

# **Базы данных**

## **Лекция 01 – Введение**

**Преподаватель: Поденок Леонид Петрович, 505а-5**

**+375 17 293 8039 (505а-5)**

**+375 17 320 7402 (ОИПИ НАНБ)**

**prep@lsi.bas-net.by**

**ftp://student:2ok\*uK2@Rwox@lsi.bas-net.by**

**Кафедра ЭВМ, 2024**

2024.02.06

# Оглавление

Введение в системы баз данных.....	3
На основе файловых систем.....	3
Системы баз данных.....	5
Пользователи.....	5
Прикладные программы.....	8
База данных.....	9
Система управления базами данных.....	10
Трехуровневая архитектура ANSI-SPARC.....	11
Схема базы данных (резюме).....	19
Независимость от данных.....	21
Система управления базами данных.....	22
Функции СУБД.....	22
ACID.....	25
Модель «Сущность-связь».....	27
ER-диаграмма.....	28
Символы ERD-сущностей.....	29
Символы ERD-связей.....	31
Символы ERD-атрибутов.....	32
Символы физических ER-диаграмм.....	34
Поля.....	34
Ключи.....	35
Типы.....	36
Нотация ER-диаграмм.....	37
Установка и первичная настройка postgres.....	39

# Введение в системы баз данных

Базы данных (БД) — неотъемлемая часть современной жизни (наиболее значимые сферы применения — коммуникационные системы, транспорт, финансовые системы, наука).

Дальнейшее расширение применения — практически везде, где есть необходимость хранения и поиска данных.

## На основе файловых систем

Начало истории развития БД может характеризоваться следующими этапами:

### 1) начальное накопление данных — книги

Это бумажные картотеки (карточки, папки) обычно индексируемые по какому-либо признаку для ускорения поиска (например, картотека библиотеки) — ручные операции, *эффективен только поиск по индексу* (по фамилии авторов, названию книги) и *практически невозможен по сложным критериям* (найти среднее число книг написанных авторами с фамилией начинающейся на «А» и т.п.).

Рост требований по поиску разнообразной информации (усложнение запросов, рост числа данных, требование быстрого ответа на запрос), а также рост мощностей и доступности вычислительной техники (особенно появление магнитных носителей (ленты, диски)) привело к появлению файловых систем.

**2) Файловая система** — набор системных программных средств, которые выполняют некоторые операции для пользователей (ввод данных, операции с файлами, генерация фиксированного набора специализированных отчетов), каждая программа определяет жестко свои собственные данные (структура и методы доступа) и управляет ими, все данные децентрализованы и хранятся в местах их обработки (например, по отделам предприятия).

Данные в этих системах хранятся как наборы записей (record) в файлах, каждая запись содержит поля (field) хранящих определенные характеристики.

## **Ограничения файловых систем:**

1) разделение и изоляция данных — данные для обработки должны выбираться из нескольких файлов (сложность одновременной обработки данных из нескольких файлов);

2) дублирование данных — (накопление файлов с данными по одному объекту в различных отделах) — неэкономное использование ресурсов (дисковое пространство) и риск нарушения целостности данных (ошибки, если данные в разных отделах различаются, требуются проверки данных);

3) зависимость от данных — (физическая структура и способы доступа различны для разных приложений) сложно изменять файлы (добавить новое поле), для преобразования данных нужны специальные программы-конверторы и изменение приложений;

4) несовместимость форматов файлов — (если приложения создаются с использованием различных языков программирования (COBOL, PL/1)) необходима выработка общего формата хранения (затраты времени);

5) фиксированные запросы (рост количества приложений) — кол-во приложений пропорционально количеству типов запросов ==> с ростом к-ва запросов растет числа приложений (документирование и поддержка функциональности системы — усложнение сопровождения, снижение мер безопасности по защите).

Ограничения файловых систем — следствие двух фундаментальных причин:

1) определение данных содержится внутри приложений, а не отдельно от них;

2) кроме приложений нет других инструментов для доступа к данным и их обработки.

## Системы баз данных

Система баз данных (СБД) — компьютеризированная специализированная система для хранения информации и доступа к ней. Компоненты СБД:

- пользователи;
- прикладные программы;
- базы данных;
- система управления базами данных.



Рисунок 1 – Компоненты системы баз данных

### Пользователи

- 1) Пользователи делятся на 4 группы:
- 1) администраторы;
  - 2) разработчики баз данных;
  - 3) прикладные программисты;
  - 4) простые (наивные) пользователи.

## **Администраторы**

- администраторы данных;
- администраторы баз данных.

**Администраторы данных** — отвечают за *управление данными*, включая планирование БД, разработку и сопровождение стандартов, бизнес правил (описывают основные характеристики данных с точки зрения организации) и деловых процедур, а также *концептуальное* и *логическое* проектирование БД;

**Администраторы баз данных** — отвечают за физическую реализацию БД, включая физическое проектирование, обеспечение безопасности и целостности данных, а также обеспечение максимальной производительности приложений и пользователей (более технический характер по сравнению с администратором данных).

## **Разработчики баз данных**

- разработчики логической базы данных;
- разработчики физической базы данных.

**разработчики логической базы данных** занимаются идентификацией данных, связей между данными и устанавливают ограничения, накладываемые на хранимые данные — (ответ на вопрос **ЧТО?**);

**разработчики физической базы данных** по готовой логической модели создают физическую реализацию (формирование таблиц, выбор структур хранения, методов доступа, мер защиты) — (ответ на вопрос **КАК?**).

## **Прикладные программисты**

Создают приложения предоставляющие пользователям необходимые функциональные возможности (действия над базой данных).

## **Пользователи (клиенты БД)**

- наивные пользователи — осуществляют доступ к БД через прикладные программы;
- опытные пользователи — могут осуществлять доступ к БД с использованием языков запросов или создавать собственные прикладные программы.

## Прикладные программы

- пользователи;
- **прикладные программы;**
- базы данных;
- система управления базами данных.

Прикладные программы обеспечивают удобный для пользователей доступ к БД.

Реализуются с использованием языков программирования 3-го (процедурные языки — C, COBOL, Fortran, Ada, Pascal) или 4-го поколения (SQL, QBE). 4-е поколение (“4GL”) — непроцедурные языки, возможно генерирование прикладного приложения по параметрам, заданных пользователем.

Языки программирования делятся на:

- языки представления информации (языки запросов или генераторы форм и отчетов);
- специализированные языки (электронных таблиц и БД);
- генераторы приложений для определения, вставки, обновления или извлечения сведений из БД;
- языки очень высокого уровня для генерации кода приложений.



## **База данных**

- пользователи;
- прикладные программы;
- **базы данных;**
- система управления базами данных.

*База данных* (БД) — совокупность логически связанных данных, хранящихся в компьютеризованной системе и отражающих некоторую предметную область человеческой деятельности.

БД — единое, большое хранилище данных (набор интегрированных записей с самоописанием), содержащее данные с минимальной долей избыточности, к которым может обращаться большое число пользователей.

Описание данных называется системным каталогом или словарем данных, а сами элементы описания — метаданные (данные о данных) обеспечивают независимость между программами и данными.

## **Система управления базами данных**

- пользователи;
- прикладные программы;
- базы данных;
- **система управления базами данных.**

Система управления базами данных (СУБД) — программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать БД, а также осуществлять к ней контролируемый доступ.

Главные преимущества СУБД — преодоление ограничений файловых систем (в основном из-за обеспечения централизованного управления данными).

## Трехуровневая архитектура ANSI-SPARC

Основная цель СУБД — предоставление пользователям данных в абстрактном представлении, т.е. сокрытие от пользователей особенностей хранения и управления данными.

Традиционный подход к построению систем ранее был сосредоточен только на определении данных из двух разных представлений:

- представление пользователя (внешняя схема)
- представление компьютера (внутренняя схема).

*С точки зрения пользователя*, определение данных находится в контексте отчетов и экранов, предназначенных для помощи людям в выполнении их конкретной работы.

Требуемая структура данных из представления пользователя меняется в зависимости от бизнес-среды и индивидуальных предпочтений этого пользователя.

*С точки зрения компьютера*, данные определяются в терминах файловых структур для хранения и поиска.

Требуемая структура данных для компьютерного хранения зависит от конкретной используемой компьютерной технологии и необходимости эффективной обработки данных.

Для удовлетворения конкретных потребностей бизнеса на протяжении многих лет эти два традиционных представления данных определялись аналитиками для каждого приложения.

Как правило, внутреннюю схему, определенную для исходного приложения, нельзя сразу использовать для последующих приложений, что в результате приводило к созданию избыточных и часто противоречивых определений одних и тех же данных. Данные определялись компоновкой физических записей и последовательно обрабатывались в ранних информационных системах.

Признание этой проблемы привело исследовательскую группу ANSI/X3/SPARC по системам управления базами данных к выводу, что в идеальной среде управления данными необходимо третье представление данных — «концептуальная схема».

## Концептуальная схема

**Концептуальная схема представляет собой единое интегрированное определение данных в рамках предприятия, которое не привязано к какому-либо отдельному применению данных и не зависит от того, как данные физически хранятся или к ним осуществляется доступ.**

Целью этой концептуальной схемы является предоставление последовательного определения значений и взаимосвязей данных, которые можно использовать для интеграции, совместного использования и управления целостностью данных.

Отчет ANSI/SPARC был задуман как *основа* для компьютерных систем, *способных обмениваться информацией*. Все поставщики баз данных приняли терминологию трех схем, но реализовали ее несовместимыми способами. В течение следующих двадцати лет различные группы пытались определить стандарты для концептуальной схемы и ее сопоставления с базами данных и языками программирования. К сожалению, ни у одного из поставщиков не было сильного стимула делать свои форматы совместимыми с форматами конкурентов. Было подготовлено несколько отчетов, но не стандартов.

По мере развития практики администрирования данных и появления новых графических методов термин «схема» уступил место термину «модель».

Концептуальная модель представляет собой представление данных, которое согласовывается между конечными пользователями и администраторами баз данных, охватывающее те объекты, о которых важно хранить данные, значение данных и взаимосвязь данных друг с другом.

Цель трехуровневой архитектуры состоит в том, чтобы отделить представления пользователей друг от друга, что достигается отделением пользовательского представления базы данных от ее физического представления:

- трехуровневая архитектура допускает *независимые настраиваемые* представления пользователей. Это значит, что каждый пользователь должен иметь возможность доступа к одним и тем же данным, но иметь различное настраиваемое представление о них. Представления должны быть независимыми – изменения в одном представлении не должны влиять на другие.
- трехуровневая архитектура скрывает информацию о физическом хранилище от пользователей, т.е. пользователям не нужно иметь дело с информацией о физическом хранилище БД.
- администратор базы данных (администратор данных) должен иметь возможность изменять структуры хранения базы данных, не затрагивая представления пользователей.
- на внутреннюю структуру базы данных не должны влиять изменения физических аспектов хранилища, например, переход на новый диск. или подлежащую файловую систему.

Архитектура ANSI-SPARC имеет три уровня:

- 1) **Внешний**, реализующий представления пользователя и описывающие различные внешние представления данных. **Для данной базы данных может быть много внешних схем.**
- 2) **Концептуальный**, реализующий способ описания того, какие данные хранятся во всей базе данных и как эти данные взаимосвязаны.

Концептуальная схема описывает все элементы данных и отношения между ними вместе с ограничениями целостности. **Для каждой базы данных существует только одна концептуальная схема.**

- 3) **Внутренний**, определяющий как база данных физически представлена в компьютерной системе. Внутренняя схема на самом нижнем уровне содержит определения хранимых записей, методов представления, полей данных и индексов. **Для каждой базы данных существует только одна внутренняя схема.**

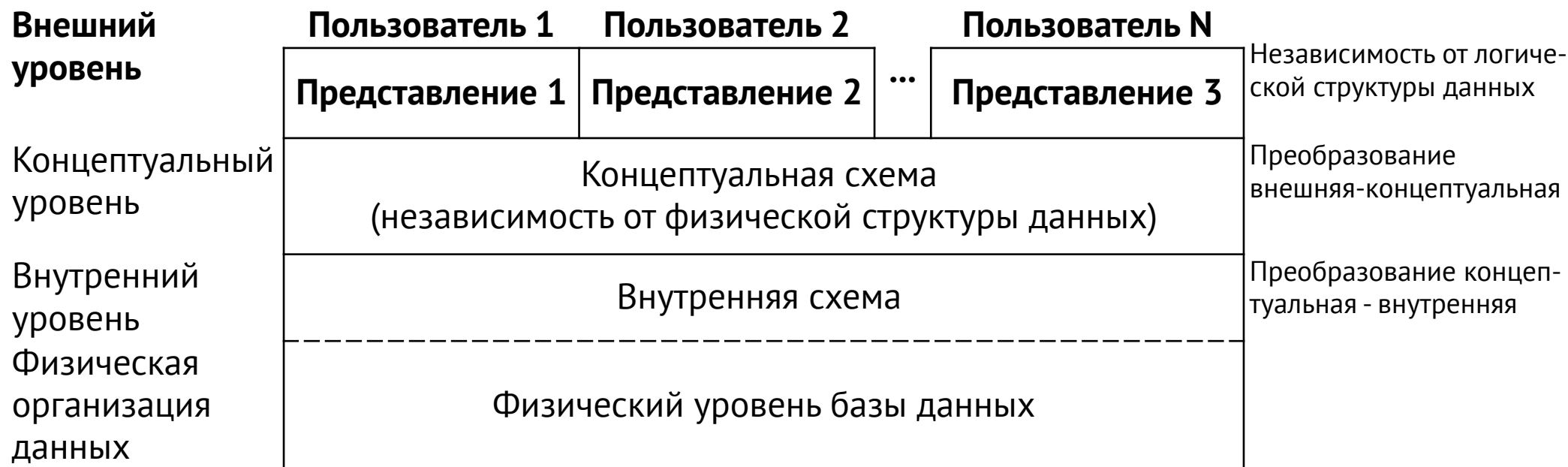


Рисунок 2 – Трехуровневая архитектура ANSI-SPARC.

### Внешний уровень

Внешний уровень – представление БД с точки зрения пользователей.

Для каждой группы пользователей описываются только необходимая им часть БД.

Т.е. каждый пользователь имеет свое представление о «реальном мире», и не видит лишние с его точки зрения данные.

Разные представления (view) могут по-разному отображать одни и те же данные (например, дату).

Часть данных при этом может не храниться в БД (так называемые производные и вычисляемые данные), а получаться по мере надобности (например, возраст сотрудников), что не потребует лишних обновлений БД и уменьшит ее объем.

Внешний  
уровень

**Концептуальный  
уровень**

Внутренний  
уровень

Физическая  
организация дан-  
ных

Пользователь 1	Пользователь 2	...	Пользователь N
Представление 1	Представление 2	...	Представление 3
<b>Концептуальная схема</b> <b>(независимость от физической структуры данных)</b>			
<b>Внутренняя схема</b>			
<b>Физический уровень базы данных</b>			

Независимость от логической  
структуры данных

Преобразование  
внешняя-концептуальная

Преобразование концеп-  
туальная - внутренняя

## Концептуальный уровень

Концептуальный уровень — это обобщающее представление БД.

Уровень описывает, какие данные хранятся в БД, а также связи между ними.

Является промежуточным уровнем в архитектуре и содержит логическую структуру всей БД с точки зрения администратора базы данных (DBA), а именно:

- все сущности;
- их атрибуты;
- связи;
- накладываемые на данные ограничения;
- семантическая информация о данных;
- информация о мерах безопасности и поддержки целостности данных.

Все внешние представления получаются из данных этого уровня, но этот уровень не содержит никаких сведений о методах хранения данных (например, может быть информация о длине полей их типе, названии, но нет данных об объеме в байтах и т.п.).

Внешний уровень	Пользователь 1	Пользователь 2	...	Пользователь N	Независимость от логической структуры данных
	Представление 1	Представление 2		Представление 3	
Концептуальный уровень	Концептуальная схема (независимость от физической структуры данных)				Преобразование внешняя-концептуальная
<b>Внутренний уровень</b>	<b>Внутренняя схема</b>				Преобразование концептуальная - внутренняя
Физическая организация данных	Физический уровень базы данных				

### Внутренний уровень

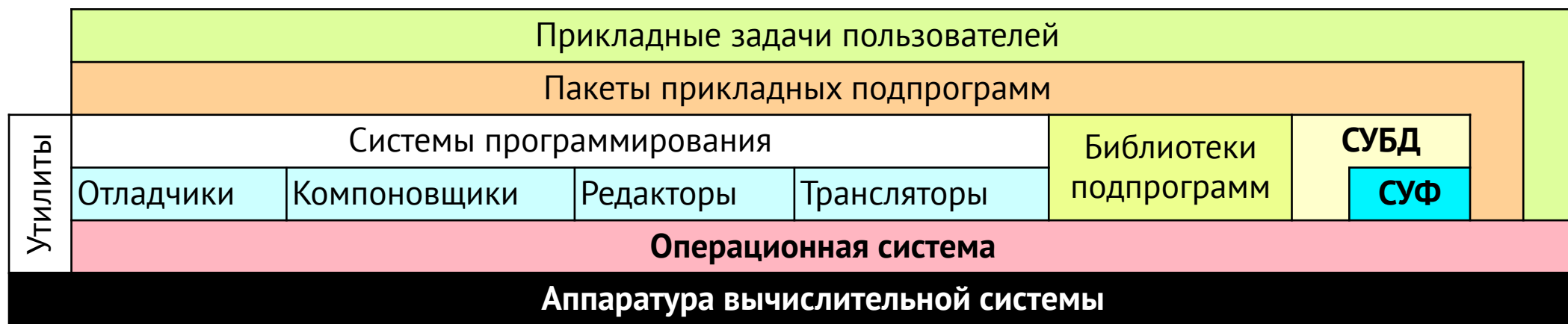
Внутренний уровень — это физическое представление базы данных в компьютере.

Уровень описывает, как информация хранится в БД, в частности, описывает физическую реализацию. Уровень предназначен для достижения оптимальной производительности и обеспечения экономного распределения дискового пространства.

На этом уровне содержится описание структур данных и описание организации файлов базы данных, а также осуществляется взаимодействие СУБД с методами доступа операционной системы для работы с данными.

Уровень содержит информацию о распределении дискового пространства, описание подробностей хранимых записей (реальная длина — байты), сведения о размещении записей, сжатии и шифровании данных.





### Физический уровень

Ниже внутреннего уровня находится физический уровень, который контролируется операционной системой, в том числе и под руководством СУБД, причем разделение функций ОС и СУБД варьируется от системы к системе.

Физический уровень использует только известные операционной системе элементы.

## Схема базы данных (резюме)

Общее описание базы данных называется схемой базы данных.

Для каждого уровня архитектуры ANSI-SPARC существуют свои схемы.

На внешнем уровне имеется несколько внешних схем или подсхем, которые обеспечивают различные представления данных для разных пользователей.

Для концептуального и внешнего уровня имеются соответственно *концептуальная* и *внешняя* схемы, которые для каждой БД существуют в единственном экземпляре.

**Концептуальная схема** описывает все элементы данных, связи между ними, ограничения целостности и т.п.

**Внутренняя схема** описывает определения хранимых записей, методов представления, описания полей данных, индексов и т.п.

СУБД отвечает за установление соответствия между этими схемами (проверка их непротиворечивости — например, чтобы можно было каждую внешнюю схему вывести на основе концептуальной схемы (на основе данных внутренней схемы), проверить соответствие имен и т.п.).

Концептуальная схема связана с внутренней посредством концептуально-внутреннего отображения (для поиска фактической записи на физическом устройстве хранения, которая соответствует логической записи в концептуальной схеме с учетом ограничений).

Каждая внешняя схема связана с концептуальной схемой с помощью внешне-концептуального отображения (отображение имен пользовательского представления на соответствующую часть концептуальной схемы). Например, пользователь хочет получить данные из БД, при этом его запрос преобразуется к виду принятому на концептуальном уровне (отображение внешний-концептуальный, например, изменение имен полей для получения логического описания), затем логическое описание отображается на внутренний уровень (отображение концептуальный-внутренний) для доступа к реальным данным, затем производится обратный процесс.

Следует различать описание БД и саму БД:

описание БД – схема базы данных, создается в процессе проектирования (редко);

сама БД (детализация) – информация содержащаяся в таблицах может меняться часто).

Совокупность данных, хранящихся в БД на определенный момент времени называется *состоянием* БД (состояний может быть различное множество).

## Развитие

На концепции трех схем основана усовершенствованная методология информационного моделирования IDEF1X.

Еще одна – Zachman Framework (1987). В этой структуре модель с тремя схемами превратилась в пирог из шести слоев.

### Структура Захмана

Контекстуальный перечень (перечень важных сущностей)
Контекстуальная модель (сущности и связи)
Логическая модель (атрибуты и идентификаторы)
Физическая модель (ключи, требования к хранению)
Язык определения данных (листинг конфигурации)
Экземпляры данных (хранилище операций)

## Независимость от данных

Независимость от данных — основополагающий принцип построения СУБД. В соответствии с этим принципом, в системе должны поддерживаться отдельные представления данных для пользователя («логическое представление») и для системных механизмов среды хранения БД («физическое представление»).

Такое разделение избавляет пользователя от необходимости знания принятого способа хранения БД и позволяет динамически в процессе эксплуатации системы оптимизировать способ хранения БД для обеспечения более высокой производительности системы и (или) более рационального использования ресурсов памяти.

Различают два типа независимости от данных:

**логическая независимость от данных** — защищенность внешних схем от изменений, вносимых в концептуальную схему — добавление и удаление новых сущностей, атрибутов и связей должны осуществляться без изменений уже существующих внешних схем или изменения прикладных программ;

**физическая независимость от данных** — защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему (изменения в структурах хранения, модификация индексов и т.п. должны осуществляться без изменения концептуальной и внешней схем, пользователи могут заметить изменения только по изменению производительности).

# Система управления базами данных

**СУБД (DBMS)** — программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать БД, а также осуществлять к ней контролируемый доступ.

Фактически это прослойка между БД и пользователем (прикладной программой) для скрывания особенностей хранения и управления данными (абстрагирование).

## Функции СУБД

Для СУБД характерно наличие следующих функций и служб (сервисов) (первые восемь — Кодд, 1982):

1) СУБД должна предоставлять пользователям возможность сохранять, изменять и обновлять данные в БД. Это основная функция СУБД.

Способ реализации этой функции должен скрывать от пользователя детали физической реализации системы (абстрагирование).

2) СУБД должна иметь доступный конечным пользователям каталог, в котором хранится описание элементов данных — *системный каталог* — хранилище информации, описывающей данные БД (метаданные):

- имена, типы и размеры элементов данных;
- имена связей;
- накладываемые на данные ограничения поддержки целостности;
- имена санкционированных пользователей;
- описание схем архитектуры БД;
- статистические данные (счетчики событий);

### **Зачем нужен системный каталог:**

- централизованный контроль доступа к данным;
- определение смысла данных для понимания их предназначения (семантика);
- упрощается определение владельца данных или прав доступа к ним;
- облегчается протоколирование изменений БД;
- последствия изменений могут быть определены до их внесения в БД (усиление безопасности и целостности данных);

3) СУБД должна иметь механизм, который гарантирует выполнение либо всех операций обновления данной транзакции, либо ни одной из них.

Транзакция – набор действий выполняемых пользователем (прикладной программой) с целью доступа или изменения БД (при этом в БД вносится сразу несколько изменений).

Если во время внесения изменений происходит сбой, например, вносимые данные вызывают нарушение целостности данных, то все изменения должны быть отменены (возвращение в непротиворечивое состояние БД).

4) СУБД должна иметь механизм, который гарантирует корректное обновление БД при параллельном выполнении операций обновления многими пользователями (реализуется в рамках поддержки транзакций).

5) СУБД должна предоставлять средства восстановления БД на случай ее повреждения или разрушения (реализуется в рамках поддержки транзакций).

6) СУБД должна иметь механизм, гарантирующий возможность доступа к БД только санкционированных пользователей. Соккрытие части ненужных для пользователя данных и защита от любого несанкционированного доступа.

7) СУБД должна обладать способностью к интеграции с коммуникационным программным обеспечением.

8) СУБД должна обладать инструментами контроля за тем, чтобы данные и их изменения соответствовали заданным правилам. Это называется *целостностью базы данных*, выражающееся в корректности и непротиворечивости хранимых данных.

Целостность является одним из типов защиты БД и выражается в виде ограничений или правил сохранения непротиворечивости данных при их изменении — если ограничения нарушаются, изменения в БД не вносятся.

9) СУБД должна обладать инструментами поддержки независимости программ от фактической структуры БД.

Эта независимость достигается за счет реализации механизма поддержки представлений, например, в рамках трехуровневой концепции.

Достичь полной логической независимости от данных очень сложно, поскольку СУБД легко адаптируется к добавлению нового объекта (атрибута, связи и т.п.), но не к их удалению.

Некоторые СУБД запрещают вносить изменения в существующие компоненты концептуальной схемы.

10) СУБД должна предоставлять некоторый набор различных вспомогательных служб, предназначенных, в том числе, для эффективного администрирования БД – импорт данных, резервное сохранение, реорганизация индексов, сборка мусора, перераспределение памяти.

# ACID

**Atomicity** – атомарность.

**Consistency** – согласованность, целостность.

**Isolation** – изолированность.

**Durability** – стойкость.

**Атомарность** гарантирует, что никакая транзакция не будет зафиксирована в системе частично.

Будут либо выполнены все ее подоперации, либо не выполнено ни одной. Поскольку на практике невозможно одновременно и атомарно выполнить всю последовательность операций внутри транзакции, вводится понятие «отката» (rollback) – если транзакцию не удастся полностью завершить, результаты всех ее до сих пор произведенных действий будут отменены и система вернется во «внешне исходное» состояние – со стороны будет казаться, что транзакции и не было. Возможно, счетчики, индексы и другие внутренние структуры могут измениться, но, если СУБД разработана без ошибок, это не повлияет на внешнее ее поведение.

**Согласованность** является более широким понятием. Например, в банковской системе может существовать требование равенства суммы, списываемой с одного счета, сумме, зачисляемой на другой. Это бизнес-правило и оно не может быть гарантировано только проверками целостности, его должны соблюсти программисты при написании кода транзакций. Если какая-либо транзакция произведет списание, но не произведет зачисления, то система останется в некорректном состоянии и свойство согласованности будет нарушено.

В ходе выполнения транзакции согласованность не требуется. В вышеприведенном примере списание и зачисление будут, скорее всего, двумя разными подоперациями в рамках одной транзакции и между их выполнением внутри транзакции будет видно несогласованное состояние системы.



При выполнении требования изоляции никаким другим транзакциям эта несогласованность не будет видна. Атомарность гарантирует, что транзакция либо будет полностью завершена, либо ни одна из операций транзакции не будет выполнена. Тем самым эта промежуточная несогласованность является скрытой.

**Изолированность** – во время выполнения транзакции параллельные транзакции не должны оказывать влияния на ее результат. Изолированность – требование дорогое, поэтому во многих реальных БД существуют режимы, не полностью изолирующие транзакцию.

**Стойкость** – независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершенной транзакцией, должны остаться сохраненными после возвращения системы в работу. Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

# Модель «Сущность-связь»

Модель сущность-связь (или ER-модель — Entity-Relationship model) описывает взаимосвязанные вещи, представляющие интерес в конкретной области знаний.

Базовая модель ER состоит из типов сущностей (которые классифицируют эти самые важные вещи) и определяет отношения (связи), которые могут существовать между сущностями (экземплярами этих типов сущностей).

В разработке программного обеспечения модель ER обычно формируется для представления вещей, о которых необходимо помнить при выполнении бизнес-процессов. В этом качестве ER-модель становится абстрактной моделью данных, которая определяет структуру данных или информации, которая может быть реализована в базе данных, обычно реляционной.

В процессе проектирования баз данных схема, созданная на основе ER-модели, преобразуется в конкретную схему базы данных на основе выбранной модели данных — реляционной, объектной, иерархической, сетевой, ...

**ER-модель представляет собой формальную конструкцию, которая сама по себе не предписывает никаких графических средств её визуализации и может быть просто словесным описанием в виде текста.**

В качестве ``стандартной графической нотации'', с помощью которой можно визуализировать ER-модель, используется *диаграмма «сущность-связь»* (Entity-Relationship diagram, ERD, ER-диаграмма).

Понятия «ER-модель» и «ER-диаграмма» обычно не различают, однако для визуализации ER-моделей могут быть использованы и другие графические нотации, либо вместо графического представления может использоваться текстовое описание.

Модель была разработана Питером Ченом для проектирования базы данных и опубликована в статье 1976 года с вариантами согласно данной идеи, существовавшими ранее. Он же предложил и самую популярную графическую нотацию для ER-модели.

## ER-диаграмма

Схема «сущность-связь» (также ERD или ER-диаграмма) — это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы.

ER-диаграммы чаще всего применяются для проектирования и отладки реляционных баз данных в сфере образования, исследования и разработки программного обеспечения и информационных систем для бизнеса.

ER-диаграммы (или ER-модели) полагаются на стандартный набор символов, включая прямоугольники, ромбы, овалы и соединительные линии, для отображения сущностей, их атрибутов и связей.

Эти диаграммы устроены по тому же принципу, что и грамматические структуры — сущности выполняют роль существительных, а связи — глаголов.

**Множество** — набор, совокупность каких-либо (вообще говоря любых) объектов — элементов этого множества.

Два множества равны тогда и только тогда, когда содержат в точности одинаковые элементы.

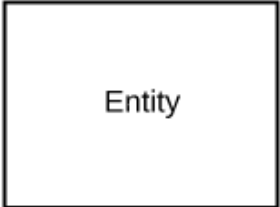
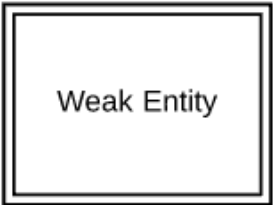
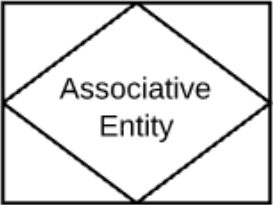
### Отношения между элементами множества

N:1 каждый элемент множества **R** является образом хотя бы одного элемента множества **L**.

1:1

## Символы ERD-сущностей


Под понятием «сущности» подразумеваются объекты или понятия, несущие важную информацию. С точки зрения грамматики, они, как правило, обозначаются существительными, например, «товар», «клиент», «заведение» или «промоакция». Ниже представлены три наиболее распространенных типа сущностей, используемых в ER-диаграммах.

Символ	Название	Описание
	Независимая (сильная) сущность	Не зависит от других сущностей и часто называется «родительской», так как у нее в подчинении обычно находятся слабые сущности. Независимые сущности сопровождаются первичным ключом, который позволяет идентифицировать каждый экземпляр сущности.
	Зависимая (слабая) сущность	Сущность, которая зависит от сущности другого типа. Не сопровождается первичным ключом и не имеет значения в схеме без своей родительской сущности.
	Ассоциативная сущность	Соединяет экземпляры сущностей разных типов. Также содержит атрибуты, характерные для связей между этими сущностями.

# Символы ERD-связей

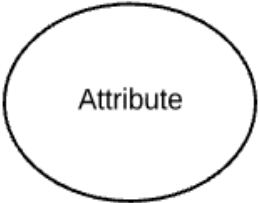

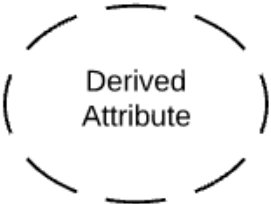
Связи используются в схемах «сущность-связь» для обозначения взаимодействия между **двумя сущностями**.

Грамматически связи, как правило, выражаются глаголами, например, «назначить», «закрепить», «отследить», и несут полезную информацию, которую невозможно получить, опираясь только на типы сущностей.

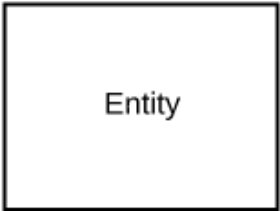
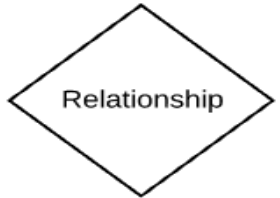

Символ	Название	Описание
	Связь	Отношение между сущностями
	Слабая связь	Связь между зависимой сущностью и ее «хозяином».

## Символы ERD-атрибутов

ERD-атрибуты характеризуют сущности, позволяя пользователям лучше разобраться в устройстве базы данных. Атрибуты содержат информацию о сущностях, выделенных в концептуальной ER-диаграмме.

Символ	Название	Описание
	Атрибут	Характеризует сущность, а также отношения между двумя или более элементами (даты зачисления/увольнения)
	Многозначный атрибут	Атрибут, которому может быть присвоено несколько значений (дни работы заведения).
	Производный атрибут	Атрибут, чье значение можно вычислить, опираясь на значения связанных с ним атрибутов.

## Для проектирования табличных БД достаточно

Символ	Название	Описание
	Сущность	Сопровождаются первичным ключом, который позволяет идентифицировать каждый экземпляр сущности.
	Связь	Отношение между сущностями
	Атрибут	Характеризует сущность, а также отношения между двумя или более элементами (даты зачисления/увольнения)

## Обозначение связей на ER-диаграммах

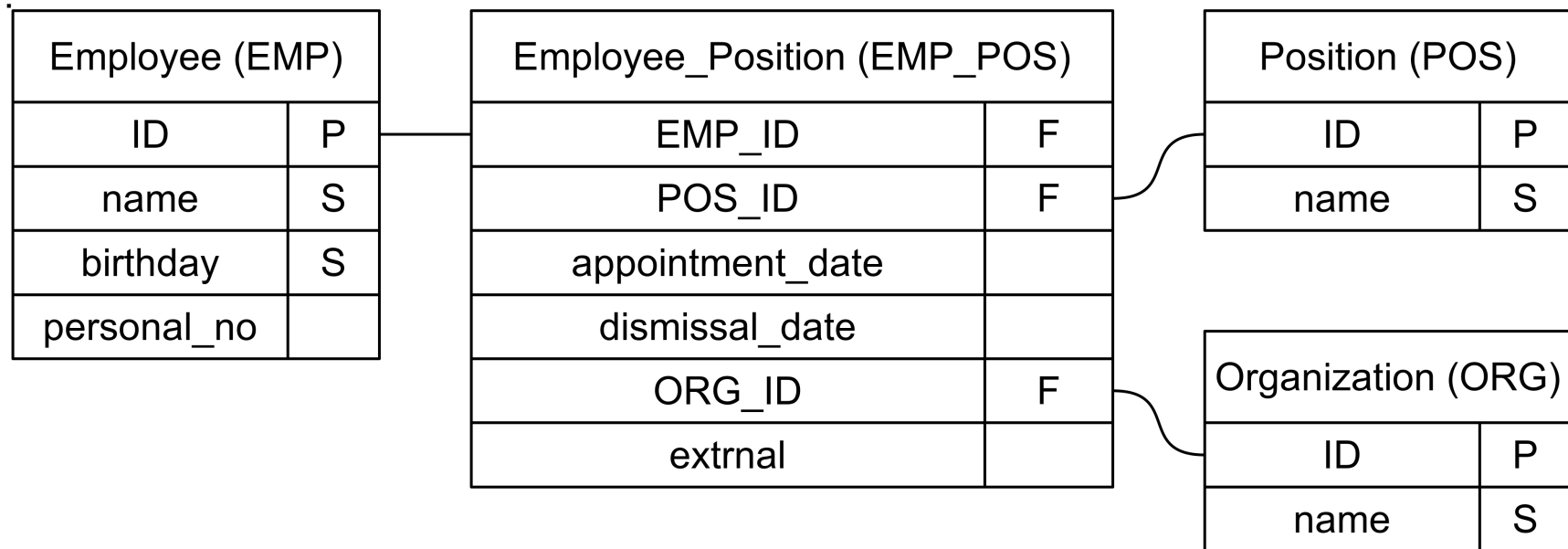
- 1:1 **Каждый** экземпляр сущности L связан **только с одним** экземпляром сущности R и наоборот – **каждый** экземпляр сущности R связан **только с одним** экземпляром сущности L (информация, обязательная к учету для всех работников).
- 1/0:1 **Некоторые** экземпляры сущности L могут быть связаны **только с одним** из экземпляров сущности R, при этом **каждый** экземпляр сущности R связан с только одним экземпляром сущности L (L – дополнительная информация о работнике R).
- N:1 **Каждый** экземпляр сущности L может быть связан **только с одним** экземпляром сущности R, **каждый** экземпляр сущности R может быть связан **с одним или несколькими** экземплярами сущности L. (Авто может иметь только один цвет, один цвет могут иметь несколько авто).
- N:1/0 **Каждый** экземпляр сущности L может быть связан **только с одним** экземпляром сущности R, экземпляр сущности R может быть связан **с одним или несколькими** экземплярами сущности L или не быть связанным ни с одним. («LookUp Table» или «Справочник» – авто может иметь только один цвет, один цвет могут иметь несколько авто).
- N:M **Каждый** экземпляр сущности L может быть связан **с одним или несколькими** (M) экземплярами сущности R и наоборот – **каждый** экземпляр сущности R может быть связан **с одним или несколькими** (N) экземплярами сущности L.



## Символы физических ER-диаграмм

**Физическая модель данных** — самый детальный уровень ER-схем, где представлен процесс добавления информации в базу данных. Физические модели «сущность-связь» отображают всю структуру таблицы, включая названия столбцов, типы данных, ограничения столбцов, первичные и внешние ключи, а также отношения между таблицами.

Таблицы представляют собой еще один способ отображения сущностей.



Ключевые составляющие таблиц «сущность-связь»

**Поля**

**Ключи**

Ключи применяются с целью связывания между собой разных таблиц в базе данных.

## **Первичные ключи**

Первичный ключ — это атрибут или сочетание атрибутов, идентифицирующих один конкретный экземпляр сущности.

## **Внешние ключи (foreign keys )**

Внешний ключ создается каждый раз, когда атрибут привязывается к сущности посредством единичной или множественной связи.

## **Типы**

Под типом подразумевается тип данных в соответствующем поле таблицы.

Однако это также может быть и тип сущности, то есть описание ее составляющих.

Например, у сущности «книга» могут быть следующие типы:

- автор;
- название;
- дата публикации;
- издательство.

## Нотация ER-диаграмм







Нотаций много (Бахмана, ОМТ, IDEF, UML), однако наиболее интуитивной является нотация Crow's Foot («вороньи лапки»).

### Кардинальность и ординальность

Под *кардинальностью* подразумевается максимальное число связей, которое может быть установлено между экземплярами разных сущностей.

*Ординальность*, в свою очередь, указывает минимальное количество связей между экземплярами двух сущностей.

Кардинальность и ординальность отображаются на соединительных линиях согласно выбранному формату нотации.

	One
	Many
	One (and only one)
	Zero or one
	One or many
	Zero or many

# Установка и первичная настройка postgres

PostgreSQL — это продвинутая система управления объектно-реляционными базами данных. Базовый пакет postgresql Сервер PostgreSQL можно найти в подпакете postgresql-server.

Пакет	Описание
postgresql	содержит клиентские программы, необходимые для доступа к серверу СУБД PostgreSQL, а также документацию в формате HTML для всей системы. Эти клиентские программы могут располагаться на том же компьютере, что и сервер PostgreSQL, или на удаленном компьютере, который обращается к серверу PostgreSQL через сетевое соединение.
postgresql-docs	psql – консольный клиент. HTML-документация иногда бывает отдельным пакетом
postgresql-server	ПО сервера.

## Установка на локальной системе

```
# dnf postgresql postgresql-docs postgresql-server
```

В процессе установки будет создан пользователь postgres, который является хозяином каталога /usr/local/pgsql/data, в котором будет располагаться кластер базы данных

## Первичная настройка

```
$ su postgres -  
$ /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data  
$ /usr/local/pgsql/bin/pg_ctl -D /usr/local/pgsql/data -l logfile start  
$ /usr/local/pgsql/bin/createdb test  
$ /usr/local/pgsql/bin/psql test
```

pg\_ctl – утилита для управления сервером.

## Запуск сервера в системе с systemd

```
$ sudo systemctl enable postgresql  
$ sudo systemctl start postgresql
```

## Проверка работы

```
$ sudo -u postgres psql -c "SELECT version();" 
```

## Создание пользователя и базы данных

```
$ sudo -i -u postgres  
$ psql
```

```
$ sudo -u postgres createuser -D -A -P user  
$ sudo -u postgres createdb -O user user_db
```