Database Systems

Introduction and the Relational Model

Utah Tech University—Department of Computing

Spring 2024

Overview

What is this class about?

- Relational Database Management Systems (RDBMS)
  - The most important kind of database for most applications
  - What most people think of as a database
- Other models: key-value, columnar stores, etc.
  - More specialized: we will glance at them but not much more

Our approach:

1. The relational model, SQL, schema design, queries, etc.
2. Using databases from Python (or other languages)
3. How databases are implemented (SQLite will be our model)
   - On-disk data storage
   - Query plans and query execution
   - Indexing
   - Caching
   - ACID transactions, logging, failure recovery
   - Concurrency
4. What else is out there?

Attendance, distractions, etc.

- Attendance is not required in that you will not be graded for being here
  - Exception: excessive absense without making arrangements will result in failing (see the syllabus)
- You are responsible for what we talk about in class, and much of what we cover will *not* be available elsewhere
  - Assignment instructions, tips, etc.
  - If you miss class, you may not be able to complete the homework
- This is an in-person class. I will attempt to stream it via Zoom on request if there is a good reason, but the AV system is flaky and it will probably fail on some days
  - Do not depend on Zoom
- You are expected to take notes: bring pen and paper
- Laptops and mobile devices are not allowed in class unless specifically called for
  - Not even for notes or following along with demos
  - Exceptions need documentation

Textbook

There is a textbook:

Database Systems: The Complete Book, 2nd edition
by Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom

You can buy it, rent it, or find a PDF copy online. Paperback or rental is about $60.

Should you buy it?

- I will use it for teaching, but do not plan to require readings
- Lots of good info, and if you are someone who will go to the text for more info, might be worth it
- Most people probably do not need to buy it

CodeGrinder

You should have a Linux (including WSL) or Mac OS environment to work on

- We will use CodeGrinder for autograding most assignments
- I recommend Debian 12 (Bookworm) or Ubuntu 22.04 for WSL users
- First steps: install CodeGrinder, sqlite3, and Python
  - `sudo apt install sqlite3 python3`

Example

Databases usually model something from the real world.

Say I want to store information for a music player: artists and albums.

Two kinds of data:

1. Artists
2. Albums

Flat files

It would be pretty easy to store this info in a couple of files, maybe using CSV (a lot of data science works this way):

- One file for each kind of data
- One line for each entry, commas/tabs between fields of each entry

To use the data:

- Open the file in Python
- Parse the file a line at a time (the standard library can do this)
- Making changes? Re-write the entire file

Flat file example

```
Artists
+-----------------------------------+
| name                   country    |
+-----------------------------------+
| "Radiohead",           "England"  |
| "Franz Ferdinand",     "Scotland" |
| "The Killers",         "USA"      |
+-----------------------------------+

Albums
+---------------------------------------------+
| name              artist               year |
+---------------------------------------------+
| "OK Computer",    "Radiohead",         1997 |
| "Kid A",          "Radiohead",         2000 |
| "Hot Fuss",       "The Killers",       2004 |
+---------------------------------------------+
```

Tasks we might want to perform:

- Find all artists from England
- Count all albums from 2000

```python
for line in artists_file:
    fields = parse(line)
    if fields[1] == 'England':
        print(fields[0])

count = 0
for line in albums_file:
    fields = parse(line)
    if fields[2] == 2000:
        count += 1
print('found', count, 'albums from 2000')
```

Flat file problems

Data integrity:

- Is the artist name spelled consistently?
- Are the years all valid years?
- Are any artists present in an album listing but missing from the artist table?
- What if a band changes its name?

Complexity:

- What if an album has multiple artists?

Implementation:

- What if I have millions of artists and albums?
- What if another application wants to use my database?
    - Taking turns? (redundant code)
    - Concurrently? (data corruption)

Durability:

- What if it crashes while updating a record?
- What if I want to replicate it for high availability?

Relational databases

A relational database handles these problems for you:

- Data integrity: enforce rules about individual fields and about the relationships between different data
- Complexity: rich and flexible model that can represent complex data, normalization
- Implementation: applications use a query language, DBMS figures out how to execute the query efficiently
- Durability: ACID transactions protect against concurrent queries, crashes, and consistency violations

Key idea: separate data modeling and indexing from querying

Counterexample: key-value stores and denormalization

One of the most successful and ubiquitous classes of software ever made

Relational model

A *tuple* is a set of attribute values, also called a *record*.

- The values of a tuple are normally atomic/scalar, though modern databases relax this
- The special value *NULL* is a member of every domain
- Tuples are often called *rows* and attributes *columns*

A *relation* is an unordered set of *tuples* with the same attributes, also called a *table*.

Primary keys

A relation's *primary key* uniquely identifies a single tuple

- A *natural key* is composed of data that is part of the record
- A *surrogate key* or *synthetic key* is an identifier (usually an integer) added to a tuple purely to serve as a unique identifier

```
Artists
+---------------------------------------+
| id   name                  country    |
+---------------------------------------+
| 374  Radiohead             England    |
| 569  Franz Ferdinand       Scotland   |
| 725  The Killers           USA        |
+---------------------------------------+
```

- All popular databases can auto-generate an integer primary key at tuple insertion time.
- If you do not request it (and make it part of the tuple) some will still generate an integer id and hide it from you

Foreign keys

A *foreign key* is a set of attributes in a relation that refers to the primary key of another relation.

```
Artists
+-------------------------------------+
| id   name                  country  |
+-------------------------------------+
| 374  Radiohead             England  |
| 569  Franz Ferdinand       Scotland |
| 725  The Killers           USA      |
+-------------------------------------+

Albums
+---------------------------------+
| name          artist_id  year |
+---------------------------------+
| OK Computer   374        1997 |
| Kid A         374        2000 |
| Hot Fuss      725        2004 |
+---------------------------------+
```

This example permits artists with many albums, but not albums with many artists, a 1-to-many arrangement.

Associative tables

To represent a many-to-many relationship, add a new relation that links two tables (and possibly holds other attributes relavent to the connection).

```
Problems                             Problem set <--> Problem               Problem sets
+------------------------------+     +-------------------------------------+ +---------------------------+
| id  problem_name    problem |     | problem_id  problem_set_id  weight | | id  problem_set   points |
|  1  SQL warmups     ...      |     |          1               1       1 | |  1  Week 1        10       |
|  2  B-tree scanner  ...      |     |          2               3       1 | |  2  Week 2        10       |
|  3  Query planner   ...      |     |          3               3       2 | |  3  Review        5        |
+------------------------------+     +-------------------------------------+ +---------------------------+
```

By expressing these as foreign key relationships, the database can ensure that all the problems in a problem set actually exist, etc.

Crash course in SQL

- https://sqlbolt.com/

The relational algebra