15-721 DATABASE SYSTEMS

Lecture #01 – Course Introduction & History of Database Systems

Andy Pavlo // Carnegie Mellon University // Spring 2016

WHY YOU SHOULD TAKE THIS COURSE

WHY YOU SHOULD TAKE THIS COURSE

WHY YOU SHOULD TAKE THIS COURSE

WHY YOU SHOULD TAKE THIS COURSE

WHY YOU SHOULD TAKE THIS COURSE

friendster

facebook

WHY YOU SHOULD TAKE THIS COURSE

Databases are **still** a hot field.

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.

TODAY'S AGENDA

Course Outline

History of Database Systems

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
→ North American street skills

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
Join Algorithms
Logging & Recovery Methods
Query Optimization, Execution, Compilation
New Storage Hardware

BACKGROUND

I assume that you have already taken an intro course on databases (e.g., 15-415/615).

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.

BACKGROUND

All projects will be written in C++11.

You will be working on a large code-base that contains portions of Postgres that we (CMU) did not write.

Be prepared to debug a multi-threaded program.

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

Immediately after 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

→ Relationship advice

TEACHING ASSISTANTS

Head TA: Joy Arulraj

→ Main contact for questions about programming projects.

Thug TA: Mu Li

→ Helping out with logistics and grading scripts.

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 (two sentences).
→ System used and how it was modified (one sentence).
→ Workloads evaluated (one sentence).

Submission Form:
http://cmudb.io/15721-s16-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 **Peloton**.

→ In-memory, hybrid DBMS based on Postgres
→ Modern code base (C++11, Multi-threaded)

We will provide more details about how to get started with the first project next class.

PROGRAMMING PROJECTS

Do all development on your local machine.
→ Peloton only builds on Linux.
→ We will provide a Vagrant configuration.

Do all benchmarking using DB Lab cluster.
→ We will provide login details later in semester.

Generous hardware donation from **MemSQL**.

PROJECTS #1 AND #2

We will provide you with test cases and scripts for the first two programming projects.

Project #1 will be completed individually.

Project #2 will be done in a group of **three**.
→ 30 people in the class
→ 10 groups of 3 people

☠ 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.

PROJECT #3

Each group 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 can't pick same idea).

You don't have to pick a topic until after Spring Break.

We will provide sample project topics.

PROJECT #3

Project deliverables:
→ Proposal
→ Project Update
→ Code Review
→ Final Presentation
→ Code Drop

PROJECT #3 – PROPOSAL

Five minute presentation to the class that discusses the high-level topic.

Each proposal must discuss:
→ What files you will need to modify.
→ How you will test whether your implementation is correct.
→ What workloads you will use for your project.

PROJECT #3 – STATUS UPDATE

Five minute presentation to update the class about the current status of your project.

Each presentation should include:
→ Current development status.
→ Whether anything in your plan has changed.
→ Any thing that surprised you.

PROJECT #3 – CODE REVIEW

Each group will be paired with another group and provide feedback on their code.

Grading will be based on participation.

PROJECT #3 – FINAL PRESENTATION

10 minute presentation on the final status of your project during the scheduled final exam.

You'll want to include any performance measurements or benchmarking numbers for your implementation.

Demos are always hot too...

PROJECT #3 – CODE DROP

A project is **not** considered complete until:

→ The code can merge into the master branch without any conflicts.

→ All comments from code review are addressed.

→ The project includes test cases that correctly verify that implementation is correct.

→ The group provides documentation in both the source code and in separate Markdown files.

FINAL EXAM

Written long-form examination on the mandatory readings and topics discussed in class. Closed notes.

Will be held on the last day of class (Wednesday April 27th) in this room.

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 (10%)
Project #1 (10%)
Project #2 (20%)
Project #3 (40%)
Final Exam (20%)
Extra Credit (+10%)

COURSE MAILING LIST

On-line Discussion through Piazza:
http://piazza.com/cmu/spring2016/15721

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 ABRIDGED
HISTORY OF DATABASES

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

WHAT'S REALLY NEW WITH NEWSQL?
Under Submission, 2015.

HISTORY REPEATS ITSELF

Old database issues are still relevant today.

The “SQL vs. NoSQL” debate is reminiscent of “Relational vs. CODASYL” debate.

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

1960S – IBM IMS

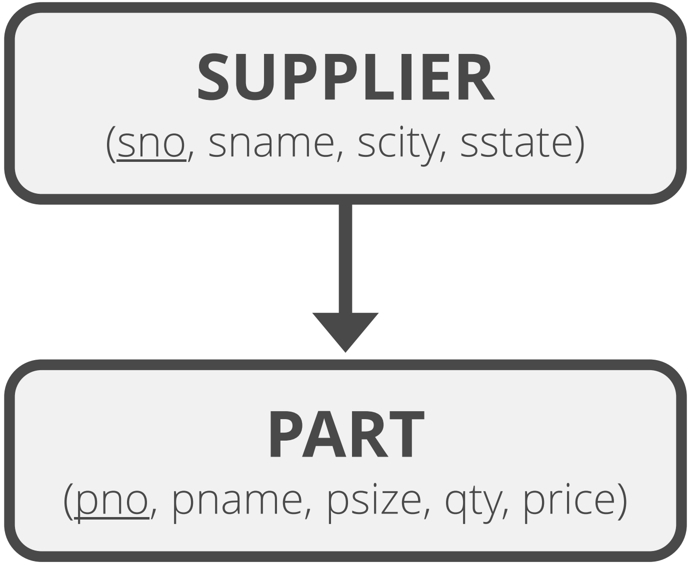
First database system developed to keep track of purchase orders for Apollo moon mission.

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

HIERARCHICAL DATA MODEL

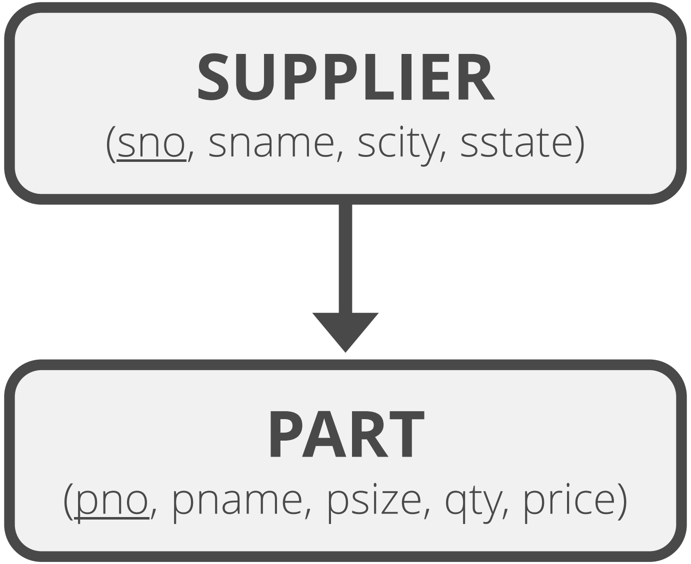
Schema

Instance



HIERARCHICAL DATA MODEL

Schema

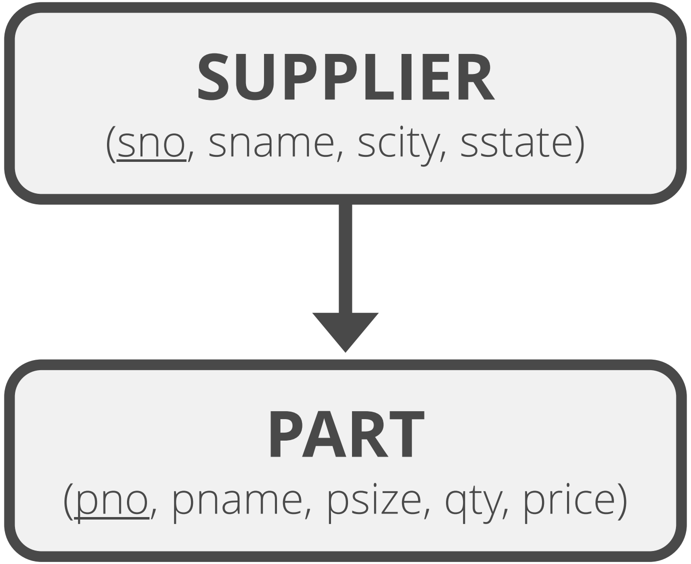


Instance

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

HIERARCHICAL DATA MODEL

Schema



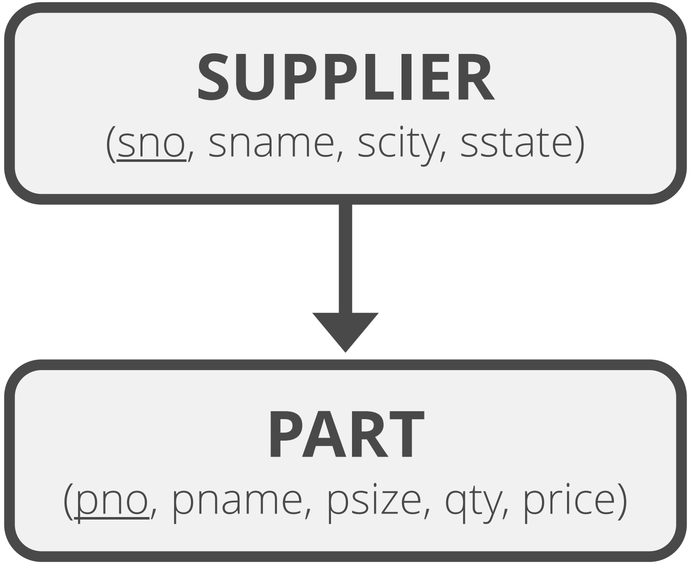
Instance

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

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

(sno, sname, scity, sstate)

PART

(pno, pname, psize, qty, price)

				parts
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

(pno, pname, psize, qty, price)

pno	pname	psize	qty	price
999	Batteries	Large	14	$99

1970s – CODASYL

COBOL people got together and proposed a standard for how programs will access a database. Lead by Charles Bachman.

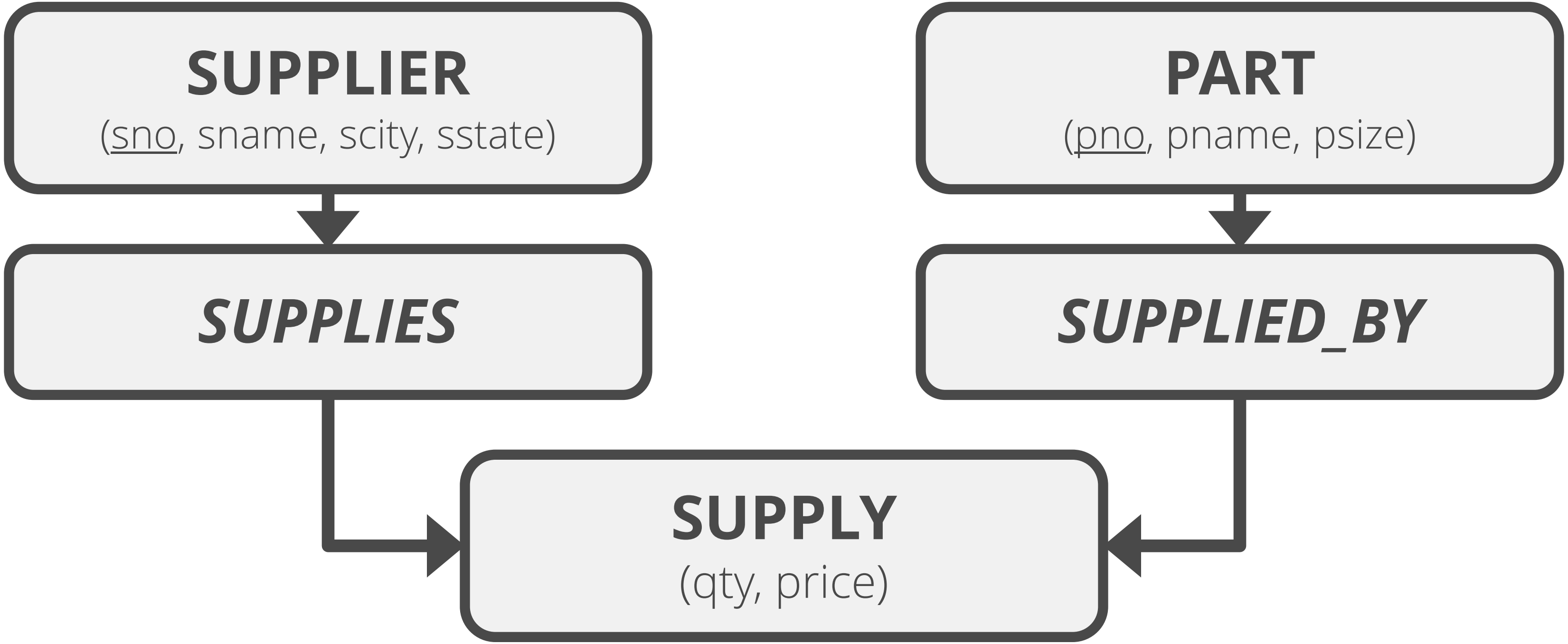
→ Network data model.

→ Tuple-at-a-time queries.

Bachman

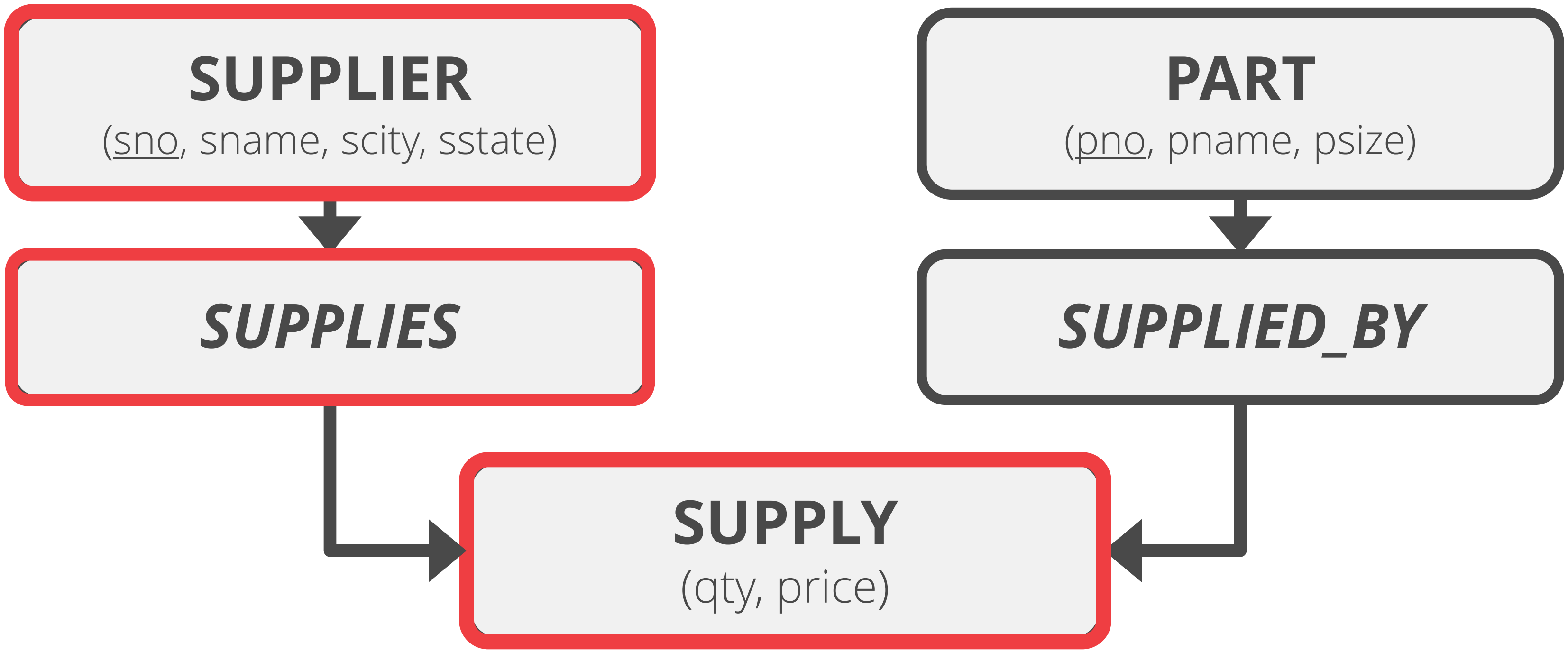
NETWORK DATA MODEL

Schema



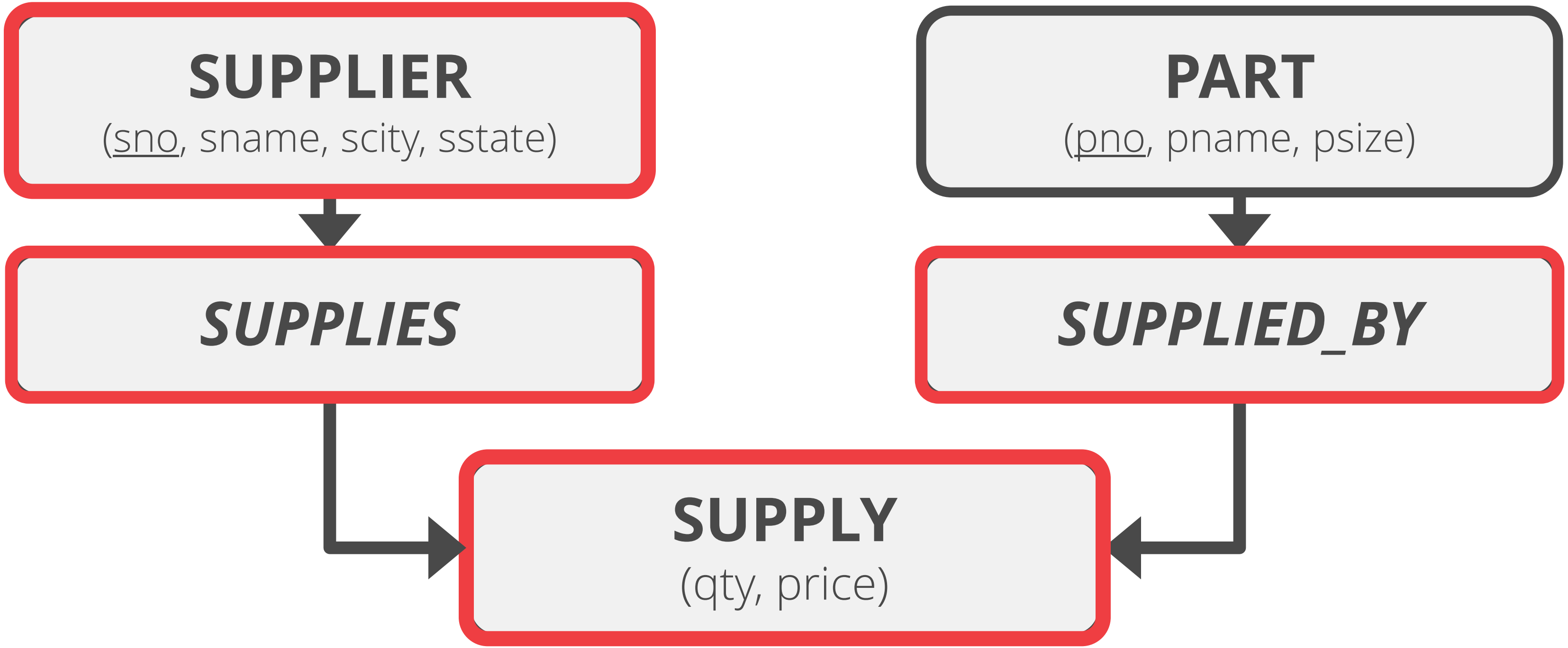
NETWORK DATA MODEL

Schema



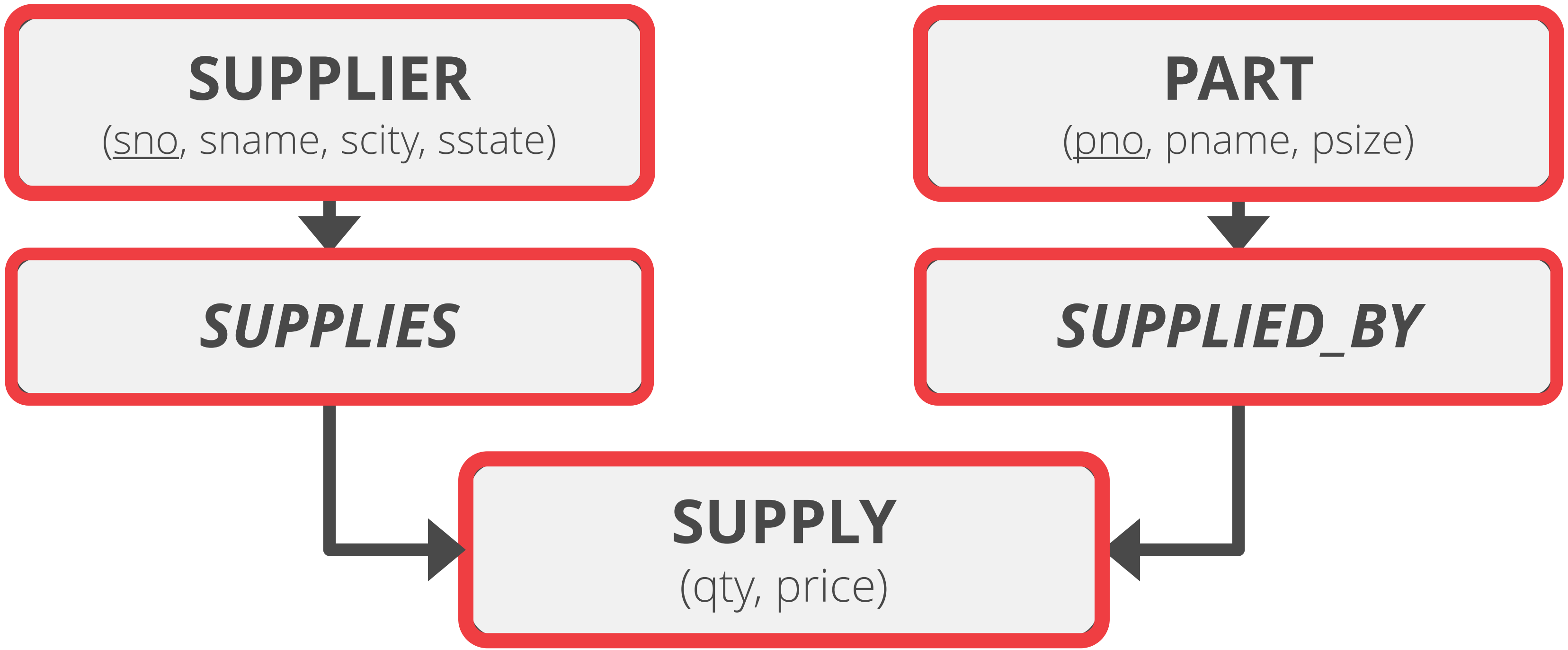
NETWORK DATA MODEL

Schema

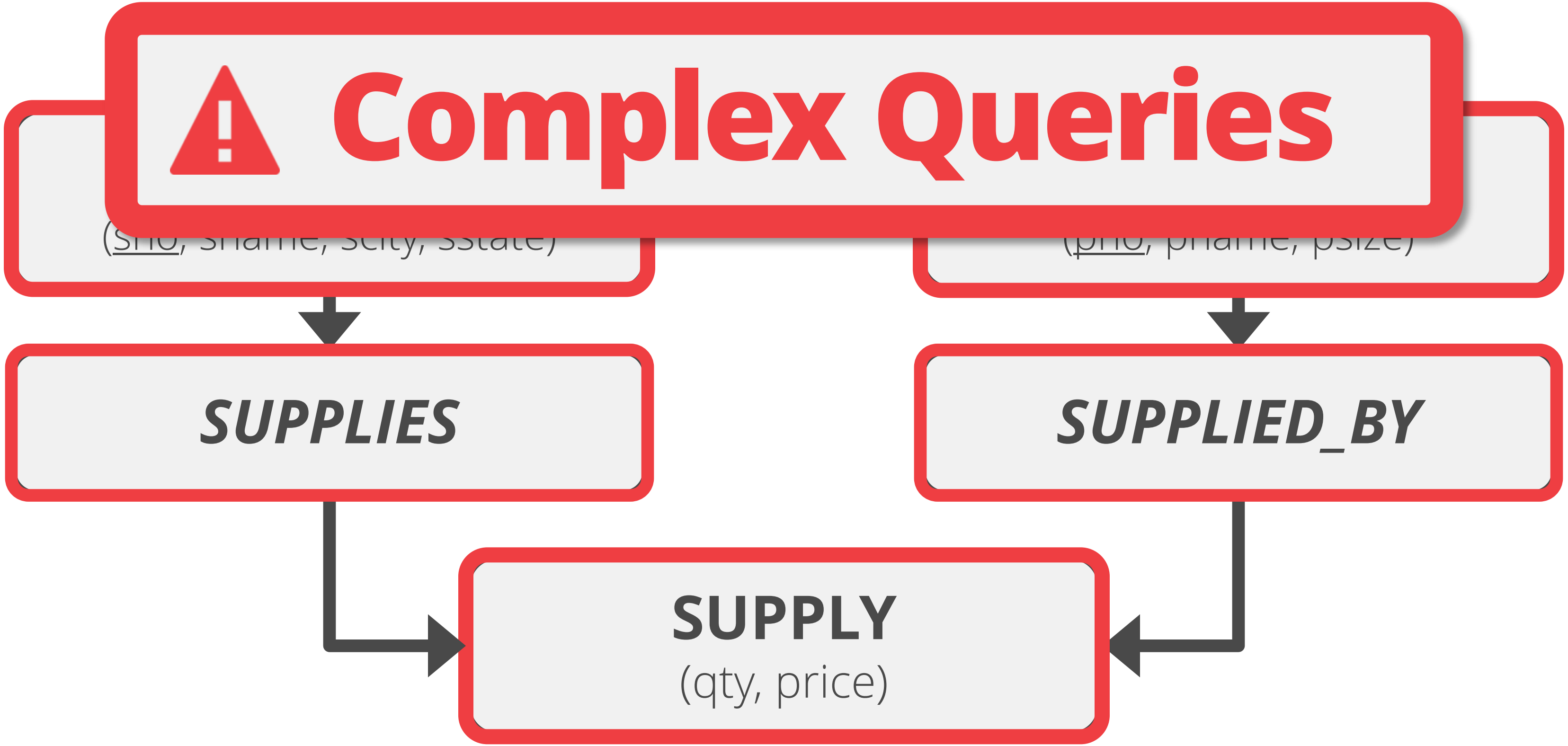


NETWORK DATA MODEL

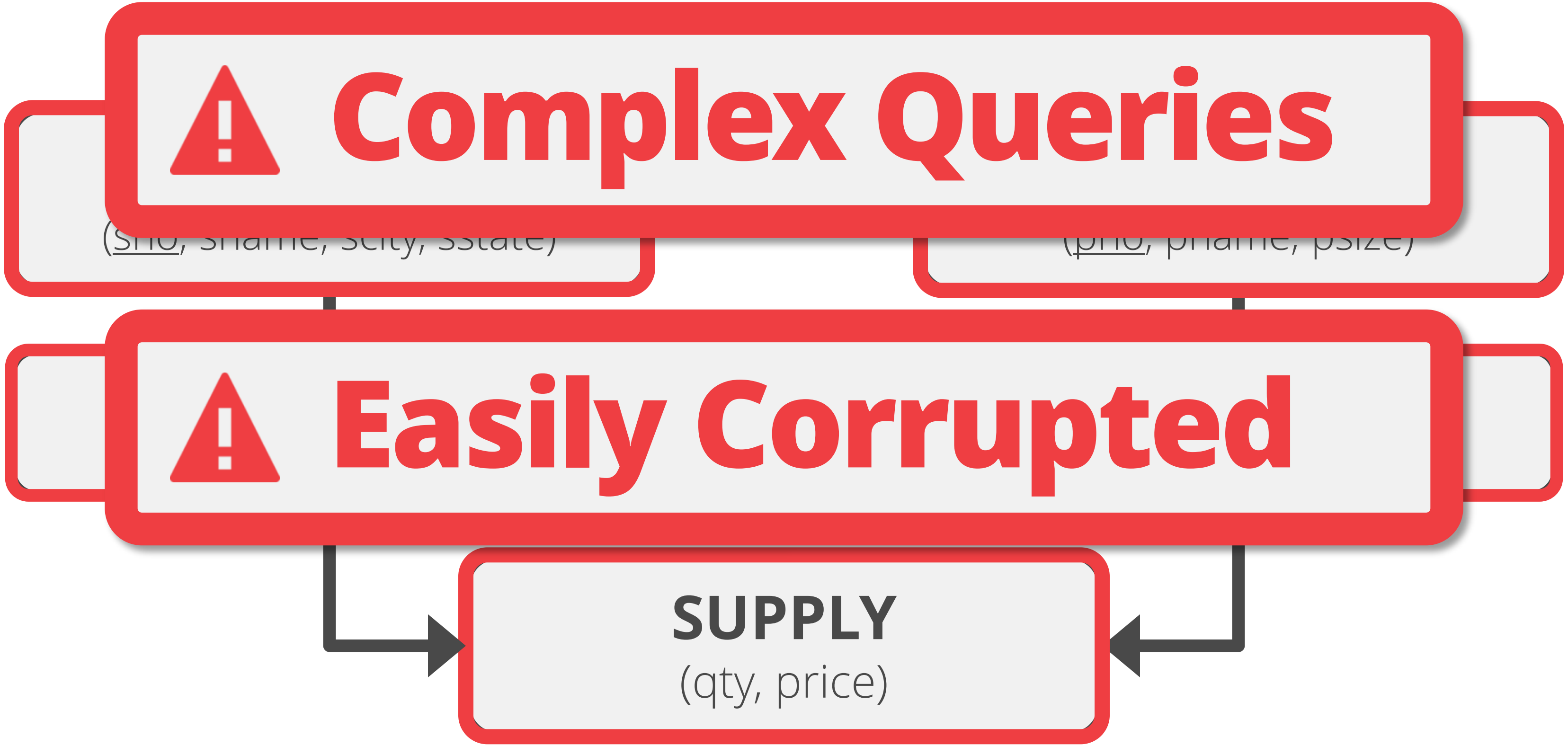
Schema



NETWORK DATA MODEL

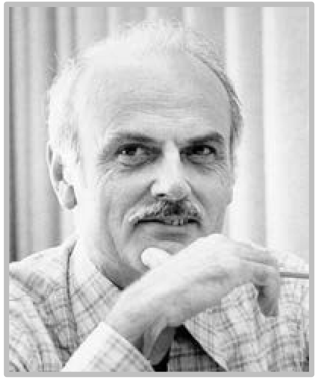


NETWORK DATA MODEL



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.



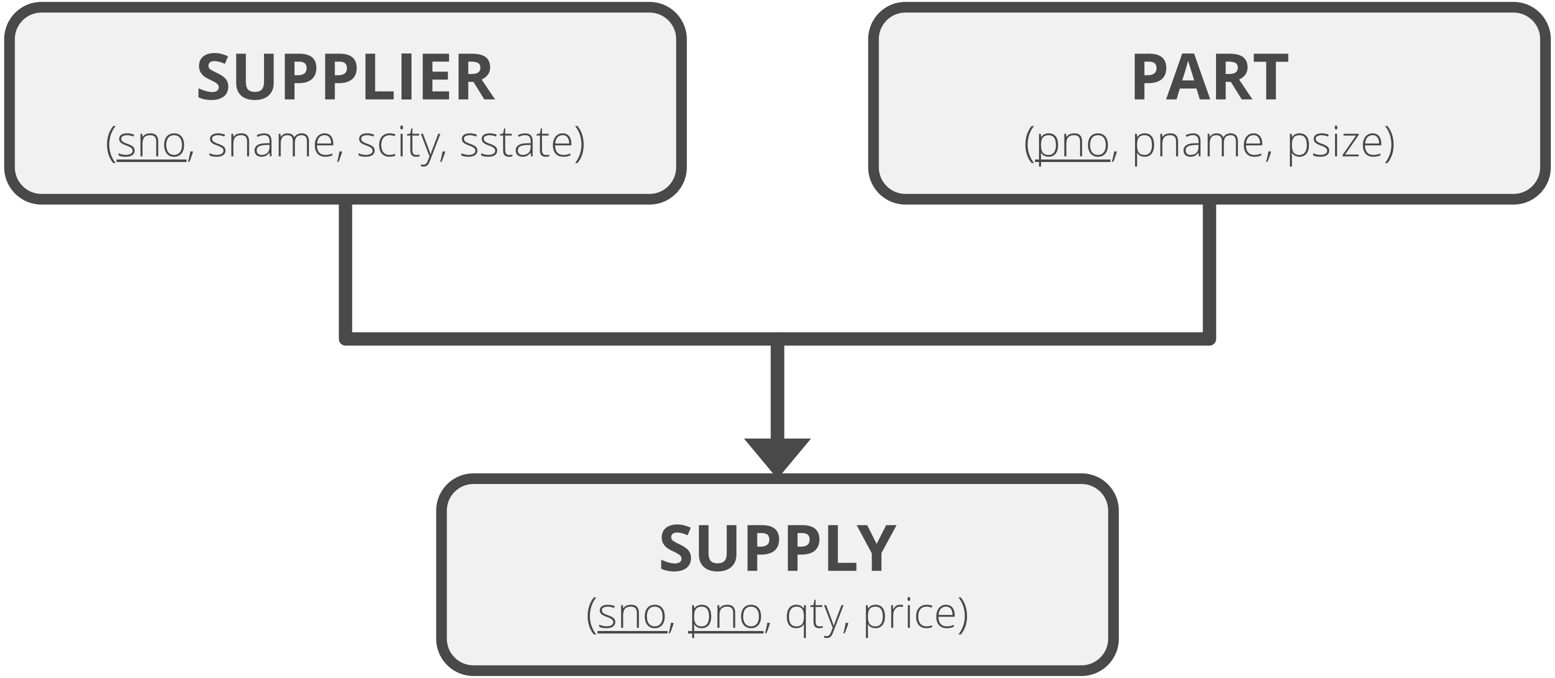
Codd

Database abstraction to avoid this maintenance:

→ Store database in simple data structures.
→ Access data through high-level language.
→ Physical storage left up to implementation.

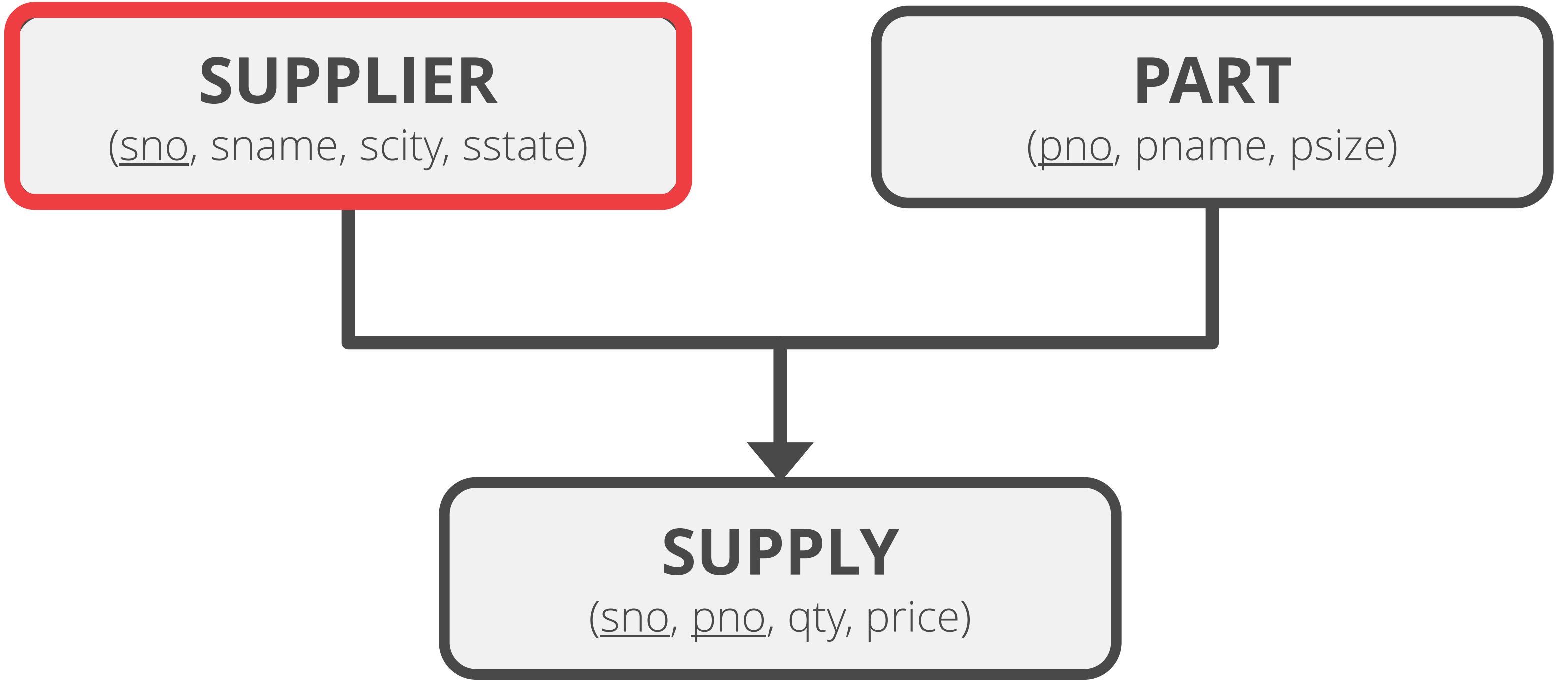
RELATIONAL DATA MODEL

Schema



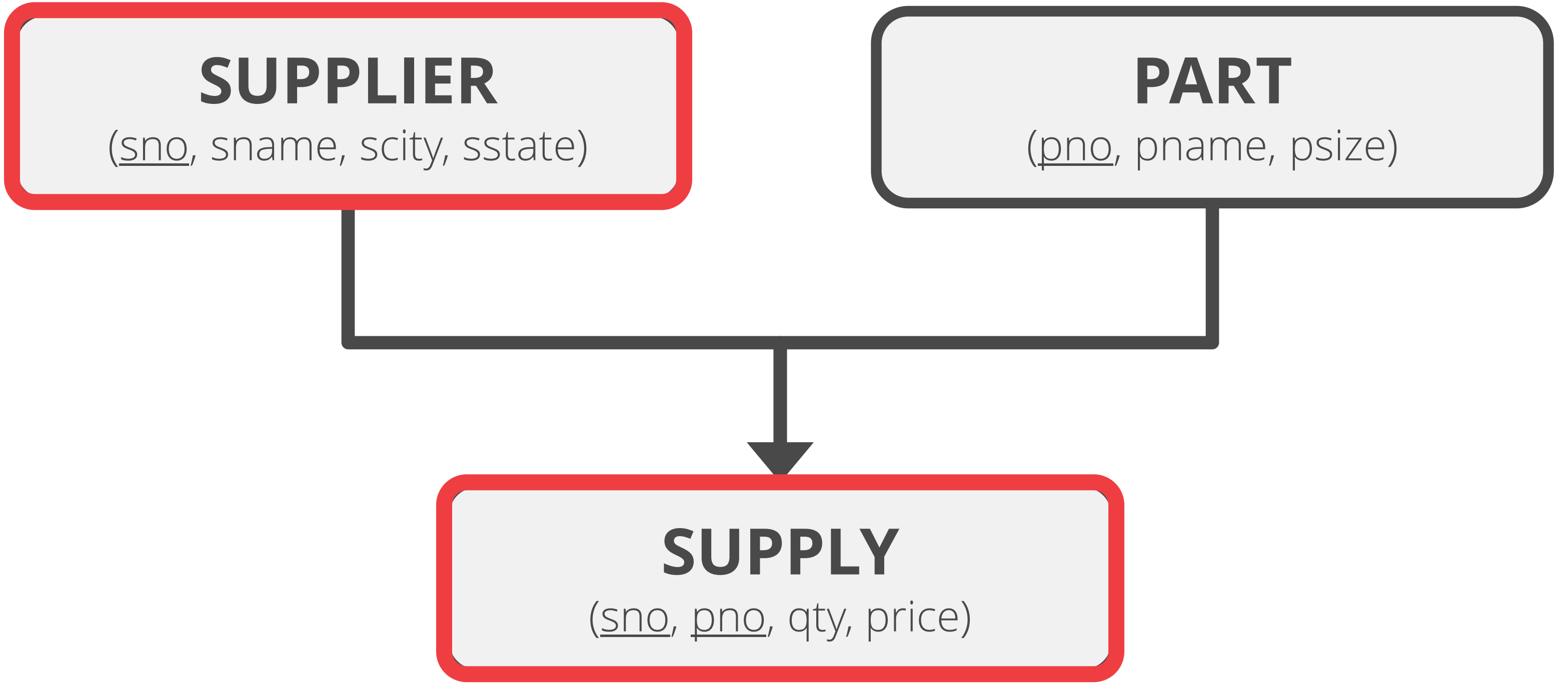
RELATIONAL DATA MODEL

Schema



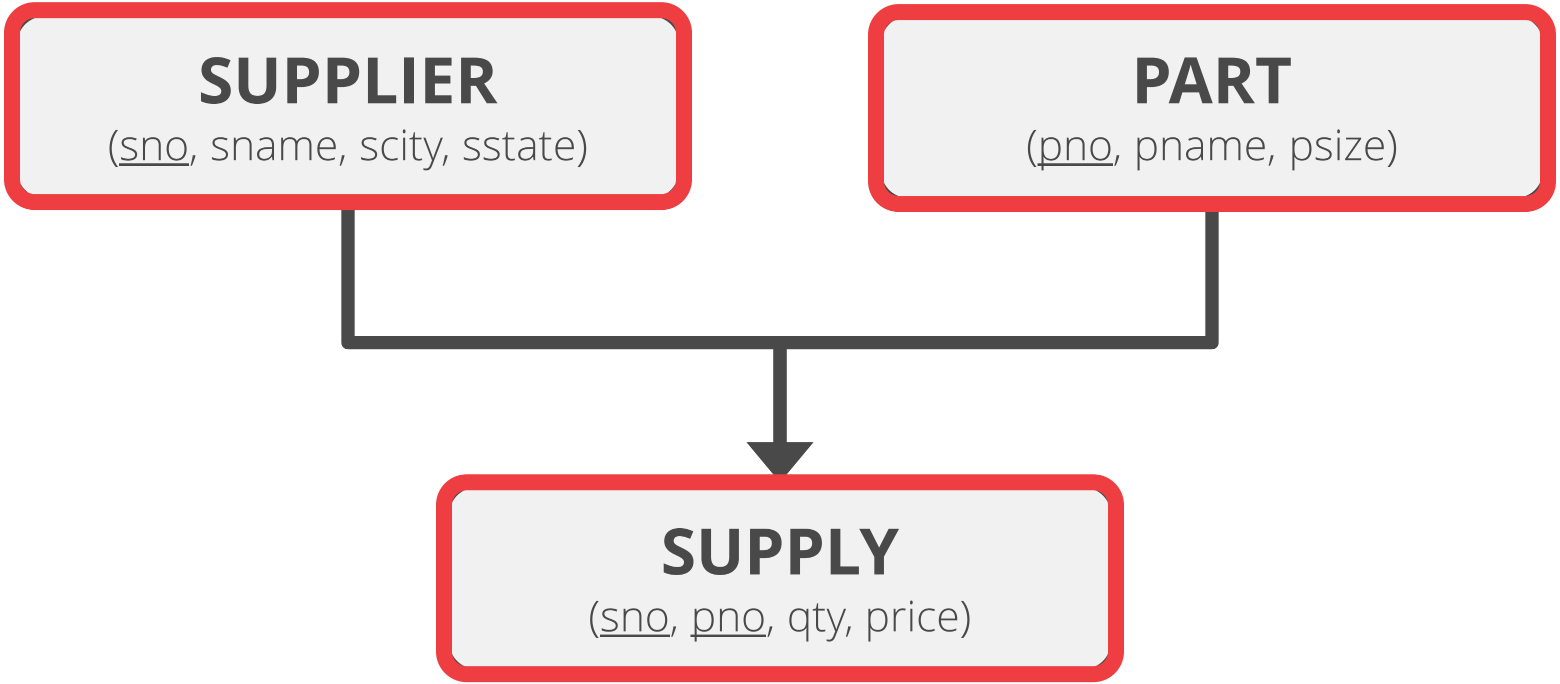
RELATIONAL DATA MODEL

Schema



RELATIONAL DATA MODEL

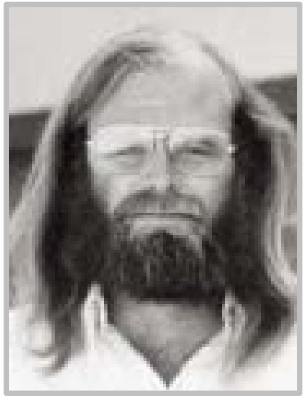
Schema



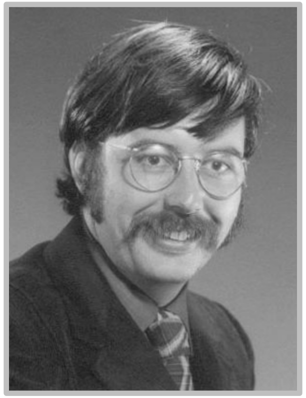
1970s – RELATIONAL MODEL

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.
→ SQL becomes the standard.

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[];
}
```

OBJECT-ORIENTED MODEL

Application Code

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

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[];
}
```

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

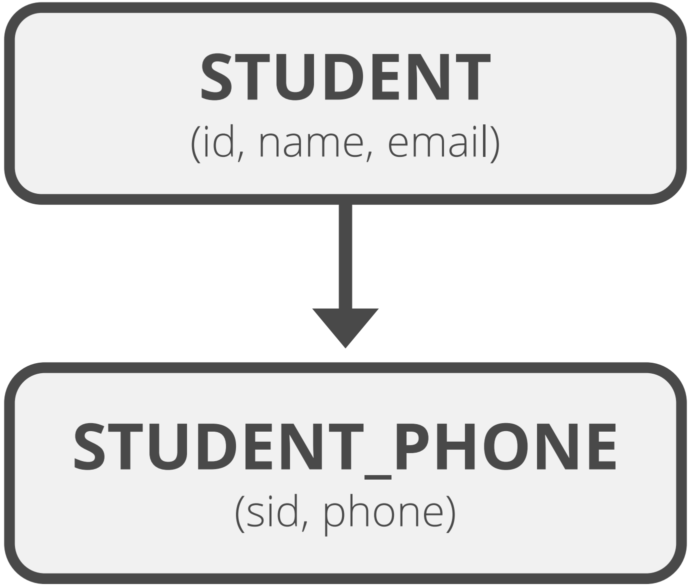
Application Code

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

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

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

Relational Schema



OBJECT-ORIENTED MODEL

Application Code

```
class Student {
   int id;
   String name;
   String email;
   String 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

⚠ **Complex Queries**

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

```
  "email": "ante@up.com",
  "phone": [
      "444-444-4444",
      "555-555-5555"
   ]
}
```

OBJECT-ORIENTED MODEL

Complex Queries

No Standard API

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

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

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.

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)

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 **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
→ All closed-source (as of 2016).

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 Google is current trendsetter.
→ Oracle borrows ideas from anybody.

The relational model has won for operational databases.

NEXT CLASS

Disk vs. In-Memory DBMSs

Project #1 Discussion

Reminder: First reading review is due at 12:00pm on Wednesday January 13th.