Lecture 1: Course Introduction & History of Database Systems

CREATING THE NEXT®

Welcome!

- This course focuses on the design and implementation of database management systems (DBMSs).
- We will study the internals of modern database management systems.
- We will cover the core concepts and fundamentals of the components that are used in high-performance transaction processing systems (OLTP) and large-scale analytical systems (OLAP).

Course Outline & Logistics

Why you should take this course?

- You want to learn how to make database systems **scalable**, for example, to support web or mobile applications with millions of users.
- You want to make applications that are highly **available** (*i.e.*, minimizing downtime) and operationally robust.
- You have a natural curiosity for the way things work and want to know what goes on inside major websites and online services.
- You are looking for ways of making systems easier to maintain in the long run,even as they grow and as requirements and technologies change.
- If you are good enough to write code for a database system, 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
  - Working on a systems programming project

Course Topics

- Logging & Recovery Methods
- Concurrency Control
- Query Optimization
- New Hardware (NVM, GPU)

Background

- I will assume that you have already taken an intro course on database systems (*e.g.*, GT 4400).
- We will discuss modern variations of classical algorithms that are designed for today's hardware.
- Things that we will **not** cover: SQL, Relational Algebra, Basics of Operating Systems, Computer Architecture, Algorithms + Data Structures.

Background

- All programming assignments will be written in C++17.
- You will learn how to debug and profile multi-threaded programs.
- Assignment 1 will help get you caught up with C++.

Course Logistics

- Course Web Page
  - Schedule: https://www.cc.gatech.edu/ jarulraj/courses/8803-s23/
- Discussion Tool: Piazza
  - For all technical questions, please use Piazza. Don’t email me directly.
  - All non-technical questions should be sent to me
- Grading Tool: Gradescope
  - You will get immediate feedback on your assignment.
  - You can iteratively improve your score over time.
- Office Hours
  - Both in-person and remote participation allowed
  - Sign-up sheet posted on Piazza.

Course Logistics

- Course Policies
  - Programming assignments and exercise sheets must be your own work.
  - You may **not** copy source code from other people or the web.
  - Plagiarism will **not** be tolerated.
- Academic Honesty
  - Refer to Georgia Tech Academic Honor Code.
  - If you are not sure, ask me.

Late Policy

- You are allowed **four** penalty-free slip days for the entire course.
- After those slip days are used, you will lose 25% of an assignment's points for every 24 hrs it is late.
- Mark on your submission (1) how many days you are late and (2) how many late days you have left.

Teaching Assistants

- Ashmita Raju
  - M.S. (Computer Science)
  - Worked at Microsoft
- Ishwarya Sivakumar
  - M.S. (Computer Science)
  - Worked at Wells Fargo
- If you are acing through the assignments, you might want to hack on the video database system system (codenamed EVA) that we are building.
- Drop us a note if you are interested!

Course Rubric

- Programming Assignments (20%)
- Project (20%)
- Exams (Exam 1: 15%, Exam 2: 20%)
- Exercise Sheets (3 × 5%)
- Class Participation (10%)

Programming Assignments

- Four assignments based on the BuzzDB academic DBMS.
- Goal is to familiarize you with the internals of database management systems.
- We will use Gradescope for giving you immediate feedback on programming assignments and Piazza for providing clarifications.
- We will provide you with test cases and scripts for the programming assignments.
- If you have not yet received an invite from Gradescope, you can use the entry code that has been shared on Canvas.

Machine Setup

- Operating System (OS): Ubuntu 18.04
- **Build System**: cmake
- Testing Library: Google Testing Library (gtest)
- **Continuous Integration (CI)** System: Gradescope
- Memory Error Detector: valgrind memcheck

C++ Topics

- STL map
- File I/O
- Threading (later assignments)
- Smart Pointers (later assignments)

Assignment 1

- **Goal:** Help brush up your C++ programming skills
- Knowledge of basic data structures and algorithm design

Project

- It will be an open-ended assignment based on the EVA database system.
- You can choose to do the project individually or in a group of two students.
  - Relevant to any topic discussed in class.
  - Goal is to get your creative juices flowing!
  - You don't have to pick a topic until midway through the course.
  - We will provide sample topics.
  - This assignment can be a conversation starter in interviews.

Project – Deliverables

- Proposal: 1-page report
- Checkpoint Report: 2-page report
- Final Report: 2-page report + 5-min presentation
- More details on course website

Exercise Sheets

- Four pencil-and-paper tasks.
- You will need to upload the sheets to Gradescope.
- We will share the grading rubric via Gradescope.

Exercise Sheet #1

- Hand in one page with the following information:
  - Digital picture (ideally 2x2 inches of face)
  - Name, interests, More details on Gradescope
- The purpose of this sheet is to help me:
  - know more about your background for tailoring the course, and
  - recognize you in class

Class Participation

- Starting from lecture 2, we will be use Point Solutions for interactive quizzes.
- Please install the app or use the website:
- **Application Link**
- **Website Link**
- The session ID will be: **8803asi**

History of Database Systems

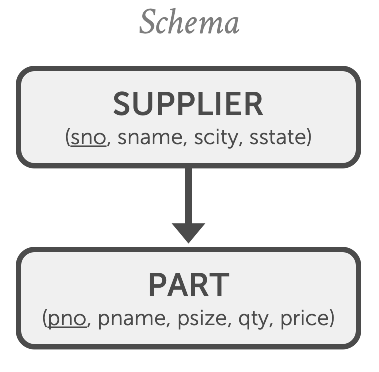
History Repeats Itself

- Reference
- Design decisions in early database systems are still relevant today.
- The “SQL vs. NoSQL” debate is reminiscent of “Relational vs. CODASYL” debate.
- Old adage: he who does not understand history is condemned to repeat it.
- Goal: ensure that future researchers avoid replaying history.

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)

Hierarchical Data Model

students

sno	sname	scity	sstate	parts
1001	Electrical Parts	New York	NY	part-1
1002	Auto Parts	Boston	MA	part-2

part-1

pno	pname	psize	qty	price
999	Fridge	Large	10	100

part-2

pno	pname	psize	qty	price
888	Batteries	Small	14	99

Hierarchical Data Model

- Advantages
  - No need to reinvent the wheel for every application
  - **Logical data independence**: New record types may be added as the logical requirements of an application may change over time.

Hierarchical Data Model

- Limitations
  - Information is repeated.
  - **Tree** structured data model is very restrictive: Existence depends on parent tuples.
  - No **Physical data independence**: Cannot freely change storage organization to tune a database application because there is no guarantee that the applications will continue to run
  - Optimization: A tuple-at-a-time user interface forces the programmer to do manual query optimization, and this is often hard.

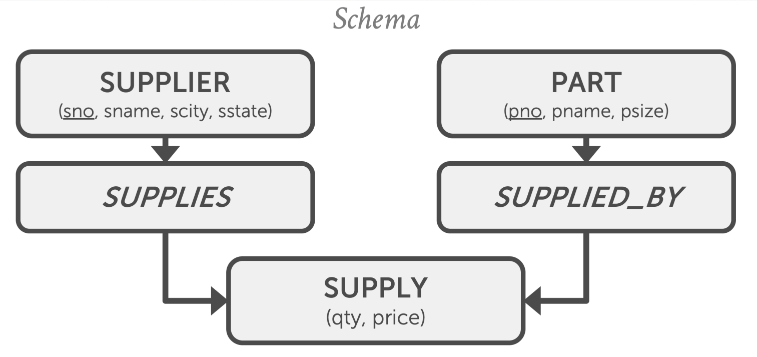
1960s – IDS

- Integrated Data Store
- Developed internally at GE in the early 1960s.
- GE sold their computing division toHoneywell in 1969.
- One of the first DBMSs:
  - Network data model.
  - Tuple-at-a-time queries.

1960s – CODASYL

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

Network Data Model



Schema

SUPPLIER
(sno, sname, scity, sstate)

PART
(pno, pname, psize)

SUPPLIES

SUPPLIED_BY

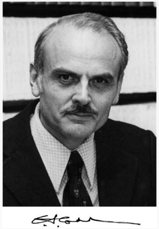
SUPPLY
(qty, price)

Network Data Model

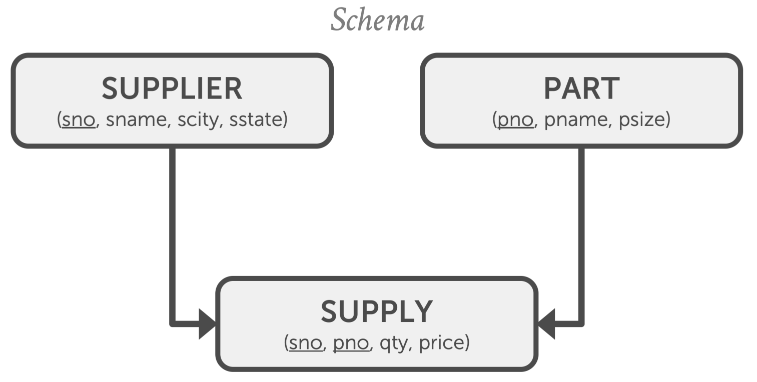
- Advantages
  - **Graph** structured data models are less restrictive
- Limitations
  - Poorer physical and logical data independence: Cannot freely change storage organizations or change application schema
  - **Slow loading and recovery:** Data is typically stored in one large network. This much larger object had to be bulk-loaded all at once, leading to very long load times.

1970s – Relational Data 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 declarative language.
  - Physical storage left up to implementation.



1970s – Relational Data Model



Schema

SUPPLIER
(sno, sname, scity, sstate)

PART
(pno, pname, psize)

SUPPLY
(sno, pno, qty, price)

Relational Data Model

- Advantages
  - Set-a-time languages are good, regardless of the data model, since they offer physical data independence
  - Logical data independence is easier with a simple data model than with a complex one.
  - Query optimizers can beat all but the best tuple-at-a-time DBMS application programmers.

1970s – Relational Data Model

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

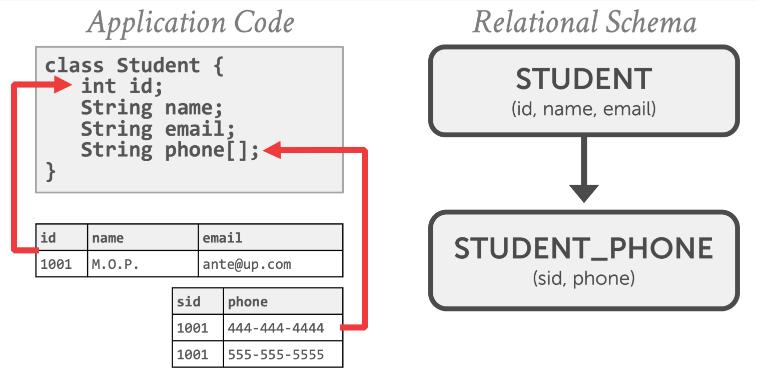
1980s – Relational Data Model

- The relational model wins.
  - IBM comes out with DB2 in 1983.
  - “SEQUEL” becomes the standard (SQL).
- Many new “enterprise” DBMSs, but Oracle wins marketplace.
- **Examples:** Teradata, Informix, Tandem, *e.t.c.*

1980s – Object-Oriented Data Model

- Avoid **relational-object impedance mismatch** by tightly coupling objects and database.
- Analogy: Gluing an apple onto a pancake
- Objects are treated as a first class citizen.
- Objects may have many-to-many relationships and are accessed using pointers.
- Few of these original DBMSs from the 1980s still exist today but many of the technologies exist in other forms (*e.g.*, JSON, XML)
- **Examples:** Object Store, Mark Logic, *e.t.c.*

1980s – Object-Oriented Data Model



Application Code

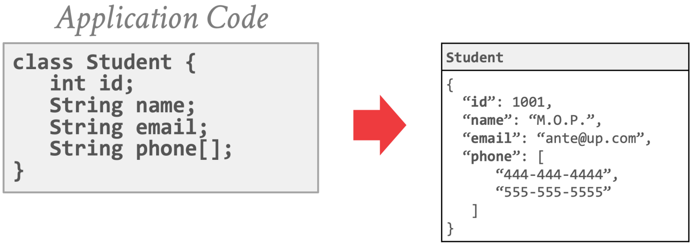
```
class Student {
   int id;
   String name;
   String email;
   String phone[];
}
```

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

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

Relational Schema

1980s – Object-Oriented Data 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”
   ]
}
```

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 Decomposition Storage Model (*i.e.*, columnar storage)

NETEZZA
PARACCEL
monetdb
Greenplum
DATAllegro
VERTICA

2000s – NoSQL Systems

- Focus on high-availability & high-scalability:
  - Schema-less (*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
ScaleArc
SAP HANA
VOLTDB
NUODB
MEMSQL
HyPer
JustOne DB

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.

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.

snowflake
FAUNA
amazon REDSHIFT
amazon DynamoDB
xeround The Cloud Database
Amazon Aurora
Microsoft
Google Spanner

2010s – Specialized Systems

- Shared-disk DBMSs
- Embedded DBMSs
- Times Series DBMS
- Multi-Model DBMSs
- Blockchain DBMSs

2010s – Specialized Systems

AXIBASE
ClickHouse
BIGCHAINDB
levelDB
BuntDB
LEANXCALE
[tile]DB
GoshawkDB
CDB
Trafodion
ThinkSQL RDBMS
FlureeDB
accumulo
Stardog
BLAZINGDB
TiDB
STARCOUNTER
BAGRI
tdb
SequoiaDB
CRATE.IO
WIREDTIGER
YugaByte
gun
kinetica
actorDB
CUBRID
presto
TILE38
mldb.ai
SQREAM
vectorwise
Spark
AntidoteDB
SYLVA
piladb
ArangoDB
quasardb
HIVE
MAPD
OrientDB
RexDB
VITESSE DATA
Hortonworks STINGER
MariaDB
QuestDB
PERFORMANCE DISTILLED
HarperDB
FATHOMDB
INFINITEGRAPH
SCYLLA.
HYPERTABLE
paradigm4
influxdb
citusdata
IMDB
RocksDB
TIMESCALE

Conclusion

Parting Thoughts

- There are many innovations that come from both industry and academia.
  - Lots of ideas start in academia but few build complete DBMSs to verify them.
  - IBM was the vanguard during 1970-1980s but now there is no single trendsetter.
  - The era of cloud systems has begun.
- The relational model has won for operational databases.

Next Class

- Recap of topics covered in the CS 4420/6422
  - Storage Management
  - Access Methods
  - Query Execution
- Submit exercise sheet #1 via Gradescope.