CS 133: Databases

Fall 2016
Lec 1 – 08/31

Prof. Beth Trushkowsky

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)

Data!

• Data is everywhere (and it can be BIG)
  – Banking, airline reservations
  – Social media, clicking anything on the internet

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*

  ```
  CREATE VIEW CourseInfo AS
  SELECT cid, COUNT(*) AS enrollmnt
  FROM Enrolled
  GROUP BY cid;
  ```

- **External schema** (“views”)
  - view each course’s enrollment
    `CourseInfo(cid: string, enrollmnt: integer)`

---

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**
  
  ```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? (should be $250!)

Example from Jun Yang

---

Concurrent Access

**Example from Jun Yang**

Alice withdraws $100

```pseudocode
get balance; $400
if balance > amount
    withdraw amount;
    newBalance = balance - amount;
    write balance = newBalance;
```

Bob withdraws $50

```pseudocode
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

Example from Jun Yang
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

• 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

- 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
- Piazza for questions about labs, problem sets, etc.: piazza.com/hmc/fall2016/cs133/home
- Assignment submission on Sakai
- Office hours
  - TBD, likely to be Tuesday and Wednesday afternoon
  - Location: Olin 1261 (workroom) or Olin 1267 (my office)
- Grutors
  - Carli and Max
  - Hours: TBD

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)

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

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>
Adding and Removing Tuples

- Insert a single tuple

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

- Delete tuples that satisfy condition (*predicate*)

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

<table>
<thead>
<tr>
<th>Enrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
</tr>
<tr>
<td>45</td>
</tr>
<tr>
<td>45</td>
</tr>
<tr>
<td>78</td>
</tr>
</tbody>
</table>

Defining Keys

- Specified in schema definition

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

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

• 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),
UNIQUE (sid, grade))

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

SQL: Single Relation Queries

Students

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>login</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>Alice</td>
<td>alicious</td>
<td>3.4</td>
</tr>
<tr>
<td>67</td>
<td>Bob</td>
<td>bobtastic</td>
<td>3.9</td>
</tr>
<tr>
<td>78</td>
<td>Carl</td>
<td>carl</td>
<td>2.5</td>
</tr>
</tbody>
</table>

SELECT name
FROM Students
WHERE gpa > 3.7;

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

Multi-Relation Queries

Enrolled

<table>
<thead>
<tr>
<th>SID</th>
<th>CID</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>CS133</td>
<td>A</td>
</tr>
<tr>
<td>45</td>
<td>CS121</td>
<td>B</td>
</tr>
<tr>
<td>78</td>
<td>CS5</td>
<td>A</td>
</tr>
</tbody>
</table>

SELECT S.name, E.CID
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade = “B”;

<table>
<thead>
<tr>
<th>S.name</th>
<th>E.CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>CS121</td>
</tr>
</tbody>
</table>
Basic Query: Select-From-Where

\[
\text{SELECT [DISTINCT] } A_1, A_2, \ldots, A_n \\
\text{FROM } R_1, R_2, \ldots, R_n \\
\text{WHERE } \text{condition(s)}; 
\]

Relation List
Relations used in query, implicitly JOINed.

Target List
Attributes from relation list.

Comparisons
Conjunctive (“AND”), and Disjunctive (“OR”)

Also called an SPJ (select-project-join)

Query Semantics

Conceptual query evaluation steps:
1. do FROM clause: cross-product of tables
2. do WHERE clause: check conditions, discard tuples that fail
3. do SELECT clause: delete unwanted fields
4. do DISTINCT: eliminate duplicate tuples

Actually very inefficient in practice!
An optimizer will find more efficient strategies to get the same answer.

(1) FROM: Cross-Product

FROM Students S, Enrolled E

<table>
<thead>
<tr>
<th>S.SID</th>
<th>S.name</th>
<th>S.login</th>
<th>S.gpa</th>
<th>E.SID</th>
<th>E.CID</th>
<th>E.grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>Alice</td>
<td>alicious</td>
<td>3.4</td>
<td>45</td>
<td>CS133</td>
<td>A</td>
</tr>
<tr>
<td>67</td>
<td>Bob</td>
<td>bobtastic</td>
<td>3.9</td>
<td>45</td>
<td>CS121</td>
<td>B</td>
</tr>
<tr>
<td>78</td>
<td>Carl</td>
<td>carl</td>
<td>2.5</td>
<td>78</td>
<td>CS5</td>
<td>A</td>
</tr>
</tbody>
</table>

(2) WHERE: Discard tuples that fail conditions

WHERE S.sid=E.sid AND E.grade='B'

<table>
<thead>
<tr>
<th>S.SID</th>
<th>S.name</th>
<th>S.login</th>
<th>S.gpa</th>
<th>E.SID</th>
<th>E.CID</th>
<th>E.grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>Alice</td>
<td>alicious</td>
<td>3.4</td>
<td>45</td>
<td>CS133</td>
<td>A</td>
</tr>
<tr>
<td>67</td>
<td>Bob</td>
<td>bobtastic</td>
<td>3.9</td>
<td>45</td>
<td>CS133</td>
<td>A</td>
</tr>
<tr>
<td>78</td>
<td>Carl</td>
<td>carl</td>
<td>2.5</td>
<td>45</td>
<td>CS121</td>
<td>B</td>
</tr>
<tr>
<td>45</td>
<td>Alice</td>
<td>alicious</td>
<td>3.4</td>
<td>45</td>
<td>CS121</td>
<td>B</td>
</tr>
<tr>
<td>67</td>
<td>Bob</td>
<td>bobtastic</td>
<td>3.9</td>
<td>78</td>
<td>CS5</td>
<td>A</td>
</tr>
<tr>
<td>78</td>
<td>Carl</td>
<td>carl</td>
<td>2.5</td>
<td>78</td>
<td>CS5</td>
<td>A</td>
</tr>
</tbody>
</table>
(3) SELECT: Delete Unwanted Fields

```
SELECT S.name, E.CID
FROM Students S, Enrolled E
WHERE S.sid = E.sid AND E.grade = "B";
```
Query Semantics (cntd)

Conceptual query evaluation steps:
1. do **FROM** clause: *cross-product* of tables
2. do **WHERE** clause: check conditions, discard tuples that fail
3. Remove fields not in **SELECT, GROUP BY, or HAVING** clauses
4. do **GROUP BY**: partition into groups
5. do **HAVING**: delete groups that do not meet conditions

*Result: one answer tuple per qualifying group*

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