

# Lecture 2.

## **First DBMSs. Navigational Approach**

Students follow a curriculum and take exams

Group list (Word)

Curriculum Plans  
(Excel)

Exam Session  
Results  
(PDF)

---

**What's the difference between a set of files and a database?**

Students follow a curriculum and take exams

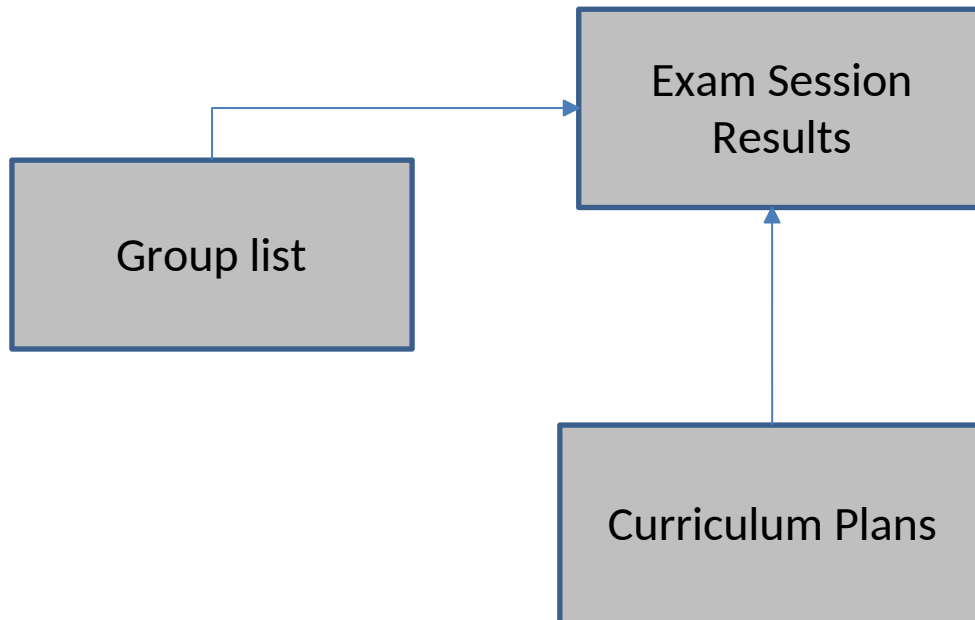
Group list (Word)

Curriculum Plans  
(Excel)

Exam Session  
Results  
(PDF)

---

**What's the difference between a set of files and a database?**



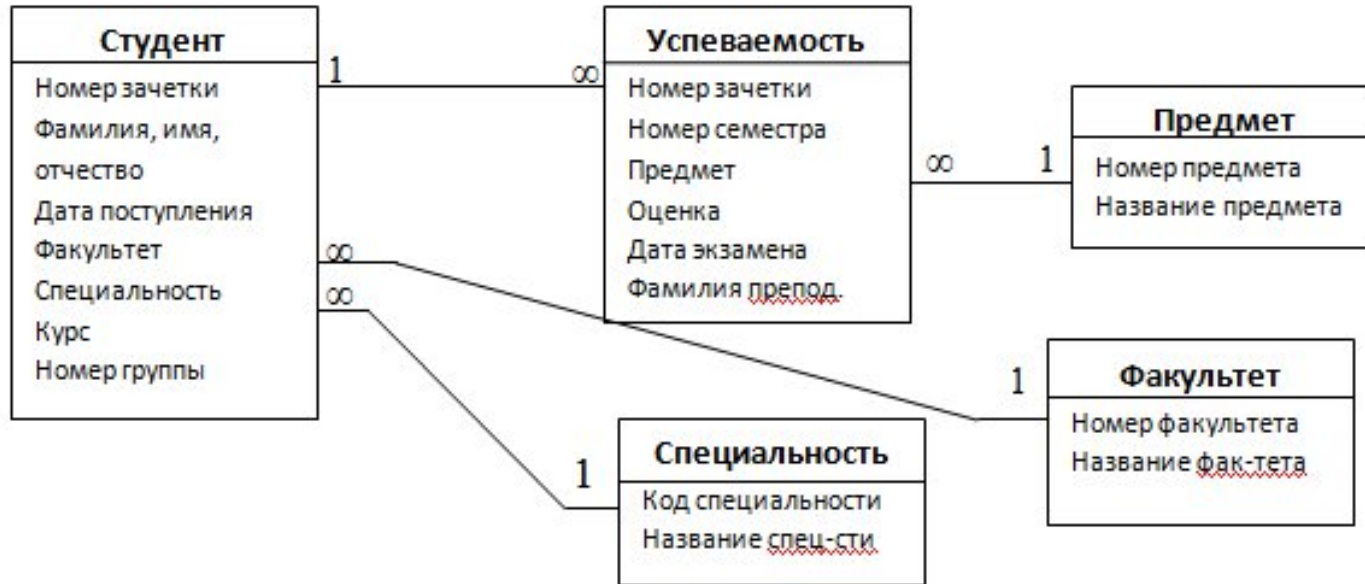
**1. Data Structure**

Entities and their attributes

**2. Relationships Between Data**

How entities are related to one another

# Students follow a curriculum and take exams



**Data Structure + Relationships Between Data**

# Evolution of Data Models

**"Relationship"** is the key concept that distinguishes a database from a simple file or collection of files.

But how should these relationships be represented in a database?

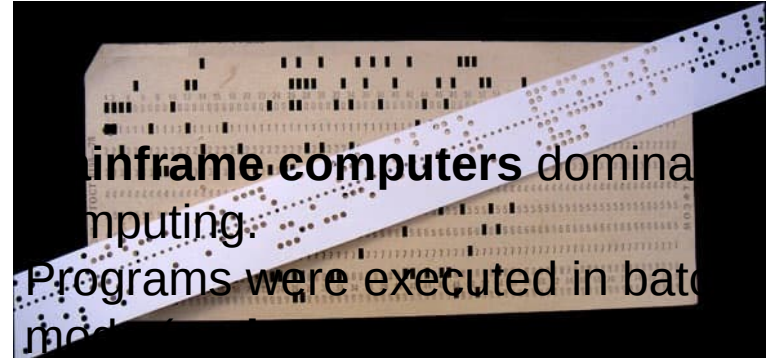
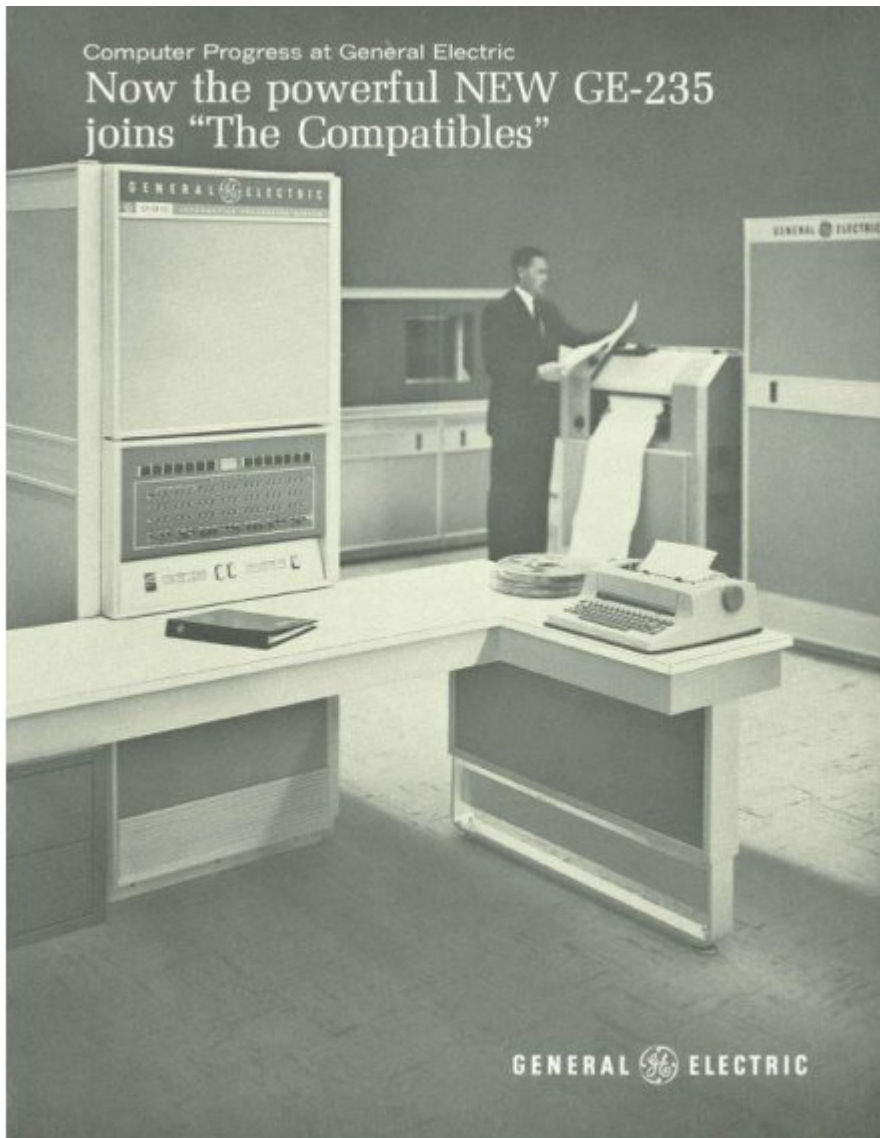
Over time, different approaches — known as **data models** — have been developed.

# Evolution of Data Models

---

Era	Data Models in Use
1960s	<ul style="list-style-type: none"><li>• Files without relationships</li><li>• Hierarchical and Network models (navigational approach, imperative programming)</li></ul>
1970s	<ul style="list-style-type: none"><li>• Hierarchical and Network models</li><li>• <b>Relational model</b> introduced (mathematical foundation, declarative programming)</li></ul>
1980s	<ul style="list-style-type: none"><li>• Relational model becomes dominant</li><li>• Hierarchical and Network models still in use</li></ul>
1990s	Relational model widely adopted as the standard
2000s	<ul style="list-style-type: none"><li>• Relational model remains prevalent</li><li>• Emergence of NoSQL models for scalability and flexible schemas</li></ul>

# Computers in 1961



- **inframe computers** dominated computing.
- Programs were executed in batch mode.
- Primary programming languages: **Fortran, Algol, COBOL.**
- **No universal operating systems** existed.
- **No file systems** as we know them today.
- Computers began to be used for **business applications** (e.g., payroll, inventory).
- Access was limited to **large organizations only**—there were **no individual or general-purpose users.**

# **Separate Data Files and COBOL**

# **The Problem of Processing Data Files**

---

Data was stored in structured files, and custom applications had to be written in programming languages to work with them.

By the late 1950s, every computer manufacturer used **proprietary languages and data formats**, leading to:

- **Hardware-dependent programs**
- **Huge costs** for development, maintenance, and training

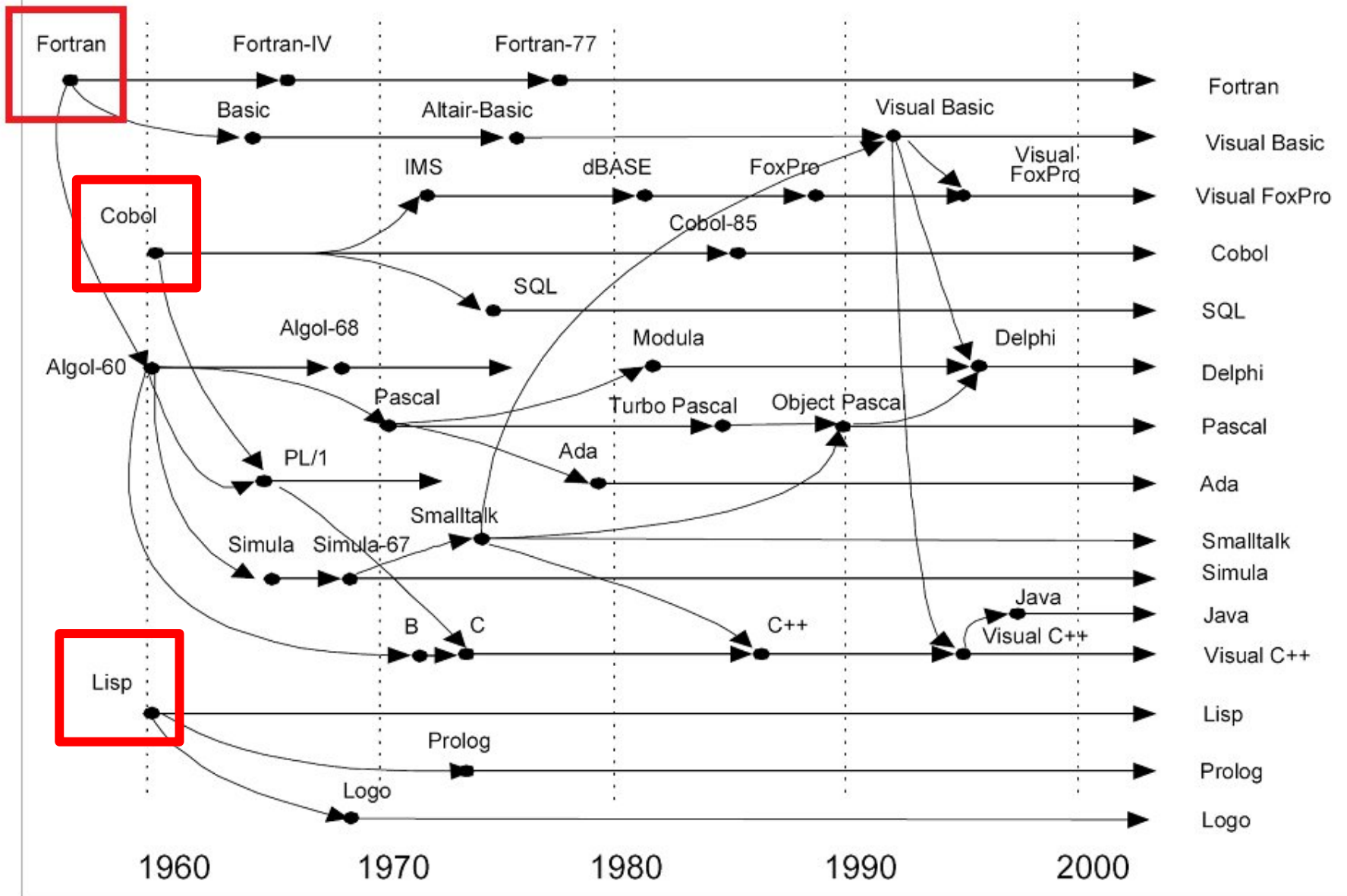
# **COBOL – a Special-Purpose Language for Business Applications**

In 1959, at the initiative of the U.S. Department of Defense, the CODASYL (Conference on Data Systems Languages) committee was formed to develop a universal programming language that would:

- Run on different computer systems
- Be understandable not only to programmers but also to managers and business professionals
- Be specifically designed for data processing tasks (e.g., finance, logistics, reporting)

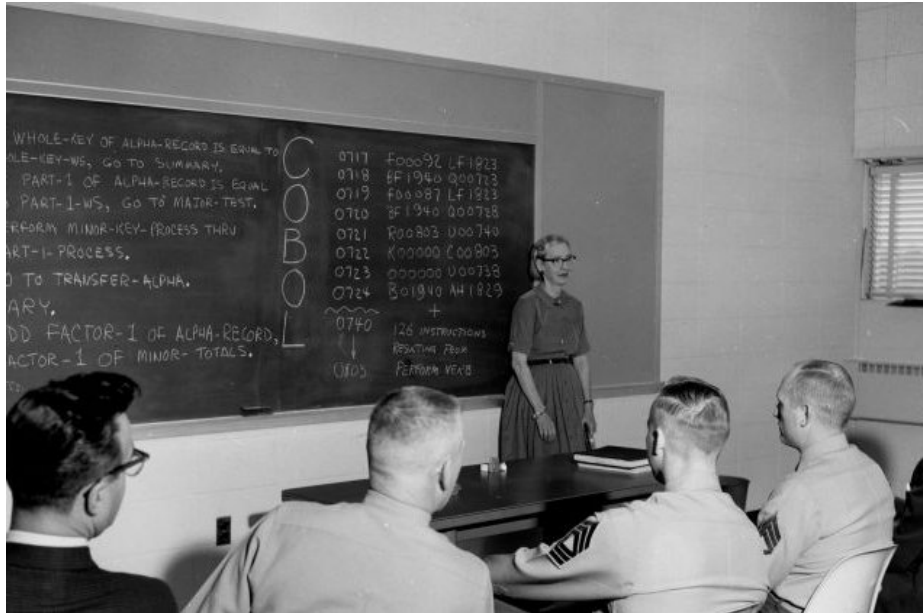
The Department of Defense ensured widespread adoption of COBOL, making it a cornerstone of business computing for decades.

# ГЕНЕАЛОГИЧЕСКОЕ ДЕРЕВО ЯЗЫКОВ ПРОГРАММИРОВАНИЯ



# COBOL – The First Language for Business Applications

---



Grace Hopper served as a technical advisor on the project.

Her language Flow-Matic became the foundation for COBOL — providing its syntax and philosophy.

"Programs should be written so that even an accountant can read them"

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

- Data structures (records)
- Structured files

# Support for Structured Files in COBOL

COBOL supported the following file types based on access method:

- SEQUENTIAL - sequential access
- INDEXED - direct access by a unique key (index)
- RELATIVE - direct access by record number (fixed-length records)

INDEXED and RELATIVE file types were introduced in 1968.

Indexed files served as the precursor to **database tables with primary keys**.

With these capabilities, **COBOL effectively became an embedded mini-DBMS**.

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_
```

## Reading a Structured File (COBOL)

---

# **COBOL – The First Language for Business Applications**

COBOL was the first attempt to standardize structured data storage and access at the programming language level.

- Indexed files in COBOL were the precursor to database tables with primary keys.
- File operations in COBOL laid the groundwork for transactional operations in modern DBMSs.
- Many modern file formats are direct descendants of COBOL file structures.

COBOL became the dominant language for business applications for several decades—and it is still in use today.

# **DBMS for solving data management challenges**

# Application/Developer Responsibilities When Working with Files

---

- **Data Management:** Reading/writing files and parsing their internal structure.
- **Data Integrity:** Enforcing data constraints and validation rules.
- **Multi-user Access:** Implementing locking mechanisms and resolving concurrency conflicts.
- **Performance:** Searching, sorting, and filtering data using custom algorithms.
- **Scalability:** Performance degrades as data volume grows; scaling is difficult.
- **Reliability & Recovery:** Managing backups and implementing change logging manually.
- **Query Capabilities:** Writing code to filter, aggregate, and join data—no built-in query language.
- **Security:** Controlling access via OS-level permissions or custom application logic.

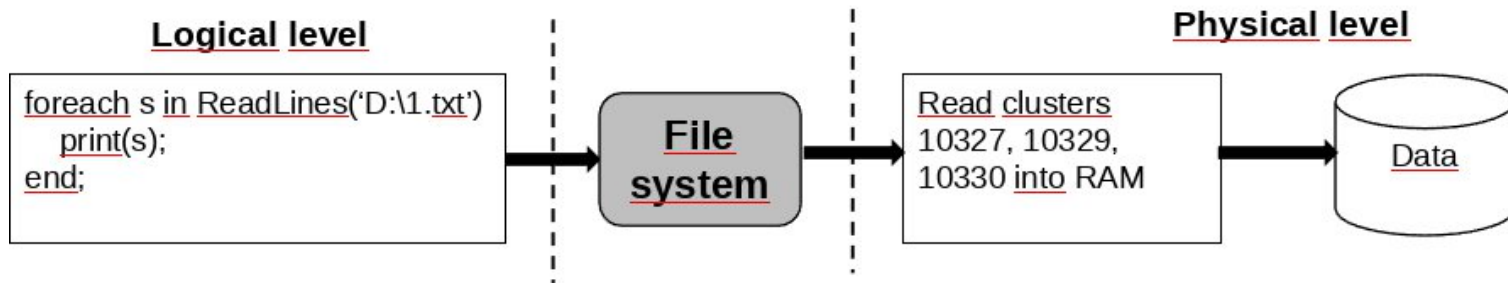
# Delegating Data Management Tasks to a DBMS

---

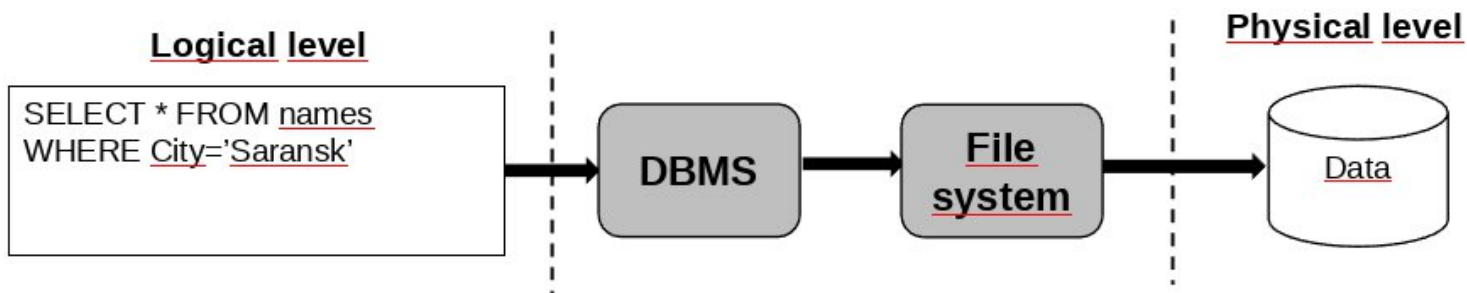
A DBMS takes over a vast layer of infrastructure-level logic that, when using plain files, must be implemented by the developer.

This allows the application to focus on **business logic**, rather than on how to store, retrieve, search, or protect data.

## Working with a file via file system



## Working with a database via DBMS



**DBMS or Plain Files?**

# DBMS or Files?

---

**Use plain files** when a DBMS would be overkill and slower:

- **Data is static or rarely changes**
- **No concurrent access** required
- **No complex queries** needed
- **Small data volume** (up to tens of thousands of records)
- **Simplicity is more important than reliability**

Typical Use Cases for Files:

- **Configuration files:** config.json, .env, settings.yaml, nginx.conf
- **Static reference data:** country lists, currency codes
- **Intermediate/temporary data:** CSV, binary formats
- **Scientific/engineering data files** (e.g., MATLAB, NumPy): numerical arrays are more efficiently stored in binary format

## DBMS

- Data **changes frequently and concurrently**
- There are **relationships between entities**
- You need **transactions, security, and recovery mechanisms**
- The data volume is **large**, requiring **indexes and query optimization**
- You require **SQL support or complex analytics**

### Typical Use Cases for a DBMS:

- **Web applications** with user accounts and dynamic data
- **Analytical dashboards**, monitoring systems, ERP systems
- **Booking systems**, warehouse management, financial platforms
- **Healthcare systems**, banking applications, government registries

# **The World's First DBMS**

# The World's First DBMS: IDS (Integrated Data Store)

---

## Task:

Develop a system for managing inventory and production operations for a large corporation.

## Goal:

Create a **centralized data repository** accessible simultaneously by multiple applications.

## Charles Bachman (1924-2017)

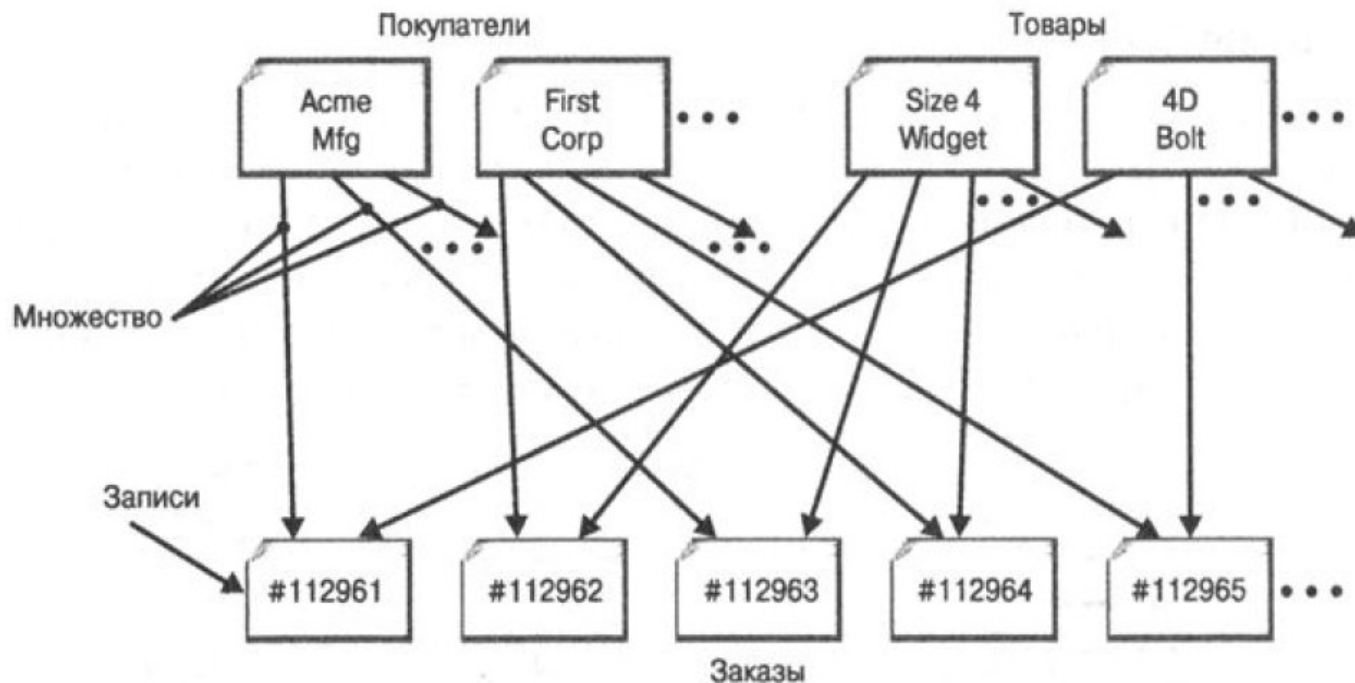
- Worked at General Electric
- Between 1963 and 1964, he developed the Integrated Data Store (IDS) DBMS for the GE-235 mainframe (with just 20 KB of RAM)
- Turing Award in 1973 году.



# IDS Architecture and Key Concepts

## 1. Network Data Model

- Entities (records) are connected by arbitrary **many-to-many relationships**.
- The data structure resembles a **directed graph**.
- Relationships are implemented using **pointers**—physical memory or disk addresses.



# Network Data Model – Key Concepts

## Record

Has an internal structure: fields (attributes) with defined data types.

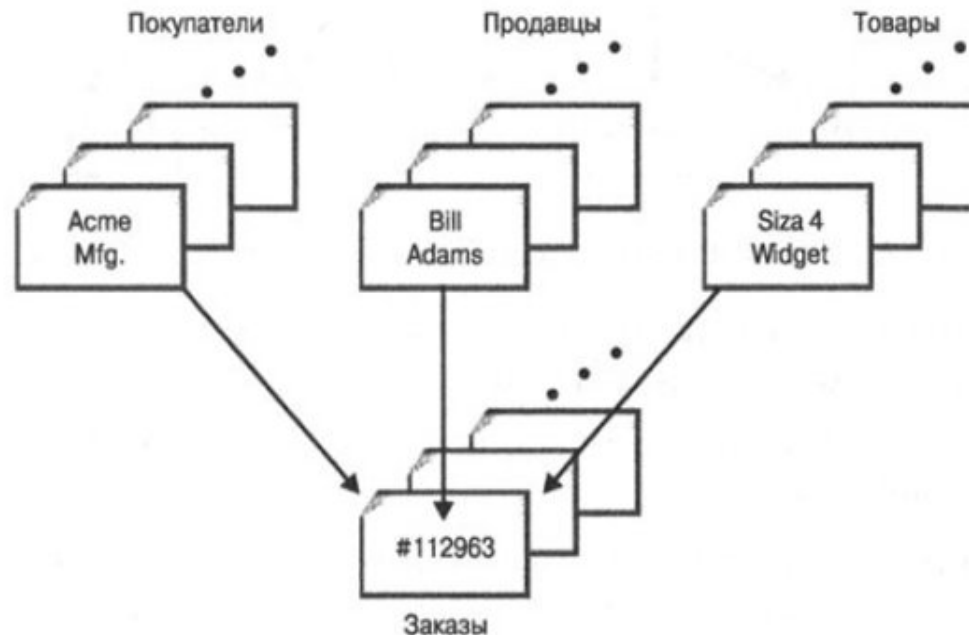
- Departments (DEPARTMENT): DEPT-ID, DEPT-NAME
- Employees (EMPLOYEE): EMP-ID, EMP-NAME, EMP-POSITION
- Projects (PROJECT): PROJ-ID, PROJ-TITLE

# Network Data Model – Key Concepts

## Set

This is a logical "owner-member" relationship.

- One owner is a record of one type.
- Multiple members are records of another (or the same) type.
- A record can belong to multiple sets, enabling many-to-many relationships."



# Network Data Model – Key Concepts

## Set

This is a logical "owner-member" relationship.

- One owner — a record of one type
- Multiple members — records of another type or the same type
- A record can belong to multiple sets, resulting in a **many-to-many relationship**
  
- **DEPT-EMP set:** owner is a DEPARTMENT, members are EMPLOYEE records
- **PROJ-EMP set:** owner is a PROJECT, members are EMPLOYEE records

An employee can belong to a department and simultaneously participate in multiple projects.

# Network Data Model – Key Concepts

## Pointers

Relationships between records are implemented via **physical memory** or **disk addresses**, enabling fast navigation along the links.

Pointer types:

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

# Network Data Model – Key Concepts

## Schema/subschema

- **Schema** – the complete database structure (all record types, sets, and relationships)
- **Subschema** – a customized "view" of the database for a specific application

## 2. Data Manipulation Language (DML)

- An embedded data manipulation language that could be used within an application (e.g., in COBOL).
- Navigation through the data structure: move to the next record, find the owner, etc.

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. Separation of code and data

- Programs (COBOL + DML) — separate
- Data structures and pointers — centralized in the database
- System tables — store metadata about sets, record types, and storage locations

The network data model was the first attempt to separate logical structure from physical storage, providing application independence from the underlying data storage.

# IDS Architecture and Key Concepts

## 4. Transaction Support

- IDS ensured **logical integrity of operations** by executing DML commands sequentially and managing pointer consistency.
- It maintained a **transaction log**, recording all changes made to the data.



# The emergence of IDS was a revolution in data management

## Before IDS

- Each application worked with its own separate files.
- There was no centralized data management.
- Changing the data structure required rewriting all applications.

## After IDS

- The concept of a **database as a shared resource** emerged.
- Formal **data structures, relationships, data manipulation languages**, and **transactions** were introduced.
- The era of **Database Management Systems (DBMS)** began.

# DBMS IDMS – Commercial Implementation of the Network Model

---

- The company Goodrich rewrote IDS in the ISL (Intermediate System Language) and renamed the DBMS to IDMS (Integrated Database Management System). This made it easy to port IDMS to IBM mainframes.
- Later, John Cullinane's company, Cullinane Corporation, acquired IDMS. It became the first successful software vendor specializing in mainframe software, starting in 1968.

# **Significance of the CODASYL DBTG Network Model**

---

Although the network model is no longer used in its pure form today, its core ideas remain influential:

- **Graph databases** (e.g., Neo4j, Amazon Neptune) are modern successors of the network model.
- **Pointers and indexes** are still used internally by relational DBMSs to accelerate join operations.
- The concept of a **subschema** was the precursor to modern **views** (VIEW) and **security roles**.

# **Hierarchical Model.**

## **The First Industrial DBMS**

# IMS Database Management System

---

- **IBM IMS** (Information Management System). The first industrial-strength DBMS deployed enterprise-wide, introduced in 1968.
- Developed to automate the management of large volumes of data related to the design, manufacturing, and logistics of spacecraft for NASA's Apollo program.
- Designed for real-time operation, high performance, and fault tolerance.

IMS consists of two main components:

- **IMS DB** (Database) - a hierarchical database management system
- **IMS TM** (Transaction Manager) - a transaction processing system

# Hierarchical Data Model

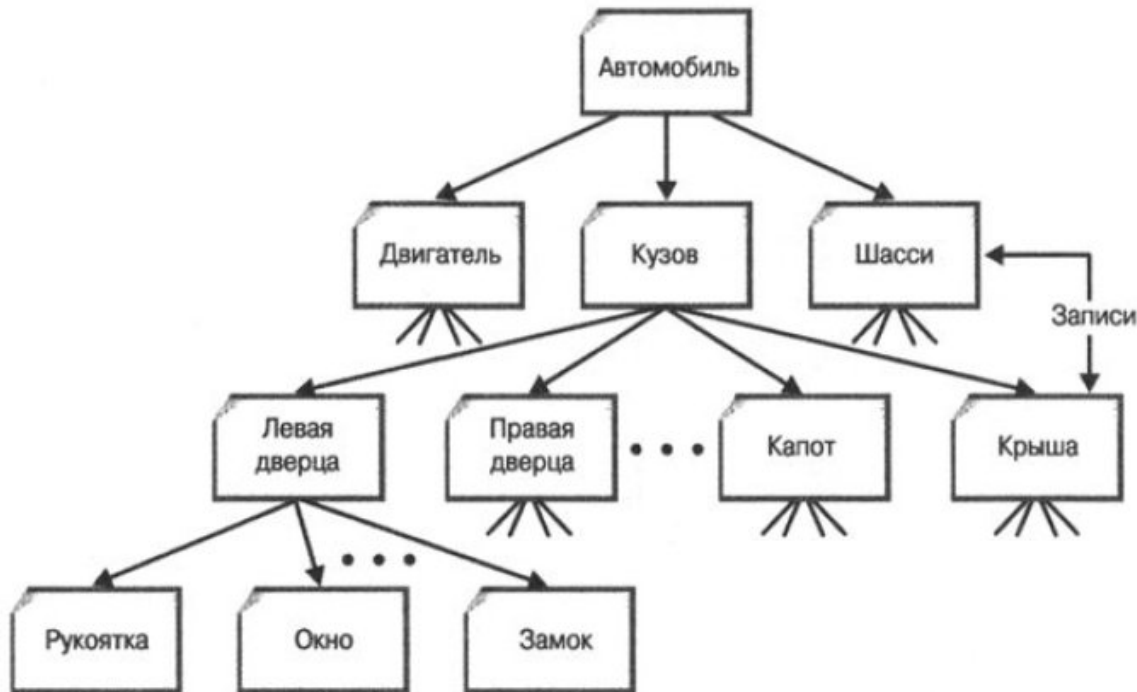
The hierarchical model organizes data in a **tree structure**:

- There is a single **root element**.
- Each element (node) can have **multiple children** but **only one parent**.
- Relationships are strictly **one-to-many**.

# Hierarchical Data Model

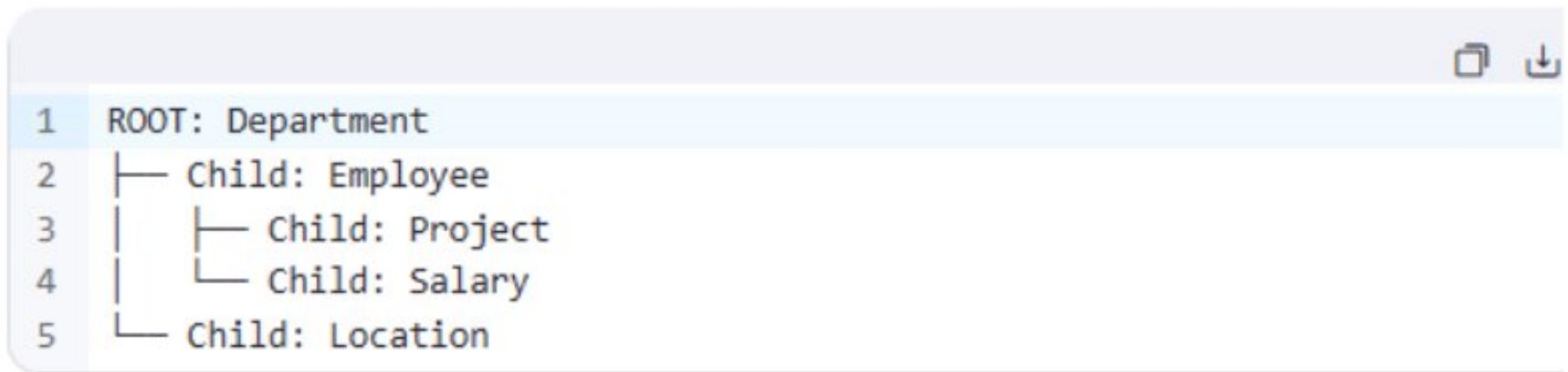
## Production Planning Tasks

- Simple structure
- One-to-many (parent-child) relationship
- High performance (speed)
- Rigid relationship sets



# Hierarchical Data Model in IMS

Data is organized into hierarchical structures called **segments**:



- Each segment has a **type** and **fields**.
- Data access follows a **navigational approach**: from the root down through the tree.
- There is no SQL—instead, **procedural API calls** embedded in programming languages (such as COBOL) are used.

# Hierarchical Data Model

## 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.”

# Typical Data Manipulation Commands

- Find a specific database tree (e.g., the Analytics Department).
- Move from one tree to another.
- Navigate from one record to another within the same tree (e.g., from a department to its first employee).
- Traverse records in hierarchical order (e.g., depth-first or breadth-first traversal).
- Insert a new record at a specified position.
- Delete the current record.

# Hierarchical Databases Today

Why IMS hierarchical DBMS is still relevant today

- **Proven Stability & Reliability.** Decades of mission-critical use in finance, insurance, and government sectors.
- **Unmatched Performance.** Optimized for high-volume, low-latency transaction processing (e.g., thousands of transactions per second).
- **Seamless Integration with Modern Technologies.** Supports REST APIs, JSON, XML, and integration with cloud platforms (IBM Z, IBM Cloud). Can interoperate with relational and NoSQL systems via middleware.

# Hierarchical Databases Today

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.

InterSystems Health | Business | Government

Попробовать Партнерам Разработчикам Контакты Россия Поиск

ОТРАСЛИ РЕШЕНИЯ ПОДДЕРЖКА & ОБУЧЕНИЕ НОВОСТИ РЕСУРСЫ О НАС

## Cache

### Высокопроизводительная база данных

РЕШЕНИЯ > Cache

### База данных для критически важных приложений

InterSystems Cache® — 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.

# Modern Analogues of the Hierarchical Model

---

Although the pure hierarchical model is now obsolete, its core ideas are still used in:

- **XML/JSON** – nested data structures.
- **File systems** – folders and files organized in a tree.
- **NoSQL databases** – e.g., MongoDB (embedded documents), Firebase Realtime Database
- **LDAP directories for network resources** – inherently hierarchical.

# Modern Analogues of the Hierarchical Model

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`.

# Modern Analogues of the Hierarchical Model

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".

# **Navigational approach to data processing**

## **Navigational Approach in Early DBMSs**

The **navigational approach** is a procedural, imperative method of data access in which the programmer manually "traverses" the data structure, following links (pointers) from one record to another to locate the required information.

# Navigational Approach in Early DBMSs

## 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
```

# Navigational Approach in Early DBMSs

## 2. 🌐 Сетевая модель — IDS / IDMS

### 📌 Структура:

Сеть:

Клиент → Заказы → Позиции

Товар → Позиции ← та же запись!

### 🔍 Навигация:

### 💡 Особенности:

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

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
```

# Navigational Approach in Early DBMSs

## Strengths of the Navigational Approach:

- **High Performance for Known Access Paths.** Direct pointer-based navigation enabled extremely fast traversal along predefined relationships.
- **Efficient Use of Limited Hardware Resources.** Optimized for mainframes with minimal memory and storage—no query parsing or optimization overhead.
- **Fine-Grained Control for Developers.** Programmers could precisely manage data access and transaction flow, crucial for mission-critical systems.
- **Predictable Behavior.** Execution was deterministic and transparent—ideal for real-time and embedded applications (e.g., IMS in NASA's Apollo program).

# Navigational Approach in Early DBMSs

## Weaknesses of the Navigational Approach:

- **Complex and Error-Prone Programming.** Developers had to manually code every step of data traversal, increasing development time and bug risk.
- **Inflexible Schema Changes.** Modifying data structures often required rewriting large portions of application logic.
- **Poor Support for Ad Hoc Queries.** No declarative query language (like SQL)—new queries demanded new code.
- **Tight Coupling Between Code and Data Structure.** Applications were highly dependent on physical storage layout, reducing maintainability and portability.