6.005 Elements of Software Construction

Fall 2008

6.005 elements of software construction

Relational Databases

Rob Miller
Fall 2008

Databases

Database servers sit behind big web sites like Amazon and eBay

- Databases are the standard way to maintain the state of a web site

Databases are embedded in many applications

- Firefox browsing history is stored as a database on disk
- Subversion stores your source code in a database

Embedded database is an alternative to saving and loading a file format

- Instead of saving Java heap objects to a file with a textual format like XML, you can store the data in a database instead

Benefits of Using a Database

Persistence

- Databases are **persistent** by default – updates to the database are immediately stored on disk
- Usually robust to program crashes and hardware reboots
- Contrast with objects in the Java heap, which disappear on a crash

Query performance

- Databases build and maintain **indexes** to answer complex queries quickly, e.g. "find books written by Stephen King in 2004"

Concurrency

- Databases provide an effective synchronization mechanism, **transactions**, that allows safe concurrent updates to a pile of relational data

Relational Databases

A relational database is a set of named tables

- A table has a fixed set of named **columns** (aka fields or attributes) and a varying set of unnamed **rows** (aka records or tuples)

Person table — 3 columns, 2 rows

First Name	LastName	Email
Daniel	Jackson	dnj@mit.edu
Rob	Miller	rcm@mit.edu

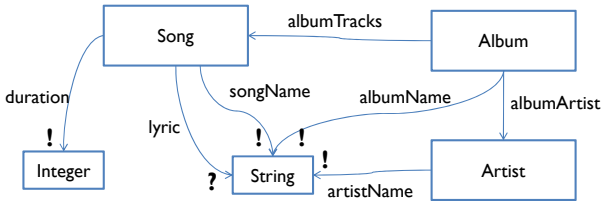
- Each cell in the table stores a value of a primitive data type
  - e.g. string, integer, date, time
  - object references are represented by integer IDs

A table represents a relation

- In general, a mathematical relation is a set of *n*-tuples (a binary relation is special case, which is a set of pairs)

Example

An object model we want to store in a database



Pure Relational View

One table per binary relation

songName relation

songId	songName
1	Mr. Brightside
2	Somebody Told Me
3	Girlfriend

duration relation

songId	duration
1	4:17
2	5:57
3	3:42

lyric relation

songId	lyric
3	Hey, hey, you, you, I don't like your girlfriend...

albumName relation

albumId	albumName
101	Hot Fuss
102	The Best Damn Thing

albumTracks relation

albumId	songId
101	1
101	2
102	3

Class/Relation View

Often all the exactly-one (!) relations for a class are combined into a single table

Song relation

songId	songName	duration
1	Mr. Brightside	4:17
2	Somebody Told Me	5:57
3	Girlfriend	3:24

- This table actually represents the Song class
- Analogy to objects on the Java heap
  - **id** column is the object's address in memory, and other columns are fields of the object
- The **id** column is usually automatically generated by the database system so that all songs have a unique ID
  - Analogy: Java's **new** operator automatically generates a fresh address

Bad Designs

Relations with other multiplicities (+, *, ?) generally should not be combined

- Otherwise, ? relation would force columns to have empty cells

songId	songName	lyric
1	Mr. Brightside	
2	Somebody Told Me	
3	Girlfriend	Hey, hey, you, you, ...

  - Sometimes this is done anyway for performance reasons, just like nulls are sometimes useful for Java field values
- Multiplicity + and * would force columns to become arrays

albumId	albumName	albumTracks
101	Hot Fuss	1, 2, 3, ...

albumId	albumName	track1	track2	track3	track4
101	Hot Fuss	1	2	3	4

Querying a Relational Database

SQL ("Structured Query Language")

- SQL is a standard language for querying (and mutating) a relational database
- Most database systems support some flavor of SQL
- SQL's SELECT statement offers a compact language for retrieving subsets of relational data
  - Find all songs longer than 5 minutes

    SELECT songName FROM Song WHERE duration > 300
- If you know nothing else about SQL, you should know about SELECT
  - Note that SQL is case-insensitive, so SELECT and select are the same, as are songName and songname

Relational Algebra

SELECT is based on a few simple operations that can be performed on relations

- Each operation takes one or more relations and produces a relation
  - PROJECT filters the columns
  - SELECT filters the rows
  - PRODUCT adjoins columns from two relations
  - RENAME renames columns
- A relation is a set of rows, so the usual set operations also apply
  - UNION
  - INTERSECTION
  - DIFFERENCE

Projection

Projection keeps a set of named columns and discards the rest

SELECT **songId, duration**
FROM Song

songId	duration
1	4:17
2	4:57
3	5:57

Selection

Selection keeps the subset of rows that match a predicate and discards the rest

SELECT *
FROM Song
WHERE duration > 300

songId	songName	duration
2	Somebody Told Me	5:57

- Like filtering on the rows

Product

Cartesian product

- The Cartesian product of two relations R1 and R2 is the result of concatenating each row in R1 with all rows in R2

```
SELECT  *
FROM    Song, Album
```

songId	songName	duration	albumId	albumName
1	Mr. Brightside	4:17	101	Hot Fuss
2	Somebody Told Me	5:57	101	Hot Fuss
3	Girlfriend	3:24	101	Hot Fuss
1	Mr. Brightside	4:17	102	The Best Damn Thing
2	Somebody Told Me	5:57	102	The Best Damn Thing
3	Girlfriend	3:24	102	The Best Damn Thing

Joins

A join is a special case of Cartesian product

- When the two relations share a column, we only want to concatenate rows that have the same value for that column

```
SELECT  *
FROM    Song, AlbumTracks
WHERE   Song.songId = AlbumTracks.songId
```

use the table name to disambiguate columns with the same name

songId	songName	duration	albumId	songId
1	Mr. Brightside	4:17	101	1
~~2~~	~~Somebody Told Me~~	~~5:57~~	~~101~~	~~1~~
~~3~~	~~Girlfriend~~	~~3:24~~	~~101~~	~~1~~
~~1~~	~~Mr. Brightside~~	~~4:17~~	~~101~~	~~2~~
2	Somebody Told Me	5:57	102	2
~~3~~	~~Girlfriend~~	~~3:24~~	~~102~~	~~2~~

- Join can be represented by a product followed by a selection

Question

How do I get a list of songName, albumName pairs?

songName	albumName
Mr. Brightside	Hot Fuss
Somebody Told Me	Hot Fuss
Girlfriend	The Best Damn Thing

```
SELECT   songName, albumName
FROM     Song, AlbumTracks, Album
WHERE    Song.songId = AlbumTracks.songId
         AND AlbumTracks.albumId = Album.albumId
```


Other Set Operations

Union, intersection, difference of relations

- Find songs longer than 5 minutes or containing "midnight" in the lyric

```
SELECT songId FROM Song WHERE duration > 300
  UNION
SELECT songId FROM Lyrics WHERE lyric LIKE '%midnight%'
```

- Find songs longer than 5 minutes for which we have the lyrics

```
SELECT songId FROM Song WHERE duration > 300
  INTERSECT
SELECT songId FROM Lyrics
```

- Find albums that don't have any tracks

```
SELECT albumId FROM Album
  EXCEPT
SELECT albumId FROM AlbumTracks
```

- These operations are rarely used in practice, because select predicates can usually do the job, and database systems are good at optimizing SELECT

Aggregate Functions

Accumulating a column of data into a single value

- How long is the album Thriller?

```
SELECT   SUM(duration)
FROM     Song, Album, AlbumTracks
WHERE    Song.songId = AlbumTracks.songId
         AND Album.albumId = AlbumTracks.albumId
         AND Album.albumName = "Thriller"
```

SUM
10:14

Other aggregate functions

- AVG
- COUNT
- MAX
- MIN

Grouping

GROUP BY computes aggregate functions on subsets of the tuples

- How long is each album?

```
SELECT    albumName, SUM(duration)
FROM      Song, Album, AlbumTracks
WHERE     Song.songId = AlbumTracks.songId
          AND Album.albumId = AlbumTracks.albumId
GROUP BY  albumName
```

albumName	SUM
Hot Fuss	10:14
The Best Damn Thing	3:47

Exercise

Write SELECT statements for the following queries

- Find the name of the album with the song named "Girlfriend"
- Find names of albums for which we have lyrics (for at least one song)
- List all albums, showing album name and number of songs

Mutating the Database

Insert a row

```
INSERT INTO Song
VALUES (4, "Thriller", 6:02)
```

Update rows

```
UPDATE Song
SET songName="Smile Like You Mean It", duration=4:57
WHERE songId = 1
```

Delete rows

```
DELETE FROM Song
WHERE songName = "Girlfriend"
```

songId	songName	duration
1	Mr. Brightside	4:17
2	Somebody Told Me	5:57
3	Girlfriend	3:24

Concurrency in Databases

Transactions allow concurrent database modifications

- A transaction is a block of SQL statements that need to execute together

Transactions implement ACID semantics

- Atomicity – either the full effects of a transaction are recorded or no trace of it will be found
- Consistency – a transaction is recorded only if it preserves invariants
  - e.g., every AlbumTrack row must contain an albumId that exists in Album and a songId that exists in Song
- Isolation – if two transactions operate on the same data, the outcome will always be same as executing them sequentially one after the other
- Durability – if the transaction completes, its effects will never be lost

Transaction Example

Transfer money between bank accounts

```
BEGIN TRANSACTION
SELECT balance FROM Account WHERE accountId = 1
      and put it in local variable balance1
SELECT balance FROM Account WHERE accountId = 2
      and call it balance2
balance1 -= 100
balance2 += 100
UPDATE Account SET balance=balance1 WHERE accountId = 1
UPDATE Account SET balance=balance2 WHERE accountId = 2
COMMIT
```


Transactions vs. Locks

Transaction is tentative until successful commit

- COMMIT fails if a simultaneous transaction changed the same rows and managed to commit first
- If commit fails, the transaction is **rolled back** – i.e., it has no effect on the database
- Your program can retry the transaction if the commit failed

Database handles low-level concurrency mechanisms

- e.g. it may lock the rows touched, or detect conflicts at commit time

Transactions are widely considered easier to program

- locking discipline and granularity (database, table, row) is managed by the database implementer
- programmer just has to think about which statements need to execute in isolation, without acquiring or releasing locks
- active research on **transactional memory** is trying to bring the notion of transactions to the shared memory paradigm (like Java objects)

Summary

Relations as database tables

- Relational database is a relation-centric implementation of an object model

Normal form

- All rows are unique, no entries can be null

Relational algebra for querying

- Project, select, and join operators combine relations
- SQL **select** statement uses all three operators

Transactions support concurrency

- Widely considered easier than locks