CSE544
Data Management
Lecture 1: Introduction,
Relational Data Model
Course Staff

• Instructor: Dan Suciu
  – Office hours: Mondays, 2:30-3:20

• TA: Kyle Deeds
  – Office hours: Fridays, 11:30-12:20
Goals of the Class

• **Relational Data Model**
  – Data models
  – Data independence
  – Declarative query language.

• **Relational Database Systems**
  – Storage
  – Query execution
  – Query optimization
A Note for Non-Majors

- For the Data Science option: take 414
- For the Advanced Data Science option: take 544
- 544 is an advanced class, not an introduction.
- Unsure? Look at the short quiz on the website.
Readings

- **Lecture notes (the slides)**
  - Posted on class website after each lecture

- **Background**

- **Paper reviews**
  - Mix of old seminal papers and new papers
  - Papers are available on class website
Class Resources

Website: lectures, assignments
  • Lectures 1/17 and 2/21 moved to Friday

Canvas: zoom, videos

Ed: discussion board
Other Resources

• **Database course at CMU**
  – Low level: storage, transactions

• **Database theory course at Berkeley**
  – Theory: complexity, information theory

• Our course is in between
Evaluation

• Assignments 40%
• Reviews 10%
• Project 40%
• Intangibles 10%
Assignments – 40%

• **HW1**: Local DBMS (postgres) *posted!*  
• **HW2**: Cloud-based DBMS (snowflake)  
• **HW3**: Query Execution and SimpleDB  
• **HW4**: Datalog

• See course calendar for deadlines  
• Late assignments w/ **very** valid excuse
Paper reviews – 10%

• Recommended length: ½ page – 1 page
  – Summary of the main points of the paper
  – Critical discussion of the paper
  – R1 due on 1/10

• Grading: credit/partial-credit/no-credit

• Submit review before the lecture
Project – 40%

Work alone, or in a team of 2-3 students

Topic:
• Come up with your own or choose from our list
• Best: related to your research
• Must be about databases / data management
• Must involve some significant engineering
• Open ended
Project  – 40%

Dates posted on the calendar page:

- **P1**: Form groups
- **P2**: Project proposal
- **P3**: Milestone report
- **P4**: Poster presentation Wed. March 6
- **P5**: Project final report
Intangibles 10%

• Class participation

• Exceptionally good reviews, or homework, or project

• Etc, etc
How to Turn In

- Homeworks: gitlab
- Project: gitlab
- Reviews: google forms
Now onward to the world of databases!
Database

- **Database** = collection of files storing inter-related data about real world entities and relationships.

- **Entities**: e.g. products, suppliers, customers, employees, warehouses

- **Relationships**: e.g. suppliers-products, customer-products, employee-manages-employee
Database Management System

- **DBMS**: a software system designed to provide data management services

- **Examples**
  - Oracle, DB2 (IBM), SQL Server (Microsoft)
  - Snowflake, Redshift
  - PostgreSQL, Duckdb, MySQL, Sqlite
Database Example

A database of products and suppliers:

• **Product**: has a name, a price, a color

• **Supplier**: has a name, the products it supplies, city
Flat File Strawman

- Store data in csv files
- Manage your data in python
Flat File Strawman

• Store data in csv files
• Manage your data in python

**Product**\(\text{name, price, color}\)

- iPhone, 599, gray
- iPhone, 999, black
- Gizmo, 399, blue
- Pizza, 29, red
Flat File Strawman

- Store data in csv files
- Manage your data in python

**Product** (name, price, color)

- iPhone, 599, gray
- iPhone, 999, black
- Gizmo, 399, blue
- Pizza, 29, red

**Supplier** (name, product, city)

- ACME, iPhone, Seattle
- Walmart, iPhone, Renton
- Costco, Pizza, Seattle
Flat File Strawman

• Store data in csv files
• Manage your data in python

Product (name, price, color)

- iPhone, 599, gray
- iPhone, 999, black
- Gizmo, 399, blue
- Pizza, 29, red
Flat File Strawman

• Store data in csv files
• Manage your data in python

Product(name, price, color)

Find the price of Gizmo:

- iPhone, 599, gray
- iPhone, 999, black
- Gizmo, 399, blue
- Pizza, 29, red
Flat File Strawman

- Store data in csv files
- Manage your data in python

<table>
<thead>
<tr>
<th>Product (name, price, color)</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone, 599, gray</td>
</tr>
<tr>
<td>iPhone, 999, black</td>
</tr>
<tr>
<td>Gizmo, 399, blue</td>
</tr>
<tr>
<td>Pizza, 29, red</td>
</tr>
</tbody>
</table>

Find the price of Gizmo:

```python
with open('product.csv') as f:
    r = csv.reader(f, delimiter=',
    for t in r:
        if t[0] == "Gizmo":
            print(t[1])
```
Issues with Flat Files

Need to implement many generic tasks:
• Data integrity
• Efficient implementation
• Concurrency, durability
• Etc.
Flat Files: Data Integrity

• Price should be a number

• Supplier names should be unique

• Suppliers may supply >1 products

• Each supplier in a single city
Flat Files: Implementation

• How do we update/insert/delete?

• Data may be larger than main memory

• A query may traverse multiple files

• Data may be distributed
Flat Files: Concurrency, Durability

• What if multiple applications touch the data?

• What if the system crashes in the middle of an update?
Database Management System

A software system designed to provide data management services:

- Enforce integrity constraints
- Evaluate queries efficiently
- Handle concurrency, recovery
- ...

Important Terminology

- Architecture
- Workloads
Architecture: Single Client

E.g. data analytics

Application and database on the same computer
E.g. sqlite, postgres
Two-tier Architecture
Client-Server

E.g. accounting, banking, ...

Connection:
ODBC, JDBC

Database server
E.g. Oracle, DB2, ...

Applications:
Java
Three-tier Architecture

E.g. Web commerce

Application server
E.g. java, python, ruby-on-rails

Connection
(ODBC, JDBC)

Database server
E.g. Oracle

Browser

http
Cloud Databases

E.g. large-scale analytics or...

ODBC, JDBC

http

Sharded database
E.g. Spark, Snowflake

...social networks

App server
Types of Workloads

• **OLTP** – online transaction processing
  – This is how data is generated
  – Single point read/update
  – Concurrency, transactions

• **OLAP** – online analytics processing
  – Complex queries w/ aggregates
  – Data often comes from OLTP
  – Updates are very rare
Summary

• **DBMS**: store and manage your data

• Everywhere:
  – On your phone: sqlite
  – On your laptop: sqlite, postgres, duckdb,…
  – In the cloud: snowflake/redshift/bigquery

• All use the **Relational Data Model**
Relational Data Model
Data Model

Mathematical formalism that models data. Several exist:
• Relational
• Key/value / Document / XML / Json
• Graph
• Arrays / matrices / tensors
• Hierarchical, network…
Data Model

Mathematical formalism that models data. Several exist:

• Relational
• Key/value / Document / XML / Json
• Graph
• Arrays / matrices / tensors
• Hierarchical, network…
Data Model

Mathematical formalism that models data. Several exist:

- Relational
- Key/value / Document / XML / Json
- Graph
- Arrays / matrices / tensors
- Hierarchical, network…
Data Model

Mathematical formalism that models data. Several exist:

• Relational
• Key/value / Document / XML / Json
• Graph
• Arrays / matrices / tensors
• Hierarchical, network…
Data Model

Mathematical formalism that models data. Several exist:

- Relational
- Key/value / Document / XML / Json
- Graph
- Arrays / matrices / tensors
- Hierarchical, network…
Data Model

Mathematical formalism that models data. Several exist:

• Relational
• Key/value / Document / XML / Json
• Graph
• Arrays / matrices / tensors
• Hierarchical, network…
Relational Data Model

- **Database**: collection of relations
- **Relation** (aka Table): set of tuples
- **Tuple** (aka row, record): \( t \in \text{Dom}_1 \times \cdots \times \text{Dom}_n \)
## Relational Data Model

**Product**\( (\text{name, price, color}) \)

<table>
<thead>
<tr>
<th>name</th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>599</td>
<td>Gray</td>
</tr>
<tr>
<td>iPhone</td>
<td>999</td>
<td>Black</td>
</tr>
<tr>
<td>Gizmo</td>
<td>399</td>
<td>Blue</td>
</tr>
<tr>
<td>Pizza</td>
<td>29</td>
<td>red</td>
</tr>
</tbody>
</table>
Relational Data Model

**Product** (name, price, color)

<table>
<thead>
<tr>
<th>name</th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>599</td>
<td>Gray</td>
</tr>
<tr>
<td>iPhone</td>
<td>999</td>
<td>Black</td>
</tr>
<tr>
<td>Gizmo</td>
<td>399</td>
<td>Blue</td>
</tr>
<tr>
<td>Pizza</td>
<td>29</td>
<td>red</td>
</tr>
</tbody>
</table>

**Supplier** (name, product, city)

<table>
<thead>
<tr>
<th>name</th>
<th>product</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>iPhone</td>
<td>Seattle</td>
</tr>
<tr>
<td>Walmart</td>
<td>iPhone</td>
<td>Renton</td>
</tr>
<tr>
<td>Costco</td>
<td>Pizza</td>
<td>Seattle</td>
</tr>
</tbody>
</table>
Discussion

• **Rows** in a relation:
  – Ordering immaterial (a relation is a set)
  – All rows are distinct – **sets**
  – Query answers may have duplicates – **bags**
  – Relation is flat: no lists, collections, arrays in a relation

• **Attributes** of a record:
  – Ordering is immaterial (mostly…)
  – Applications refer to columns by their names

• **Domain** of each column is a primitive type
Instance v.s. Schema

• **Relation schema:**
  – Relation name
  – Name of each field/column/attribute
  – Domain/type of each field

• **Relational instance:**
  – A set of records
Primary Key

• A key is an attribute, or a set of attributes whose value uniquely identify the record

• There may be more than one: we choose one that we call primary key
Primary Key

Product\((name, price, color)\)

<table>
<thead>
<tr>
<th>name</th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>599</td>
<td>Gray</td>
</tr>
<tr>
<td>iPhone</td>
<td>999</td>
<td>Black</td>
</tr>
<tr>
<td>Gizmo</td>
<td>399</td>
<td>Blue</td>
</tr>
<tr>
<td>Pizza</td>
<td>29</td>
<td>red</td>
</tr>
</tbody>
</table>

Supplier\((name, product, city)\)

<table>
<thead>
<tr>
<th>name</th>
<th>product</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>iPhone</td>
<td>Seattle</td>
</tr>
<tr>
<td>Walmart</td>
<td>iPhone</td>
<td>Renton</td>
</tr>
<tr>
<td>Costco</td>
<td>Pizza</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

Primary key = \(name\)
### Primary Key

**Product** *(name, price, color)*

<table>
<thead>
<tr>
<th>name</th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>599</td>
<td>Gray</td>
</tr>
<tr>
<td>iPhone</td>
<td>999</td>
<td>Black</td>
</tr>
<tr>
<td>Gizmo</td>
<td>399</td>
<td>Blue</td>
</tr>
<tr>
<td>Pizza</td>
<td>29</td>
<td>red</td>
</tr>
</tbody>
</table>

**Supplier** *(name, product, city)*

<table>
<thead>
<tr>
<th>name</th>
<th>product</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>iPhone</td>
<td>Seattle</td>
</tr>
<tr>
<td>Walmart</td>
<td>iPhone</td>
<td>Renton</td>
</tr>
<tr>
<td>Costco</td>
<td>Pizza</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

Primary key = ???

Primary key = name
Primary Key

**Product** (name, price, color)

<table>
<thead>
<tr>
<th>name</th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>599</td>
<td>Gray</td>
</tr>
<tr>
<td>iPhone</td>
<td>999</td>
<td>Black</td>
</tr>
<tr>
<td>Gizmo</td>
<td>399</td>
<td>Blue</td>
</tr>
<tr>
<td>Pizza</td>
<td>29</td>
<td>red</td>
</tr>
</tbody>
</table>

Primary key = name, price color

**Supplier** (name, product, city)

<table>
<thead>
<tr>
<th>name</th>
<th>product</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>iPhone</td>
<td>Seattle</td>
</tr>
<tr>
<td>Walmart</td>
<td>iPhone</td>
<td>Renton</td>
</tr>
<tr>
<td>Costco</td>
<td>Pizza</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

Primary key = name
### Primary Key

#### Product

<table>
<thead>
<tr>
<th>pid</th>
<th>name</th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>p001</td>
<td>iPhone</td>
<td>599</td>
<td>Gray</td>
</tr>
<tr>
<td>p002</td>
<td>iPhone</td>
<td>999</td>
<td>Black</td>
</tr>
<tr>
<td>p003</td>
<td>Gizmo</td>
<td>399</td>
<td>Blue</td>
</tr>
<tr>
<td>p004</td>
<td>Pizza</td>
<td>29</td>
<td>red</td>
</tr>
</tbody>
</table>

#### Supplier

<table>
<thead>
<tr>
<th>name</th>
<th>product</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>iPhone</td>
<td>Seattle</td>
</tr>
<tr>
<td>Walmart</td>
<td>iPhone</td>
<td>Renton</td>
</tr>
<tr>
<td>Costco</td>
<td>Pizza</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

Primary key = \(\text{pid}\)

Primary key = \(\text{name}\)

Good practice to have a single attribute as primary key
Foreign Key

• An attribute whose values are keys of another relation is called a **foreign key**
• Also called “semantic pointer”
Foreign Key

**Product**\((\text{pid}, \text{name}, \text{price}, \text{color})\)  

<table>
<thead>
<tr>
<th>pid</th>
<th>name</th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>p001</td>
<td>iPhone</td>
<td>599</td>
<td>Gray</td>
</tr>
<tr>
<td>p002</td>
<td>iPhone</td>
<td>999</td>
<td>Black</td>
</tr>
<tr>
<td>p003</td>
<td>Gizmo</td>
<td>399</td>
<td>Blue</td>
</tr>
<tr>
<td>p004</td>
<td>Pizza</td>
<td>29</td>
<td>red</td>
</tr>
</tbody>
</table>

**Supplier**\((\text{name}, \text{product}, \text{city})\)  

<table>
<thead>
<tr>
<th>name</th>
<th>product</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>iPhone</td>
<td>Seattle</td>
</tr>
<tr>
<td>Walmart</td>
<td>iPhone</td>
<td>Renton</td>
</tr>
<tr>
<td>Costco</td>
<td>Pizza</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

**Product** is ambiguous. Why?
### Foreign Key

**Product**\((\text{pid, name, price, color})\)  
**Supplier**\((\text{name, pid, city})\)

<table>
<thead>
<tr>
<th>pid</th>
<th>name</th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>p001</td>
<td>iPhone</td>
<td>599</td>
<td>Gray</td>
</tr>
<tr>
<td>p002</td>
<td>iPhone</td>
<td>999</td>
<td>Black</td>
</tr>
<tr>
<td>p003</td>
<td>Gizmo</td>
<td>399</td>
<td>Blue</td>
</tr>
<tr>
<td>p004</td>
<td>Pizza</td>
<td>29</td>
<td>red</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>pid</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>p002</td>
<td>Seattle</td>
</tr>
<tr>
<td>Walmart</td>
<td>p002</td>
<td>Renton</td>
</tr>
<tr>
<td>Costco</td>
<td>p004</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

pid is a foreign key
Foreign Key

Product(pid, name, price, color)

<table>
<thead>
<tr>
<th>pid</th>
<th>name</th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>p001</td>
<td>iPhone</td>
<td>599</td>
<td>Gray</td>
</tr>
<tr>
<td>p002</td>
<td>iPhone</td>
<td>999</td>
<td>Black</td>
</tr>
<tr>
<td>p003</td>
<td>Gizmo</td>
<td>399</td>
<td>Blue</td>
</tr>
<tr>
<td>p004</td>
<td>Pizza</td>
<td>29</td>
<td>red</td>
</tr>
</tbody>
</table>

Supplier(name, pid, city)

<table>
<thead>
<tr>
<th>name</th>
<th>pid</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>p002</td>
<td>Seattle</td>
</tr>
<tr>
<td>Walmart</td>
<td>p002</td>
<td>Renton</td>
</tr>
<tr>
<td>Costco</td>
<td>p004</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

pid is a foreign key

What if Walmart sells both iPhones?
Relational Data Model

**Product** *(pid, name, price, color)*

<table>
<thead>
<tr>
<th>pid</th>
<th>name</th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>p001</td>
<td>iPhone</td>
<td>599</td>
<td>Gray</td>
</tr>
<tr>
<td>p002</td>
<td>iPhone</td>
<td>999</td>
<td>Black</td>
</tr>
<tr>
<td>p003</td>
<td>Gizmo</td>
<td>399</td>
<td>Blue</td>
</tr>
<tr>
<td>p004</td>
<td>Pizza</td>
<td>29</td>
<td>red</td>
</tr>
</tbody>
</table>

**Supplier** *(name, city)*

<table>
<thead>
<tr>
<th>name</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>Seattle</td>
</tr>
<tr>
<td>Walmart</td>
<td>Renton</td>
</tr>
<tr>
<td>Costco</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

**Supply** *(pid, name)*

<table>
<thead>
<tr>
<th>pid</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>p002</td>
<td>ACME</td>
</tr>
<tr>
<td>p001</td>
<td>Walmart</td>
</tr>
<tr>
<td>p002</td>
<td>Walmart</td>
</tr>
<tr>
<td>p004</td>
<td>Costco</td>
</tr>
</tbody>
</table>
Summary

Relational data model:
• Data is stored in flat relations
• No prescription of the physical storage
• Access to the data through high-level declarative language:
  – SQL (next lecture)
  – Relational Algebra