Lecture #01

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

History of Databases

@Andy_Pavlo // 15-721 // Spring 2020

The Steven Moy Foundation for Keeping it Real

TODAY'S AGENDA

Course Logistics Overview
History of Databases

WHY YOU SHOULD TAKE THIS COURSE

DBMS developers are in demand and there are many challenging unsolved problems in data management and processing.

If you are good enough to write code for a DBMS, then you can write code on almost anything else.


salesforce


snowflake


amazon


NUODB


LinkedIn


salesforce


cloudera


snowflake


Google


salesforce


HyPer


salesforce


amazon


impira


facebook


Google


amazon

COURSE OBJECTIVES

Learn about modern practices in database internals and systems programming.

Students will become proficient in:

→ Writing correct + performant code
→ Proper documentation + testing
→ Code reviews
→ Working on a large code base

COURSE TOPICS

The internals of single node systems for in-memory databases. We will ignore distributed deployment problems.

We will cover state-of-the-art topics.
This is **not** a course on classical DBMSs.

COURSE TOPICS

Concurrency Control

Indexing

Storage Models, Compression

Parallel Join Algorithms

Networking Protocols

Logging & Recovery Methods

Query Optimization, Execution, Compilation

BACKGROUND

I assume that you have already taken an intro course on databases (e.g., 15-445/645).

We will discuss modern variations of classical algorithms that are designed for today's hardware.

Things that we will **not** cover:
SQL, Serializability Theory, Relational Algebra, Basic Algorithms + Data Structures.

COURSE LOGISTICS

Course Policies + Schedule:

→ Refer to course web page.

Academic Honesty:

→ Refer to CMU policy page.

→ If you're not sure, ask me.

→ I'm serious. Don't plagiarize or I will wreck you.

OFFICE HOURS

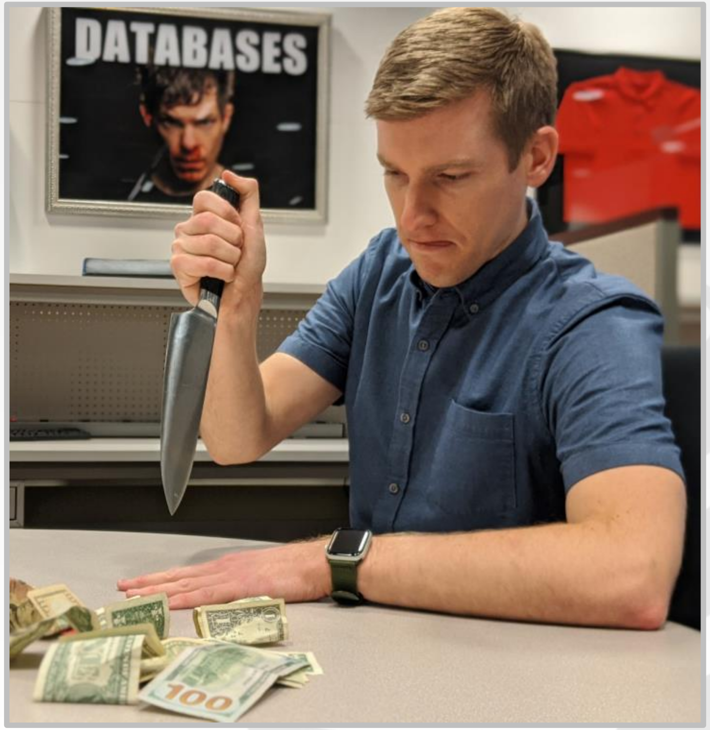
Before class in my office:
→ Mon/Wed: 1:30 – 2:30
→ Gates-Hillman Center 9019

Things that we can talk about:
→ Issues on implementing projects
→ Paper clarifications/discussion
→ How to get a database dev job.
→ How to handle the police

TEACHING ASSISTANTS

Head TA: Matt Butrovich

→ 2^{nd} Year PhD Student (CSD)
→ Lead architect/developer of CMU's DBMS project.
→ Professional Pit Fighter / Boxer
→ Reformed Gang Member (LAX)
→ Vicious AF.



COURSE RUBRIC

Reading Assignments

Programming Projects

Final Exam

Extra Credit

READING ASSIGNMENTS

One mandatory reading per class (★). You can skip **four** readings during the semester.

You must submit a synopsis **before** class:
→ Overview of the main idea (three sentences).
→ Main finding/takeaway of paper (one sentence).
→ System used and how it was modified (one sentence).
→ Workloads evaluated (one sentence).

Submission Form:
https://cmudb.io/15721-s20-submit

☠ PLAGIARISM WARNING ☠

Each review must be your own writing.

You may **not** copy text from the papers or other sources that you find on the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

PROGRAMMING PROJECTS

Projects will be implemented in CMU's new DBMS "**name to be determined**".

→ In-memory, hybrid DBMS
→ Modern code base (C++17, Multi-threaded, LLVM)
→ Strict coding / documentation standards
→ Open-source / MIT License
→ Postgres-wire protocol compatible

PROGRAMMING PROJECTS

Do all development on your local machine.
→ The DBMS only builds on Linux + OSX.
→ We will provide a Vagrant configuration.

Do all benchmarking using Amazon EC2.
→ We will provide details later in semester.

PROJECTS #1 AND #2

We will provide you with test cases and scripts for the first two programming projects.
→ We will teach you how to profile the system.

Project #1 will be completed individually.

Project #2 will be done in a group of **three**.
→ 36 people in the class
→ ~12 groups of 3 people

PROJECT #3

Each group (3 people) will choose a project that is:
→ Relevant to the materials discussed in class.
→ Requires a significant programming effort from all team members.
→ Unique (i.e., two groups cannot pick same idea).
→ Approved by me.

You don't have to pick a topic until after you come back from Spring Break.
We will provide sample project topics.

☠ PLAGIARISM WARNING ☠

These projects must be all of your own code.

You may **not** copy source code from other groups or the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

FINAL EXAM

Take home exam.
Long-form questions on the mandatory readings and topics discussed in class.

Will be given out in class on April 22^{nd}.

EXTRA CREDIT

We are writing an encyclopedia of DBMSs. Each student can earn extra credit if they write an entry about one DBMS.

→ Must provide citations and attributions.

Additional details will be provided later.

This is optional.

☠ PLAGIARISM WARNING ☠

The extra credit article must be your own writing. You may **not** copy text/images from papers or other sources that you find on the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

GRADE BREAKDOWN

Reading Reviews (15%)
Project #1 (10%)
Project #2 (20%)
Project #3 (45%)
Final Exam (10%)
Extra Credit (+10%)

COURSE MAILING LIST

On-line Discussion through Piazza:

https://piazza.com/cmu/spring2020/15721

If you have a technical question about the projects, please use Piazza.

→ Don't email me or TAs directly.

All non-project questions should be sent to me.

Andy's

HISTORY OF DATABASES

WHAT GOES AROUND COMES AROUND
Readings in DB Systems, 4th Edition, 2006.

WHAT'S REALLY NEW WITH NEWSQL?
SIGMOD Record, vol. 45, iss. 2, 2016

HISTORY REPEATS ITSELF

Old database issues are still relevant today.

The **SQL vs. NoSQL** debate is reminiscent of **Relational vs. CODASYL** debate from the 1970s.

→ Spoiler: The relational model almost always wins.

Many of the ideas in today's database systems are not new.

1960s – IDS

Integrated Data Store

Developed internally at GE in the early 1960s.

GE sold their computing division to Honeywell in 1969.

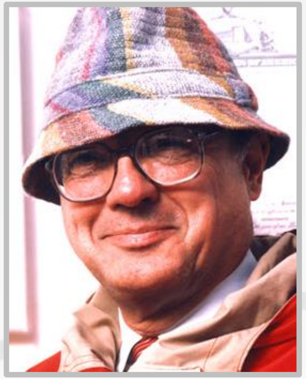
One of the first DBMSs:

→ Network data model.

→ Tuple-at-a-time queries.

1960s – CODASYL

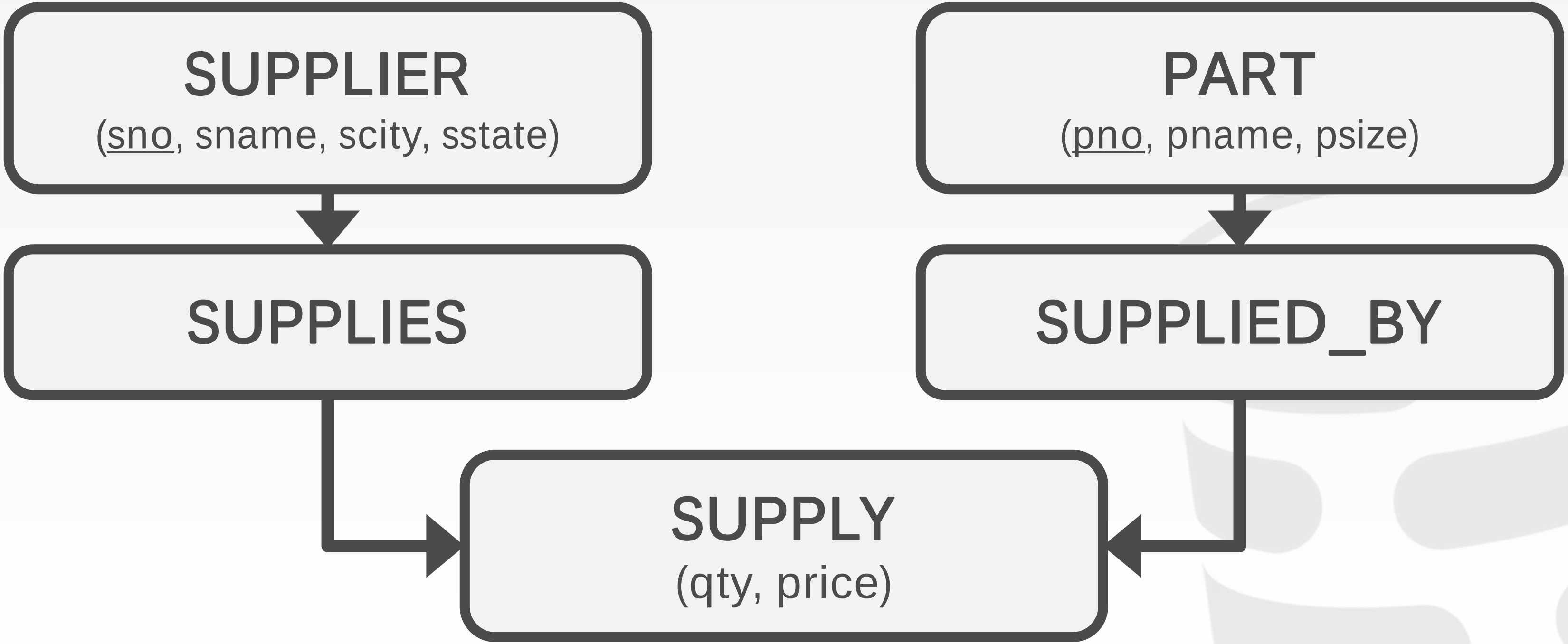
COBOL people got together and proposed a standard for how programs will access a database. Lead by Charles Bachman.
→ Network data model.
→ Tuple-at-a-time queries.


Bachman

Bachman also worked at Culliane Database Systems in the 1970s to help build **IDMS**.

NETWORK DATA MODEL

Schema



NETWORK DATA MODEL

Instance

SUPPLIER

sno	sname	scity	sstate
1001	Dirty Rick	New York	NY
1002	Squirrels	Boston	MA

PART

pno	pname	psize
999	Batteries	Large

SUPPLIES

parent	child

SUPPLIED_BY

parent	child

SUPPLY

qty	price
10	$100
14	$99

NETWORK DATA MODEL

Instance

SUPPLIER

sno	sname	scity	sstate
1001	Dirty Rick	New York	NY
1002	Squirrels	Boston	MA

PART

pno	pname	psize
999	Batteries	Large

SUPPLIES

parent	child

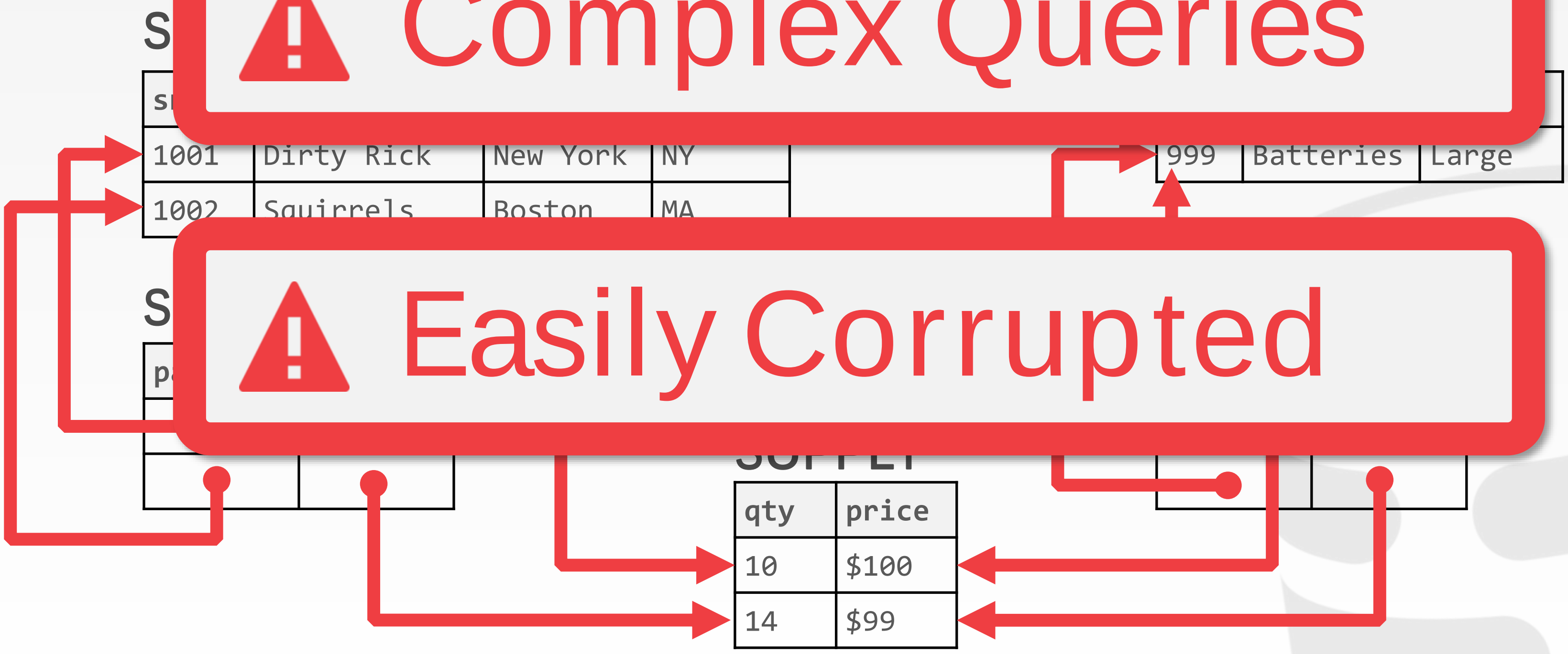
SUPPLIED_BY

parent	child

SUPPLY

qty	price
10	$100
14	$99

NETWORK DATA MODEL



⚠ Complex Queries

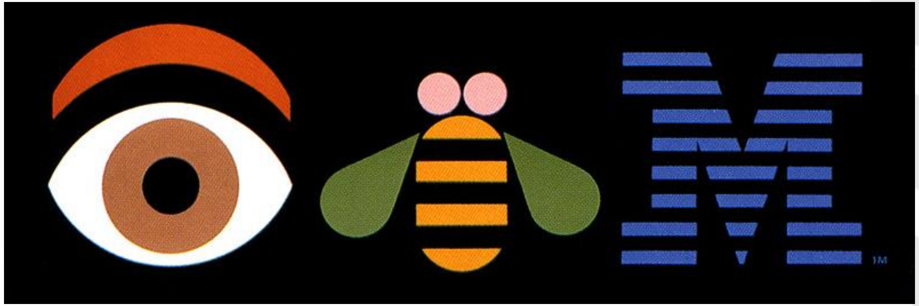
⚠ Easily Corrupted

1960S – IBM IMS

Information **M**anagement **S**ystem

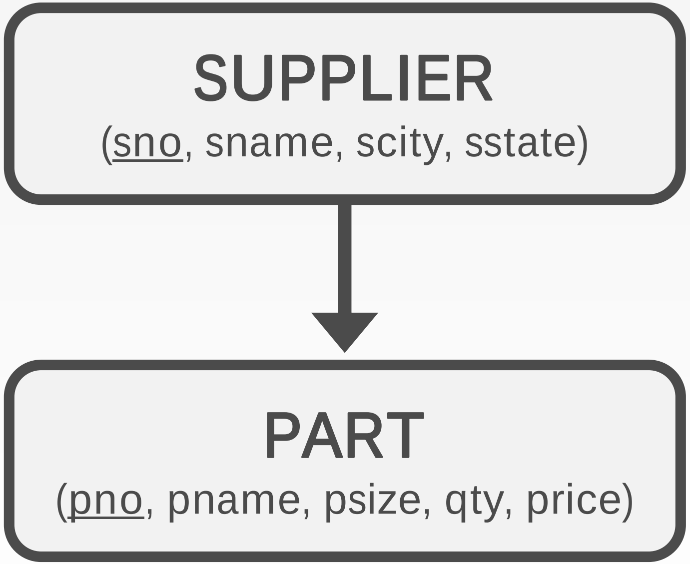
Early database system developed to keep track of purchase orders for Apollo moon mission.

→ Hierarchical data model.
→ Programmer-defined physical storage format.
→ Tuple-at-a-time queries.



HIERARCHICAL DATA MODEL

Schema



Instance

sno	sname	scity	sstate	parts
1001	Dirty Rick	New York	NY	
1002	Squirrels	Boston	MA	

pno	pname	psize	qty	price
999	Batteries	Large	10	$100

pno	pname	psize	qty	price
999	Batteries	Large	14	$99

HIERARCHICAL DATA MODEL

⚠ Duplicate Data

⚠ No Independence

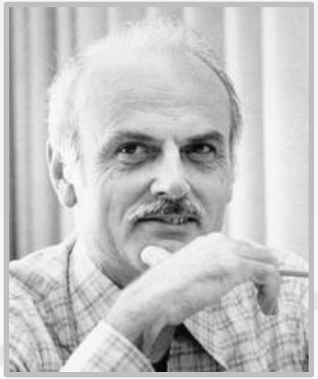
				parts
1002	Squirrels	Boston	MA	

	price
	$100

(pno, pname, psize, qty, price)

pno	pname	psize	qty	price
999	Batteries	Large	14	$99

1970s – RELATIONAL MODEL



Codd

Ted Codd was a mathematician working at IBM Research. He saw developers spending their time rewriting IMS and Codasyl programs every time the database's schema or layout changed.

Database abstraction to avoid this maintenance:
→ Store database in simple data structures.
→ Access data through high-level language.
→ Physical storage left up to implementation.

DERIVABILITY, REDUNDANCY AND CONSISTENCY OF RELATIONS
STORED IN LARGE DATA BANKS

E. F. Codd
Research Division
San Jose, California

ABSTRACT: The large, integrated data banks of the future will contain many relations of various degrees in stored form. It will not be unusual for this set of stored relations to be redundant. Two types of redundancy are defined and discussed. One type may be employed to improve accessibility of certain kinds of information which happen to be in great demand. When either type of redundancy exists, those responsible for control of the data bank should know about it and have some means of detecting any "logical" inconsistencies in the total set of stored relations. Consistency checking might be helpful in tracking down unauthorized (and possibly fraudulent) changes in the data bank contents.

RJ 599(# 12343) August 19, 1969

ATION

maticia
aw deve
writing
y time t
yout ch
avoid
data str
-level la
implem

Information Retrieval

P. BAXENDALE, Editor

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 needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity

CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. Introduction

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence*—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

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 noninferential 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").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

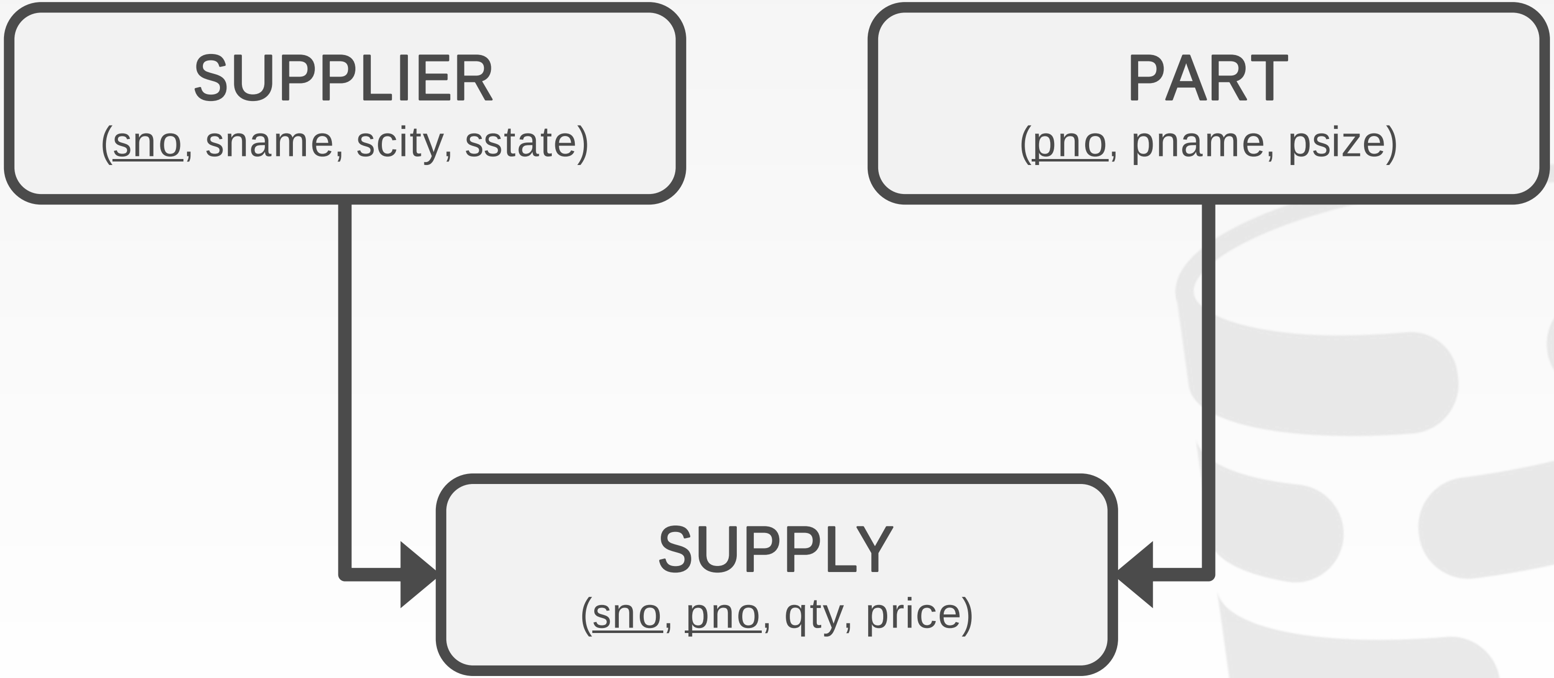
1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

RELATIONAL DATA MODEL

Schema



RELATIONAL DATA MODEL

Instance

SUPPLIER

sno	sname	scity	sstate
1001	Dirty Rick	New York	NY
1002	Squirrels	Boston	MA

PART

pno	pname	psize
999	Batteries	Large

SUPPLY

sno	pno	qty	price
1001	999	10	$100
1002	999	14	$99

RELATIONAL DATA MODEL

Instance

SUPPLIER

sno	sname	scity	sstate
1001	Dirty Rick	New York	NY
1002	Squirrels	Boston	MA

PART

pno	pname	psize
999	Batteries	Large

SUPPLY

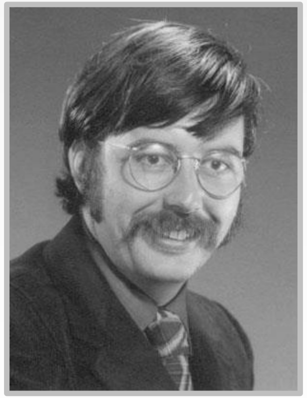
sno	pno	qty	price
1001	999	10	$100
1002	999	14	$99

1970s – RELATIONAL MODEL

Early implementations of relational DBMS:

→ **System R** – IBM Research
→ **INGRES** – U.C. Berkeley
→ **Oracle** – Larry Ellison

Gray



Stonebraker

Ellison

1980s – RELATIONAL MODEL

The relational model wins.
→ IBM comes out with DB2 in 1983.
→ "SEQUEL" becomes the standard (SQL).

Many new "enterprise" DBMSs but Oracle wins marketplace.

Stonebraker creates Postgres.

1980s – OBJECT-ORIENTED DATABASES

Avoid "relational-object impedance mismatch" by tightly coupling objects and database.

Few of these original DBMSs from the 1980s still exist today but many of the technologies exist in other forms (JSON, XML)

OBJECT-ORIENTED MODEL

Application Code

```
class Student {
    int id;
    String name;
    String email;
    String phone[];
}
```

id	name	email
1001	M.O.P.	ante@up.com

sid	phone
1001	444-444-4444
1001	555-555-5555

Relational Schema

STUDENT
(id, name, email)

STUDENT_PHONE
(sid, phone)

OBJECT-ORIENTED MODEL

Application Code

```
class Student {
    int id;
    String name;
    String email;
    String phone[];
}
```

Student

```
{
  "id": 1001,
  "name": "M.O.P.",
  "email": "ante@up.com",
  "phone": [
      "444-444-4444",
      "555-555-5555"
   ]
}
```

OBJECT-ORIENTED MODEL

```
String email;
String phone[];
}
```

```
"email": "ante@up.com",
```

Complex Queries

No Standard API

1990s – BORING DAYS

No major advancements in database systems or application workloads.

→ Microsoft forks Sybase and creates SQL Server.
→ MySQL is written as a replacement for mSQL.
→ Postgres gets SQL support.
→ SQLite started in early 2000.

2000s – INTERNET BOOM

All the big players were heavyweight and expensive. Open-source databases were missing important features.

Many companies wrote their own custom middleware to scale out database across single-node DBMS instances.

2000s – DATA WAREHOUSES

Rise of the special purpose OLAP DBMSs.
→ Distributed / Shared-Nothing
→ Relational / SQL
→ Usually closed-source.

Significant performance benefits from using **columnar data storage** model.

2000s – NoSQL SYSTEMS

Focus on high-availability & high-scalability:

→ Schemaless (i.e., "Schema Last")
→ Non-relational data models (document, key/value, etc)
→ No ACID transactions
→ Custom APIs instead of SQL
→ Usually open-source

2010s – NewSQL

Provide same performance for OLTP workloads as NoSQL DBMSs without giving up ACID:

→ Relational / SQL
→ Distributed
→ Usually closed-source

dbShards
H-Store
Clustrix
Cockroach LABS
Comdb2
ScaleArc
SAP HANA
VOLTDB
NUODB
TiDB
Google Spanner
MEMSQL
HyPer
YugaByte

2010s – HYBRID SYSTEMS

Hybrid **T**ransactional-**A**nalytical **P**rocessing.

Execute fast OLTP like a NewSQL system while also executing complex OLAP queries like a data warehouse system.

→ Distributed / Shared-Nothing
→ Relational / SQL
→ Mixed open/closed-source.

2010s – CLOUD SYSTEMS

First database-as-a-service (DBaaS) offerings were "containerized" versions of existing DBMSs.

There are new DBMSs that are designed from scratch explicitly for running in a cloud environment.

2010s – SHARED-DISK ENGINES

Instead of writing a custom storage manager, the DBMS leverages distributed storage.

→ Scale execution layer independently of storage.

→ Favors log-structured approaches.

This is what most people think of when they talk about a **data lake**.

2010s – GRAPH SYSTEMS

Systems for storing and querying graph data.

Their main advantage over other data models is to provide a graph-centric query API

→ Recent research demonstrated that is unclear whether there is any benefit to using a graph-centric execution engine and storage manager.

2010s – TIMESERIES SYSTEMS

Specialized systems that are designed to store timeseries / event data.

The design of these systems make deep assumptions about the distribution of data and workload query patterns.

M3

TIMESCALE

influxdb

VICTORIA METRICS

ClickHouse

2010s – SPECIALIZED SYSTEMS

Embedded DBMSs

Multi-Model DBMSs

Blockchain DBMSs

Hardware Acceleration

2010s: SPECIALIZED SYSTEMS

- Embedded DBMSs
- Multi-Model DBMSs
- Blockchain DBMSs
- Hardware Acceleration

PARTING THOUGHTS

The demarcation lines of DBMS categories will continue to blur over time as specialized systems expand the scope of their domains.

I believe that the relational model and declarative query languages promote better data engineering.

NEXT CLASS

In-Memory Databases

Make sure that you submit the first reading review

https://cmudb.io/15721-s20-submit