

Лекция 2.

Первые СУБД. Сетевая и иерархическая модели данных. Навигационный подход.

Студенты проходят предметы по плану и сдают экзамены

Списки групп
(Word)

Учебные планы
(Excel)

Итоги сессии
(PDF)

Набор файлов и база данных – в чем отличие?

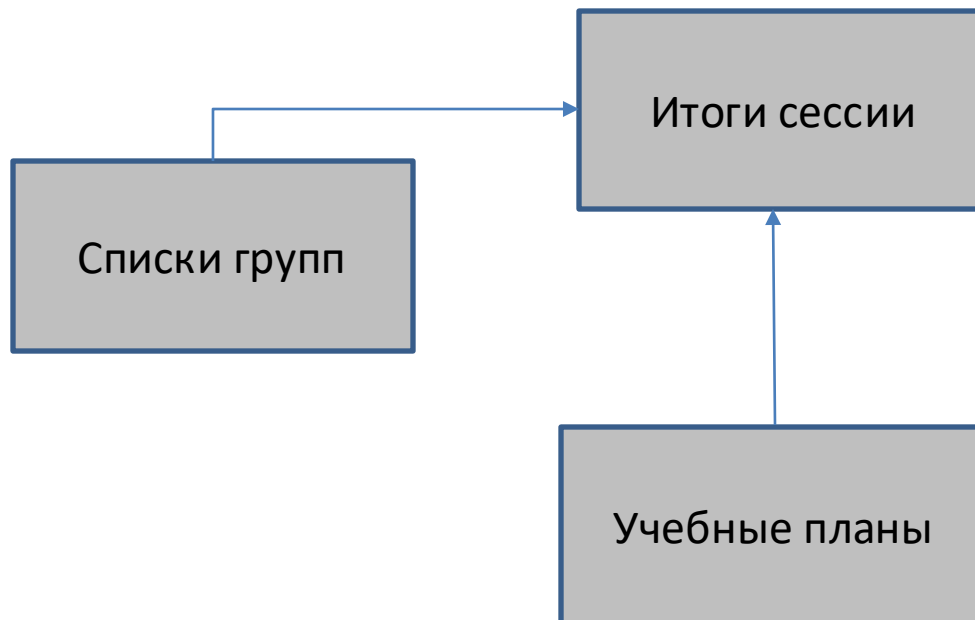
Студенты проходят предметы по плану и сдают экзамены

Списки групп
(Word)

Учебные планы
(Excel)

Итоги сессии
(PDF)

Набор файлов и база данных – в чем отличие?



- 1. Структура данных**
Сущности и их атрибуты
- 2. Связи между данными**
Как сущности связаны друг с другом

Студенты проходят предметы по плану и сдают экзамены



Структура данных + Связи между данными

Эволюция моделей данных

Связь - ключевое слово, отличающее базу данных от простого файла или набора файлов. Как представить эту связь в базе данных?

В разное время применялись различные подходы (**модели данных**).

Годы	Используемые модели данных
1960-е	1. Файлы без связей 2. Сетевая и иерархическая (<i>навигационный подход, императивное программирование</i>)
1970-е	1. Сетевая и иерархическая модель 2. Реляционная модель (<i>математическая база, декларативное программирование</i>)
1980-е	1. Реляционная модель 2. Сетевая и иерархическая модели
1990-е	Реляционная модель
2000-е	1. Реляционная модель 2. NoSQL

Отдельные файлы данных и SOVOL

Проблема обработки файлов с данными

Данные хранятся в структурированных файлах, для работы с ними пишутся приложения на языках программирования.

К концу 1950-х годов каждый производитель компьютеров использовал свои языки и форматы:

- Программы привязаны к железу
- Огромные затраты на разработку, сопровождение и обучение

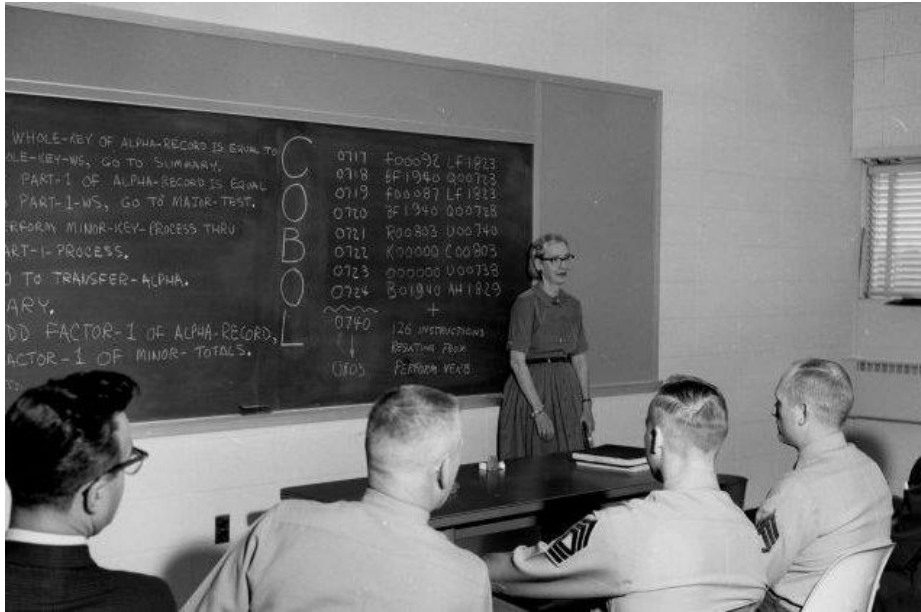
COBOL – стандартизированный язык для бизнес-приложений

По инициативе Министерства обороны США в 1959 году создается комитет CODASYL (Conference on Data Systems Languages) для разработки единого языка программирования, который:

- Будет работать на разных компьютерах
- Будет понятен не только программистам, но и менеджерам
- Сфокусирован на обработке данных (финансы, логистика, отчеты)

Минобороны обеспечило массовое использование COBOL.

COBOL – первый язык для бизнес-приложений



Грейс Хоппер –
технический советник в
проекте.

Ее язык Flow-Matic был взят
за основу нового языка
(синтаксис и философия)

"Программы должны писаться так, чтобы их мог читать бухгалтер"

COBOL = **C**ommon **B**usiness **O**riented **L**anguage

- Структуры данных (записи)
- Структурированные файлы

Поддержка структурированных файлов

COBOL предоставляет **встроенные средства описания структуры файлов и операций над ними** — без необходимости вызывать внешние API. Это одна из его главных фишек.

Всё делится на 3 основные секции в программе:

1. **FILE SECTION** — описание структуры файла.
2. **WORKING-STORAGE SECTION** — буферы для чтения/записи.
3. **PROCEDURE DIVISION** — операции: OPEN, READ, WRITE, REWRITE, CLOSE.

Поддержка структурированных файлов

Поддерживаемые типы файлов по способу доступа:

- SEQUENTIAL - последовательный
- INDEXED - прямой по уникальному индексу
- RELATIVE - по номеру записи фиксированного размера

INDEXED и RELATIVE - с 1968 года.

Индексированные файлы - прообраз таблиц с первичными ключами. COBOL становится встроенной мини-СУБД

Типы файлов по способу доступа

1. SEQUENTIAL (последовательный файл)

- Записи хранятся одна за другой.
- Чтение/запись — **только последовательно**, с начала до конца.
- Аналог: текстовый файл или лог.

```
cobol
```

```
1  SELECT PAYROLL-FILE ASSIGN TO "PAYROLL.DAT"  
2      ORGANIZATION IS SEQUENTIAL  
3      ACCESS MODE IS SEQUENTIAL  
4      FILE STATUS IS WS-FILE-STATUS_
```

→ Используется для отчётов, выгрузок, логов.

Типы файлов по способу доступа

2. INDEXED (индексированный файл)

- Записи имеют **уникальный ключ** (например, ID сотрудника).
- Доступ: **последовательный или прямой (по ключу)**.
- Физически: данные + индексный файл (как B-дерево).

cobol

```
1 SELECT EMPLOYEE-FILE ASSIGN TO "EMPLOYEE.DAT"  
2     ORGANIZATION IS INDEXED  
3     ACCESS MODE IS DYNAMIC  
4     RECORD KEY IS EMP-ID  
5     FILE STATUS IS WS-FILE-STATUS_
```

→ Можно:

- `READ EMPLOYEE-FILE KEY IS 123` — прямой доступ,
- `READ NEXT` — последовательный обход.

💡 Это прообраз реляционных таблиц с первичным ключом.

Типы файлов по способу доступа

3. RELATIVE (относительный файл)

- Записи доступны по номеру записи (RBA — Relative Byte Address).
- Фиксированная длина записи.
- Быстрый прямой доступ: “дай мне 5-ю запись”.

cobol

```
1  SELECT INVENTORY-FILE ASSIGN TO "INVENTORY.DAT"  
2      ORGANIZATION IS RELATIVE  
3      ACCESS MODE IS RANDOM  
4      RELATIVE KEY IS WS-REC-NUM  
5      FILE STATUS IS WS-FILE-STATUS_
```

→ Использовался для справочников, матриц, массивов.

cobol

```
1 IDENTIFICATION DIVISION_
2 PROGRAM-ID_ READ-EMP_
3
4 ENVIRONMENT DIVISION_
5 INPUT-OUTPUT SECTION_
6 FILE-CONTROL_
7     SELECT EMP-FILE ASSIGN TO "EMP.DAT"
8     ORGANIZATION IS INDEXED
9     ACCESS MODE IS SEQUENTIAL
10    RECORD KEY IS EMP-ID_
11
12 DATA DIVISION_
13 FILE SECTION_
14 FD EMP-FILE_
15 01 EMP-RECORD_
16    05 EMP-ID    PIC 9(5)_
17    05 EMP-NAME  PIC X(30)_
18
19 WORKING-STORAGE SECTION_
20 01 WS-EOF      PIC X VALUE 'N'_
21
22 PROCEDURE DIVISION_
23     OPEN INPUT EMP-FILE
24     PERFORM UNTIL WS-EOF = 'Y'
25         READ EMP-FILE
26             AT END MOVE 'Y' TO WS-EOF
27             NOT AT END DISPLAY EMP-NAME
28     END-READ
29 END-PERFORM
30 CLOSE EMP-FILE
31 STOP RUN_
```

Чтение структурированного файла (COBOL)

COBOL – первый язык для бизнес-приложений

COBOL - первая попытка стандартизировать структурированное хранение и доступ к данным на уровне языка программирования.

- Индексированные файлы - прообраз таблиц с первичными ключами.
- Файловые операции в COBOL - прообраз транзакционных операций в СУБД
- Многие современные форматы файлов - наследники COBOL-файлов

COBOL стал доминирующим языком для бизнес-приложений на несколько десятилетий. Жив до сих пор.

**СУБД для решения задач,
возникающих при работе с
данными**

Задачи приложения/программиста при работе с файлами

- **Управление данными.** Чтение/запись файлов, разбор структуры.
- **Целостность данных.** Контроль ограничений на данные.
- **Многопользовательский доступ.** Реализация блокировок, разрешение конфликтов.
- **Производительность.** Поиск, сортировка, фильтрация данных с помощью различных алгоритмов.
- **Масштабируемость.** С ростом объёма данных производительность падает, сложно масштабировать.
- **Надежность и восстановление.** Создание резервных копий, логирование изменений в данных.
- **Язык запросов.** Написание кода для фильтрации, агрегации, соединения данных.
- **Безопасность.** Управление доступом к файлу на уровне ОС или собственной логики.

Делегирование СУБД задач по управлению данными

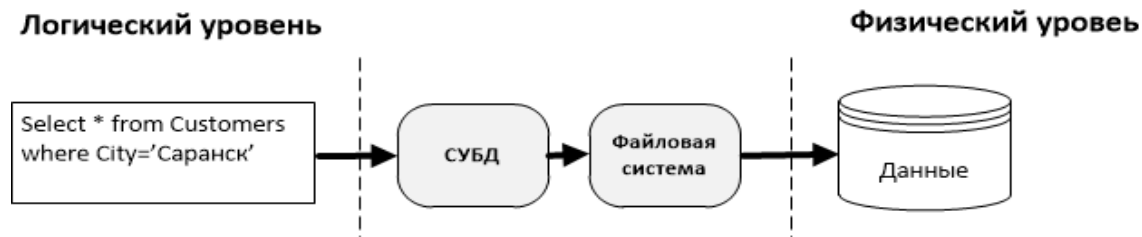
СУБД берёт на себя огромный пласт инфраструктурной логики, которая при работе с файлами ложится на плечи разработчика.

Это позволяет прикладному приложению сосредоточиться на бизнес-логике, а не на том, как хранить, искать и защищать данные.

Работа с файлом



Работа с базой данных



СУБД или обычные файлы?

Файлы (СУБД будет избыточной и медленной)

- Данные статичны или меняются редко.
- Нет параллельного доступа.
- Нет сложных запросов.
- Небольшой объем (до десятков тысяч записей).
- Простота важнее надежности.

- Конфигурационные файлы (config.json, .env, settings.yaml, nginx.conf)
- Статические справочники (список стран, коды валют)
- Промежуточные/временные данные (CSV, бинарные форматы)
- Файлы данных в научных/инженерных расчетах (Matlab, NumPy)
 - массивы чисел эффективнее хранить в бинарном виде

СУБД

- Данные меняются часто и конкурентно.
 - Есть связи между сущностями.
 - Нужны транзакции, безопасность, восстановление.
 - Объем большой, нужны индексы и оптимизация.
 - Требуется SQL или сложная аналитика.
-
- Веб-приложения с пользователями и динамическими данными
 - Аналитические дашборды, системы мониторинга, ERP
 - Системы бронирования, складской учет, финансовые платформы
 - Медицинские системы, банковские приложения, госреестры

Первая СУБД в мире

Первая СУБД IDS (Integrated Data Store)

Задача: написать систему управления производственными запасами и операциями для крупной компании

Цель: создать единое хранилище данных, к которому могут обращаться разные приложения одновременно.

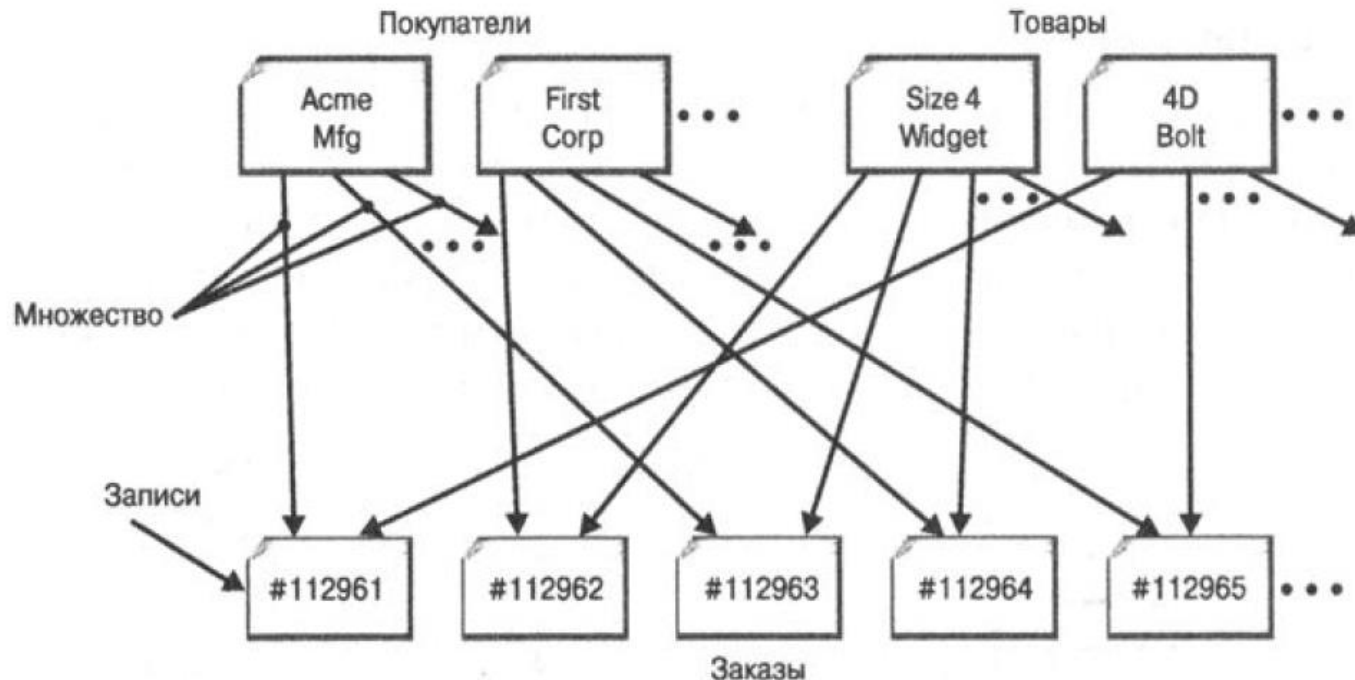
Чарльз Бахман (1924-2017)

- Работал в General Electric
- В 1963-1964 годах создал СУБД Integrated Data Store (IDS) для мэйнфрейма GE-235 (ОЗУ 20 килобайт)
- Получил за нее премию Тьюринга в 1973 году.



1. Сетевая модель данных

- Сущности (записи) связаны между собой произвольными связями "многие-ко-многим".
- Структура данных напоминает **ориентированный граф**.
- Связи реализованы через **указатели (pointers)** - физические адреса в памяти или на диске.



Сетевая модель данных - ключевые понятия

- **Запись (Record)**

Имеет внутреннюю структуру: поля-атрибуты, типы данных

Отделы (DEPARTMENT): DEPT-ID, DEPT-NAME

Сотрудники (EMPLOYEE): EMP-ID, EMP-NAME, EMP-POSITION

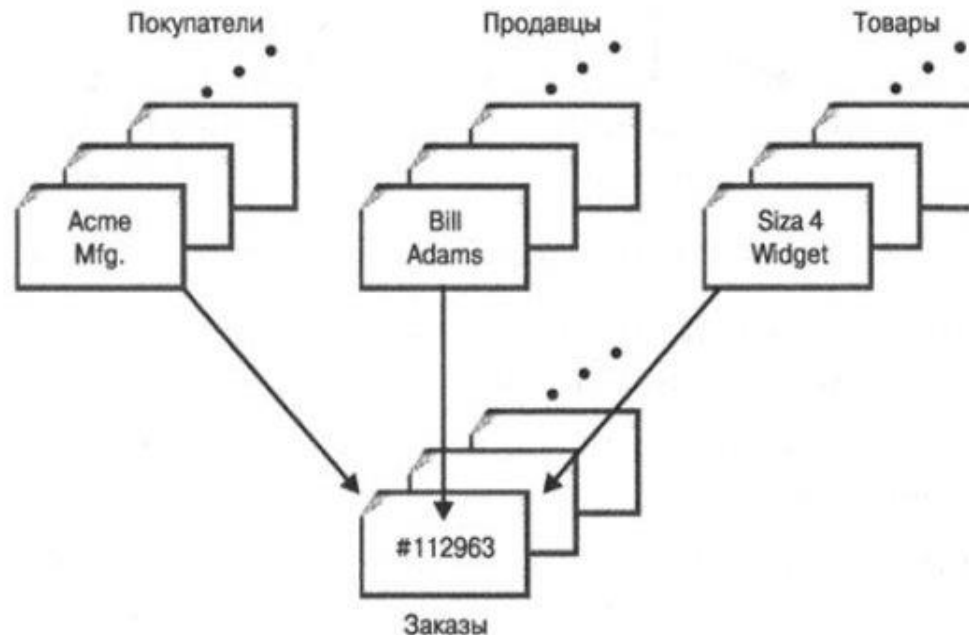
Проекты (PROJECT): PROJ-ID, PROJ-TITLE

Сетевая модель данных - ключевые понятия

• Набор (Set)

Это логическая связь "владелец (owner)-члены отношения (members)".

- Один владелец - запись одного типа
- Множество членов - записи другого или того же типа
- Запись может быть членом нескольких наборов - получаем связь "многие-ко-многим"



Сетевая модель данных - ключевые понятия

- **Набор (Set)**

Это логическая связь "владелец (owner)-члены отношения (members)".

- Один владелец - запись одного типа
- Множество членов - записи другого или того же типа
- Запись может быть членом нескольких наборов - получаем связь "многие-ко-многим"

Набор DEPT-EMP: владелец DEPARTMENT, члены - EMPLOYEE

Набор PROJ-EMP: владелец PROJECT, члены - EMPLOYEE

Один сотрудник может работать в отделе и одновременно участвовать в проектах.

Сетевая модель данных - ключевые понятия

- **Указатели (Pointers).**

Связи между записями через физические адреса в памяти или на диске. Позволяют быстро перемещаться по связям.

Типы указателей:

- Next - на следующую запись в наборе
- Prior – на предыдущую запись в наборе
- Owner – на владельца набора

Сетевая модель данных - ключевые понятия

- **Схема и подсхема (Schema/subschema).**

Schema - полная структура БД (все записи, наборы, связи)

Subschema - "вид" БД для конкретного приложения

2. Data Manipulation Language (DML)

- Встроенный язык манипулирования данными, который можно было использовать внутри приложения (например, на COBOL)
- Навигация по структуре данных: перейти к следующему, найти владельца, и т.п.

cobol

```
1 FIND OWNER WITHIN DEPT-EMP WHERE DEPT-NAME = "IT"
2 FIND FIRST MEMBER WITHIN DEPT-EMP → нашли первого сотрудника отдела
3 DO WHILE NOT END OF SET
4     PRINT EMP-NAME
5     FIND NEXT MEMBER WITHIN DEPT-EMP → переходим к следующему
6 END DO
```

3. Разделение кода и данных

- Программы (COBOL + DML) — отдельно.
- Структуры данных и указатели — централизованно в базе.
- Системные таблицы (до их времени) — хранили метаданные о наборах, типах записей, расположении.

Сетевая модель данных - первая попытка отделить логическую структуру от физической для независимости приложения от хранилища.

4. Поддержка многопользовательского режима

IDS позволяла **нескольким программам (задам)** одновременно работать с одной и той же **базой данных** — например, одна добавляет заказ, другая обновляет остатки товара.

 **Механизм: управление блокировками на уровне записей**

- При обращении к записи система **автоматически устанавливала блокировку (lock)**.
- Блокировки были **эксклюзивными (exclusive)** — пока одна программа изменяет запись, другие должны ждать.
- Блокировки снимались после завершения операции или транзакции.

4. Поддержка многопользовательского режима

 Монолитная, но многозадачная архитектура

- IDS работала как часть операционной среды GE 225 (позже — GECOS).
- Поддерживала многозадачность на уровне ОС — несколько задач (процессов) могли вызывать функции IDS одновременно.
- У IDS был единый каталог структур данных (data dictionary), к которому все задачи обращались через системные вызовы.

5. Поддержка транзакций

- IDS обеспечивала логическую целостность операций через последовательное выполнение команд DML и контроль указателей.
- IDS вела журнал операций (log) — записывала, какие изменения были сделаны.



6. Контроль целостности связей

- При удалении владельца (например, CUSTOMER) система **не позволяла оставить «сиротские» записи** (например, ORDER без владельца), если это нарушало структуру набора.
- Программист должен был **вручную отключить все дочерние записи** перед удалением владельца — иначе возникала ошибка.

“Это — зачатки каскадного удаления и ограничений внешних ключей, но реализованные на уровне DML, а не декларативно.”

Появление IDS - революция в работе с данными

До IDS:

- Каждое приложение работало со своими файлами.
- Не было централизованного управления данными.
- Изменение структуры = переписывание всех программ.

После IDS:

- Появилось понятие **базы данных как общего ресурса**.
- Введены **структуры данных, связи, языки манипулирования, транзакции**.
- Началась эра **систем управления базами данных**.

Сетевая модель данных CODASYL DBTG

Распространение и влияние IDS

- IDS была **внутренней системой GE**, но её идеи быстро распространились.
- В 1967–1969 гг. **CODASYL** (комитет, создавший COBOL) сформировал **DBTG (Data Base Task Group)** — чтобы стандартизировать сетевые СУБД на основе IDS.
- В 1971 г. вышел **стандарт CODASYL DBTG**, ставший основой для:
 - **IDMS** (Cullinane)
 - **DMS-1100** (Univac)
 - **IMAGE** (HP)
 - и других сетевых СУБД 1970–80-х.

“💡 Таким образом, **IDS** — это прародитель всех сетевых **СУБД**, а Бахман — их «крёстный отец».”

Стандарт CODASYL DBTG

1. Сетевая модель данных (Network Data Model)

Это главная инновация стандарта. В отличие от иерархической модели (например, IBM IMS), где каждый дочерний элемент может иметь только одного родителя, сетевая модель позволяет одному записью (record) иметь несколько родителей.

“ Преимущество: гибкое представление сложных отношений между сущностями (многие-ко-многим).”

Основные компоненты сетевой модели:

- **Запись (Record)** — единица данных, аналог "строки" в реляционной модели.
- **Связь (Set)** — структура, определяющая отношения между записями:
 - **Один владелец (Owner)** — одна запись (один экземпляр).
 - **Много членов (Members)** — множество записей, связанных с владельцем.
 - **Связь — ориентированный граф**, где записи связаны через указатели.

Стандарт CODASYL DBTG

2. Три уровня архитектуры (ANSI/SPARC-like до его появления!)

CODASYL DBTG ввёл идею трёхуровневой архитектуры БД задолго до ANSI/SPARC (1975):

УРОВЕНЬ	ОПИСАНИЕ
Внешний уровень (External Schema)	Представление данных для конкретного приложения или пользователя. Может
Концептуальный уровень (Conceptual Schema)	Полное описание всей структуры базы данных: все записи, все связи, целостност
Физический уровень (Internal Schema)	Как данные физически хранятся на диске: указатели, порядок записей, методы д

“💡 Это была революция: отделение логики от физического хранения — основа современных СУБД!”

Стандарт CODASYL DBTG

3. Язык манипулирования данными (DML — Data Manipulation Language)

CODASYL DBTG определил императивный язык для работы с данными. Он не был декларативным, как SQL, а требовал программного управления:

- Приложение должно **обходить** структуру данных с помощью указателей.
- Операторы:
 - **FIND** — найти запись по ключу.
 - **GET NEXT** — перейти к следующей записи в наборе.
 - **STORE** , **MODIFY** , **DELETE** — изменять данные.
 - **CONNECT** / **DISCONNECT** — управлять связями.

Пример (упрощённо):

```
cobol
```

```
1 FIND EMPLOYEE WITH NAME = 'Иванов'  
2 CONNECT EMPLOYEE TO DEPARTMENT WITH ID = 10
```

“ ! Это требует знания структуры данных на уровне программиста — низкий уровень абстракции. ”

Стандарт CODASYL DBTG

4. Язык определения схемы (DDL — Data Definition Language)

Определял структуру БД: какие есть записи, какие связи, их типы, ключи.

Пример DDL-описания:

```
codasyl
1 SET EMPLOYEE-OF-DEPT
2   OWNER DEPARTMENT
3   MEMBER EMPLOYEE
4 END-SET
```

Также описывались:

- Первичные ключи (в виде "owner key")
- Порядок обхода (например, по алфавиту)
- Ограничения целостности (например, "не допускать дубликатов")

5. Поддержка транзакций и целостности

Хотя не так развито, как в современных СУБД, CODASYL DBTG уже включал:

- Логику восстановления (реестры изменений)
- Контроль параллелизма (блокировки)
- Целостность ссылок — нельзя удалить владельца, пока есть члены

СУБД IDMS - коммерческая реализация сетевой модели

- Компания Goodrich переписала IDS на язык ISL (Intermediate System Language) и переименовала СУБД в **IDMS** (Integrated Database Management System). После этого IDMS можно было легко переносить на мейнфреймы IBM.
- IDMS приобрела компания Cullinane Джона Калинана - первая успешная фирма по продаже ПО для мейнфреймов (с 1968 года).

Значение сетевой модели CODASYL DBTG

 Цитата из отчёта DBTG (1971):

"«Целью модели является обеспечение независимости приложений от физического представления данных, при сохранении высокой эффективности доступа»."

Эта фраза — фундамент всей современной теории баз данных.

Значение сетевой модели CODASYL DBTG

- Первая попытка стандартизировать СУБД — до этого каждая система была уникальной.
- Заложила основы разделения логики и физики данных.
- Вдохновила многие последующие стандарты (SQL, X/Open, ANSI/SPARC).
- Даже сейчас в некоторых legacy-системах (особенно в банковской сфере, военной индустрии) работают СУБД на основе CODASYL (например, **IDMS** до сих пор используется в крупных банках США!).

Значение сетевой модели CODASYL DBTG

Хотя сегодня сетевая модель не используется в чистом виде, её идеи живы:

- **Графовые базы данных** (Neo4j, Amazon Neptune) — современные наследники сетевой модели.
- **Указатели и индексы** — используются внутри реляционных СУБД для ускорения JOIN'ов.
- **Концепция subschema** — прообраз современных представлений (VIEW) и ролей безопасности.
- **DBTG заложил основы** для будущих стандартов языков определения и манипулирования данными.

Иерархическая модель.

Первая промышленная СУБД

СУБД IMS

- **IBM IMS** (Information Management System). Первая промышленная СУБД, использовавшаяся в масштабах предприятия. 1968 год
- Для автоматизации управления большими объемами данных, связанных с проектированием, производством и логистикой космических аппаратов (NASA, проект Apollo).
- Проектировалась для работы в реальном времени и высокой производительности и отказоустойчивости.

Состоит из двух компонентов:

- IMS DB (Database) - иерархическая СУБД
- IMS TM (Transaction Manager) - система управления транзакциями

Иерархическая модель данных

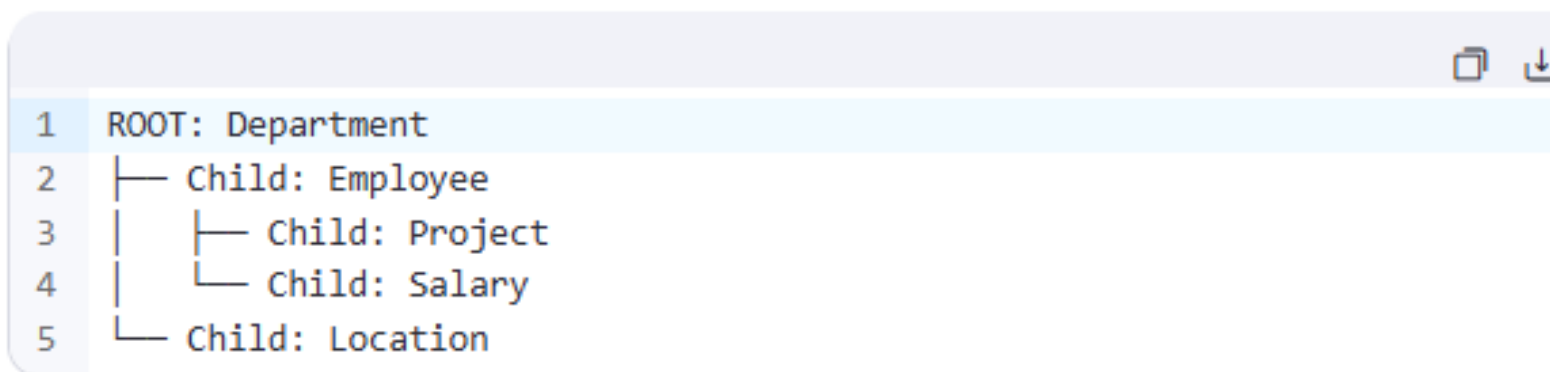
Иерархическая модель — это способ организации данных в виде дерева:

- Есть корневой элемент.
- Каждый элемент (узел) может иметь **несколько потомков**, но **только одного родителя**.
- Связи — только «**один-ко-многим**».

Иерархическая модель данных

Как работает иерархическая модель в IMS?

Данные организованы в иерархические структуры — "сегменты" (segments):



- Каждый сегмент имеет тип и поля.
- Для доступа к данным — **навигационный подход**: от корня вниз по дереву.
- Нет SQL — используются **процедурные вызовы API** (встраивались в COBOL, PL/I и др.).



Иерархическая модель данных

- ✔ Была де-факто стандартизация через IMS от IBM
 - IMS стал промышленным стандартом для иерархических баз данных.
 - Другие производители (Honeywell, Siemens, Bull) создавали свои иерархические СУБД, но совместимость с IMS была главной целью.
 - IBM опубликовала спецификации форматов, DDL, DML, API — и многие их перенимали.

Иерархическая модель данных

Пример структуры IMS (DDL-подобный псевдокод)

```
text 📄 ⬇  
1 DBD NAME=EMPDB  
2   SEGM NAME=DEPARTMENT  PARENT=0      (корень)  
3     FIELD NAME=DEPTID   TYPE=C  SIZE=3  
4     FIELD NAME=DEPTNAME TYPE=C  SIZE=20  
5  
6   SEGM NAME=EMPLOYEE    PARENT=DEPARTMENT  
7     FIELD NAME=EMPID    TYPE=C  SIZE=5  
8     FIELD NAME=EMPNAME  TYPE=C  SIZE=30  
9  
10  SEGM NAME=PROJECT     PARENT=EMPLOYEE  
11    FIELD NAME=PROJID   TYPE=C  SIZE=4  
12    FIELD NAME=PROJNAME TYPE=C  SIZE=25
```

“Это не SQL — это **DBD (Database Descriptor)** — спецификация структуры IMS.”

Типичные операторы для манипулирования данными

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

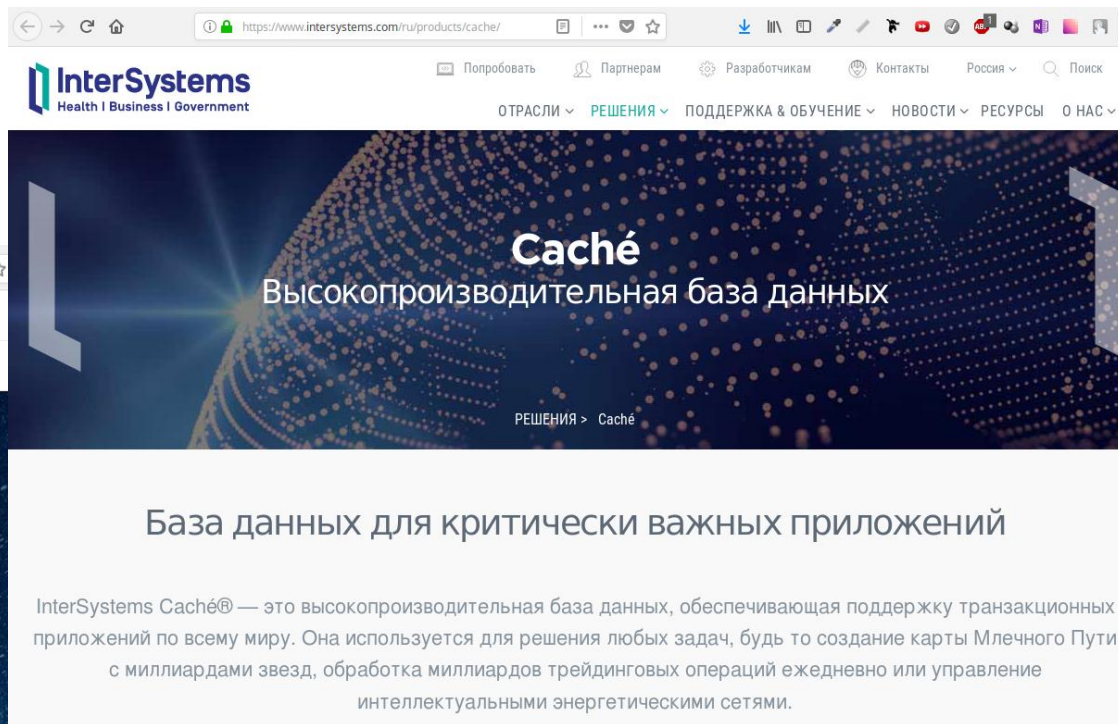
Иерархические БД сегодня

Почему IMS до сих пор актуальна?

Несмотря на возраст, IMS остаётся востребованной благодаря:

- **Надёжности и стабильности** — работает десятилетиями без простоев.
- **Высокой производительности** — обрабатывает миллионы транзакций в день.
- **Интеграции с современными технологиями** — REST API, JSON, XML, Java, облачные решения.
- **Низкой стоимости владения** — для уже работающих систем дешевле поддерживать IMS, чем мигрировать.

Иерархические БД сегодня



InterSystems
Health | Business | Government

Попробовать | Партнерам | Разработчикам | Контакты | Россия | Поиск

ОТРАСЛИ | РЕШЕНИЯ | ПОДДЕРЖКА & ОБУЧЕНИЕ | НОВОСТИ | РЕСУРСЫ | О НАС

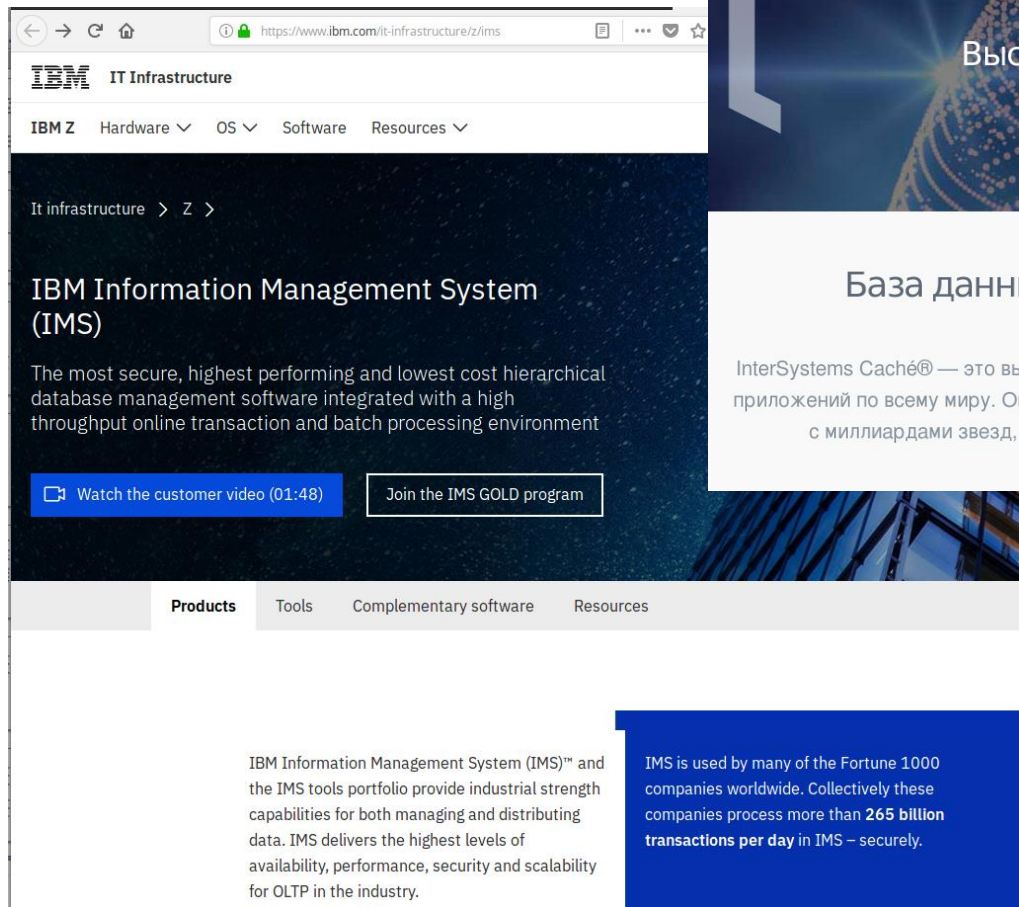
Caché

Высокопроизводительная база данных

РЕШЕНИЯ > Caché

База данных для критически важных приложений

InterSystems Caché® — это высокопроизводительная база данных, обеспечивающая поддержку транзакционных приложений по всему миру. Она используется для решения любых задач, будь то создание карты Млечного Пути с миллиардами звезд, обработка миллиардов трейдинговых операций ежедневно или управление интеллектуальными энергетическими сетями.



IT Infrastructure

IBM Z | Hardware | OS | Software | Resources

It infrastructure > Z >

IBM Information Management System (IMS)

The most secure, highest performing and lowest cost hierarchical database management software integrated with a high throughput online transaction and batch processing environment

[Watch the customer video \(01:48\)](#) [Join the IMS GOLD program](#)

Products | Tools | Complementary software | Resources

IBM Information Management System (IMS)™ and the IMS tools portfolio provide industrial strength capabilities for both managing and distributing data. IMS delivers the highest levels of availability, performance, security and scalability for OLTP in the industry.

IMS is used by many of the Fortune 1000 companies worldwide. Collectively these companies process more than **265 billion transactions per day** in IMS – securely.

Современные аналоги иерархической модели

Хотя чистая иерархическая модель устарела, её идеи используются в:

- XML / JSON — вложенные структуры данных.
- Файловые системы — папки и файлы.
- NoSQL: MongoDB (вложенные документы), Firebase Realtime DB.
- LDAP-каталоги — тоже иерархические.

Современные аналоги иерархической модели

Пример XML (иерархия сотрудников):

```
xml
1 <company>
2   <department name="IT">
3     <employee id="101">
4       <name>Alice</name>
5       <salary>70000</salary>
6       <projects>
7         <project>Website</project>
8         <project>API</project>
9       </projects>
10    </employee>
11   <employee id="102">
12     <name>Bob</name>
13     <salary>65000</salary>
14   </employee>
15 </department>
16 </company>
```

→ Это **дерево**: `company` → `department` → `employee` → `name`, `salary`, `projects` → `project`.

Современные аналоги иерархической модели

Пример JSON (та же структура):

```
json
1 v {
2 v   "company": {
3 v     "department": {
4       "name": "IT",
5 v     "employees": [
6 v       {
7         "id": 101,
8         "name": "Alice",
9         "salary": 70000,
10        "projects": ["Website", "API"]
11       },
12 v      {
13        "id": 102,
14        "name": "Bob",
15        "salary": 65000
16      }
17    ]
18  }
19 }
20 }
```

→ Тоже **дерево**: объекты и массивы вложены друг в друга. Нет ссылок "вбок" — только "вглубь".

Современные примеры иерархических структур

- Конфигурационные файлы: `package.json`, `pom.xml`, `web.xml`, `appsettings.json`
- API-ответы: REST, GraphQL (часто возвращают вложенные объекты)
- Документы: HTML, DOCX, SVG — всё это иерархические структуры
- NoSQL: MongoDB, Firebase — документоориентированные БД, где документ = иерархический объект (JSON/BSON)
- UI-деревья: React/Vue компоненты, DOM — тоже деревья

“🌐 Весь веб — построен на иерархии: HTML → DOM → JSON → компоненты → маршруты.”

Навигационный подход к работе с данными

Навигационный подход в первых СУБД

Навигационный подход — это процедурный, императивный способ доступа к данным, при котором программист вручную «проходит» по структуре данных, следуя связям (указателям) от одной записи к другой, чтобы найти нужную информацию.

Навигационный подход в первых СУБД

1. 🌲 Иерархическая модель — IBM IMS

📌 Структура:

Дерево:

Клиент → Заказы → Позиции → Товары

🔍 Навигация:

💡 Особенности:

- Жёсткая структура: только один родитель.
- Навигация — строго сверху вниз.
- Простота и скорость для предсказуемых путей.

```
cobol
```

```
1 FIND CUSTOMER WHERE NAME = 'ИВАНОВ'
2 IF FOUND
3     FIND FIRST ORDER WITHIN CUST-ORDERS
4     PERFORM UNTIL NO MORE ORDERS
5         DISPLAY ORDER_DATE
6         FIND FIRST ITEM WITHIN ORDER-ITEMS
7         PERFORM UNTIL NO MORE ITEMS
8             DISPLAY ITEM_DESC
9             FIND NEXT ITEM WITHIN ORDER-ITEMS
10        END-PERFORM
11        FIND NEXT ORDER WITHIN CUST-ORDERS
12    END-PERFORM
13 END-IF
```

Навигационный подход в первых СУБД

2. Сетевая модель — IDS / IDMS

Структура:

Сеть:

Клиент → Заказы → Позиции

Товар → Позиции ← та же запись!

Особенности:

- Запись может иметь **нескольких владельцев**.
- Можно **входить в данные с разных сторон**.
- Гибкость, но сложность управления указателями.

Навигация:

cobol

```
1  FIND PRODUCT WHERE PROD_ID = 'P100'  
2  IF FOUND  
3      FIND FIRST ITEM WITHIN PROD-ITEMS  
4      PERFORM UNTIL NO MORE ITEMS  
5          FIND OWNER WITHIN ORDER-ITEMS → переходим к ORDER  
6          FIND OWNER WITHIN CUST-ORDERS → переходим к CUSTOMER  
7          DISPLAY CUSTOMER_NAME, ORDER_DATE  
8          FIND NEXT ITEM WITHIN PROD-ITEMS  
9      END-PERFORM  
10 END-IF
```

Лекция Бахмана "The Programmer as Navigator", 1973

Это — философское обоснование навигационного подхода.

 Основные идеи лекции:

1. **Программист — не писатель запросов, а исследователь структуры данных.**
Он «навигирует» по сети записей, выбирая маршрут, как капитан по карте.
2. **Структура данных — это ландшафт, по которому нужно уметь ходить.**
Связи (указатели) — это дороги. Наборы — это тропы. DML — компас.
3. **Сетевая модель (IDS) позволяет выбирать путь, иерархическая (IMS) — диктует его.**
"In a hierarchy, you can only go down one path. In a network, you choose your path."
4. **Будущее — за абстракцией, но навигация — фундамент.**
Бахман предвидел, что системы станут скрывать сложность — но понимание навигации останется ключевым для проектировщиков.

Навигационный подход в первых СУБД

✔ Сильные стороны навигационного подхода

ПРЕИМУЩЕСТВО	ОБЪЯСНЕНИЕ
Высокая производительность	Прямой доступ по указателям = минимум операций чтения.
Полный контроль над путём доступа	Программист сам выбирает маршрут — можно оптимизировать.
Эффективен для предсказуемых структур	Идеален для биллинга, логистики, сборочных единиц.
Минимальные накладные расходы	Нет оптимизатора запросов, парсера SQL — всё просто и быстро.
Хорошо ложится на железо 1960–70-х	Работает даже на системах с 32 КБ памяти.

Навигационный подход в первых СУБД

✗ Слабые стороны навигационного подхода

НЕДОСТАТОК	ОБЪЯСНЕНИЕ
Сложность разработки и сопровождения	Код разрастается, навигация запутывается, легко ошибиться.
Жёсткая привязка к структуре данных	Изменил схему — переписывай все программы.
Нет абстракции	Программист должен знать физическую структуру (указатели, наборы)
Трудно писать сложные запросы	Нет JOIN, агрегатов, подзапросов — всё вручную.
Невозможно отделить логику от данных	Приложение и структура БД неразрывно связаны.
Нет декларативности	Нельзя сказать "что нужно", только "как это получить".