CS 133: Databases

Fall 2019
Lec 01 – 09/03
Introduction & Relational Model

Prof. Beth Trushkowsky

Data!

• Data is everywhere
  – Banking, airline reservations
  – Social media, clicking anything on the internet

Goals for Today

• What is a database anyway?

• Important DBMS features
  – and challenges!

• Course logistics

• Relational data model
  – Why it’s great
  – What it looks like (intro to SQL)

So, what is a database?

From the textbook:

• *Database*: a collection of data, typically describing the activities of one or more related organizations

• *Database system, Database Management System (DBMS)*: software designed to assist in maintaining and utilizing large collections of data
DBMS desiderata

- Ask questions (queries) about data
- Add and update data
- Persist the data (keep it around)

E.g., banking application
- Query: What is Alice’s balance?
- Update: Alice deposits $100
- Persist: Alice hopes her money is still there after a power outage...

Sounds easy!

- Store data in text files
  - Accounts separated by newlines
  - Fields separated by commas
- Query: what is Alice’s balance?

Abstracting data management

- Can come up with tricks to optimize a particular query/application
  - End up redoing this work for new apps

Relational DBMS to the rescue

[There should be] a clear boundary between the logical and physical aspects of database management

Edgar F. Codd
Turing award, 1981

Physical Independence
- Applications need not know how data is physically structured and stored
- Instead, have logical data model
- Leave the implementation details and optimization to DBMS
Relational DBMS to the rescue

- **Relational data model**: data is stored in relations
- **Example**: *Banking* info

<table>
<thead>
<tr>
<th>account</th>
<th>branch</th>
<th>name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>Claremont</td>
<td>Alice</td>
<td>200</td>
</tr>
<tr>
<td>67</td>
<td>Claremont</td>
<td>Bob</td>
<td>100000</td>
</tr>
<tr>
<td>78</td>
<td>Pasadena</td>
<td>Carl</td>
<td>987654</td>
</tr>
</tbody>
</table>

- A declarative query language
  - Specify what answers a query should return, but not how the query is executed
  - E.g., SQL, Datalog (subset of Prolog)

**Query**: what is Alice’s balance?
```
SELECT balance
FROM Banking
WHERE name = "Alice";
```

Relational Model: Levels of Abstraction

- **Conceptual/Logical schema**
  - *Students* (sid: string, name: string, login: string, gpa: real)
  - *Courses* (cid: string, cname: string, credits: integer)
  - *Enrolled* (sid: string, cid: string, grade: string)

- **Physical schema**
  - Store the relations as unsorted files
  - Create indexes on Students.sid and Courses.cid

- **External schema** ("views")
  - view each course’s enrollment
    ```
    CREATE VIEW CourseInfo AS
    SELECT cid, COUNT(*) as enrollmnt
    FROM Enrolled
    GROUP BY cid;
    ```

Data Independence

- **Logical data independence**
  - Protected from changes in conceptual schema

- **Physical data independence**
  - Protected from changes in physical schema

Modern DBMS Features

- **Logical data model**
  - We focus on relational in this course
  - May touch on others, e.g., XML, Document
  - Data independence!

- **Declarative language**
  - Queries
  - Updates

- **Persistence**

*But wait, there’s more...*
Concurrent Access

- Banking example: ATM withdrawal pseudocode
  ```
  get balance;
  if balance > amount
    withdraw amount;
    newBalance = balance - amount;
    write balance = newBalance;
  ```

- Alice and Bob share an account.
  - Alice goes to one ATM, withdraws $100
  - Bob goes to another ATM, withdraws $50

- Initial balance = $400
- Final balance?

Example from Jun Yang

Concurrent Access

Alice withdraws $100
```plain
get balance; \$400
if balance > amount
  withdraw amount;
  newBalance = balance - amount;
  write balance = newBalance;
``` 

Bob withdraws $50
```plain
get balance; \$400
if balance > amount
  withdraw amount;
  newBalance = balance - amount;
  write balance = newBalance; \$350
``` 

What can we do?

Final balance = $300!!

Example from Jun Yang

System Failures

- Banking example: balance transfer
  ```
  decrement account X by $100
  increment account Y by $100
  ```

- What if power goes out after first instruction?

- DBMS buffers and updates some data in memory before writing to disk
  - what if power goes out before write to disk?

- Keep a log of updates, undo/redo upon recovery

Example from Jun Yang

Modern DBMS Features (cntd)

- Logical data model
- Declarative language
- Persistence

- Concurrent access
- Fault tolerance
- Performance!
  - Lots of queries
  - Lots of data
Simplified RDBMS Architecture

Course Overview

• Design principles behind DBMS!

• “Bottom-up” order of topics to show role of abstraction and algorithms for efficiency/optimization
  – Physical data organization
  – Relational algebra and SQL
  – Query evaluation and optimization
  – Transactions, concurrency control, recovery
  – Database design

Course Objectives

• Provide a solid background in database management system design principles

• Promote understanding of these principles through hands-on exercises implementing the internals of a relational database management system

• Further develop students' ability to reason about algorithm and software design, optimization, and tradeoffs generally applicable in computer science

Labs: SimpleDB

• Labs 1-4: Implement key features of a (simplified) DBMS in Java
  – Files, Storage
  – Relational Operators
  – Query Optimizer
  – Locking with Transactions

• Lab 5: database design

  Lab 1: Getting started “due” next Wednesday
Grade Components (see syllabus)

• Weekly problem sets 14% 70 pts
• (5) Labs 40% 200 pts
• Midterm 20% 100 pts
• Final 20% 100 pts
• Participation 6% 30 pts

Administrivia

• Course website: https://www.cs.hmc.edu/~beth/courses/cs133/current
  — Syllabus, calendar, lab descriptions
• Textbook: Database Management Systems 3rd Edition, by Ramakrishnan and Gehrke
• Piazza for questions about labs, problem sets, etc.: piazza.com/hmc/fall2019/cs133/home
• Assignment submission
  — Problem sets → Sakai
  — Lab assignments → Gradescope
• Grutors
  — Ivy Liu

The Relational Model

• Many RDBMS vendors, including open-source
  — Oracle
  — MySQL
  — PostgreSQL
  — SQLite
  — DB2
  — SQL Server
  — ...

• We’ll touch on other data models as well

Key Concepts: Relational Model

• Database: collection of relations

• Relation: list of attributes

• Relations have sets of tuples

• Schema (metadata)
  — Specification of how data is to be structured logically
  — Contains attribute types
  — Defined at set-up
Relational Model: Synonyms

<table>
<thead>
<tr>
<th>More formal</th>
<th>........</th>
<th>Less formal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation</td>
<td>Table</td>
<td></td>
</tr>
<tr>
<td>Tuple</td>
<td>Row</td>
<td>Record</td>
</tr>
<tr>
<td>Attribute</td>
<td>Column</td>
<td>Field</td>
</tr>
<tr>
<td>Domain</td>
<td>Type</td>
<td></td>
</tr>
</tbody>
</table>

Structured Query Language (SQL)

- **Data definition language** (DDL)
  - Define the schema (create, change, delete relations)
  - Specify constraints, user permissions

- **Data modification language** (DML)
  - Find data that matches criteria
  - Add, remove, update data
  - *The DBMS is responsible for efficient evaluation!*

- Co-invented by Don Chamberlin (HMC ‘66!)

A Relation Instance

- An *instance* of a relation is its contents at a given time
- *cardinality:* # tuples
- *arity:* # attributes

<table>
<thead>
<tr>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>45</td>
</tr>
<tr>
<td>67</td>
</tr>
<tr>
<td>78</td>
</tr>
</tbody>
</table>

SQL: Creating Relations

- Create **Students** relation:
  ```sql
  CREATE TABLE Students ( 
  sid CHAR(20), 
  name CHAR(20), 
  login CHAR(100), 
  SSN CHAR(12), 
  gpa FLOAT); 
  ```

- Create **Enrolled** relation:
  ```sql
  CREATE TABLE Enrolled ( 
  sid CHAR(20), 
  cid CHAR(20), 
  grade CHAR(2)); 
  ```

- Domain info is type of **Integrity constraint** (IC)
  - IC: a condition on the database schema, restricts data that can be stored
Adding and Removing Tuples

• Insert a single tuple

```
INSERT INTO Students (sid, name, login, SSN, gpa) VALUES (45, 'Alice', 'alicious', '000-00-0000, 3.4);
```

• Delete tuples that satisfy condition (*predicate*)

```
DELETE FROM Students S WHERE S.name = 'Alice';
```

Integrity Constraints: Keys

• *Superkey* is a set of field(s) that
  – Uniquely identifies a tuple
  – *Candidate key*: does so *minimally*
  – *Primary key*: a chosen candidate key

<table>
<thead>
<tr>
<th>SID</th>
<th>Name</th>
<th>Login</th>
<th>SSN</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>Alice</td>
<td>alicious</td>
<td>000-00-0000</td>
<td>3.4</td>
</tr>
<tr>
<td>67</td>
<td>Bob</td>
<td>bobtastic</td>
<td>000-00-0001</td>
<td>3.9</td>
</tr>
<tr>
<td>78</td>
<td>Carl</td>
<td>carl</td>
<td>000-00-0010</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Integrity Constraints: Foreign Keys

• Referential integrity, logical “pointer”
  – Set of fields in one relation refer to primary key of another

<table>
<thead>
<tr>
<th>Students</th>
<th>Enrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>Name</td>
</tr>
<tr>
<td>45 Alice</td>
<td>alicious</td>
</tr>
<tr>
<td>67 Bob</td>
<td>bobtastic</td>
</tr>
<tr>
<td>78 Carl</td>
<td>carl</td>
</tr>
</tbody>
</table>

Defining Key Constraints

• Specified in schema definition

```
CREATE TABLE Students (sid CHAR(20),
                        name CHAR(20),
                        login CHAR(10),
                        SSN CHAR(20),
                        gpa FLOAT,
                        PRIMARY KEY (sid),
                        UNIQUE (SSN));
```

```
CREATE TABLE Enrolled (sid CHAR(20),
                       cid CHAR(20),
                       grade CHAR(2),
                       PRIMARY KEY (sid,cid),
                       FOREIGN KEY (sid) REFERENCES Students);
```

```
INSERT INTO
Enrolled(sid,cid,grade)
VALUES (43,CS133,D);
```

```
INSERT INTO Students (sid, name, login, SSN, gpa) VALUES (45, 'Alice', 'alicious', '000-00-0000, 3.4);
```

```
DELETE FROM Students S WHERE S.name = 'Alice';
```
Primary and Candidate Keys in SQL

- Possibly many *candidate keys* (specified using `UNIQUE`), one of which is chosen as the *primary key*.
- Keys must be used carefully!
- Example:
  
  "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid))
```

```
CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid),
UNIQUE (cid, grade))
```

"Students can take only one course, and no two students in a course receive the same grade."

---

SQL: Single Relation Queries

```
CREATE TABLE Students
(SID INT, name VARCHAR(255),
login VARCHAR(255),
gpa REAL);
```

```
SELECT name
FROM Students
WHERE gpa > 3.7;
```

```
SELECT *
FROM Students S
WHERE S.gpa > 3.7;
```

---

**Query Execution: Teaser**

*Query optimizer* transforms a declarative query into a pipeline of dataflow *operators* called a *query execution plan*.

```
SELECT name
FROM Students
WHERE gpa > 3.7;
```

- **Project** `(name)`
- **Filter** `(gpa > 3.7)`
- **Students**
- **Iterators!!**

---

**SQLite Demo**

The database is in the file you specify. The file is created if it doesn’t exist.

SQL statements end with a semicolon. Capitalization looks nice, but not required.

These two settings for `mode` and `header` make query results easier to read.

Also see: “Resources” on course website and www.sqlite.org