Язык SQL

Вводная лекция
Введение в теорию баз данных

Е. П. Моргунов
Сибирский государственный университет науки и технологий
имени академика М. Ф. Решетнева
г. Красноярск
Институт информатики и телекоммуникаций
emorgunov@mail.ru

Компания Postgres Professional
г. Москва

Версия от 18.10.2018

0.1. Основы и история

0.1.1. Подход на основе файлов

Данные хранились в плоских файлах (flat files). Структура: записи, разделенные на поля фиксированной длины.

Проблемы:

- Трудно организовать совместную обработку данных из разных файлов.
- Дублирование данных (это затраты времени на повторный ввод и обработку данных, расход дисковой памяти на их хранение, рассогласование данных, которые в одном месте могли быть изменены, а в другом – нет).
- Зависимость приложений от данных (физическая структура и методы доступа к данным определялись непосредственно в программном коде, поэтому при изменении структуры данных необходимо откорректировать все модули приложения, в которых эти данные используются).
- Невозможность выполнения разовых (ad hoc) запросов, которые изначально не были предусмотрены.
- Нет гарантий защищенности и целостности.
- Проблемы с восстановлением данных после сбоя.
- Сложность организации совместного доступа к данным.
- Нет никакого контроля за доступом к данным, кроме того, который налагается прикладными программами.

0.1.2. Подход на основе баз данных (1)

- **База данных** – коллекция логически связанных данных и их описаний, предназначенных для удовлетворения информационных потребностей предприятия (организации).
- **База данных** — это некоторый набор перманентных (постоянно хранимых) данных, используемых прикладными программными системами какого-либо предприятия.
- **Описания данных** – системный каталог, или словарь данных, или метаданные (данные о данных).
- **Система управления базами данных (СУБД)** – программная система, позволяющая пользователям определять, создавать, поддерживать данные и управлять доступом к базе данных.
- **Система баз данных** — это компьютеризированная система хранения записей, т. е. компьютеризированная система, основное назначение которой — хранить информацию, предоставляя пользователям средства ее извлечения и модификации.

0.1.2. Подход на основе баз данных (2)

Главные компоненты системы: данные, аппаратное обеспечение, программное обеспечение, процедуры и пользователи.

- **Данные**
  Данные в БД являются интегрированными и разделяемыми.
  Интегрированные – объединяют различные данные, исключается избыточность хранения данных.
  Разделяемые – обеспечивается совместный доступ к данным (возможно, параллельный).
- **Аппаратное обеспечение**
- **Программное обеспечение**
  - сервер базы данных (database server) или система управления базами данных, СУБД (Database Management System — DBMS)
  - утилиты
  - средства разработки приложений
  - средства проектирования
  - генераторы отчетов

0.1.2. Подход на основе баз данных (3)

- **Процедуры**
  - вход в систему
  - запуск и останов сервера баз данных
  - выполнение резервного копирования базы данных
  - обработка сбоев и т. д.
- **Пользователи**
  - прикладные программисты
  - конечные пользователи
  - администраторы баз данных (АБД) (database administrators (DBA))
  - администраторы данных (АД) (data administrators (DA))
  - проектировщики базы данных
- **Администратор данных**
  - администратор должен разбираться в данных
  - понимать нужды предприятия по отношению к данным *на уровне высшего управляющего звена* в руководстве предприятием
- **Администратор *базы* данных,** или **АБД**

  Администратор базы данных, в отличие от администратора данных, должен быть профессиональным специалистом в области информационных технологий. Это *технический* специалист, ответственный за реализацию решений администратора данных. Он отвечает за физическую реализацию БД, обеспечение целостности, защищенности, производительности.

0.1.3. История баз данных

- Середина 1960-х гг. Иерархические СУБД (IMS компании IBM)
- Середина 1960-х гг. Сетевые СУБД (IDMS/R компании Computer Associates)
- 1970 г. Доктор E. F. Codd предложил реляционную модель.
- 1970-е гг. Предложены первые реляционные СУБД (Ingres в Berkeley и System R в компании IBM, язык SQL).
- 1976 г. П. Чен предложил модель «сущность–связь» (entity–relation) – ER-модель.
- 1979 г. Появление коммерческих реляционных СУБД Oracle, Ingres, DB2
- 1987 г. Стандарт ISO языка SQL (последующие выпуски стандарта: 1989, 1992 (SQL2), 1999 (SQL:1999), 2003 (SQL:2003), 2008 (SQL:2008), 2011 (SQL:2011)
- 1990-е гг. Появление объектно-ориентированных СУБД и объектно-реляционных СУБД
- 1990-е гг. Появление хранилищ данных (data warehousing)
- Середина 1990-х гг. Интеграция web с базами данных
- Середина 1990-х гг. – по настоящее время. Развитие open source СУБД (PostgreSQL, MySQL и др.)

0.1.4. Преимущества баз данных

- Возможность совместного доступа к данным
- Сокращение избыточности данных (допустима контролируемая избыточность)
- Обеспечение согласованности данных (Data consistency), т. е. устранение противоречивости данных (до некоторой степени) (это следствие предыдущего пункта)
- Возможность поддержки транзакций при параллельной работе разных пользователей и программ

  Транзакция (transaction) — это логическая единица работы.
- Обеспечение целостности данных
- Организация защиты данных
- Возможность введения стандартизации
- Обеспечение независимости от данных (data independency)
- Экономия на масштабе за счет централизации данных
- Улучшенный доступ к данным (возможность писать ad hoc запросы)
- Более надежные процедуры резервного копирования данных и восстановления после сбоев

0.1.5. Недостатки баз данных

- Сложность (нужны специалисты)
- Большой масштаб (следствие – требуется мощное аппаратное обеспечение, много памяти, быстрые процессоры)
- Высокая стоимость коммерческих СУБД (но есть альтернатива – свободное ПО, например, PostgreSQL)
- Высокая стоимость перехода от устаревших систем, основанных на файлах, к современным СУБД
- Более низкая скорость работы, чем у специализированных приложений, написанных с учетом специфики решаемой задачи
- Больший масштаб проблем в случае сбоя сервера баз данных

0.2. Системы баз данных

0.2.1. Архитектура ANSI/SPARC (1)

- В 1971 г. подразделением DBTG (Data Base Task Group) организации CODASYL (Conference on Data Systems Language) был предложен двухуровневый подход: системное представление (system view), называемое *схемой* и пользовательские представления (user views), называемые *подсхемами.*
- Комитет под названием Standards Planning and Requirements Committee (SPARC) Американского национального института стандартов (The American National Standards Institute (ANSI)), т. е. ANSI/X3/SPARC, в 1975 г. предложил трехуровневую архитектуру.
- Хотя архитектура ANSI/SPARC не стала стандартом, она является базисом для понимания функциональности СУБД.
- Архитектура ANSI/SPARC включает три уровня: внутренний, концептуальный, внешний.

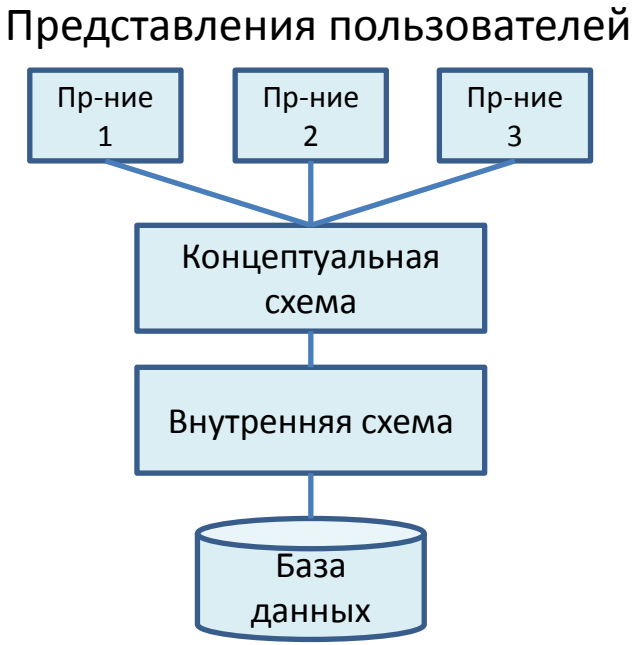
0.2.1. Архитектура ANSI/SPARC (2)

Внешний уровень

Концептуальный уровень

Внутренний уровень

Физическая организация данных



0.2.1.1. Уровни архитектуры ANSI/SPARC (1)

Внешний уровень (*пользовательский логический)* наиболее близок к пользователям, т. е. связан со способами представления данных для отдельных пользователей.

- Одни и те же данные могут быть представлены разным пользователям по-разному.

Концептуальный уровень (называемый также *общим логическим* или просто *логическим,* без дополнительного определения) является «промежуточным» уровнем между двумя первыми. Он описывает, какие данные хранятся в базе данных.

- Если внешний уровень связан с *индивидуальными* представлениями пользователей, то концептуальный уровень связан с *обобщенным* представлением пользователей. Концептуальное представление — это представление всего содержимого базы данных. Оно представляет:
  - все сущности, их атрибуты и связи между ними
  - ограничения, накладываемые на данные
  - смысловую информацию о данных
  - информацию о защищенности данных и их целостности

0.2.1.1. Уровни архитектуры ANSI/SPARC (2)

Внутренний уровень (иногда называемый также *физическим*. См. ниже) наиболее близок к физическому хранилищу информации, т. е. связан со способами сохранения информации на физических устройствах. Он описывает, каким образом данные хранятся в базе данных.

- Внутренний уровень отвечает за следующие вещи:
  - выделение пространства для хранения данных и индексов
  - описания хранимых записей (с указанием размеров хранимых элементов данных)
  - размещение записей
  - методы сжатия и шифрования данных

Ниже внутреннего уровня лежит **физический уровень.** Он может реализовываться операционной системой при участии СУБД.

0.2.1.2. Схемы, отображения, экземпляры

- Общее описание базы данных – **схема** базы данных (intension).
- Совокупность данных в базе данных в конкретный момент времени – **экземпляр** базы данных (extension).
- В соответствии с уровнями абстракции:
  - внешние схемы (подсхемы). Их может быть много (больше одной)
  - концептуальная схема
  - внутренняя схема
- СУБД отвечает за отображение между этими схемами:
  - «концептуальная/внутренняя»
  - «внешняя/концептуальная»

0.2.1.3. Независимость от данных

- Приложения зависимы от данных, если сведения об организации данных и способе доступа к ним встроены в саму логику и программный код приложения.
- **Независимость от данных** можно определить как невосприимчивость приложений к изменениям в физическом представлении данных и в методах доступа к ним.
- **Хранимое поле** — это наименьшая единица хранимых данных.

  Следует различать тип поля и экземпляр поля.
- **Хранимая запись** — это набор взаимосвязанных хранимых полей.

  Следует различать тип записи и экземпляр записи.
- **Хранимый файл** — это набор всех существующих в настоящий момент экземпляров хранимых записей одного и того же типа.
- В базах данных *логическая* (с точки зрения разработчика приложения) запись не всегда совпадает с соответствующей *хранимой* записью.
- База данных должна иметь возможность **расти** и развиваться, не нарушая логическую структуру существующих приложений. Например, добавленные новые хранимые поля будут невидимы для существующих приложений.
- Главная цель трехуровневой архитектуры – обеспечение независимости от данных.

0.2.2. Языки баз данных

- Используется термин подъязык данных (sublanguage).
- Любой подъязык данных является комбинацией по крайней мере двух подчиненных языков:
  - языка определения данных (Data Definition Language — DDL), который позволяет формулировать определения или объявления объектов базы данных (таблиц, представлений, индексов и др.)
  - языка манипулирования данными (Data Manipulation Language — DML), который поддерживает операции с такими объектами или их обработку:
    - добавление данных
    - изменение
    - выборка (извлечение)
    - удаление
- Подъязык данных – как правило, SQL (Structured Query Language).
- Подъязык данных может быть встроен в базовый язык или использоваться интерактивно.
- DML как правило бывают непроцедурными, т. е. требуют только указать, *какие* данные требуется получить, но не нужно указывать, *каким образом* это сделать.

0.2.3. Данные и модели данных

- **Модель данных** — интегрированная совокупность концепций для описания данных (и манипулирования ими), взаимосвязей между данными и ограничений на данные, имеющих место в организации.
- Модель данных включает три компонента:
  - Структурная часть — правила конструирования базы данных
  - Манипуляционная часть — определяет типы операций, которые допустимо выполнять над данными
  - Множество ограничений целостности — требования, предъявляемые к данным и их соотношениям
- Все модели данных можно разделить на три группы:
  - объектные (object-based)
  - основанные на записях (record-based)
  - физические (physical) модели
- **Физические модели данных**
  - unifying model
  - frame memory

0.2.3.1. Объектные модели данных

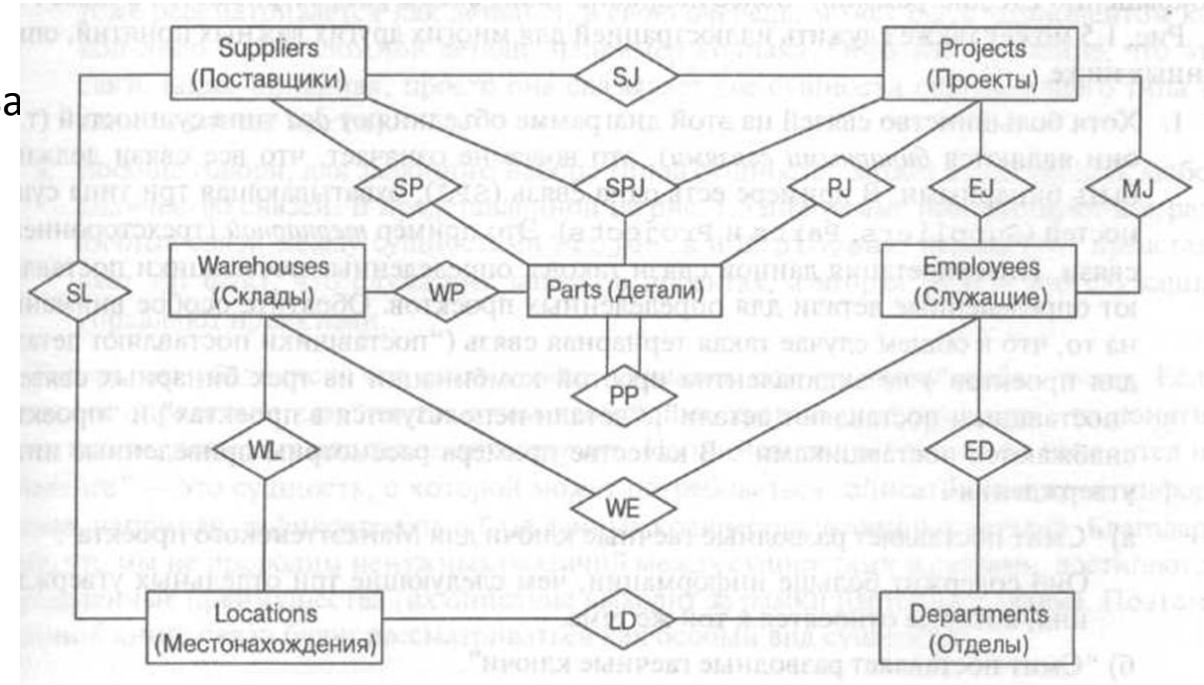
Эти модели основаны на следующих понятиях:

- **Сущность** – любой объект, информация о котором может (должна) быть представлена в базе данных.
- **Связь** – соединяет две или более сущностей и указывает вид взаимодействия между ними. Связи также должны быть представлены в базе данных. Связь можно понимать как сущность особого типа.
- **Свойства** – описывают (детализируют) сущности и связи.

Виды моделей

- Entity–Relationship (ER-модель)
- Семантические
- Функциональные
- Объектно-ориентирова

Диаграмма «сущность–связь» (ER-диаграмма)



0.2.3.2. Модели данных, основанные на записях (1)

- В таких моделях база данных состоит из некоторого количества записей фиксированного формата, возможно, различных типов.
- Каждый тип записей определяет фиксированное число полей, возможно, фиксированной длины.
- Три главных типа таких моделей:
  - реляционная (relational data model)
  - сетевая (network data model)
  - иерархическая (hierarchical data model)

0.2.3.2. Модели данных, основанные на записях (2)

- **Иерархическая модель.** Данные представлены в виде коллекций **записей**, а связи представлены множествами (sets). Записи организованы в виде дерева. При этом записи являются узлами (nodes) графа, а связи – ребрами. Связи на уровне реализации представлены в виде указателей. В отличие от сетевой модели, каждый узел может иметь только один родительский узел.
- **Сетевая модель.** Является *расширением* иерархической модели. Записи организованы в виде графа. Каждый узел может иметь более одного родительского узла.
- Эти две модели требуют от пользователя знать физическую организацию базы данных, чтобы работать с данными. Таким образом снижается уровень независимости от данных.
- Системы, реализованные на основе таких моделей, являются *навигационными*. Это означает, что для получения данных нужно указать, каким образом их нужно извлекать.
- В настоящее время СУБД, построенные на основе иерархической и сетевой моделей, встречаются редко.

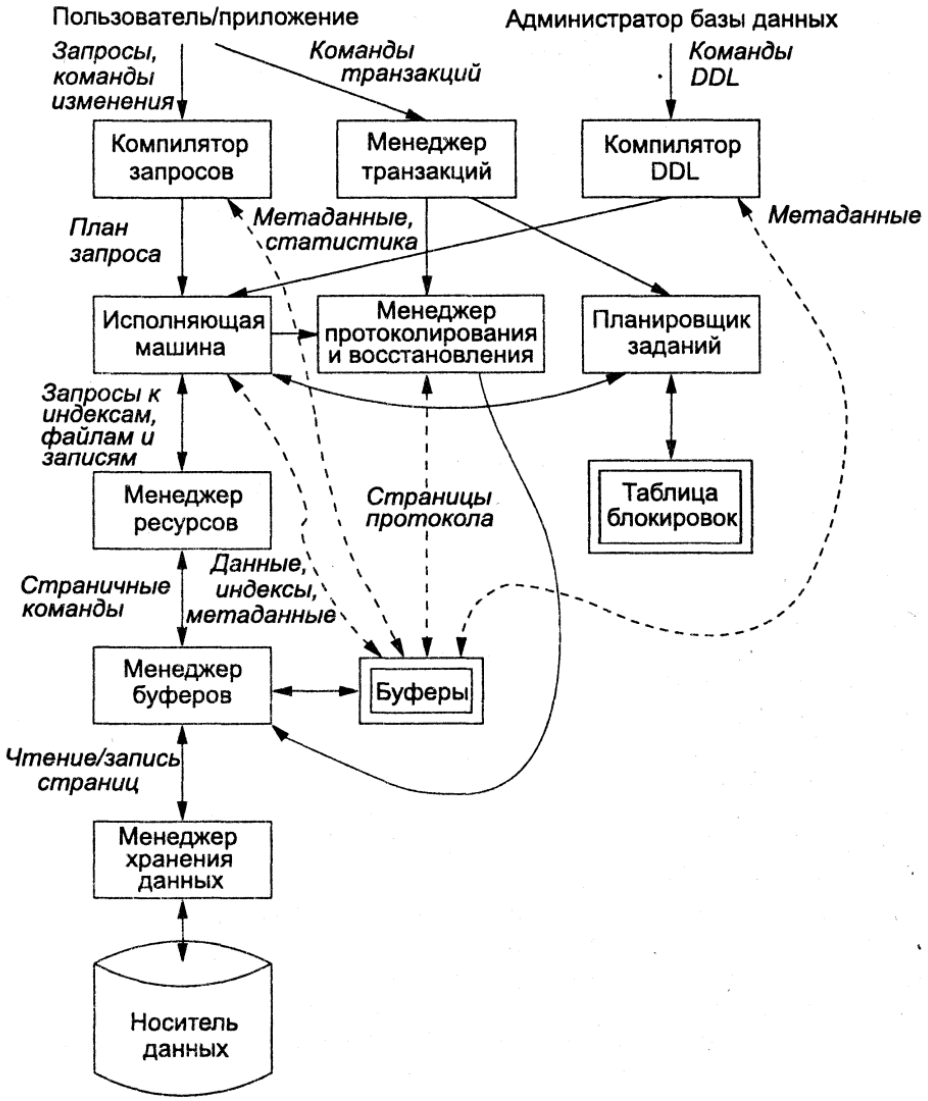
0.2.4. Функции системы управления базой данных (1)

- Хранение, извлечение и обновление данных
- Поддержание словаря данных (системного каталога)
  - имена и типы элементов данных
  - имена таблиц
  - ограничения целостности, накладываемые на данные
  - имена авторизованных пользователей СУБД и их права доступа к объектам базы данных
  - статистика использования объектов базы данных
- Поддержка транзакций
  - Транзакция — одно из важнейших понятий теории баз данных.
  - Она означает набор операций над базой данных, рассматриваемых как единая и неделимая единица работы, выполняемая полностью или не выполняемая вовсе, если произошел какой-то сбой в процессе выполнения транзакции.
  - Таким образом, транзакции являются средством обеспечения согласованности данных.

0.2.4. Функции системы управления базой данных (2)

- Поддержка параллельности (concurrency) в использовании базы данных
- Восстановление данных после сбоев
- Авторизация пользователей
- Обеспечение доступа к базе данных с компьютеров локальной или глобальной сети
- Защита и поддержка целостности данных
- Реализация принципа независимости от данных
  - Приложения не должны зависеть от фактической структуры базы данных
- Обслуживание базы данных (с помощью различных утилит)
- Оптимизация и выполнение запросов к базе данных

0.2.5. Компоненты СУБД



0.3. Реляционная модель

0.3.1. История реляционной модели

Реляционная модель была впервые предложена Эдгаром Ф. Коддом (E. F. Codd) в статье «A relational model of data for large shared data banks» в 1970 г.

Целью было:

1. Добиться высокой степени независимости приложений от данных (т. е. от изменений внутреннего представления данных: организации файлов, упорядочивания записей и изменения путей доступа).
2. Добиться согласованности данных и решить проблему избыточности данных (было введено понятие нормализованных отношений, о котором речь пойдет в следующих лекциях).

Исследовательские проекты:

- System R в компании IBM, вторая половина 1970-х гг.
  - Разработка языка SQL и первых коммерческих реляционных СУБД
- Проект INGRES в Калифорнийском университете в Беркли (University of California at Berkeley).
  INGRES – INteractive Graphics REtrieval System
  Майкл Стоунбрейкер (Michael Stonebraker)

0.3.2. Основные положения реляционной модели

В реляционной системе выполняются как минимум три условия:

- **Структурный аспект.** Данные в базе *воспринимаются* пользователем, как таблицы (и никак иначе).
- **Аспект целостности.** Эти таблицы отвечают определенным условиям целостности.
- **Аспект обработки.** В распоряжении пользователя имеются операторы манипулирования таблицами, которые генерируют новые таблицы на основании уже имеющихся и среди которых есть, по крайней мере, операторы сокращения (restrict), проекции (project) и объединения (join).

- Операция *сокращения* извлекает указанные строки из таблицы (в названии этой операции подразумевается, что число строк ее результата меньше или равно числу строк исходной таблицы). Операцию сокращения иногда называют *выборкой.*
- Операция *проекции* предназначена для извлечения определенных столбцов из таблицы.
- Операция *соединения* предназначена для получения комбинации двух таблиц на основе общих значений в общих столбцах.
- Эти определения являются нестрогими!

0.3.3. Модельная база данных

Таблица «Студенты» (students)

Номер зачетной книжки (record_book)	Ф. И. О. (name)	Серия паспорта (psp_ser)	Номер паспорта (psp_num)
55500	Иванов Иван Петрович	0402	645327
55800	Климов Андрей Иванович	0402	673211
55865	Новиков Николай Юрьевич	0202	554390

Таблица «Успеваемость» (progress)

Номер зачетной книжки (record_book)	Предмет (discipline)	Учебный год (acad_year)	Семестр (term)	Оценка (mark)
55500	Физика	2017/2018	1	5
55500	Математика	2017/2018	1	4
55800	Физика	2017/2018	1	4
55800	Физика	2017/2018	2	5

0.3.4. Основные сведения

- **Реляционная модель данных** – данные представлены посредством строк в таблицах.
- В теории баз данных эти таблицы называют **отношениями (relations)** – поэтому и базы данных называются **реляционными**. Отношение – это математический термин. При определении свойств таких отношений используется теория множеств. В терминах данной теории строки таблицы будут называться **кортежами (tuples)**, а колонки – **атрибутами**. Отношение имеет заголовок, который состоит из атрибутов, и тело, состоящее из кортежей. Количество атрибутов называется **степенью отношения**, а количество кортежей – **кардинальным числом**.
- Реляционные системы используют *декларативный* подход к получению данных для обработки. Это означает, что требуется только указать, какие данные нужны, но не нужно предписывать способ их получения.
- Реляционная модель требует, чтобы данные воспринимались в виде таблиц. Однако это требование относится только к логической структуре базы данных, т. е. только к внешнему и концептуальному уровням архитектуры ANSI/SPARC. На физическом уровне могут использоваться какие угодно структуры хранения данных и механизмы доступа.

0.3.4.1. Кортежи

- Если дана коллекция типов T_i (i = 1, 2, . . ., n), которые не обязательно все должны быть разными, то значением кортежа (или кратко *кортежем),* определенным с помощью этих типов (назовем его t), является множество упорядоченных троек в форме $<A_i,T_i,v_i>$, где A_i — имя атрибута, T_i — имя типа и v_i — значение типа T_i. Кроме того, кортеж t должен соответствовать требованиям:
- Значение n – это **степень,** или **арность** t.
- Упорядоченная тройка $<A_i, T_i, v_i>$ является **компонентом** t.
- Упорядоченная пара $<A_i, T_i>$ представляет собой **атрибут** t и однозначно определяется именем атрибута A_i.
- Значение v_i — это **значение атрибута**, соответствующее атрибуту A_i кортежа t. Тип T_i — это соответствующий **тип атрибута**. Полное множество атрибутов составляет **заголовок кортежа** t.
- **Тип кортежа** t определен заголовком t, а сам заголовок и этот тип кортежа имеют такие же атрибуты (и поэтому такие же имена и типы атрибутов) и такую же степень, как t.

MAJOR_P# : P#	MINOR_P# : P#	QTY : QTY
P2	P4	7

0.3.4.2. Отношения

- Отношение это математический термин. Необходимо различать переменную отношения (relation variable) и значение отношения.
- Каждое отношение, точнее, каждое *значение* отношения, состоит из двух частей: набора пар «*имя_столбца : имя_типа*» (**заголовка**) и множества кортежей, согласованных с этим заголовком (**тела**). Кардинальность **r** отношения определяется как кардинальность этого множества (*кардинальностью множества* называется количество элементов множества).
- Отношение r имеет такие же атрибуты (следовательно, такие же имена и типы атрибутов) и такую же степень, как заголовок.
- Тело отношения r представляет собой множество кортежей, имеющих один и тот же заголовок.

0.3.4.3. Свойства отношений

Каждое отношение обладает следующими свойствами:

1. Каждый кортеж содержит точно одно значение (соответствующего типа) для каждого атрибута. Атрибуты могут иметь **значения в виде отношений**.

2. Атрибуты не характеризуются каким-либо упорядочением (например, слева направо).

3. Кортежи не характеризуются каким-либо упорядочением (например, сверху вниз).

4. В отношении отсутствуют дубликаты кортежей.

0.3.5. Основные понятия реляционной модели (1)

Ограничения

При работе с базами данных часто приходится следовать различным **ограничениям**, которые могут быть обусловлены спецификой конкретной предметной области:

- номер зачетной книжки состоит из пяти цифр и не может быть отрицательным
- серия документа, удостоверяющего личность, является четырехзначным числом, а номер документа, удостоверяющего личность — шестизначным числом
- номер семестра может принимать только два значения — 1 (осенний семестр) и 2 (весенний семестр)
- оценка может принимать только три значения — 3 (удовлетворительно), 4 (хорошо) и 5 (отлично)

0.3.5. Основные понятия реляционной модели (2)

Ключи

- Ключи служат для идентификации строк в таблицах и для связи таблиц между собой.
- Потенциальный ключ — это комбинация атрибутов таблицы, позволяющая уникальным образом идентифицировать строки в ней.
- Ключ может состоять и только лишь из одного атрибута таблицы.
  Например, в таблице «Студенты» таким идентификатором может быть атрибут «Номер зачетной книжки».
  В качестве потенциального ключа данной таблицы могут также служить два ее атрибута, взятые вместе: «Серия паспорта» и «Номер паспорта». Ни один из них в отдельности не может использоваться в качестве уникального идентификатора. В таком случае ключ будет составным.
- Потенциальный ключ должен быть не избыточным, т. е. никакое подмножество атрибутов, входящих в него, не должно обладать свойством уникальности.

0.3.5. Основные понятия реляционной модели (3)

Ключи (продолжение)

- При наличии в таблице более одного потенциального ключа один из них выбирается в качестве так называемого первичного ключа, а остальные будут являться альтернативными ключами.
- Предположим, что в таблице «Студенты» нет строки с номером зачетной книжки 55900, тогда включать строку с таким номером зачетной книжки в таблицу «Успеваемость» не имеет смысла. Таким образом, значения столбца «Номер зачетной книжки» в таблице «Успеваемость» должны быть согласованы со значениями такого же столбца в таблице «Студенты».
- Атрибут «Номер зачетной книжки» в таблице «Успеваемость» является примером того, что называется внешним ключом. Таблица, содержащая внешний ключ, называется ссылающейся таблицей (referencing table). Таблица, содержащая соответствующий потенциальный ключ, называется ссылочной (целевой) таблицей (referenced table). В таких случаях говорят, что внешний ключ ссылается на потенциальный ключ в ссылочной таблице.
- Внешний ключ может быть составным, т. е. может включать более одного атрибута. Внешний ключ не обязан быть уникальным.

0.3.5. Основные понятия реляционной модели (4)

Ссылочная целостность

- Проблема обеспечения того, чтобы база данных не содержала неверных значений внешних ключей, известна как проблема ссылочной целостности. Ограничение, согласно которому значения внешних ключей должны соответствовать значениям потенциальных ключей, называется ограничением ссылочной целостности (ссылочным ограничением).
- Обеспечением выполнения ограничений ссылочной целостности занимается СУБД, а от разработчика требуется лишь указать атрибуты, служащие в качестве внешних ключей.

- При проектировании баз данных часто предусматривается, что при удалении строки из ссылочной таблицы соответствующие строки из ссылающейся таблицы должны быть также удалены, а при изменении значения столбца, на который ссылается внешний ключ, должны быть изменены значения внешнего ключа в ссылающейся таблице. Этот подход называется каскадным удалением (обновлением).

0.3.6. Операторы манипулирования таблицами

Сокращение Таблица «Студенты» (students)

Номер зачетной книжки	Ф. И. О.	Серия паспорта	Номер паспорта
55500	Иванов Иван Петрович	0402	645327

Проекция Таблица «Успеваемость» (progress)

Предмет	Учебный год	Семестр	Оценка
Физика	2017/2018	1	5
Математика	2017/2018	1	4
Физика	2017/2018	1	4
Физика	2017/2018	2	5

Соединение Таблицы «Студенты» (students) и «Успеваемость» (progress)

Номер зачетной книжки	Ф. И. О.	Серия паспорта	Номер паспорта	Предмет	Учебный год	Семестр	Оценка
55500	Иванов Иван Петрович	0402	645327	Физика	2017/2018	1	5
55500	Иванов Иван Петрович	0402	645327	Математика	2017/2018	1	4
55800	Климов Андрей Иванович	0402	673211	Физика	2017/2018	1	4
55800	Климов Андрей Иванович	0402	673211	Физика	2017/2018	2	5

0.3.7. Ряд замечаний

- Определение реляционной системы требует, чтобы база данных только *воспринималась пользователем* как набор таблиц. Таблицы в реляционной системе являются **логическими,** а не физическими структурами. На самом деле, на физическом уровне система может использовать любую из существующих структур памяти (последовательный файл, индексирование, хэширование, цепочку указателей и т. п.), лишь бы существовала возможность отображать эти структуры в виде таблиц на логическом уровне.
- Таким образом, таблицы представляют собой *абстракцию* способа физического хранения данных, в которой все нюансы реализации на уровне физической памяти *скрыты от пользователя.*
- На самом деле реляционная теория вообще не может определить внутренний уровень. Она определяет лишь то, как база данных *представлена пользователю.*
- У реляционных баз данных есть одно свойство, определяемое так называемым **информационным принципом:** *все информационное наполнение базы данных представлено одним и только одним способом, а именно — явным заданием значений, помещенных в позиции столбцов в строках таблицы.* Этот метод представления — *единственно* возможный для реляционных баз данных (естественно, на логическом уровне). В частности, **нет никаких** *указателей,* связывающих одну таблицу с другой.

0.3.8. Замкнутость реляционной системы

- Свойство **замкнутости** реляционных систем означает, что результат выполнения операции имеет тот же тип, что и объекты, над которыми проводилась операция (все они являются отношениями). Но это значит, что к *результатам операций можно снова применить какую-либо операцию.*
- Например, можно сформировать проекцию соединения, соединение двух сокращений, сокращение проекции и т.д. Другими словами, можно использовать **вложенные реляционные выражения**, т.е. выражения, в которых операнды представлены выражениями, а не простыми именами таблиц.
- Значения переменных отношения изменяются с помощью операций **реляционного присваивания,** причем привычные нам операции обновления **INSERT, UPDATE** и **DELETE** можно считать сокращенной формой записи операций реляционного присваивания определенных типов.

0.3.9. Базовые отношения и представления

- Исходные (заданные) переменные отношения называются **базовыми переменными отношения,** а присвоенные им значения называются **базовыми** отношениями. Отношение, которое получено или может быть получено из базового отношения в результате выполнения каких-либо реляционных выражений, называется **производным** отношением.
- Реляционные системы обычно поддерживают еще один вид именованных переменных отношения, называемых **представлениями**. В любой конкретный момент их значение является *производным* отношением.
- Выражение, фактически определяющее представление**,** не вычисляется, а просто *запоминается* системой.
- В реляционной модели представления являются одним из способов обеспечения *логической независимости от данных*.
- Базовые переменные отношения «реально существуют» в том смысле, что они воплощают в себе данные, которые действительно хранятся в базе данных.
- Представления, наоборот, «реально не существуют», а просто предоставляют различные способы просмотра «реальных» данных.

0.3.10. Различная терминология

Формальные термины	Альтернатива 1	Альтернатива 2
Relation (отношение)	Table (таблица)	File (файл)
Tuple (кортеж)	Row (строка)	Record (запись)
Attribute (атрибут)	Column (столбец)	Field (поле)

0.4. Реляционная алгебра и реляционное исчисление

0.4.1. Введение (1)

- Важной составной частью реляционной модели является манипуляционный механизм, т. е. языки запросов, позволяющие извлекать и обновлять данные.
- Это реляционная алгебра и реляционное исчисление предложенные Э. Коддом в 1971 г.
- Неформально можно определить реляционную алгебру как *высокоуровневый процедурный* язык: она может использоваться для того, чтобы указать СУБД, как сформировать новое отношение из одного или более отношений в базе данных.
- Аналогично реляционное исчисление можно описать как *непроцедурный* язык: оно может использоваться для того, чтобы сформулировать определение нового отношение в терминах одного или более отношений, существующих в базе данных.
- Реляционная алгебра и реляционное исчисление формально эквивалентны друг другу, т. е. для каждого выражения реляционной алгебры существует эквивалентное выражение реляционного исчисления (и наоборот).

0.4.1. Введение (2)

- Реляционная алгебра и реляционное исчисление являются формальными языками, они не являются дружественными к пользователю.
- Они имеют значение поскольку:
  - Иллюстрируют базовые операции, требующиеся от любого языка манипулирования данными (DML)
  - Выступают в роли стандарта для сравнения других реляционных языков

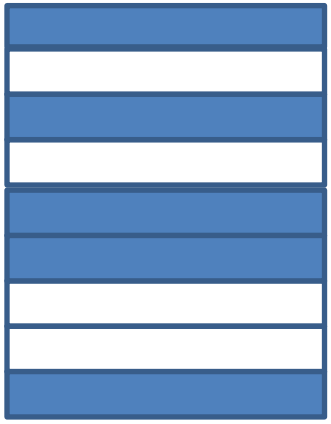
- Реляционное исчисление используется для оценки мощи реляционных языков. Язык, который позволяет получить любое отношение, которое можно получить с помощью реляционного исчисления, называется **реляционно-полным**.
- Реальные языки зачастую не только являются реляционно-полными, но и имеют большую выразительность, чем реляционная алгебра и реляционное исчисление, за счет наличия дополнительных операций, таких как вычисления, суммирования, упорядочивание.

0.4.2. Реляционная алгебра

- Реляционная алгебра — это коллекция *операций*, которые принимают отношения в качестве операндов и возвращают отношение в качестве результата.
- Результат выполнения одной операции может использоваться в качестве входных данных (операнда) другой операции. Поэтому возможно использование вложенных операций. Это свойство называется **замкнутостью (closure)**.
- Алгебра, определенная Э. Коддом, включала две группы по 4 операции:
- Традиционные операции с множествами — объединение (Union), пересечение (Intersection), разность (Set difference) и декартово произведение (Cartesian product)
- Специальные реляционные операции — сокращение (Restriction), или выборка (Selection), проекция (Projection), соединение (Join) и деление (Division)
- Эти операции можно разделить на базовые (выборка, проекция, декартово произведение, объединение и разность) и дополнительные (соединение, пересечение и деление).
- Операции выборки и проекции являются унарными (работают с одним отношением), остальные операции — бинарные (работают с двумя отношениями).
- Отметим, что все эти операции предназначены *только для чтения* (т. е. они «читают», но не обновляют свои операнды).

0.4.2.1. Выборка (Selection), или сокращение (Restriction)

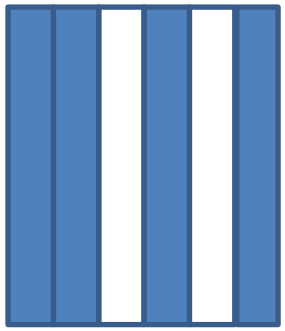
- Эта операция выполняется на одном отношении R и определяет отношение, которое содержит только те кортежи исходного отношения R, которые удовлетворяют заданному условию (предикату).
- К. Дейт использует такое обозначение: **R** WHERE p
- Некоторые другие авторы — такое: $\boldsymbol{\sigma}_{predicate}(\mathbf{R})$
- Выражение p — это **предикат**, или **условие сокращения**



Примеры.
Students WHERE record_book = 55500
Students WHERE record_book = 55500 OR record_book = 55800
Progress WHERE acad_year = '2017/2018' AND term = 1

0.4.2.2. Проекция (Projection)

- Эта операция выполняется на одном отношении R, которое имеет атрибуты X, Y, . . ., Z (и, возможно, другие атрибуты) и определяет отношение, которое содержит вертикальное подмножество исходного отношения R, извлекая значения указанных атрибутов и отбрасывая дубликаты полученных кортежей. В таком случае **проекция** отношения R по атрибутам X, Y, . . . , Z определяется с помощью следующего выражения (К. Дейт): **R** { X, Y, . . . , Z }.
- Его заголовок формируется из заголовка отношения R путем удаления всех атрибутов, не указанных в множестве { X, Y, . . . , Z }.
- Некоторые авторы используют обозначение: $\mathbf{П}_{a_1 \dots , a_n}(\mathbf{R})$

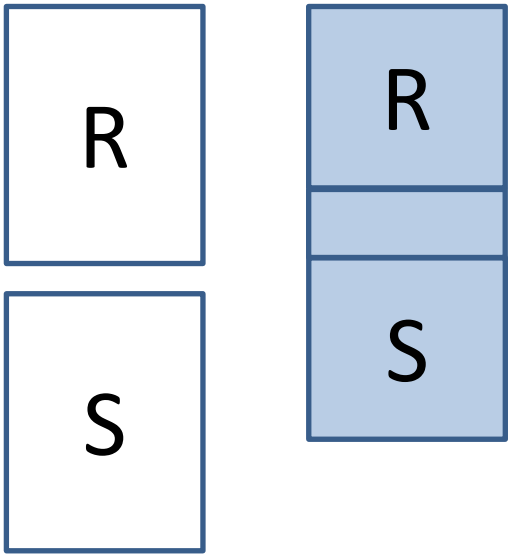


Пример.
Progress { acad_year, term }

Учебный год (acad_year)	Семестр (term)
2017/2018	1
2017/2018	2

0.4.2.3. Объединение (Union)

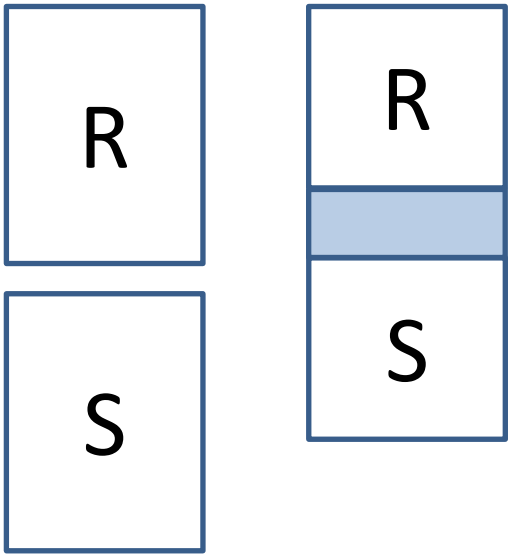
- Объединением отношений R и S является отношение того же типа с телом, которое состоит из всех кортежей, присутствующих в R или в S, или в обоих отношениях. При этом кортежи-дубликаты устраняются.
- Отношения R и S должны принадлежать к **одному типу**. Они должны быть совместимыми по объединению (union compatible), т. е. иметь одинаковые заголовки.
- К. Дейт использует такое обозначение: **R** UNION **S**
- Некоторые другие авторы — такое: **R** ∪ **S**



Пример.
Students1 UNION Students2
(предположим, что оба эти отношения имеют тот же тип, что и отношение Students)

0.4.2.4. Пересечение (Intersection)

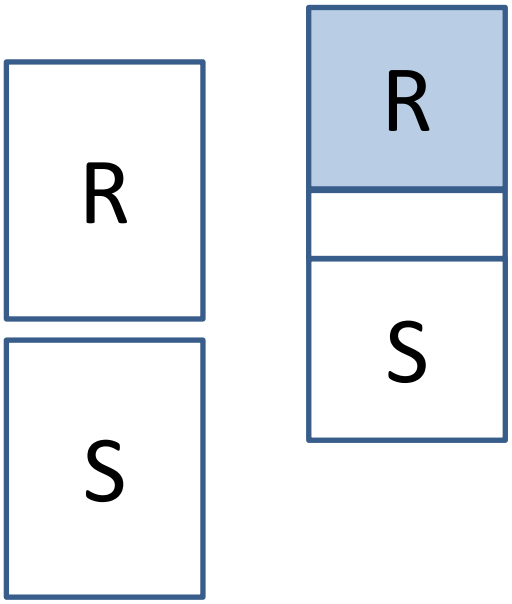
- Пересечением отношений R и S является отношение того же типа с телом, которое состоит из всех кортежей, присутствующих одновременно в R и в S.
- Отношения R и S должны принадлежать к **одному типу**. Они должны быть совместимыми по объединению (union compatible), т. е. иметь одинаковые заголовки.
- К. Дейт использует такое обозначение: **R** INTERSECT **S**
- Некоторые другие авторы — такое: **R** ∩ **S**



Пример.
Students1 INTERSECT Students2
(предположим, что оба эти отношения имеют тот же тип, что и отношение Students)

0.4.2.5. Разность (Set difference)

- Разностью отношений R и S (в указанном порядке) является отношение того же типа с телом, которое состоит из всех кортежей, присутствующих в R, но не присутствующих в S.
- Отношения R и S должны принадлежать к **одному типу**. Они должны быть совместимыми по объединению (union compatible), т. е. иметь одинаковые заголовки.
- Важно: оператор MINUS характеризуется направленностью (некоммутативностью), так же, как вычитание в обычной арифметике.



- К. Дейт использует такое обозначение:
  R MINUS **S**
- Некоторые другие авторы — такое:
  R – **S**

Пример.
Students1 MINUS Students2
(предположим, что оба эти отношения имеют тот же тип, что и отношение Students)

0.4.2.6. Декартово произведение (Cartesian product) (1)

- Декартово произведение отношений R и S, не имеющих общих атрибутов – это отношение, заголовок которого представляет собой (теоретико-множественное) объединение заголовков отношений R и S, а тело состоит из всех кортежей t, таких, что t является (теоретико-множественным) объединением кортежа, принадлежащего к отношению R, и кортежа, принадлежащего к отношению S. Следует отметить, что кардинальность результата равна произведению кардинальностей входных отношений, R и S, а степень результата — сумме степеней входных отношений.
- Если необходимо сформировать декартово произведение двух отношений, имеющих общие имена атрибутов, то следует вначале воспользоваться оператором RENAME, чтобы переименовать атрибуты должным образом.
- К. Дейт использует такое обозначение: **R** TIMES **S**
- Некоторые другие авторы — такое: **R** × **S**

0.4.2.6. Декартово произведение (Cartesian product) (2)

Отношение R

Last_name	First_name
Иванов	Петр
Петров	Иван

Отношение S

Discipline	Exam_date
Web-программирование	12.01.2018
Базы данных	15.01.2018
Язык C	21.01.2018

Отношение R × S

Last_name	First_name	Discipline	Exam_date
Иванов	Петр	Web-программирование	12.01.2018
Иванов	Петр	Базы данных	15.01.2018
Иванов	Петр	Язык C	21.01.2018
Петров	Иван	Web-программирование	12.01.2018
Петров	Иван	Базы данных	15.01.2018
Петров	Иван	Язык C	21.01.2018

0.4.2.7. Соединение (Join) (1)

- Соединение равносильно выполнению операции выборки (selection) над результатом декартова произведения: R JOIN S = (R TIMES S) WHERE p
- Имеются следующие виды соединений:
  - θ-соединение
  - Эквисоединение (частный случай θ-соединения)
  - Естественное соединение
  - Внешнее соединение
  - Полусоединение
- **θ-соединение** определяет отношение, которое содержит кортежи из декартова произведения отношений R и S, удовлетворяющие предикату p. Предикат p имеет форму $R.a_i\ \theta\ S.b_i$, где θ может являться одним их операторов сравнения ($<, \leq, >, \geq, =, \neq$), а $R.a_i$ и $S.b_i$ – атрибуты отношений R и S соответственно.
- **Эквисоединение** – это частный случай θ-соединения. Он имеет место, когда в качестве оператора сравнения используется только проверка на равенство (=).
- **Естественное соединение** – это частный случай эквисоединения. Он имеет место, когда отношения R и S соединяются по всем общим атрибутам. Один экземпляр каждого из общих атрибутов исключается из результирующего отношения.

0.4.2.7. Соединение (Join) (2)

Естественное соединение
Таблицы «Студенты» (students) и «Успеваемость» (progress)
Общий атрибут – record_book

Students JOIN Progress эквивалентно следующему выражению:

(Students TIMES (Progress RENAME record_book AS p_record_book)
WHERE record_book = p_record_book)

Степень результата естественного соединения равна сумме степеней отношений R и S минус число общих атрибутов этих отношений.

Номер зачетной книжки	Ф. И. О.	Серия паспорта	Номер паспорта	Предмет	Учебный год	Семестр	Оценка
55500	Иванов Иван Петрович	0402	645327	Физика	2017/2018	1	5
55500	Иванов Иван Петрович	0402	645327	Математика	2017/2018	1	4
55800	Климов Андрей Иванович	0402	673211	Физика	2017/2018	1	4
55800	Климов Андрей Иванович	0402	673211	Физика	2017/2018	2	5

0.4.2.7. Соединение (Join) (3)

Внешнее соединение
Таблицы «Студенты» (students) и «Успеваемость» (progress)
Общий атрибут – record_book

Левое внешнее соединение отношений R и S – это соединение, в котором кортежи из R, которые не имеют соответствующих значений в общих атрибутах отношения S, также включаются в результирующее отношение. Отсутствующие значения атрибутов отношения S заменяются на NULL.

Номер зачетной книжки	Ф. И. О.	Серия паспорта	Номер паспорта	Предмет	Учебный год	Семестр	Оценка
55500	Иванов Иван Петрович	0402	645327	Физика	2017/2018	1	5
55500	Иванов Иван Петрович	0402	645327	Математика	2017/2018	1	4
55800	Климов Андрей Иванович	0402	673211	Физика	2017/2018	1	4
55800	Климов Андрей Иванович	0402	673211	Физика	2017/2018	2	5
55865	Новиков Николай Юрьевич	0202	554390				

Правое внешнее соединение определяется аналогично. **Полное внешнее соединение** сохраняет в результирующем отношении кортежи из обоих исходных отношений, заменяя отсутствующие значения на NULL.

0.4.2.7. Соединение (Join) (4)

- **Полусоединение** определяет отношение, которое содержит кортежи из отношения R, которые участвуют в соединении R и S, удовлетворяющие предикату p.
- R SEMIJOIN S
- Полусоединение выполняет соединение двух отношений, а затем выполняет проекцию по атрибутам первого операнда.
- Таблицы «Студенты» (students) и «Успеваемость» (progress)
  Общий атрибут – record_book
  (Students TIMES (Progress RENAME record_book AS p_record_book)
  WHERE record_book = p_record_book AND discipline = 'Математика')
  {record_book, name, psp_ser, psp_name}

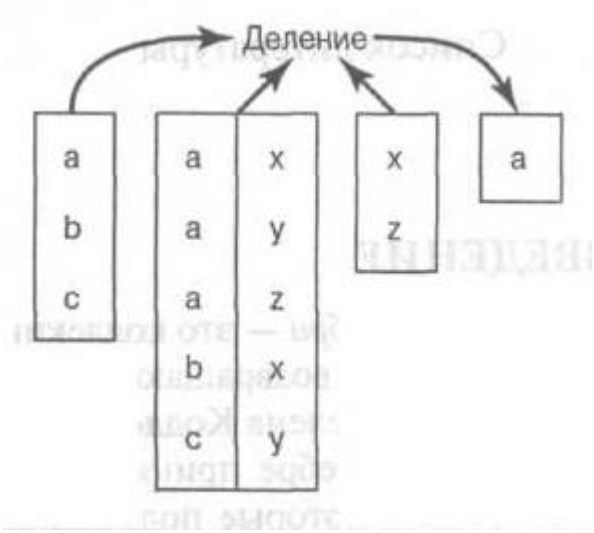
Номер зачетной книжки	Ф. И. О.	Серия паспорта	Номер паспорта
55500	Иванов Иван Петрович	0402	645327

0.4.2.8. Деление (Division)

- Предположим, что отношения R и S, соответственно, имеют следующие атрибуты: $X_1, X_2, \ldots, X_m$ и $Y_1, Y_2, \ldots, Y_n$. Здесь ни один из атрибутов X_i (i = 1, 2, . . ., m) не имеет одинакового имени с любым из атрибутов Y_j (j = 1, 2, . . ., n). Пусть отношение T имеет следующие атрибуты: $X_1, X_2, \ldots, X_m$ и $Y_1, Y_2, \ldots, Y_n$. Это означает, что T имеет заголовок, представляющий собой (теоретико-множественное) объединение заголовков отношений R и S. Будем рассматривать множества $\{X_1, X_2, \ldots, X_m\}$ и $\{Y_1, Y_2, \ldots, Y_n\}$, соответственно, как составные атрибуты X и Y. В таком случае операция деления R на S по T (где R — *делимое, S — делитель,* а T — *посредник*) может быть представлена с помощью следующего выражения:

R DIVIDEBY S PER T

Результат представляет собой отношение с заголовком { X } и телом, состоящим из таких кортежей { X x }, присутствующих в R, что кортеж { X x, Y y } присутствует в T для всех кортежей { Y y }, присутствующих в S. Иными словами, данный результат состоит из тех значений X, присутствующих в R, для которых соответствующие значения Y в T включают все значения Y из S.



0.4.3. Реляционное исчисление (1)

- Реляционное исчисление получило название от той части символьной логики, которая называется *исчислением предикатов.* В контексте баз данных оно существует в двух формах: в форме предложенного Э. Коддом *реляционного исчисления кортежей* и в форме предложенного Лакруа и Пиро *реляционного исчисления доменов.*
- В логике первого порядка (или теории исчисления предикатов) под *предикатом* подразумевается истинностная функция с параметрами. После подстановки значений вместо параметров функция становится выражением, называемым *суждением,* которое может быть истинным или ложным.
- Например, предложения «Иван Петров является сотрудником данной организации» и «Иван Петров имеет более высокую зарплату, чем Петр Иванов» являются суждениями, поскольку можно определить их истинность или ложность. В первом случае функция «является сотрудником данной организации» имеет один параметр («Иван Петров»), а во втором случае функция «имеет более высокую зарплату, чем» имеет два параметра («Иван Петров» и «Петр Иванов»).

0.4.3. Реляционное исчисление (2)

- Если предикат содержит переменную, например в виде «x является сотрудником этой организации», то у этой переменной должна быть соответствующая *область определения.* При подстановке вместо переменной x одних значений из ее области определения данное суждение может оказаться истинным, а при подстановке других — ложным.
- Если P — предикат, то множество всех значений переменной x, при которых P становится истинным, можно символически записать следующим образом:

$$\{x \mid P(x)\}$$

- Предикаты могут соединяться с помощью логических операций ∧ (AND), ∨ (OR) и ~ (NOT) с образованием составных предикатов.

0.4.3. Реляционное исчисление (3)

Задача: Выбрать номера поставщиков и названия городов, в которых находятся поставщики детали с номером P2

- Реляционная алгебра:

■ Выполнить соединение отношений поставщиков S и поставок SP по атрибуту S#.

■ С помощью операции сокращения выделить из результатов этого соединения кортежи, которые относятся к детали с номером P2.

■ Сформировать проекцию результатов этой операции сокращения по атрибутам s# и CITY.

- Реляционное исчисление:

■ Получить атрибуты s# и CITY для таких поставщиков, для которых в отношении SP существует запись о поставке с тем же значением атрибута s# и со значением атрибута P#, равным P2.

- В реляционном исчислении просто указывается, в чем *заключается* проблема, тогда как в реляционной алгебре задается *процедура решения* этой проблемы. На самом деле *реляционная алгебра и реляционное исчисление логически эквивалентны.*

0.5. Язык SQL

0.5.1. Введение (1)

- Язык SQL был разработан в лаборатории IBM Research в начале 1970-х годов. Первой серьезной реализацией этого языка был продукт-прототип System R компании IBM. Первой коммерческой реализацией СУБД, основанной на SQL, была СУБД Oracle (конец 1970-х гг.).
- Разработчики языка: **Donald D. Chamberlin и Raymond F. Boyce** (1947–1974).
- Стандарты языка SQL: SQL-86, SQL-89, SQL-92 (SQL2), SQL:1999 (SQL3), SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL:2016.
- К. Дейт считает, что в наше время ни одна реальная СУБД не поддерживает стандарт полностью, а поддерживает «надмножество подмножества» стандарта: большинство СУБД не поддерживают некоторые средства, обусловленные стандартом, и в то же время предлагают другие средства, которые не определены этим стандартом.
- SQL — язык очень большого объема. Его спецификация содержит свыше 2000 страниц, не считая больше 300 страниц исправлений.
- В стандарте SQL есть противоречия.

0.5.1. Введение (2)

- Некоторый язык принято называть **реляционно полным,** если он по своим возможностям, по крайней мере, не уступает реляционному исчислению.
- Реляционный язык может быть основан как на реляционной алгебре, так и на реляционном исчислении.
- Когда язык SQL только разрабатывался, предполагалось что он будет отличаться как от реляционной алгебры, так и от реляционного исчисления.
- На сегодняшний день ситуация складывается таким образом, что язык SQL в чем-то похож на реляционную алгебру, в чем-то на реляционное исчисление, а в чем-то отличается от них обоих.

0.5.1. Введение (3)

- Фундаментальным объектом данных в языке SQL является не отношение, а скорее таблица, и таблицы SQL содержат (вообще говоря) *не множества*, а *мультимножества* строк (в мультимножествах допускаются повторения элементов). Таким образом, в языке SQL нарушается *информационный принцип.*
- Одно из следствий этого факта состоит в том, что основные операторы SQL являются не реляционными операторами в полном смысле этого слова, а аналогами реляционных операторов, предназначенных для работы с мультимножествами.
- Другим следствием является то, что теоремы и их побочные результаты, которые являются справедливыми в реляционной модели, не обязательно выполняются в языке SQL.

0.5.1. Введение (4)

- В языке SQL вместо терминов *отношение* и *переменная отношения* используется термин **таблица,** а вместо терминов *кортеж* и *атрибут* — **строка** и **столбец.**
- В язык SQL не включен прямой аналог операции **реляционного присваивания.** Но эту операцию можно эмулировать, сначала удалив все строки из целевой таблицы, а затем выполнив для нее операции INSERT ... SELECT ...
- Стандарт SQL включает спецификации стандартного каталога, именуемого в нем **информационной** схемой. **Каталог** в языке SQL состоит из дескрипторов (метаданных) для отдельной базы данных, а схема состоит из дескрипторов той части базы данных, которая принадлежит отдельному пользователю. Каждый каталог должен содержать одну и только одну схему с именем INFORMATION_SCHEMA.

0.5.1. Введение (5)

- Язык SQL первоначально разрабатывался конкретно как **подъязык** данных. Однако после включения в стандарт в конце 1996 года такого средства, как **постоянные хранимые модули SQL** (SQL Persistent Stored Modules — SQL/PSM), стандарт SQL стал полностью поддерживать все вычислительные конструкции. Сейчас в нем предусмотрены процедурные операторы, например, CALL, RETURN, SET, CASE, IF, LOOP, LEAVE, WHILE, REPEAT.
- Язык SQL — это непроцедурный язык, который является стандартным средством работы с данными во всех реляционных СУБД.
- Операторы (команды), написанные на этом языке, лишь указывают СУБД, какой результат должен быть получен, но не описывают процедуру получения этого результата. СУБД сама определяет способ выполнения команды пользователя.

0.5.1. Введение (6)

- В языке SQL традиционно выделяются группа операторов определения данных (Data Definition Language — DDL), группа операторов манипулирования данными (Data Manipulation Language — DML) и группа операторов, управляющих привилегиями доступа к объектам базы данных (Data Control Language — DCL).
- К операторам DDL относятся команды для создания, изменения и удаления таблиц, представлений и других объектов базы данных.
- К операторам DML относятся команды для выборки строк из таблиц, вставки строк в таблицы, обновления и удаления строк.
- Тексты запросов и команд SQL не чувствительны к регистру символов (case insensitive). Например, ключевое слово FROM можно написать как from или From.
- Имена атрибутов, псевдонимов, таблиц и т. д. также не чувствительны к регистру символов. Регистр учитывается, только если идентификатор заключен в кавычки. Например, варианты имени столбца name и ‘Name’ будут восприниматься как различные.

0.5.2. Способы использования языка SQL (1)

- Операторы языка SQL могут выполняться как **непосредственно** (т. е. интерактивно, с подключенного терминала), так и в виде части прикладной программы (т. е. операторы SQL могут быть **внедренными,** а значит, могут смешиваться с операторами базового языка этой программы).
- Фундаментальный принцип, лежащий в основе технологии внедрения операторов SQL**,** называется **принципом двухрежимности.** Он заключается в том, что *любое выражение SQL, которое можно использовать интерактивно, можно применять и путем внедрения в прикладную программу.* Конечно, существует множество различий в деталях между интерактивными операторами SQL и их внедренными аналогами. В частности, операции выборки требуют существенной дополнительной обработки в вычислительной среде базового языка.

0.5.2. Способы использования языка SQL (2)

Программный SQL может иметь следующие разновидности:

- встроенный SQL – разновидность, наиболее широко применяемая в реляционных СУБД;
- динамический SQL – усовершенствованная форма встроенного SQL; с его помощью создаются утилиты общего назначения для работы с базами данных;
- SQL API – альтернативная разновидность программного SQL – интерфейс вызовов функций, применяемый в нескольких популярных СУБД.

0.5.3. Порядок обработка запросов в СУБД

- Выполняется синтаксический анализ SQL-запроса. СУБД разделяет запрос на отдельные слова, затем проверяет, правильно ли в запросе указана команда, используются ли допустимые предложения и т. д. На этом этапе обнаруживаются синтаксические ошибки.
- Осуществляется проверка семантической правильности SQL-запроса. Посредством обращения к системному каталогу выясняется, существуют ли в базе данных таблицы, указанные в инструкции, существуют ли указанные столбцы и являются ли их имена однозначными, имеет ли пользователь привилегии, необходимые для выполнения инструкции. На этом этапе обнаруживаются семантические ошибки.
- Выполняется планирование запроса – СУБД исследует возможные способы его выполнения. Выясняется, например, может ли быть использован индекс для ускорения поиска. После исследования возможных альтернатив СУБД выбирает одну из них.
- Запрос выполняется, и результаты возвращаются клиенту.

0.5.4. Встроенный SQL (пример) (1)

```
main()
{
    exec sql include sqlca;
    exec sql begin declare section;
        int officenum;                  /* Номер офиса (задает
                                           пользователь) */
        char cityname[ 16 ];            /* Город */
        char regionname[ 11 ];          /* Регион */
        float targetval;                /* План продаж */
        float salesval;                 /* Объем продаж */
    exec sql end declare section;

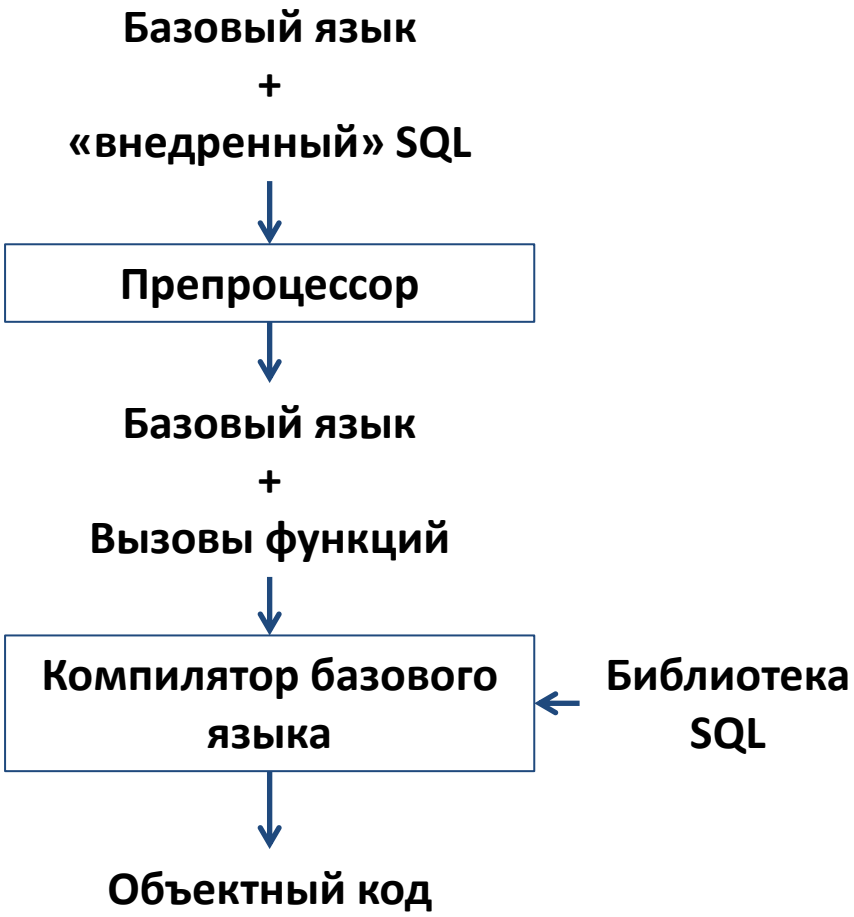

    /* Настройка обработки ошибок */
    exec sql whenever sqlerror goto query_error;
    exec sql whenever not found goto bad_number;

    /* Запрос номера офиса у пользователя */
    printf( " Введите номер офиса : " );
    scanf( "%d", &officenum );
```

0.5.4. Встроенный SQL (пример) (2)

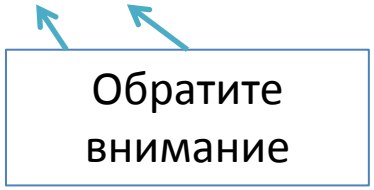
```
    /* Выполнение SQL-запроса */
    exec sql select city, region, target, sales
             from offices
             where office = :officenum
             into :cityname, :regionname,
                  :targetval, :salesval;
    /* Вывод результатов */
    printf( "Город %s\n", cityname );
    printf( "Регион: %s\n", regionname );
    printf( "План %f\n" , targetval );
    printf( "Продажи: %f\n", salesval );
    exit();
    query_error:
        printf( "Ошибка SQL: %s\n", sqlca.sqlstate ) ;|
        exit();
    bad_number:
       printf( "Неверный номер офиса.\n" );
       exit();
}
```

0.5.5. Обработка программного кода с инструкциями SQL



0.5.6. Динамический SQL

- **Выполнение оператора с входными параметрами (PostgreSQL)**
- Более эффективно выполнять произвольный оператор SQL можно, *подготовив его один раз*, а затем запуская подготовленный оператор столько, сколько нужно. Также можно подготовить обобщённую версию оператора, а затем выполнять специализированные его версии, подставляя в него параметры. Подготавливая оператор, поставьте знаки вопроса там, где позже хотите подставить параметры. Например:
- ```
  EXEC SQL BEGIN DECLARE SECTION;
      const char *stmt = "INSERT INTO test1 VALUES( ?, ? );";
  EXEC SQL END DECLARE SECTION;
  EXEC SQL PREPARE mystmt FROM :stmt;
      . . .
  EXEC SQL EXECUTE mystmt USING 42, 'foobar';
  ```



- Когда подготовленный оператор больше не нужен, его следует освободить:
- `EXEC SQL DEALLOCATE PREPARE` *имя*`;`
  ```

0.5.7. Сравнение статического и динамического SQL

- **Простота**. Статический SQL относительно прост; даже самый сложный его элемент – курсоры – можно легко освоить, вспомнив концепцию файлового ввода-вывода. Динамический SQL довольно сложен; в нем осуществляется динамическое формирование инструкций, используются структуры данных переменной длины, выполняется распределение памяти и решаются другие сложные задачи.
- **Производительность**. Во время компиляции программы, использующей статический SQL, создается план выполнения всех встроенных инструкций; инструкции динамического SQL компилируются непосредственно на этапе выполнения. В результате производительность статического SQL, как правило, намного выше, чем динамического. Производительность динамического SQL существенно зависит от дизайна приложения; минимизация накладных расходов на компиляцию позволяет достичь производительности статического SQL.
- **Гибкость**. Динамический SQL дает программе возможность решать на этапе выполнения, какие конкретно инструкции SQL она будет выполнять. Статический SQL требует, чтобы все инструкции SQL были написаны заранее, на этапе создания программы; тем самым он ограничивает гибкость программы.

0.5.8. Интерфейс SQL/CLI

- CLI означает Call-Level Interface.
- Интерфейс прикладного программирования. При этом подходе программа взаимодействует с СУБД с применением набора функций, называемого интерфейсом прикладного программирования (Application Program Interface – API). Вызывая функции API, программа передает в СУБД инструкции SQL и получает обратно результаты запросов. В этом случае специальный препроцессор не требуется.
- Пользователям предоставляется специальная библиотека функций, представляющая собой интерфейс прикладного программирования (API) между приложениями и СУБД. Прикладная программа вызывает функции SQL API для передачи в СУБД инструкций SQL и для получения от СУБД результатов запросов и служебной информации.

0.5.8.1. Интерфейс SQL/CLI (порядок работы)

1. Программа получает доступ к базе данных путем вызова одной или нескольких API-функций, подключающих программу к СУБД, а зачастую - и к конкретной базе данных или схеме.
2. Для пересылки инструкции SQL в СУБД программа формирует инструкцию в виде текстовой строки (зачастую в переменной языка программирования) и затем передает эту строку в качестве параметра при вызове API-функции.
3. Программа вызывает API-функции для проверки состояния переданной в СУБД инструкции и для обработки ошибок.
4. Если инструкция SQL представляет собой запрос на выборку, то при вызове API-функций для получения результатов запроса программа записывает последние в свои переменные; обычно за один вызов возвращается одна строка или один столбец данных.
5. Свое обращение к базе данных программа заканчивает вызовом API-функции, отключающей ее от СУБД.

0.6. Жизненный цикл разработки системы с базой данных

0.6.1. Стадии жизненного цикла разработки системы с базой данных

Стадии жизненного цикла разработки системы с базами данных не идут только последовательно, а могут чередоваться.

- Планирование БД
- Определение системы
- Сбор и анализ требований
- Проектирование БД
  - –Концептуальное
  - –Логическое
    - Выбор СУБД
  - –Физическое
- Прототипирование (и новая итерация проектирования БД)
- Проектирование приложений (параллельно с проектированием БД)
- Реализация
- Преобразование и загрузка данных
- Тестирование
- Функционирование

0.6.2. Проектирование базы данных (1)

Восходящий подход (bottom-up)

- Работа начинается с самого нижнего уровня атрибутов (т. е. свойств сущностей и связей), которые на основе анализа существующих между ними связей группируются в отношения, представляющие типы сущностей и связи между ними.
- *Нормализация* – вариант восходящего подхода. Она предусматривает идентификацию требуемых атрибутов с последующим созданием из них нормализованных таблиц, основанных на функциональных зависимостях между этими атрибутами.
- Восходящий подход в наибольшей степени приемлем для проектирования простых баз данных с относительно небольшим количеством атрибутов.
- Однако использование этого подхода существенно усложняется при проектировании баз данных с большим количеством атрибутов, установить среди которых все существующие функциональные зависимости довольно затруднительно.

0.6.2. Проектирование базы данных (2)

Нисходящий подход (top-down)

- Начинается этот подход с разработки моделей данных, которые содержат несколько высокоуровневых сущностей и связей, затем работа продолжается в виде серии нисходящих уточнений низкоуровневых сущностей, связей и относящихся к ним атрибутов.
- Нисходящий подход демонстрируется в концепции модели «сущность–связь». В этом случае работа начинается с выявления сущностей и связей между ними, интересующих данную организацию в наибольшей степени. Затем сущности дополняются конкретными атрибутами.

Комбинированные подходы

0.6.3. Фазы проектирования базы данных

- *Концептуальное проектирование* – процесс конструирования модели данных, используемых на предприятии, независимой от всех деталей реализации: СУБД, прикладных программ, языков программирования, аппаратной платформы и др. Строится на основе требований пользователей.
- *Логическое проектирование* – процесс конструирования модели данных, используемых на предприятии, на основе конкретной модели (например, реляционной), но независимо от конкретной СУБД и других деталей физической реализации. Концептуальная модель отображается на логическую на основе выбора целевой модели данных, например, реляционной. Мы знаем тип СУБД (реляционная, сетевая, иерархическая, объектно-ориентированная), но не знаем деталей физической реализации хранения данных, индексов и др. Используется нормализация. Нужно показать, что транзакции пользователя выполняются.
- *Физическое проектирование* – процесс реализации БД в среде целевой СУБД. Он описывает отношения, индексы, ограничения целостности.
- Проектирование БД – это итерационный процесс.

0.7. Модель данных «сущность–связь»

0.7.1. Введение (1)

- Чтобы добиться полного понимания характера данных и способов их использования в организации, необходимо применять в процессе обмена информацией между специалистами общую модель, которая не усложнена техническими подробностями и не допускает двойных толкований.
- Структура (схема) БД может быть описана с использованием различных языков и нотаций.
- Наиболее распространенным средством абстрактного представления структур баз данных является ER-модель, или модель «сущность–связь» (entity–relationship model).
- ER-моделирование является представителем так называемого *семантического моделирования*.
- ER-модель была предложена Питером Ченом в 1976 г.
- Существуют и другие методы семантического моделирования.

0.7.1. Введение (2)

- С ER-моделью связана и соответствующая технология построения диаграмм сущностей и связей, которые называются ER-диаграммами. В принципе, возможно использование и других нотаций, даже текстового описания.
- ER-моделирование представляет собой нисходящий подход к проектированию базы данных, который начинается с выявления наиболее важных данных, называемых сущностями (entities), и связей (relationships) между данными, которые должны быть представлены в модели. Затем в модель вносятся дополнительные сведения, например, указывается информация о сущностях и связях, называемая *атрибутами* (attributes), а также все ограничения, относящиеся к сущностям, связям и атрибутам.
- Основные концепции ER-модели:
  - Типы сущностей
  - Типы связей
  - Свойства (атрибуты)

0.7.2. Типы сущностей (1)

- **Тип сущности (entity type)** – группа объектов с одинаковыми свойствами, которая рассматривается в конкретной предметной области как имеющая независимое существование.
- Сущности, выявленные (идентифицированные) в одной и той же предметной области разными разработчиками, могут различаться.
- **Физическое существование:** Работник, Объект недвижимости, Клиент, Деталь, Поставщик, Изделие.
- **Концептуальное существование:** Осмотр объекта недвижимости, Продажа объекта недвижимости, Рабочий стаж.
- **Экземпляр сущности (Entity occurrence)**– однозначно идентифицируемый объект, который относится к сущности определенного типа.
- Если это не приводит к искажению смысла, то вместо терминов *«тип сущности»* и *«экземпляр сущности»* используется более общий термин «*сущность».*

0.7.2. Типы сущностей (2)

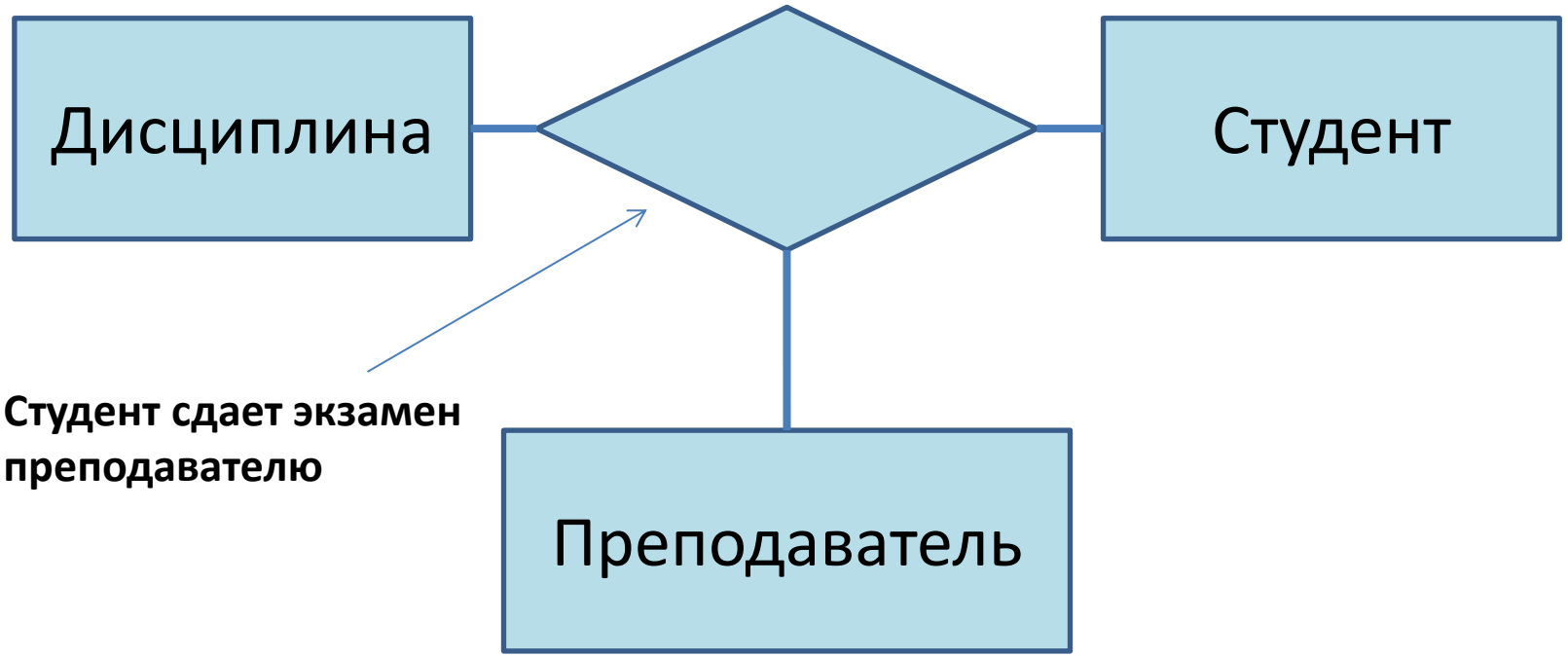
- Каждый тип сущности изображается в виде прямоугольника с именем сущности внутри него.
- В качестве имени обычно применяется существительное в единственном числе.

0.7.3. Типы связей

- **Тип связей (relationship type)** – набор значимых ассоциаций между разными типами сущностей.
- **Экземпляр связи (relationship occurrence)** – однозначно идентифицируемая ассоциация, которая включает по одному экземпляру сущности из каждого участвующего в связи типа сущности.
- *Экземпляр связи* обозначает все конкретные экземпляры сущностей, участвующие в этой связи.
- Если это не приводит к искажению смысла, то вместо терминов *«тип связи»* и *«экземпляр связи»* используется более общий термин *«связь».*
- Имя связи должно являться глаголом. По возможности в каждой конкретной ER-модели все имена связей должны быть уникальными.
- Связи должны иметь *направление*.

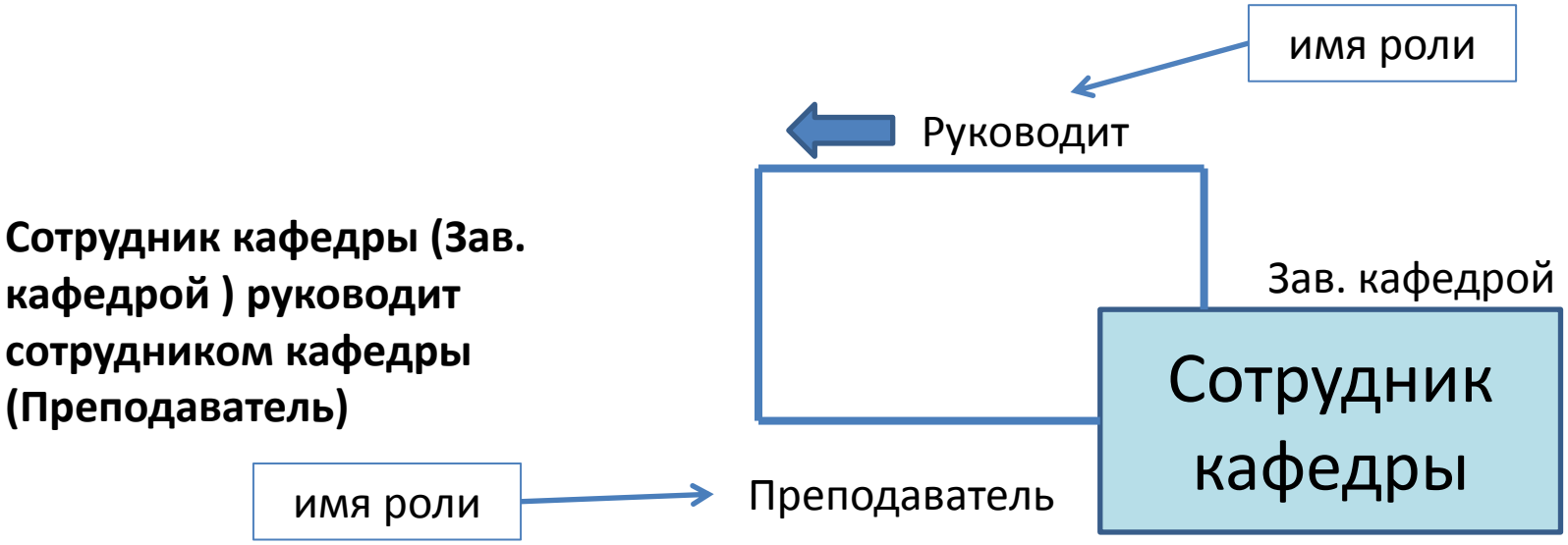
0.7.4. Степень типа связи

- **Степень типа связи (Degree of a relationship type)** – количество типов сущностей, которые охвачены данной связью.
- Если в связи участвует два типа сущностей, то связь называется бинарной, если три типа сущностей – трехсторонней (тернарной).
- Для описания связей со степенью больше двух принято применять термин *сложная связь.*



0.7.5. Рекурсивная связь

- **Рекурсивная связь** – связь, в которой *один и тот же тип* сущности участвует более одного раза в *разных ролях.*
- Связям могут присваиваться *ролевые имена* для указания назначения каждой сущности, участвующей в данной связи.



Сотрудник кафедры (Зав. кафедрой) руководит сотрудником кафедры (Преподаватель)

0.7.6. Атрибуты (1)

- **Атрибут (Attribute)** – свойство типа сущности или типа связи.
- Например, сущность staff (Персонал) может быть описана с помощью атрибутов staffNo (Табельный номер работника), name (Имя), position (Должность) и salary (Зарплата).
- Атрибуты содержат значения, которые описывают каждый экземпляр сущности и составляют основную часть информации, сохраняемой в базе данных.
- Связь, которая соединяет сущности, также может иметь атрибуты, аналогичные атрибутам типа сущности.

0.7.6. Атрибуты (2)

- **Домен атрибута (Attribute domain)** – набор допустимых значений одного или нескольких атрибутов.
- Например, количество комнат в объекте недвижимости может находиться в пределах от 1 до 15 для каждого экземпляра сущности. Следовательно, набор допустимых значений для атрибута rooms (Количество комнат) сущности PropertyForRent можно определить как набор целых чисел от 1 до 15.
- Различные атрибуты могут совместно использовать один и тот же домен.
- Например, атрибуты address (Адрес) типов сущностей Branch (Отделение компании), PrivateOwner (Владелец объекта недвижимости) и BusinessOwner (Владелец делового предприятия) разделяют один и тот же домен, который включает все возможные адреса.
- Атрибуты подразделяются на *простые* и *составные, однозначные* и *многозначные,* а также *производные.*

0.7.6.1. Простые и составные атрибуты

- **Простой атрибут (Simple attribute)** – атрибут, состоящий из одного компонента с независимым существованием.
- *Простые* атрибуты не могут быть разделены на более мелкие компоненты. Примером простых атрибутов является атрибут position (Должность) или salary (Зарплата) сущности Staff. Простые атрибуты иногда называют *элементарными.*
- **Составной атрибут** (**Composite attribute**) – атрибут, состоящий из нескольких компонентов, каждый из которых характеризуется независимым существованием.
- Некоторые атрибуты могут быть разделены на более мелкие компоненты, которые характеризуются независимым существованием. Например, атрибут address (Адрес) сущности Branch, представляющей отделение компании, со значением ' 123045, г. Киров, ул. Новая,163' может быть разбит на отдельные атрибуты street ('Новая, 163 '), city (' Киров ') и postcode (' 123045 ').
- Решение о моделировании атрибута address в виде простого атрибута или разбиении его на атрибуты street, city и postcode зависит от того, как рассматривается атрибут address в пользовательском представлении — как единое целое или как набор отдельных компонентов.

0.7.6.2. Однозначные и многозначные атрибуты

- **Однозначный атрибут (Single-valued attribute)** – атрибут, который содержит одно значение для каждого экземпляра сущности определенного типа.
- Большинство атрибутов являются однозначными. Например, для каждого отдельного экземпляра сущности Branch всегда имеется единственное значение в атрибуте номера отделения компании (branchNo), скажем, 'B003'.
- **Многозначный атрибут (Multi-valued attribute)** – атрибут, который содержит несколько значений для каждого экземпляра сущности определенного типа.
- Например, сущность Branch может иметь несколько значений для атрибута telNo (Номер телефона отделения компании). Следовательно, атрибут telNo в этом случае будет многозначным.
- Многозначный атрибут допускает присутствие определенного количества значений (возможно, в заданных пределах, определяющих максимальное и минимальное количество). Например, атрибут telNo отделения компании может иметь от одного до трех значений.

0.7.6.3. Производные атрибуты

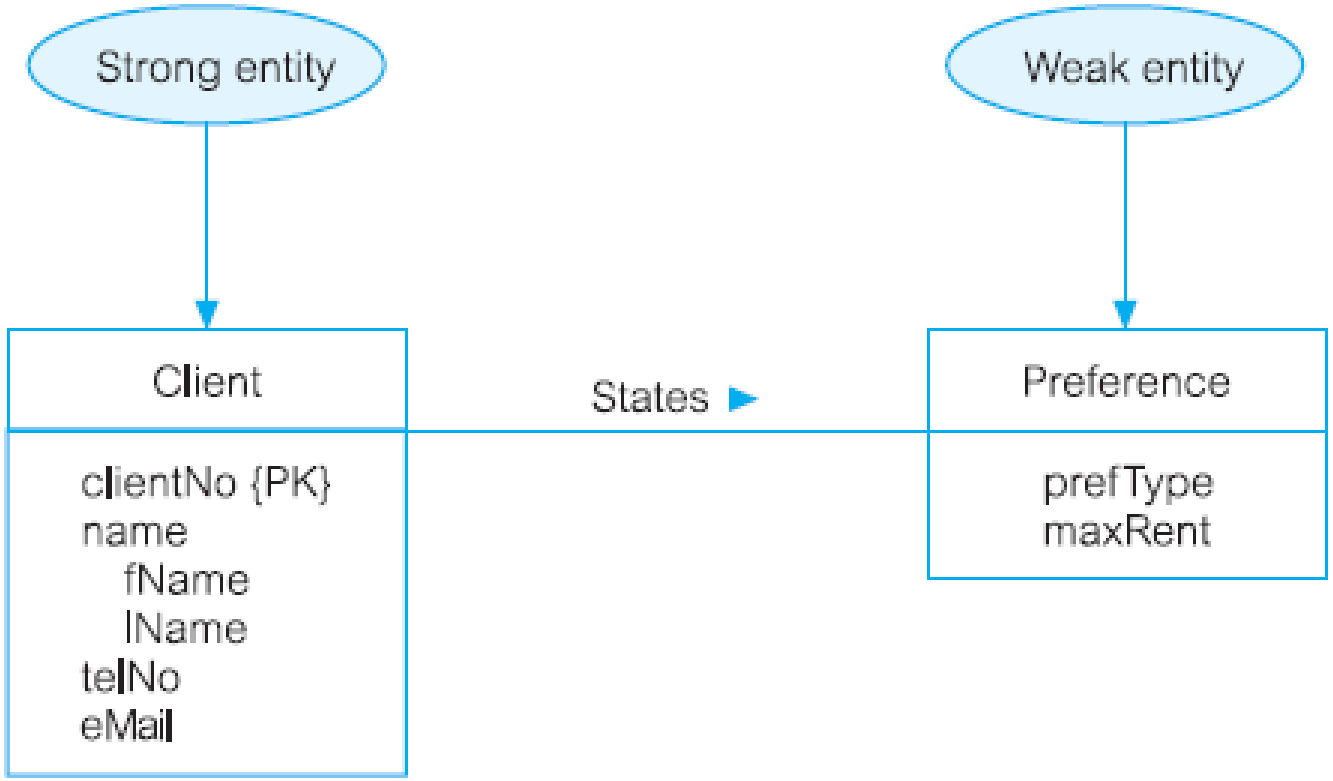
- **Производный** атрибут (**Derived attribute**) – атрибут, который представляет значение, производное от значения связанного с ним атрибута или некоторого множества атрибутов, принадлежащих некоторому (не обязательно данному) типу сущности.
- Например, значение атрибута duration (Срок действия) сущности Lease (Договор аренды) вычисляется на основе атрибутов rentstart (Начало срока аренды) и rentFinish (Конец срока аренды), которые также относятся к типу сущности Lease.
- В некоторых случаях значение атрибута является производным от многих экземпляров сущности одного и того же типа. Например, атрибут totalStaff (Общее количество сотрудников) сущности типа staff может быть вычислен на основе подсчета общего количества экземпляров сущности staff .

0.7.7. Ключи

- **Потенциальный ключ** (**Candidate key**) – атрибут или минимальный набор атрибутов, который однозначно идентифицирует каждый экземпляр типа сущности.
- Потенциальный ключ не может содержать значения NULL.
- **Первичный ключ (Primary key)** – потенциальный ключ, который выбран для однозначной идентификации каждого экземпляра сущности определенного типа.
- Выбор первичного ключа сущности осуществляется с учетом суммарной длины атрибутов, минимального количества необходимых атрибутов в ключе, а также наличия гарантий уникальности его значений в текущий момент времени и в обозримом будущем.
- **Составной ключ (Composite key)** – потенциальный ключ, который состоит из двух или нескольких атрибутов.

0.7.8. Сущности сильного и слабого типов (1)

- **Сущность сильного типа (Strong entity type)** – тип сущности, существование которого не зависит от какого-то иного типа сущности.
- **Сущность слабого типа (Weak entity type)** – тип сущности/существование которого зависит от какого-то другого типа сущности.



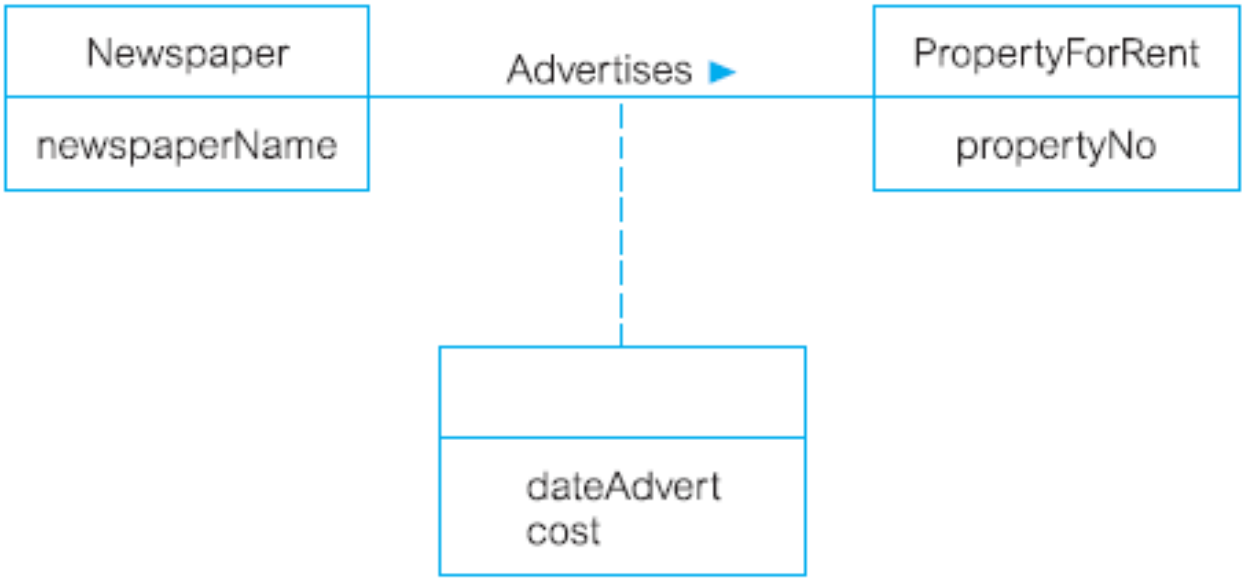
0.7.8. Сущности сильного и слабого типов (2)

- Характерной особенностью слабой сущности является то, что каждый экземпляр сущности нельзя однозначно идентифицировать с помощью только тех атрибутов, которые относятся к сущности этого типа. Например, обратите внимание, что для сущности Preference нет первичного ключа. Это означает, что каждый экземпляр сущности типа Preference невозможно идентифицировать только с помощью атрибутов этой сущности. Однозначно идентифицировать каждое пожелание клиента можно только, учитывая связь конкретного пожелания с некоторым клиентом, который однозначно идентифицируется с использованием первичного ключа для сущности типа Client, а именно: clientNo. В данном примере сущность Preference рассматривается как зависимая в своем существовании от сущности Client, которая описывает будущего арендатора, желающего арендовать объекты недвижимости конкретного типа и на определенных условиях.
- Сущности слабого типа иногда называют *дочерними, зависимыми* или *подчиненными,* а сущности сильного типа — *родительскими, сущностями-владельцами* или *доминантными.*

0.7.9. Атрибуты связей

- Чтобы подчеркнуть различие между сущностью и связью, обладающей атрибутом, линия, которая соединяет прямоугольник с именем атрибута (атрибутов) и саму связь, отображается как штриховая.

Газета рекламирует арендуемый объект недвижимости

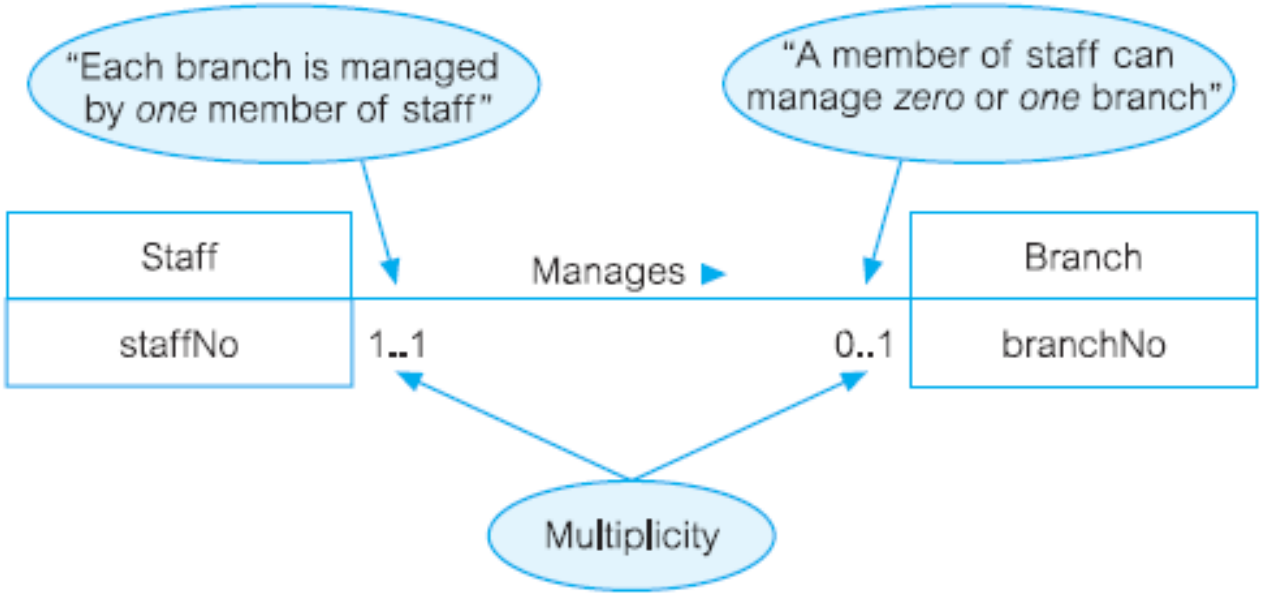


0.7.10. Структурные ограничения

- Основным типом ограничения, накладываемого на связь, является кратность.
- **Кратность (Multiplicity)** – количество (заданное как одно значение или как диапазон значений), возможных экземпляров сущности некоторого типа, которые могут быть связаны с одним экземпляром сущности другого типа с помощью определенной связи.
- Ограничения кратности описывают способ формирования связи между сущностями. Они отражают требования (или бизнес-правила), установленные пользователем или предприятием.
- Наиболее распространенной степенью связи является двухсторонняя. Двухсторонние связи обычно обозначаются как связи
  - **«один к одному» (1:1)**
  - **«один ко многим» (1:*)**
  - **или «многие ко многим» (*:*)**.
- Важно отметить, что не все ограничения предметной области могут быть легко представлены в виде ER-модели. Например, с помощью ER-модели сложно представить условие, согласно которому сотрудник компании получает дополнительный день отпуска за каждый год стажа работы в компании.

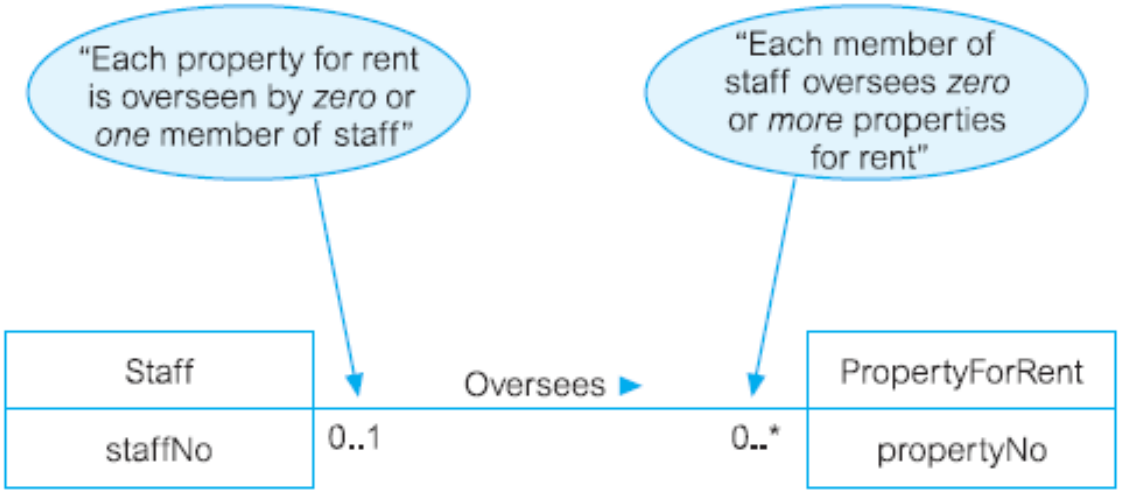
0.7.10.1. Связь «один к одному»

- ER-диаграмма связи Staff *Manages* Branch.
- Чтобы показать, что под управлением любого сотрудника компании может находиться нуль или одно отделение, на этой схеме рядом с сущностью Branch помещено обозначение 0. .1.
- А для указания на то, что в любом отделении всегда имеется один менеджер, рядом с обозначением сущности Staff помещено обозначение 1. .1.



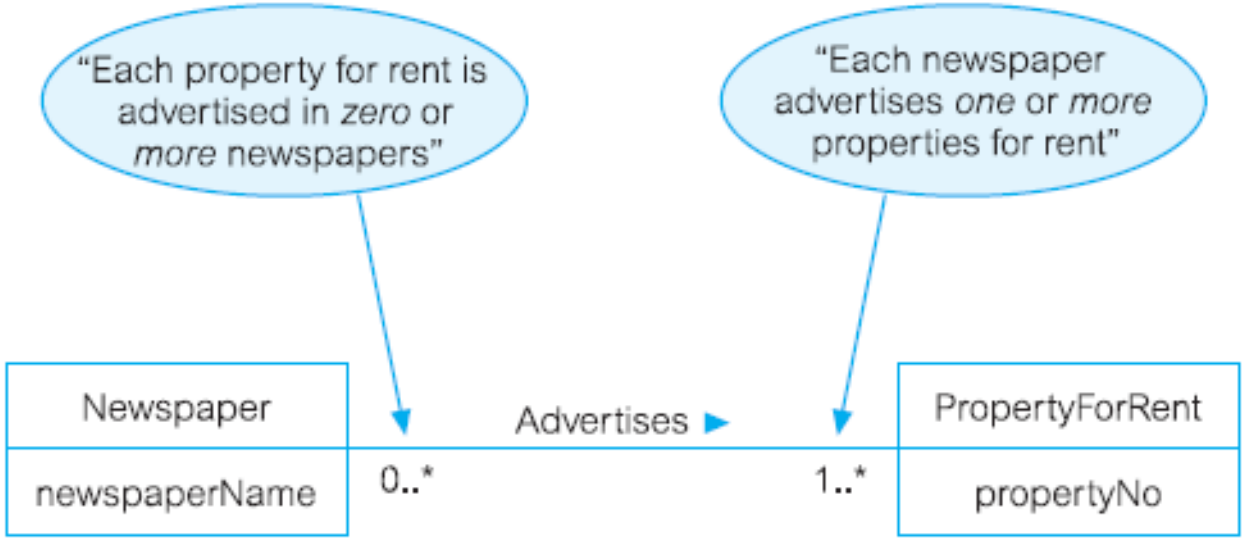
0.7.10.2. Связь «один ко многим»

- ER-диаграмма связи Staff *Oversees* PropertyForRent.
- Для указания того, что количество арендуемых объектов недвижимости, находящихся под управлением любого сотрудника компании, может составлять от нуля и больше, на этой схеме рядом с изображением сущности PropertyForRent помещено обозначение 0. . *.
- А для указания на то, что каждым арендуемым объектом недвижимости управляет нуль или один сотрудник компании, рядом с изображением сущности Staff помещено обозначение 0. . 1.
- Для связей типа 1:* должно быть выбрано имя, которое имеет смысл, если связь рассматривается в направлении от «одного» ко «многим».



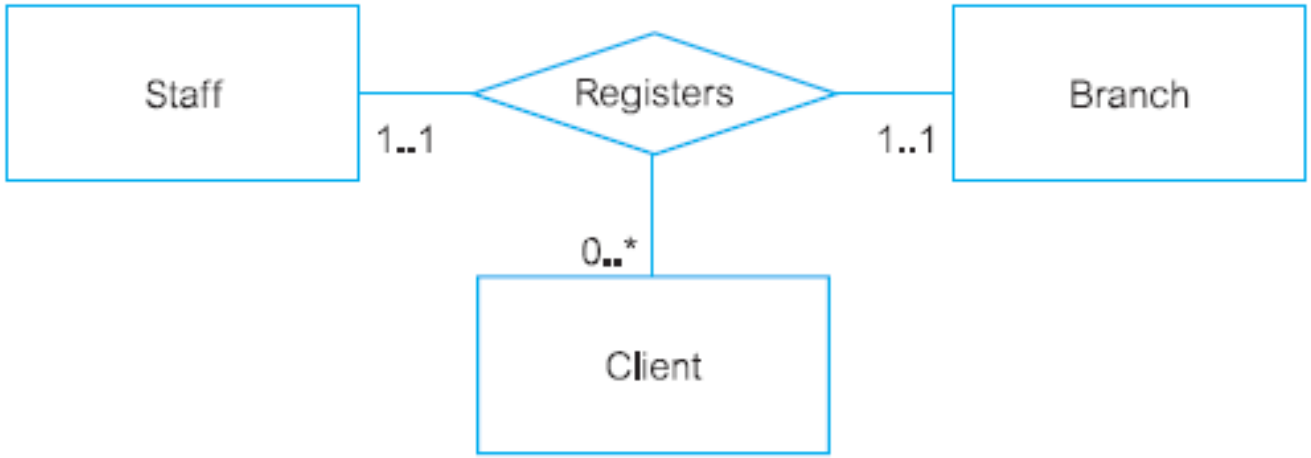
0.7.10.3. Связь «многие ко многим»

- ER-диаграмма связи Newspaper *Advertises* PropertyForRent.
- Для указания на то, что каждая газета может публиковать рекламу об одном или нескольких объектах недвижимости, сдаваемых в аренду, рядом с изображением сущности PropertyForRent показано обозначение 1..*.
- Для указания на то, что количество газет, в которых публикуется реклама о сдаваемых в аренду объектах недвижимости, может составлять нуль и больше, рядом с изображением сущности Newspaper помещено обозначение 0. . *.
- Для связи *:* может быть выбрано имя, которое имеет смысл при рассмотрении этой связи либо в том, либо в ином направлении.



0.7.10.4. Кратность сложных связей

- **Кратность сложной связи** – количество (заданное как одно значение или как диапазон значений) экземпляров сущности определен него типа в n-арной связи, определяемое после фиксации остальных (n – 1) значений.



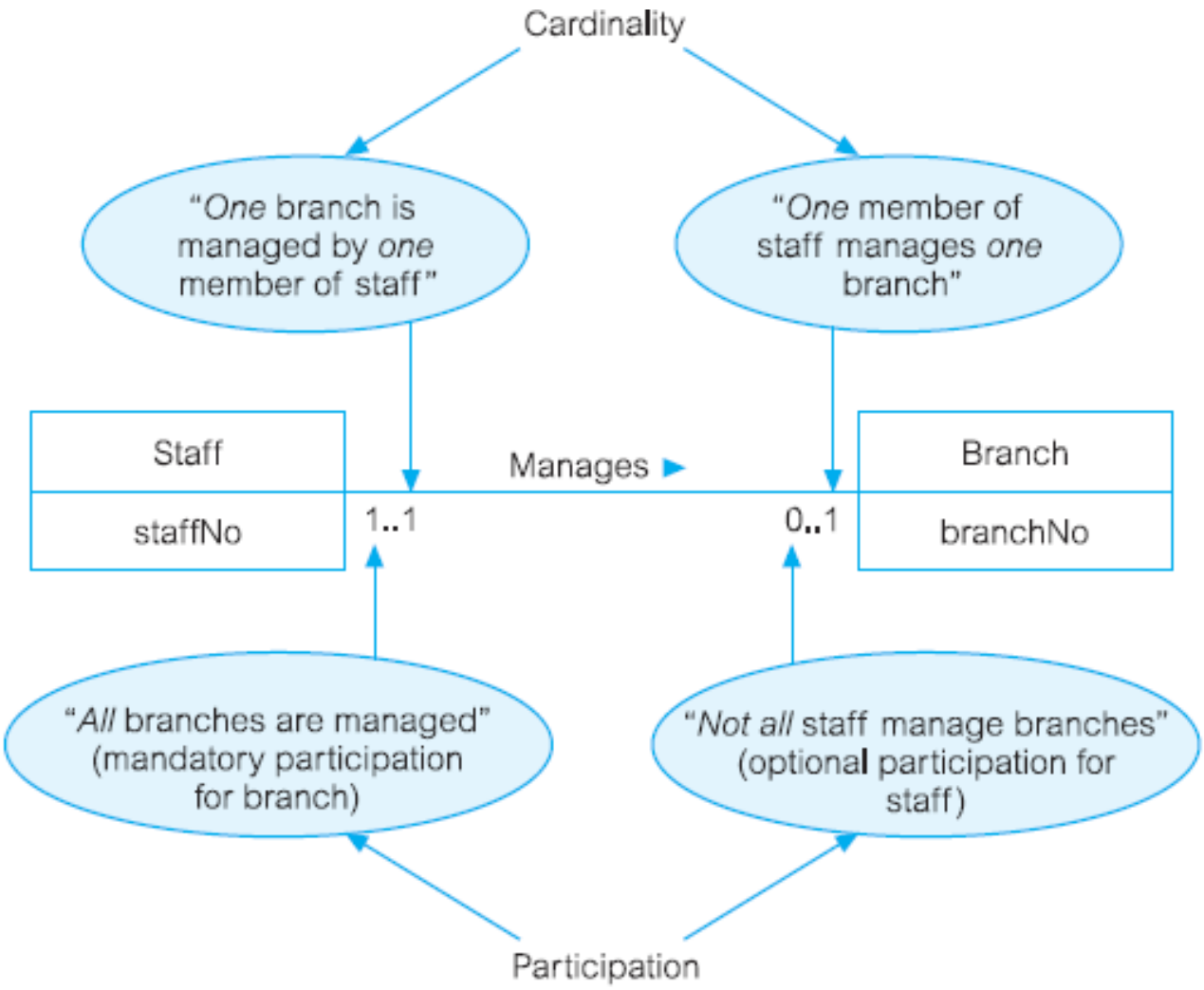
0.7.10.5. Ограничения кардинальности и степени участия (1)

- Ограничения кратности фактически состоят из двух отдельных ограничений, известных как *кардинальность* и *степень участия.*
- **Кардинальность (Cardinality)** – определяет максимальное количество возможных экземпляров связи для каждой сущности, участвующей в связи конкретного типа.
- Понятие *кардинальности двухсторонней связи* является обобщением понятия *кратности,* которое выше рассматривалось как связь «один к одному» (1:1), «один ко многим» (1:*) и «многие ко многим» (*:*).
- **Степень участия (Participation)** – определяет, участвуют ли в связи все или только некоторые экземпляры сущности.
- Ограничение степени участия определяет, должны ли участвовать в конкретной связи все экземпляры сущности (такое условие принято называть *обязательным участием)* или только некоторые экземпляры (такое условие называется *необязательным участием).*

0.7.10.5. Ограничения кардинальности и степени участия (2)

- Степень участия сущностей в связи проявляется в виде минимальных значений для диапазонов кратности на каждой стороне связи. Необязательное участие представляется минимальным значением 0, а обязательное участие – минимальным значением 1.
- Важно учитывать, что степень участия конкретной сущности в связи представлено минимальным значением на *противоположной* стороне этой связи, т. е. минимальным значением кратности, стоящим возле связанной сущности.

0.7.10.5. Ограничения кардинальности и степени участия (3)



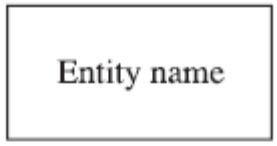
0.7.11. Расширенная модель «сущность–связь»

- ER-модель, оснащенная дополнительными семантическими концепциями, называется *расширенной моделью «сущность–связь»* (Enhanced Entity-Relationship — EER).
- Наиболее важные и полезные дополнительные концепции EER-модели:
  - уточнение/обобщение (specialization/generalization)
  - агрегирование (aggregation)
  - композиция (composition)

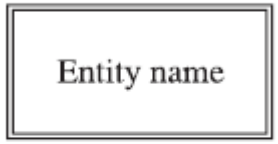
0.7.12. Альтернативные нотации

- Мы использовали нотацию (систему обозначений), основанную на языку UML (Unified Modeling Language).
- Существуют другие нотации:
  - Нотация П. Чена
  - Нотация «вороньи лапки» (Crow’s Feet)

0.7.12.1. Нотация П. Чена (1)



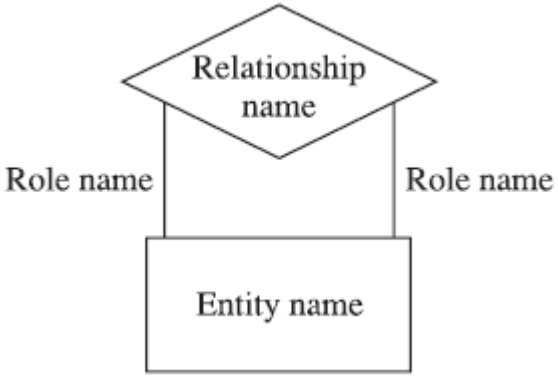
Сильная сущность



Слабая сущность

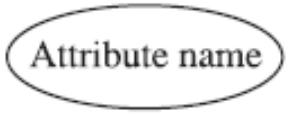
Связь

Связь, ассоциированная со слабой сущностью

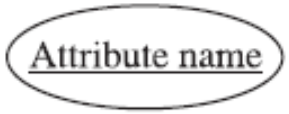


Рекурсивная связь с именами ролей, указывающими, какие функции выполняет сущность в связи

0.7.12.1. Нотация П. Чена (2)

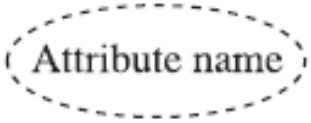


Атрибут

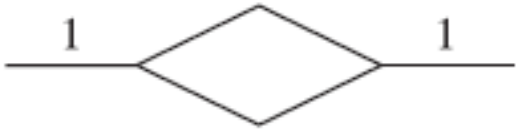


Атрибут первичного ключа

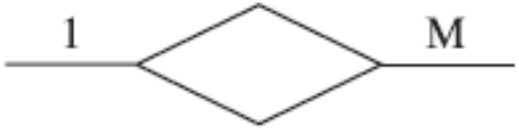
Многозначный атрибут



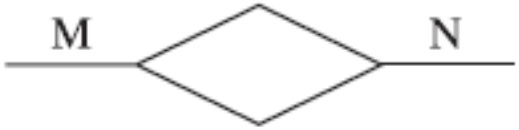
Производный атрибут



Связь «один к одному» (1:1)



Связь «один ко многим» (1:M)



Связь «многие ко многим» (M:N)

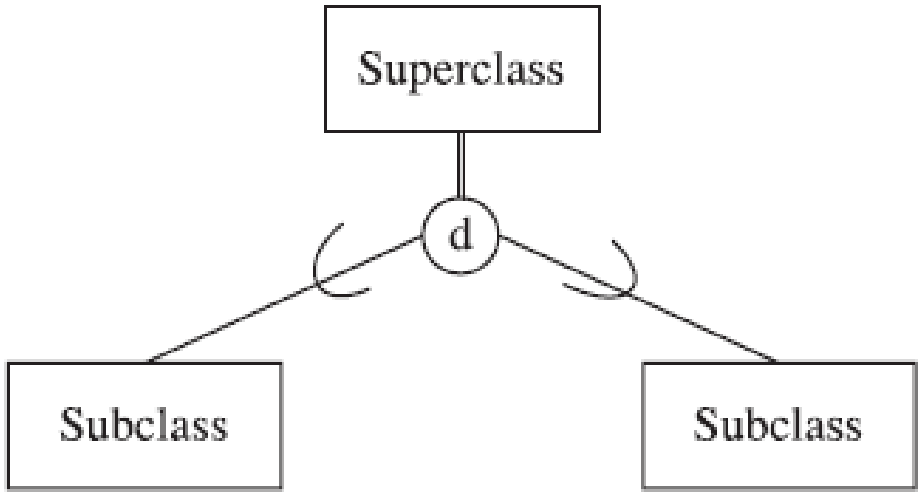
0.7.12.1. Нотация П. Чена (3)

Связь «один ко многим» с обязательным участием сущностей A и B

Связь «один ко многим» с необязательным участием сущности A и обязательным участием сущности B

Связь «один ко многим» с необязательным участием сущностей A и B

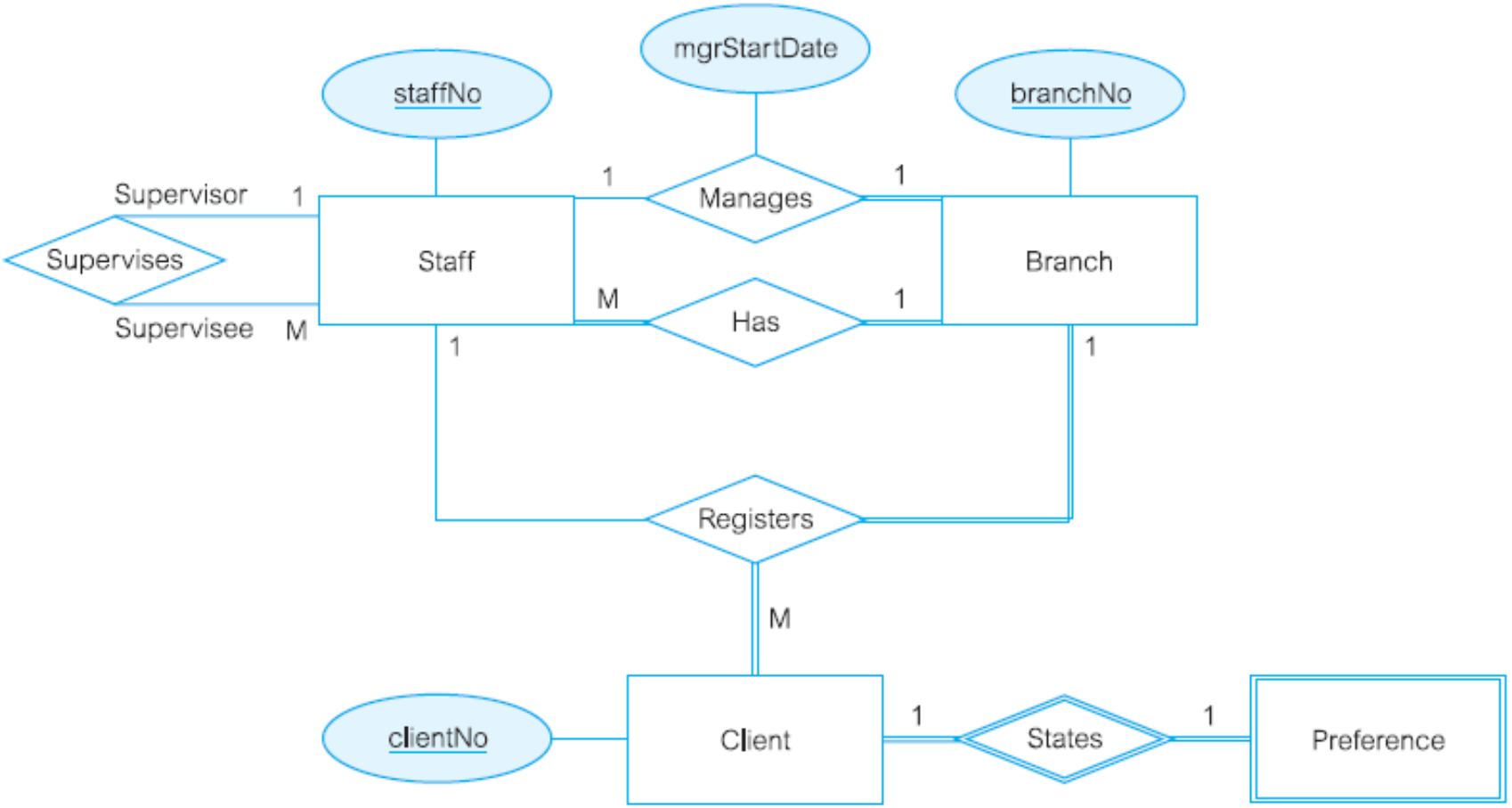
0.7.12.1. Нотация П. Чена (4)



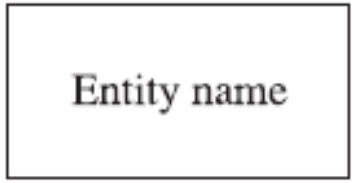
Уточнение/обобщение

Если в кружке стоит буква «d», связь является непересекающейся,
а если в кружке стоит буква «o», связь не является непересекающейся.
Двойная линия от суперкласса к кружку обозначает обязательное участие,
одинарная линия обозначает необязательное участие.

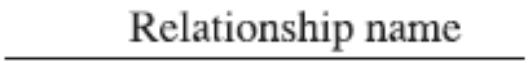
0.7.12.1. Нотация П. Чена (пример)



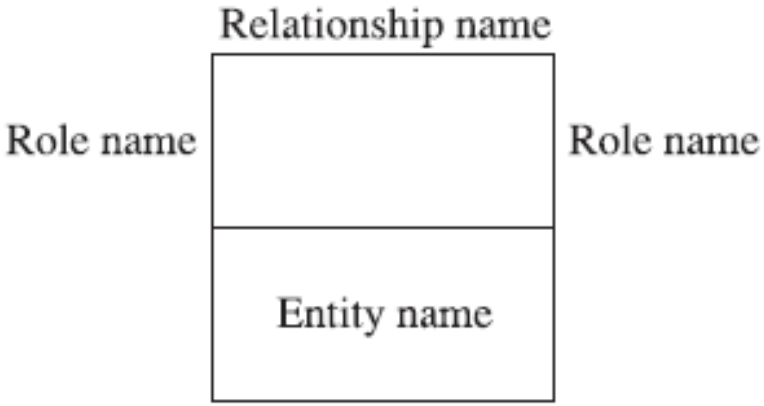
0.7.12.2. Нотация «вороньи лапки» (1)



Сущность

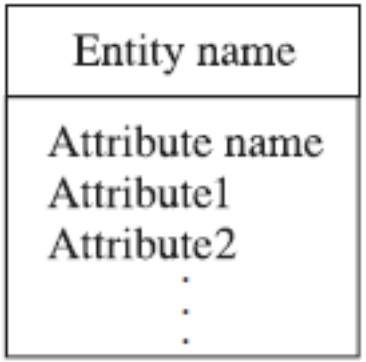


Связь

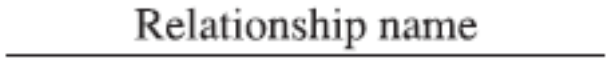


Рекурсивная связь с именами ролей, указывающими, какие функции выполняет сущность в связи

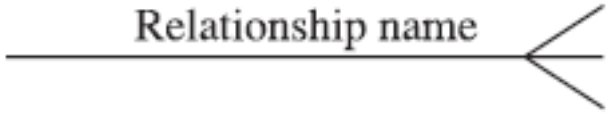
0.7.12.2. Нотация «вороньи лапки» (2)



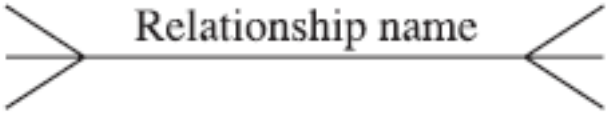
Атрибуты перечисляются в нижней части обозначения сущности.
Атрибуты первичного ключа подчеркиваются.
Многозначный атрибут помещается в фигурные скобки ({}).



Связь «один к одному»

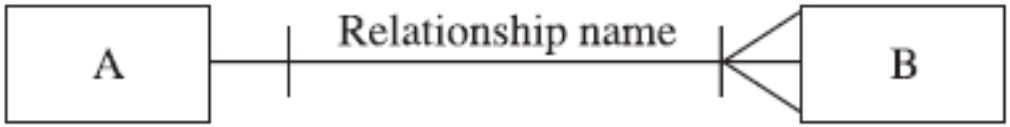


Связь «один ко многим»

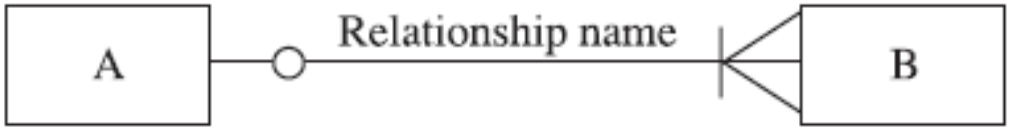


Связь «многие ко многим»

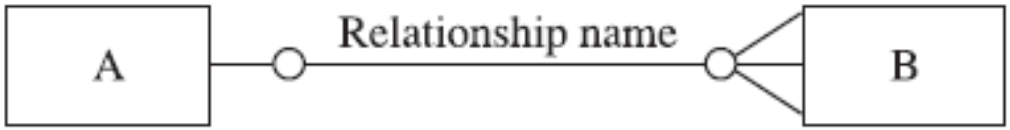
0.7.12.2. Нотация «вороньи лапки» (3)



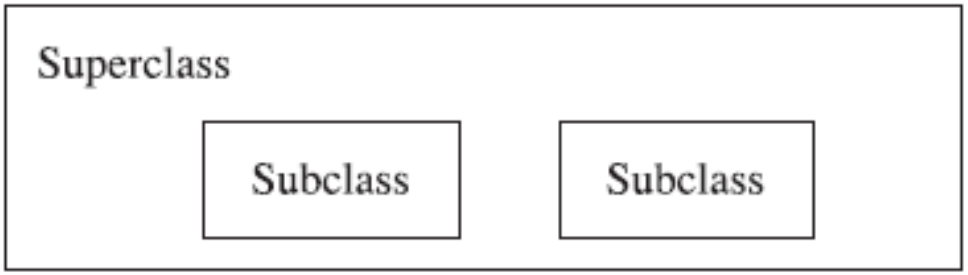
Связь «один ко многим» с обязательным участием сущностей A и B



Связь «один ко многим» с необязательным участием сущности A и обязательным участием сущности B

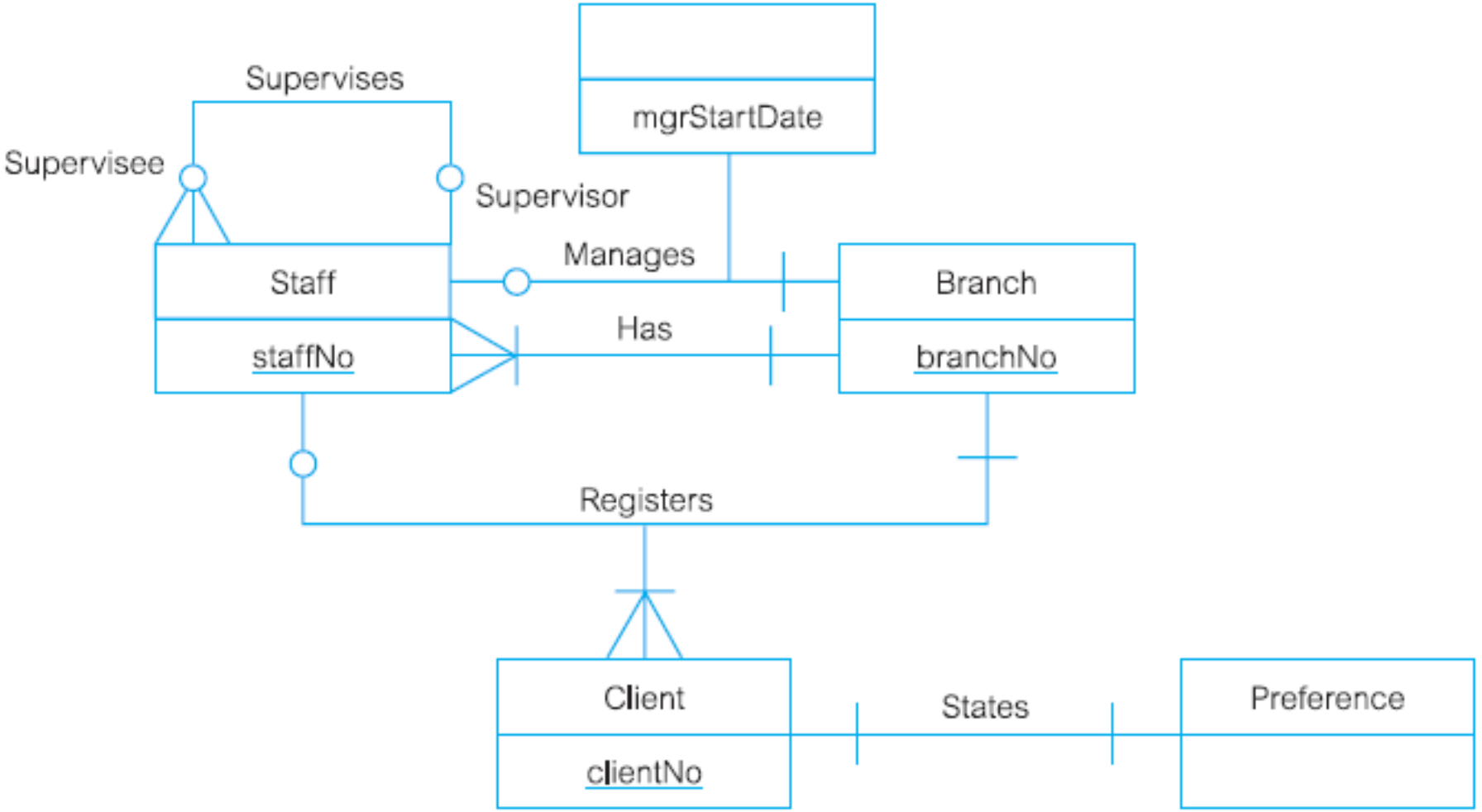


Связь «один ко многим» с необязательным участием сущностей A и B



Для представления иерархии уточнения/обобщения используются прямоугольники, заключенные в прямоугольник

0.7.12.2. Нотация «вороньи лапки» (пример)



0.8. Нормализация

0.8.1. Введение (1)

- **Нормализация.** Метод создания набора отношений с заданными свойствами на основе требований к данным, установленных в некоторой организации.
- Цель: идентифицировать подходящий набор отношений, который будет адекватно поддерживать требования к данным некоторого предприятия:
  - Минимальное количество атрибутов
  - Атрибуты, которые логически тесно связаны, должны находиться в одном отношении
  - Минимальная избыточность, т. е. каждый атрибут должен быть представлен только один раз, за исключением атрибутов внешних ключей

0.8.1. Введение (2)

- Нормализация – это формальный метод, который можно использовать на любой стадии процесса проектирования.
- I подход – автономный метод проектирования базы данных «снизу-вверх».
- II подход – метод проверки структуры отношений, созданных с помощью одного из методов проектирования «сверху-вниз», например, ER-моделирования.
- Общее назначение процесса нормализации заключается в следующем:

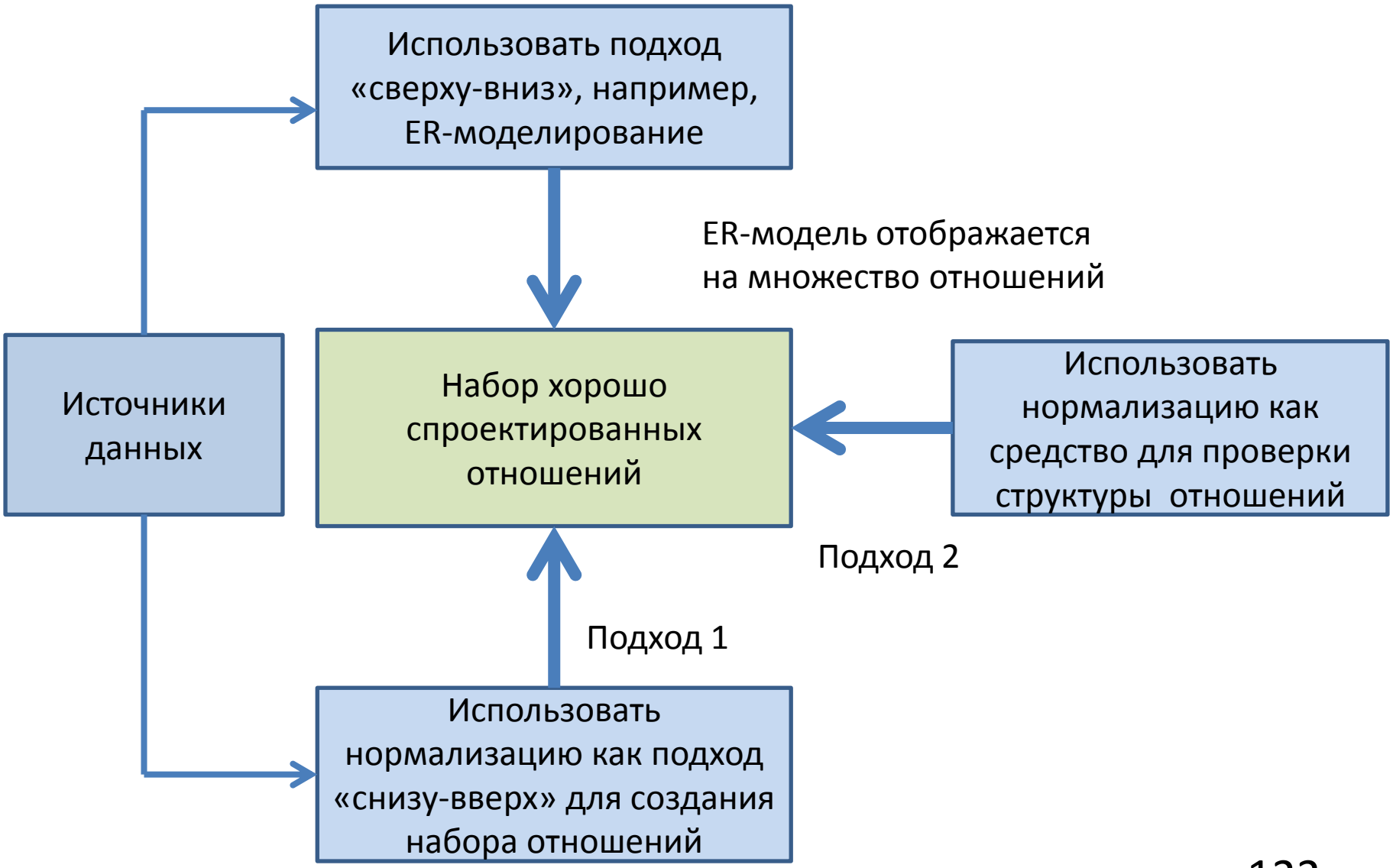
❑ исключение некоторых типов избыточности;

❑ устранение некоторых аномалий обновления (см. далее);

❑ разработка проекта базы данных, который является достаточно «качественным» представлением реального мира, интуитивно понятен и может служить хорошей основой для последующего расширения;

❑ упрощение процедуры применения необходимых ограничений целостности.

0.8.1. Введение (3)



0.8.2. Избыточность данных и аномалии обновления (1)

- При проектировании реляционной базы данных важным является группирование атрибутов для минимизации избыточности данных.
- Потенциальные преимущества такой базы данных включают:
  - Обновления данных в БД будут производиться с использованием минимального числа операций, тем самым снижая возможности для возникновения несогласованности данных в БД
  - Снижение размера хранилища данных, следовательно снижение стоимости хранения данных
- *Избыточность присутствует в виде внешних ключей*, значения которых совпадают со значениями первичных ключей или потенциальных ключей в ссылочных таблицах. Таким образом моделируются взаимосвязи между данными.

0.8.2. Избыточность данных и аномалии обновления (2)

Staff (staffNo, sName, position, salary, branchNo)
Branch (branchNo, bAddress)
StaffBranch (staffNo, sName, position, salary, branchNo, bAddress)

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

В отношении StaffBranch есть избыточность данных: сведения о филиале повторяются для каждого работника этого филиала.

При работе с отношениями, содержащими избыточные данные, могут возникать проблемы, которые называются *аномалиями обновления* и подразделяются на аномалии *вставки, удаления* и *модификации.*

0.8.2.1. Аномалии вставки

Тип 1. Дублирование сведений

- При вставке сведений о новых сотрудниках в отношение StaffBranch необходимо указать и сведения об отделении компании, в котором эти сотрудники работают. Например, при вставке сведений о новом сотруднике отделения 'B007' требуется ввести сведения о самом отделении 'B007', которые должны соответствовать сведениям об этом же отделении в других строках отношения StaffBranch.

Тип 2. Невозможность ввода сведений, т. к. появляются пустые значения

- Для вставки сведений о новом отделении компании, которое еще не имеет собственных сотрудников, требуется присвоить значение NULL всем атрибутам описания персонала отношения StaffBranch, включая и табельный номер сотрудника staffNo. Но поскольку атрибут staffNo является первичным ключом отношения StaffBranch, то попытка ввести значение NULL в атрибут staffNo вызовет нарушение целостности сущностей и потому будет отклонена. Следовательно, в отношение StaffBranch невозможно ввести строку о новом отделении компании, содержащую значение NULL в атрибуте staffNo.

Схема с двумя отношениями Branch и Staff избавляет нас от этих аномалий.

0.8.2.2. Аномалии удаления

- При удалении из отношения StaffBranch строки с информацией о последнем сотруднике некоторого отделения компании сведения об этом отделении будут полностью удалены из базы данных.
- Например, после удаления из отношения StaffBranch строки для сотрудника 'Mary Howe' с табельным номером ' SA9‘ из базы данных неявно будут удалены все сведения об отделении с номером B0071.
- Однако структура отношений Staff и Branch позволяет избежать возникновения этой проблемы, поскольку строки со сведениями об отделениях компании хранятся отдельно от строк со сведениями о сотрудниках. Связывает эти два отношения только общий атрибут branchNo. При удалении из отношения Staff строки с номером сотрудника ' SA9' сведения об отделении 'BO07 ' в отношении Branch останутся нетронутыми.

0.8.2.3. Аномалии модификации (1)

- При попытке изменения значения одного из атрибутов для некоторого отделения компании в отношении StaffBranch (например, адреса отделения 'B003 ') необходимо обновить соответствующие значения в строках для всех сотрудников этого отделения.
- Если такой модификации будут подвергнуты не все требуемые строки отношения StaffBranch, база данных будет содержать противоречивые сведения.
- В частности, в нашем примере для отделения компании с номером 'B003' в строках, относящихся к разным сотрудникам, ошибочно могут быть указаны разные значения адреса этого отделения.

0.8.2.3. Аномалии модификации (2)

- Отношение StaffBranch подвержено аномалиям обновления, но этих аномалий можно избежать путем декомпозиции первоначального отношения на отношения Staff и Branch.
- С декомпозицией крупного отношения на более мелкие связаны два важных свойства.
- **Свойство 1**. Соединение без потерь (lossless-join).
- Гарантирует, что любой экземпляр первоначального отношения может быть определен с помощью соответствующих экземпляров более мелких отношений
- **Свойство 2.** Сохранение зависимостей (dependency preservation).
- Гарантирует, что ограничения на первоначальное отношение можно поддерживать, просто применяя некоторые ограничения к каждому из более мелких отношений.
- Иными словами, для проверки того, не нарушается ли ограничение, которое распространялось на первоначальное отношение, нет необходимости выполнять операции соединения на более мелких отношениях.

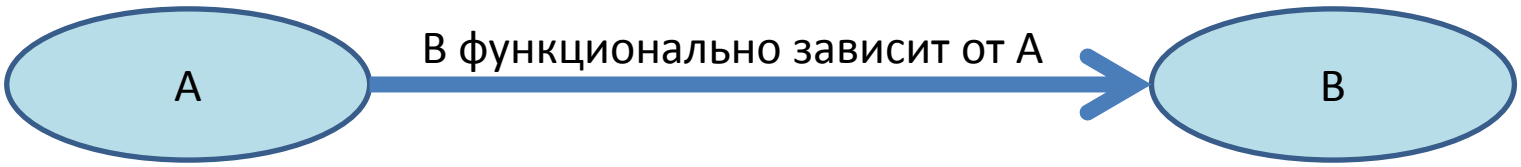
0.8.3. Функциональные зависимости (1)

- *Функциональная зависимость* (functional dependency) описывает связь между атрибутами и является одним из основных понятий нормализации.
- **Функциональная зависимость.** Описывает связь между атрибутами отношения. Например, если в отношении R, содержащем атрибуты A и B, атрибут B функционально зависит от атрибута A (что обозначается как A –> B), то каждое значение атрибута A связано только с одним значением атрибута B. (Причем атрибуты A и B могут быть и составными.)
- Если нам известно значение атрибута A, то при рассмотрении отношения с такой зависимостью в любой момент времени во всех строках этого отношения, содержащих указанное значение атрибута A, мы найдем одно и то же значение атрибута B. Таким образом, если две строки имеют одно и то же значение атрибута A, то они обязательно имеют одно и то же значение атрибута B. Однако для заданного значения атрибута B может существовать несколько различных значений атрибута A.

0.8.3. Функциональные зависимости (2)

- Функциональная зависимость является смысловым (или семантическим) свойством атрибутов отношения.
- Функциональная зависимость может быть транзитивной:

  если A –> B и B –> C, то A –> C.



- **Детерминант.** Детерминантом функциональной зависимости называется атрибут или группа атрибутов, расположенная на диаграмме функциональной зависимости слева от стрелки.
- Атрибут A является детерминантом атрибута B.

0.8.3. Функциональные зависимости (3)

- Детерминанты функциональных зависимостей должны иметь *минимальное* число атрибутов, достаточное для того, чтобы поддерживать функциональную зависимость с атрибутами на правой стороне зависимости. Это требование называется **полной функциональной зависимостью**.
- **Полная функциональная зависимость.** Показывает, что, если A и B являются атрибутами какого-то отношения, то B *полностью* функционально зависит от A, если B функционально зависит от A, но не зависит от любого собственного подмножества A.
  ПРИМЕЧАНИЕ. Атрибуты A и B могут быть составными.
- Функциональная зависимость A –> B является **полной функциональной** зависимостью, если удаление какого-либо атрибута из множества A приводит к тому, что зависимость прекращает существовать.
- Функциональная зависимость A –> B является **частичной зависимостью**, если существует какой-то атрибут, который можно удалить из множества A и при этом зависимость сохранится.
- Пример частичной зависимости: staffNo, sName –> branchNo

0.8.4. Процесс нормализации (1)

- Нормализация — это формальный метод анализа отношений на основе их первичного ключа (или потенциальных ключей) и существующих функциональных зависимостей.
- Если некоторое требование не удовлетворяется, то противоречащее данному требованию отношение должно быть разделено на отношения, каждое из которых (в отдельности) удовлетворяет всем требованиям нормализации.
- Чаще всего нормализация осуществляется в виде нескольких последовательно выполняемых этапов, каждый из которых соответствует определенной нормальной форме, обладающей известными свойствами. В ходе нормализации формат отношений становится все более ограниченным (строгим) и менее восприимчивым к аномалиям обновления.
- Для создания отношений приемлемого качества обязательно только выполнение требований первой нормальной формы (1НФ). Все остальные формы могут использоваться по желанию проектировщиков. Как правило, добиваются третьей нормальной формы (3НФ), чтобы избежать аномалий обновления.
- Нормализация позволяет **избежать избыточности** и, следовательно, возникновения некоторых аномалий обновления.

0.8.4. Процесс нормализации (2)

- Процесс нормализации заключается в замене данной переменной отношения некоторым набором ее **проекций,** составленным таким образом, чтобы обратное соединение этих проекций позволяло вновь получить исходную переменную отношения. Иначе говоря, этот процесс является обратимым, т. е. декомпозиция всегда выполняется без потерь информации.
- В дальнейших примерах предполагается, что множество функциональных зависимостей уже даны и первичные ключи определены.
- Важно, чтобы к началу выполнения нормализации были уже хорошо понятны значения всех атрибутов и взаимосвязей между ними. Эта информация используется для проверки того, находится ли некоторое отношение в конкретной нормальной форме.

0.8.5. Ненормализованная форма (ННФ) (1)

- **Ненормализованная форма (ННФ).** Таблица находится в ненормализованной форме, если она содержит одну или несколько повторяющихся групп данных.
- *Повторяющейся группой* называется группа, состоящая из одного или нескольких атрибутов таблицы, в которой возможно наличие нескольких значений для единственного значения ключевого атрибута (атрибутов) таблицы.
- Два способа устранения повторяющихся групп:

1. При первом способе повторяющиеся группы устраняются путем ввода соответствующих данных в пустые столбцы строк с повторяющимися данными. Иначе говоря, пустые места при этом заполняются дубликатами неповторяющихся данных. Этот способ часто называют «выравниванием» («flattening») таблицы.
2. При втором способе один атрибут или группа атрибутов назначаются ключом ненормализованной таблицы, а затем повторяющиеся группы изымаются и помещаются в отдельные отношения вместе с копиями ключа исходной таблицы. Далее в новых отношениях устанавливаются свои первичные ключи.

0.8.5. Ненормализованная форма (ННФ) (2)

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	50	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-11	10-June-12	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Повторяющаяся группа = (propertyNo, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

0.8.6. Первая нормальная форма (1НФ) (1)

- **Первая нормальная форма (1НФ).** Отношение, в котором на пересечении каждой строки и каждого столбца содержится одно и только одно значение.
- Для преобразования ненормализованной таблицы в первую нормальную форму (1НФ) в исходной таблице следует найти и устранить все повторяющиеся группы данных.

0.8.6. Первая нормальная форма (1НФ) (2)

Способ 1

- Повторяющиеся группы устраняются путем ввода соответствующих данных в пустые столбцы строк с повторяющимися данными. Иначе говоря, пустые места при этом заполняются дубликатами неповторяющихся данных.
- Потенциальные ключи этого отношения являются составными: (clientNo, propertyNo), (clientNo, rentstart), (propertyNo, rentstart). В качестве первичного ключа выберем (clientNo, propertyNo) и разместим атрибуты первичного ключа в левой части отношения.

ClientRental

clientNo	propertyNo	cName	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	John Kay	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
CR76	PG16	John Kay	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	450	CO93	Tony Shaw
CR56	PG4	Aline Stewart	6 Lawrence St, Glasgow	1-Sep-11	10-Jun-12	350	CO40	Tina Murphy
CR56	PG36	Aline Stewart	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
CR56	PG16	Aline Stewart	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

0.8.6. Первая нормальная форма (1НФ) (3)

Способ 2

- Один атрибут или группа атрибутов назначаются ключом ненормализованной таблицы, а затем повторяющиеся группы изымаются и помещаются в отдельные отношения вместе с копиями ключа исходной таблицы. Далее в новых отношениях устанавливаются свои первичные ключи.

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Client (<u>clientNo</u>, cName)

PropertyRentalOwner (<u>clientNo</u>, <u>propertyNo</u>, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

PropertyRentalOwner

clientNo	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
CR76	PG16	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	450	CO93	Tony Shaw
CR56	PG4	6 Lawrence St, Glasgow	1-Sep-11	10-Jun-12	350	CO40	Tina Murphy
CR56	PG36	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
CR56	PG16	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

0.8.7. Вторая нормальная форма (2НФ) (1)

- Вторая нормальная форма основана на понятии полной функциональной зависимости.
- **Вторая нормальная форма (2НФ).** Отношение, которое находится в первой нормальной форме и каждый атрибут которого, не входящий в состав первичного ключа, характеризуется полной функциональной зависимостью от этого первичного ключа
- Нормализация отношений 1НФ с приведением к форме 2НФ предусматривает устранение частичных зависимостей.
- Если в отношении между атрибутами существует частичная зависимость, то функционально-зависимые атрибуты удаляются из него и помещаются в новое отношение вместе с копией их детерминанта.
- Вторая нормальная форма применяется к отношениям с составными ключами, т. е. к таким отношениям, первичный ключ которых состоит из двух или нескольких атрибутов. Дело в том, что отношение с первичным ключом на основе единственного атрибута всегда находится, по крайней мере, в форме 2НФ.

Переход к второй нормальной форме (2НФ)

Функциональные зависимости в отношении ClientRental

- fd1 clientNo, propertyNo –> rentStart, rentFinish (первичный ключ)
- fd2 clientNo –> cName (частичная зависимость)
- fd3 propertyNo –> pAddress, rent, ownerNo, oName (частичная зависимость)
- fd4 ownerNo –> oName (транзитивная зависимость)
- fd5 clientNo, rentStart –> propertyNo, pAddress, rentFinish, rent, ownerNo, oName (потенциальный ключ)
- fd6 propertyNo, rentStart –> clientNo, cName, rentFinish (потенциальный ключ)
- После выявления функциональных зависимостей процесс нормализации отношения ClientRental предусматривает проверку его принадлежности ко второй нормальной форме. Для этого требуется найти *хотя бы один случай частичной зависимости* от первичного ключа. Нетрудно заметить, что атрибут имени клиента cName частично зависит от первичного ключа, иначе говоря, он зависит только от атрибута clientNo (эта зависимость представлена выше как fd2).

Переход к второй нормальной форме (2НФ) (продолжение)

- Кроме того, атрибуты объекта недвижимости (pAddress, rent, ownerNo, oName) также частично зависят от первичного ключа, но на этот раз только от атрибута propertyNo (эта зависимость представлена выше как fd3). В свою очередь, атрибуты арендованных объектов недвижимости (rentstart и rentFinish) полностью функционально зависят от первичного ключа в целом, т.е. от атрибутов clientNo и propertyNo (эта зависимость представлена выше как fdl).
- Обратите внимание на наличие *транзитивной зависимости* (transitive dependence) от первичного ключа (эта зависимость представлена выше как fd4). Хотя транзитивная зависимость также может послужить причиной аномалий обновления, тем не менее ее присутствие в отношении не нарушает ограничений для формы 2НФ. Такие зависимости будут устранены при переходе к форме 3НФ.
- Итак, обнаружены частичные зависимости внутри отношения ClientRental, а это означает, что данное отношение не находится во второй нормальной форме.

Переход к второй нормальной форме (2НФ) (продолжение)

- Для преобразования отношения ClientRental в форму 2НФ необходимо создать новые отношения, причем таким образом, чтобы атрибуты, не входящие в первичный ключ, были перемещены в них вместе с копией той части первичного ключа, с которой эти атрибуты связаны полной функциональной зависимостью. Применение такого процесса в нашем случае приведет к созданию трех новых отношений — Client, Rental и PropertyOwner.

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-12	31-Aug-13
CR76	PG16	1-Sep-13	1-Sep-14
CR56	PG4	1-Sep-11	10-Jun-12
CR56	PG36	10-Oct-12	1-Dec-13
CR56	PG16	1-Nov-14	10-Aug-15

PropertyOwner

propertyNo	pAddress	rent	ownerNo	oName
PG4	6 Lawrence St, Glasgow	350	CO40	Tina Murphy
PG16	5 Novar Dr, Glasgow	450	CO93	Tony Shaw
PG36	2 Manor Rd, Glasgow	375	CO93	Tony Shaw

Client (clientNo, cName)

Rental (clientNo, propertyNo, rentStart, rentFinish)

PropertyOwner (propertyNo, pAddress, rent, ownerNo, oName)

0.8.8. Третья нормальная форма (3НФ) (1)

- **Третья нормальная форма (3НФ).** Отношение, которое находится в первой и во второй нормальных формах и не имеет атрибутов, не входящих в первичный ключ, которые находились бы в транзитивной функциональной зависимости от этого первичного ключа.
- Нормализация отношений 2НФ с образованием отношений 3НФ предусматривает *устранение транзитивных зависимостей*. Если в отношении существует транзитивная зависимость между атрибутами, то транзитивно зависимые атрибуты удаляются из него и помещаются в новое отношение вместе с копией их детерминанта.
- Хотя отношения 2НФ в меньшей степени обладают избыточностью данных, чем отношения 1НФ, они все еще могут быть подвержены аномалиям обновления. Так, при попытке обновления в отношении PropertyOwner имени владельца недвижимости (например, Tony Shaw с номером C093 (атрибут ownerNo)) потребуется обновить две строки отношения PropertyOwner. Если обновить только одну из этих двух строк, база данных попадет в противоречивое состояние. Эта аномалия обновления вызывается транзитивной зависимостью, присутствующей в данном отношении. Она может быть устранена путем приведения данного отношения к третьей нормальной форме.

0.8.8. Третья нормальная форма (3НФ) (2)

Переход к третьей нормальной форме (3НФ)

- Нормализация отношений 2НФ с образованием отношений 3НФ предусматривает *устранение транзитивных зависимостей*. Если в отношении существует транзитивная зависимость между атрибутами, то транзитивно зависимые атрибуты удаляются из него и помещаются в новое отношение вместе с копией их детерминанта.
- Все не входящие в первичный ключ атрибуты отношений Client и Rental функционально зависимы только от их первичных ключей. Следовательно, отношения Client и Rental не имеют транзитивных зависимостей и поэтому они находятся в третьей нормальной форме (3НФ).

Переход к третьей нормальной форме (3НФ) (продолжение)

- Все не входящие в первичный ключ атрибуты отношения PropertyOwner функционально зависят от первичного ключа, за исключением атрибута oName, который зависит также и от атрибута ownerNo (зависимость fd4). Это типичный пример транзитивной зависимости, которая имеет место при наличии зависимости не входящего в первичный ключ атрибута (oName) от одного или нескольких других атрибутов, также не входящих в первичный ключ (ownerNo).
- fd3 propertyNo –> pAddress, rent, ownerNo, oNarne (Первичный ключ)
- fd4 ownerNo –> oName (Транзитивная зависимость)
- Для преобразования отношения PropertyOwner в третью нормальную форму необходимо прежде всего удалить упомянутую выше транзитивную зависимость путем создания двух новых отношений PropertyForRent и Owner.

PropertyOwner

propertyNo	pAddress	rent	ownerNo	oName
PG4	6 Lawrence St, Glasgow	350	CO40	Tina Murphy
PG16	5 Novar Dr, Glasgow	450	CO93	Tony Shaw
PG36	2 Manor Rd, Glasgow	375	CO93	Tony Shaw

0.8.8. Третья нормальная форма (3НФ) (4)

Переход к третьей нормальной форме (3НФ) (продолжение)

- PropertyForRent (propertyNo, pAddress, rent, ownerNo)
- Owner (ownerNo, oName)

PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

0.8.8. Третья нормальная форма (3НФ) (5)

Переход к третьей нормальной форме (3НФ) -- результат

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-12	31-Aug-13
CR76	PG16	1-Sep-13	1-Sep-14
CR56	PG4	1-Sep-11	10-Jun-12
CR56	PG36	10-Oct-12	1-Dec-13
CR56	PG16	1-Nov-14	10-Aug-15

PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

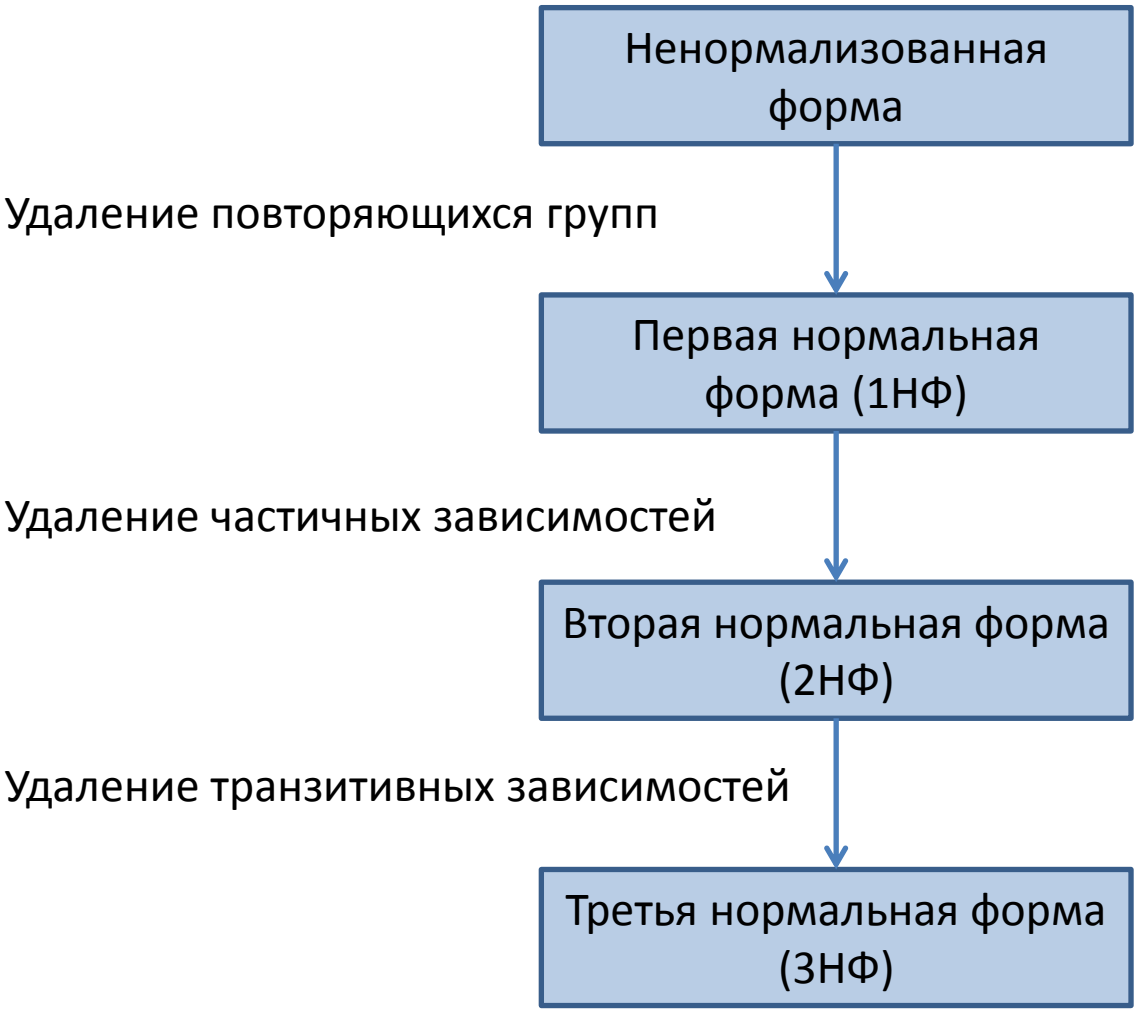
ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

Исходное отношение ClientRental может быть восстановлено путем соединения отношений Client, Rental, PropertyForRent и Owner. Данная цель достигается за счет использования первичных и внешних ключей. Данную процедуру иначе называют декомпозицией *без потерь (lossless).*

0.8.9. Общее определение второй и третьей нормальных форм

- Определения второй (2НФ) и третьей (3НФ) нормальных форм, приведенные выше, не допускают наличия частичных или транзитивных зависимостей от первичного ключа отношения. В общих определениях форм 2НФ и 3НФ учитываются потенциальные ключи отношения.
- **Вторая нормальная форма (2НФ)**. Отношение, находящееся в первой нормальной форме, в котором каждый атрибут, отличный от атрибута потенциального ключа, является полностью функционально зависимым от какого-либо потенциального ключа.
- **Третья нормальная форма (3НФ)**. Отношение, находящееся в первой и второй нормальной форме, в котором ни один атрибут, отличный от атрибута потенциального ключа, не является транзитивно зависимым ни от одного потенциального ключа.
- На практике чаще всего результаты декомпозиции являются одинаковыми, независимо от того, используются ли определения форм 2НФ и 3НФ, основанные на первичных ключах, или общие определения.

0.8.10. Общая схема нормализации



0.8.11. Нормальная форма Бойса–Кодда (НФБК) (1)

- Применение общих определений 2НФ и 3НФ может позволить выявить дополнительную избыточность, вызванную зависимостями от всех потенциальных ключей.
- Но даже после ввода этих дополнительных ограничений в отношениях все еще могут существовать зависимости, которые приводят к появлению избыточности в отношениях 3НФ.
- С учетом этого недостатка третьей нормальной формы была разработана более строгая нормальная форма, получившая название нормальной формы Бойса–Кодда (НФБК).
- Нормальная форма Бойса–Кодда (НФБК) основана на функциональных зависимостях, в которых учитываются все потенциальные ключи отношения. Кроме того, в НФБК предусмотрены более строгие ограничения по сравнению с общим определением 3НФ.

0.8.11. Нормальная форма Бойса–Кодда (НФБК) (2)

- **Нормальная форма Бойса–Кодда (НФБК).** Отношение находится в НФБК тогда и только тогда, когда каждый его детерминант является потенциальным ключом.
- Для проверки принадлежности отношения к НФБК необходимо найти все его детерминанты и убедиться в том, что они являются потенциальными ключами.
- Детерминантом является один атрибут или группа атрибутов, от которой полностью функционально зависит другой атрибут.

0.9. Методология проектирования баз данных

0.9.1. Введение (1)

Проектирование базы данных является одним из этапов жизненного цикла разработки системы с базами данных.

- Планирование БД
- Определение системы
- Сбор и анализ требований
- Проектирование БД
  - Концептуальное проектирование
  - Логическое проектирование
  - Физическое проектирование
- Прототипирование (и новая итерация проектирования БД)
- Проектирование приложений (параллельно с проектированием БД)
- Реализация
- Преобразование и загрузка данных
- Тестирование
- Функционирование

0.9.1. Введение (2)

- **Методология проектирования.** Структурированный подход, предусматривающий использование специализированных процедур, технических приемов, инструментов и документации, и предназначенный для поддержки и упрощение процесса проектирования.
- Методология проектирования предусматривает разбиение всего процесса на несколько этапов, каждый из которых, в свою очередь, состоит из нескольких стадий.
- На каждой стадии разработчику предлагается набор технических приемов, позволяющих решать задачи, стоящие перед ним на данной стадии разработки.
- Кроме того, методология предлагает методы планирования, координации, управления, оценки хода разработки проекта, а также структурированный подход к анализу и моделированию всего набора требований, предъявляемых к базе данных, и позволяет выполнить эти действия стандартизированным и организованным образом.

0.9.1. Введение (3)

- В предлагаемой методологии весь процесс проектирования базы данных подразделяется на три основных стадии:
  - концептуальное проектирование;
  - логическое проектирование;
  - физическое проектирование.
- **Концептуальное проектирование** – процесс конструирования модели данных, используемых на предприятии, независимой от всех деталей реализации.
- Концептуальная модель не зависит от
  - СУБД
  - прикладных программ
  - языков программирования
  - аппаратной платформы
  - вопросов производительности и др.
- Строится на основе требований пользователей.

0.9.1. Введение (4)

- **Логическое проектирование** – процесс конструирования модели данных, используемых на предприятии, на основе конкретной модели данных, но независимо от конкретной СУБД и других деталей физической реализации.
- Концептуальная модель отображается на логическую на основе выбора целевой модели данных, например, реляционной. Мы знаем тип СУБД (реляционная, сетевая, иерархическая, объектно-ориентированная), но не знаем деталей физической реализации хранения данных, индексов и др.
- **Физическое проектирование** – создание описания конкретной реализации базы данных, размещаемой во внешней памяти в среде целевой СУБД.
- Описываются базовые отношения (base relations), индексы, ограничения целостности.

- Проектирование БД – это итерационный процесс.

Концептуальное проектирование базы данных

Шаг 1. Построить концептуальную модель данных.

Шаг 1.1. Идентифицировать типы сущностей.

Шаг 1.2. Идентифицировать типы связей.

Шаг 1.3. Идентифицировать атрибуты и связать их с типами сущностей или связей.

Шаг 1.4. Определить домены атрибутов.

Шаг 1.5. Определить атрибуты потенциальных, первичных и альтернативных ключей.

Шаг 1.6. Рассмотреть необходимость использования расширенных концепций моделирования (необязательный шаг).

Шаг 1.7. Проверить модель на предмет избыточности.

Шаг 1.8. Проверить концептуальную модель относительно реализуемости пользовательских транзакций.

Шаг 1.9. Обсудить концептуальную модель с пользователями.

Логическое проектирование базы данных для реляционной модели

Шаг 2. Построить логическую модель данных.

Шаг 2.1. Создать отношения для логической модели данных.

Шаг 2.2. Проверить отношения с помощью правил нормализации.

Шаг 2.3. Проверить соответствие отношений требованиям пользовательских транзакций.

Шаг 2.4. Проверить ограничения целостности данных.

Шаг 2.5. Обсудить разработанную логическую модель данных с пользователями.

Шаг 2.6. Слияние локальных логических моделей данных в глобальную модель (необязательный шаг).

Шаг 2.7. Проверить возможность расширения модели в будущем.

Физическое проектирование базы данных для реляционной СУБД

Шаг 3. Перенести логическую модель в среду целевой СУБД.

Шаг 3.1. Спроектировать базовые отношения.

Шаг 3.2. Спроектировать представление производных (derived) данных.

Шаг 3.3. Спроектировать общие ограничения.

Шаг 4. Спроектировать организацию файлов и индексов.

Шаг 4.1. Проанализировать транзакции.

Шаг 4.2. Выбрать способы организации файлов.

Шаг 4.3. Выбрать индексы.

Шаг 4.4. Оценить потребность в дисковой памяти.

Шаг 5. Спроектировать представления пользователей.

Шаг 6. Спроектировать механизмы защиты.

Шаг 7. Рассмотреть необходимость введения контролируемой избыточности.

Шаг 8. Организация мониторинга и настройка функционирования работающей системы.

0.9.3. Концептуальное проектирование (1)

Шаг 1. Построить концептуальную модель данных.

- Цель: создать концептуальную модель на основе требований к данным предприятия.
- Концептуальная модель включает:
  - типы сущностей
  - типы связей
  - атрибуты и домены атрибутов
  - первичные и альтернативные ключи
  - ограничения целостности

Концептуальное моделирование поддерживается документацией, включая ER-диаграммы и словарь данных.

0.9.3. Концептуальное проектирование (2)

Шаг 1.1. Идентифицировать типы сущностей.

- На основе пользовательских требований
- Реальные объекты в предметной области

- Проблемы:
  - Пользователи используют примеры и частные случаи, а не обобщенные термины.
  - Вместо наименований должностей используют имена конкретных людей.
  - Трудно решить, считать ли конкретный объект сущностью, связью или атрибутом.
- После выделения каждой сущности ей следует присвоить определенное осмысленное имя, которое обязательно должно быть понятно пользователям. Выбранное имя и описание сущности помещается в словарь данных. Если это возможно, следует установить и внести в документацию данные об ожидаемом количестве экземпляров каждой сущности. Если сущность известна пользователям под разными именами, все дополнительные имена рекомендуется определить как синонимы или псевдонимы и также занести в словарь данных.

Шаг 1.1. Идентифицировать типы сущностей (продолжение).

- **Пример словаря данных с описаниями сущностей**

Имя сущности	Описание	Псевдонимы	Частота
Staff	Общее обозначение для всех сотрудников компании *DreamHome*	Employee	Каждый сотрудник компании работает в одном конкретном отделении
PropertyForRent	Общее обозначение для всех объектов недвижимости	Property	Каждый объект недвижимости имеет одного владельца и передается в аренду в определенном отделении компании, в котором этим объектам недвижимости управляет один сотрудник. Объект недвижимости, сдаваемый в аренду, осматривают многие клиенты, а один клиент берет в аренду на определенный период

Шаг 1.2. Идентифицировать типы связей.

- Искать глаголы в требованиях пользователя
  - Staff *Manages* PropertyForRent
    (Сотрудник компании *управляет* Объектом недвижимости)
  - PrivateQwner *Owns* PropertyForRent
    (Владелец объекта недвижимости *владеет* Объектом недвижимости)
  - PropertyForRent *AssociatedWith* Lease
    (Объект недвижимости *сдается в аренду по* Договору аренды)
- Связи:
  - бинарные
  - множественные
  - рекурсивные (сущность «Работник» имеет атрибут «Код начальника»)
- Нужно использовать ER-диаграммы.
- Определить множественность (кратность) связей (1:M, M:N, 1:1).
- Проверить отсутствие ловушек типа «разветвление» и типа «разрыв».
- Проверить соблюдение требования об участии каждой сущности, по меньшей мере, в одной связи.

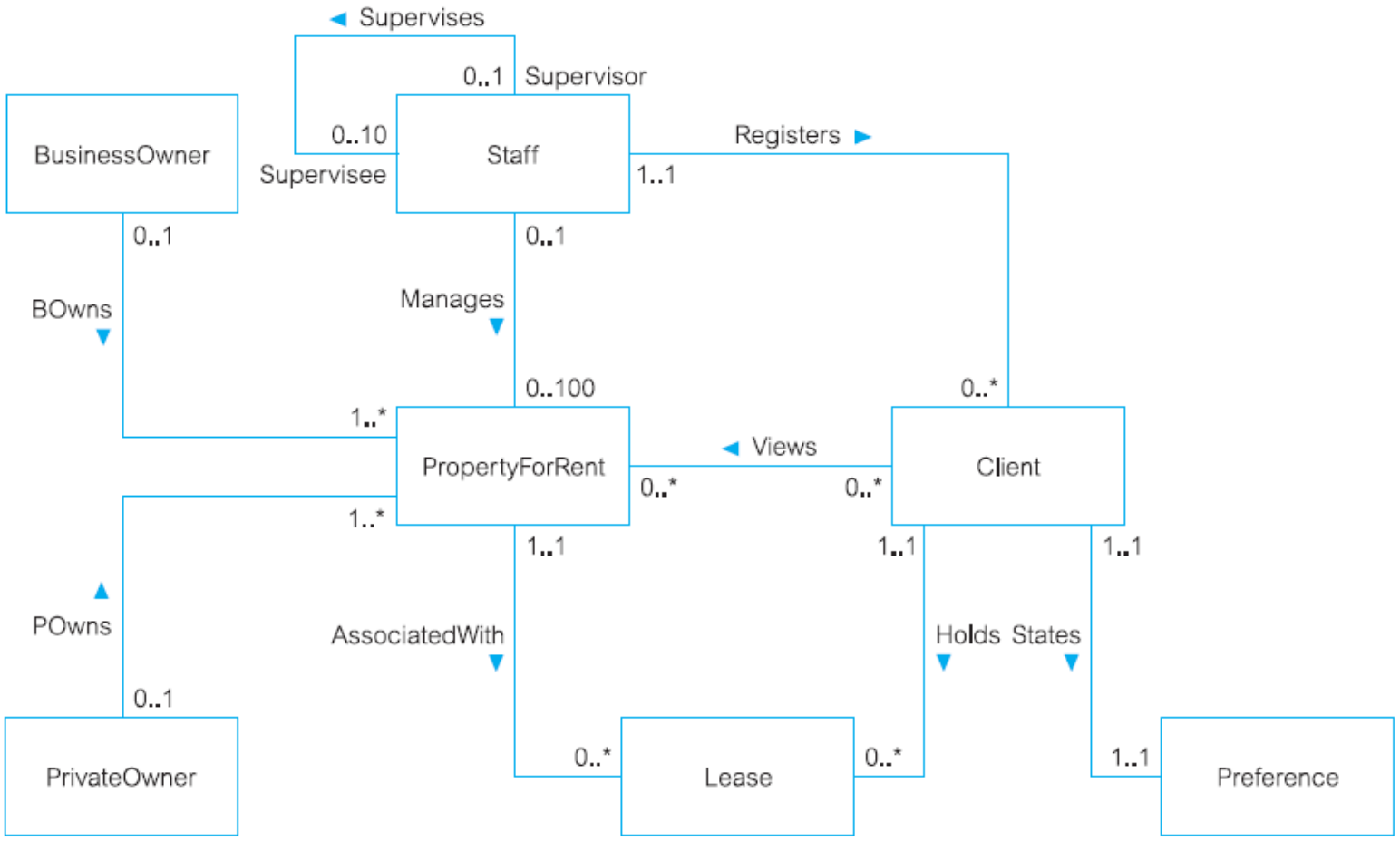
Шаг 1.2. Идентифицировать типы связей (продолжение).

- **Пример словаря данных с описаниями типов связей**

Имя сущности	Кратность	Наименование связи	Кратность	Имя сущности
Staff	0..1	*Manages*	0..100	PropertyForRent
	0..1	*Supervises*	0..10	Staff
PropertyForRent	1..1	*AssociatedWith*	0..*	Lease

Шаг 1.2. Идентифицировать типы связей (продолжение).

- Пример ER-диаграммы



Шаг 1.3. Идентифицировать атрибуты и связать их с сущностями или связями.

- Простые и составные атрибуты
  - Определяем на основе требований пользователя. Например: «Адрес» может быть простым атрибутом, содержащим все элементы адреса. Но можно раздробить этот атрибут на компоненты: «Город», «Улица», «Дом», «Квартира».
- Порожденные (производные) атрибуты
  - Их значения вычисляются на основе значений других атрибутов. Например: «Возраст» и «Дата рождения». Очень часто подобные атрибуты вообще не отображаются в концептуальной модели данных. Однако если производный атрибут показан в модели данных, следует непременно указать, что он является производным.

Шаг 1.3. Идентифицировать атрибуты и связать их с сущностями или связями (продолжение).

- Однозначные и многозначные атрибуты
  - Например: сущность «Работник», многозначный атрибут «Номер телефона».
- Для каждого атрибута нужно записать:
  - имя и описание
  - тип данных и длину
  - все псевдонимы, под которыми он известен
  - простой/составной (если он составной, то указать все входящие в него атрибуты)
  - является ли многозначным
  - является ли вычисляемым (и как это делается)
  - значение по умолчанию (если есть)

0.9.3. Концептуальное проектирование (9)

Шаг 1.3. Идентифицировать атрибуты и связать их с сущностями или связями (продолжение).

- **Пример словаря данных с описаниями атрибутов**

Имя сущности	Атрибуты	Описание	Тип данных и длина	Пустые значения	Многозначный	. . .
Staff	staffNo	Однозначно определяет сотрудника компании	Строка длиной до 5 символов	Нет	Нет	
	name fName lName	Имя сотрудника компании Фамилия сотрудника компании	Строка длиной до 15 символов Строка длиной до 15 символов	Нет Нет	Нет Нет	
	position	Наименование должности сотрудника компании	Строка длиной до 10 символов	Нет	Нет	

Шаг 1.4. Определить домены атрибутов.

- Домен – некоторое множество значений, элементы которого выбираются для присвоения значений одному или нескольким атрибутам.

ПРИМЕРЫ

- Домен атрибута, включающий допустимые значения табельных номеров (staffNo). Он состоит из строк переменной длины, которые могут включать до пяти символов; первые два символа должны быть буквенными, а следующие — состоять из цифр от 1 до 3, представляющих собой числа от 1 до 999.
- Возможные значения атрибута sex сущности Staff, которые могут быть представлены как «M» или «F». Домен этого атрибута включает две односимвольные строки со значением «M» или «F».

Шаг 1.4. Определить домены атрибутов (продолжение).

- Домены должны содержать следующие данные:
  - набор допустимых значений для атрибута;
  - сведения о размере и формате каждого из атрибутов.
- В доменах может быть указана и другая дополнительная информация, например сведения о допустимых операциях со значениями атрибутов, а также данные о том, какие атрибуты можно использовать для сравнения с другими атрибутами или при построении комбинаций из нескольких атрибутов.
- После определения доменов атрибутов их имена и характеристики помещаются в словарь данных. Одновременно обновляются записи словаря данных, относящиеся к атрибутам: в них заносятся имена назначенных каждому атрибуту доменов вместо обозначения типов данных и информации о размерности.

Шаг 1.5. Определить атрибуты потенциальных, первичных и альтернативных ключей.

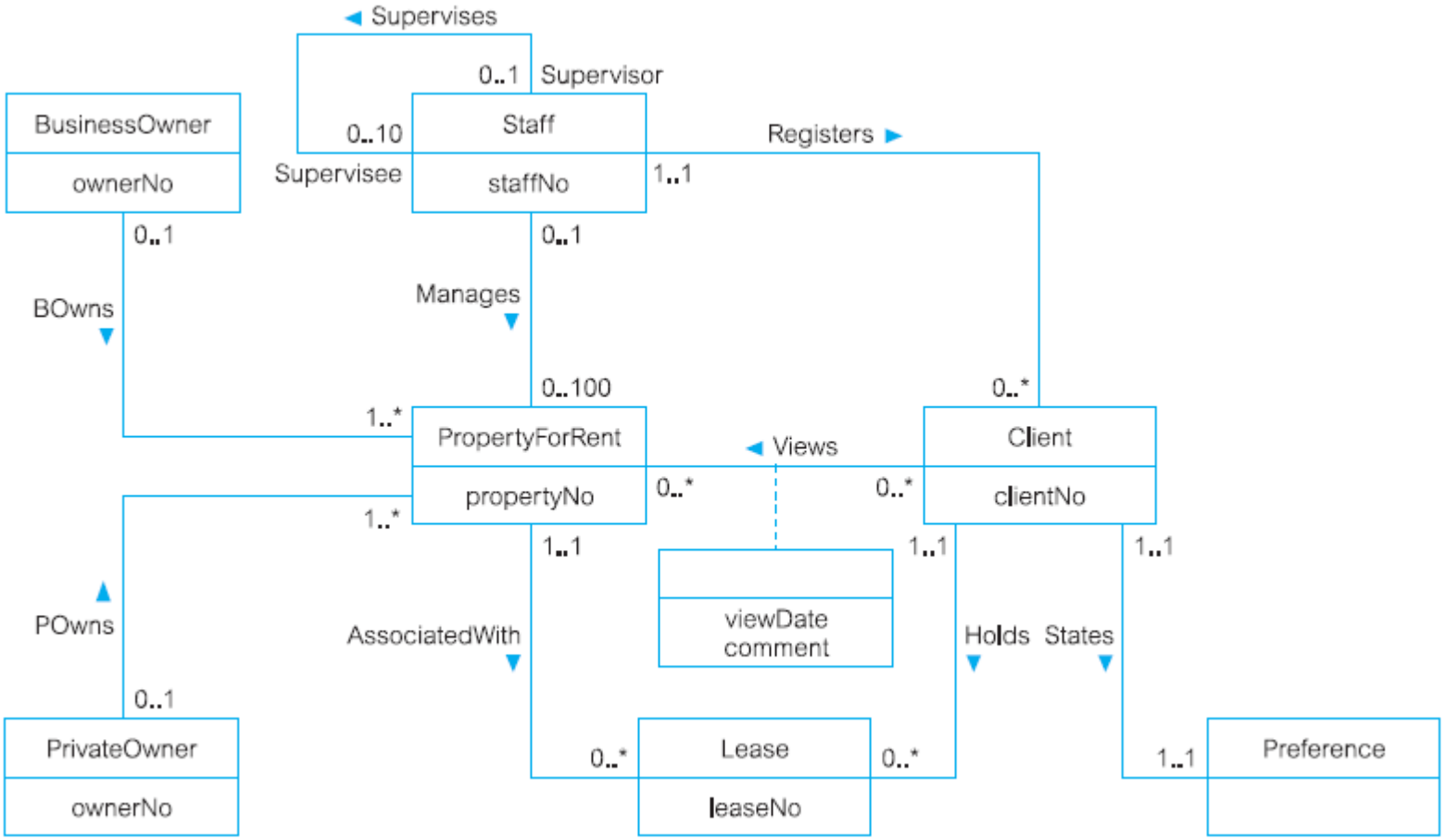
- *Потенциальным ключом* называется атрибут или минимальный набор атрибутов заданной сущности, позволяющий однозначно идентифицировать каждый ее экземпляр.
- Для некоторых сущностей возможно наличие нескольких потенциальных ключей. В этом случае среди них нужно выбрать один ключ, который будет называться *первичным ключом.* Все остальные потенциальные ключи будут называться *альтернативными ключами.*
- Рекомендации по выбору ключей:
  - минимальное число атрибутов;
  - мала вероятность изменения значений атрибутов ключа;
  - малая длина текстовых значений, входящих в состав ключа;
  - малое наибольшее значение (для числовых атрибутов ключа);
  - ключ-кандидат, наиболее удобный с точки зрения пользователя.

Шаг 1.5. Определить атрибуты потенциальных, первичных и альтернативных ключей (продолжение).

- В процессе определения первичного ключа устанавливается, является ли данная сущность сильной или слабой.
- Если выбрать первичный ключ для данной сущности оказалось возможным, то такую сущность принято называть *сильной.* И наоборот, если выбрать первичный ключ для заданной сущности невозможно, то ее называют *слабой.*
- Определение первичных ключей для слабых сущностей может быть выполнено только на стадии логического проектирования.
- После выбора первичных и альтернативных ключей сущностей (если таковые определены) сведения о них необходимо поместить в словарь данных.

Шаг 1.5. Определить атрибуты потенциальных, первичных и альтернативных ключей (продолжение).

- ER-диаграмма после добавления первичных ключей



Шаг 1.6. Рассмотреть необходимость использования расширенных концепций моделирования (необязательный шаг).

- **Цель.** Рассмотреть необходимость использования таких расширенных понятий моделирования, как уточнение/обобщение, агрегирование и композиция.
- Подход, требующий обобщения, предусматривает необходимость выявить общие особенности разных сущностей для определения обобщающей сущности суперкласса. Агрегирование может применяться для обозначения связи «has-a» (включает) или «is-part-of» (входит в состав) между типами сущностей; в такой связи одна сущность представляет «целое», а другая — ее «часть». Композиция (особый тип агрегирования) может применяться для определения взаимосвязи между типами сущностей, которая обусловливает строгую принадлежность и совпадение срока существования между «целым» и «частью».

Шаг 1.7. Проверить модель на предмет избыточности.

- Проверить связи типа 1:1.
- Возможно, что в процессе определения сущностей были обнаружены две сущности, которые соответствуют в данной организации одному и тому же концептуальному объекту. Например, допустим, что обнаружены две сущности (Client и Renter), которые фактически являются одинаковыми; иными словами, Client — это синоним для Renter. В таком случае эти две сущности должны быть объединены.
- Удалить избыточные связи.
- Связь является избыточной, если представленная в ней информация может быть получена с помощью других связей. Разработчик стремится создать минимальную модель данных, а поскольку избыточные связи не нужны, они должны быть удалены. На ER-диаграмме наличие избыточных связей можно относительно легко обнаружить, поскольку оно проявляется в том, что между двумя сущностями имеется несколько путей. Но такая ситуация не всегда означает, что одна из связей является избыточной, так как они могут представлять разные ассоциации между сущностями.

Шаг 1.8. Проверить концептуальную модель относительно реализуемости пользовательских транзакций.

- Для этого должна быть предпринята попытка выполнить все необходимые операции вручную с помощью данной модели.
- Если все транзакции удалось выполнить таким образом, то проверка соответствия концептуальной модели данных требуемым транзакциям считается успешной.
- Но если невозможно провести вручную все транзакции, это означает, что модель данных содержит дефекты, которые должны быть устранены. В таком случае, вероятно, в модели данных не учтены какие-либо сущности, связи или атрибуты.
- Существует два возможных способа проверки того, что локальная концептуальная модель данных поддерживает требуемые транзакции.

1. Описание транзакций.
2. Проверка с применением путей выполнения транзакций.

Шаг 1.9. Обсудить концептуальную модель с пользователями.

- **Цель.** Обсуждение локальных концептуальных моделей данных с конечными пользователями с целью подтверждения того, что данная модель полностью соответствует спецификации требований пользовательского представления.

0.9.4. Логическое проектирование (1)

Шаг 2. Построить логическую модель данных.

- **Цель.** Преобразовать концептуальную модель данных в логическую модель данных и затем проверить ее на предмет структурной корректности и способности поддерживать требуемые транзакции.
- **Логическое проектирование баз данных.** Процесс конструирования общей информационной модели на основе моделей данных отдельных пользователей. Эта модель является независимой от особенностей реально используемой СУБД и других физических условий.
- Если при проектировании базы данных используется всего одно представление пользователя или несколько представлений, но с применением централизованного подхода для управления ими, тогда шаг 2.6 опускается.
- Если же используется несколько различных представлений пользователей с применением подхода интеграции представлений, тогда шаги 2.1 – 2.5 повторяются для требуемого числа моделей данных, каждая из которых соответствует представлениям различных пользователей о базе данных. А на шаге 2.6 эти модели данных объединяются.

0.9.4. Логическое проектирование (2)

Шаг 2.1. Создать отношения для логической модели данных.

- **Сильные типы сущностей**
  - Для каждой из них создать отношение, включающее все простые атрибуты. Составные атрибуты разбить на простые.
- **Слабые типы сущностей**
  - Для каждой из них создать отношение, включающее все простые атрибуты. Первичный ключ слабой сущности полностью или частично определяется на основе сильной сущности-владельца.

Шаг 2.1. Создать отношения для логической модели данных (продолжение).

- **Бинарные связи типа 1:M (родительская и дочерняя сущности)**
  - Сущность, находящаяся на стороне «1» считается родительской, а сущность на стороне «M» – дочерней.
  - Копия атрибутов первичного ключа родительской сущности помещается в отношение, реализующее дочернюю сущность, выполняя роль внешнего ключа.
  - Если связь также имеет атрибуты, то их тоже нужно поместить в дочернее отношение.
  - Например. Для реализации связи **Студент** ***Сдает*** **Экзамен** атрибут «Номер зачетной книжки» первичного ключа отношения **Студент** помещается в отношение **Экзамен** , где он выполняет роль внешнего ключа.

Шаг 2.1. Создать отношения для логической модели данных (продолжение).

- **Бинарные связи типа 1:1**

  В этом случае кардинальность нельзя использовать для определения родительской и дочерней сущностей. Для определения этого используется ограничение «участие», или «членство».

  Обязательное членство на обеих сторонах связи

  Обе сущности нужно объединить в одно отношение. В качестве первичного ключа выбрать любой из первичных ключей. Второй из них станет альтернативным ключом.

  Обязательное членство на одной стороне связи

  Необязательное членство на обеих сторонах связи

Шаг 2.1. Создать отношения для логической модели данных (продолжение).

- **Бинарные связи типа M:N**
  - Создать отношение, представляющее связь, и включить в него все атрибуты, описывающие эту связь.
  - Поместить атрибуты, являющиеся первичными ключами сущностей, участвующих в связи, в новое отношение. Они будут внешними ключами.
  - Один или оба этих внешних ключа будут образовывать первичный ключ нового отношения, возможно, с добавлением атрибутов, описывающих саму связь.
  - Пример. Сущности Студент и Дисциплина, связь «Сдает дисциплину». Первичным ключом нового отношения будут атрибуты «Номер зачетной книжки», «Код дисциплины» и, возможно, «Номер семестра».

0.9.4. Логическое проектирование (6)

Шаг 2.1. Создать отношения для логической модели данных (продолжение).

- **Многозначные атрибуты**
  - Для каждого многозначного атрибута, имеющегося в какой-то сущности, создать новое отношение, представляющее этот атрибут. Включить первичный ключ этой сущности в новое отношение, он будет служить в качестве внешнего ключа.
- ПРИМЕР. В представлении Branch в соответствии с той ситуацией, что каждое отделение может иметь до трех номеров телефонов, атрибут telNo сущности Branch был определен как многозначный.

Branch (branchNo, street, city, postcode)
Primary Key branchNo

Telephone (telNo, branchNo)
Primary Key telNo
Foreign Key branchNo **references** Branch(branchNo)

0.9.4. Логическое проектирование (7)

Шаг 2.2. Проверить отношения с помощью правил нормализации.

- Идентифицировать функциональные зависимости между атрибутами (чтобы это сделать, нужно понять суть каждого атрибута)
- Для избежания проблем, связанных с избыточностью данных, требуется привести каждое отношение хотя бы к 3NF. Заметим, что процесс формирования отношений из концептуальной модели данных должен порождать отношения, которые сразу находятся в 3НФ.

Шаг 2.3. Проверить соответствие отношений требованиям пользовательских транзакций.

- Это делается вручную, используя отношения, первичные и внешние ключи, ER-диаграммы

Шаг 2.4. Проверить ограничения целостности данных.

- Ограничения целостности служат для того, чтобы данные не стали неполными, неточными или несогласованными.
- Здесь идет речь о том, какие ограничения целостности нужны, а не о том, как этого достичь.

- **Ограничения целостности бывают следующие:**
- Обязательное наличие допустимых значений
  - Не допускаются значения NULL. Например: каждый работник должен иметь конкретную должность.
- Ограничения домена
  - Каждый атрибут имеет множество допустимых значений. Например: пол может иметь два значения – мужской и женский.
- Множественность связей
  - Например: в каждом отделе предприятия могут работать много работников, но каждый работник работает только в одном отделе.
- Целостность сущностей
  - Первичные ключи не могут содержать значений NULL.

Шаг 2.4. Проверить ограничения целостности данных (продолжение).

- **Ограничения целостности бывают следующие (продолжение) :**
- Ссылочная целостность
  - Внешний ключ связывает каждую запись в подчиненном отношении с родительской записью в родительском отношении, имеющей соответствующее значение первичного или уникального ключа.
  - Например: отношения «Студенты» и «Успеваемость» связаны по атрибуту «Номер зачетной книжки». Тогда номер зачетной книжки в записях, относящихся к данному студенту и находящихся в отношении «Успеваемость», должен совпадать с номером зачетной книжки в записи об этом студенте, находящейся в отношении «Студенты»

Шаг 2.4. Проверить ограничения целостности данных (продолжение).

- **Ограничения целостности бывают следующие (продолжение) :**
- Ссылочная целостность (продолжение)

  Проблемы:

  1. Допускаются ли значения NULL в атрибутах внешнего ключа? Если класс членства подчиненной сущности обязательный, тогда не допускаются. Если класс членства необязательный, то допускаются.

  2. Как обеспечить ссылочную целостность? Для связи вида 1:M могут иметь место следующие случаи:
  - вставка записи в дочернее отношение
  - удаление записи из дочернего отношения
  - обновление внешнего ключа в дочернем отношении
  - вставка записи в родительское отношение
  - удаление записи из родительского отношения (стратегии: NO ACTION, CASCADE, SET NULL, SET DEFAULT)
  - обновление первичного ключа в родительском отношении (те же стратегии)

Шаг 2.4. Проверить ограничения целостности данных (продолжение).

- **Ограничения целостности бывают следующие (продолжение) :**
- Общие ограничения (бизнес-правила)
  - Например: число подчиненных у руководителя не может превышать некоторого предела.

Шаг 2.5. Обсудить разработанную логическую модель данных с пользователями.

- Если пользователи не считают полученную модель соответствующей тем требованиям, которые были предъявлены к данным предприятия, тогда необходимо повторить некоторые предшествующие шаги

Шаг 2.6. Слияние локальных логических моделей данных в глобальную модель (необязательный шаг).

- Не выполняется, если имеется только одно представление пользователя о данных предприятия или для построения глобальной логической модели был выбран централизованный подход, при котором требования отдельных пользователей сначала объединяются в единый документ, а затем выполняется построение модели.
- Выполняется, если был выбран подход интеграции представлений пользователей о базе данных, когда логические модели отдельных пользователей объединяются в единую логическую модель.

Шаг 2.7. Проверить возможность расширения модели в будущем.

- Проектировать нужно по возможности так, чтобы логическая модель была расширяемой и могла дополняться с учетом возникновения новых требований бизнеса.

0.9.5. Физическое проектирование реляционной базы данных (1)

Результат стадии логического проектирования:

- Логическая модель данных, включающая ER-диаграммы (диаграммы отношений), реляционную схему и документацию, в т. ч. словарь данных.
- На стадии логического проектирования отвечают на вопрос ЧТО должно быть сделано.
- На стадии физического проектирования отвечают на вопрос КАК это должно быть сделано.
- **Физическое проектирование базы данных.** Процесс подготовки описания реализации базы данных во внешней памяти; описание должно включать основные отношения, файловую организацию, индексы, обеспечивающие эффективный доступ к данным, а также все соответствующие ограничения целостности и средства защиты.

Шаг 3. Перенести логическую модель в среду целевой СУБД.

- **Цель.** Создание базовой функциональной схемы реляционной базы данных на основе глобальной логической модели данных, которая может быть реализована в целевой СУБД.

Шаг 3.1. Спроектировать базовые отношения.
Шаг 3.2. Спроектировать представление производных (derived) данных.
Шаг 3.3. Спроектировать общие ограничения.

Шаг 4. Спроектировать организацию файлов и индексов.

- **Цель.** Определение оптимальной файловой организации для хранения базовых отношений, а также индексов, необходимых для достижения приемлемой производительности; иными словами, определение способа хранения отношений и строк во внешней памяти.

Шаг 4.1. Проанализировать транзакции.
Шаг 4.2. Выбрать способы организации файлов.
Шаг 4.3. Выбрать индексы.
Шаг 4.4. Оценить потребность в дисковой памяти.

Шаг 5. Спроектировать представления пользователей.

- **Цель.** Спроектировать пользовательские представления, необходимость в создании которых была выявлена на стадии сбора и анализа требований, составляющей часть жизненного цикла системы с базой данных.
- В многопользовательской СУБД такие представления играют главную роль при определении структуры базы данных и соблюдении правил защиты. Основные преимущества пользовательских представлений:
  - независимость от данных
  - упрощение структуры приложения
  - учет потребностей пользователей.

0.9.5. Физическое проектирование (5)

Шаг 6. Спроектировать механизмы защиты.

- **Цель.** Проектирование средств защиты для базы данных в соответствии с требованиями пользователей.
- Назначение этого этапа состоит в определении способов реализации требований к защите. Состав средств защиты зависит от конкретной системы. Поэтому проектировщик должен знать, какие возможности предлагает рассматриваемая им целевая СУБД. Реляционные СУБД, как правило, предоставляют средства защиты базы данных следующих типов:
  - защита системы;
  - защита данных.

Шаг 6. Спроектировать механизмы защиты (продолжение).

- Средства защиты системы регламентируют доступ и эксплуатацию базы данных на уровне системы; к ним, в частности, относится аутентификация пользователя по имени и паролю.
- Средства защиты данных регламентируют доступ и использование объектов базы данных (таких как отношения и представления), а также действия, которые могут быть выполнены пользователями с конкретными объектами.
- Правила доступа можно создавать с помощью команд GRANT и REVOKE.

Шаг 7. Рассмотреть необходимость введения контролируемой избыточности.

- **Цель.** Определение необходимости ввода контролируемой избыточности за счет ослабления условий нормализации для повышения производительности системы.
- Результатом нормализации является логическая модель базы данных — структурно цельная система с минимальной избыточностью.
- Но в некоторых случаях оказывается, что нормализованная модель не обеспечивает максимальной производительности при обработке данных.
- Следовательно, при некоторых обстоятельствах может оказаться необходимым ради повышения производительности пожертвовать частью той выгоды, которую обеспечивает модель с полной нормализацией.

Шаг 8. Организация мониторинга и настройка функционирования работающей системы.

- **Цель.** Обеспечить текущий контроль функционирования работающей системы и добиться повышения ее производительности для устранения последствий принятия неоптимальных проектных решений или учета изменившихся требований.
- Одной из целей физического проектирования является обеспечение эффективного хранения данных и получения доступа к ним.
- Для оценки эффективности можно использовать следующие показатели:

1. Число транзакций, обрабатываемых в единицу времени.
2. Время отклика системы.
3. Объем дисковой памяти для хранения базы данных.

Шаг 8. Организация мониторинга и настройка функционирования работающей системы (продолжение).

- Настройка работающей системы может принести целый ряд выгод:

1. Настройка системы позволяет устранить потребность в дополнительном оборудовании.
2. Настройка системы дает возможность снизить требования к аппаратному обеспечению и, как результат, снизить расходы на обслуживание базы данных.
3. Точная настройка системы позволяет сократить время ответа на запрос и повысить производительность базы данных, что, в свою очередь, повышает эффективность работы как отдельных сотрудников (пользователей базы данных), так и всей организации в целом.
4. Сокращение времени ответа на запрос улучшает психологическое состояние персонала.
5. Чем короче время ответа базы данных на запрос, тем более удовлетворенным чувствует себя клиент фирмы.

0.10. Учебная предметная область

0.10.1. Описание предметной области (1)

- В качестве предметной области выберем пассажирские авиаперевозки. Ее оригинальное описание и описание базы данных «Авиаперевозки» можно найти по адресам https://postgrespro.ru/education/demodb и https://postgrespro.ru/docs/postgrespro/current/demodb-bookings.html.
- Компания имеет парк самолетов. Компоновки салонов у самолетов одной модели одинаковые.
- Каждый аэропорт имеет трехбуквенный код, название аэропорта не всегда совпадает с названием города.
- Формируются маршруты перелетов между городами. Конечно, каждый такой маршрут требует указания не только города, но и аэропорта, поскольку в городе может быть и более одного аэропорта
- На основе перечня маршрутов формируется расписание полетов (или рейсов). В расписании указывается плановое время отправления и плановое время прибытия, а также тип самолета, выполняющего этот рейс.

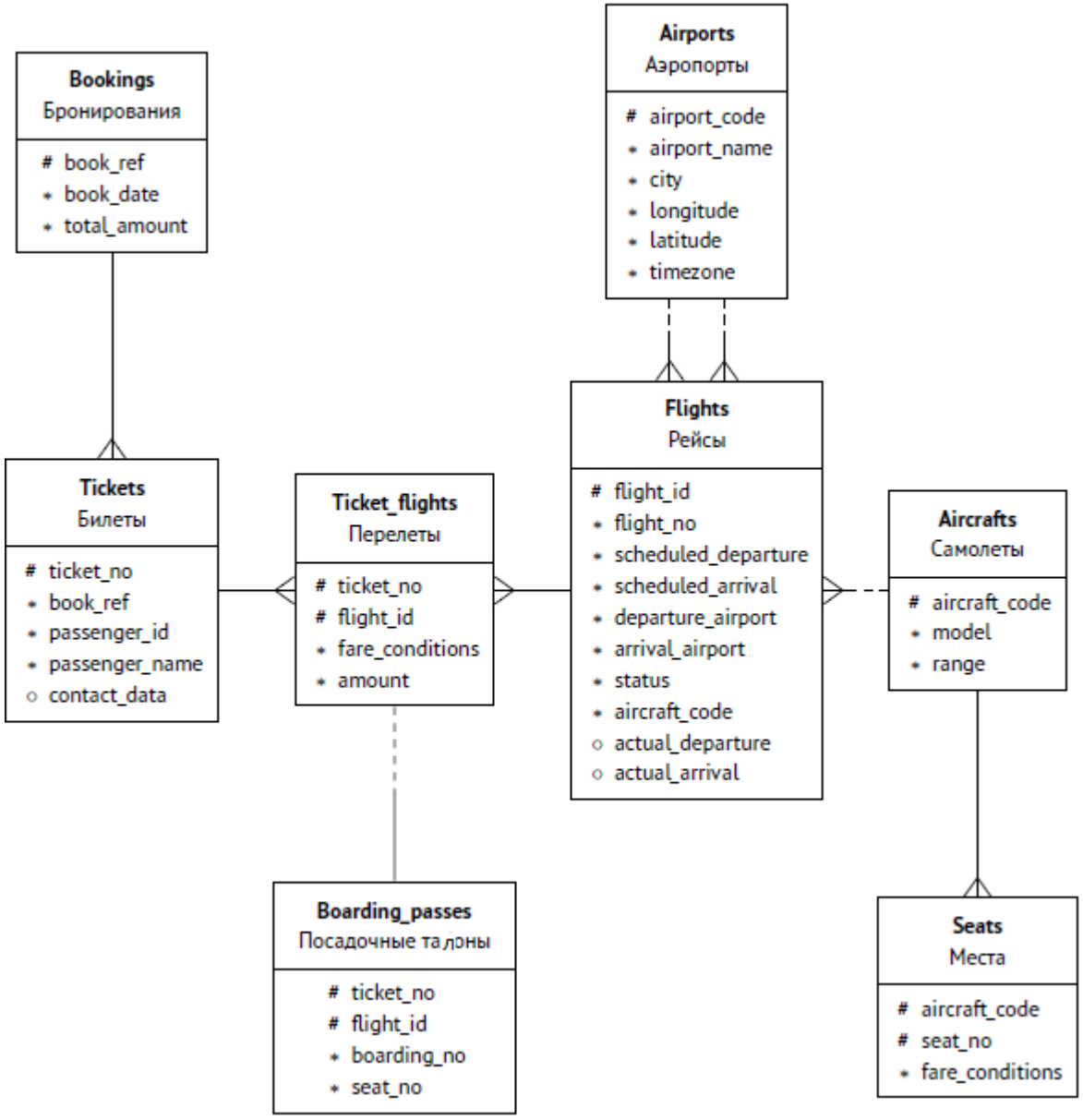
0.10.1. Описание предметной области (2)

- При фактическом выполнении рейса возникает необходимость в учете дополнительных сведений, а именно: фактического времени отправления и фактического времени прибытия, а также статуса рейса: Scheduled (можно бронировать), On Time (идет регистрация), Delayed (задержан), Departed (вылетел), Arrived (прибыл), Cancelled (отменен).
- Полет начинается с бронирования электронного авиабилета. Каждый такой билет имеет уникальный номер, состоящий из 13 цифр. В рамках одной процедуры бронирования может быть оформлено несколько билетов на различных пассажиров, но каждая такая процедура имеет уникальный шестизначный номер (шифр) бронирования, состоящий из заглавных букв латинского алфавита и цифр. Для каждой процедуры бронирования записывается дата бронирования и рассчитывается общая стоимость оформленных билетов.
- В каждый билет, кроме его тринадцатизначного номера, записывается идентификатор пассажира, а также его имя и фамилия (в латинской транскрипции) и контактные данные. В качестве идентификатора пассажира используется номер документа, удостоверяющего личность.

0.10.1. Описание предметной области (3)

- В каждый электронный билет может быть вписано более одного перелета, например: Красноярск — Москва, Москва — Анапа, Анапа — Москва, Москва — Красноярск. Для каждого перелета указывается номер рейса, аэропорты отправления и назначения, время вылета и время прибытия, а также стоимость перелета. Кроме того, указывается и так называемый класс обслуживания: экономический, бизнес и др.
- Когда пассажир прибывает в аэропорт отправления и проходит регистрацию билета, оформляется так называемый посадочный талон. Этот талон связан с авиабилетом: в талоне указывается такой же номер, который имеет электронный авиабилет данного пассажира. Кроме того, в талоне указывается номер рейса и номер места в самолете. Указывается также и номер посадочного талона — последовательный номер, присваиваемый в процессе регистрации билетов на данный рейс.

0.10.2. Схема базы данных



0.10.3. Развертывание учебной базы данных (1/7)

- Войдите в систему как пользователь postgres:

```
su – postgres
```

- Скачайте с сайта компании «Постгрес Профессионапьный» учебную базу данных «Авиаперевозки»
  https://postgrespro.ru/education/demodb
  ВАЖНО! Нужно использовать версию БД от 13.10.2016.
- Должен быть запущен сервер баз данных PostgreSQL:

```
pg_ctl  start  -D /usr/local/pgsql/data
```

- Для проверки запуска сервера выполните команду

```
pg_ctl  status  -D /usr/local/pgsql/data
```

или

```
ps -ax | grep postgres | grep -v grep
```

0.10.3. Развертывание учебной базы данных (2)

- Извлеките БД из архива

```
unzip demo-small-20161013.zip
```

- Создайте БД

```
psql -f demo_small.sql -U postgres
```
(для ОС Debian)

```
psql -f demo_small.sql
```
(для ОС Xubuntu)

- Запустите утилиту psql и подключитесь к этой базе данных

```
psql -d demo -U postgres
```
(для ОС Debian)

```
psql -d demo
```
(для ОС Xubuntu)

- Укажите текущую схему bookings в базе данных (в ней находятся все данные)

```
demo=# set search_path = bookings;
```

Литература

1. Гарсиа-Молина, Г. Системы баз данных. Полный курс : пер. с англ. / Гектор Гарсиа-Молина, Джеффри Ульман, Дженнифер Уидом. – М. : Вильямс, 2003. – 1088 с.
2. Грофф, Дж. SQL. Полное руководство : пер. с англ. / Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель. – 3-е изд. – М. : Вильямс, 2015. – 960 с.
3. Дейт, К. Дж. Введение в системы баз данных : пер. с англ. / Крис Дж. Дейт. – 8-е изд. – М. : Вильямс, 2005. – 1328 с.
4. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика : пер. с англ. / Томас Коннолли, Каролин Бегг. – 3-е изд. – М. : Вильямс, 2003. – 1436 с.
5. Кузнецов, С. Д. Основы баз данных : учеб. пособие / С. Д. Кузнецов. – 2-е изд., испр. – М. : Интернет-Университет Информационных Технологий ; БИНОМ. Лаборатория знаний, 2007. – 484 с.
6. Лузанов, П. PostgreSQL для начинающих / П. Лузанов, Е. Рогов, И. Лёвшин ; Postgres Professional. – М., 2017. – 146 с.
7. Моргунов, Е. П. Язык SQL. Базовый курс : учеб.-практ. пособие. / Е. П. Моргунов ; под ред. Е. В. Рогова, П. В. Лузанова ; Postgres Professional. – М., 2017. – 257 с.
8. PostgreSQL [Электронный ресурс] : официальный сайт / The PostgreSQL Global Development Group. – http://www.postgresql.org.
9. Postgres Professional [Электронный ресурс] : российский производитель СУБД Postgres Pro : официальный сайт / Postgres Professional. – http://postgrespro.ru.

Полезные ссылки

- Bruce Momjian
  http://momjian.us/main/presentations/sql.html
- Software Ideas Modeler https://www.softwareideas.net/
- William Kent. A Simple Guide to Five Normal Forms in Relational Database Theory // Communications of the ACM. – 1983. – 26(2), Feb. – P. 120–125. http://www.bkent.net/Doc/simple5.htm
- Моргунов, Е. П. Технологии разработки программ в среде операционных систем Linux и FreeBSD. Вводный курс [Текст] : учеб. пособие / Е. П. Моргунов, О. Н. Моргунова. – Красноярск, 2018. – 207 с. http://www.morgunov.org/programming.html

Задание

Для выполнения практических заданий необходимо использовать книгу:

Моргунов, Е. П. Язык SQL. Базовый курс : учеб.-практ. пособие / Под ред. Е. В. Рогова, П. В. Лузанова ; Postgres Professional. – М., 2017. – 257 с. https://postgrespro.ru/education/books/sqlprimer

1. Прочитать введение и главу 1.
2. Установить ОС Linux (Debian или другой). Указания по установке СУБД PostgreSQL приведены в главе 2, параграф 2.1. Можно воспользоваться виртуальной машиной VirtualBox или аналогичной. Можно использовать уже настроенную ОС Debian или Xubuntu (в виде виртуальной машины), полученную у преподавателя https://cloud.sibsau.ru/edu/morgunov/.
3. Развернуть учебную базу данных «Авиаперевозки» https://postgrespro.ru/education/demodb (см. главу 2, параграф 2.3). Использовать версию БД от 13.10.2016.