CSCI 1320
Creating Modern Web Applications
Lecture 14: Databases
Web Applications

Front End

Web Browser

HTTP

Back End

Web Server

Database

Server
Question

Which is not true about SQL databases?

A. They are designed to store large amounts of data
B. They provide robustness by safeguarding the data against hardware failure
C. They offer a generic interface for accessing the data
D. They let multiple users access and set data simultaneously
E. They make changing the data schema (format) easy
Why Use Databases

- Efficient ways of storing large amounts of data
  - Designed to scale for data
  - Designed to scale for multiple servers
- Ensure the integrity of the data
  - Against hardware failures
  - Against multiple simultaneous updates
  - Against inconsistent updates
- Allow easy and efficient access to the data
  - Query language makes any access possible
  - Query optimization can make access efficient
Database Systems

• MySQL, PostgreSQL, Oracle, SqlServer, DB2, Derby, Sqlite3...
  • All support core SQL (more or less)

• Many support extensions
  • Regular expressions, transitive closure
  • XML-based querying (using XQUERY)
  • User defined functions and operators
  • Cluster or distributed operation
  • Additional data types

• What do you want to rely on
  • Different environments might have different availabilities
  • Migrating can be problematic (esp. with extensions)
  • System updates change things as well
What the Database System Does

- Stores the data
- Ensures the integrity of the data
- Understands the query language
- Compile & execute queries efficiently
- Allows data to be updated
Data Storage

• Web applications are mainly data-centric
  • Generally in the form of web requests and web pages
  • Examples: amazon, expedia, ...

• How do these applications work?
  • What data do they use?
  • What do they do with that data?
  • How do user interactions affect the data?
  • How does the data affect user interactions?
Data Storage Requirements

• The storage needs to be robust
  • Don’t want to lose a user’s purchase
• Need to handle multiple requests at once
  • Only one person can buy a particular airplane seat
• The data is important
  • The data is the company
Securing the Data

• Providing robust data storage is complex
  • But a common necessity
  • Can be separated from the actual application
  • Done by database systems so you don’t have to
    • Database system is 20% code to do the work
    • And 80% code to handle exceptions, errors, conflicts, recovery
    • You don’t want to write this yourself

• Database systems are integral to web applications
  • Combined with Node.JS/PHP that might be all you need
Data Example: FreeDB

- Database of all CDs published
- Provided as lots of text files in a tar image
  - With semi-parsable records (not XML or JSON)
  - Provide disk title, artist, genre, length
  - Provide track title, artist, offsets
  - Provide additional information and comments
- It would be nice to have web access to this data
  - To find CDs
  - To add comments
  - To correct mistakes and typos (lots of these)
  - To manage one's own collection
- We've considered the application earlier
  - Now let's consider the data
Entity-Relationship Diagrams
Relational Databases (SQL)

• Organize the data in a way that is independent of its use
  • Can support arbitrary queries of the data
  • User doesn’t have to know what the queries are in advance

• In general data consist of
  • Facts (actual data)
  • Relationships between the facts (pointers)

• Relational databases make all relationships implicit
  • Based on matching values, not on links
Relational Databases

- A relational database is a set of TABLES (relations)
  - Each table holds a coherent set of data
- A table is divided into FIELDS (columns, attributes)
  - Each field holds data of a single type
- The table’s ROWS (tuples) contain the actual data
  - Value for each field of the table
  - A row is a single data instance
- One (or more) fields might be the KEY
  - Uniquely identify the row
Relational Data

Hypothetical Relational Database Model

<table>
<thead>
<tr>
<th>PubID</th>
<th>Publisher</th>
<th>PubAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>03-4472822</td>
<td>Random House</td>
<td>123 4th Stree, New York</td>
</tr>
<tr>
<td>04-7733903</td>
<td>Wiley and Sons</td>
<td>45 Lincoln Blvd, Chicago</td>
</tr>
<tr>
<td>03-4859223</td>
<td>O'Reilly Press</td>
<td>77 Boston Ave, Cambridge</td>
</tr>
<tr>
<td>03-3920886</td>
<td>City Lights Books</td>
<td>99 Market, San Francisco</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AuthorID</th>
<th>AuthorName</th>
<th>AuthorBDay</th>
</tr>
</thead>
<tbody>
<tr>
<td>345-28-2938</td>
<td>Haile Selassie</td>
<td>14-Aug-92</td>
</tr>
<tr>
<td>392-48-9965</td>
<td>Joe Blow</td>
<td>14-Mar-15</td>
</tr>
<tr>
<td>454-22-4012</td>
<td>Sally Hemmings</td>
<td>12-Sep-70</td>
</tr>
<tr>
<td>663-59-1254</td>
<td>Hannah Arendt</td>
<td>12-Mar-06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ISBN</th>
<th>AuthorID</th>
<th>PubID</th>
<th>Date</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-34532-482-1</td>
<td>345-28-2938</td>
<td>03-4472822</td>
<td>1990</td>
<td>Cold Fusion for Dummies</td>
</tr>
<tr>
<td>1-38482-995-1</td>
<td>392-48-9965</td>
<td>04-7733903</td>
<td>1985</td>
<td>Macrame and Straw Tying</td>
</tr>
<tr>
<td>2-35921-499-4</td>
<td>454-22-4012</td>
<td>03-4859223</td>
<td>1852</td>
<td>Fluid Dynamics of Aqueducts</td>
</tr>
<tr>
<td>1-38278-293-4</td>
<td>663-59-1254</td>
<td>03-3920886</td>
<td>1967</td>
<td>Beads, Baskets &amp; Revolution</td>
</tr>
</tbody>
</table>
BeiberFeed Data

• What data is there
• What are the facts and relationships
• How could it be organized
### CDQuery Database Schema

<table>
<thead>
<tr>
<th>ARTIST</th>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISK</th>
<th>ID</th>
<th>Title</th>
<th>ArtistID</th>
<th>Length</th>
<th>Genre</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TRACK</th>
<th>ID</th>
<th>Name</th>
<th>DiskID</th>
<th>ArtistID</th>
<th>Length</th>
<th>Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EXTENDED</th>
<th>DiskID</th>
<th>TrackID</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5/1/17
Obtaining Information from the Database

• Want to get information from the database
  • Not all of it at once
  • Information for a particular purpose
  • To answer a particular question

• What might we want to get from the CD database?
  • What would you like to know?
Sample Questions to Ask

• CDs of a given artist
• CDs with a particular song
• What artist is on the most CDs
• CDs are associated with ‘nsync’
• What CDs have a track by Justin Bieber
• CDs that have ‘Paris’ in the title with artist Brel
Inverted Indices

- Create a new relation to make text lookup easier
  - Word and where it occurs
  - This is an inverted index
- Wherever a word occurs
  - Type: T=title, N=track name, A=artist, D=disk data, I=track data
  - ID: disk id, artist id or track id
  - Alternatives: multiple word relations, word number
- CDs that have the word ‘Paris’ in the title

<table>
<thead>
<tr>
<th>WORDS</th>
<th>Word</th>
<th>Type</th>
<th>ID</th>
</tr>
</thead>
</table>

CDs that have the word ‘Paris’ in the title: 5 / 1 / 17
Example Query

CDs that have the word ‘Paris’ in the title

SELECT d.title
FROM Disk d, Words w
WHERE w.word = ‘paris’ AND w.type = ‘T’ AND w.id = d.id
SQL Basics

• **SELECT** <result> **FROM** <source> **WHERE** <condition>

• **SELECT**: define the resultant table
  • List of fields and where they come from
  • Expression, Expression AS name, *
    • Generally just a table.field name
    • Can be real expressions: field + 1
    • Can be grouping expressions: COUNT(*)

• **FROM**: what tables to use as the input
  • Either a list of table names or <table variable> pairs
  • Variables provide shorted names for easier access
  • Variables allow names to be listed multiple times
SQL WHERE

- **WHERE** clause specifies the actual query
- Sequence of relational expressions
  - Separated by **AND** and **OR**
  - X.field = ‘value’
  - X.field = Y.field
- Can also have nested SELECTS
  - X.field IN (SELECT ...)
- Also set operations on tables
- Also string operations
  - Value **LIKE** pattern
    - % is a wildcard, _ matches any single character
  - Some databases allow regular expressions (not standard)
SQL Examples

• SELECT d.title FROM Disk d
  WHERE d.title LIKE '%Paris%'

• SELECT d.title FROM Disk d, Words w
  WHERE w.word = 'paris' AND w.type = 'T' AND w.id = d.id

• SELECT d.title FROM Disk d, Words w
  WHERE w.word = 'beatles' AND w.type = 'A' AND d.artistid = w.id

• SELECT count(d.title) FROM Disk d, Artist a
  WHERE a.id = d.artistid AND a.name = 'Madonna'

• SELECT DISTINCT d.title FROM Disk d, Track T, Words w
  WHERE w.word = 'bieber' AND w.type = 'A' AND T.artistid = w.id AND T.diskid = d.id
Queries

• Assume the data is stored on disk as tables
  • Table contains rows
  • Rows contain data

• How might the database find CDs with a given title?
  • Scanning a table to look for entries
  • Creating an index for the table
    • Fast access based on a value
Indices

• How do indices work
  • How do they work in memory
    • Hash tables, trees (red-black trees)
    • Looking for a range of values rather than a particular value
  • B-trees (block-trees) versus Binary trees
    • Balanced trees with variables number of keys at each level
    • Minimize I/O operations
    • Can be scanned or accessed as a index
  • Bucket-based hash tables
• Indices can cover multiple columns at once
• Why not index everything?
  • How many indices are needed for 10 fields?
  • What is the cost of an index?
    • Storage, update time, creation time
  • Actually, this is one of today’s trends – column-store databases
Embedded SQL

• SQL is used inside programs
  • Actually built into some languages
  • Create a string representing the query
  • Pass that string to the database to interpret

• Concepts
  • **Connection**: connection to the database
    • Accessed by a URL giving type, host, database, user, pwd, ...
  • **Statement**: set of SQL statements to be executed as one
    • Typically a single query or set of updates
  • **ResultSet**: iterator over a returned table
    • Can iterate over tuples in the returned values
    • Can access fields of the current tuple
Using Embedded SQL Safely

• Queries and updates are built as strings
  • Provides flexibility
  • Lets program add user-entered values to the queries
  • Can be problematic if user values are non-standard

• Prepared statements
  • Queries with variables are defined as strings
  • Variables in the query are represented by $, $i, or ?
  • Values are passed when the query is used
  • Can be faster (database can optimize the query once)
  • Safer and more secure
  • Use when possible (and it's always possible)
Next Time

• More on databases
• Next Monday/Wednesday: Elevator talks
SQL INSERT Statement

• INSERT INTO table ( field, ..., field )
  VALUES ( val,...val)

  • List of fields is optional, but should be there
    • Avoids problems in future, reminds reader of what is what

  • Values can be
    • DEFAULT
    • Constants [ true, false, 0, ... ]
    • Built-in Functions [ CURRENT_TIMESTAMP ]
    • Variables – use $, $i, ? and pass values separately
    • Results of queries [ (SELECT id FROM Artist A WHERE A.name = ‘nsync’) ]

• INSERT INTO table (field, ..., field) SELECT ...
SQL UPDATE Statement

- **UPDATE** table **SET** field = value **WHERE** condition
  - SET field = value, field = value, ...
  - Values as in an insert statement
  - **WHERE** as in normal select
- Can update multiple rows at once
  - Be careful with the condition
SQL DELETE Statement

- DELETE FROM table WHERE condition
  - Removes all rows in table where condition is true
  - If condition is omitted, deletes all rows
Data Organization

• Databases need to be efficient
  • The key to this is data organization
  • Structuring and organizing the database for performance

• How would you organize the CD data in a program?
  • Might think about objects – what are the objects?
  • Might think about access to data (e.g. Java Maps)
  • Why is this a trick question?
Data Organization

• Suppose you want to find
  • All CDs with a given title
  • All CDs with a given artist
  • All CDs containing a particular song
  • All CDs containing a certain phrase in the title
  • All CDs containing a particular song by a particular artist
Disk Data Organization

• What happens when the data is on disk
  • Do the same algorithms and data structures apply?
  • How do you measure the costs in this case?
  • I/O operations rather than lookups, compares
    • What happens if you use solid-state disks?
Relational Data Organization

- Traditional, in-memory, data organization
  - Data is represented explicitly
  - Relationships are represented explicitly (as links)
  - This assumes you know what will be asked

- Relational data organization
  - Data is organized into tables (or relations)
  - Relationships between tables are IMPLICIT
    - Based on data values, not links or pointers
    - Defined dynamically as needed
Querying the Database

• We need a language to express what data we want
  • Used to describe how to get the data
  • Query language

• What should the result of a query be?
  • Set of values
  • Set of related values
  • A table (just like the data tables in the database)

• Using tables as the result of queries
  • Is a nice clean model
  • Allows queries to be nested
  • Allows queries to define new tables
    • Both real and virtual
Query Languages

• We want a standard language to express queries
  • Several languages have been developed
    • Find all X satisfying Y
    • Operations on tables: project, product, select, union, ...
    • Query-by-example
  • All have equivalent power
    • Can do a lot, can’t do everything (transitive closure)

• Language should also handle
  • Setting up the database (defining tables)
  • Changing values in the database (update)
  • Adding data to the database (insert)
Query Languages

- **SQL** has become the standard
  - Language for building tables given tables
  - Used both directly and inside programs
  - SELECT for query
  - INSERT for insert
  - UPDATE for update
  - CREATE, GRANT, DROP, ... for maintenance

- **XQUERY** is an extension to handle XML structures

- **NOSQL** is becoming more common for web apps