

## Лекция 3.

# Реляционная модель данных

# Компьютеры в начале 1970-х годов

---



- ЭВМ используются в различных сферах
- Мини-компьютеры (PDP фирмы DEC) доступны небольшим предприятиям
- Компьютеры становятся интерактивными
- Развиваются многопользовательские ОС и файловые системы
- Сетевые и иерархические СУБД
- Множество языков программирования
- Появляются идеи об использовании компьютеров непрофессионалами



# **Навигационный подход - плюсы и минусы**

# 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".

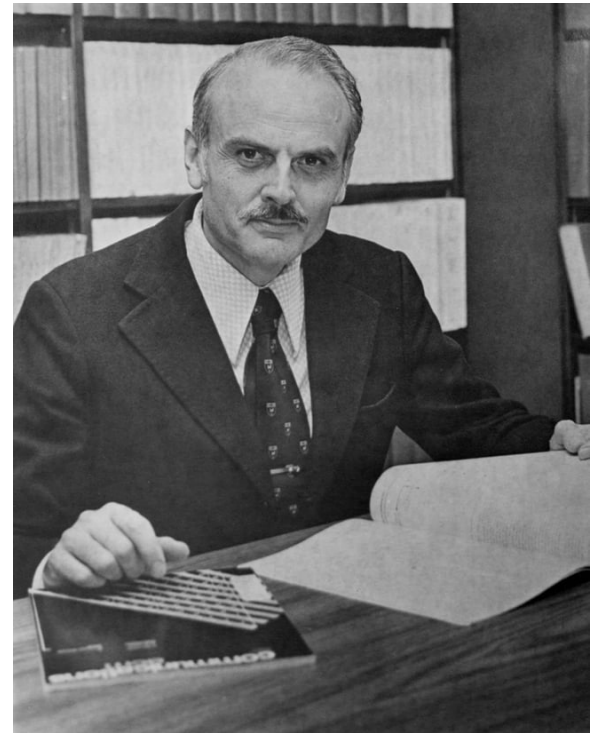
# Проблемы навигационного подхода

- СУБД – инструмент программиста
  - Для использования нужна высокая квалификация программиста
  - Для выполнения любых запросов к данным нужно писать программный код
- В базе данных смешивается логическая и физическая реализация данных.

# **Критика навигационного подхода Э.Ф. Коддом**

## Эдгар Франк "Тед" Кодд (1923-2003)

- Докторская степень по информатике и вычислительной технике
- С 1965 года работал в Исследовательском центре IBM
- Предложил свой подход к организации баз данных
- В 1981 году получил премию Тьюринга



# Осознание проблемы

Кодд, будучи математиком (PhD по теории вычислений, окончил Оксфорд), был поражён:

- Отсутствием математической основы у существующих моделей.
- Зависимостью логики приложения от физического хранения — поменял структуру → переписывай весь код.
- Хрупкостью систем — ошибка в указателе → крах всей базы.
- Сложностью обучения и сопровождения — нужно было знать всю структуру БД назубок.

*“Он задался вопросом: «Почему нельзя работать с данными так же строго и абстрактно, как с числами в математике?» ”*

# Критика навигационного подхода и альтернатива

🔑 Часть 3: Философия Кодда — “Данные должны быть гражданами первого сорта”

Кодд верил:

“□ Данные важнее программ.

Программы меняются — данные живут десятилетиями.

Поэтому модель данных должна быть стабильной, строгой и независимой.”

“□ Математика — лучший фундамент.

Только на основе теории множеств и логики можно построить надёжные,

предсказуемые, оптимизируемые системы.”

“⊖ Никаких указателей.

Указатели — это “физика”, а физика не должна лезть в логику.”

“⚡ Декларативность — свобода.

Скажи системе *что* тебе нужно — пусть она сама решает *как*.”

# Реляционная модель Кодда

# Реляционная модель: начало

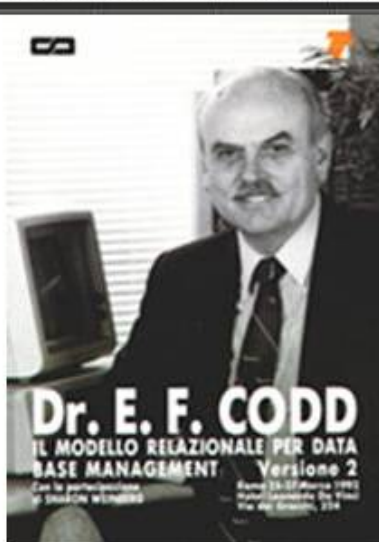
## A Relational Model of Data for Large Shared Data Banks

E. F. Codd  
*IBM Research Laboratory, San Jose, California*

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the “connection trap”).



**Relational Model**

Activity Code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

Key = 24

Activity Code	Date	Route No.
24	01/12/01	I-95
24	02/08/01	I-66

Date	Activity Code	Route No.
01/12/01	24	I-95
01/15/01	23	I-495
02/08/01	24	I-66

«Реляционная модель данных для больших совместно используемых банков данных» Association of Computer Machinery journal (1970 год)

# Основные тезисы статьи

## 1. Данные должны представляться в виде отношений (таблиц)

Кодд предложил отказаться от иерархических и сетевых моделей данных, доминировавших в то время, и использовать **математическую теорию отношений** (реляций) для представления данных. Каждая сущность или набор данных — это таблица (отношение), где строки — кортежи, а столбцы — атрибуты. Это обеспечивает **простоту, гибкость и математическую строгость**.

## Основные тезисы статьи

### 2. Независимость данных от физического представления

Кодд настаивал на логической независимости структуры данных от их физического хранения. Пользователь и приложения должны работать с логической структурой (таблицами), не зная, как данные физически организованы на диске. Это упрощает разработку, обслуживание и масштабирование систем.

## Основные тезисы статьи

### 3. Декларативный язык запросов на основе реляционной алгебры и исчисления

Кодд предложил использовать **реляционную алгебру** и **реляционное исчисление** как основу для языка запросов. Это позволяет формулировать запросы **декларативно** — описывать *что* нужно получить, а не *как* это сделать. Именно эта идея позже легла в основу SQL.

## Основные тезисы статьи

### 4. Целостность и согласованность данных через ключи и ограничения

Кодд ввёл понятия **первичного ключа** (уникально идентифицирует строку) и **внешнего ключа** (обеспечивает ссылочную целостность между таблицами). Он подчеркнул важность **ограничений целостности**, которые должны поддерживаться СУБД автоматически, а не приложением.

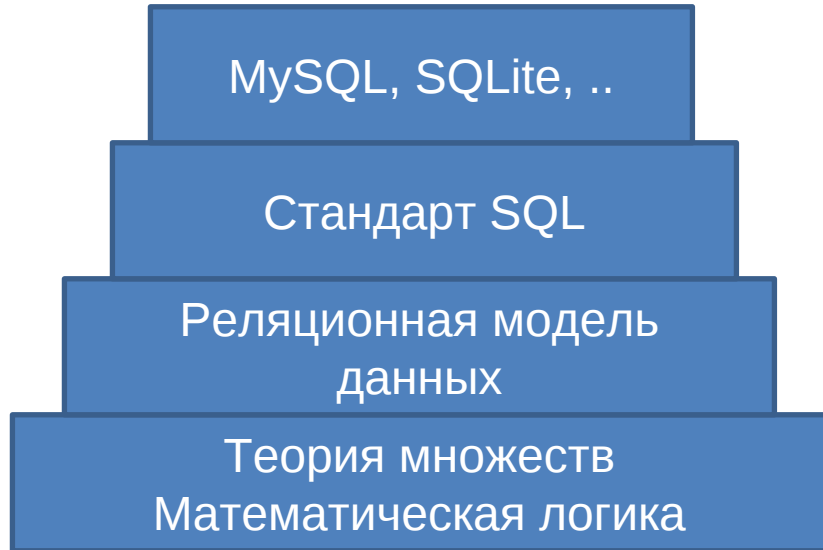
# Основные тезисы статьи

## 5. Теоретическая основа для манипуляции и поиска данных

Кодд показал, что с помощью ограниченного набора операций реляционной алгебры (проекция, выборка, соединение, объединение и др.) можно выполнять **любые операции над данными**. Это обеспечивает **полноту и универсальность модели**, делая её пригодной для любых задач обработки больших банков данных.

# Математика реляционной модели

# Математические основы реляционной модели



*тип данных, таблица, строка таблицы*

*домен, отношение (relation), кортеж,  
реляционная алгебра и исчисление*

*множество, декартово произведение,  
кортеж, предикаты, кванторы, логика  
первого порядка*

# Отношения (relations)

# Отношения (relations) в математике

В математике, **отношение** между двумя множествами — это **подмножество декартова произведения** этих множеств.

**Формальное определение:**

Пусть даны два множества:

- $A$  — первое множество,
- $B$  — второе множество.

Тогда **декартово произведение**  $A \times B$  — это множество всех упорядоченных пар  $(a, b)$ , где  $a \in A$ ,  $b \in B$ .

“Отношение  $R$  между  $A$  и  $B$  — это любое подмножество  $R \subseteq A \times B$ .”

То есть, отношение — это **набор пар**, в которых первый элемент связан со вторым по какому-то правилу или признаку.

# Отношения (relations) в математике

---

Слово "отношение" здесь — перевод английского "relation", которое происходит от латинского "relatio" — "связь", "соотношение".

В математике оно означает **связь между элементами**. То есть, если пара  $(a, b) \in R$ , то мы говорим:

|"Элемент  $a$  находится в отношении  $R$  с элементом  $b$ "."

Это может быть:

- "равно",
- "меньше",
- "является родителем",
- "работает в отделе",
- и т.д.

То есть, **отношение** — это способ описать, как элементы одного множества "относятся" к элементам другого (или того же) множества.

# Математические основы реляционной модели

$a_1^1$	$a_2^1$	...	$a_n^1$	$a_1^1$	$a_2^1$	...	$a_n^1$	...	$a_1^1$	$a_2^1$	...	$a_n^1$
$a_1^2$	$a_2^2$	...	$a_n^2$	$a_1^2$	$a_2^2$	...	$a_n^2$		$a_1^2$	$a_2^2$	...	$a_n^2$
...	...	...	...	...	...	...	...		...	...	...	...
$a_1^m$	$a_2^m$	...	$a_n^m$	$a_1^m$	$a_2^m$	...	$a_n^m$		$a_1^m$	$a_2^m$	...	$a_n^m$
$a_1^1$	$a_2^1$	...	$a_n^1$	$a_1^1$	$a_2^1$	...	$a_n^1$	...	$a_1^1$	$a_2^1$	...	$a_n^1$
$a_1^2$	$a_2^2$	...	$a_n^2$	$a_1^2$	$a_2^2$	...	$a_n^2$		$a_1^2$	$a_2^2$	...	$a_n^2$
...	...	...	...	...	...	...	...		...	...	...	...
$a_1^m$	$a_2^m$	...	$a_n^m$	$a_1^m$	$a_2^m$	...	$a_n^m$		$a_1^m$	$a_2^m$	...	$a_n^m$
...				...				...	...			
$a_1^1$	$a_2^1$	...	$a_n^1$	$a_1^1$	$a_2^1$	...	$a_n^1$	...	$a_1^1$	$a_2^1$	...	$a_n^1$
$a_1^2$	$a_2^2$	...	$a_n^2$	$a_1^2$	$a_2^2$	...	$a_n^2$		$a_1^2$	$a_2^2$	...	$a_n^2$
...	...	...	...	...	...	...	...		...	...	...	...
$a_1^m$	$a_2^m$	...	$a_n^m$	$a_1^m$	$a_2^m$	...	$a_n^m$		$a_1^m$	$a_2^m$	...	$a_n^m$

**Отношение степени  $n$**  - подмножество  $R$  декартового произведения множеств  $A_1 \times A_2 \times \dots \times A_n$ .

- Все элементы отношения есть однотипные кортежи. Поэтому кортежи можно считать аналогами строк в простой таблице.
- Отношение включает в себя не все возможные кортежи (есть критерий, задающий семантику отношения).

## Пример бинарного отношения

---

Множество  $A = \{\text{Вова, Петя, Маша, Лена}\}$ . Известно:

1. Вова любит только себя (эгоист).
2. Петя любит Машу (взаимно).
3. Маша любит Петю (взаимно).
4. Маша любит себя.
5. Лена любит Петю (несчастливая любовь).

Это бинарное отношение "**любить**", заданное на множестве  $A^2$ .

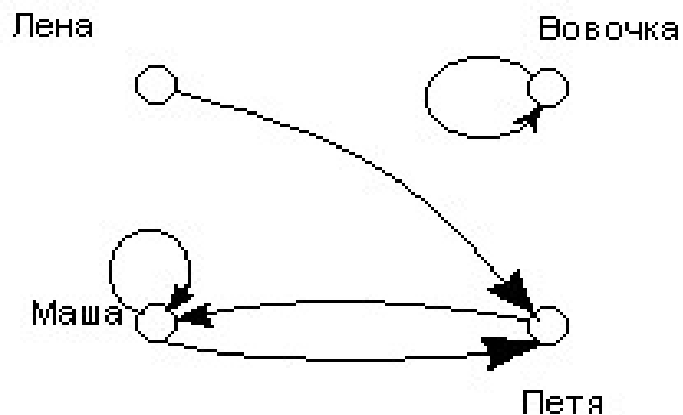
Как представить это отношение, чтобы с ним удобно было работать на компьютере?

Способ 1. Произвольный текст. Тяжело обрабатывать алгоритмами.

1. Вова любит только себя (эгоист).
2. Петя любит Машу (взаимно).
3. Маша любит Петю (взаимно).
4. Маша любит себя.
5. Лена любит Петю (несчастливая любовь).

Способ 1. Произвольный текст. Тяжело обрабатывать алгоритмами.

Способ 2. Граф взаимоотношений. Наглядно, но хранить данные в компьютере в графическом виде неудобно.



# Пример бинарного отношения

## Способ 3. Матрица взаимоотношений.

Кого Кто	Вова	Петя	Маша	Лена
Вова	Любит			
Петя			Любит	
Маша		Любит	Любит	
Лена		Любит		

# Пример бинарного отношения

## Способ 3. Матрица взаимоотношений.

Кого Кто	Вова	Петя	Маша	Лена
Вова	Любит			
Петя			Любит	
Маша		Любит	Любит	
Лена		Любит		

*Негибко – при появлении нового человека придется изменять и строки, и столбцы.*

# Пример бинарного отношения

## Способ 3. Матрица взаимоотношений.

Кого Кто	Вова	Петя	Маша	Лена
Вова	Любит			
Петя			Любит	
Маша		Любит	Любит	
Лена		Любит		

*Негибко – при появлении нового человека придется изменять и с троки, и столбцы.*

## Способ 4. Таблица фактов.

Кто любит	Кого любят
Вовочка	Вовочка
Петя	Маша
Маша	Петя
Маша	Маша
Лена	Петя

# Пример бинарного отношения

## Способ 3. Матрица взаимоотношений.

Кого Кто	Вова	Петя	Маша	Лена
Вова	Любит			
Петя			Любит	
Маша		Любит	Любит	
Лена		Любит		

*Негибко – при появлении нового человека придется изменять и с троки, и столбцы.*

## Способ 4. Таблица фактов.

Кто любит	Кого любят
Вовочка	Вовочка
Петя	Маша
Маша	Петя
Маша	Маша
Лена	Петя

*При появлении новых лиц в таблицу добавляются только строки.*

# **Термины и понятия реляционной модели**

# Основные понятия реляционной модели



Рис. 2.1. Наглядное представление основных понятий реляционной модели

- Данные хранятся в **отношениях**, воспринимаемых как таблицы. Отношением  $R$ , определенном на множествах  $D_1, D_2, \dots, D_n$ , называется подмножество из  $D_1 \times D_2 \times \dots \times D_n$ .
- Строки - **кортежи (tuples)**, столбцы - **атрибуты (attributes)**
- **Степень отношения** - количество атрибутов (столбцов)
- **Мощность отношения** - количество кортежей (строк)

# Основные понятия реляционной модели



Рис. 2.1. Наглядное представление основных понятий реляционной модели

- Множества  $D_1, D_2, \dots, D_n$  – **домены** отношения. Это **множество допустимых значений** для атрибута. Домен имеет уникальное имя, определен на простом типе данных или на другом домене.
- Например, домен "Имя" - множество строк длиной до 50 символов, домен "Оценка" - {2, 3, 4, 5}

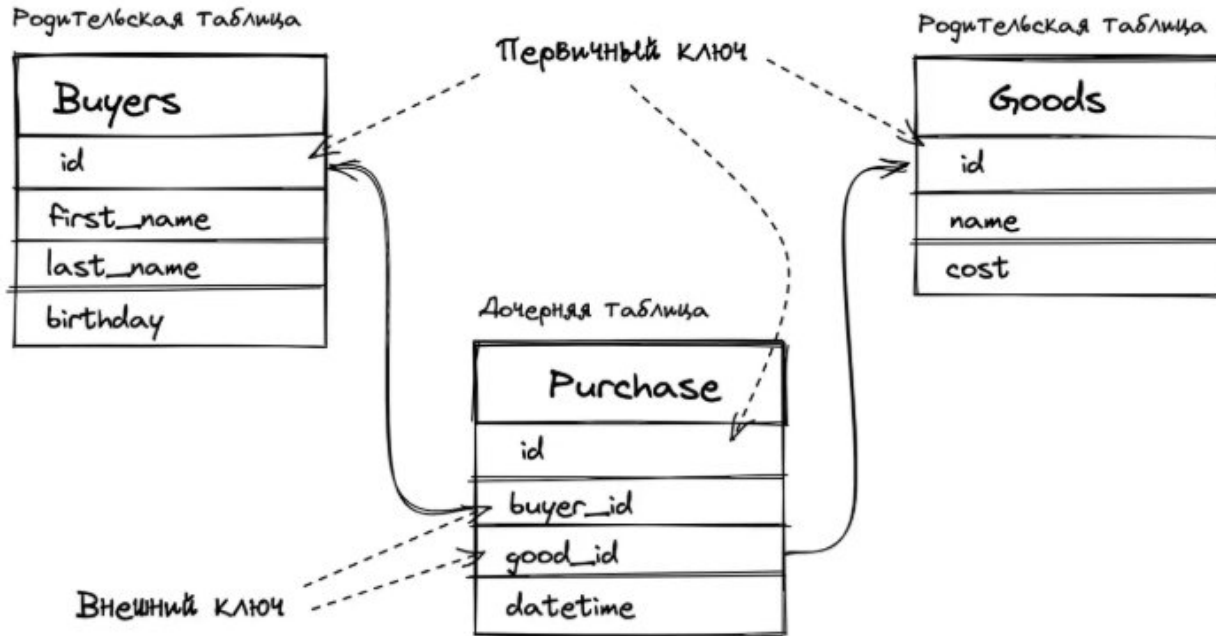
# Основные понятия реляционной модели



Рис. 2.1. Наглядное представление основных понятий реляционной модели

- **Заголовок** отношения – набор  $\{ \langle A_1:D_1 \rangle, \langle A_2:D_2 \rangle, \dots, \langle A_n:D_n \rangle \}$ . Заголовок статичен.
- **Тело** отношения – множество кортежей отношения. Каждый кортеж – множество пар вида  $\langle \text{Имя\_атрибута}:\text{Значение} \rangle$ . Тело отношения изменяется.

# Основные понятия реляционной модели



- **Суперключ** – набор атрибутов, однозначно идентифицирующий кортеж.
- **Кандидатский суперключ** – минимальный суперключ (без лишних атрибутов)
- **Первичный ключ** – выбранный кандидатский суперключ.
- **Внешний ключ** (foreign key) — это атрибут (или набор атрибутов) в одной таблице, который ссылается на первичный ключ другой таблицы (или той же самой таблицы)

# **Реляционная алгебра и реляционное исчисление**

# Понятия «алгебра» и «исчисление»

**Алгебраическая система** — это множество с заданными на нём **операциями** (например,  $+$ ,  $\times$ ,  $\cup$ , JOIN и т.д.) и, возможно, **аксиомами**, которым эти операции подчиняются (например, ассоциативность, коммутативность).

---

## Примеры алгебр:

- **Элементарная алгебра:** операции  $+$ ,  $-$ ,  $\times$ ,  $\div$  над числами.
- **Булева алгебра:** операции  $\wedge$ ,  $\vee$ ,  $\neg$  над истинностными значениями.
- **Множественная алгебра:** операции  $\cup$ ,  $\cap$ ,  $\setminus$  над множествами.
- **Реляционная алгебра:** операции SELECT, PROJECT, JOIN, UNION и др. над отношениями (таблицами).

## Ключевая идея алгебры:

“Вы строите результат, применяя последовательность операций к исходным объектам.”

Это **процедурный, конструктивный подход**:

→ Берём  $A$ , применяем операцию  $f$ , получаем  $B$ ,

→ затем применяем  $g$  к  $B$ , получаем  $C$  — и так далее.

**Исчисление** - система правил и операций для вычисления чего либо.

Формальная система правил, позволяющая выводить («вычислять») одни выражения из других на основе синтаксических преобразования и логических аксиом.

- **Исчисление высказываний** - система логических операций.
- **Вариационное исчисление** - методы нахождения экстремумов функций.
- **Дифференциальное исчисление** - система правил и методов для работы с производными и дифференциалами.
- **Интегральное исчисление** - система правил и методов для вычисления интегралов.

Большинство исчислений, связанных с логикой, имеют **декларативный** характер, потому что их цель — описывать *что истинно*, а не как это вычислить.

## 1. Исчисление высказываний (пропозициональная логика)

- Декларативно: вы задаёте формулу (например,  $(A \rightarrow B) \wedge A \Rightarrow B$ ) и спрашиваете: *истинна ли она?*
- Не важно, *как* вы это проверяете — таблицей истинности, резолюцией, семантическими таблицами — важно *что* вы хотите доказать.

## 2. Исчисление предикатов первого порядка

- Декларативно: вы пишете формулу с кванторами и предикатами —  $\exists x \forall y P(x,y)$  — и интересуетесь, **выполняется ли она в данной модели.**

# Логическое исчисление

💡 Ключевая идея исчисления:

“Вы описываете, *каким условиям* должен удовлетворять результат, а система «выводит» («исчисляет»), что это за результат.”

Это **декларативный, описательный** подход:

- Я хочу все  $x$ , такие что  $P(x)$  истинно.
- Система сама найдёт такие  $x$ .

Декларативность — это **уровень абстракции**. Чем выше уровень, тем больше вы описываете *цель*, а не *механизм*. Логические исчисления — как раз инструменты такого высокого уровня.

# Алгебра и декларативное исчисление

---

“Алгебра — как поваренная книга:

«Возьми яйца, добавь муку, взбей, испеки».

“Исчисление — как заказ в ресторане:

«Я хочу омлет с сыром».

Как его приготовить — не твоё дело, повар (система) сам “исчислит” нужные шаги.”

---

 **Итог — простыми словами:**

- **Алгебра** — это набор действий, которые нужно выполнить, чтобы получить результат.
- **Исчисление** — это набор правил, по которым система сама выведет результат на основе описания условий.
- **Алгебра** говорит “как”, исчисление — “что”.

Оба подхода мощны и дополняют друг друга — особенно в теории баз данных, где алгебра используется для выполнения, а исчисление — для описания и оптимизации.

# Реляционная алгебра и реляционное исчисление

---

## Почему математика так важна?

- Чёткость и однозначность: нет неопределённых понятий — всё строго формализовано.
- Независимость от реализации: логическая модель отделена от физического хранения.
- Возможность оптимизации: зная алгебру, СУБД может перестраивать запрос для ускорения.
- Гарантии корректности: можно доказывать эквивалентность запросов, проверять целостность и т.д.

# Реляционная алгебра

# Реляционная алгебра и реляционное исчисление

---

Реляционная алгебра - набор **операций над отношениями (таблицами)**, которые позволяют получать новые отношения.

► Реляционная алгебра — процедурный язык

Операции:

- Объединение ( $\cup$ )
- Пересечение ( $\cap$ )
- Разность ( $-$ )
- Декартово произведение ( $\times$ )
- Проекция ( $\pi$ )
- Селекция ( $\sigma$ )
- Соединение ( $\bowtie$ )
- Деление ( $\div$ )

“Это основа для оптимизаторов SQL: любой SQL-запрос можно разложить на цепочку этих операций.”

# Основные операции реляционной алгебры

## 1. Объединение (U — Union)

Объединяет две таблицы **одинаковой структуры**, убирая дубликаты.

“Условие: одинаковое количество и совместимые типы атрибутов.”

Пример:

Таблица А: студенты физфака

Имя	Курс
Иван	2
Мария	3

Таблица В: студенты матфака

Имя	Курс
Сергей	2
Мария	3

А ∪ В:

Имя	Курс
Иван	2
Мария	3
Сергей	2

# Основные операции реляционной алгебры

## 2. Разность (– — Difference)

Возвращает строки, которые есть в первой таблице, но отсутствуют во второй.

Пример:

1	A – B:	Таблица А: студенты физфака
2	Имя   Курс	Имя   Курс
3	----- -----	----- -----
4	Иван   2	Иван   2
		Мария   3
		Таблица В: студенты матфака
		Имя   Курс
		----- -----
		Сергей   2
		Мария   3

(Мария есть в обеих — исключается)

# Основные операции реляционной алгебры

## 3. Декартово произведение ( $\times$ — Cartesian Product)

Каждая строка первой таблицы соединяется с каждой строкой второй.

Пример:

```
1 Таблица C: преподаватели
2 Преподаватель
3 -----
4 Петров
5 Сидоров
6
7 Таблица D: предметы
8 Предмет
9 -----
10 Математика
11 Физика
12
13 C × D:
14 Преподаватель | Предмет
15 -----|-----
16 Петров       | Математика
17 Петров       | Физика
18 Сидоров      | Математика
19 Сидоров      | Физика
```

# Основные операции реляционной алгебры

## 4. Проекция ( $\pi$ — Projection)

Выбирает указанные столбцы (атрибуты) из таблицы.

Пример:

```
1  $\pi[\text{Имя}] (A)$ :  
2 Имя  
3 ----  
4 Иван  
5 Мария
```

Таблица A: студенты физфака

Имя	Курс
-----	-----
Иван	2
Мария	3

## 5. Селекция ( $\sigma$ — Selection)

Выбирает строки, удовлетворяющие условию.

Пример:

```
1  $\sigma[\text{Курс}=3] (A)$ :  
2 Имя | Курс  
3 -----|-----  
4 Мария | 3
```

# Основные операции реляционной алгебры

## 6. Переименование ( $\rho$ — Rename)

Переименовывает атрибуты или отношение.

Пример:

1	$\rho$ [Студент-Имя] (A):	Таблица A: студенты физфака
2	Студент   Курс	Имя   Курс
3	----- -----	----- -----
4	Иван   2	Иван   2
5	Мария   3	Мария   3

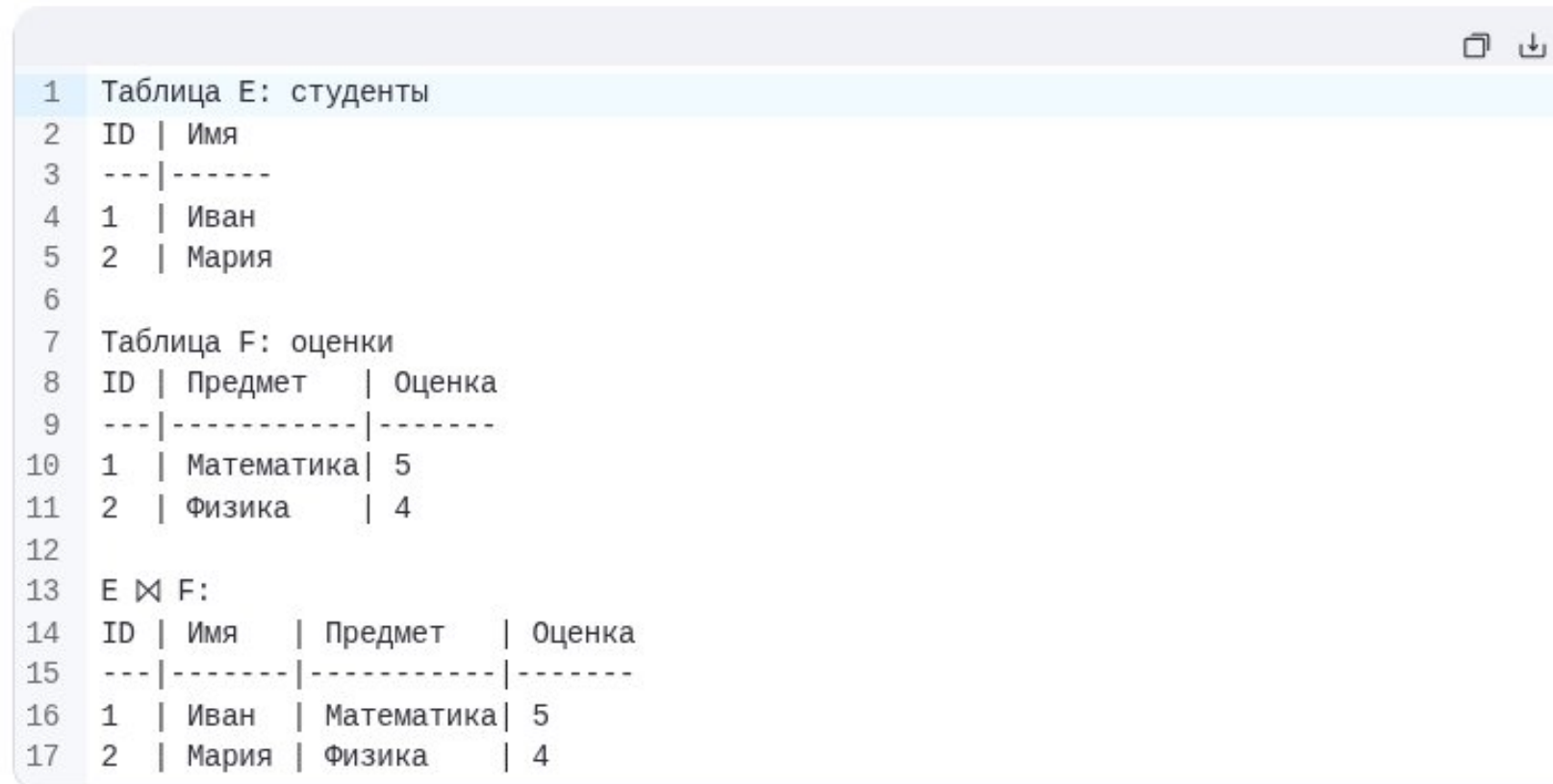
# Основные операции реляционной алгебры

## 7. Соединение ( $\bowtie$ — Join)

Соединяет две таблицы по условию (обычно по равенству атрибутов).

“Чаще всего используется естественное соединение (natural join) — по совпадающим именам атрибутов.”

Пример:



```
1 Таблица E: студенты
2 ID | Имя
3 ---|-----
4 1  | Иван
5 2  | Мария
6
7 Таблица F: оценки
8 ID | Предмет  | Оценка
9 ---|-----|-----
10 1  | Математика| 5
11 2  | Физика    | 4
12
13 E ⋈ F:
14 ID | Имя    | Предмет  | Оценка
15 ---|-----|-----|-----
16 1  | Иван  | Математика| 5
17 2  | Мария | Физика    | 4
```

# Основные операции реляционной алгебры

## 8. Деление ( $\div$ — Division)

Находит строки из первой таблицы, которые связаны со ВСЕМИ строками из второй.

“Полезно для запросов типа: “найти студентов, сдавших ВСЕ предметы”.”

Пример:

1	Таблица G: зачёты студентов	
2	Студент	Предмет
3	-----	-----
4	Иван	Математика
5	Иван	Физика
6	Мария	Математика
7		
8	Таблица H: все обязательные предметы	
9	Предмет	
10	-----	
11	Математика	
12	Физика	
13		
14	G $\div$ H:	
15	Студент	
16	-----	
17	Иван	

(Мария не сдала Физику  $\rightarrow$  не попадает)

# Реляционное исчисление

**Реляционная исчисление** - система вывода (получения) кортежей, удовлетворяющих логической формуле.

- Реализуется в виде декларативного языка запросов к отношениям, основанного на математической логике первого порядка.
- В отличие от реляционной алгебры (как получить результат), реляционное исчисление описывает что нужно получить, не указывая конкретных шагов.

# Реляционное исчисление

## 📌 Пример (Tuple Relational Calculus):

Найти имена всех сотрудников из отдела "IT":

```
1 { t.Имя | Сотрудник(t) ∧ t.Отдел = "IT" }
```

Читается:

«Множество значений `t.Имя`, таких что `t` — кортеж из отношения `Сотрудник` и его атрибут `Отдел` равен "IT"».

Это **не алгоритм**, а **логическое описание** того, что мы хотим получить, а не как это получить.

# Математическая логика первого порядка

Логика первого порядка (англ. *First-Order Logic*, FOL), также называемая исчислением предикатов первого порядка, — это формальная система в математической логике, которая позволяет выражать утверждения о **объектах предметной области**, их свойствах (предикатах) и отношениях между ними, используя:

- переменные (для объектов),
- предикаты (для свойств и отношений),
- логические связки ( $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\rightarrow$ ,  $\leftrightarrow$ ),
- кванторы ( $\forall$  — "для всех",  $\exists$  — "существует"),
- функции (необязательно),
- константы (имена конкретных объектов).

**!** Важно: в логике первого порядка кванторы применяются только к **объектам** (индивидам), а не к предикатам или функциям. Это отличает её от логики *второго порядка*, где можно квантифицировать и по предикатам.

# Математическая логика первого порядка

---

## Пример 1: Математика

“Для любого числа  $x$  существует число  $y$ , такое что  $y = x + 1$ ”

Формально:

$$1 \quad \forall x \exists y (y = x + 1)$$

Здесь:

- $x, y$  — переменные,
- “+” — функция,
- “=” — предикат,
- $\forall, \exists$  — кванторы.

# Математическая логика первого порядка

## Пример 2: База данных / реальный мир

“Каждый студент сдал хотя бы один экзамен”

Предположим:

- $\text{Студент}(x)$  —  $x$  является студентом,
- $\text{Сдал}(x, y)$  — студент  $x$  сдал экзамен  $y$ .

Тогда:

$$1 \quad \forall x (\text{Студент}(x) \rightarrow \exists y \text{Сдал}(x, y))$$

# Логика первого порядка и реляционная модель данных

Это ключевой момент! Связь очень глубокая — реляционная модель баз данных основана на логике первого порядка.

---


 Основная идея Эдгара Кодда (1970):

Кодд предложил реляционную модель как **приложение математической логики и теории множеств** к базам данных. Он показал, что:

“Данные можно рассматривать как отношения (таблицы), а запросы — как выражения логики первого порядка.”

# Реляционное исчисление

- Реляционная алгебра — процедурный язык: вы задаёте последовательность операций (выборка, проекция, соединение и т.д.).
- Реляционное исчисление — декларативный язык: вы описываете свойства результирующих кортежей.

“ Теорема Кодда: любое выражение реляционной алгебры можно выразить в реляционном исчислении, и наоборот — при условии, что выражение **безопасно** (не порождает бесконечные результаты). Это называется **реляционной полнотой**. ”

То есть, оба подхода эквивалентны по выразительной силе, но отличаются стилем: алгебра — “как”, исчисление — “что”.

Есть два основных вида реляционного исчисления:

1. Исчисление кортежей (Tuple Relational Calculus, TRC)
2. Исчисление доменов (Domain Relational Calculus, DRC)

Мы сосредоточимся на исчислении кортежей, так как оно более интуитивно и ближе к SQL.

# Реляционное исчисление кортежей

## ✓ 1. Отношения = предикаты

В логике первого порядка предикат  $P(x_1, x_2, \dots, x_n)$  истинен или ложен для набора значений.

В реляционной модели:

- Таблица `Сотрудники(Имя, Возраст, Отдел)` — это **отношение**.
- Каждая строка — это **факт**, для которого предикат истинен.

Например, строка `("Анна", 30, "IT")` означает, что предикат `Сотрудники("Анна", 30, "IT")` — истинен.

👉 То есть: **таблица** — это множество кортежей, для которых предикат истинен → это экстенционал предиката.

# Синтаксис исчисления кортежей

## ✓ 2. SQL-запросы $\approx$ выражения логики первого порядка

Большинство SQL-запросов (особенно `SELECT ... WHERE ...`) выражаются в логике первого порядка.

Общий вид выражения:

```
1 { t | P(t) }
```

где:

- `t` — переменная кортежа (результат),
- `P(t)` — логическое условие (предикат), которому должен удовлетворять кортеж `t`.

Условие может включать:

- кванторы:  $\exists$  (существует),  $\forall$  (для всех),
  - логические связки:  $\wedge$  (и),  $\vee$  (или),  $\neg$  (не),
  - сравнения атрибутов.
-

# Примеры исчисления кортежей

Предположим, у нас есть следующие отношения (таблицы):

## Сотрудники (Employees)

EMP_ID	NAME	DEPT_ID	SALARY
1	Иван	10	50000
2	Мария	20	60000
3	Сергей	10	55000

## Отделы (Departments)

DEPT_ID	DEPT_NAME
10	Продажи
20	Разработка

# Примеры исчисления кортежей

✓ Пример 1: Найти всех сотрудников с зарплатой > 50000

Исчисление кортежей:

text

```
1 { t | Employees(t) ∧ t.salary > 50000 }
```

Реляционная алгебра:

text

```
1  $\sigma_{\{salary > 50000\}}(Employees)$ 
```

Результат: Мария (60000), Сергей (55000)

---

# Примеры исчисления кортежей

## ✓ Пример 2: Найти имена сотрудников из отдела "Продажи"

Исчисление кортежей:

```
text
1 { t.name | Employees(t) ∧ ∃d (Departments(d) ∧ d.dept_name = 'Продажи' ∧ t.dept_id = d.dept_id)
```

Здесь:

- `t` — кортеж из Employees,
- `d` — кортеж из Departments,
- `∃d(...)` — существует отдел с именем "Продажи", связанный с сотрудником.

Реляционная алгебра:

```
text
1 π_{name}( σ_{dept_name='Продажи'}(Employees ⋈ Departments) )
```

Результат: Иван, Сергей

---

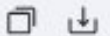
# Примеры исчисления кортежей

## ✓ Пример 3: Найти сотрудников, у которых максимальная зарплата

Это сложнее — нужно выразить, что не существует другого сотрудника с большей зарплатой.

Исчисление кортежей:

text



```
1 { t | Employees(t) ∧ ¬∃s (Employees(s) ∧ s.salary > t.salary) }
```

Читается: “все такие  $t$  из Employees, для которых не существует  $s$  из Employees с зарплатой больше, чем у  $t$ ”.

Результат: Мария (60000)

# Примеры исчисления кортежей

## ✓ Пример 4: Найти отделы, в которых нет сотрудников

Исчисление кортежей:

```
text
```

```
1 { d | Departments(d) ∧ ¬∃e (Employees(e) ∧ e.dept_id = d.dept_id) }
```

Читается: “все отделы  $d$ , для которых не существует сотрудника  $e$ , связанного с этим отделом”.

(В нашем примере такого отдела нет — все отделы имеют сотрудников.)

# Безопасные выражения

Не все выражения в реляционном исчислении допустимы. Например:

```
text
```

```
1 { t | ¬Employees(t) } – НЕБЕЗОПАСНО!
```

Это выражение возвращает все возможные кортежи, которых нет в `Employees` — теоретически бесконечное множество.

👉 **Безопасное выражение** — это выражение, в котором все переменные ограничены существующими значениями из отношений (доменами).

# **Реляционная модель данных - определение и значение**

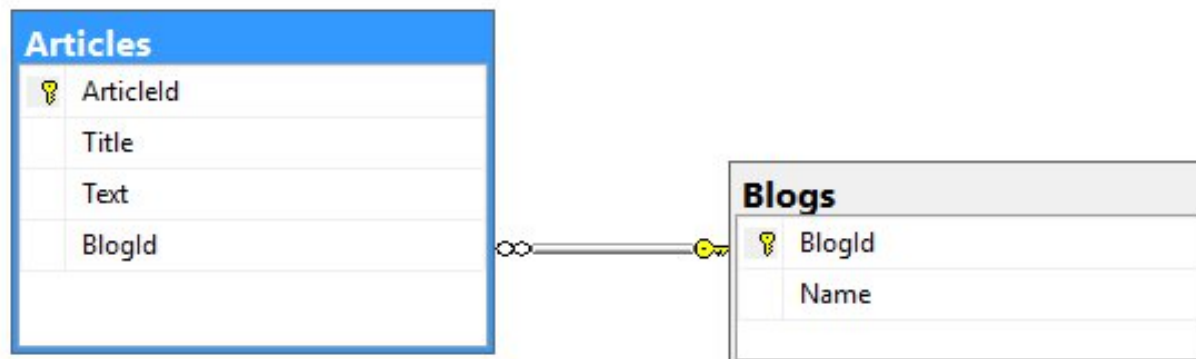
**Реляционная модель данных** — это математическая модель, основанная на теории множеств и логике первого порядка, предложенная Эдгаром Коддом в 1970 году.

- Описывает данные как отношения (таблицы), состоящие из кортежей (строк), где каждый атрибут (столбец) принадлежит определённому домену (множеству допустимых значений).
- Операции над данными формализованы через реляционную алгебру и исчисление.
- Связи между таблицами реализуются «по месту», в момент выполнения операции соединения (JOIN) в запросе через совпадение значений (обычно внешних ключей).

## 🧠 Что значит "связываются по месту"?

Это означает:

- Нет физических связей между строками разных таблиц.
- Нет predetermined путей доступа (как в иерархии или сети).
- Связи логические, и их можно устанавливать динамически, в зависимости от запроса.
- Можно соединять любые таблицы, если есть семантически подходящие столбцы (например, совпадающие ID).
- Можно делать произвольные комбинации, агрегации, фильтрации — гибкость на уровне логики, а не физики.



Реляционная модель произвела революцию: сделала данные независимыми от физического хранения, декларативными и математически строгими.

АСПЕКТ	ИЕРАРХИЧЕСКАЯ/СЕТЕВАЯ МОДЕЛЬ	РЕЛЯЦИОННАЯ МОДЕЛЬ
Тип связей	Физические указатели	Логические (по значениям)
Гибкость запросов	Только predetermined пути	Любые JOIN'ы, подзапросы, агрегаты
Независимость данных	Низкая — приложения зависят от структуры	Высокая — логика отделена от хранения
Изменение схемы	Сложно и опасно	Относительно просто
Основа для оптимизации	Нет — путь фиксирован	Да — оптимизатор выбирает план JOIN'ов

# Навигационный и реляционный подходы

Основное достоинство реляционной модели – применение принципов **нечисловой (ассоциативной) обработки данных**.

- В реляционной модели запросы к БД не ограничены физическими указателями, поэтому для их реализации не нужно писать программы.
- Запросы будут продолжать работать даже после логической реорганизации базы данных.
- Реляционные приложения намного проще навигационных.

# **Реляционная модель - от теории к практике**

# Спор Кодда с Бахманом (конференция ACM SIGMOD, 1973)

---

Этот обмен стал символом двух миров:

- Кодд: идеалист, математик, пророк будущего — верил, что правильная абстракция приведёт к масштабируемости, надёжности и простоте.
- Бахман: практик, инженер, мастер оптимизации — верил, что производительность и контроль важнее теоретической чистоты.

💡 **Смысл спора:**

Не о том, кто «прав», а о том, что важнее — простота или мощь?

— *Для пользователя или — для инженера-системщика?*

# **Первые реляционные СУБД**

# Проект System R

IBM вначале проигнорировала идеи Кодда, потому что у неё уже была успешная, прибыльная и широко используемая система — IMS (иерархическая СУБД), и реляционная модель казалась медленной, нереализуемой на практике и угрожающей существующему бизнесу.

Однако в 1973 году в исследовательской лаборатории IBM в Сан-Хосе начат проект **System R** - прототип реляционной СУБД.

- **Язык SQL** для формулирования запросов (Чемберлен, Бойс) как интерфейс для пользователя БД. Декларативный, близок к естественному, простой синтаксис.
- **Оптимизатор** – автоматически транслировал высокоуровневый запрос в эффективный план его выполнения.
- **Компилятор** запросов – сохранял планы запросов для дальнейшего использования.

IBM не спешила коммерциализировать System R, пока ...

# СУБД Oracle

## ...не появился Oracle

- В 1977 году Ларри Эллисон, Боб Минер и Эд Оутс основали компанию Software Development Laboratories (SDL).
- Они прочитали статьи Кодда и отчёт по System R (который IBM сделала публичным!) — и решили: *“IBM не делает продукт — сделаем мы!”*
- В 1979 году выпустили Oracle V2 — первую коммерческую реляционную СУБД в мире (V1 не существовало — маркетинг 😊).

“ Это был шок для IBM: стартап обошёл гиганта на его же идее.”

# Проект Ingres

Проект **Ingres** (**IN**teractive **GR**elational System) – начат в Калифорнийском университете в Беркли в 1973 году.

Руководитель - профессор Майкл Стоунбрейкер (Тьюринговская премия в 2014 году).

## 💡 Технические особенности раннего Ingres

- **Язык запросов QUEL** — разработан как альтернатива SQL. Был мощным и логически стройным, но в итоге проиграл SQL в "войне стандартов".
- **Архитектура на основе процессов** — каждый пользовательский запрос обрабатывался отдельным процессом (в отличие от современных потоковых архитектур).
- **Оптимизатор запросов** — одна из первых систем, где серьёзно занимались оптимизацией выполнения запросов.
- **Поддержка транзакций и восстановления** — реализованы механизмы отката и восстановления после сбоев.

# СУБД Ingres

В середине 1980-х годов проект начал коммерциализироваться. В 1980 году Стоунабрейкер и его команда основали компанию **Relational Technology, Inc. (RTI)**, которая позже стала **Ingres Corporation**.

## Почему QUEL проиграл SQL?

Несмотря на техническое превосходство QUEL, рынок выбрал **SQL**, потому что:

- IBM активно продвигала SQL через свой проект **System R**.
- SQL стал стандартом ANSI/ISO.
- Компании предпочитали "стандарт", даже если он был менее элегантен.

# СУБД Postgres

В 1986 году Майкл Стоунабрейкер начал новый проект в Беркли — **Postgres** (от **Post-Ingres**), чтобы преодолеть ограничения реляционной модели (например, поддержка сложных типов данных, объектно-реляционные возможности).

В 1994 году из Postgres вырос проект **PostgreSQL**, который стал **открытым исходным кодом** и до сих пор активно развивается. PostgreSQL считается прямым наследником идей Ingres и Postgres.

# СУБД DB2

После успеха System R IBM решила создать коммерческие продукты на его основе.

## ► SQL/DS (Structured Query Language/Data System) — 1981 год

- Первая коммерческая СУБД IBM, основанная на System R.
- Работала под операционной системой DOS/VSE и позже — VM/CMS.
- Поддерживала SQL, но была ориентирована на средние системы, а не мейнфреймы.

## ► Db2 for MVS — 1983 год

- Официальный релиз: июнь 1983 года.
- Db2 стала первой реляционной СУБД для мейнфреймов IBM (MVS, позже z/OS) — ключевой платформы для крупных корпораций и банков.
- Полностью поддерживала SQL и была совместима с идеями System R.

# Реляционный подход: дополнительные преимущества

Кроме повышения продуктивности программистов и простоты использования:

- Реляционная модель хорошо подходит к использованию в архитектуре клиент-сервер. Обмен высокоуровневыми запросами и ответами.
- Декларативный язык SQL позволяет компиляторам организовать параллельную обработку данных.
- Реляционные данные хорошо приспособлены к графическим пользовательским интерфейсам (электронные таблицы).

# **Навигационный подход к реляционным данным**

# Компьютеры в начале 1980-х годов

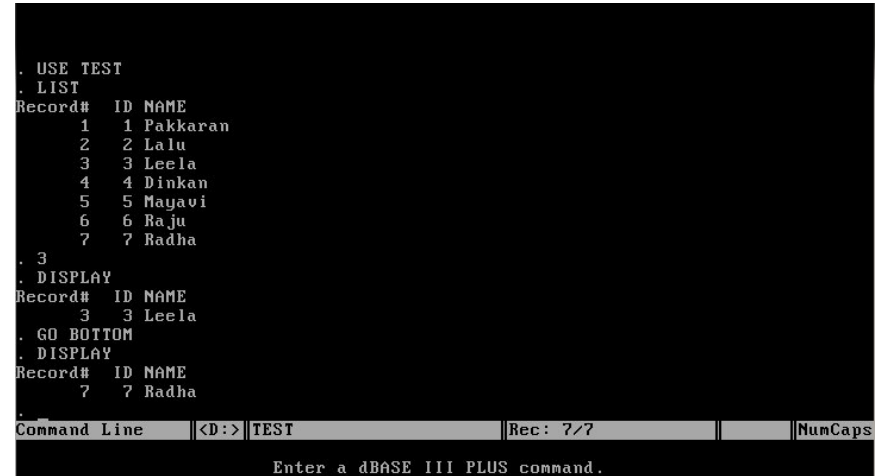
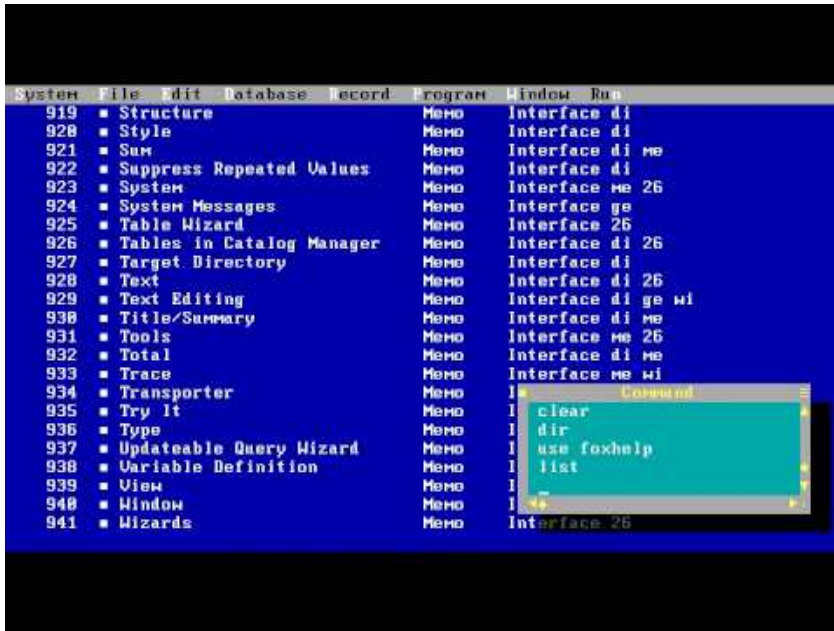
---



- Появляются персональные микро-компьютеры
- Ресурсы микро-компьютера очень ограничены, сложная ОС для него не нужна
- С компьютерами работает все больше непрофессионалов



# Навигационный подход к реляционной модели



- Формат DBF
- СУБД dBase для CP/M (1980)
- FoxPro, Clipper, Paradox, ...

