
Update t1		wait B		
Update t2		wait B		
Commit	Unlocked	Unlocked		
		Locked B		
		Unlocked	Unlocked	Commit

В момент начала работы с объектом транзакция блокирует этот объект, т.е. получает монопольный доступ к нему, и освобождает его (разблокирует) после фиксации. Другие транзакции, которым необходим доступ к заблокированному объекту, ожидают, пока он не освободится.

Существует два вида блокировок: блокировка без взаимного доступа (eXclusive (монопольная)) и блокировка с взаимным доступом (Shared). Матрица совместимости для блокировок выглядит следующим образом:

A \ B	X	S	–
X	N	N	Y
S	N	Y	Y
–	Y	Y	Y

Транзакция A	Таблица 1	Таблица 2	Таблица 3	Транзакция B
Update t1	X–Locked A			
			X–Locked B	Update t3
Select t2		S–Locked A		
		S–Locked A, B		Select t2
		Unlocked B	Unlocked	Commit
Update t1		X–Locked A		
Update t2				
Commit	Unlocked	Unlocked		

§ 3.3 *Тупики*

Использование блокировки при выполнении параллельных транзакций может привести к тупиковой ситуации.

Транзакция A	Таблица 1	Таблица 2	Транзакция B
Update t1	X–Locked A		
		X–Locked B	Update t2
Update t2		wait A	
	wait B	wait A	Update t1
	wait B	wait A	

Транзакция A ждет завершения транзакции B, а транзакция B ждет завершения транзакции A. Такая ситуация называется **тупиком**. Ситуации могут быть гораздо более сложными.

Транзакция сама не может обнаружить тупик. Эта операция возлагается на СУБД.

Основой обнаружения тупиковых ситуаций является построение графа ожидания транзакций. Это ориентированный граф, вершинами которого являются транзакции. Если транзакция A ждет окончания транзакции B, то из вершины A в вершину B проводится дуга. Наличие цикла в графе говорит о возникновении тупика.

Разрушение тупика выполняется прерыванием одной из транзакций, находящейся в тупике. После заданного интервала времени прерванная транзакция снова начинает работу.

Типы блокировок

Объектами блокировок могут быть:

- база данных целиком, тогда это будет монопольный доступ к БД;
- отношение (таблица);
- кортеж (запись);
- поле данных.

При редактировании одной или нескольких записей удобно блокировать только изменяемые записи, тогда другие транзакции могут продолжать работу с остальными записями таблицы. При удалении всех строк таблицы лучше заблокировать всю таблицу.

Оптимистическое блокирование (optimistic locking) – стратегия блокирования набора данных, при которой раздел, содержащий изменяемую запись, блокируется только на время внесения изменений в запись программой, но не пользователем.

Пессимистическое блокирование (pessimistic locking) – стратегия блокирования набора данных, при которой раздел, содержащий изменяемую запись, блокируется на все время внесения изменений в запись пользователем и не доступна для редактирования другим пользователям.

§ 3.4 *Ошибки блокировки*

Может показаться, что блокировка записей более эффективна, чем блокировка таблицы, но не всегда.

Пример. Транзакция T1 выполняет запрос, в котором суммирует все выплаты сотрудника Иванова. При этом она блокирует все записи в таблице, для которых выполняется условие: «имя сотрудника = Иванов». В этот момент вторая транзакция T2 добавляет запись в эту же самую таблицу, в добавляемой записи находится еще одна сумма, начисленная Иванову. Результат выполнения первой транзакции будет зависеть от того, успеет или нет вторая транзакция записать данные, если нет, то новая запись не попадет под блокировку и транзакция T1 выдаст неправильный результат.

§ 3.5 *Блокировки в Oracle*

Блокировки данных не хранятся как отдельный ресурс, а содержатся непосредственно в блоках данных. Это позволяет избежать таких проблем, как эскалация блокировок. Ниже перечислены пять основных классов блокировок в Oracle. Первые три – общие (используются во всех базах данных Oracle), а две остальные – только в OPS (Oracle Parallel Server – параллельный сервер).

1. Блокировки ЯМД (DML locks). ЯМД означает язык манипулирования данными (Data Manipulation Language), то есть операторы SELECT, INSERT, UPDATE и DELETE. К блокировкам ЯМД относятся, например, блокировки строки данных или блокировка на уровне таблицы, затрагивающая все строки таблицы.
2. Блокировки ЯОД (DDL locks). ЯОД означает язык определения данных (Data Definition Language), то есть операторы CREATE, ALTER и так далее. Блокировки ЯОД защищают определения структур объектов.
3. Внутренние блокировки (internal locks) и защелки (latches). Защелки – это простые низкоуровневые средства обеспечения последовательности обращений. Защелки обычно запрашиваются системой в режиме ожидания. Это означает, что, если защелку нельзя установить, запрашивающий сеанс приостанавливает работу на короткое время, а затем пытается повторить операцию. Другие защелки могут запрашиваться в оперативном режиме, то есть процесс будет делать что-то другое, не ожидая возможности установить защелку. Защелки выделяются случайным образом. Внутренние блокировки – более сложное средство обеспечения очередности доступа, они позволяют запрашивающему «встать в очередь» в ожидании освобождения ресурса. Запрашивающий защелку сразу уведомляется об освобождении ресурса. В случае внутренней блокировки запрашивающий полностью блокируется.
4. Распределенные блокировки (distributed locks). Эти блокировки используются сервером OPS для согласования ресурсов машин, входящих в кластер. Распределенные блокировки устанавливаются экземплярами баз данных, а не отдельными транзакциями.
5. Блокировки параллельного управления кэшем (PCM – Parallel Cache Management Locks). Такие блокировки защищают блоки данных в кэше при использовании их несколькими экземплярами баз данных.

§ 3.6 *Блокировки в MS SQL Server*

SQL Server поддерживает три основных типа блокировок:

1. Shared Lock – разделяемая блокировка, которая используется при выполнении операции чтения данных. Позволяется чтение данных другой транзакцией, но запрещено изменение данных.
2. Exclusive Lock – монопольная блокировка, которая применяется при изменении данных. Эта блокировка полностью запрещает доступ к данным другими транзакциями.
3. Update Lock – блокировка обновления, которая является промежуточной между разделяемой и монопольной блокировкой. Используется, когда транзакция хочет обновить данные в какой-то ближайший момент времени, но не сейчас, и, когда этот момент придет, не хочет ожидать другой транзакции. В этом случае другим транзакциям разрешается устанавливать разделяемые блокировки, но не позволяет устанавливать монопольные.

§ 4 Уровни изолированности транзакций

Существует 4 стандартных уровня (ANSI SQL-92) изолированности транзакций:

- **Read Uncommitted (Dirty Read)** – «грязное» (или «незафиксированное») чтение. Транзакция может читать не подтвержденные изменения, сделанные в других транзакциях. Например, если транзакции A и B стартовали, и поменяли записи, то они обе видят изменения друг друга.
- **Read Committed** – невоспроизводимое (или неповторяемое) чтение. Транзакция может читать только те изменения, которые были подтверждены другими транзакциями. Например, если транзакции A и B стартовали и поменяли записи, то они не видят изменения друг друга. Транзакция A увидит изменения транзакции B только тогда, когда транзакция B завершится по commit. Перечитывание данных в транзакции может выдавать разные результаты.
- **Repeatable Read** – воспроизводимое (или повторяемое) чтение. Транзакция видит только те данные, которые существовали на момент ее старта.
- **Serialized** – сериализуемость. Транзакция выполняются так, как будто никаких других транзакций в этот момент не существует. Или, транзакции выполняются так, как будто они выполняются последовательно.

Типы транзакций в InterBase

```
SET TRANSACTION [{READ WRITE | READ ONLY}] [{WAIT | NO WAIT}] [[ISOLATION
LEVEL] {SNAPSHOT [TABLE STABILITY] | READ COMMITTED [[NO] RECORD_VERSION]}]
[RESERVING <reserving_clause>];

<reserving_clause> = table [, table …] [FOR [SHARED | PROTECTED] {READ | WRITE}] [,
<reserving_clause>]
```

READ WRITE / READ ONLY (константы write и read) – операторы внутри транзакции могут или не могут модифицировать данные. По умолчанию READ WRITE, т.е. допускается и чтение и запись. Для

readonly баз данных IB 6 транзакции могу стартовать как read или write, но любые операции изменения данных будут вызывать сообщение об ошибке.

WAIT / NO WAIT (константы wait и nowait) – Режимы обработки конфликтов блокировок. Если транзакция стартует в режиме WAIT (по умолчанию), и при выполнении операции обнаруживается конфликт, то операция "замораживается" до разрешения конфликта. В режиме NO WAIT сообщение о конфликте выдается приложению немедленно (возникает ошибка), а операция, которая привела к конфликту, отменяется. В случае взаимоблокировки двух wait-транзакций сервер автоматически обнаруживает эту ситуацию, и разблокирует одну из транзакций (как будто она стартовала как nowait) через интервал времени, определенный в IBCONFIG параметром DEADLOCK_TIMEOUT, который по умолчанию равен 10 секундам.

SNAPSHOT (константа concurrency) – уровень изоляции, эквивалентный Repeatable Read. На самом деле этот уровень изоляции ближе к "изолированность образа", т.к. не допускает фантомов. Все операции в транзакции с данным уровнем изоляции видят только те данные, которые существовали на момент старта этой транзакции (даже если они впоследствии были изменены или удалены другими транзакциями). По умолчанию.

SNAPSHOT TABLE STABILITY (константа consistency) – изолированность образа (воспроизводимое чтение) с возможностью резервирования таблиц для чтения и записи. Уникальный режим, будет подробно рассмотрен чуть дальше.

READ COMMITTED (константа read_committed) – уровень изоляции ReadCommitted. Т.е. в данной транзакции все изменения, которые были подтверждены другими транзакциями, будут видны немедленно. Имеет две опции:

NO RECORD_VERSION (константа no_rec_version) – запрещает чтение измененных, но не подтвержденных (non committed) записей. При чтении таких записей возникает ошибка "deadlock". По умолчанию для режима READ COMMITTED.

RECORD_VERSION (константа REC_VERSION) – разрешает чтение только подтвержденных (committed) данных. Именно этот режим является режимом по умолчанию в BDE.

§ 5 Транзакции в Oracle

Неявные транзакции: транзакция автоматически начитается или с начала сеанса или с момента окончания предыдущей транзакции.

Типы транзакций: Read Write, Read Only. Транзакция Read Only эквивалентна при чтении уровню изолированности SERIALIZABLE. Такая транзакция видит только изменения, зафиксированные на момент ее начала, но в этом режиме не разрешена вставка, изменение и удаление данных (другие сеансы могут изменять данные, но транзакция только для чтения – нет).

Уровни изолированности: Read Committed, Serializable.

Механизм многоверсионности. Основные характеристики:

- Согласованность по чтению для запросов. Запросы выдают согласованные результаты на момент начала их выполнения. Изменяемые данные помещаются в область сегмента отката.
- Неблокируемые запросы. Запросы, изменяющие данные, не блокируют запросы, читающие данные, и читающие запросы не блокируют изменяющие, как это бывает в других СУБД.

Операторы управления транзакциями: SET TRANSACTION, COMMIT, ROLLBACK. SAVEPOINT

§ 6 Транзакции в MS SQL Server

Явные транзакции: каждый оператор выполняется в своей транзакции, если он не находится в блоке begin tran – commit (rollback).

Наличие вложенных транзакций. Пример

```
begin tran
.....
    begin tran
    .....
    commit (rollback)
.....
commit (rollback)
```

Подтверждение вложенной транзакции ни на что не влияет. Откат вложенной транзакции откатывает самую внешнюю транзакцию.

Операторы управления транзакциями: SET TRANSACTION, BEGIN TRANSACTION, COMMIT, ROLLBACK, SAVEPOINT

§ 7 Технология MVCC

Управление конкурентным доступом с помощью многоверсионности (англ. MVCC — MultiVersion Concurrency Control) — один из механизмов обеспечения одновременного конкурентного доступа к БД, заключающийся в предоставлении каждому пользователю т. н. «снимка» БД, обладающего тем свойством, что вносимые данным пользователем изменения в БД невидимы другим пользователям до момента фиксации транзакции. Этот способ управления позволяет добиться того, что пишущие транзакции не блокируют читающих, и читающие транзакции не блокируют пишущих.

PostgreSQL поддерживает одновременную модификацию БД несколькими пользователями с помощью механизма Multiversion Concurrency Control (MVCC). Благодаря этому соблюдаются требования ACID, и практически отпадает нужда в блокировках чтения.

Глава 14. Архитектура клиент-сервер

§ 1 Введение

Основной принцип технологии «клиент-сервер» заключается в разделении функций стандартного интерактивного приложения на три группы, имеющие различную природу.

- Первая группа – это функции ввода и отображения данных (компонент представления).
- Вторая группа объединяет чисто прикладные функции, определяющие основные алгоритмы решения задач приложения (прикладной компонент).
- Третьей группе относятся функции хранения и управления информационными ресурсами (компонент управления).
- Можно выделить четвертую группу, выполняющую служебные функции, играющие роль связок между остальными.

В централизованной архитектуре эти части приложения располагаются внутри одной исполняемой программы.

В децентрализованной архитектуре эти задачи могут быть по-разному распределены между серверным и клиентским процессами. Различия в реализации технологии «клиент-сервер» определяются несколькими факторами: во-первых, тем, в какие виды программного обеспечения интегрирован каждый из этих компонентов, во-вторых, тем, какие механизмы программного обеспечения используются для реализации функций всех групп, в-третьих, как логические компоненты распределяются между компьютерами в сети, в-четвертых, какие механизмы используются для связи компонентов между собой.

В связи с этим выделяют четыре модели:

- модель файлового сервера;
- модель удаленного доступа к данным;
- модель сервера базы данных;
- модель сервера приложений.

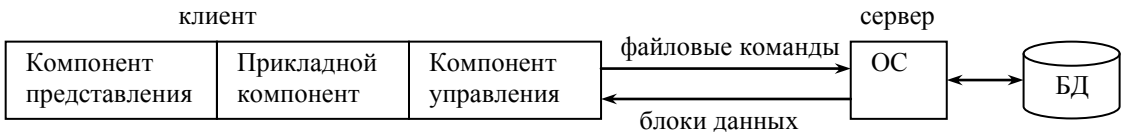
§ 2 Модель файлового сервера

Файлы базы данных хранятся на сервере, который содержит только систему доступа к информационным ресурсам – файлам. На клиенте функционирует приложение, содержащее компонент управления данными, компонент представления и прикладной компонент.

На клиенте выполняется копия СУБД и прикладная программа, а база данных содержится в разделяемых файлах, которые находятся на файловом сервере. Когда прикладная программа обращается к базе данных, СУБД направляет запрос на чтение или запись блока информации.

Недостатки:

- высокий сетевой трафик;
- узкий спектр операций манипулирования данными;
- отсутствие средств безопасности доступа к данным (защита только на уровне файловой системы).



§ 3 Модель удаленного доступа к данным

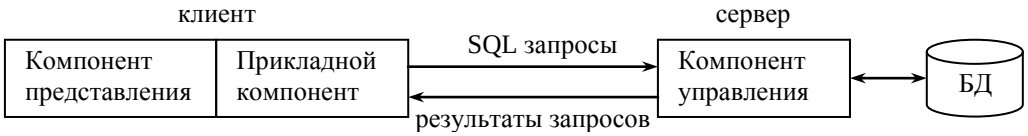
На сервере находится ядро СУБД и база данных. На клиенте располагаются компонент представления и прикладной компонент. Клиент обращается к информационным ресурсам с помощью запросов, обычно на языке SQL. На сервере СУБД обрабатывает запрос и возвращает клиенту результат, оформленный как блок данных. Инициатор манипуляций с данными – программы клиента. СУБД обслуживает только запросы.

Достоинства:

- разгрузка сервера БД по сравнению с централизованной архитектурой;
- разгрузка сети по сравнению с файловым сервером – SQL запросы и ответы;
- унификация интерфейса "клиент-сервер" в виде языка SQL, язык используется не только в качестве средства доступа к данным, но и как стандарта общения клиента и сервера.

Недостатки:

- загруженность сети;
- отсутствует централизованное управление данными, так как на клиенте находятся и компонент представления и прикладной компонент.



§ 4 Модель сервера базы данных

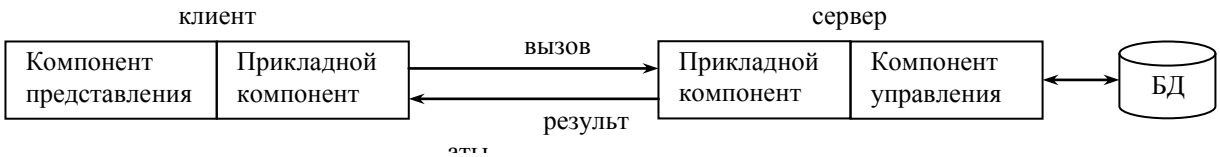
На сервере находится ядро СУБД и база данных. На клиенте находится компонент представления. Прикладной компонент разделен между сервером и клиентом. На сервере он представлен в виде хранимых процедур, триггеров и событий. В этой модели сервер является активным, потому что не только клиент, но и сам сервер, используя механизм триггеров, может инициировать обработку данных.

Достоинства:

- снижение трафика по сравнению с моделью удаленного доступа к данным – посылаются команды на выполнение процедур, ответ – только данные необходимые для работы клиентской части прикладного компонента или для представления;
- сокращение дублирования алгоритмов обработки данных за счет разделения процедур между несколькими приложениями;
- централизованное администрирование базы данных, сохранение целостности.

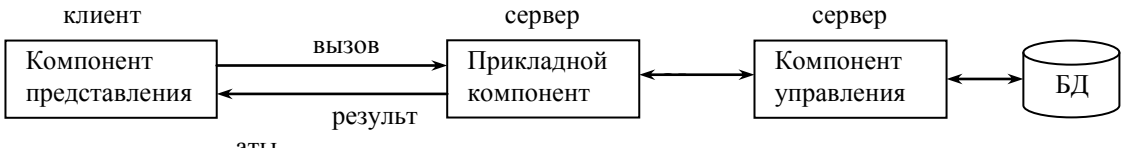
Недостатки:

- большая загрузка сервера – функции СУБД, запуск и выполнение хранимых процедур, запуск и выполнение триггеров, мониторинг событий, возвращение данных клиенту;
- ограниченность языка для написания хранимых процедур.



§ 5 Модель сервера приложений

Для разгрузки сервера была предложена трехуровневая модель сервера приложений. Клиент содержит компонент представления. Прикладной компонент находится на сервере приложений. СУБД и компонент управления находится на сервере БД.



§ 6 Поддержка технологий клиент-сервер в стандарте языка SQL

Установление соединения с базой данных:

```
CONNECT TO { <база данных>
[ AS <имя подключения к базе данных>]
[ USER <имя пользователя>] | DEFAULT }
```

Клиент может иметь только одно активное соединение и любое число пассивных соединений. Все запросы от клиента к базе данных направляются и обрабатываются сервером активного соединения. Изменение активного соединения:

```
SET CONNECTION {<имя подключения> | DEFAULT}
```

Разрыв соединения с базой данных:

```
DISCONNECT {<имя подключения> | ALL | CURRENT}
```

Глава 15. Распределенные базы данных

§ 1 Определение

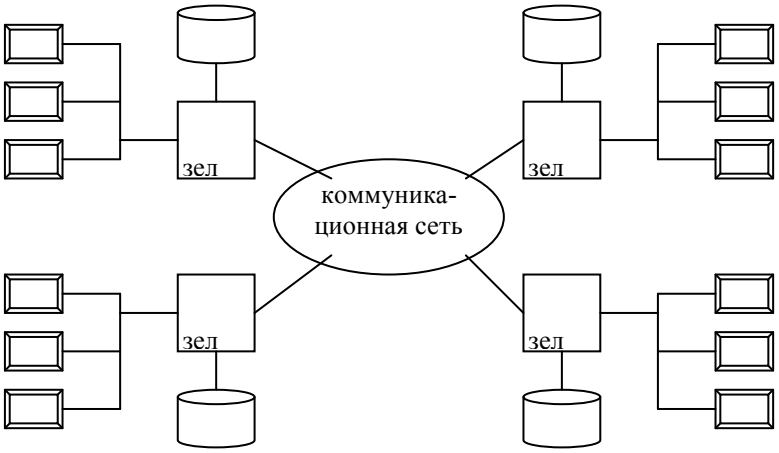
Распределенная база данных состоит из набора узлов, связанных вместе коммуникационной сетью. Каждый узел обладает:

- своими собственными базами данных;
- собственными локальными пользователями;
- собственной СУБД;
- собственным программным обеспечением для управления транзакциями;
- собственным локальным диспетчером передачи данных.

В распределенной базе данных:

- Пользователь на локальном узле может выполнять операции с данными так, как будто этот узел вовсе и не является частью распределенной системы.
- Узлы работают согласованно, поэтому пользователь может получить доступ к данным на любом другом узле сети, как будто все данные находятся на его собственном узле.

Предполагается, что узлы разделены физически, а также географически, но достаточно, чтобы они были распределены логически.



Преимущества и примеры

Предприятие имеет распределенную структуру. Распределенная база данных обеспечивает:

- эффективность обработки – данные хранятся в близи о того места, где они более всего необходимы;
- повышенная доступность – можно обеспечить доступ к остальным данным.

Основной недостаток – сложность с технической точки зрения.

Примеры использования: в банковской системе, авиалинии.

Примеры распределенных СУБД: Oracle 7, INGRES/STAR, DB2.

Фундаментальный принцип

- Для пользователя распределенная система должна выглядеть точно так же, как нераспределенная система.

По отношению к пользователям, выполняющим операции управления данными. Операции определения данных могут быть расширены (указание узла хранения данных). Все проблемы должны возникать на внутреннем уровне или уровне разработки системы и не должны быть видны пользователю.

Для поддержания фундаментального принципа требуется выполнение 12 правил.

§ 2 Правила

§ 2.1 *Локальная автономия*

Операции на данном узле управляются данным узлом, т.е. функционирование любого узла не зависит от успешного выполнения некоторых операций на каком-то другом узле. Владение, управление и учет данных осуществляются локально.

Абсолютной автономии достичь невозможно, поэтому речь идет о максимально возможной.

§ 2.2 *Независимость от центрального узла*

Все узлы рассматриваются как равные, централизованное управление отсутствует. Выполнение первого правила влечет выполнение и второго (является логическим следствием), но так как выполнение первого не достигается в полной мере, то второе может рассматриваться как отдельная цель.

Центральный узел может быть узким местом системы и делает ее уязвимой.

§ 2.3 Непрерывное функционирование

Повышается надежность системы за счет того, что при отказе одного узла остальные продолжают работать.

За счет этого повышается доступность системы, а также за счет репликации данных.

§ 2.4 Независимость от расположения

Прозрачность расположения. Пользователи не должны знать, в каком физическом месте хранятся данные.

§ 2.5 Независимость от фрагментации

В системе поддерживается **фрагментация данных**, если некоторое отношение разделено на части (фрагменты). При этом должны выполняться условия:

- все фрагменты 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;
- проекции не должны допускать потерю информации (для вертикальной фрагментации).

Фрагментация желательна для повышения производительности системы, поскольку лучше хранить данные там, где они наиболее часто используются. При такой организации многие операции будут локальными, а объем пересылаемых по сети данных снизится.

Существует два типа фрагментации – **горизонтальная** и **вертикальная**, которые связаны с реляционными операциями выборки и проекции соответственно.

Реконструкцию исходного отношения на основе его фрагментов можно осуществить с помощью операций соединения (для вертикальных фрагментов) и объединения (для горизонтальных фрагментов).

Независимость от фрагментации (прозрачность фрагментации) означает, что пользователь работает с базой данных как будто она нефрагментирована. Системный оптимизатор отвечает за определение фрагментов, к которым необходимо обеспечить физический доступ для выполнения запроса пользователя (нужно ли обращаться к другим узлам или нет).

Например, пусть, отношение d (детали) разделено на два отношения d_Moscow (детали, изготавливаемые в Москве) и d_Spb (детали, изготавливаемые в СПб) (Fragment d into d_Moscow at site 'Moscow' Where city = 'Москва', d_Spb at site 'Spb' Where city='СПб'), тогда запрос Select id_d, name From d Where color = 'red' оптимизатор должен преобразовать к виду: Select id_d, name From d_Moscow Where color = 'red' Union Select id_d, name From d_Spb Where color= 'red'.

Если запрос имеет вид Select id_d, name From d Where color = 'red' and city = 'Москва', тогда оптимизатор, преобразовав его к виду Select id_d, name From d_Moscow Where color = 'red' and city = 'Москва' Union Select id_d, name From d_Spb Where color= 'red' and city = 'Москва', из хранящегося в каталоге описания отношений d_Moscow и d_SPb оптимизатор должен определить, что второй запрос даст пустое множество, тогда запрос можно записать следующим образом Select id_d, name From d_Moscow Where color = 'red' и следует обращаться только к узлу в городе Москва.

§ 2.6 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.

Репликация полезна по двум причинам:

- достигается большая производительность (приложения могут работать с локальными копиями, не обмениваясь данными с удаленными узлами);
- обеспечивает большую доступность (реплицированный объект остается доступным для обработки до тех пор, пока остается хотя бы одна его реплика).

Главный недостаток репликации – проблема распространения обновления – при обновлении реплицируемого объекта, все копии этого объекта также должны обновляться.

Независимость от репликации подразумевает, что системный оптимизатор отвечает за определение физического доступа именно к тем репликам, которые необходимы для удовлетворения заданного пользовательского запроса.

Трудности возникают из-за того, что некоторый узел, содержащий данный объект, может быть недоступен именно в момент обновления. Тогда немедленное обновление (а значит, и исполнение транзакции) будет провалено, если одна из этих копий недоступна в текущий момент времени. Общая схема устранения этой проблемы называется схемой первичной копии.

- Одна копия каждого реплицируемого объекта называется первичной копией, а все остальные – вторичными.
- Первичные копии различных объектов находятся на различных узлах.
- 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.

В некоторых коммерческих СУБД используется менее амбициозная форма репликации, в которой распространение обновления гарантируется в будущем, но не обязательно в рамках соответствующей транзакции. Такое откладываемое распространение не гарантирует совместимость базы данных в произвольный момент времени.

§ 2.7 *Обработка распределенных запросов*

Вопрос оптимизации запросов имеет особое значение для распределенной системы. В глобальной сети интенсивность обмена данными в тысячу раз медленнее интенсивности обмена данными для жесткого диска. Вследствие этого основным требованием к распределенным системам является минимизация использования сети, т.е. сокращение до минимума количества и объема пересылаемых в сети сообщений.

При выполнении охватывающего несколько узлов запроса существует довольно много способов перемещения данных по сети.

В распределенной системе и оптимизация должна быть распределенной. Общий процесс оптимизации обычно состоит из этапа глобальной оптимизации, который сопровождается несколькими этапами локальной оптимизации.

Рассмотрим базу данных поставщиков и деталей:

Отношение	количество кортежей	узел
p (id_p, city) – поставщики	10 000	A
d (id_d, color) – детали	100 000	B
dp (id_p, id_d) – поставки	1 000 000	A

Запрос "Получить сведения о находящихся в Москве поставщиках красных деталей".

Размер кортежа = 25 байт (200 бит).
Интенсивность обмена данными = 50 000 бит/с.
Задержка доступа = 0,1 с.
Число красных деталей = 10.
Число поставок, выполняемых поставщиками из Москвы = 100 000.

Рассмотрим 6 возможных стратегий обработки запроса с вычислением для каждой стратегии общего времени передачи данных по формуле:

T = общая задержка доступа + (общий объем данных / интенсивность обмена данными) =
= (число сообщений / 10) + (число бит / 50 000)

4. Переместить отношение детали на узел A и выполнить запрос на узле A.
   $T_1 = 0{,}1 + (100\,000 * 200) / 50\,000 \approx 400$ с (6,67 мин)
5. Переместить отношения поставщики и поставки на узел B и выполнить запрос на узле B.
   $T_2 = 0{,}2 + ((10\,000 + 1\,000\,000) * 200) / 50\,000 \approx 4\,040$ с (1,12 ч)
6. Соединить отношения поставщики и поставки на узле A, выбрать из полученного результата кортежи для поставщиков из Москвы, а затем для каждого кортежа проверить, не является ли соответствующая деталь красной.
   $T_3 \approx 20\,000$ с (5,56 ч)
7. Выбрать из отношения детали на узле B кортежи, соответствующие красным деталям, а затем для каждого кортежа проверить, не поставляется ли соответствующая деталь поставщиком из Москвы.
   $T_4 \approx 2$ с
8. Соединить отношения поставщики и поставки на узле A, выбрать из полученного результата кортежи для поставщиков из Москвы, результат разбить на проекции по атрибутам id_p, id_d, а затем переместить на узел B. Завершить выполнение запроса на узле B.
   $T_5 = 0{,}1 + (100\,000 * 200) / 50\,000 \approx 400$ с (6,67 мин)
9. Выбрать из отношения детали на узле B кортежи, соответствующие красным деталям, а затем переместить результат на узел A. Завершить выполнение запроса на узле A.
   $T_6 = 0{,}1 + (10 * 200) / 50\,000 \approx 0{,}1$ с

Интенсивность обмена данными и задержка доступа являются важными факторами, влияющими на выбор той или иной стратегии.

В примере игнорировался вопрос о том, какой узел получает окончательные результаты.

Некоторые стратегии позволяют выполнять параллельную обработку на двух узлах. Время отклика в такой системе может оказаться меньше времени отклика в централизованной системе.

§ 2.8 *Управление распределенными транзакциями*

Управление обработкой транзакций включает два аспекта: управление восстановлением и управление параллелизмом.

В распределенной системе выполнение одной транзакции может быть связано с исполнением кода на нескольких узлах.

Управление восстановлением

Управление восстановлением основано на протоколе двухфазной фиксации:

- Когда при обработке транзакции встретился оператор commit, координатор направляет уведомление всем участникам – узлам, выполняющим распределенную транзакцию, "подготовиться к фиксации". Это означает, что каждый узел должен сохранить все записи журнала транзакций. Если запись прошла успешно, то узел посылает сообщение координатору "ОК", иначе "не ОК".
- Когда координатор получил ответы от всех участников, он заносит записи в собственный журнал транзакций, если все ответили "ОК", то принимает решение о фиксации транзакции, если хотя бы один узел ответил "не ОК" – откат. Затем координатор информирует каждого участника о своем решении, и каждый участник локально фиксирует или выполняет откат транзакции.

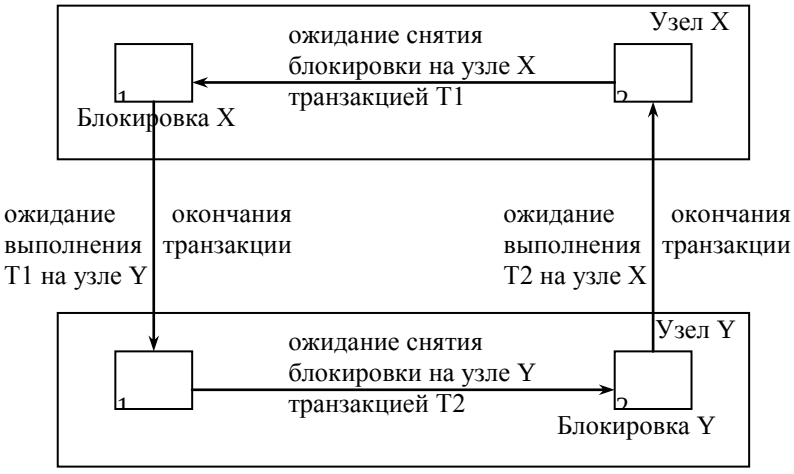
Особенности:

- Функции координатора не должны присваиваться ни одному из узлов. Обычно они выполняются узлов, где была запущена транзакция. Таким образом, каждый узел должен для одних транзакций выполнять роль узла-координатора, а для других – узла-участника.
- При двухфазной фиксации координатор обменивается данными с каждым узлом-участником, что означает большое количество сообщений и большие накладные расходы.
- Если узел А действует как участник процесса двухфазной фиксации, координируемого узлом В, то узел А должен выполнять любые действия, предписываемые узлом В. При этом неизбежна утрата локальной автономности.
- Существуют модификации протокола двухфазной фиксации с меньшим числом сообщений.

Управление параллелизмом

Управление параллелизмом основано на блокировке. В распределенной системе запросы на проверку, установку и снятие блокировок являются сообщениями, что влечет за собой дополнительные накладные расходы.

Другой проблемой является то, что блокировка может привести к глобальному тупику, который охватывает два или более узлов. Ни один из узлов не может обнаружить тупик, используя только информацию, которая сосредоточена на этом узле.



§ 2.9 *Независимость от аппаратного обеспечения*

Возможность запуска копий одной и той же СУБД на разном аппаратном обеспечении с тем, чтобы разные компьютеры могли работать в распределенной системе как равные партнеры.

§ 2.10 *Независимость от операционной системы*

Возможность запуска копий одной и той же СУБД на разных операционных системах с тем, чтобы разные версии СУБД могли работать в одной распределенной системе.

§ 2.11 *Независимость от сети*

В системе должны поддерживаться разные типы сетей.

§ 2.12 *Независимость от СУБД*

Разные СУБД должны работать совместно, т.е. все СУБД поддерживают один и тот же интерфейс (хотя бы официальный стандарт языка SQL). СУБД должна иметь надстроечную программу, которая называется **шлюзом** и взаимодействует с другой СУБД.

Шлюз должен выполнять следующие функции:

- Поддержка протокола обмена информацией между различными СУБД.
- Динамическая поддержка языка SQL.
- Отображение между типами данных СУБД (длина переменных, кодировки символов, формат чисел с плавающей запятой, управление датой и временем, упорядочивание и т.д.).
- Отображение каталога СУБД.
- Поддержка протокола двухфазной фиксации.
- Поддержка блокировки.

§ 3 Управление каталогом

Каталог распределенной системы содержит не только обычные данные, касающиеся базовых отношений, представлений, индексов и т.д., но также и всю информацию, необходимую для обеспечения независимости размещения, фрагментации и репликации. В таком случае возникает вопрос: где и как следует хранить системный каталог?

1. Централизованный каталог. Весь каталог хранится в одном месте на центральном узле. Не достигается независимость от центрального узла.
2. Полностью реплицированный каталог. Весь каталог полностью хранится на каждом узле. Утрачивается автономность функционирования, так как при обновлении каждого каталога это обновление придется распространить на каждый узел.
3. Секционированный каталог. На каждом узле содержится его собственный каталог для объектов, хранимых на этом узле. Общий каталог является объединением всех разъединенных локальных каталогов. Выполнение нелокальных операций становится дорогостоящим, поскольку потребуется осуществить доступ к половине имеющихся узлов для поиска удаленного объекта.
4. Комбинация первого и третьего вариантов. На каждом узле хранится собственный локальный каталог, кроме того, на одном центральном узле хранится унифицированная копия всех этих локальных каталогов.
5. Каталог системы R*. Системное имя объекта содержит идентификатор создателя объекта, идентификатор узла создателя, локальное имя, идентификатор узла хранения (такое имя гарантировано от каких-либо изменений даже при перемещении этого объекта на другой узел). На каждом узле поддерживаются таблица системных имен (включающих идентификатор узла создателя), элемент каталога объектов, созданных на этом узле и элемент каталога объектов, хранимых в данный момент на этом узле. Объект будет найден с помощью двух попыток удаленного доступа. При миграции объекта потребуется выполнить следующие действия: вставить элемент каталога на новом узле, удалить элемент каталога на старом узле, обновить элемент каталога на узле создателе.

Пример:

	A		B		C
1	detail A delivery B		detail A delivery B		detail A delivery B
2	detail C		delivery B		
3			delivery		detail

Операция перемещения таблицы delivery с узла B на узел C:
1. Из B.3 удаляем запись,
2. В B.2 изменяем delivery C,
3. В C.3 добавляем delivery.

Доступ к detail на узле B:
1. Просмотр имен: таблица detail создана на узле A,
2. Обращение к узлу A: чтение A.2: таблица хранится на узле C,
3. Обращение к узлу C: передача данных из таблицы detail.

Глава 16. Использование SQL в программах

§ 1 Использование SQL в приложениях

Прежде чем выполнить SQL команду СУБД строит план выполнения команды на основе метаданных: определяет последовательность соединения таблиц, определяет порядок сортировки, определяет, какой индекс использовать и т. д.

SQL команды могут быть включены в программу двумя способами:

- Statement-level interface (SLI) – программа соединяет команды на языке программирования и SQL конструкции. Перед компиляцией программы предкомпилятор обрабатывает SQL команды, которые преобразуются в вызовы процедур на языке программирования. В процессе выполнения программы эти процедуры соединяются с СУБД. SQL команды могут быть записаны двумя способами:
  - встроенный SQL (Embedded SQL или Static SQL) – обычные команды (пример – SQLJ)
  - динамический SQL – команды записаны как строковые константы на языке программирования и поэтому они неизвестны во время компиляции, а обрабатываются уже при выполнении программы, это требует дополнительного времени для отправки команды на подготовку к исполнению.
- Call-level interface (CLI) – программа пишется только на языке программирования. Все SQL команды представляют собой строковые константы, распознаваемые при выполнении программы (так же как динамический SQL), (Пример: JDBC, ODBC).

Для выполнения SQL команд необходимо использовать специальный синтаксис.

В некоторых случаях приложение должно использовать некоторый диалект языка SQL, поддерживаемый СУБД, таким образом, может возникнуть проблема при переносе приложения на другую СУБД. При использовании ODBC, JDBC команды записываются на универсальном языке, который транслируется в диалект СУБД с помощью соответствующих драйверов СУБД.

Чтобы избежать затрат времени на подготовку SQL команды, формируемой в процессе выполнения программы, рекомендуется разрабатывать хранимые процедуры, в этом случае план обработки команд формируется заранее.

§ 2 Встроенный SQL

Языки программирования могут позволять включать операторы языка SQL непосредственно в текст программы.

§ 2.1 *Соединение*

Прежде чем выполнять запросы, необходимо установить SQL соединение с SQL сервером. Такое соединение открывает сессию на сервере. Пока сессия открыта, можно выполнять транзакции. Отсоединение от SQL сервера приводит к закрытию сессии.

Установление соединения:

```
CONNECT TO { DEFAULT | <имя базы данных> }
[ AS <имя соединения> ] [ USER <имя пользователя> ]
```

Отсоединение:

```
DISCONNECT  { DEFAULT | <имя базы данных> }
```

Можно установить несколько соединений одновременно, но активным будет только одно соединение.

Переключение на другое соединение (активизация соединения):

```
SET CONNECTION TO { DEFAULT | <имя соединения> }
```

§ 2.2 *Выполнение запросов*

В запросах можно использовать обычные переменные, с типами данных, определенными в языке программирования. Для использования обычных переменных в запросах их необходимо описать в специальной секции.

```
EXEC SQL BEGIN DECLARE SECTION;
    <определение переменных>;
EXEC SQL END DECLARE SECTION;
```

Пример

```
EXEC SQL BEGIN DECLARE SECTION;
    unsigned long num_enrolled;
    char *crs_code, *semester;
EXEC SQL END DECLARE SECTION;
EXEC SQL
    Select C.Enrollment Into :num_enrolled From Class C Where C.CrsCode = :crs_code
    AND C.Semester = :semester;
```

В примере num_enrolled, crs_code, semester – переменные, описанные в специальной секции для описания переменных SQL. В запросе эти переменные выделяются двоеточием. В запросе добавлена опция Into, чтобы записать в переменные результаты операции выборки. Это работает только в случае, когда результатом является только один кортеж (одна запись).

Пример

```
#define OK “00000”
EXEC SQL BEGIN DECLARE SECTION;
    char SQLSTATE[6];
    unsigned long num_enrolled;
    char *crs_code, *semester;
EXEC SQL END DECLARE SECTION;
EXEC SQL
    Select C.Enrollment Into :num_enrolled From Class C Where C.CrsCode = :crs_code
    AND C.Semester = :semester;
if  ( strcmp(SQLSTATE, OK) != 0 )
    printf(“Select statement failed\n”);
```

В примере определена специальная переменная – SQLSTATE, чтобы проследить, успешно ли выполнился запрос. В случае успешного выполнения она принимает значение “00000”.

§ 2.3 *Курсор*

Если запрос возвращает несколько записей (кортежей), то вместо переменных используется курсор.

Определение курсора

```
DECLARE <имя курсора> [ INSENSITIVE ] [ SCROLL ] CURSOR FOR <имя таблицы или
представления или select запрос>
[ ORDER BY <список столбцов> ]
[ FOR { READ ONLY | UPDATE [ OF <список столбцов> ] } ];
```

INSENSITIVE – при открытии курсора создается копия таблицы, и вся работа осуществляется над записями этой копии (работает со снимком). Все действия, выполненные над таблицей после открытия курсора, никак не отразятся на записях, доступ к которым выполняет курсов, т. е. не будут видны курсору. Все изменения, сделанные в курсоре, никак не отразятся на базовых таблицах, т. е. будут потеряны.

SCROLL – разрешает любые способы передвижения по записям курсора, иначе только на следующую запись.

READ ONLY – работает только в паре с INSENSITIVE. Запрещает редактирование записей курсора.

Открытие курсора:

```
OPEN <имя курсора>;
```

Закрытие курсора:

```
CLOSE <имя курсора>;
```

Чтобы работать с одной записью курсора, ее необходимо получить.

Получение одной записи:

```
FETCH [ [ <направление перемещения указателя записи> ] FROM ] <имя курсора>
INTO <список переменных>
```

<направление перемещения указателя записи> – принимает следующие значения:

FIRST – на первую запись,
NEXT – на следующую запись,
PRIOR – на предыдущую запись,
LAST – на последнюю запись,
ABSOLUTE n – на n-ую запись от начала таблицы (n>0),
RELATIVE n – на n-ую запись от текущей записи (n может быть любого знака).

Пример

```
#define OK “00000”
#define EndOfScan “02000”
EXEC SQL BEGIN DECLARE SECTION;
```

```
        char SQLSTATE[6];
        unsigned long stud_id;
        char grade[1];
        char *crs_code, *semester;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE GetEnrolled INSENSITIVE CURSOR FOR
        Select T.StudId, T.Grade
        From Transcript T
        Where T.CrsCode = :crs_code AND T.Semester = :semester
        FOR READ ONLY;

EXEC SQL OPEN GetEnrolled;
if  ( strcmp(SQLSTATE, OK) != 0 ){
        printf("Cannot open cursor \n");
        exit(1);
}

EXEC SQL FETCH GetEnrolled INTO :stud_id, :grade;
        while (strcmp(SQLSTATE, OK) == 0 ){
        <операторы>
        EXEC SQL FETCH GetEnrolled INTO :stud_id, :grade;
}

if  ( strcmp(SQLSTATE, EndOfScan) != 0 ){
        printf("Something fishy: error before end-of-scan \n");
        exit(1);
}

EXEC SQL CLOSE GetEnrolled;
```

Команды COMMIT AND CHAIN, ROLLBACK AND CHAIN начинают новую транзакцию сразу же после завершения предыдущей, не дожидаясь оператора начала транзакции.

§ 2.4 *Динамически формируемые запросы*

Можно сформировать запрос в процессе работы программы. В этом случае необходимо сформированную строку преобразовать в SQL выражение с помощью команды PREPARE. А затем выполнить SQL выражение с помощью команды EXECUTE.

Пример

```
printf ("Which column of Class would you like to see?");
scanf ("%s", column); // get user input (Enrollment or Room)
// Incorporate user input into SQL statement
sprintf (my_sql_stmt, "Select C.%s FROM Class C Where C.Crs = ? And C.Semester = ?", column);
EXEC SQL PREPARE st1 FROM :my_sql_stmt;
EXEC SQL EXECUTE st1 INTO :some_string_var USING :crs_code, :semester;
```

PREPARE посылает строку с запросом менеджеру базы данных и преобразует ее в SQL запрос с именем «st1», который можно неоднократно использовать в EXEC SQL EXECUTE.

Запрос имеет два параметра, отмеченных символом «?». Вместо этих символов подставляются значения, указанные после USING.

Аналогично можно подготовить динамически формируемый курсор или вызов хранимой процедуры.

Пример

```
my_sql_stmt = "Select T.StudId, T.Grade \
FROM Transcript T \
Where T.CrsCode = ? And T.Semester = ?";
EXEC SQL PREPARE st2 FROM  :my_sql_stmt;
EXEC SQL DECLARE GetEnrolled INSENSITIVE CURSOR FOR st2;
EXEC SQL OPEN GetEnrolled;
EXEC SQL FETCH GetEnrolled INTO :stud_id, :grade;
```

```
        while (strcmp(SQLSTATE, OK) == 0 ){
        <операторы>
        EXEC SQL FETCH GetEnrolled INTO :stud_id, :grade;
}
EXEC SQL CLOSE GetEnrolled;
```

Пример

```
my_sql_stmt = “CALL Deregister(?, ?, ?)”;
EXEC SQL PREPARE st3 FROM  :my_sql_stmt;
EXEC SQL EXECUTE st3 USING :crs_code, :semester, :stud_id;
```

Динамические запросы менее эффективны, чем статические.

§ 3 JDBC

JDBC – это API интерфейс для работы с базой данных на языке программирования java. JDBC разработан Sun Microsystems и является составной частью java. Он представляет собой call-level интерфейс для выполнения запросов.

Приложение соединяется с СУБД с помощью JDBC модуля, называемого Менеджером драйверов, который выбирает соответствующий СУБД драйвер. Драйвер преобразует посылаемые приложением SQL команды в формат, соответствующий СУБД, и затем отсылает команду самой СУБД.

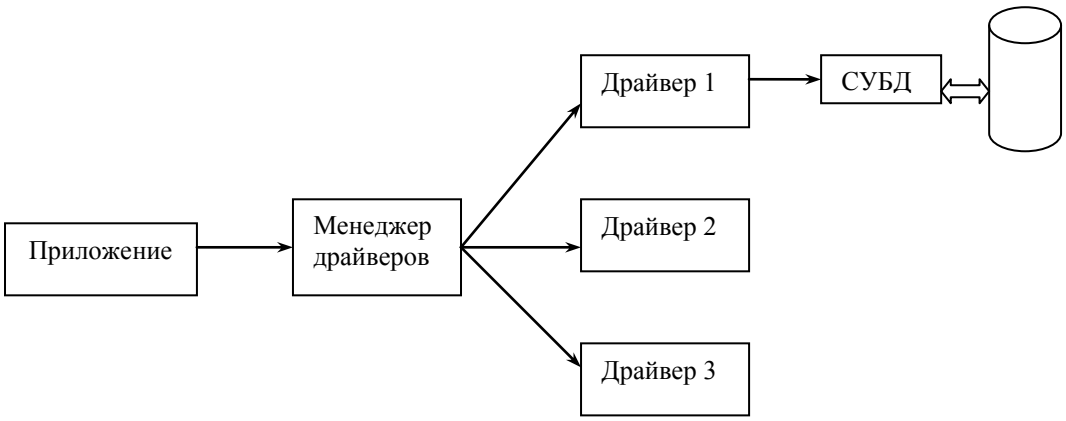


Рис. 1 – Схема соединения через JDBC

JDBC включает классы DriverManager и Statement, методы которых реализуют функции доступа к данным.

Пример

```
import java.sql.*
    Connection con1;
    try {
                // Use the right driver for your database
                // загружает драйвер
        Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
                // устанавливает соединение, которое сохраняется в переменной con1
                // DriverManager.getConnection(url, userId, password)
        con1 = DriverManager.getConnection( “jdbc:odbc:http://server.com/StudentDB:800”, ”Admin”,
”qwerty”);
    } catch (ClassNotFoundExeption e) {
        System.err.println(“Cannot load driver\n”);
        System.exit(1);
    } catch (SQLExeption e) {
        System.err.println(“Cannot connect\n”);
        System.exit(1);
    }
```

```
    // создает объект для выполнения запроса
Statement stat1 = con1.createStatement();
String myQuery = “Select * From tblStudent”
// выполняет запрос executeQuery или executeUpdate
ResultSet res1 = stat1.executeQuery(myQuery);
    // обрабатывает результат запроса res1
long studId;
String studName;
    // res1.next() – перемещение курсора на следующую запись
    // возвращает false, когда записи закончились
while (res1.next()) {
    studId = res1.getLong(“intStudentNumber”);
    studName = res1.getString(“txtStudentName”); }
    // уничтожает объект
stat1.close();
    // выполняет отсоединение
con1.close();
```

Классы Connection, Statement, DriverManager находятся в пакете «java.sql.*»

По умолчанию каждый объект Statement представляет собой отдельную транзакцию. Чтобы создать другие транзакции, необходимо отключить режим автоматической фиксации с помощью команды: «con.setAutoCommit(false);». Фиксация и откат транзакции выполняются посредством команд: «con.commit();» «con.rollback();».

§ 4 SQLJ

Программы, использующие call-level interface, работают медленно, поэтому для java был разработан statement-level interface, называемый SQLJ, поскольку подготовка и выполнение SQL команд требуют разного взаимодействия с СУБД.

SQLJ – аналог встроенного SQL для java программ. Программы транслируются предкомпилятором, который заменяет команды SQL на вызовы run-time пакета, который обращается к СУБД через JDBC драйвера. Предкомпилятор проверяет правильность SQL команд.

```
import java.sql.*
        // объект iterator содержит данные, получаемые запросом,
        // и является курсором
        // определяемый ниже объект включает два столбца
#SQL iterator GetEnrolledIter(int studentId, String studentName);
GetEnrolledIter iter1;

int year;
year=2;
        // java переменные могут быть включены как параметры
        // в SQL запрос с префиксом «:»
        // AS ассоциирует имена атрибутов со столбцами объекта
#SQL iter1 = {Select T.intStudentNumber AS “studentId”, T.txtStudentName AS “studentName”
From tblStudent T Where T.intStudyYear = :year}

int id;
String studName;
while (iter1.next()) {
        id = iter.studentId();
        studName = iter1.studentName();
}
iter1.close()
```

§ 5 ODBC

ODBC – это API интерфейс для работы с базой данных на уровне call-level интерфейса. Архитектура ODBC похожа на архитектуру JDBC. В отличие от JDBC ODBC не является объектно-ориентированной и взаимодействует с СУБД на самом низком уровне, поэтому необходимо использовать функции для выделения и освобождения памяти при выполнении SQL команд.

Глава 17. Безопасность

Аспекты безопасности

- правовые, общественные, этические (имеет ли право некоторое лицо получить информацию о вкладе);
- физические условия (закрыт ли данный компьютер);
- организационные (организация доступа);
- аппаратное обеспечение (меры безопасности на аппаратном уровне);
- безопасность операционной системы;
- безопасность СУБД.

В любой базе данных существует конфиденциальная информация, доступ к которой может быть разрешен лишь ограниченному кругу лиц. Можно сформулировать следующие требования к безопасности:

- Данные должны быть доступны не всем пользователям.
- Одним пользователям разрешено обновлять данные, а другим лишь просматривать.

В современных СУБД поддерживается один из двух широко распространенных подходов к вопросу обеспечения безопасности данных:

- избирательный подход;
- обязательный подход.

§ 1 Избирательное управление доступом

Когда пользователь создает какой-либо объект базы данных (таблица, домен, процедура и т.п.), он становится владельцем этого объекта. Обычно эти функции выполняет администратор. Остальные пользователи могут получить **права** (привилегии или полномочия) для работы с объектом. Разные пользователи обладают разными правами доступа к одному и тому же объекту.

Право доступа включает:

- кому предоставляется доступ,
- к каким объектам,
- какие операции можно выполнять над объектом.

Кроме этого, правило может разрешать назначение привилегий другим пользователям.

Права устанавливаются и отменяются специальными операторами языка SQL:

```
GRANT <действия> ON <объект> TO <пользователь> [WITH GRANT OPTION]
REVOKE [GRANT OPTION FOR] <действия> ON <объект> FROM <пользователь> {RESTRICT
| CASCADE}
<действия> = {USAGE, INSERT, DELETE, UPDATE, SELECT}
```

Примеры:

- GRANT SELECT ON Detail, Delivery TO Anna, Elena
- GRANT SELECT, UPDATE (Detail, Weight), DELETE ON Detail TO Misha WITH GRANT OPTION
- REVOKE DELETE ON Detail FROM Misha CASCADE

Для современных баз данных с большим количеством пользователей актуальна проблема их объединения в группы.

§ 2 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.

Каждому объекту данных присваивается некоторый **классификационный уровень**, например: секретно, совершенно секретно, для служебного пользования и т.д., а каждый пользователь обладает некоторым **уровнем допуска** с такими же градациями, что и в уровне классификации. Предполагается, что эти уровни образуют строгий иерархический порядок. Тогда на основе этих сведений можно сформулировать два очень простых правила безопасности:

- Пользователь имеет доступ к объекту, только если его уровень допуска больше или равен уровню классификации объекта.
- Пользователь может модифицировать объект, только если его уровень допуска равен уровню классификации объекта. Любая информация, записанная пользователем, автоматически приобретает уровень допуска, равный уровню классификации пользователя.

Глава 18. Хранилища данных и технология OLAP

§ 1 Введение

Хранилище данных (Data Warehouse) – предметно-ориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для поддержки принятия решений (место для данных, предназначенных для анализа). Основная характеристика – цель использования данных.

Витрина данных (data mart) – подмножество хранилища данных, организованное для решения аналитических задач конкретного подразделения фирмы (т.е. по одному из направлений её деятельности) или конкретных пользователей.

Хранилище данных содержит непротиворечивые консолидированные исторические данные и предоставляет инструментальные средства для их анализа с целью поддержки принятия стратегических решений. Информационные ресурсы хранилища данных формируются на основе фиксируемых на протяжении продолжительного периода времени моментальных снимков баз данных оперативной информационной системы и, возможно, различных внешних источников.

Обычно данные получаются в результате ежедневных операций, связанных с некоторым процессом (производственный процесс, обучение в школе, наблюдения и т.д.). Такие системы называются online transaction processing (OLTP). Данные в таких системах организованы так, чтобы получить быстрый доступ к ним (быстрый доступ к небольшому объему данных).

В хранилищах данных выполняются операции по анализу данных, которые могут занимать продолжительное время и охватывать очень большие объемы данных.

В хранилищах данных применяются технологии баз данных, OLAP, глубинного анализа данных, визуализации данных.

Основные характеристики хранилищ данных:

- содержит исторические данные;
- хранит подробные сведения, а также частично и полностью обобщенные данные;
- данные в основном являются статическими;
- нерегламентированный, неструктурированный и эвристический способ обработки данных;
- средняя и низкая интенсивность обработки транзакций;
- непредсказуемый способ использования данных;
- предназначено для проведения анализа;
- ориентировано на предметные области;
- поддержка принятия стратегических решений;
- обслуживает относительно малое количество работников руководящего звена.

Термин **OLAP** (On-Line Analytical Processing) служит для описания модели представления данных и соответственно технологии их обработки в хранилищах данных.

В OLAP применяется многомерное представление агрегированных данных для обеспечения быстрого доступа к стратегически важной информации в целях углубленного анализа. Приложения OLAP должны обладать следующими основными свойствами:

- многомерное представление данных;
- поддержка сложных расчетов;
- правильный учет фактора времени.

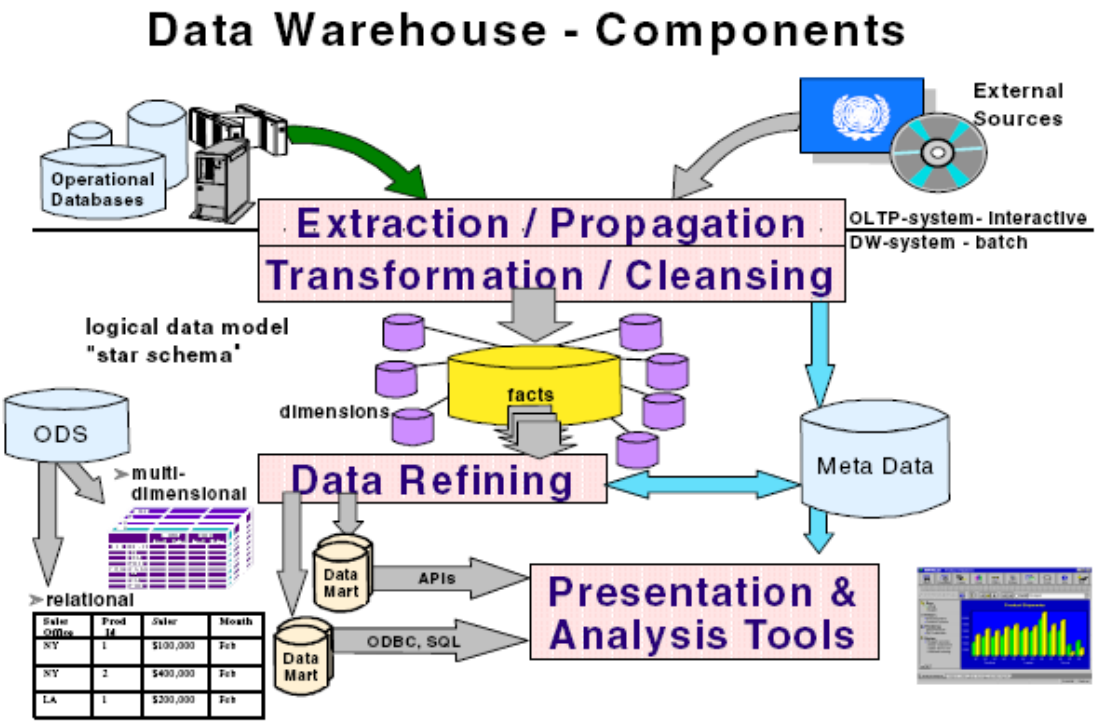


Figure 10. Data warehouse components

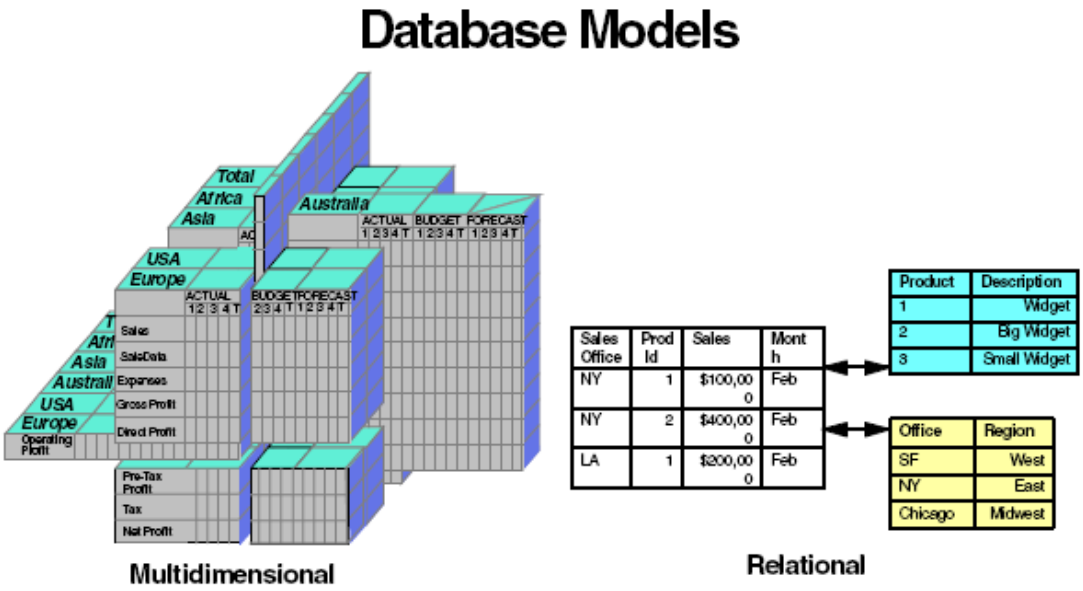


Figure 12. Physical database models

§ 2 Правила Кодда для OLAP систем

В 1993 году Кодд опубликовал труд под названием «OLAP для пользователей-аналитиков: каким он должен быть». В нем он изложил основные концепции оперативной аналитической обработки и определил 12 правил, которым должны удовлетворять продукты, предоставляющие возможность выполнения оперативной аналитической обработки.

Концептуальное многомерное представление. OLAP-модель должна быть многомерной в своей основе. Многомерная концептуальная схема или пользовательское представление облегчают моделирование и анализ так же, впрочем, как и вычисления.

Прозрачность. Пользователь способен получить все необходимые данные из OLAP-машины, даже не подозревая, откуда они берутся. Вне зависимости от того, является OLAP-продукт частью средств пользователя или нет, этот факт должен быть незаметен для пользователя. Если OLAP предоставляется клиент-серверными вычислениями, то этот факт также, по возможности, должен быть невидим для пользователя. OLAP должен предоставляться в контексте истинно открытой архитектуры, позволяя

пользователю, где бы он ни находился, связываться при помощи аналитического инструмента с сервером. В дополнение к этому прозрачность должна достигаться и при взаимодействии аналитического инструмента с гомогенной и гетерогенной средами БД.

Доступность. OLAP должен предоставлять свою собственную логическую схему для доступа в гетерогенной среде БД и выполнять соответствующие преобразования для предоставления данных пользователю. Более того, необходимо заранее позаботиться о том, где и как, и какие типы физической организации данных действительно будут использоваться. OLAP-система должна выполнять доступ только к действительно требующимся данным, а не применять общий принцип «кухонной воронки», который влечет ненужный ввод.

Постоянная производительность при разработке отчетов. Производительность формирования отчетов не должна существенно падать с ростом количества измерений и размеров базы данных.

Клиент-серверная архитектура. Требуется, чтобы продукт был не только клиент-серверным, но и чтобы серверный компонент был бы достаточно интеллектуальным для того, чтобы различные клиенты могли подключаться с минимумом усилий и программирования.

Общая многомерность. Все измерения должны быть равноправны, каждое измерение должно быть эквивалентно и в структуре, и в операционных возможностях. Правда, допускаются дополнительные операционные возможности для отдельных измерений (видимо, подразумевается время), но такие дополнительные функции должны быть предоставлены любому измерению. Не должно быть так, чтобы базовые структуры данных, вычислительные или отчетные форматы были более свойственны какому-то одному измерению.

Динамическое управление разреженными матрицами. OLAP системы должны автоматически настраивать свою физическую схему в зависимости от типа модели, объемов данных и разреженности базы данных.

Многопользовательская поддержка. OLAP-инструмент должен предоставлять возможности совместного доступа (запроса и дополнения), целостности и безопасности.

Неограниченные перекрестные операции. Все виды операций должны быть дозволены для любых измерений.

Интуитивная манипуляция данными. Манипулирование данными осуществлялось посредством прямых действий над ячейками в режиме просмотра без использования меню и множественных операций.

Гибкие возможности получения отчетов. Измерения должны быть размещены в отчете так, как это нужно пользователю.

Неограниченная размерность и число уровней агрегации. Исследование о возможном числе необходимых измерений, требующихся в аналитической модели, показало, что одновременно может использоваться до 19 измерений. Отсюда вытекает настоятельная рекомендация, чтобы аналитический инструмент был способен одновременно предоставить как минимум 15 измерений, а предпочтительнее 20. Более того, каждое из общих измерений не должно быть ограничено по числу определяемых пользователем-аналитиком уровней агрегации и путей консолидации.

§ 3 Основные элементы и операции OLAP

В основе OLAP лежит понятие гиперкуба, или многомерного куба данных, в ячейках которого хранятся анализируемые данные.

Факт – это числовая величина, которая располагается в ячейках гиперкуба. Один OLAP-куб может обладать одним или несколькими показателями.

Измерение (dimension) – это множество объектов одного или нескольких типов, организованных в виде иерархической структуры и обеспечивающих информационный контекст числового показателя. Измерение принято визуализировать в виде ребра многомерного куба.

Объекты, совокупность которых и образует измерение, называются членами измерений (members). Члены измерений визуализируют как точки или участи, откладываемые на осях гиперкуба.

Ячейка (cell) – атомарная структура куба, соответствующая полному набору конкретный значений измерений.

Иерархия – группировка объектов одного измерения в объекты более высокого уровня. Например – день-месяц-год. Иерархии в измерениях необходимы для возможности агрегации и детализации значений показателей согласно их иерархической структуре. Иерархия целиком основывается на одном измерении и формируется из уровней.

В OLAP-системах поддерживаются следующие базовые операции:

- **поворот;**
- **проекция** (При проекции значения в ячейках, лежащих на оси проекции, суммируются по некоторому предопределенному закону);
- **раскрытие** (drill-down) (Одно из значений измерения заменяется совокупностью значений из следующего уровня иерархии измерения; соответственно заменяются значения в ячейках гиперкуба);
- **свертка** (roll-up/drill-up) (Операция, обратная раскрытию);

- **сечение** (slice-and-dice).

§ 4 Типы OLAP

Выбор способа хранения данных зависит от объема и структуры детальных данных, требований к скорости выполнения запросов и частоты обновления OLAP-кубов. В настоящее время применяются три способа хранения данных.

- MOLAP (Multidimensional OLAP)
- ROLAP (Relational OLAP)
- HOLAP (Hybrid OLAP)

§ 4.1 *MOLAP*

В MOLAP-системах детальные и агрегированные данные хранятся в многомерной базе данных. Хранение данных в многомерных структурах позволяет манипулировать данными как многомерным массивом, благодаря чему скорость вычисления агрегатных значений одинакова для любого из измерений. Однако в этом случае многомерная база данных оказывается избыточной, так как многомерные данные полностью содержат детальные реляционные данные.

Преимущества MOLAP

- Высокая производительность. Поиск и выборка данных осуществляется значительно быстрее, чем при многомерном концептуальном взгляде на реляционную базу данных.
- Структура и интерфейсы наилучшим образом соответствуют структуре аналитических запросов.
- Многомерные СУБД легко справляются с задачами включения в информационную модель разнообразных встроенных функций.

Недостатки MOLAP

- MOLAP могут работать только со своими собственными многомерными БД и основываются на патентованных технологиях для многомерных СУБД, поэтому являются наиболее дорогими. Эти системы обеспечивают полный цикл OLAP-обработки и либо включают в себя, помимо серверного компонента, собственный интегрированный клиентский интерфейс, либо используют для связи с пользователем внешние программы работы с электронными таблицами.
- По сравнению с реляционными, очень неэффективно используют внешнюю память, обладают худшими по сравнению с реляционными БД механизмами транзакций.
- Отсутствуют единые стандарты на интерфейс, языки описания и манипулирования данными.
- Не поддерживают репликацию данных, часто используемую в качестве механизма загрузки.

§ 4.2 *ROLAP*

ROLAP-системы позволяют представлять данные, хранимые в классической реляционной базе, в многомерной форме или в плоских локальных таблицах на файл-сервере, обеспечивая преобразование информации в многомерную модель через промежуточный слой метаданных. Агрегаты хранятся в той же БД в специально созданных служебных таблицах. В этом случае гиперкуб эмулируется СУБД на логическом уровне.

Преимущества ROLAP

- Реляционные СУБД имеют реальный опыт работы с очень большими БД и развитые средства администрирования. При использовании ROLAP размер хранилища не является таким критичным параметром, как в случае MOLAP.
- При оперативной аналитической обработке содержимого хранилища данных инструменты ROLAP позволяют производить анализ непосредственно над хранилищем (потому что в подавляющем большинстве случаев корпоративные хранилища данных реализуются средствами реляционных СУБД).
- В случае переменной размерности задачи, когда изменения в структуру измерений приходится вносить достаточно часто, ROLAP системы с динамическим представлением размерности являются оптимальным решением, так как в них такие модификации не требуют физической реорганизации БД, как в случае MOLAP.

- Системы ROLAP могут функционировать на гораздо менее мощных клиентских станциях, чем системы MOLAP, поскольку основная вычислительная нагрузка в них ложится на сервер, где выполняются сложные аналитические SQL-запросы, формируемые системой.
- Реляционные СУБД обеспечивают значительно более высокий уровень защиты данных и хорошие возможности разграничения прав доступа.

Недостатки ROLAP

- Ограниченные возможности с точки зрения расчета значений функционального типа.
- Меньшая производительность, чем у MOLAP. Для обеспечения сравнимой с MOLAP производительности реляционные системы требуют тщательной проработки схемы БД и специальной настройки индексов. Но в результате этих операций производительность хорошо настроенных реляционных систем при использовании схемы "звезда" сравнима с производительностью систем на основе многомерных БД.

§ 4.3 *HOLAP*

Детальные данные остаются в той же реляционной базе данных, где они изначально находились, а агрегатные данные хранятся в многомерной базе данных.

§ 5 Моделирование многомерных кубов на реляционной модели данных

§ 5.1 *Схема «Звезда»*

Схема типа звезды (Star Schema) – схема реляционной базы данных, служащая для поддержки многомерного представления содержащихся в ней данных.

Особенности ROLAP-схемы типа «звезда»

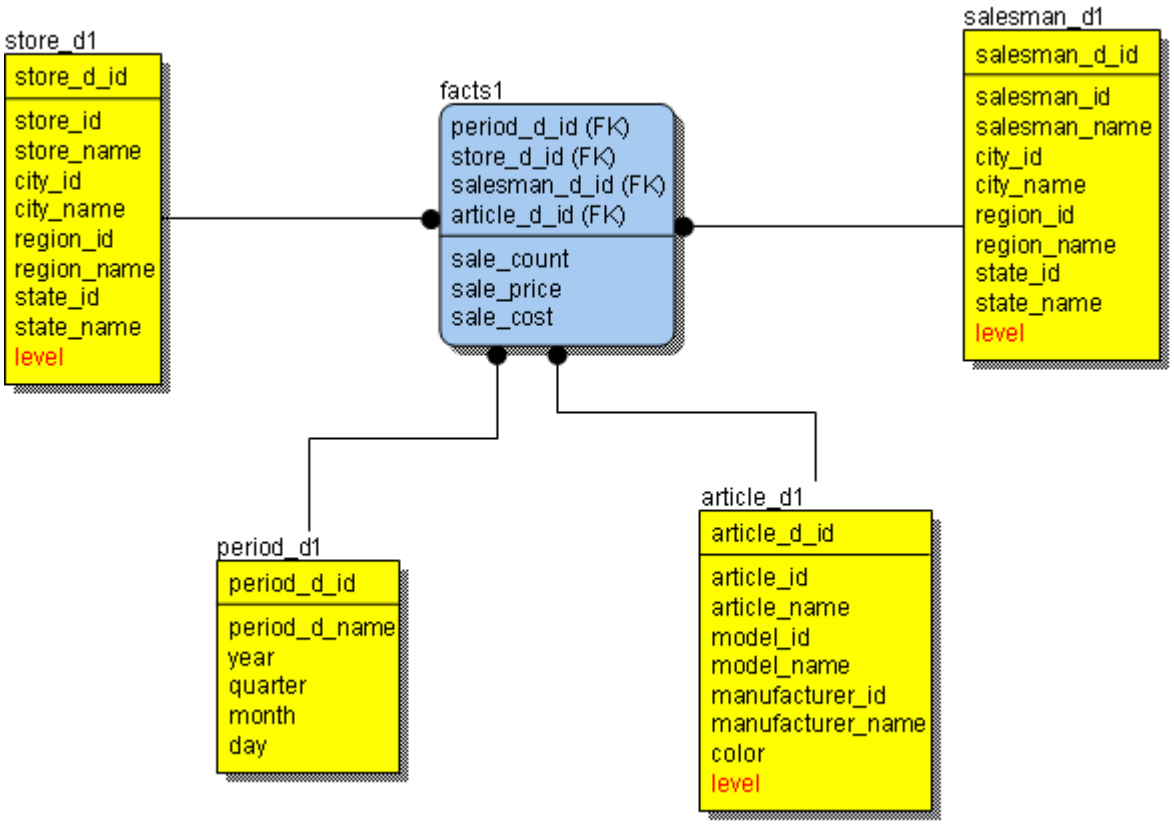
- Одна таблица фактов (fact table), которая сильно денормализована. Является центральной в схеме, может состоять из миллионов строк и содержит суммируемые или фактические данные, с помощью которых можно ответить на различные вопросы.
- Несколько денормализованных таблиц измерений (dimensional table). Имеют меньшее количество строк, чем таблицы фактов, и содержат описательную информацию. Эти таблицы позволяют пользователю быстро переходить от таблицы фактов к дополнительной информации.
- Таблица фактов и таблицы размерности связаны идентифицирующими связями, при этом первичные ключи таблицы размерности мигрируют в таблицу фактов в качестве внешних ключей. Первичный ключ таблицы факта целиком состоит из первичных ключей всех таблиц размерности.
- Агрегированные данные хранятся совместно с исходными.

Преимущества

Благодаря денормализации таблиц измерений упрощается восприятие структуры данных пользователем и формулировка запросов, уменьшается количество операций соединения таблиц при обработке запросов. Некоторые промышленные СУБД и инструменты класса OLAP / Reporting умеют использовать преимущества схемы "звезда" для сокращения времени выполнения запросов.

Недостатки

Денормализация таблиц измерений вносит избыточность данных, возрастает требуемый для их хранения объем памяти. Если агрегаты хранятся совместно с исходными данными, то в измерениях необходимо использовать дополнительный параметр - уровень иерархии.



§ 5.2 *Схема «Снежинка»*

Схема типа снежинки (Snowflake Schema) – схема реляционной базы данных, служащая для поддержки многомерного представления содержащихся в ней данных, является разновидностью схемы типа «звезда» (Star Schema).

Особенности ROLAP-схемы типа «снежинка»

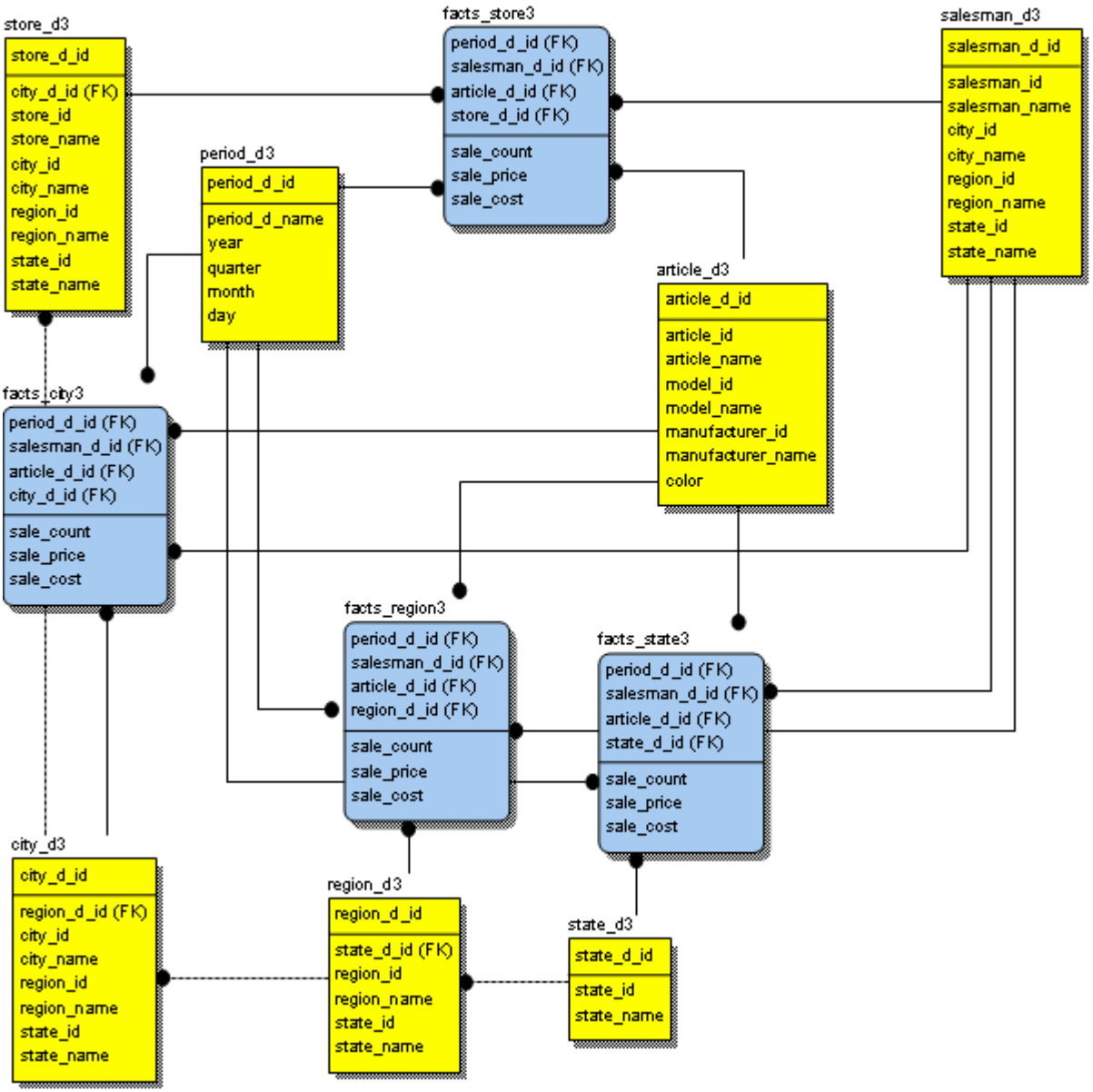
- Одна таблица фактов (fact table), которая сильно денормализована. Является центральной в схеме, может состоять из миллионов строк и содержать суммируемые или фактические данные, с помощью которых можно ответить на различные вопросы.
- Несколько таблиц измерений (dimensional table), которые нормализованы в отличие от схемы "звезда". Имеют меньшее количество строк, чем таблицы фактов, и содержат описательную информацию. Эти таблицы позволяют пользователю быстро переходить от таблицы фактов к дополнительной информации. Первичные ключи в них состоят из единственного атрибута (соответствуют единственному элементу измерения).
- Таблица фактов и таблицы размерности связаны идентифицирующими связями, при этом первичные ключи таблицы размерности мигрируют в таблицу фактов в качестве внешних ключей. Первичный ключ таблицы факта целиком состоит из первичных ключей всех таблиц размерности.
- В схеме "снежинка" агрегированные данные могут храниться отдельно от исходных.

Преимущества

Нормализация таблиц измерений в отличие от схемы "звезда" позволяет минимизировать избыточность данных и более эффективно выполнять запросы, связанные со структурой значений измерений.

Недостатки

За нормализацию таблиц измерений иногда приходится платить временем выполнения запросов.


store_d3
store_d_id
city_d_id (FK)
store_id
store_name
city_id
city_name
region_id
region_name
state_id
state_name
facts_store3
period_d_id (FK)
salesman_d_id (FK)
article_d_id (FK)
store_d_id (FK)
sale_count
sale_price
sale_cost
salesman_d3
salesman_d_id
salesman_id
salesman_name
city_id
city_name
region_id
region_name
state_id
state_name
period_d3
period_d_id
period_d_name
year
quarter
month
day
article_d3
article_d_id
article_id
article_name
model_id
model_name
manufacturer_id
manufacturer_name
color
facts_city3
period_d_id (FK)
salesman_d_id (FK)
article_d_id (FK)
city_d_id (FK)
sale_count
sale_price
sale_cost
facts_region3
period_d_id (FK)
salesman_d_id (FK)
article_d_id (FK)
region_d_id (FK)
sale_count
sale_price
sale_cost
facts_state3
period_d_id (FK)
salesman_d_id (FK)
article_d_id (FK)
state_d_id (FK)
sale_count
sale_price
sale_cost
city_d3
city_d_id
region_d_id (FK)
city_id
city_name
region_id
region_name
state_id
state_name
region_d3
region_d_id
state_d_id (FK)
region_id
region_name
state_id
state_name
state_d3
state_d_id
state_id
state_name