Outline

- **Overview of modeling**
- **SQL (Chapter 3)**
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)
- **Relational Model (Chapter 2)**
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
DBMSs to the Rescue

- Provide a systematic way to answer many of these questions...
- Aim is to allow easy management of high volumes of data
  - Storing, Updating, Querying, Analyzing ....

What is a Database?
- A large, integrated collection of (mostly structured) data
  - Typically models and captures information about a real-world enterprise
    - Entities (e.g. courses, students)
    - Relationships (e.g. John is taking CMSC 424)
  - Usually also contains:
    - Knowledge of constraints on the data (e.g. course capacities)
    - Business logic (e.g. pre-requisite rules)
    - Encoded as part of the data model (preferable) or through external programs

DBMSs to the Rescue

- Massively successful for highly structured data
  - Why? Structure in the data (if any) can be exploited for ease of use and efficiency
    - If there is no structure in the data, hard to do much
    - Contrast managing emails vs managing photos
  - Much of the data we need to deal with is highly structured
  - Some data is semi-structured
    - E.g.: Resumes, Webpages, Blogs etc.
  - Some has complicated structure
    - E.g.: Social networks
  - Some has no structure
    - E.g.: Text data, Video/Image data etc.
Structured vs Unstructured Data

- A lot of the data we encounter is structured
  - Some have very simple structures
    - E.g. Data that can be represented in tabular forms
    - Significantly easier to deal with
  - We will focus on such data for much of the class

<table>
<thead>
<tr>
<th>Account</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>bname</td>
<td>cname</td>
</tr>
<tr>
<td>acct_no</td>
<td>cstreet</td>
</tr>
<tr>
<td>balance</td>
<td>ccity</td>
</tr>
</tbody>
</table>

| Downtown | Jones       |
| Mianus   | Main        |
| Perry    | Harrison    |
| R.H      | Rye         |

| A-101    | Main        |
| A-215    | North       |
| A-102    | North       |
| A-305    | Park        |

| A-H      | Harrison    |
| A-215    | Rye         |
| A-102    | Rye         |
| A-305    | Pittsfield  |

Balance:
- Downtown: 500
- Mianus: 700
- Perry: 400
- R.H: 350

Structured vs Unstructured Data

- Some data has a little more complicated structure
  - E.g graph structures
    - Map data, social networks data, the web link structure etc.
  - Can convert to tabular forms for storage, but may not be optimal
  - Queries often reason about graph structure
    - Find my “Erdos number”
    - Suggest friends based on current friends
  - Growing importance in recent years in a variety of domains: Biological, social networks, web...
Structured vs Unstructured Data

- Increasing amount of data in a semi-structured format
  - XML – Self-describing tags (HTML ?)
  - Complicates a lot of things
  - We will discuss this toward the end

- A huge amount of data is unfortunately unstructured
  - Books, WWW
  - Amenable to pretty much only text search... so far
    - Information Retrieval research deals with this topic
  - What about Google search?
    - Google search is mainly successful because it uses link structure (in its original incarnation)

- Video ? Music ?
  - Can represent in DBMS’s, but can’t really operate on them

DBMSs to the Rescue

- Massively successful for highly structured data
  - Two Key Concepts:
    - Data Modeling: Allows reasoning about the data at a high level
      - e.g. “emails” have “sender”, “receiver”, “…”,
      - Once we can describe the data, we can start “querying” it
    - Data Abstraction/Independence:
      - Layer the system so that the users/applications are insulated from the low-level details
Data modeling

- **Data model**: A collection of concepts that describes how data is represented and accessed
- **Schema**: A description of a specific collection of data, using a given data model

Some examples of data models that we will see
- Relational, Entity-relationship model, XML, JSON...
- Object-oriented, object-relational, semantic data model, RDF...

Why so many models?
- Tension between descriptive power and ease of use/efficiency
- More powerful models → more data can be represented
- More powerful models → harder to use, to query, and less efficient

DBMSs to the Rescue: Data Abstraction

- Probably *the* most important purpose of a DBMS
- Goal: Hiding *low-level details* from the users of the system
  - Alternatively: the principle that
    - *applications and users should be insulated from how data is structured and stored*
  - Also called *data independence*

- Through use of *logical abstractions*
Data Abstraction

**What data users and application programs see?**

**What data is stored?**
- describe data properties such as data semantics, data relationships

**How data is actually stored?**
- e.g. are we using disks? Which file system?

---

Data Abstraction

**Logical Data Independence**
- Protection from logical changes to the schema

**Physical Data Independence**
- Protection from changes to the physical structure of the data
Data Abstractions: Example

A “view” Schema

course_info(#registered,...)

Logical Schema

students(sid, name, major, ...)
courses(cid, name, ...)
enrolled(sid, cid, ...)

Physical Schema

all students in one file ordered by sid
courses split into multiple files by colleges

View Level

View 1
View 2
... View n

Logical Level

Physical Level

What about a Database System?

› A DBMS is a software system designed to store, manage, facilitate access to databases

› Provides:
  ◦ Data Definition Language (DDL)
    • For defining and modifying the schemas
  ◦ Data Manipulation Language (DML)
    • For retrieving, modifying, analyzing the data itself
  ◦ Guarantees about correctness in presence of failures and concurrency, data semantics etc.

› Common use patterns
  ◦ Handling transactions (e.g. ATM Transactions, flight reservations)
  ◦ Archival (storing historical data)
  ◦ Analytics (e.g. identifying trends, Data Mining)
Relational DBMS: SQL

- **SQL** (sequel): Structured Query Language
- **Data definition (DDL)**
  - `create table instructor (``
    - `ID` char(5),
    - `name` varchar(20),
    - `dept_name` varchar(20),
    - `salary` numeric(8,2))`
- **Data manipulation (DML)**
  - Example: Find the name of the instructor with ID 22222
    ```
    select name
    from instructor
    where instructor.ID = '22222'
    ```

Current Industry Outlook

- **Relational DBMSs**
  - Oracle, IBM DB2, Microsoft SQL Server, Sybase
- **Open source alternatives**
  - MySQL, PostgreSQL, Apache Derby, BerkeleyDB (mainly a storage engine – no SQL), neo4j (graph data) …
- **Data Warehousing Solutions**
  - Geared towards very large volumes of data and on analyzing them
  - Long list: Teradata, Oracle Exadata, Netezza (based on FPGAs), Aster Data (founded 2005), Vertica (column-based), Kickfire, Xtremedata (released 2009), Sybase IQ, Greenplum (eBay, Fox Networks use them)
  - Usually sell package/services and charge per TB of managed data
  - Many (especially recent ones) start with MySQL or PostgreSQL and make them parallel/faster etc..
Web Scale Data Management, Analysis

- Ongoing debate/issue
  - Cloud computing seems to eschew DBMSs in favor of homegrown solutions
  - E.g. Google, Facebook, Amazon etc...

- MapReduce: A paradigm for large-scale data analysis
  - Hadoop: An open source implementation
  - Apache Spark: a better open source implementation

- Why?
  - DBMSs can’t scale to the needs, not fault-tolerant enough
    - These apps don’t need things like transactions, that complicate DBMSs (???)
  - Mapreduce favors Unix-style programming, doesn’t require SQL
    - Try writing SVMs or decision trees in SQL
  - Cost
    - Companies like Teradata may charge $100,000 per TB of data managed

Current Industry Outlook

- Bigtable-like
  - Called “key-value stores”
  - Think highly distributed hash tables
  - Allow some transactional capabilities – still evolving area
  - PNUTS (Yahoo), Apache Cassandra (Facebook), Dynamo (Amazon), and many many others

- Mapreduce-like
  - Hadoop (open source), Pig (@Yahoo), Dryad (@Microsoft), Spark
  - Amazon EC2 Framework
  - Not really a database – but increasing declarative SQL-like capabilities are being added (e.g. HIVE at Facebook)

- Much ongoing research in industry and academia
DBMS at a glance

- Data Models
  - Conceptual representation of the data
- Data Retrieval
  - How to ask questions of the database
  - How to answer those questions
- Data Storage
  - How/where to store data, how to access it
- Data Integrity
  - Manage crashes, concurrency
  - Manage semantic inconsistencies

- Not fully disjoint categorization !!

CMSC424: Database Design
Introduction/Overview

Professor: Pete Keleher
keleher@umd.edu
Outline

- Overview of modeling
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics

History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86, SQL-89, SQL-92
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.
- Several alternative syntaxes to write the same queries
Different Types of Constructs

- **Data definition language (DDL):** Defining/modifying schemas
  - **Integrity constraints:** Specifying conditions the data must satisfy
  - **View definition:** Defining views over data
  - **Authorization:** Who can access what
- **Data-manipulation language (DML):** Insert/delete/update tuples, queries
- **Transaction control:**
- **Embedded SQL:** Calling SQL from within programming languages
- **Creating indexes, Query Optimization control...**

SQL: Data Definition Language

The SQL **data-definition language (DDL)** allows the specification of information about relations, including:

- The schema for each relation.
- **Keys**
- The domain of values associated with each attribute.
- Integrity constraints
- Also: other information such as
  - The set of indices to be maintained for each relations.
  - Security and authorization information for each relation.
  - The physical storage structure of each relation on disk.
Keys (more later)

- Let $K \subseteq R$ (R is a set of columns)
- $K$ is a superkey of $R$ if values for $K$ are sufficient to identify a unique row of any possible table
  - Example: $\{ID\}$ and $\{ID, name\}$ are both superkeys of instructor.
- Superkey $K$ is a candidate key if $K$ is minimal (i.e., no subset of it is a superkey)
  - Example: $\{ID\}$ is a candidate key for Instructor
- One candidate key can be the primary key
  - Typically one that is small and immutable (doesn’t change often)
  - Chosen by app/user
- Keys are unique!

SQL Constructs: Data Definition Language

- CREATE TABLE <name> ( <field> <domain>, ... )

```sql
create table instructor ( 
  ID char(5),
  name varchar(20) not null,
  dept_name varchar(20),
  salary numeric(8,2),
  primary key (ID),
  foreign key (dept_name) references department
); create table department ( 
  dept_name varchar(20),
  building varchar(15),
  budget numeric(12,2) check (budget > 0),
  primary key (dept_name)
);```

```sql```
SQL Constructs: Data Definition Language

- CREATE TABLE <name> ( <field> <domain>, ... )

```sql
create table instructor
(
    ID char(5),
    name varchar(20) not null,
    dept_name varchar(20),
    salary numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department
)
```

- create table department
  (
    dept_name varchar(20),
    building varchar(15),
    budget numeric(12,2) check (budget > 0),
    primary key (dept_name)
  )

- drop table student
- delete from student
  - Keeps the empty table around
- alter table
  - alter table student add address varchar(50);
  - alter table student drop tot_cred;

SQL Constructs: Data Definition Language
SQL Constructs: Insert/Delete/Update Tuples (DML)

- INSERT INTO <name> (<field names>) VALUES (<field values>)
  insert into instructor values (‘10211’, ‘Smith’, ‘Biology’, 66000);
  insert into instructor (name, ID) values (‘Smith’, ‘10211’);
  -- NULL for other two
  insert into instructor (ID) values (‘10211’);
  -- FAIL

- DELETE FROM <name> WHERE <condition>:
  delete from department where budget < 80000;
  • Syntax is fine, but this command may be rejected because of referential integrity constraints.

SQL Constructs: Insert/Delete/Update Tuples

- DELETE FROM <name> WHERE <condition>
  delete from department where budget < 60000;

<table>
<thead>
<tr>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biology</td>
<td>Watson</td>
<td>90000</td>
</tr>
<tr>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>Elec. Eng.</td>
<td>Taylor</td>
<td>85000</td>
</tr>
<tr>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
</tbody>
</table>

Figure 2.5 The department relation.

We can choose what happens:
1. Reject the delete, or
2. Delete the rows in Instructor (may be a cascade), or
3. Set the appropriate values in Instructor to NULL.
DELETE FROM <name> WHERE <condition>

delete from department where budget < 60000;

create table instructor
(ID varchar(5),
 name varchar(20) not null,
 dept_name varchar(20),
 salary numeric(8,2) check (salary > 29000),
 primary key (ID),
 foreign key (dept_name) references department
   on delete set null);

We can choose what happens:
(1) Reject the delete (nothing), or
(2) Delete the rows in Instructor (on delete cascade), or
(3) Set the appropriate values in Instructor to NULL (on delete set null)

DELETE FROM <name> WHERE <condition>

◦ Delete all classrooms with capacity below average
  
delete from classroom where capacity <
    (select avg(capacity) from classroom);
  • Problem: as we delete tuples, the average capacity changes

◦ Solution used in SQL:
  • First, compute avg capacity and find all tuples to delete
  • Next, delete all tuples found above (without recomputing avg or retesting the tuples)
**SQL Constructs: Insert/Delete/Update Tuples**

- UPDATE <name> SET <field name> = <value> WHERE <condition>
  - Increase all salaries over $100,000 by 6%, all other receive 5%.
  - Write two update statements:
    - update instructor
      set salary = salary * 1.05
      where salary < 10000;
    - update instructor
      set salary = salary * 1.06
      where salary > 100000;
  - The order is important
  - Can be done better using the case statement

```sql
update instructor
set salary = salary * 1.06
where salary > 100000;
update instructor
set salary = salary * 1.05
where salary <= 10000;
```

---

**SQL Constructs: Insert/Delete/Update Tuples**

- UPDATE <name> SET <field name> = <value> WHERE <condition>
  - Increase all salaries over $100,000 by 6%, all other receive 5%.
  - Can be done better using the case statement

```sql
UPDATE instructor
SET salary =
  CASE
    WHEN salary > 100000
      THEN salary * 1.06
    WHEN salary <= 100000
      THEN salary * 1.05
  END;
```

DONE