

COS 316

Precept:

SQL

History

- E. F. Codd. 1970. A relational model of data for large shared data banks. Commun. ACM 13, 6 (June 1970)
 - definitive model for relational database management systems (RDBMS).
- 1970s. Structured English Query Language (SEQUEL)
 - IBM Corporation based on Codd's model
 - Called SEQUEL later became SQL (still pronounced "sequel")
 - Structured Query Language
- 1979. Relational Software, Inc. (now Oracle) introduced the first commercially available implementation of SQL
- 1986. ANSI Standard
- Today. ANSI/ISO Standard
 - Although many vendors have their own variations

Relational Database

- What really is relational? ~**tables**
 - Present data as a collection of tables
 - Use “relational” operators to manipulate data in tabular form
- A table represents one “entity type”
- A row represents an instance of that type
 - Rows are called **records**
 - Unique key to identify each row.
- Columns are called **attributes**
- Link to rows in other tables by adding a column for unique keys of the linked row in other tables
 - Foreign keys

Tables, Tuples and Attributes

BOOKS ← tables / relations

ID	TITLE	YEAR	AUTHOR
198	Harry Potter and the Philosopher's Stone	1997	ROWLING
090	Game of Thrones	1991	MARTIN
134	A Clash of Kings	1992	MARTIN
<i>INTEGER</i>	<i>TEXT</i>	<i>INTEGER</i>	<i>TEXT</i>

tuples/ records

primary key

Each column has an attribute / type

Tables, Tuples and Attributes

BOOKS

KEY	TITLE	YEAR	AUTHOR
198	Lord of the Rings	1954	1712
090	Game of Thrones	1991	2000
134	A Clash of Kings	1992	2000

AUTHORS

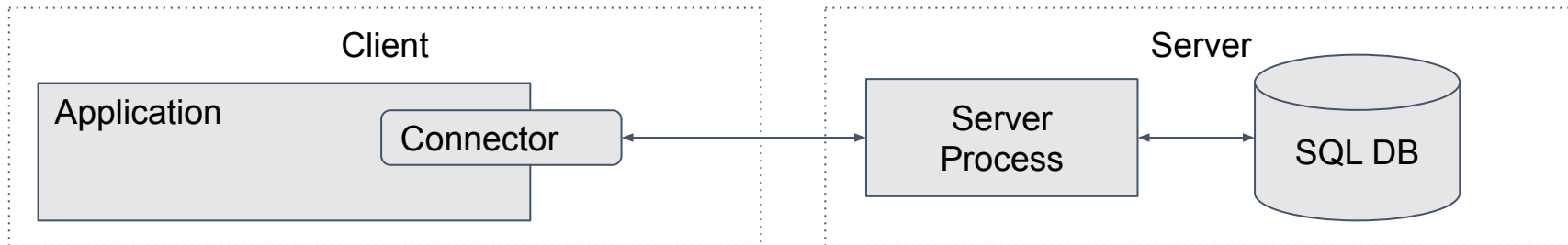
KEY	FIRST	LAST	YEAR
1712	J RR	Tolkien	1892
2000	George RR	Martin	1948
1311	Charles	Dickens	1812

Popular RDBMS

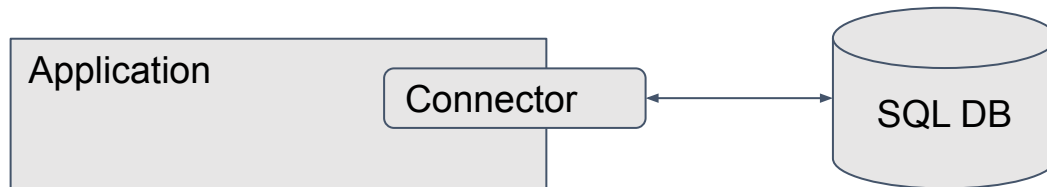
- MySQL – <https://www.mysql.com>
- Postgres – <https://www.postgresql.org>
- SQLite – <https://www.sqlite.org>
 - lightweight in terms of setup, database administration, and required resource
 - features: self-contained, serverless, zero-configuration, transactional

RDBMS Architectures

MySQL, Postres, etc.



SQLite



SQLite Storage Classes*

- NULL
 - Value is a NULL value
- INTEGER
 - Value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value
- REAL
 - Value is a floating point value, stored as an 8-byte IEEE floating point number
- TEXT
 - Value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE)
- BLOB.
 - The value is a blob of data, stored exactly as it was input

* <https://www.sqlite.org/datatype3.html>

Using SQLite - Setup

```
> cd <Precepts repo>
> git pull                # update with precept6
> cd precept6
> go get github.com/mattn/go-sqlite3
```

Using SQLite - Windows

- Start the X server on their host machine.
 - Windows key to open search, then type XLaunch
- In the command prompt on the host machine, before running `vagrant ssh`, set the environment variable `DISPLAY` to `localhost:0.0`
 - In a cmd shell, type: `set DISPLAY=localhost:0.0`
 - In a bash shell, type: `export DISPLAY=localhost:0.0`
 - Run `vagrant ssh` as normal

Using SQLite - Locally

- SQLite 3 should already be installed
- Optional: download DB Browser for SQLite

<https://sqlitebrowser.org/dl>

Exercise Dataset

- MovieLens –

<https://grouplens.org/datasets/movielens/>

MovieLens

4 different tables contained in the MovieLens database:

- Movies
 - movieId: represent the movie id
 - title : represent the full movie title
 - year : year of release
 - genre : a pipe-separated list of genres associated with the movie
- Links
 - movieId: represent the movie id
 - imdbId : can be used to generate a link to the IMDb site
 - tmdbId : can be used to generate a link to the The Movie DB site
- Ratings (made by users)
 - userId & movieId: represent the user id and movie id
 - rating : uses a 5-star scale, with 0.5 star increments
 - timestamp : use the epoch format (seconds since midnight of January 1, 1970 on UTC time zone)
- Tags (added by users)
 - userId & movieId: represent the user id and movie id
 - tag : represent user-generated textual metadata
 - timestamp : use the epoch format (seconds since midnight of January 1, 1970 on UTC time zone)

Go and SQL (1) - Import SQLite Database Driver

```
import (  
    "database/sql"  
    _ "github.com/mattn/go-sqlite3"  
)
```

- Load database driver anonymously, aliasing its package qualifier to _
 - none of its exported names are visible
- Driver registers itself as being available to the database/sql package, but in general nothing else happens with the exception that the init function is run.

Go and SQL (2) - Opening a Database

```
conn, err := sql.Open("sqlite3",  
    "file:MovieLens.db")
```

- Create a sql.DB using sql.Open()
- First argument: driver name - driver uses to register itself with database/sql
- Second argument: driver-specific syntax that tells the driver how to access the underlying datastore
 - See <https://github.com/mattn/go-sqlite3>

Go and SQL (3) - Data types

Go	SQLite
<code>nil</code>	<code>null</code>
<code>int</code>	<code>integer</code>
<code>int64</code>	<code>integer</code>
<code>float64</code>	<code>real</code>
<code>bool</code>	<code>integer</code>
<code>[]byte</code>	<code>blob</code>
<code>string</code>	<code>text</code>
<code>time.Time</code>	<code>timestamp/datetime</code>

Go and SQL (4) - Queries

```
var (  
    title string  
    genres string  
)  
rows, err := db.Query("select title, genres from Movies where year = 1933;")  
if err != nil {  
    log.Fatal(err)  
}  
defer rows.Close()  
for rows.Next() {  
    err := rows.Scan(&title, &genres)  
    if err != nil {  
        log.Fatal(err)  
    }  
    log.Println(title, genres)  
}  
err = rows.Err()  
if err != nil {  
    log.Fatal(err)  
}
```

Go and SQL (5) - More Queries

```
err = db.QueryRow("select title from Movies where movieId = ?", 1).Scan(&title)
if err != nil {
    log.Fatal(err)
}
fmt.Println(title)
```

Go and SQL (6) - Preparing Queries

```
stmt, err := db.Prepare("select title from Movies where year = ?")
if err != nil {
    log.Fatal(err)
}
defer stmt.Close()
rows, err = stmt.Query(1995)
if err != nil {
    log.Fatal(err)
}
defer rows.Close()
for rows.Next() {
    err := rows.Scan(&title)
    if err != nil {
        log.Fatal(err)
    }
    log.Println(title)
}
if err = rows.Err(); err != nil {
    log.Fatal(err)
}
```

Go and SQL (7) - Updates

```
stmt, err = db.Prepare("INSERT INTO movies(movieId,title, year, genres) VALUES(?,?,?,?)")
if err != nil {
    log.Fatal(err)
}
res, err := stmt.Exec(193611,"Terminator: Dark Fate", 2019, "Action|Sci-Fi|Thriller")
if err != nil {
    log.Fatal(err)
}
lastId, err := res.LastInsertId()
if err != nil {
    log.Fatal(err)
}
rowCnt, err := res.RowsAffected()
if err != nil {
    log.Fatal(err)
}
log.Printf("ID = %d, affected = %d\n", lastId, rowCnt)
```

Go and SQL Exercise

1. Write a function to find and print the oldest movies in the database
2. Write a function to find and print a movie by name
3. Use JOIN (<https://www.sqlitetutorial.net/sqlite-join/>) to list movie titles alongside their ratings

Go and SQL Exercise - Solutions

1. Write a function to find and print the oldest movies in the database:
 - a. `rows, err := db.Query("select * from Movies order by year asc")`
2. Write a function to find and print a movie by name:
 - a. `rows, err := db.Query("select * from Movies where title = ?", title)`
3. Use JOIN (<https://www.sqlitetutorial.net/sqlite-join/>) to list movie titles alongside their ratings:
 - a. `rows, err := db.Query("select title, rating from Movies as t1 INNER JOIN Ratings as t2 on t1.movieId = t2.movieId")`