Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

History of Databases

@Andy_Pavlo // 15-721 // Spring 2020
amazon

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.
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
→ 2nd 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
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.
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\textsuperscript{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

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

Schema

SUPPLIER
(sno, sname, scity, sstate)

PART
(pno, pname, psize)

SUPPLIES

SUPPLIED_BY

SUPPLY
(qty, price)
## NETWORK DATA MODEL

### Instance

**SUPPLIER**

<table>
<thead>
<tr>
<th>sno</th>
<th>surname</th>
<th>scity</th>
<th>sstate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Dirty Rick</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>1002</td>
<td>Squirrels</td>
<td>Boston</td>
<td>MA</td>
</tr>
</tbody>
</table>

**PART**

<table>
<thead>
<tr>
<th>pno</th>
<th>pname</th>
<th>psize</th>
</tr>
</thead>
<tbody>
<tr>
<td>999</td>
<td>Batteries</td>
<td>Large</td>
</tr>
</tbody>
</table>

**SUPPLIES**

<table>
<thead>
<tr>
<th>parent</th>
<th>child</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SUPPLY**

<table>
<thead>
<tr>
<th>qty</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>$100</td>
</tr>
<tr>
<td>14</td>
<td>$99</td>
</tr>
</tbody>
</table>

**SUPPLIED_BY**

<table>
<thead>
<tr>
<th>parent</th>
<th>child</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NETWORK DATA MODEL

Instance

**SUPPLIER**

<table>
<thead>
<tr>
<th>sno</th>
<th>sname</th>
<th>scity</th>
<th>sstate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Dirty Rick</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>1002</td>
<td>Squirrels</td>
<td>Boston</td>
<td>MA</td>
</tr>
</tbody>
</table>

**PART**

<table>
<thead>
<tr>
<th>pno</th>
<th>pname</th>
<th>psize</th>
</tr>
</thead>
<tbody>
<tr>
<td>999</td>
<td>Batteries</td>
<td>Large</td>
</tr>
</tbody>
</table>

**SUPPLIES**

<table>
<thead>
<tr>
<th>parent</th>
<th>child</th>
</tr>
</thead>
</table>

**SUPPLIED_BY**

<table>
<thead>
<tr>
<th>parent</th>
<th>child</th>
</tr>
</thead>
</table>

**SUPPLY**

<table>
<thead>
<tr>
<th>qty</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>$100</td>
</tr>
<tr>
<td>14</td>
<td>$99</td>
</tr>
</tbody>
</table>
NETWORK DATA MODEL

Complex Queries

Easily Corrupted
**1960S – IBM IMS**

**Information Management System**

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

SUPPLIER
(sno, sname, scity, sstate)

PART
(pno, pname, psize, qty, price)

Instance

<table>
<thead>
<tr>
<th>sno</th>
<th>sname</th>
<th>scity</th>
<th>sstate</th>
<th>parts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Dirty Rick</td>
<td>New York</td>
<td>NY</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Squirrels</td>
<td>Boston</td>
<td>MA</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pno</th>
<th>pname</th>
<th>psize</th>
<th>qty</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>999</td>
<td>Batteries</td>
<td>Large</td>
<td>10</td>
<td>$100</td>
</tr>
<tr>
<td>999</td>
<td>Batteries</td>
<td>Large</td>
<td>14</td>
<td>$99</td>
</tr>
</tbody>
</table>
HIERARCHICAL DATA MODEL

Supplier
(sno, sname, scity, sstate)

Part
(pno, pname, psize, qty, price)

Duplicate Data

No Independence
1970s – RELATIONAL MODEL

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.
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. He proposed database abstraction to avoid this maintenance: store database in simple data structures; access data through high-level language; physical storage left up to implementation.

A Relational Model of Data for Large Shared Data Banks

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

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 of databases and most applications programs would remain disrupted when the internal representation of data is changed and even when minor aspects of the internal representation are changed. Changes in data representation will often be as desirable or necessary as changes in language. HBC permits an orderly, repeatable method of managing changes in representation; and the cost of handling such changes. The model described in Section 2 is an abstraction on relations (other than logical infinity) and is designed and supplied to make the problem of redundancy and consistency in the user's model of the system a non-issue.

1. Relational Model and Normal Form

1.1 Introduction

This paper is concerned with the application of elementary relationship theory to systems which provide access to large bodies of heterogeneous data. Though a paper by Chen [3] is a primary application of relations to data systems has been to deductive question-answering systems, papers by Abadi et al. [4] and Bylkin et al. [5] provide numerous illustrations of work in this area.

In contrast, the problems treated here arise from data independence—the independence of application programs and terminal activities that arise from the growth of data types and change in data representation—and certain kinds of data anonymity which are expected to become troublesome even in non-heterogeneous systems.

The relational view of file systems is described in Section 1. The view in Section 2 is not intended to be an exact version of any computer system; it is intended as a model which can be used to specify requirements and guide the evaluation of computer systems. A set of queries is defined in Section 3, and the concept of an abstract data type is introduced.
RELATIONAL DATA MODEL

Schema

SUPPLIER
(sno, sname, scity, sstate)

PART
(pno, pname, psize)

SUPPLY
(sno, pno, qty, price)
RELATIONAL DATA MODEL

Instance

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sno</td>
<td>Pno</td>
</tr>
<tr>
<td>Sname</td>
<td>Pname</td>
</tr>
<tr>
<td>Scity</td>
<td>Psize</td>
</tr>
<tr>
<td>Sstate</td>
<td></td>
</tr>
</tbody>
</table>

**SUPPLIER**

<table>
<thead>
<tr>
<th>Sno</th>
<th>Sname</th>
<th>Scity</th>
<th>Sstate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Dirty Rick</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>1002</td>
<td>Squirrels</td>
<td>Boston</td>
<td>MA</td>
</tr>
</tbody>
</table>

**PART**

<table>
<thead>
<tr>
<th>Pno</th>
<th>Pname</th>
<th>Psize</th>
</tr>
</thead>
<tbody>
<tr>
<td>999</td>
<td>Batteries</td>
<td>Large</td>
</tr>
</tbody>
</table>

**SUPPLY**

<table>
<thead>
<tr>
<th>Sno</th>
<th>Pno</th>
<th>Qty</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>999</td>
<td>10</td>
<td>$100</td>
</tr>
<tr>
<td>1002</td>
<td>999</td>
<td>14</td>
<td>$99</td>
</tr>
</tbody>
</table>
RELATIONAL DATA MODEL

Instance

<table>
<thead>
<tr>
<th>sno</th>
<th>sname</th>
<th>scity</th>
<th>sstate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Dirty Rick</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>1002</td>
<td>Squirrels</td>
<td>Boston</td>
<td>MA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</th>
<th>pname</th>
<th>psize</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>999</td>
<td>Batteries</td>
<td>Large</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</th>
<th>qty</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>999</td>
<td>10</td>
<td>$100</td>
</tr>
<tr>
<td>1002</td>
<td>999</td>
<td>14</td>
<td>$99</td>
</tr>
</tbody>
</table>

SUPPLIER

PART

SUPPLY
1970s – RELATIONAL MODEL

Early implementations of relational DBMS:
→ **System R** – IBM Research
→ **INGRES** – U.C. Berkeley
→ **Oracle** – Larry 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

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

Relational Schema

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>M.O.P.</td>
<td><a href="mailto:ante@up.com">ante@up.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>444-444-4444</td>
</tr>
<tr>
<td>1001</td>
<td>555-555-5555</td>
</tr>
</tbody>
</table>
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

Application Code

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

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
2010s – HYBRID SYSTEMS

Hybrid Transactional-Analytical Processing.

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.
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 it 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.
2010s – SPECIALIZED SYSTEMS

Embedded DBMSs
Multi-Model DBMSs
Blockchain DBMSs
Hardware Acceleration
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