

## ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ В ТЕОРИЮ БАЗ ДАННЫХ.....	5
1.1 Информация и данные. Основные определения.....	5
1.2. Актуальные модели данных.....	6
1.3. Типы баз данных, компоненты баз данных.....	8
1.4. Жизненный цикл и ресурсы БД.....	11
1.5. История развития технологий хранения данных.....	12
Тестовые задания для самопроверки. ....	13
2. КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ ДАННЫХ.....	15
2.1. Сущности, атрибуты и экземпляры сущностей. ....	15
2.2. Связи и наборы связей.....	16
2.3. Дополнительные элементы концептуальной модели данных.....	17
2.4. Некоторые вопросы концептуального проектирования. ....	20
2.5. Концептуальный словарь данных. ....	23
Вопросы для самостоятельного изучения по итогам лекции. ....	24
Тестовые задания для самопроверки. ....	24
3. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ И ВВЕДЕНИЕ В SQL.....	27
3.1. Определение реляционной модели данных. ....	27
3.2. Язык запросов SQL и реализация в нем функционала реляционной модели данных.....	28
3.3. Реляционные ограничения целостности.....	31
3.4. Введение в пользовательские представления. ....	35
Вопросы для самостоятельного изучения по итогам лекции. ....	36
Тестовые задания для самопроверки. ....	36
4. ОСНОВЫ РЕЛЯЦИОННОЙ АЛГЕБРЫ. ТРИГГЕРЫ И ПРОГРАММИРОВАНИЕ SQL.....	39
4.1. Принципы работы SQL инструкции (запроса).....	39
4.2. Операции реляционной алгебры. ....	41
4.3. Агрегатные операторы, группировка, триггеры. ....	47

Вопросы для самостоятельного изучения по итогам лекции. ....	50
Тестовые задания для самопроверки. ....	50
5. РАЗРАБОТКА ПРИЛОЖЕНИЙ БАЗ ДАННЫХ.....	54
5.1. Универсальный API доступа к данным ODBC.....	54
5.2. Встроенный SQL в приложениях баз данных.....	56
5.3. Использование курсоров в встроенном SQL. ....	59
Вопросы для самостоятельного изучения по итогам лекции. ....	62
Тестовые задания для самопроверки. ....	63
6. РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ БАЗ ДАННЫХ.....	65
6.1. Идентификация web-ресурсов и протокол взаимодействия с web-ресурсами http.....	65
6.2. Форматы веб-данных. ....	68
6.3. Трехуровневая архитектура веб-приложений. Клиентская программа (фронт-энд).....	73
Вопросы для самостоятельного изучения по итогам лекции.....	77
7. ПОНЯТИЕ BIG DATA. noSQL СРЕДСТВА ОБРАБОТКИ И ХРАНЕНИЯ ДАННЫХ. ....	78
7.1. Определение больших данных (BigData) и их свойств. История возникновения термина BigData.....	78
7.2. Функция MapReduce, как инструмент работы с BigData.....	80
7.3. noSQL модели хранения данных. ....	83
Тестовые задания для самопроверки. ....	86
Ответы на тестовые задания. ....	89
Список литературы .....	90
Сведения об авторе .....	90

# 1. ВВЕДЕНИЕ В ТЕОРИЮ БАЗ ДАННЫХ.

## 1.1 Информация и данные. Основные определения.

**Информация** – любые сведения о явлении, процессе.

**Данные** – информация, представленная в определенной форме для обработки ее человеком или компьютером. Информация хранится в виде данных и извлекается из данных, чаще всего аналитическим или логическим путем.

**Информационная система** – механизм для хранения, поиска, извлечения и модификации некоторой информации.

**Предметная область** – часть реального мира, с которым имеет дело ИС в зависимости от своего назначения.

Три этапа обработки информации в ИС (рисунок 1).



Рисунок 1. Три этапа обработки информации.

**База данных** – совокупность специальным образом организованных и взаимосвязанных данных.

Рассмотрим существующие варианты реализации БД. Наиболее актуальных и часто используемых в настоящее время два: технология веб-сервер и технология сервер приложений. Рассмотрим их отдельно.

В упрощенном виде **веб-серверная технология** показана на рисунке 2.



Рисунок 2. Веб-серверная технология в общем виде.

ПК браузер, в данном случае выступает источником API и необходимых библиотек, чтобы приложение, которое находится на веб-сервере или на стороне ПК могло как вести информационный обмен с БД, так и выдавать пользователю результаты его запросов к данным в удобном виде.

*Сервер приложений* показан на рисунке 3.



Рисунок 3. Технология сервера приложений.

В данном случае приложение находится на мобильном устройстве. Сервер приложений является связующим звеном между мобильным устройством и БД, принимая на себя основную процессорную нагрузку при переработке и выдаче результатов запросов пользователя. Используется API мобильных ОС (iOS, Android).

## 1.2. Актуальные модели данных.

В настоящий момент времени актуальными моделями хранения данных являются реляционная модель, noSQL модель и модель хранилища данных. Большинство баз данных функционируют в одной из перечисленных моделей. Позже мы будем подробно говорить на каждой из них, пока же сфокусируемся на их принципиальных различиях.

**Реляционная модель** хранения данных в общем, упрощенном виде показана на рисунке 4.



Рисунок 4. Упрощенная модель реляционной модели данных.

По своей структуре данная модель представляет собой коллекцию из плоских двухмерных таблиц, каждая из которых хранит информацию о каком-нибудь объекте наблюдения. Например, одной из таблиц для рис. 4 могла бы быть таблица “Студент”, если тематика БД — это университет, и мы хотим наблюдать за успехами студентов, которые в нем учатся.

Схема **хранилищ данных** также, как и реляционная модель работает по структуре связанных друг с другом таблиц. Однако таблицы содержат в себе уже не данные об объекте наблюдения, а точки зрения и показатели фактов. Накладывая точку зрения на факты, можно получить конкретные значения для требуемой сущности.

**noSQL** модель работает принципиально иначе. Принцип показан на рисунке 5.



Рисунок 5. Модель noSQL.

В данном случае понятие таблицы заменяется понятиями “дерево” и “лес”. Каждая сущность, или группа сущностей обозначается через ее “ствол”, который притягивает к себе все факты, возникающие от существования этой сущности. В зависимости от типа noSQL модели, деревья и лес могут также называться колонками или же мажорными и минорными ключами.

### 1.3. Типы баз данных, компоненты баз данных.

На верхнем уровне группировки выделяют два основных типа организации баз данных: однопользовательские и многопользовательские. Наглядно различия между типами показаны на рисунке 6.

Приложение	Пример пользователя	Количество пользователей	Примерный размер
Контакты менеджера по продажам	Менеджер по продажам	1	2000 строк
Прием пациентов в больнице	Больница	15-50	100000 строк
CRM	Отдел продаж, работы с клиентами, etc	500	10 млн. строк
ERP	Вся организация	5000	10 млн. + строк
E-commerce	Интернет пользователи	Возможно свыше 1 млн.	1 млрд + строк
Data mining	Бизнес-аналитики	25	100000 - 1 млн. строк

Рисунок 6. Примеры БД по типам.

*Однопользовательской* тут считается база данных менеджера по продажам (очевидно, потому что у этой БД только и всегда один пользователь – менеджер, в данном случае). Остальные приведенные примеры относятся к многопользовательским. В общем виде подходы к этим типам БД похожие, но при администрировании их могут существенно различаться.

Для всех типов данных структура компонентов будет одинакова, она приведена на рисунке 7.



Рисунок 7. Общая (типовая) структура компонентов базы данных.

Пользователь, через приложение баз данных (программное средство, как правило обладающее упрощенным GUI, использующее при реализации своего функционала базу данных), с использованием средства под названием СУБД (DBMS) получает необходимые для своей работы данные.

Основные *функции приложения баз данных* показаны в таблице 1.

Таблица 1. Функции приложения баз данных

Основные функции приложения баз данных
Обработка форм
Обработка запросов пользователей
Создание и обработка отчетов
Реализация логики приложения
Управление приложением

В наиболее известной реляционной модели данных к приведенной выше структуре компонентов добавится еще один слой в виде подязыка запросов к БД SEQUEL или SQL (рисунок 8).



Рисунок 8. Структура компонентов реляционной базы данных.

**СУБД** (*система управления базами данных*) – средство, предоставляющее пользователю инструментарий для взаимодействия с БД. Основные функции любой СУБД представлены в таблице 2.

Таблица 2. Функции СУБД

Создание баз данных
Создание таблиц
Создание вспомогательных элементов БД
Модификация данных
Чтение данных
Обслуживание элементов БД
Создание политик и правил
Управление целостностью данных
Восстановление и backup данных

Основные и вспомогательные элементы БД показаны на рисунке 9.

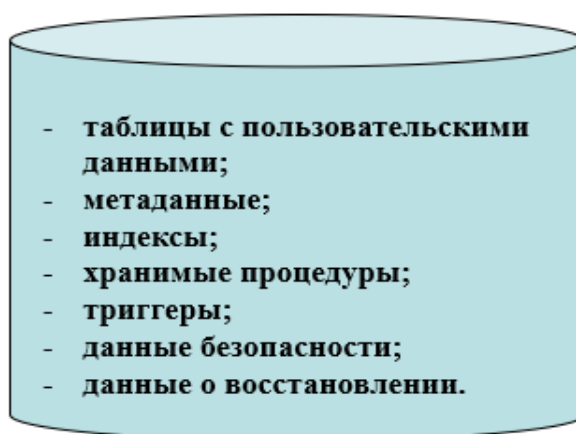


Рисунок 9. Основные и вспомогательные элементы БД.

Под *вспомогательными элементами* БД понимаются элементы оптимизации доступа и обработки данных, такие как: индексы, хранимые процедуры, триггеры, пользовательские представления; а также элементы организации безопасности данных, такие как данные безопасности, данные о восстановлении и метаданные.

**Модификация и чтение данных** – это элементы операций, обычно называемые CRUD (от англ. CREATE, RENAME, UPDATE, DELETE – соответственно создание, переименование, изменение, удаление).



Остальные 4 категории напрямую связаны с реализацией функции администратора БД и напрямую на саму структуру БД не влияют.

#### **1.4. Жизненный цикл и ресурсы БД.**

**Жизненный цикл базы данных** – время эффективной работы сложившейся структуры данных. Состоит из следующих этапов:

- проектирование;
- реализация;
- эксплуатация;
- модернизация и развитие;
- полная реорганизация.

**Организационные ресурсы БД** (лица, организующие функционирование базы данных):

- конечные пользователи;
- администраторы БД;
- разработчики приложений.

Один из ключевых организационных ресурсов БД — это ее администратор.

Основные **функции администратора базы данных**:

- анализ предметной области;
- проектирование структуры БД и связей между данными;
- первоначальная загрузка и ведение БД;
- защита данных;
- обеспечение восстановления БД;
- анализ обращений пользователей;
- анализ эффективности функционирования БД;
- работа с конечными пользователями;
- подготовка и релизы системных средств.

## 1.5. История развития технологий хранения данных.

Основные вехи истории показаны на рисунке 10.

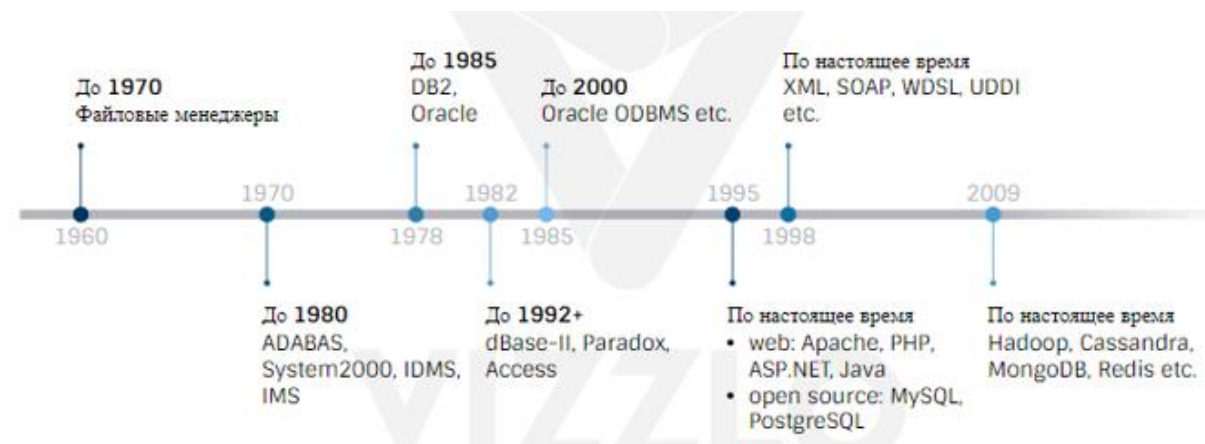


Рисунок 10. Вехи истории развития БД

1. До 1970 года отдельных технологий по хранению данных не существовало. Взаимодействие с данными происходило на уровне файлов в специальных программах, файловых менеджерах на IBM PC.
2. 1980 год – первые программы, специализирующиеся на хранении и обработке данных: ADABAS, System2000. Принято считать, что данное ПО работало в рамках пререляционной модели данных.
3. 1978-85 год – появление первых реляционных СУБД (Oracle, IBM DB2).
4. 1992 год, появление первой однопользовательской СУБД MS Access.
5. С 1995 года, появление open source, freeware ПО СУБД MySQL и PostgreSQL. Появление серверных web технологий хранения, обработки и выдачи данных (комплекс Apache, язык динамической разметки страниц интернета PHP).
6. 1998 год – изобретение удобных способов унификации данных через универсальный расширенный язык разметки XML.
7. 2010 год – появление noSQL технологий хранения данных.

### *Тестовые задания для самопроверки.*

1. Символьный тип данных CHAR характеризует следующая особенность:
  - А) значение не чувствительно к регистру символов
  - Б) фиксированное ограничение количество символов
  - В) указание ограничения по количеству символов необязательно
  
2. Какая модель баз данных не будет рассмотрена в курсе Управление данными?
  - А) реляционная модель
  - Б) объектно-ориентированная модель
  - В) noSQL модель
  
3. Базой данных для практикумов и лабораторных работ дисциплины Управление данными является:
  - А) Oracle 12c
  - Б) Microsoft Azure Cloud DB
  - В) Microsoft SQL Server Express
  
4. Ограничение CONSTRAINT, это
  - А) ограничение, накладывается таблицу, ограничивает операции с ней
  - Б) ограничение, накладывается на столбец, создает индекс первичного ключа
  - В) ограничение, накладывается на базу данных, временно ограничивает возможность работать с ней
  
5. Диалект языка SQL, который будет изучаться на практикумах дисциплины Управление данными, это:
  - А) transact-SQL
  - Б) PL/SQL
  - В) PL/pgSQL

6. Какой шаблон для данных типа дата используется по умолчанию?

А) дд-мм-гггг

Б) мм-дд-гггг

В) гггг-мм-дд

Г) гггг-дд-мм

7. Для сдачи ВСЕХ домашних работ, практикумов и отчетов в курсе Управление данными используется:

А) google classroom

Б) электронная почта

В) канал telegram

8. Каков размер точных чисел с типом данных smallint?

А) 1 байт

Б) 2 байта

В) 4 байта

Г) 8 байт

9. Что не указывается при задании свойства автоинкрементации?

А) стартовое значение инкрементации

Б) шаг инкрементации

В) тип данных инкрементируемого значения

10. QT creator, используемых на лабораторных работах по дисциплине Управление данными, это:

А) универсальное средство управления реляционными базами данных

Б) оболочка проектирования баз данных

В) платформа для разработки десктопного ПО

## 2. КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ ДАННЫХ.

### 2.1. Сущности, атрибуты и экземпляры сущностей.

*Сущность*, это объект реального мира, явно отличимый от других объектов. Например – игрушка-пазл в магазине, отдел игрушек в магазине, продавец отдела игрушек, домашний адрес продавца магазина игрушек, и т.д.

Чаще всего сущность идентифицирует *набор экземпляров сущностей*. При этом, следует отметить, что один и тот же экземпляр сущности может встречаться в базе данных в нескольких сущностях.

Каждая сущность описывается с помощью *набора атрибутов сущности*. При этом, каждому экземпляру сущности соответствует один и тот же набор атрибутов, что и остальным экземплярам этой сущности. Именно этим обстоятельством подчеркивается похожесть экземпляров одной сущности. При построении сущности, набор ее атрибутов можешь отличаться в зависимости от желаемого уровня детализации описываемой сущности. Так, например, при низкой детализации набором атрибутов для сущности Сотрудник могут быть ФИО и Фото, тогда как при высокой детализации, помимо перечисленных атрибутов могут быть использованы Номер\_парковочного\_места, Номер\_страховки и т.д...

Для каждого атрибута, пользователем определяется *домен допустимых значений*. Так, например, для атрибута Сотрудник.Фамилия доменом может быть набор символьных значений с ограничением в 20 символов, каждое значение которого должно начинаться с большой буквы.

В большинстве существующих моделей данных, для каждого экземпляра сущности должен быть определен ключ. *Ключ* – минимальный набор атрибутов, однозначно характеризующий каждый отдельный экземпляр сущности.

## 2.2. Связи и наборы связей.

**Связь** – это ассоциация между двумя или большим количеством сущностей (исключением является рекурсивная связь).

С точки зрения теории множеств, связь можно описать, как пересечение переменных, входящих в два или несколько множеств по определенным правилам. Так, например, экземпляры сущности Сотрудники связаны с экземплярами сущности Отделы.

Связи, если это необходимо, могут иметь *описательные атрибуты*. Так, если в связь необходимо включить лишь тех сотрудников, которые работали в Отделах 15.02.2020, к связи между сущностями необходимо добавить описательный атрибут Дата\_работы. Связи между сущностями могут быть представлены как в общем виде, так и в виде экземпляра связи. Пример нескольких экземпляров связи показан на рисунке 11.

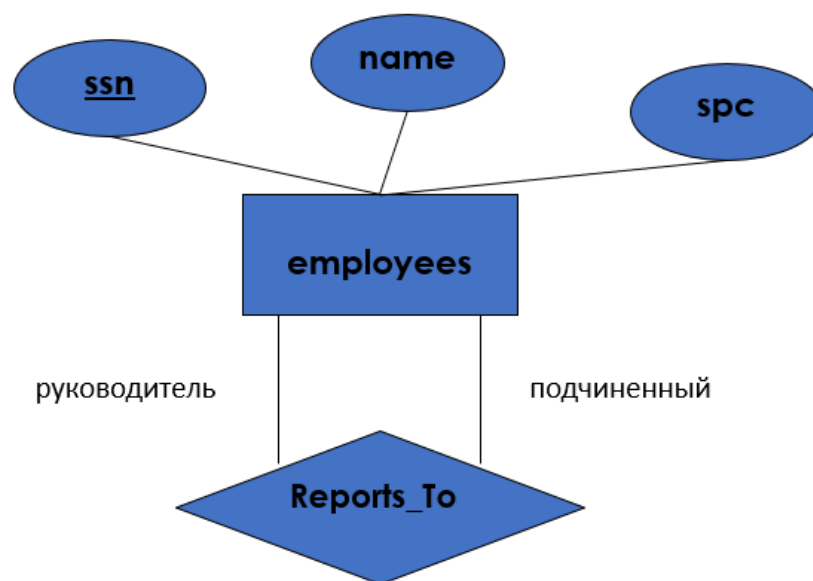


Рис. 11. Экземпляры связи “отчитывается...” сущности Сотрудники.

Необычная ситуация показанная на рисунке 11 связана с тем, что, в сущности Сотрудники могут находиться сотрудники разных ролей, например Руководитель и Подчиненный. В зависимости от того, какой экземпляр сущности участвует в связи, логика связи будет разной, что и показано на рисунке.

### 2.3. Дополнительные элементы концептуальной модели данных.

**Переменные ключа** – свойства ключей сущностей, управляющие типами связей между сущностями. Обычно выделяют три основных типа возможных связей: **один-к-одному** (один экземпляр одной сущности связан с одним и только одним экземпляром другой сущности), **один-ко-многим** (один экземпляр одной сущности связан с одним или несколькими экземплярами другой сущности) и **многие-ко-многим** (множество экземпляров одной сущности связано с множеством экземпляров другой сущности).

В графическом виде, перечисленные выше типы связей показаны на рисунке 12.

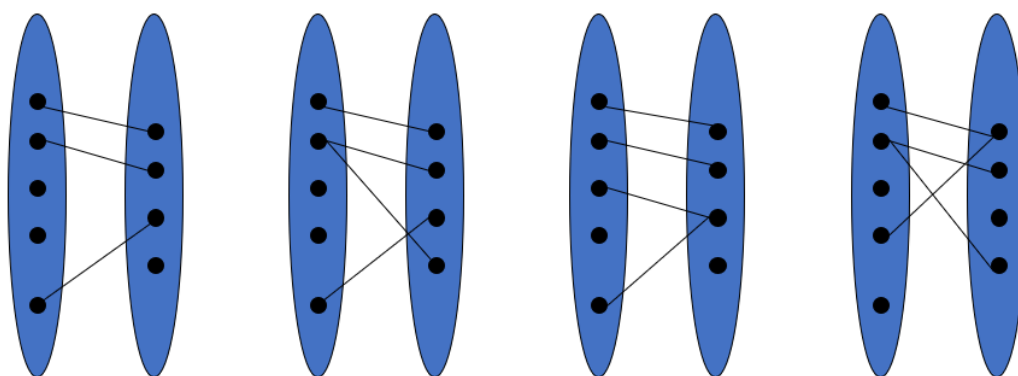


Рис. 12. Типы связей между экземплярами сущностей, управляемые переменными ключа.

Приведем несколько примеров типов связей между сущностями. Один-к-одному, связь между сущностями Сотрудник использует Служебный\_Автомобиль. За каждым сотрудником закреплен свой служебный автомобиль и его и только его он и использует.

Один-ко-многим, связь между сущностями Студент посещает Студ\_Кружок. По своему желанию, каждый студент может посещать один, а, если позволяет успеваемость и свободное время, еще и несколько кружков.

Многие-ко-многим. Связь между сущностями Музыкант играет в Оркестре. Разные музыканты регулярно собираются в разные коллективы, что, очевидно, описывается связью много Музыкантов играет в множестве Оркестров.

**Слабая сущность** – сущность, для идентификации которой необходимо наличие собственного идентифицирующего атрибута в паре с первичным ключом другой сущности (идентифицирующего владельца). Пример слабой сущности Дети (dependents) показан на рисунке 13.

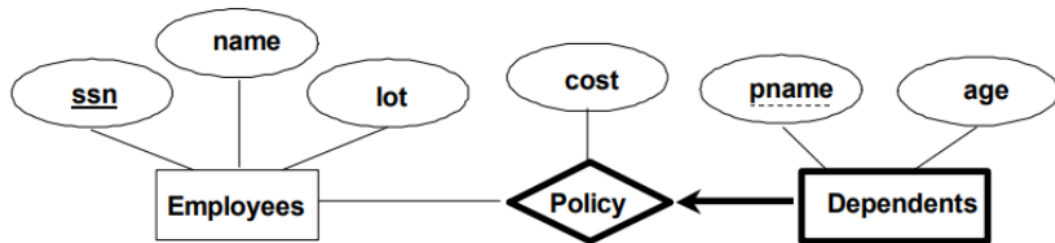


Рисунок 13. Слабая сущность Дети в связи Сотрудник обеспечивает Медицинский\_Полис Ребенку.

Предположим, что сотрудники приобретают полисы медицинского страхования для своих детей. В базе, информация о детях хранится в виде экземпляров сущности с двумя атрибутами – Имя\_ребенка (p\_name) и Возраст (Age). Очевидно, что ни один из приведенных атрибутов не обеспечивает уникальность каждого отдельного экземпляра сущности. Но, если в качестве ключа будет использована пара атрибутов ssn, pname, необходимое условие идентификации будет выполнено.

На связи со слабой сущностью накладываются два ограничения:

- связь между идентифицирующим владельцем и слабой сущностью всегда будет один-ко-многим;
- каждый экземпляр слабой сущности должен быть связан с соответствующим экземпляром идентифицирующего владельца.

**Иерархия классов сущностей.** Структура наследования атрибутов общего класса сущностей всеми их подклассами. Несмотря на общую с языками программирования логику наследования свойств классов, есть существенная



разница – при запросе к данным общего класса, каждый подкласс учитывается отдельно.

Приведем пример случая, когда декомпозиция класса на подклассы будет весьма удобна. Предположим, что группа сотрудников нашей организации работает по контракту, тогда как другая группа сотрудников работает в рамках почасовой оплаты их труда. В этом случае будет разумно разделить общий класс сущностей Сотрудники на два подкласса – Сотрудники\_Почасовая (Hourly\_Emps) и Сотрудники\_Контракт (Contract\_Emps). И тот и другой подкласс унаследуют атрибуты общего класса (например, Номер\_Сотрудника (ssn), ФИО (name) и т.д.), и, вместе с этим, будут иметь каждый свой уникальный атрибут (Зарплата\_Час (hourly\_wages) и Часов\_Работы (hours\_worked) для сущности Сотрудники\_Почасовая и Номер\_Контракта (contractid) для сущности Сотрудники\_Контракт соответственно).

На схеме связи случаи с разделением общей сущности на подсущности визуализируется элементом ISA (is-a). Пример показан на рисунке 14.

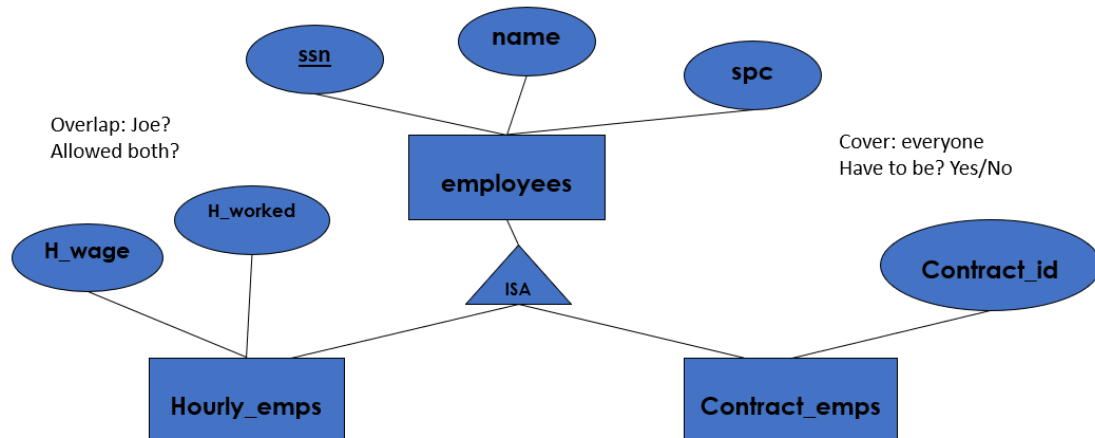


Рисунок 14. Многоуровневая иерархия ISA.

Помимо указания на присущие подклассу атрибуты, следует обратить внимание на два дополнительных условия, сопутствующих иерархии ISA – это **ограничение перекрытия** (overlap constraints) и **ограничение покрытия** (cover constraints). Ограничение перекрытия определяет – могут ли два подкласса содержать одинаковый экземпляр сущности, а ограничение покрытия определяют – могут ли все экземпляры общей сущности находиться в одном из подклассов.

**Агрегация.** Агрегация – способ реализации связи одной сущности с несколькими сущностями для тех случаев, когда организация тернарной (тройной) связи нецелесообразно. Пример агрегации приведен на рисунке 15.

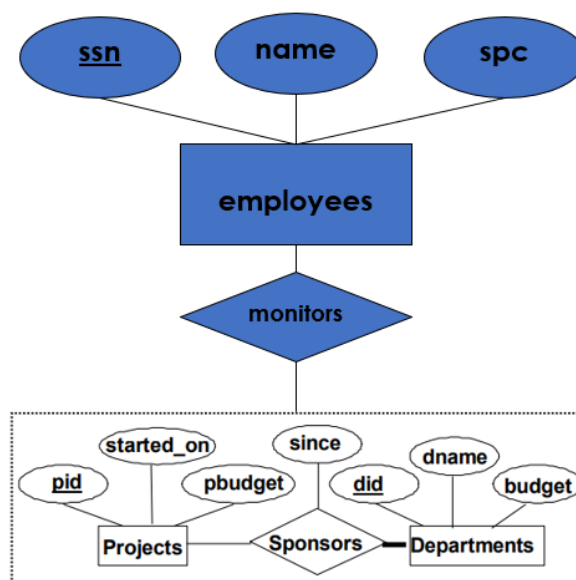


Рисунок 15. Агрегация сущностей.

Предположим, что в рамках организации, группа реализуемых Проектов (Projects) финансируется Отделами (Departments). С целью мониторинга реализуемых проектов, в каждом случае Отдел выделяет специального сотрудника (employees). Структура данных, где сущность связана с набором связей установленных между двумя другими сущностями (на рисунке 15 выделено прямоугольником) называется агрегацией.

Отметим тот факт, что логически набор связей финансирует (sponsors) не связан с набором связей мониторинг (monitors), поскольку это совершенно разные процедуры (скорее всего с разными используемыми наборами атрибутов), что не позволяет нам реализовать эту структуру хранения в виде тернарной связи.

## 2.4. Некоторые вопросы концептуального проектирования.

### *Вопрос выбора элемента хранения модели (сущность или атрибут).*

Одна из наиболее часто встречающихся проблем моделирования. Практически, любая сущность, в зависимости от обстоятельств и структуры

модели, может выступать и в качестве атрибута другой сущности. Выбор между этими двумя элементами баз данных зачастую осуществляется при проведении нормализации таблиц баз данных в ходе их проектирования.

Приведем пример, когда одна и та же логическая единица данных может выступать как сущностью, так и атрибутом сущности. Предположим, что осуществляется наблюдение за объектом Сотрудник. Разумно предположить, что организация хочет в базе данных для каждого сотрудника хранить его домашний адрес. В данном случае, Домашний\_Адрес будет выступать атрибутом сущности Сотрудник. Но что, если у сотрудника теоретически может быть несколько экземпляров Домашнего\_Адреса (адрес родителей, собственный дом, и т.д.). Очевидно, что в этом случае Домашний\_Адрес будет выступать отдельной сущностью с несколькими экземплярами внутри. Также, упомянем тот случай, когда нам необходимо использовать разные элементы Домашнего\_Адреса (дом, квартира, город) отдельно, для различных приложений или иных целей. В данном случае Домашний\_Адрес также будет создан как сущность с Атрибутами: дом, квартира, город и т.д...

На рисунке 16 приведен еще один пример вариации с выбором “сущность или атрибут”. Сотрудники работают в Отделе с момента найма до момента увольнения (промежуток времени один). Промежуток времени работы описан дополнительными атрибутами к связи Работает.

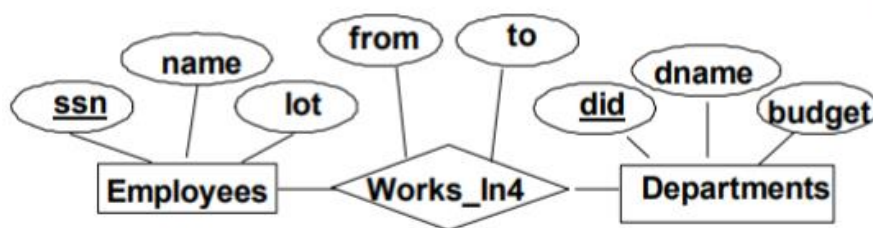


Рисунок 16. Выбор “сущность-атрибут” в пользу атрибута.

На рисунке 17 приведен пример тоже же схемы данных, но для случая, когда по какой-то причине (например - сезонная работа) сотрудники работают в Отделе в рамках нескольких временных промежутков. В этом случае будет создана сущность Длительность, и каждый временной интервал будет сохранен, как отдельный экземпляр этой сущности.

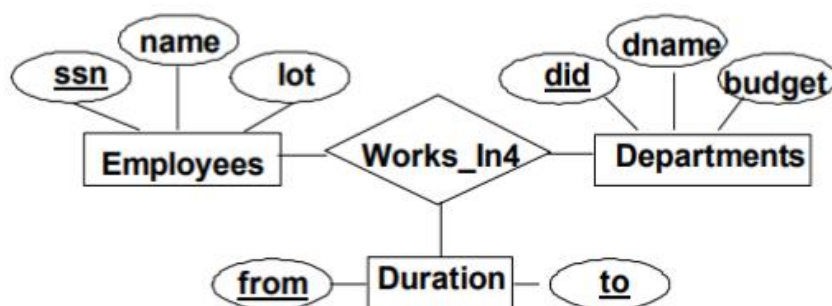


Рисунок 17. Выбор “сущность-атрибут” в пользу сущности.

Также, при принятии решения по вопросу определения “сущность-атрибут” обязательно необходимо обращать внимание на то, к какой логической модели данных планируется обратиться в ходе процесса проектирования базы данных.

***Вопрос использования бинарных связей или тернарных связей между сущностями.***

Как правило, основным элементом построения баз данных являются сущности, связанные друг с другом попарно, бинарными связями. Однако, бывают ситуации, когда применение более сложных и разветвленных связей (как правило, тернарных) более рационально и эффективно с точки зрения решения определенных задач, связанных с хранением данных. Представим ситуацию, когда объектами наблюдения являются сущности Отделы (departments), Поставщики (suppliers), Детали (parts). Отделы через контрактные обязательства закупают у поставщиков детали. На рисунке 18 показаны традиционная схема отношений, построенная на трех бинарных связях и схема отношений, построенная на одной тернарной связи.

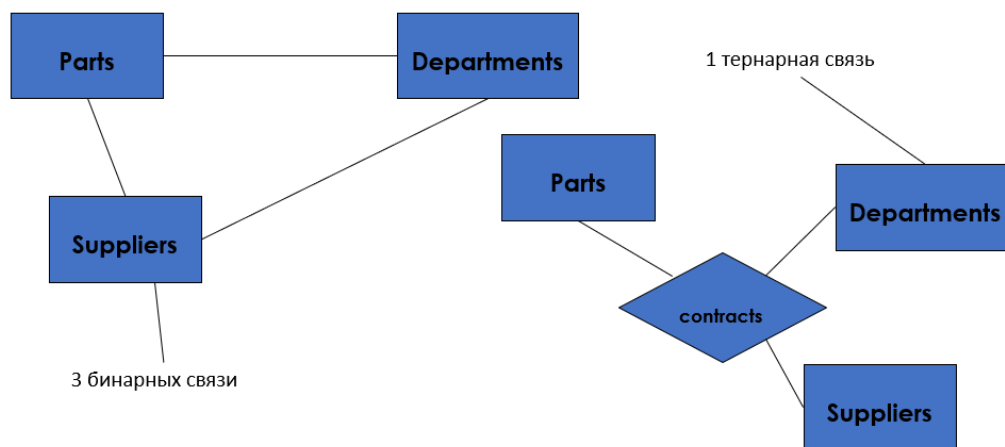


Рисунок 18. Бинарная и тернарная связи

В случае, когда приложение БД использует хранимые данные в процессе создания договоров на поставку деталей, данные будут собираться со всех трех сущностей, приведенных в схеме. В случае трех бинарных связей, данные сперва должны быть агрегированы в один массив, после чего они будут переданы на обработку на уровень приложения, тогда как в случае тернарной связи данные сразу будут готовы к обработке.

## 2.5. Концептуальный словарь данных.

**Концептуальный словарь данных** — таблица, сопровождающая концептуальную ER-модель базы данных. Основной задачей этой таблицы является фиксация имен элементов создаваемой модели для дальнейшего проектирования, типизация каждого элемента (сущность, атрибут (ключевой/не ключевой), связь (сильная/слабая)) и описание атрибута для возможности оперативного согласования работ связанных с проектированием базы данных в коллективе. Фрагмент концептуального словаря данных приведен на рисунке 19.

Концептуальное имя	Концептуальный тип	Описание
id_work	Атрибут, ключ	Уникальный идентификатор работы, выполненной художником и выставленной на аукцион
produces	Связь, идентификационно-зависимая	Связь между сущностями Artist (Художник) и Work (Работа). Родительская сущность – Artist, показывает какой художник нарисовал какую картину

Рисунок 19. Фрагмент концептуального словаря данных для базы данных “Художественный аукцион”.

На приведенном фрагменте отмечено наличие ключевого атрибута сущности id\_work. В столбце Описание дано максимально подробное и понятное описание выбранного атрибута, приведенное с целью однозначной трактовки смысла атрибута любым человеком, который будет иметь дело с этой схемой данных. Также имеется описание идентификационной-зависимой (слабой) связи с именем produces. Для этой связи тоже дано максимально подробное описание.

### ***Вопросы для самостоятельного изучения по итогам лекции.***

1. Что такое кардинальное число? Максимальная кардинальность? Минимальная кардинальность?
2. Приведите примеры, содержащие концептуальную модель со слабыми сущностями. Почему они в этой модели слабые? Как называется связь, в которой участвует слабая сущность?
3. В чем разница между связью типа ”имеет” и связью типа ”есть”? Для каждой связи привести пример.

### ***Тестовые задания для самопроверки.***

1. Какой из вариантов не является функцией СУБД?  
А) обеспечение пользователя языковыми средствами манипулирования данными

- Б) координация проектирования, реализации и ведения БД
- В) реализация языков определения и манипулирования данными
- Г) поддержка моделей пользователя
- защита и целостность данных

2. Система управления базами данных (СУБД) — это...

- А) совокупность программных средств, для создания файлов в БД
- Б) совокупность баз данных
- В) совокупность нескольких программ предназначенных для совместного использования БД многими пользователями
- Г) совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями
- Д) состоит из совокупности файлов, расположенных на одной машине

3. Чем или кем регулируется отсутствие кортежей-дубликатов:

- А) ресурсом СУБД
- Б) проектировщиком
- В) схемой данных

4. Что не относится к основным функциям СУБД:

- А) обеспечение целостности и согласованности данных
- Б) анализ массивов данных
- В) интерпретация инструкций, отправляемых пользователем или приложением

5. База данных — это...

- А) специальным образом организованная и хранящаяся на внешнем носителе совокупность взаимосвязанных данных о некотором объекте

произвольный набор информации

Б) совокупность программ для хранения и обработки больших массивов информации

В) интерфейс, поддерживающий наполнение и манипулирование данными  
компьютерная программа, позволяющая в некоторой предметной области  
делать выводы, сопоставимые с выводами человека-эксперта

6. В реляционной модели данных схема характеризует?

А) базу данных

Б) набор экземпляров отношения

В) отношение

7. Какой из вариантов связи IS-A характеризует поведение одного экземпляра сущности?

А) перекрывающий IS-A

Б) покрывающий IS-A

В) описание подходит обоим вариантам

8. Какой из вариантов связи IS-A характеризует поведение всех экземпляров сущности?

А) покрывающий IS-A

Б) перекрывающий IS-A

В) описание подходит обоим вариантам



### 3. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ И ВВЕДЕНИЕ В SQL.

#### 3.1. Определение реляционной модели данных.

**Реляционная модель данных**, это модель хранения данных (построения базы данных), состоящая из отношений и схемы отношений.

**Отношение** – это двухмерная плоская таблица, а **схема отношений** – это описание заголовков столбцов этой таблицы. Схему отношений можно также назвать метainформацией для отношения.

В реляционной модели данных отношение и схема отношений записываются следующим образом: сперва указывается название отношения, потом в скобках, разделенные запятой, перечисляются все пары названий столбцов и их типы данных.

Приведем пример отношения и его схемы отношения. Students (sid: string, name: string, login: string, age: integer). В данном случае отношением является Students (студенты), а его схемой отношения – все, что находится в скобках. Отдельно выделим использованные для элементов типы данных: string – строковая переменная, integer – целочисленная переменная. Отметим, что в логической модели реляционных данных используется ограниченный набор типов данных. Помимо уже перечисленных это может быть значение с плавающей точкой float, дата/время (темпоральный тип): date, массив данных array. Тип данных, определенный для столбца, можно считать его доменом (доменным ограничением).

При наличии нескольких экземпляров отношения, оно будет выглядеть, как показано на рисунке 20.

Sid	Name	Login	Age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Рисунок 20. Образец реляционного отношения.

### 3.2. Язык запросов SQL и реализация в нем функционала реляционной модели данных.

**Язык SQL (SEQUEL)** – это язык запросов для реляционной модели данных. С помощью инструкций этого языка осуществляется создание, модификация элементов реляционной модели хранения данных, а также манипулирование хранимыми в базе данными.

Инструкции языка представляют собой близкий к естественному языку текст обращения к СУБД. Одной из самых простых инструкций является инструкция визуализации содержимого таблицы без каких-либо ограничений. Чтобы увидеть содержимое таблицы, вводится следующая инструкция:

```
SELECT * FROM {имя_таблицы};
```

Результатом приведенной выше инструкции является визуализация отношения на экране ПК. В данной конструкции SELECT и FROM являются обязательными операторами и будут участвовать в любом запросе. Символ \* указывает на то, что результатом выполнения запроса будут все экземпляры отношения без исключения. Единственной переменной, задаваемой в данном запросе, является имя\_таблицы, к которой следует обращение. Данный запрос, через форму пользовательского приложения, через специальное приложение управления БД или любым иным доступным способом передается в реляционную СУБД, которая в свою очередь выполняет команды запроса и в ответе выдает результат в виде сформированной результатной таблицы.

Запросы с инструкцией SELECT могут быть вычисляемыми. Для этого, к базовой инструкции добавляется оператор WHERE, который позволяет сформулировать вычисляемую часть запроса. Предположим, мы хотим вывести из таблицы, приведенной на рис. 22 только тех студентов, возраст (age) которых - ровно 18 лет. Инструкция с вычисляемой частью в этом случае будет выглядеть следующим образом:

```
SELECT * FROM Students WHERE Age = 18
```

Обратите внимание, что в этом случае было явно указано название таблицы Students, а после оператора WHERE было установлено вычисляемое

ограничение выборки Возраст равен 18. Результатом этой выборки, полученным СУБД будет таблица, показанная на рисунке 21.

Sid	Name	Login	Age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2

Рисунок 21. Результат инструкции SELECT с ограничением WHERE

Язык SQL состоит из нескольких групп инструкций. Группа инструкций, поддерживающая создание, удаление, модификацию таблиц (отношений) называется Data Definition Language (DDL). Ниже будет рассмотрен принцип написания инструкций DDL SQL, которые позволяют создать, удалить и модифицировать реляционные таблицы в БД.

Для *создания реляционной таблицы* используется инструкция CREATE TABLE. В общем, наиболее минималистичном для нормальной работы виде эта инструкция выглядит следующим образом:

CREATE TABLE {имя\_таблицы} (схема отношения: пары атрибут-тип данных). Покажем пример составления инструкции для создания реляционного отношения, показанного на рисунке 20.

```
CREATE TABLE Students (sid    CHAR (20),
                        name CHAR (30),
                        login CHAR (20),
                        age  INTEGER,
                        gpa  REAL)
```

Обратим внимание на типы данных, использованные в данной инструкции. Реализация инструкции SQL такого рода приводит к созданию физической модели данных, соответственно и набор типов данных сформирован, отталкиваясь от требований СУБД, которая будет управлять данными. В данной инструкции присутствуют следующие типы данных: CHAR (character,

символьный с ограничением в 20 символов), CHAR (с ограничением в 30 символов), INTEGER (простой, целочисленный), REAL (реальные числа, дробные значения). Более подробно тема запросов и типов данных для разных СУБД будет рассмотрена в практическом курсе и в ходе лабораторных работ.

Для *изменения существующей реляционной таблицы* (любые изменения в схеме отношения) используется инструкция ALTER TABLE. В зависимости от того, какого характера изменения необходимо реализовать в схеме отношения, эта инструкция будет выглядеть по-разному. Приведем пример для случая добавления в схему отношений нового столбца. Так, для таблицы Студенты (рис. 22) добавим новый столбец Первогодки (first\_year) с типом данных (доменным ограничением) integer. Это инструкция выглядит следующим образом:

```
ALTER TABLE Students
```

```
ADD COLUMN first_year: INTEGER
```

Оператор ADD COLUMN сообщит СУБД о том, что под изменениями в схеме отношения в данном случае понимается добавление нового столбца к уже существующим. После остается только указать пару {название столбца}: {тип данных}. Обращу внимание на то, что выполняя эту инструкцию надо будет заранее решить, какими значениями будут заполнены все уже имеющиеся экземпляры отношения в этом столбце. Это могут быть как неопределенные значения NULL, так и значения по умолчанию (DEFAULT).

Для *удаления существующей реляционной таблицы* используется инструкция DROP TABLE. Стоит отметить, что при использовании этой инструкции будет удалено как отношение, так и схема отношения, иными словами – таблица будет полностью уничтожена. В случае, если схему отношения нужно сохранить, следует использовать инструкцию TRUNCATE TABLE. Приведем пример для случая удаления таблицы Студенты (Students, рис. 22). Это инструкция выглядит следующим образом:

DROP TABLE Students.

Остальные группы инструкций языка SQL, также, как и более продвинутые варианты реализации инструкций, приведенных выше будут рассмотрены в материалах практических и лабораторных работ по данному курсу.

### **3.3. Реляционные ограничения целостности.**

Одной из задач реляционных СУБД является предотвращение ввода в базу данных некорректной информации.

**Ограничение целостности** – это условие, сформулированное в схеме базы данных, ограничивающее данные, которые могут храниться в БД. Ограничение целостности первый раз определяется в процессе создания схемы отношения, после чего проверяются каждый раз, когда в отношении происходят какие-то изменения. Проверка ограничений целостности осуществляется СУБД. Задаются ограничения пользователем БД или, в некоторых случаях СУБД автоматически.

В языке SQL ограничения целостности могут быть сформулированы операторами PRIMARY KEY, CHECK, UNIQUE, FOREIGN KEY и другими. В рамках данной лекции будут рассмотрены ограничения целостности первичного ключа (PRIMARY KEY) и внешнего ключа (FOREIGN KEY).

**Первичный ключ отношения** – это атрибут, входящий в схему отношения и при этом, однозначно определяющий каждый экземпляр данного отношения (не повторяется ни при каких обстоятельствах ни для одного из экземпляров данного отношения). Первичный ключ может быть выбран как из ключей-кандидатов, как наиболее подходящий из имеющихся в схеме отношения атрибутов, так и быть искусственно создан администратором БД (как правило, для тех случаев, когда в схеме отношения нет ключей-кандидатов). Ограничения первичного ключа имеют собственное имя и задаются в процессе создания реляционного отношения с помощью оператора SQL CONSTRAINT. В качестве

примера, ниже продемонстрируем процедуру создание ограничения целостности с помощью первичного ключа для отношения, приведенного на рисунке 20.

```
CREATE TABLE Students (sid    CHAR (20),  
                        name CHAR (30),  
                        login CHAR (20),  
                        age   INTEGER,  
                        gpa   REAL,  
                        CONSTRAINT StudentsKey PRIMARY KEY (sid))
```

В данном примере созданное ограничение будет иметь имя StudentsKey, и создано оно будет на основе значений столбца sid (идентификационный номер студента). Будем полагать, что идентификационный номер студента для каждого экземпляра студента уникальный, что удовлетворяет условию целостности для первичного ключа отношения.

**Внешний ключ отношения** – элемент, обеспечивающий целостность в случае, когда информация, требуемая в одном отношении, хранится в другом отношении. В случае, когда такая информация изменяется, она должна быть проверена в другом отношении и, возможно, также модифицирована, для того, чтобы данные остались согласованными.

Для демонстрации внешнего ключа представим, что помимо таблицы Студент (рисунок 20) в базе данных есть связанная с ней таблица Записаны (Enrolled). Имеются в виду курсы, на которые записан студент. Приведем ниже схему отношения Enrolled.

Enrolled (sid: string, cid: string, grade: string).

Обратите внимание на то, что помимо атрибутов cid (идентификационный номер курса) и grade (оценка) в схеме отношения есть атрибут sid (идентификационный номер студента), который на самом деле принадлежит другому отношению – Студент, при этом являясь его первичным ключом. Для отношения Записаны этот атрибут является внешним ключом, поддерживающим целостность данных, хранимых в этом отношении. Так, если вдруг у одного из студентов изменится его идентификационный номер, то СУБД отреагирует на

эти изменения в первичном ключе отношения студент и соответствующим образом модифицирует внешний ключ в отношении Записаны. Графически связи первичного и внешнего ключей показаны на рисунке 22.

Sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Sid	Name	Login	Age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Рисунок 22. Ограничение целостности первичного и внешнего ключа в таблицах.

Из рисунка 22 следует, что внешний ключ, в отличие от первичного однозначно не идентифицирует каждый экземпляр сущности Записаны. Напротив, студент с идентификационным номером 53666 записан и посещает три курса. Отсюда следует, что основной задачей внешнего ключа является ограничение целостности в реляционных связях между сущностями. Стоит оговориться, что в примере на рис. 24 внешний ключ также является частью составного первичного ключа сущности Enrolled (sid + cid), так как только комбинация этих двух атрибутов однозначно идентифицирует каждый экземпляр этой сущности.

Ниже приведен код SQL инструкции на создание отношения Enrolled с внешним ключом sid.

```
CREATE TABLE Enrolled (sid CHAR (20),
                        cid CHAR (20),
                        grade CHAR (2),
                        PRIMARY KEY (sid, cid),
                        FOREIGN KEY (sid) REFERENCES Students)
```

Упомянутый в листинге выше элемент REFERENCES указывает на таблицу, откуда будут взяты значения для внешнего ключа sid.

Ограничение целостности, устанавливаемое при помощи внешнего ключа, также называется *ссылочной целостностью*.

Существует три варианта управления ссылочной целостностью. Все они программируют действия СУБД в случае, когда происходят изменения в источнике данных для внешнего ключа.

По умолчанию СУБД устанавливает запрет на изменения в атрибуте, являющемся источником значений для внешнего ключа. Эти изменения возможны лишь тогда, когда будет снято ограничение ссылочной целостности с отношений. В СУБД такое ограничение называется NO ACTION.

Пользователем может быть установлено ограничение, которое в случае изменений в атрибуте, являющимся источником значений для внешнего ключа выполнит аналогичные изменения во всех ссылающихся значениях внешнего ключа. Следует обратить внимание на то, что в случае удаления одного из значений, все экземпляры с соответствующим значением внешнего ключа будут полностью удалены. В СУБД такое ограничение называется CASCADE.

Пользователем может быть установлено ограничение, которое в случае изменений в атрибуте, являющимся источником значений для внешнего ключа во всех ссылающихся значениях внешнего ключа установит неопределенные значения NULL или же значения по умолчанию DEFAULT. В СУБД такое ограничение называется SET NULL или же SET DEFAULT.

Ниже показан SQL код установления разных вариантов ссылочной целостности для разных действий с данными.

```
CREATE TABLE Enrolled (sid    CHAR (20),  
                        cid.   CHAR (20),  
                        grade CHAR (2),  
                        PRIMARY KEY (sid, cid),  
                        FOREIGN KEY (sid) REFERENCES Students  
                        ON DELETE CASCADE  
                        ON UPDATE SET DEFAULT)
```

В случае удаления (delete) связанных с внешним ключом значений будет выполнено каскадное удаление соответствующих экземпляров отношения Enrolled. В случае изменений (update) в связанных с внешним ключом значениях



будет выполнена замена текущего значения в соответствующих экземплярах отношения Enrolled на значение по умолчанию (default).

### **3.4. Введение в пользовательские представления.**

*Пользовательское представление или представление (VIEW)* – это виртуальная таблица БД, содержащая внутри себя не данные сами по себе, а запрос на их формирование. Эта таблица с точки зрения СУБД практически не отличается от обычной таблицы БД, т.е. с ней можно работать с помощью запросов языка SQL (создание, изменение, удаление таблицы; вывод выборки данных на экран).

Данного рода элемент БД часто используется с целью обеспечения безопасности данных (ограничивая выборкой доступ к чувствительной информации), а также для упрощения работы с часто используемыми инструкциями на выборку данных (SELECT). Ниже будут приведены инструкции на создание и удаление представления на основе данных из таблицы на рисунке 20.

```
CREATE VIEW GoodStudents (sid, gpa)
AS SELECT sid, gpa
FROM Students
WHERE gpa > 3.0
```

Обратите внимание на то, что представление создается инструкцией CREATE VIEW. Внутри представления нет схемы отношения как в обычной таблице, а вместо этого через элемент AS приводится инструкция выборки SELECT. Далее, при обращении к представлению GoodStudents с помощью запроса на выборку SELECT пользователь может получить информацию о студентах с хорошей успеваемостью (средний балл больше 3.0). Запрос на выборку к представлению ничем не отличается от обычного запроса на выборку к таблице.

```
SELECT * FROM GoodStudents
```

Удаление представления, по аналогии с удалением таблицы осуществляется инструкцией DROP VIEW.

***Вопросы для самостоятельного изучения по итогам лекции.***

1. В чем ключевое отличие концептуальной модели данных от логической модели данных?
2. Что такое запрос к реляционной базе данных? Где можно использовать результаты этой инструкции?
3. Возможна ли сортировка на физическом уровне таблицы? Возможна ли сортировка на логическом уровне запроса или представления?
4. Приведите несколько примеров необходимости использования пользовательского представления.
5. Почему в логической noSQL модели возможно существование отношений без первичных ключей. Возможно ли это в логической реляционной модели?

***Тестовые задания для самопроверки.***

1. Столбцы таблицы называются
  - А) записи
  - Б) индексы
  - В) поля
  - Г) ключи
2. Свойства объектов в логической реляционной модели данных называют:
  - А) кортежами
  - Б) схемой
  - В) атрибутами

3. Какой тип связи имеется и допускается использовать в ER модели, но не существует в физическом модуле?

- А) многие ко многим
- Б) один к одному
- В) один ко многим

4. Программа, которая выполняет некоторые действия с информацией в базе данных, сама хранится в базе данных, при этом вызываемая пользователем это:

- А) хранимая процедура
- Б) триггер
- В) функция языка SQL

5. Представление данных — это ...

- А) таблица, содержащая действительные данные
- Б) виртуальная таблица, содержащая в теле инструкцию выборки SELECT
- В) совокупность метаданных о таблице базы данных

6. Особенность поля со свойством IDENTITY состоит в том, что ...

- А) оно имеет свойство автоматического наращивания
- Б) данные хранятся не в самом поле, а в другом месте, а в поле хранится только указатель
- В) максимальный размер числа, хранящегося в нем, не может превышать 255
- Г) оно предназначено для ввода целых чисел
- Д) автоматически превращает поле в первичный ключ

7. Оператор CREATE относится к группе операторов:

- А) манипуляции данными
- Б) описания данных
- В) задания прав доступа в базе данных

Г) защиты, восстановления данных и прочие операторы

8. Какая из перечисленных функций СУБД не является ограничением целостности?

А) PRIMARY KEY

Б) WHERE

В) CHECK

9. Какой из перечисленных вариантов ссылочной целостности не позволит удалить значение потомка, связанное с первичным ключом родителя?

А) CASCADE

Б) SET DEFAULT

В) NO ACTION

10. В каком элементе семантики SQL инструкции на выборку указывается ограничение, применяемое к массиву данных?

А) TARGET LIST

Б) RELATION LIST

В) QUALIFICATION

11. Запрос «ALTER TABLE table1 ADD c1 int NOT NULL» принадлежит группе операторов (языку):

А) определения данных (DDL)

Б) манипуляции данными (DML)

В) определения доступа к данным (DCL)

Г) управления транзакциями (TCL)

## 4. ОСНОВЫ РЕЛЯЦИОННОЙ АЛГЕБРЫ. ТРИГГЕРЫ И ПРОГРАММИРОВАНИЕ SQL.

### 4.1. Принципы работы SQL инструкции (запроса).

*SQL инструкция запроса на выборку (SELECT)*, это инструкция языка SQL, позволяющая пользователю сформировать необходимую выборку из данных, находящихся на сервере БД. Это наиболее часто применяемая инструкция языка SQL при работе с данными. Приведем типовую развернутую структуру инструкции SELECT.

SELECT [DISTINCT] target-list

FROM relation-list

WHERE qualification, где

*relation-list* – список имен отношений. Через запятую перечисляются все реляционные отношения, данные из которых понадобятся для конечной выборки. Одно и то же отношение может быть использовано в выборке несколько раз, если это необходимо. В этом и иных случаях допускается замена имени отношения произвольной переменной, которая будет использоваться в теле запроса, как имя отношения.

*target-list* – список атрибутов отношений, которые указаны в relation-list. Через запятую перечисляются все атрибуты, которые будут показаны в результатной выборке, в том порядке, в котором они были указаны в target-list.

*qualification* – условие сравнения. Варианты сравнения для выборки: отношение-значение, отношение 1 – отношение 2, по условию ( $<$ ,  $>$ ,  $=$ ,  $\leq$ ,  $\geq$ ), комбинированные с логическими операторами AND, OR, NOT).

Сформированная инструкция запрос на выборку передается в СУБД, после чего СУБД формирует в качестве результата выборку данных. В стандартном

виде (по умолчанию) последовательность действий СУБД при обработке инструкции SELECT выглядит следующим образом.

1. Рассчитать (получить данные) перечисленные отношения.
2. Исключить из результата п. 1 экземпляры отношения, которые не попадают в условия, указанные в qualification.
3. Исключить атрибуты, отсутствующие в target-list.
4. Если имеется группировка или иная сортировка данных, провести ее на последнем этапе. Принципы группировки данных для инструкции SELECT будут рассмотрены в лекции позднее.

Приведем пример обработки СУБД инструкции на выборку данных. Предположим, что к некоторым отношениям Sailors (моряки) и Reserves (Забронировано) в рамках SQL инструкции применены некоторые условия.

```
SELECT S.sname  
FROM Sailors S, Reserves R  
WHERE S.sid=R.sid AND R.bid = 103
```

Обратим внимание на то, что в relation-list данной инструкции для таблиц были созданы переменные, которые используются вместо имен таблиц в qualification. В условии qualification объединены два условия для формирования конечной выборки: массивы данных атрибутов s.sid и r.sid должны содержать идентичные значения (s.sid – первичный ключ отношения Sailors, r.sid – внешний ключ отношения Reserves), а также, должны быть выбраны экземпляры сущности Reserves, для которых значение атрибута r.bid равно 103.

Рассмотрим последовательность действий с массивом данных со стороны СУБД, рисунок 23.

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Рисунок 23. Последовательность обработки инструкции на выборку SELECT.

Сперва СУБД соберет единый массив данных из двух отношений. В него войдут все атрибуты и их значения из таблиц Sailors и Reserves. Таблица на рисунке 23, это результат объединения массивов данных двух таблиц. Далее, СУБД проверит условия, содержащиеся в qualification инструкции (WHERE S.sid=R.sid AND R.bid = 103). Единственный подходящий под оба условия сразу экземпляр выделен на рисунке рамкой (58 = 58 И bid = 103). Остальные экземпляры будут исключены из выборки. Последним будет проверит target-list. Результатом этой проверки будет выданная пользователю значение Sname rusty (значение, которое находилось в S.sname выборки с предыдущего этапа).

Следует отметить, что не во всех случаях реализация инструкции на выборку SELECT по умолчанию (как указано выше) является оптимальным вариантом. Такого рода инструкции могут быть применены на очень большие массивы данных, и логика ограничений внутри самой инструкции может быть весьма сложной, что приведет к большим потерям по вычислительным мощностям и времени в том случае, если сама инструкция выполняется не оптимальным образом.

Возможности оптимизации инструкций на выборку лежат в теории традиционной реляционной алгебры.

## 4.2. Операции реляционной алгебры.

**Реляционная алгебра**, набор операций для реляционных отношений, являющийся базой для инструкций языка SQL. Операции реляционной алгебры

могут быть представлены в инструкциях SQL в естественном виде, в ином виде, или не представлены вовсе.

На рисунке 24 перечислены все операции реляционной алгебры, а также показан приоритет их выполнения в рамках инструкции языка SQL (чем выше приоритет, тем раньше выполнится эта операция).

Операция	Приоритет
RENAME	4
WHERE	3
PROJECT	3
TIMES	2
JOIN	2
INTERSECT	2
DIVIDE BY	2
UNION	1
MINUS	1

Рисунок 24. Приоритет операций реляционной алгебры.

Ниже будут даны краткие описания операций реляционной алгебры.

**RENAME** - операция переименования (иногда, - присвоения).

Для операции достаточно только одного операнда (отношения).

Переименовывается не само отношение (таблица), а ее атрибуты.

RENAME Atr1, Atr2... AS NewAtr1, NewAtr2...

**WHERE** - выборка по условию.

Для операции достаточно только одного операнда (отношения). В тоже время, могут участвовать несколько операндов (отношений).

В результате строится новое отношение, в которое входят только те значения, которые при подстановке в условие where дают значение true.

A WHERE c



**PROJECT** - вертикальная выборка по выбранным атрибутам. Для операции достаточно только одного операнда (отношения).

В результате строится новое отношение, в которое входят значения невычеркнутых в перечислении атрибутов. Результатом часто становится таблица с единственным выбранным атрибутом.

**TIMES** - умножение одной таблицы (отношения) на другую.

Для операции обязательно наличие двух операндов (отношений). Само перемножение называется **конкатенацией**.

В результате строится новое отношение, в которое входят перемноженные друг на друга значения двух таблиц.

A TIMES B.

**JOIN** - комбинация произведения двух таблиц и заранее определенного условия.

Для операции обязательно наличие двух операндов (отношений).

В результате строится новое отношение, в которое входят перемноженные друг на друга значения двух таблиц ограниченные выбранным условием WHERE.

(A TIMES B) WHERE P.

**INTERSECT** - ищет в двух таблицах совпадающие значения.

Для операции обязательно наличие двух операндов (отношений).

В результате строится новое отношение, в которое входят кортежи, принадлежащие и первому и второму операнду.

A INTERSECT B.

***DIVIDE BY*** - результатом операции деления является набор кортежей (строк) первого отношения, которые соответствуют комбинации всех кортежей второго отношения.

Для этого нужно, чтобы во втором отношении была часть атрибутов (можно и один), которые есть в первом отношении.

В результирующем отношении присутствуют только те атрибуты первого отношения, которых нет во втором.

A ***DIVIDE BY B.***

Следует отметить, что операция деления в своем естественном виде в языке SQL не представлена. Ниже приведен пример инструкции, реализующей логику деления с использованием нескольких таблиц (используется учебная база данных СУБД MS SQL Server под названием AdventureWorks).

-- Uses AdventureWorks

```
SELECT s.BusinessEntityID AS SalesPersonID, FirstName, LastName,  
SalesQuota, SalesQuota/12 AS 'Sales Target Per Month'  
FROM Sales.SalesPerson AS s  
JOIN HumanResources.Employee AS e  
ON s.BusinessEntityID = e.BusinessEntityID JOIN Person.Person AS p  
ON e.BusinessEntityID = p.BusinessEntityID;
```

***UNION*** - оставит значения принадлежащие первой, второй таблице или обоим таблицам.

Для операции обязательно наличие двух операндов (отношений).

В результате строится новое отношение, в которое входят кортежи, первого и второго операндов без повторений.

A ***UNION B.***

***MINUS*** - оставит значения принадлежащие первой таблицы и не принадлежащие второй таблице.

Для операции обязательно наличие двух операндов (отношений).

В результате строится новое отношение, в которое входят кортежи первого операнда при условии, что их нет во втором операнде.

A MINUS B.

Для демонстрации разных вариантов построения инструкций запросов на выборку (традиционные SQL запросы и запросы с применением операций реляционной алгебры) рассмотрим три отношения с данными, рисунок 25.

Sailors				Reserves			Boats		
<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>bid</i>	<i>day</i>	<i>bid</i>	<i>bname</i>	<i>color</i>
22	Dustin	7	45.0	22	101	10/10/98	101	Interlake	blue
29	Brutus	1	33.0	22	102	10/10/98	102	Interlake	red
31	Lubber	8	55.5	22	103	10/8/98	103	Clipper	green
32	Andy	8	25.5	22	104	10/7/98	104	Marine	red
58	Rusty	10	35.0	31	102	11/10/98			
64	Horatio	7	35.0	31	103	11/6/98			
71	Zorba	10	16.0	31	104	11/12/98			
74	Horatio	9	35.0	64	101	9/5/98			
85	Art	3	25.5	64	102	9/8/98			
95	Bob	3	63.5	74	103	9/8/98			

Рисунок 25. Отношения Sailors, Reserves, Boats.

Решим задачу вывода моряков Sailors, которые брали в прокат красную или зеленую лодку. SQL код инструкции на выборку приведен ниже.

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid AND (B.color='red' OR
B.color='green')
```

В случае применения операции реляционной алгебры UNION (см. выше), код инструкции на выборку будет выглядеть несколько иначе.

```
SELECT S.sid FROM Sailors S, Boats B, Reserves R WHERE S.sid=R.sid
AND R.bid=B.bid AND B.color='red'
UNION
```

```
SELECT S.sid FROM Sailors S, Boats B, Reserves R WHERE S.sid=R.sid  
AND R.bid=B.bid AND B.color='green'
```

Обратите внимание на то, что действия СУБД во втором случае осуществляются по каждому условию B.color отдельно, после чего, результаты двух полученных массивов данных объединяются.

Далее, решим задачу вывода моряков Sailors, которые брали в прокат и красную и зеленую лодку. SQL код инструкции на выборку приведен ниже.

```
SELECT S.sid  
FROM Sailors S, Boats B1, Reserves R1, Boats B2, Reserves R2  
WHERE S.sid=R1.sid AND R1.bid=B1.bid AND S.sid=R2.sid AND  
R2.bid=B2.bid AND (B1.color='red' AND B2.color='green')
```

Обратите внимание на то, насколько усложнилась логика qualification после команды WHERE. В случае применения операции реляционной алгебры INTERSECT (см. выше), код инструкции на выборку будет выглядеть несколько иначе.

```
SELECT S.sid FROM Sailors S, Boats B, Reserves R WHERE S.sid=R.sid  
AND R.bid=B.bid AND B.color='red'  
INTERSECT  
SELECT S.sid FROM Sailors S, Boats B, Reserves R WHERE S.sid=R.sid  
AND R.bid=B.bid AND B.color='green'
```

Обратите внимание на то, что действия СУБД во втором случае осуществляются по каждому условию B.color отдельно, после чего, берется пересечение результатов этих двух массивов данных.

### 4.3. Агрегатные операторы, группировка, триггеры.

*Агрегатные функции*, набор функций, применяемых в основном на массивах числовых значениях, в результате дающий агрегатный числовой результат (одно число) в соответствии с выбранной пользователем функцией. Ниже перечислены основные агрегатные функции, чаще всего применяющиеся в инструкциях на выборку SELECT.

\* COUNT – показывает общее количество строк в выбранном атрибуте отношения, не NULL.

\* SUM - производит арифметическую сумму всех значений выбранного атрибута отношения.

\* AVG - производит усреднение всех значений выбранного атрибута отношения.

\* MAX – выводит экстремум по максимуму всех значений выбранного атрибута отношения.

\* MIN – выводит экстремум по минимуму всех значений выбранного атрибута отношения.

Приведем примеры инструкций, составленных с помощью агрегатных функций.

```
SELECT COUNT (*) FROM Sailors S
```

Результат выполнения инструкции покажет количество экземпляров отношения Sailors.

```
SELECT AVG (S.age) FROM Sailors S WHERE S.rating=10
```

Результат выполнения инструкции покажет среднее арифметическое значение возраста моряков, рейтинг которых равен 10.

*Группировка* – еще одна процедура, возможная при реализации инструкции запроса к данным SELECT. Группировка позволяет разделить полученный результатный массив данных на группы по значениям атрибутов

отношения, возможно с дополнительными условиями. Группировка всегда осуществляется последним действием и параметр *qualification* группировки всегда содержит только одно условие. Далее будут приведены операторы группировки в общем виде SQL инструкции.

SELECT [DISTINCT] target-list FROM relation-list

WHERE qualification

GROUP BY grouping-list HAVING group-qualification, где

***grouping-list*** – атрибут таблицы, по которую будет осуществляться группировка  
***group-qualification*** – условие фильтра группировки.

Далее будут приведен пример, в котором в инструкции на выборку будет команда для группировки результатных значений с дополнительным условием. Так, предположим, в таблице *Sailors* (рисунок 26) необходимо найти и вывести возраст самого юного моряка, которому больше 18 лет для рейтинга, которым обладают хотя бы 2 экземпляра сущности *Sailors*.

Сама инструкция и результат ее выполнения показаны на рисунке 26.

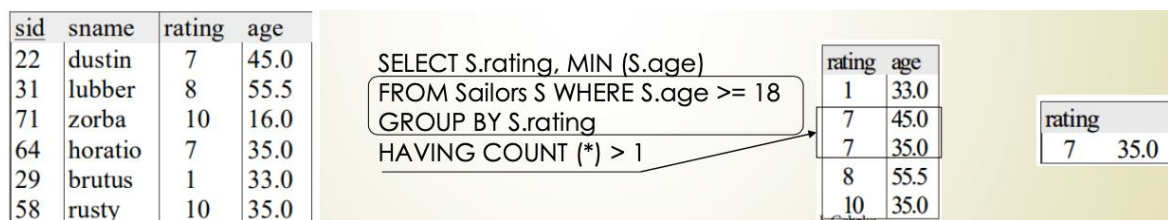


Рисунок 26. Инструкция с элементом группировки.

Последовательность выполнения СУБД инструкции на рисунке 26 следующая:

- по условию **WHERE** выбираются экземпляры, значение атрибута *age* которых больше или равно 18 (возраст).
- из выборки исключаются атрибуты, не входящие в *target-list* (*sid*, *sname*).

- применяется группировка по значению атрибута rating
- исключаются все группы, в которых нет хотя бы 2 входящих экземпляров (COUNT > 1).
- для группы с рейтингом 7 (она единственная осталась) определяется минимальное значение в массиве age (MIN (age)).

**Триггеры** – специальные хранимые процедуры, написанные в языке SQL, срабатывающие автоматически при наступлении определенного события. Состоят из трех частей:

- **событие** (помимо основной функции, запускает триггер на исполнение).

Событие может быть одной из инструкций SQL UPDATE, INSERT, DELETE.

- **условие** (проверяет, когда должен быть запущен триггер). Условие бывает или после события AFTER или вместо события INSTEAD OF.

- **действие** (что произойдет, когда триггер сработает). Это любой валидный код инструкции языка SQL.

Ниже приведен пример применения триггера.

```
CREATE TRIGGER mod_t1 on t1
AFTER UPDATE
AS
BEGIN
DECLARE @id_old INT
DECLARE @id_new INT
DECLARE @name_old VARCHAR(50)
DECLARE @name_new VARCHAR(50)

SELECT @id_old=(SELECT id FROM deleted)
SELECT @id_new=(SELECT id FROM inserted)
SELECT @name_old=(SELECT name FROM deleted)
SELECT @name_new=(SELECT name FROM inserted)
```

INSERT

INTO

t2

VALUES(@id\_old,@id\_new,@name\_old,@name\_new,GETDATE())

END

Данный триггер, после применения инструкции UPDATE (модификация данных) к одной из таблиц БД в другую таблицу БД автоматически записывает старое значение до изменения, новое значение после изменения и системную дату выполнения инструкции UPDATE (упрощенная версия журнала транзакций).

***Вопросы для самостоятельного изучения по итогам лекции.***

1. Можно ли вложить одну выборку SELECT в другую, в одном запросе? Если да, то есть ограничение на количество вложений?
2. Какие еще есть агрегатные функции у оператора SELECT приведите примеры кода с ними, на основании таблицы из лекции.
3. Какие есть события и условия для триггеров? Напишите код полезного триггера для БД из лекции.
4. Запишите операции реляционной алгебры в строгом алгебраическом виде.

***Тестовые задания для самопроверки.***

1. Какие встроенные агрегатные функции применяются в языке SQL?
  - А) SIN, COS, INTEG, DIFF2, SHIFT
  - Б) COUNT, SUM, AVG, MAX, MIN
  - В) ATAN, SUM, AVG, LOG, INC
  - Г) DIFF, CONCAT, MIN, ATAN2, LOG2
  - Д) STRCPY, PRINTF, MAX, EXIT, COUNT



2. За объединения строк таблицы в группы в синтаксисе оператора SELECT отвечает предложение:

- A) GROUP BY
- Б) HAVING
- В) WHERE
- Г) ORDER BY

3. За сортировку строк в синтаксисе оператора SELECT отвечает предложение:

- A) ORDER BY
- Б) WHERE
- В) FROM
- Г) HAVING

4. Возможно ли выполнение инструкции, которая начинается с предложения: «SELECT ProductName, SUM(Price) as Total FROM Products»

- A) возможно, если далее следует предложение GROUP BY в котором указан столбец ProductName
- Б) возможно
- В) невозможно

5. Запрос, который размещён внутри другой инструкции SELECT, UPDATE или DELETE называется:

- A) приложенным
- Б) дополнительным
- В) уточняющим
- Г) вложенным

6. Чтобы отфильтровать данные по нескольким столбцам, необходимо воспользоваться оператором ...

- A) DESC

- Б) AND
- В) NOT
- Г) OR

7. Какая из агрегатных функций возвращает количество записей в запросе:

- А) COUNT
- Б) AVG
- В) SUM
- Г) MIN
- Д) MAX

8. Какая агрегатная функция используется для расчета суммы?

- А) SUM
- Б) AVG
- В) COUNT
- Г) CONCAT

9. Инструкция для изменения таблиц, триггеров, индексов и других объектов базы данных начинается с:

- А) INSERT
- Б) BEGIN
- В) ALTER
- Г) CREATE

10. Какая инструкция удалит строку (строки) из указанной таблицы:

- А) DROP
- Б) DELETE
- В) ERASE
- Г) ALTER

11. Для удаления столбца таблицы следует выполнить инструкцию, начинающуюся с оператора:

- А) ALTER
- Б) UPDATE
- В) SELECT
- Г) DELETE

12. Запрос «SELECT CONCAT('Мой ответ', ' ', 12)» вернёт:

- А) Вернет кортеж «Мой ответ 12»
- Б) Вернет кортеж «Мой ответ», « », «12»
- В) Запрос выполнится, но будет содержать пустой столбец
- Г) запрос не будет выполнен, так как отсутствуют обязательные предложения инструкции SELECT

13. При помощи какого оператора можно изменить или присвоить имя столбцу в запросе?

- А) IN
- Б) ON
- В) LIKE
- Г) AS

14. Какой оператор нужно подставить вместо пропуска, чтобы осуществить проверку значений столбца на принадлежность к заданному множеству?  
productType \_\_\_\_ ('пирог', 'торт', 'кекс')

- А) AS
- Б) LIKE
- В) IN
- Г) ON

## 5. РАЗРАБОТКА ПРИЛОЖЕНИЙ БАЗ ДАННЫХ.

### 5.1. Универсальный API доступа к данным ODBC.

*Программный интерфейс доступа к данным*, это набор инструкций и драйверов, которые позволяют осуществлять обмен командами между программным приложением и базой данных (СУБД). В настоящее время существуют разные программные интерфейсы, большинство из которых решает узкоспециализированные задачи обеспечения взаимодействия на уровне приложения и базы данных (например ADO, JDBC и т.д.). В данной лекции будет рассмотрен наиболее универсальный на настоящий момент времени программный интерфейс, – ODBC.

**ODBC (Open Database Connectivity)** - это программный интерфейс доступа к базам данных, разработанный компанией Microsoft на основе спецификаций Call Level Interface (CLI). Стандарт CLI призван унифицировать программное взаимодействие с СУБД, сделать его независимым от поставщика СУБД и программно-аппаратной платформы.

С помощью ODBC прикладные программисты могут разрабатывать приложения для использования одного интерфейса доступа к данным, не беспокоясь о тонкостях взаимодействия с несколькими источниками. Это достигается благодаря тому, что поставщики различных баз данных создают драйверы, реализующие конкретное наполнение стандартных функций из ODBC API с учётом особенностей их продукта. На настоящий момент времени ODBC – наиболее популярное и универсальное средство обеспечения взаимодействия программных приложений и баз данных.

Для работы с ODBC необходимо этот интерфейс сперва интегрировать в операционную систему (сервера, клиентов), а также предварительно настроить. В настоящий момент времени существуют версии ODBC как для ОС MS Windows, так и для UNIX/LINUX систем, таких как Linux, MacOS и т.д.

Наиболее простая установка и настройка интерфейса ODBC происходит для ОС MS Windows (оба продукта разработаны корпорацией Microsoft, что очевидно подразумевает их высокую совместимость). В ОС MS Windows данный интерфейс устанавливается как драйвер, через автоматический установщик. В дальнейшем, настраивается в оболочке программирования для взаимодействия с разными источниками данных.

В ОС Linux и MacOS интерфейс устанавливается через оболочку bash по команде, распаковываясь из пакета. Настраивается через конфигурационные файлы, входящие в комплект поставки и конфигурационные файлы самой ОС. Это связано не только с иной архитектурой данных операционных систем, но и с дополнительными требованиями безопасности к приложениям или драйверам, которые работают в коммуникационных сетях.

В более подробном виде, принцип сборки и настройки интерфейса ODBC будет рассмотрен на лабораторных работах в рамках изучаемой дисциплины. На рисунке 27 приведен пример собранного драйвера ODBC для программного приложения баз данных, написанного в языке C++ в оболочке QT. Рассмотрен случай, когда сервер баз данных не содержит HOST-имени, а соединение происходит через IP-адрес.

```
QString connectString = "Driver={SQL Server}"; // Driver is now {SQL Server}
connectString.append("Server=10.1.1.15,5171;"); // IP,Port
connectString.append("Database=SQLDBSCHEMA;"); // Schema
connectString.append("Uid=SQLUSER;"); // User
connectString.append("Pwd=SQLPASS;"); // Pass
db.setDatabaseName(connectString);
if(db.open())
{
    ui->statusBar->showMessage("Connected");
}else{
    ui->statusBar->showMessage("Not Connected");
}
```

Рисунок 27. Листинг собранного драйвера ODBC.

В приведенном на рисунке 27 листинге осуществляется не только сборка драйвера, но также через условие if-else реализована обработка исключения “нет соединения с сервером баз данных”. Во время сборки драйвера необходимо указать следующие параметры:

***driver*** - один из драйверов, входящих в интерфейс ODBC, который будет обеспечивать взаимодействие с выбранной СУБД,

***IP, Port*** - ip-адрес сервера баз данных, с которым приложение будет работать, а также порт, через который возможно это взаимодействие; порт указывается после ip через запятую),

***schema (db)*** - название базы данных, находящейся на сервере баз данных, с которой будет работать приложение,

***user/pass*** – информация, аутентифицирующая и авторизующая приложение, как одного из пользователей базы данных.

После компиляции программного кода приложения, в случае прохождения проверки пользователя и установления соединения с сервером, программное приложение сможет передавать команды напрямую СУБД базы данных и получать оттуда результатные данные.

Приведем еще ряд актуальных программных интерфейсов доступа к данным.

***OLE DB*** (Object Linking and Embedding, Database) - набор интерфейсов, которые позволяют приложениям унифицировано работать с данными разных источников и хранилищ информации.

***ADO*** (ActiveX Data Objects) — интерфейс программирования приложений для доступа к данным. ADO позволяет представлять данные из разнообразных источников (реляционных баз данных, текстовых файлов и т. д.) в объектно-ориентированном виде.

***JDBC*** (Java DataBase Connectivity) - платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета java.sql, входящего в состав Java SE.

## **5.2. Встроенный SQL в приложениях баз данных.**

Следующим этапом после установления подключения к базе данных является программирование взаимодействия с массивами данных на сервере.

Программное приложение должно иметь возможность передать инструкцию на запись, выборку, модификацию данных СУБД, а также – получить и выдать в удобном виде результатные выборки из базы данных.

Однако, в своем естественном виде инструкции SQL не могут быть встроены в язык программирования, на котором осуществляется разработка программного средства. Это обусловлено природой данных в SQL запросе, которые представляют собой набор записей, который не привязан к физическому количеству записей в таблице. В традиционных языках программирования такой структуры данных не существует, что делает использование SQL внутри программного приложения невозможным.

С целью реализации взаимодействия десктопного приложения (в первую очередь речь идет о приложениях, написанных в языке C, C++ или Java), существует диалект языка SQL – *embedded SQL* или *встроенный SQL*. Отдельной вариацией embedded SQL является диалект языка SQL – *SQLJ* (подмножество стандарта SQL, направленное на объединение преимуществ синтаксиса языков SQL и Java ради удобства реализации бизнес-логики и работы с данными).

Принцип работы embedded SQL внутри программного приложения состоит из двух этапов.

1. Препроцессор конвертирует команду SQL в специальный API вызов.
2. Затем обычный компилятор, в обычном режиме компилирует код программы.

### ***Основные конструкции embedded SQL:***

Соединение с БД: EXEC SQL CONNECT;

Объявление переменных: EXEC SQL BEGIN (END) DECLARE SECTION;

Выражения языка: EXEC SQL Statement;

Ниже в виде листинга кода показан пример объявления переменных с помощью встроенного SQL.

```
EXEC SQL BEGIN DECLARE SECTION
```

```
char c_sname[20];
```

```
long c_sid;
```

```
short c_rating;
```

```
float c_age;
```

```
EXEC SQL END DECLARE SECTION
```

Обратите внимание на последовательность объявления переменных. В отличие от классического SQL, для каждой пары сперва объявляется тип данных (в приведенном примере это char, long, short, float), а затем – название самой переменной. Секция объявления переменных открывается и закрывается специальными командами BEGIN и END.

Фрагменты динамического SQL и SQLJ с реализацией логики приложения с краткими комментариями приведены ниже.

Пример листинга встроенного SQL (язык C).

```
char SQLSTATE[6]; //переменная для отслеживания EoF (end of file)
```

```
EXEC SQL BEGIN DECLARE SECTION //объявляем переменные для  
работы с данными
```

```
char c_sname[20];
```

```
short c_minrating;
```

```
float c_age;
```

```
EXEC SQL END DECLARE SECTION
```

```
c_minrating = random(); //значением этой переменной станет произвольное  
значение типа short
```

```
EXEC SQL DECLARE sinfo CURSOR FOR //формирование курсора  
(массива данных) для обработки на стороне приложения
```

```
SELECT S.sname, S.age FROM Sailors S
```



WHERE S.rating > :c\_minrating ORDER BY S.sname; *//классическая SQL инструкция на выборку данных, переменные с : перед названием – это переменные, созданные в приложении.*

```
do { EXEC SQL FETCH sinfo INTO :c_sname, :c_age; printf
("%s is %d years old\n", c_sname, c_age);}
while (SQLSTATE != '02000'); //построчная (FETCH) выгрузка в
программу содержимого курсора sinfo в удобном для чтения виде. 02000 – код
ошибки EoF (end of file)
```

```
EXEC SQL CLOSE sinfo; //очистка курсора sinfo
```

Пример листинга встроенного SQLJ (язык Java).

```
int sid; String name; Int rating; //задаем переменные
#sql iterator Sailors(Int sid, String name, Int rating); //задаем итератор
Sailors sailors;
...предполагаем, что приложение устанавливает рейтинг...
#sailors = {SELECT sid, sname INTO :sid, :name FROM Sailors WHERE rating
= :rating }; //возвращаем результаты
while (sailors.next()) { System.out.println(sailors.sid + " " + sailors.sname)); }
sailors.close(); //вывод результатов на экран через приложение
```

### 5.3. Использование курсоров в встроенном SQL.

Приложения, особенно интерактивные, не всегда эффективно работают с результирующим набором как с единым целым. Им нужен инструмент-посредник, позволяющий обрабатывать массив данных построчно. **Курсоры** являются расширением результирующих наборов, которые предоставляют такой механизм.

В зависимости от архитектуры приложения баз данных и желаемого распределения нагрузки по обработке данных выделяются три основных способа реализации курсоров.

**Курсоры в синтаксисе языка (наречия) SQL.** Используются в скриптах, хранимых процедурах, триггерах. Реализуются на сервере и управляются инструкциями, отправляемыми от клиента серверу.

**Серверные курсоры** интерфейса прикладного программирования (API). Курсоры API поддерживают функции курсоров в ODBC. Всякий раз, когда клиентское приложение вызывает функцию курсора API, драйвер ODBC для собственного клиента СУБД передает требование на сервер для выполнения действия в отношении серверного курсора API.

**Клиентские курсоры.** Клиентские курсоры реализуются внутренне драйвером ODBC для собственного клиента СУБД. Клиентские курсоры реализуются посредством кэширования всех строк результирующего набора на клиенте. Каждый раз, когда клиентское приложение вызывает функцию курсора API, драйвер ODBC для собственного клиента СУБД выполняет операцию курсора на строках результирующего набора, кэшированных на клиенте.

Помимо основной функции курсора, как инструмента формирования массивов данных для передачи в приложение базы данных, курсоры решают следующие задачи:

- получение выборок из отдельных строк результирующего набора;
- получение необходимого количества строк от текущей позиции в результирующем наборе;
- отслеживание и демонстрация изменения данных в строках в текущей позиции результирующего набора;
- поддержка (в зависимости от типа выбранного курсора) разных уровни видимости изменений, сделанных другими пользователями для данных, представленных в результирующем наборе.

В зависимости от решаемой задачи, программист может сформировать однонаправленный, статический или динамический курсор (тут и далее – речь про курсоры ms sql server).

**Однонаправленный курсор** указывается как FORWARD\_ONLY. Он также называется курсором firehose и поддерживает только получение строк последовательно, от начала до конца курсора. Строки нельзя получить из базы данных, пока они не будут выбраны.

Так как такой курсор не может быть прокручен назад, большинство изменений, сделанных в строках базы данных после извлечения строки, не видны. Если значение, использованное для определения положения строки в результирующем наборе, модифицируется, например в случае обновления столбца, входящего в кластеризованный индекс, то значение видимо через курсор.

Полный результирующий набор **статического курсора** создается в базе данных tempdb при открытии курсора. Статический курсор всегда отображает результирующий набор точно в том виде, в котором он был при открытии курсора. Статическими курсорами обнаруживаются лишь некоторые изменения или не обнаруживаются вовсе, но при этом в процессе прокрутки такие курсоры потребляют сравнительно мало ресурсов.

Курсор не отражает изменения в базе данных, влияющие на вхождение в результирующий набор или изменяющие значения в столбцах строк, составляющих набор строк. Статический курсор не отображает новые строки, вставленные в базу данных после открытия курсора, даже если они соответствуют критериям поиска инструкции SELECT курсора.

Статический курсор продолжает отображать строки, удаленные из базы данных после открытия курсора.

Статический курсор всегда доступен только для чтения.

**Динамические курсоры** отражают все изменения строк в результирующем наборе при прокрутке курсора. Значения типа данных, порядок и членство строк в результирующем наборе могут меняться для каждой выборки. Все инструкции

UPDATE, INSERT, DELETE, выполняемые пользователями, видимы посредством курсора. Обновления видимы сразу, если они сделаны посредством курсора с помощью функции API или специального предложения SQL.

Обновления, сделанные вне курсора, не видны до момента фиксации, если только уровень изоляции транзакций с курсорами не имеет значение READ UNCOMMITTED.

Далее будет приведен общий принцип обработки курсоров, сперва в месте его создания, а затем в логике приложения.

1. Связать курсор с результирующим набором инструкции и задать его характеристики (например возможность обновления строк).
2. Выполнить инструкцию для заполнения курсора.
3. Получить в курсор необходимые строки. Операция получения в курсор одной и более строк называется выборкой. Выполнение серии выборок для получения строк в прямом или обратном направлении называется прокруткой.
4. При необходимости выполнить операции изменения (обновления или удаления) строки в текущей позиции курсора.
5. Закрыть курсор.

Пример использования курсора показан на стр. 48 в листинге встроенного SQL.

***Вопросы для самостоятельного изучения по итогам лекции.***

1. Перечислите типы курсоров Oracle.
2. Поясните смысл использования курсорной инструкции FETCH.

Напишите простой синтаксис этой инструкции.

3. Напишите хранимую процедуру для заполнения столбца inStock таблицы WH произвольными значениями. Исходная точка – столбец заполнен значениями NULL. Всего имеется 37 строк.

4. В чем принципиальное отличие OLE DB от ODBC драйвера?

***Тестовые задания для самопроверки.***

1. На каком уровне происходит упорядочивание кортежей в реляционной модели данных:

- А) на физическом
- Б) на физическом и операционном
- В) на операционном

2. Учитываются ли скобки в операциях реляционной алгебры?

- А) да, учитываются
- Б) нет, не учитываются
- В) только в теоретико-множественных операциях реляционной алгебры

3. Какая из перечисленных моделей данных не использует инструкции языка SQL?

- А) документная модель хранения
- Б) столбцовая БД
- В) хранилище данных

4. Как называется курсор без возможности прокрутки?

- А) статический;
- Б) однонаправленный;
- В) динамический.

5. Выбрать необходимые данные из одной или нескольких взаимосвязанных таблиц, отобрать нужные поля, произвести вычисления и получить результат в виде новой таблицы можно с помощью ...

- А) запроса

- Б) схемы данных
- В) главной кнопочной формы
- Г) составной формы

6. Взаимодействие прикладных программ с большей частью СУБД осуществляется при помощи:

- А) русского языка
- Б) языка программирования C++
- В) прикладное ПО не взаимодействует с СУБД
- Г) языка структурированных запросов

7. Нормализация это...

- А) разделение единой таблицы базы данных на несколько, для дальнейшего связывания таблиц
- Б) изменение структуры базы данных с целью устранения избыточности и нарушения целостности
- В) добавление, изменение и удаление записей и таблицу

8. Что такое хранимая процедура?

- А) процедура, хранящаяся в БД
- Б) скомпилированный набор операторов Transact-SQL
- В) процедура, переданная из другой системы

9. Триггеры создаются для:

- А) корректировки БД
- Б) согласования логики связанных данных в различных таблицах
- В) поддержки целостности БД

10. XML это:

- А) язык разметки, обладающий собственным синтаксисом

- Б) утилита экспорта/импорта данных в/из СУБД
- В) язык программирования

11. Средство визуализации информации, позволяющее осуществить выдачу данных на устройство вывода или передачу по каналам связи, – это ...

- А) отчет
- Б) форма
- В) шаблон
- Г) заставка

## 6. РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ БАЗ ДАННЫХ.

**6.1. Идентификация web-ресурсов и протокол взаимодействия с web-ресурсами http.**

*Uniform Resource Identifiers* (унифицированные идентификаторы ресурсов) – специальная схема, которая идентифицирует ресурсы, проиндексированные в сети Интернет. В качестве ресурсов выступают страницы Интернет, музыка, видео, картинки и т.д. Посредством сети Интернет может осуществляться доступ к данным, хранимым на серверах баз данных.

Наиболее распространённый способ реализации URI – это url. **Единый указатель ресурса** (Uniform Resource Locator) - система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса. Ссылка, сформированная в виде url однозначно указывает на проиндексированное содержимое сети Интернет в виде файла, страницы html или иного формата, распространяемого с помощью сетевых протоколов.

Примеры URI: <http://msuniversity.ru> (ссылка на web-приложение, размещенное в сети Интернет).

<mailto:smirnovmgupi@gmail.com>. Данный URI указывает на адрес электронной почты одного из пользователей сети Интернет.

Для обеспечения взаимодействия пользователя сети интернет и необходимых ему ресурсов используется web-сервер, предоставляющий доступ к ресурсам, браузер (программное приложение на стороне клиента, формирующее запросы на основании URI) и средство передачи сообщений от браузера к web-серверу. Такие средства называются **протоколы передачи данных (TCP, IP, HTTP)**.

Логика работы любого протокола:

1. Клиент (веб-браузер) отправит запрос (составленный в виде, требуемом выбранным протоколом передачи данных) веб-серверу.
2. Веб-сервер получит запрос, сформирует на него ответ и передаст обратно веб-браузеру клиента.
3. Клиент получит ответ, создаст новые запросы.

Наиболее распространенным протоколом передачи данных в сети является протокол HTTP.

**HTTP** (HyperText Transfer Protocol) - протокол прикладного уровня передачи данных. Разрабатывался для передачи гипертекстовых документов, закодированных в формате HTML, но со временем эволюционировал в способ передачи произвольных данных.

Взаимодействие в рамках протокола HTTP делится на две фазы – кодирование запроса и кодирование ответа на запрос. Запросы формируются одним из следующих методов: OPTIONS, GET, HEAD, POST, PUT, PATCH, DELETE, TRACE, CONNECT. Ниже будет разобран пример запроса метода GET со стороны браузера пользователя Интернет.



**Строка запроса** GET ~/index.html HTTP/2

GET: поле HTTP метода (GET, POST и т.д.), который веб-пользователь желает применить к данным на веб-сервере.

~/index.html: поле URI, по которому осуществляется доступ к объекту интернета.

HTTP/2 – поле версии HTTP (актуальная версия HTTP/2, но в настоящее время осуществляется внедрение следующей версии HTTP/3).

**Тип клиента:** User-agent: Opera/...

**Типы принимаемых клиентом файлов:** Accept: text/html, image/gif, image/jpg...

Ответ – реакция веб-сервера на запрос со стороны браузера. Ниже будет разобран пример ответа на запрос метода GET.

**Строка статуса:** HTTP/2 200 OK

Версия HTTP: HTTP/2

Код статуса: 200 (варианты кодов статуса: 200 OK запрос успешен, 400 Bad Request запрос не может быть обработан сервером, 404 Not found запрошенный объект отсутствует на сервере, 505 HTTP Version not Supported)

**Дата создания (изменения) объекта:** Last-Modified: Mon, 04 May 2020...

**Количество отправляемых байтов:** Content-Length: 1024

**Тип отправляемого объекта:** Content-Type: image/jpg

При взаимодействии с протоколом HTTP (в первую очередь на уровне веб-приложения для браузера) необходимо учитывать одну существенную особенность команд протокола HTTP.

Протокол HTTP не отслеживает состояние (*state*) рабочей сессии веб-приложения. Данная особенность накладывает ряд ограничений на логику работы веб-приложения, а именно:

- нет сессионности (не отслеживается начало и конец сессии),
- протокол не запоминает предыдущих взаимодействий с веб-сервером (не сохраняется история действий пользователя),
- максимальное упрощение прямого взаимодействия с приложением (за счет исключения отслеживания состояния методы HTTP короткие и очень быстро обрабатываются),
- любая информация, требующая отслеживания состояния (логин/пароль пользователя, корзина заказов...) должна быть закодирована в каждом запросе и ответе HTTP.

Очевидно, что в некоторых случаях, когда отслеживание состояния является ключевой функцией работы веб-приложения (например – веб-приложение интернет-магазина) необходимо решение, которое добавит возможность отслеживания состояния рабочей сессии (взаимодействия веб-приложения с веб-сервером). Способами решения отслеживания состояния в HTTP являются применение cookies и динамически создаваемые url на уровне веб-сервера.

## **6.2. Форматы веб-данных.**

Основой форматирования и представления данных в сети Интернет являются два специальных формата: HTML и XML.

**HTML** (HyperText Markup Language) - стандартизированный язык разметки документов в сети Интернет. Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства. Особенности языка разметки HTML.

1. В качестве команд используются специальные тэги

2. Объект, подлежащий разметке, должен быть размещен между начальным тэгом и конечным тэгом, например: <HTML>...</HTML> или <UL>...</UL>...

3. Кодирование документа в формат HTML весьма тривиальная задача, поэтому чаще всего разметка создается в автоматическом режиме с помощью популярных прикладных программ (например, MS Word).

Список команд HTML, которые размещают в тегах ограничен, и каждая команда выполняет свою функцию. Пример функций команд HTML приведен ниже.

<UL>: неупорядоченное перечисление,

<LI>: элемент списка (перечисления),

<h1>: самый большой заголовок в документе (заголовок первого порядка),

<h2>: заголовок второго порядка, третьего порядка...,

<B>Title</B>: выделение текста жирным.

Пример страницы, написанной с помощью HTML показан на рисунке 28.

```

<HTML>
  <HEAD></HEAD>
  <BODY>
    <h1>Barns and Nobble Internet
      Bookstore</h1>
    Our inventory:

    <h3>Science</h3>
    <b>The Character of Physical
      Law</b>
    <UL>
      <LI>Author: Richard
        Feynman</LI>
      <LI>Published 1980</LI>
      <LI>Hardcover</LI>
    </UL>

    <h3>Fiction</h3>
    <b>Waiting for the Mahatma</b>
    <UL>
      <LI>Author: R.K. Narayan</LI>
      <LI>Published 1981</LI>
    </UL>

    <b>The English Teacher</b>
    <UL>
      <LI>Author: R.K. Narayan</LI>
      <LI>Published 1980</LI>
      <LI>Paperback</LI>
    </UL>
  </BODY>
</HTML>

```

Рисунок 28. Фрагмент HTML кода страницы Интернет.

Еще одним инструментом формирования страниц сети Интернет является язык XML. *XML* (eXtended Markup Language) - язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах (в отличие от HTML): разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

Благодаря гибкости системы тегов, с помощью инструментария XML можно передать любой массив данных вместе с его описанием.

Пример: Химический язык разметки.

```

<molecule> <weight>234.5</weight> <figures>...</figures>
</molecule>

```

Ниже приведен фрагмент XML-кода в виде строки, содержащей *элементы* и *атрибуты*.

```

<BOOK genre="Science" format="Hardcover">...</BOOK>

```

На каждую строку XML накладывается ряд условий и ограничений.

XML чувствителен к пробелам.

Начальный и конечный тэги должны быть одинаковыми.

Начальный (открывающий) тэг: “<“ + название элемента + “>”.

Конечный (закрывающий) тэг: “</“ + название элемента + “>”.

**Атрибуты** – это дополнительная информация об элементах.

Элемент может содержать несколько атрибутов, а может не содержать их вовсе: attribute\_name=‘attribute\_value’.

Перечисление атрибутов разделяется пробелом.

Так как рабочие тэги XML определяются пользователем и, вследствие чего, не являются строго регламентированными, контроль за правильностью документации XML возлагается на схему данных XML.

**DTD** (Document Type Definition) – схема данных для документов XML позволяющая пользователю строго определить формальную структуру языка, а также проверить новые документы XML на соответствие требованиям этой формальной структуре.

Схема данных DTD может быть как **внутренней** (прописывается внутри XML документа для каждого документа в отдельности), так и **внешней** (отдельный документ с DTD схемой, контролирующей валидность нескольких XML документов). На рисунке 29 показан фрагмент внутренней схемы DTD для XML документа.

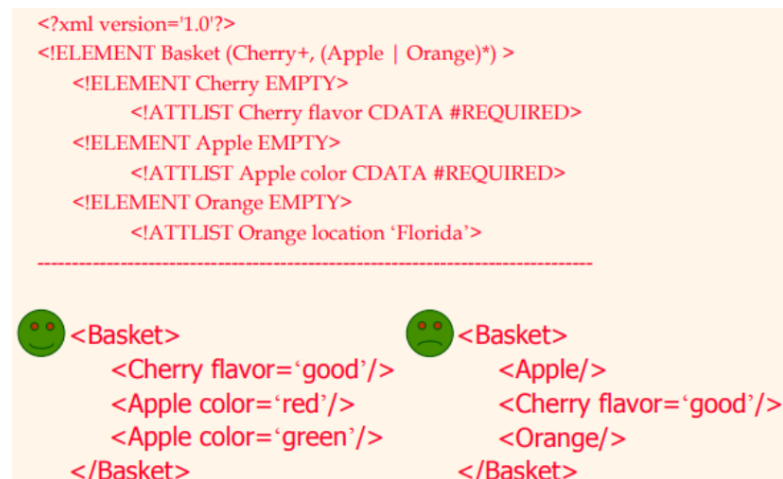


Рисунок 29. Схема DTD, корректный и некорректный XML.

В верхней части рисунка показан фрагмент DTD схемы, а в нижней части показаны два варианта XML элемента, один из которых валидный (соответствующий требованиям DTD схемы), а другой – не валидный (не соответствующий требованиям DTD схемы). Требования DTD-схемы устанавливаются через элементы и/или регулярные выражения.

**Элемент DTD** – объект, аналогичный элементам проверяемого XML. Имеет название, также может иметь ряд вложенных элементов (свойства объекта). Вложенными элементами для элемента DTD могут быть:

- другие элементы,
- #PCDATA - спарсенные (извлеченные, находящиеся в свободном доступе данные, хранящиеся на веб-сервере) символьные данные,
- EMPTY – пустой объект, без содержимого,
- ANY – любые данные, в любом формате, но без возможности проверки содержимого с помощью схемы DTD
- регулярные выражения.

Регулярные выражения DTD – элементы схемы DTD, регулирующие варианты допустимые варианты вхождений (количество) для объектов XML схемы. Ниже показаны вариации регулярных выражений DTD.

- $\text{exp}(1), \text{exp}(2), \text{exp}(3), \dots, \text{exp}(n)$  – список регулярных выражений,
- $\text{exp}^*$ , опциональное выражение с нулем или позитивным числом вхождений (этот объект может не существовать вовсе или существовать в нескольких экземплярах),
- $\text{exp}^+$ , опциональное выражение с одним или позитивным числом вхождений (этот объект должен существовать хотя бы в одном, а, возможно, и в нескольких экземплярах),
- $\text{exp}(1) \mid \text{exp}(2) \dots$ : разделение регулярных выражений

### 6.3. Трехуровневая архитектура веб-приложений. Клиентская программа (фронт-энд).

Наиболее распространенной формой архитектуры для популярных веб-приложений является *трехуровневая (трехтировая) архитектура*. Она представляет собой реализацию трех основных элементов обеспечения функционирования веб-приложения – клиентскую программу (приложение, реализованное на библиотеках веб-браузеров с GUI), сервер приложений или веб-сервер с скриптами cgi (бизнес-логика работы приложения, доступ к данным на веб-сервере) и сервер базы данных/веб-сервер (источник данных для веб-приложения). В общем виде, данная архитектура с примерами технологий реализации показана на рисунке 30.

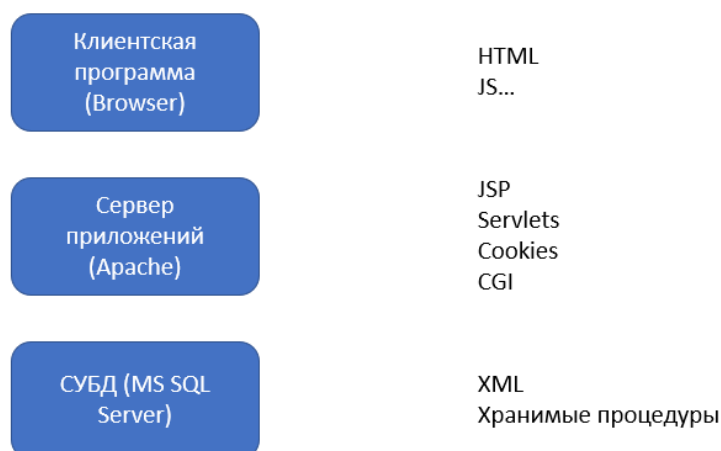


Рисунок 30. Трехуровневая архитектура веб-приложений

Далее будут рассмотрены ключевые технологии, применяемые на уровне клиентской программы и уровне сервера приложений.

**Клиентская программа** – элемент веб-приложения, позволяющий пользователю с помощью графического интерфейса (GUI) реализовывать логику обработки данных, хранящихся на веб-сервере. Функционал и графический интерфейс клиентской программы может быть реализован различными способами. Одним из наиболее распространенных способов является построение HTML-форм с элементами JavaScript.

**HTML-форма** - основной способ осуществления взаимодействия клиента с со средним уровнем архитектуры

Общий вид формы (фрагмент HTML-документа, содержащего HTML-форму):

```
<FORM ACTION="page.jsp" METHOD="GET" NAME="LoginForm"> ...  
</FORM>
```

Ключевые компоненты HTML-тэга HTML формы:

- ACTION – указание на URI по которому находится объект, необходимый для работы формы,
- METHOD – выбор HTTP метода, для реализации логики приложения (например методы GET или POST),
- NAME – название формы, которое необходимо при запуске и реализации логики скриптов на стороне клиента.

Пример HTML-формы реализующей логику авторизации клиента на веб-сервере (ввод userid, password и кнопка передачи input логина и пароля) приведен на рисунке 31.



```

<form method="POST" action="TableOfContents.jsp">
  <table align = "center" border="0" width="300">
    <tr>
      <td>Userid</td>
      <td><input type="text" name="userid" size="20"></td>
    </tr>
    <tr>
      <td>Password</td>
      <td><input type="password" name="password" size="20"></td>
    </tr>
    <tr>
      <td align = "center"><input type="submit" value="Login"
        name="submit"></td>
    </tr>
  </table>
</form>

```

Рисунок 31. Пример HTML-формы для авторизации пользователя на веб-сервере.

Следует отметить, что строгость и ограниченное количество тэгов HTML кода накладывает значительные ограничения на возможности программиста по реализации логики приложения с помощью HTML-форм. Как правило, этот инструмент используется только для формирования GUI клиентской части веб-приложения. Для расширения функциональности и возможностей HTML-формы применяются скрипты, написанные в одном из языков программирования. Чаще всего для этой цели применяется язык JavaScript.

**JavaScript** - мультипарадигменный язык программирования, обычно используется как встраиваемый язык для программного доступа к объектам приложений.

Цель применения во фронт-энде веб-приложения: добавить функциональности на уровне представления.

Примеры скриптов, используемых в HTML-документах:

- определить тип браузера и настроить (выбрать) оптимальную схему вывода html-документа под этот браузер,
- проверить и подтвердить правильность заполнения html-формы пользователем (обработка исключений),
- открытие новых окон браузера при взаимодействии пользователя с элементами GUI, поп-апы и т.д.

Скрипты JavaScript обычно встраиваются прямо в HTML-документ с тэгами `<SCRIPT>...</SCRIPT>`.

Типовые атрибуты тега `SCRIPT`: `LANGUAGE` (язык скрипта), `SRC` (внешний файл с кодом скрипта). Например: `<SCRIPT LANGUAGE="JavaScript" SRC="validate.js"></SCRIPT>` (вызов кода скрипта `validate.js` на языке JavaScript).

Пример скрипта без ссылки на файл скрипта в самом HTML: `<SCRIPT LANGUAGE="JavaScript"> <!-- alert(Welcome to the jungle!)"//--> </SCRIPT>` (вывод алерта при обращении к html-странице, содержащей этот скрипт).

Еще одной возможностью расширения функционала HTML-форм является применение таблиц стилей (*stylesheets*).

**Таблица стилей** – способ описания внешнего вида документов. Этих способов для одного документа может быть несколько, и применяться они могут в зависимости от условий вызова документа со стороны клиентского приложения (разные браузеры и т.д.).

Таблица стилей формируется с помощью специальных языков в виде кода. Файл таблицы стилей всегда сопровождает на веб-сервере выводимый с помощью этой таблицы документ. Среди специальных языков таблицы стилей выделяются два наиболее распространенных: CSS и XSL.

**Cascading style sheets** (CSS) – формальный язык описания внешнего вида документа, написанного с использованием языка разметки. Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML.

**Extensible stylesheet language** (XSL) – средство описания внешнего вида веб-страниц для документов XML.

Документ CSS может управлять множеством HTML документов, так, например, формат целого веб-сайта или веб-приложения можно изменить, просто исправив значения соответствующего CSS документа.

Каждая строка CSS состоит из трех частей: **selector** {**property**: **value**}, где **Selector** - тэг с определенным форматом (из набора допустимых в CSS), **Properly** – атрибут выбранного тэга с устанавливаемым значением, **Value** - значение атрибута.

Пример кода CSS-документа показан на рисунке 32. В качестве цвета документа устанавливается желтый (yellow), размер кегля шрифта для заголовка первого уровня документа устанавливается 36 ед., цвет заголовка третьего уровня устанавливается как голубой (blue), абзац с отступом слева на 50 ед., красного цвета (red).

```
body {background-color: yellow}  
h1 {font-size: 36pt}  
h3 {color: blue}  
p {margin-left: 50px; color: red}
```

Рисунок 32. Фрагмент таблицы стилей CSS.

***Вопросы для самостоятельного изучения по итогам лекции.***

1. Как работает схема URI mailto?
2. В чем основные недостатки толстого и тонкого клиента при применении клиент-серверной архитектуры?
3. Перечислите актуальные на настоящий момент времени серверы приложений.
4. Какие еще кроме GET и POST существуют HTTP команды? Для чего они используются?
5. Посредством чего серверы приложений взаимодействуют с базами данных (СУБД)?

## 7. ПОНЯТИЕ BIG DATA. noSQL СРЕДСТВА ОБРАБОТКИ И ХРАНЕНИЯ ДАННЫХ.

### 7.1. Определение больших данных (BigData) и их свойств. История возникновения термина BigData.

*Большие данные (англ. BigData)* - серия подходов, инструментов и методов обработки структурированных и неструктурированных данных огромных объёмов и значительного многообразия для получения воспринимаемых человеком результатов, эффективных в условиях непрерывного прироста, распределения по многочисленным узлам вычислительной сети, сформировавшихся в конце 2000-х годов, альтернативных традиционным системам управления базами данных и решениям класса Business Intelligence.

Введение термина «*большие данные*» относят к Клиффорду Линчу, редактору журнала Nature, подготовившему к 3 сентября 2008 года специальный номер журнала с темой «Как могут повлиять на будущее науки технологии, открывающие возможности работы с большими объёмами данных?», в котором были собраны материалы о феномене взрывного роста объёмов и многообразия обрабатываемых данных и технологических перспективах в парадигме вероятного скачка «от количества к качеству. Несмотря на то, что термин вводился в академической среде, и прежде всего, разбиралась проблема роста и многообразия научных данных, начиная с 2009 года термин широко распространился в деловой прессе, а к 2010 году относят появление первых продуктов и решений, относящихся исключительно и непосредственно к проблеме обработки больших данных. К 2011 году большинство крупнейших поставщиков информационных технологий для организаций в своих деловых стратегиях используют понятие о больших данных, в том числе IBM, Oracle, Microsoft, Hewlett-Packard, EMC.

В 2011 году Gartner отмечает большие данные как тренд номер два в информационно-технологической инфраструктуре. Прогнозируется, что внедрение технологий больших данных наибольшее влияние окажет на информационные технологии в производстве, здравоохранении, торговле, государственном управлении, а также в сферах и отраслях, где регистрируются индивидуальные перемещения ресурсов. С 2013 года большие данные как академический предмет изучаются в появившихся вузовских программах по науке о данных и вычислительным наукам и инженерии.

**Источниками** такого большого объема данных, который мы характеризуем, как BigData могут быть:

- логи поведения пользователей в интернете;
- GPS-сигналы от автомобилей для транспортной компании;
- данные, снимаемые с датчиков в Большом Адронном Коллайдере;
- оцифрованные книги в Российской Государственной Библиотеке;
- информация о транзакциях всех клиентов банка;
- информация о всех покупках в крупной ритейл сети.

**Основные правила** (принципы) при работе с BigData:

**1. Горизонтальная масштабируемость.** Поскольку данных может быть сколь угодно много – любая система, которая подразумевает обработку больших данных, должна быть расширяемой. Например, если в 2 раза вырос объём данных – в 2 раза увеличили количество hardware в кластере и всё продолжило работать.

**2. Отказоустойчивость.** Принцип отказоустойчивости подразумевает, что компьютеров (серверов) в кластере может быть много. Это означает, что часть этих компьютеров (серверов) будет гарантированно выходить из строя. Методы работы с большими данными должны учитывать возможность таких сбоев и переживать их без каких-либо значимых последствий.

**3. Локальность данных.** В больших распределённых системах данные распределены по большому количеству компьютеров (серверов). Если данные физически находятся на одном сервере, а обрабатываются на другом – расходы на передачу данных могут превысить расходы на саму обработку. Поэтому одним из важнейших принципов проектирования BigData-решений является принцип локальности данных – по возможности обрабатываем данные на той же машине, на которой их хранят.

## 7.2. Функция MapReduce, как инструмент работы с BigData.

**MapReduce** - модель распределённых вычислений, представленная компанией Google, используемая для параллельных вычислений над очень большими, несколько петабайт, наборами данных в компьютерных кластерах (BigData). В общем виде, модель MapReduce можно представить следующим образом (рисунок 33).

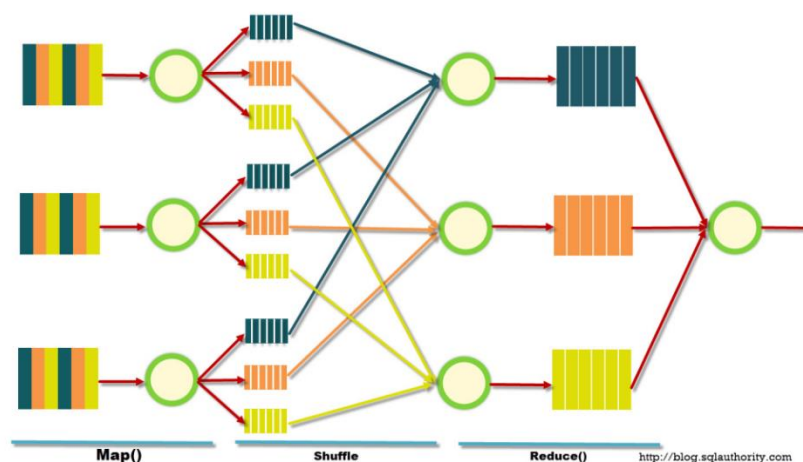


Рисунок 33. Модель MapReduce в общем виде.

Вся модель состоит из трех основных стадий: Map, Shuffle и Reduce. Первая и последняя задаются пользователем и проводятся по его команде. Процедура Shuffle проводится автоматически и пользователем не контролируется. Для наглядности, покажем конкретный пример работы функции MapReduce. На рисунке 34 показаны два варианта решения задачи определения количества (value) проданных книг А, В и С соответственно: SQL выборка и работа функции MapReduce.

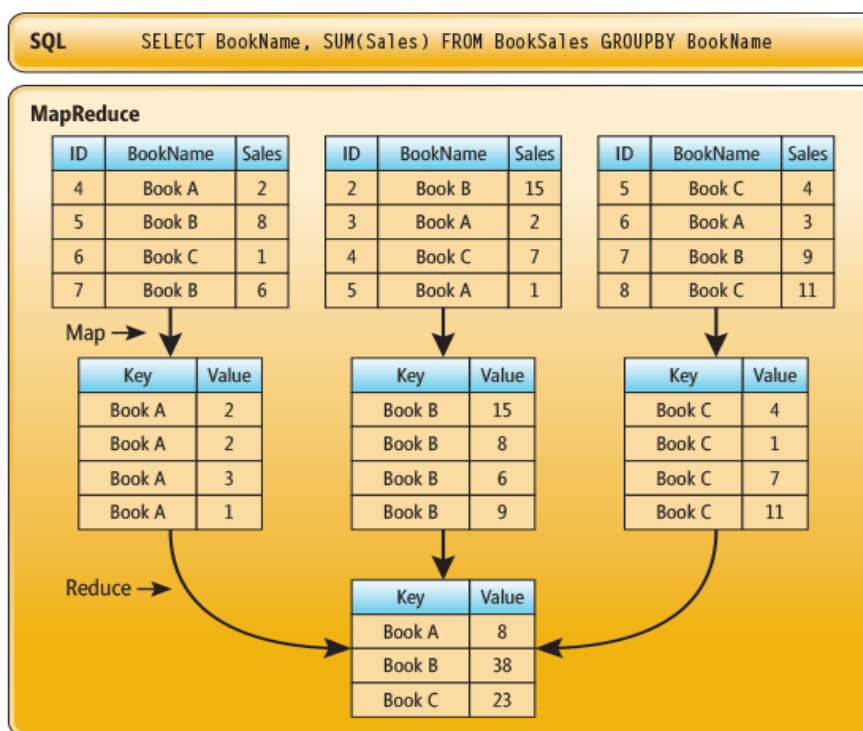


Рисунок 34. Пример реализации функции MapReduce.

Далее, рассмотрим каждый элемент функции MapReduce отдельно.

**1. Стадия Map.** На этой стадии данные предобрабатываются при помощи функции `map()`, которую определяет пользователь. Работа этой стадии заключается в предобработке и фильтрации данных. Пользовательская функция применяется к каждой входной записи. Функция `map()` примененная к одной входной записи и выдаёт множество пар **ключ-значение**. Множество – т.е. может выдать только одну запись, может не выдать ничего, а может выдать несколько пар ключ-значение. Что будет находится в ключе и в значении – решать пользователю, но ключ – очень важная вещь, так как данные с одним ключом в будущем попадут в один экземпляр функции `reduce`. На рисунке 35 приведен пример выполнения стадии Map (реализован при помощи языка Python) для случая, когда исследователь хочет выяснить, сколько раз в тексте употребляется каждое слово в отдельности.

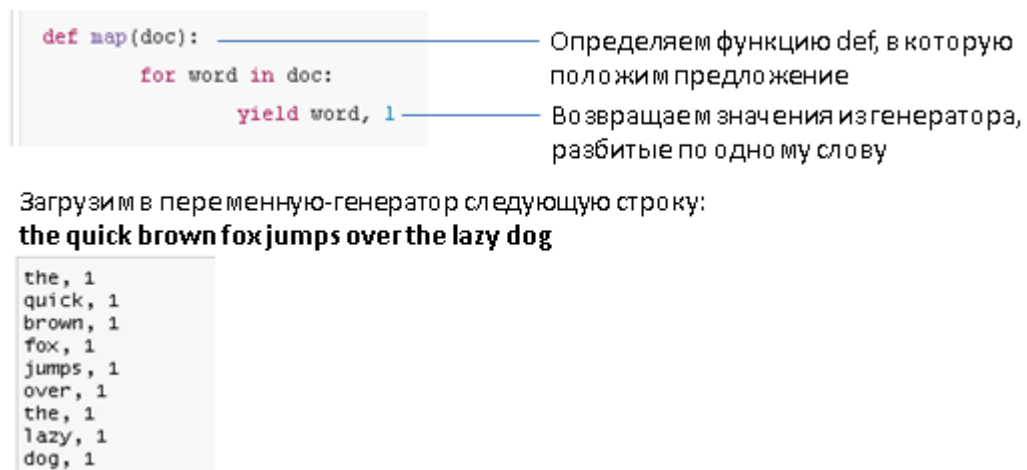


Рисунок 35. Команда Map в языке Python.

**2. Стадия Shuffle.** Проходит незаметно для пользователя. В этой стадии вывод функции map «разбивается по корзинам» – каждая корзина соответствует одному ключу вывода стадии map. В дальнейшем эти корзины послужат входом для reduce. Результат функции Map на рисунке 35 показан уже после выполнения shuffle.

**3. Стадия Reduce.** Каждая «корзина» со значениями, сформированная на стадии shuffle, попадает на вход функции reduce(). Функция reduce задаётся пользователем и вычисляет финальный результат для отдельной «корзины». Множество всех значений, возвращённых функцией reduce(), является финальным результатом MapReduce-задачи. Посмотрим, как сделать Reduce для примера, реализованного на рисунке 36.

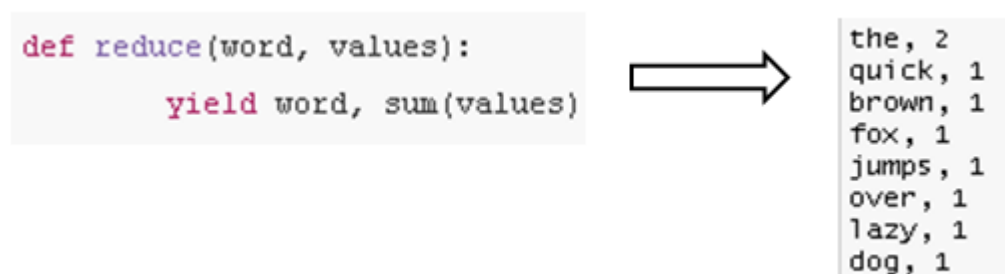


Рисунок 36. Команда Reduce в языке Python.



### 7.3. noSQL модели хранения данных.

*NoSQL* (*not only SQL, не только SQL*), термин, обозначающий ряд подходов, направленных на реализацию хранилищ баз данных, имеющих существенные отличия от моделей, используемых в традиционных реляционных СУБД с доступом к данным средствами языка SQL.

В NoSQL вместо традиционного реляционного ACID используется набор свойств BASE:

- **базовая доступность** (*basic availability*): каждый запрос гарантированно завершается (успешно или безуспешно);
- **гибкое состояние** (*soft state*): состояние системы может изменяться со временем, даже без ввода новых данных, для достижения согласования данных;
- **согласованность в конечном счёте** (*eventual consistency*) - данные могут быть некоторое время рассогласованы, но приходят к согласованию через некоторое время.

В настоящий момент времени выделяют три наиболее распространенные модели реализации noSQL хранилищ данных:

**Хранилище «ключ-значение»** (*Redis, Berkeley DB, Ryak, Amazon*).

Хранилища «ключ-значение» является простейшим хранилищем данных, использующим ключ для доступа к значению. Такие хранилища используются для хранения изображений, создания специализированных файловых систем, в качестве кэшей для объектов, а также в системах, спроектированных с прицелом на масштабируемость.

В качестве элемента хранения используется пара:

- **составной ключ**: *major keys* и *minor keys*;
- произвольное неструктурированное **значение**.

Пример реализации данных в хранилище такого типа показан на рисунке

37.

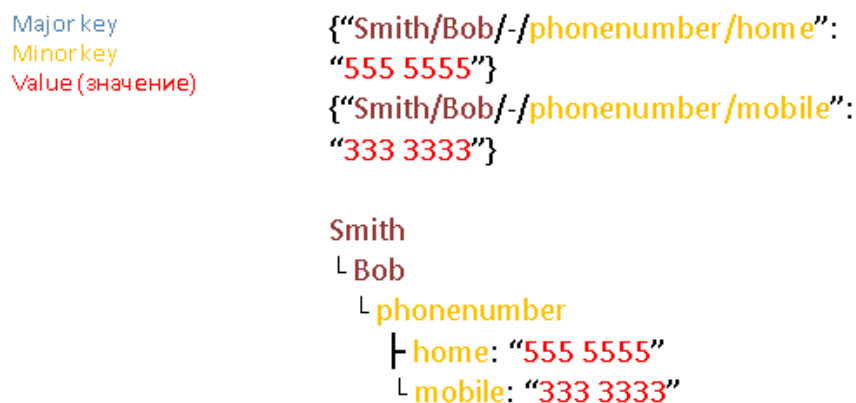


Рисунок 37. Структура данных в хранилище “ключ-значение”

### *Хранилище семейств колонок (HBase, Cassandra, Hypertable).*

В этом хранилище данные хранятся в виде разреженной матрицы, строки и столбцы которой используются как ключи. Типичным применением этого вида СУБД является веб-индексирование, а также задачи, связанные с большими данными, с пониженными требованиями к согласованности данных.

**Индексирование в поисковых системах (веб-индексирование)** - процесс добавления сведений (о сайте) роботом поисковой машины в базу данных, впоследствии использующуюся для (полнотекстового) поиска информации на проиндексированных сайтах.

Типы колонок, допустимые в хранилище:

а) **Column** (колодка) – множество пар ключ-значение (key-value);

б) **Column Family** (семейство колонок) - содержит множество колонок, у каждой из которых есть название, значение, и временная метка, и на которые ссылаются с помощью ключей строк;

в) **Keyspace** (пространство ключей) - содержит набор семейств колонок;

г) **SuperColumn** (суперколонки) - колонки, состоящие из набора подколонок.

Пример реализации данных в хранилище такого типа показан на рисунке 38.

```

users:
{ "1": { "firstName": "John",
        "lastName": "Smith",
        "phoneNumbers":
          { "home": "555 5555",
            "mobile": "333 3333"}
        },
  "2": ...
}

Get {"firstName", "lastName"}, "1"

```

Рисунок 38. Структура данных в хранилище семейств колонок.

### ***Документно-ориентированная СУБД (CouchDB, mongoDB, Exist).***

СУБД, специально предназначенная для хранения иерархических структур данных (документов). В основе ДОСУБД лежат документные хранилища имеющие структуру дерева (иногда леса). Структура дерева начинается с корневого узла и может содержать несколько внутренних и листовых узлов. Листовые узлы содержат данные, которые при добавлении документа заносятся в индексы, что позволяет даже при достаточно сложной структуре находить место (путь) искомым данным.

В отличие от хранилищ типа ключ-значение, выборка по запросу к документному хранилищу может содержать части большого количества документов без полной загрузки этих документов в оперативную память.

Документы могут быть организованы (сгруппированы) в **коллекции**.

Ключевым принципом реализации этой модели является **JSON** (JavaScript Object Notation). В качестве значений в JSON используются структуры:

**Объект** - это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }».

**Ключ** описывается строкой, между ним и значением стоит символ «:».

Пары **ключ-значение** отделяются друг от друга запятыми.

**Массив (одномерный)** - это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми.

**Значение** может быть строкой в двойных кавычках, числом, объектом, массивом, одним из литералов: true, false или null.

**Строка** - это упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки. Символы могут быть указаны с использованием последовательностей, начинающихся с обратной косой черты «\».

Пример структуры данных, описанной с помощью JSON, приведен на рисунке 39.

```
{
  "firstName": "Иван",
  "lastName": "Иванов",
  "address": {
    "streetAddress": "Московское ш., 101, кв.101",
    "city": "Ленинград",
    "postalCode": 101101
  },
  "phoneNumbers": [
    "812 123-1234",
    "916 123-4567"
  ]
}
```

Рисунок 39. Документ JSON

### ***Тестовые задания для самопроверки.***

1. Какая из функций фреймворка MapReduce выполняется без задания инструкций пользователем?

- А) shuffle
- Б) map
- В) reduce

2. Термин noSQL это:

- А) это общее обозначение принципов, направленные на воплощение механизмов управления базами данных, которые имеют ощутимые

отличия от привычных моделей с доступом к информации посредством языка SQL

Б) это набор данных с predetermined связями между ними. Эти данные организованы в виде набора таблиц, состоящих из столбцов и строк

В) логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных

3. Какая из приведенных особенностей не характерна для noSQL:

А) Применение различных типов хранилищ

Б) Линейная масштабируемость (добавление процессоров увеличивает производительность)

В) Невозможность разработки базы данных без задания схемы

Г) Инновационность: «не только SQL» открывает много возможностей для хранения и обработки данных

4. Какой вид БД наиболее приемлем для работы с BigData:

А) Реляционная БД

Б) NoSQL

В) Сетевая БД

5. Какой тип noSQL модели хранения данных характеризуется наличием в массиве данных major и minor ключей?

А) хранилище типа “ключ-значение”

Б) хранилище типа “семейство колонок”

В) документное хранилище

6. В хранилище семейств колонок, колонки, состоящие из наборов колонок называются:

- A) Column
- Б) Keyspace
- В) ColumnFamily
- Г) SuperColumn

***Ответы на тестовые задания.***

Лекция 1.

1.Б. 2.Б. 3.В. 4.Б. 5.А. 6.В. 7.А. 8.Б. 9.В. 10.В

Лекция 2.

1.Б. 2.Г. 3.А. 4.Б. 5.А. 6.В. 7.А. 8.А.

Лекция 3.

1.В. 2.В. 3.А. 4.А. 5.Б. 6.А. 7.Б. 8.Б. 9.В. 10.В. 11.А.

Лекция 4.

1.Б. 2.А. 3.А. 4.А. 5.Г. 6.Б. 7.А. 8.А. 9.В. 10.Б. 11.А. 12.А.

Лекция 5.

1.В. 2.А. 3.А. 4.Б. 5.А. 6.Г. 7.Б. 8.Б. 9.В. 10.А. 11.А.

Лекция 7.

1.А. 2.А. 3.В. 4.Б. 5.А. 6.Г.

## Список литературы

1. **Основы использования и проектирования баз данных : Учебник / В. М. Илюшечкин.** — Мск.: Юрайт, 2017. — 214 с.: ил..
2. **Базы данных. Проектирование, реализация и сопровождение. Теория и практика : Пер. с англ..** — М.: Вильямс, 2017. — 1440 с.: ил..
3. **Database Processing: Fundamentals, Design and Implementation : англ. / D. M. Kroenke, D. J. Auer.** — PEARSON, 2017. — 639 с.: ил..
4. **Теория и практика построения баз данных : Учебник / Д. Кренке.** — Спб.: Питер, 2005. — 859 с.: ил.. 5. **Базы данных: Модели и языки : Учебник / С. Кузнецов.** — Мск.: Бином, 2008. — 720 с.: ил..
5. **Базы данных: Модели и языки : Учебник / С. Кузнецов.** — Мск.: Бином, 2008. — 720 с.: ил..



### Сведения об авторе

Смирнов Михаил Вячеславович, кандидат экономических наук, доцент кафедры “Предметно-ориентированные информационные системы” Института комплексной безопасности и специального приборостроения.