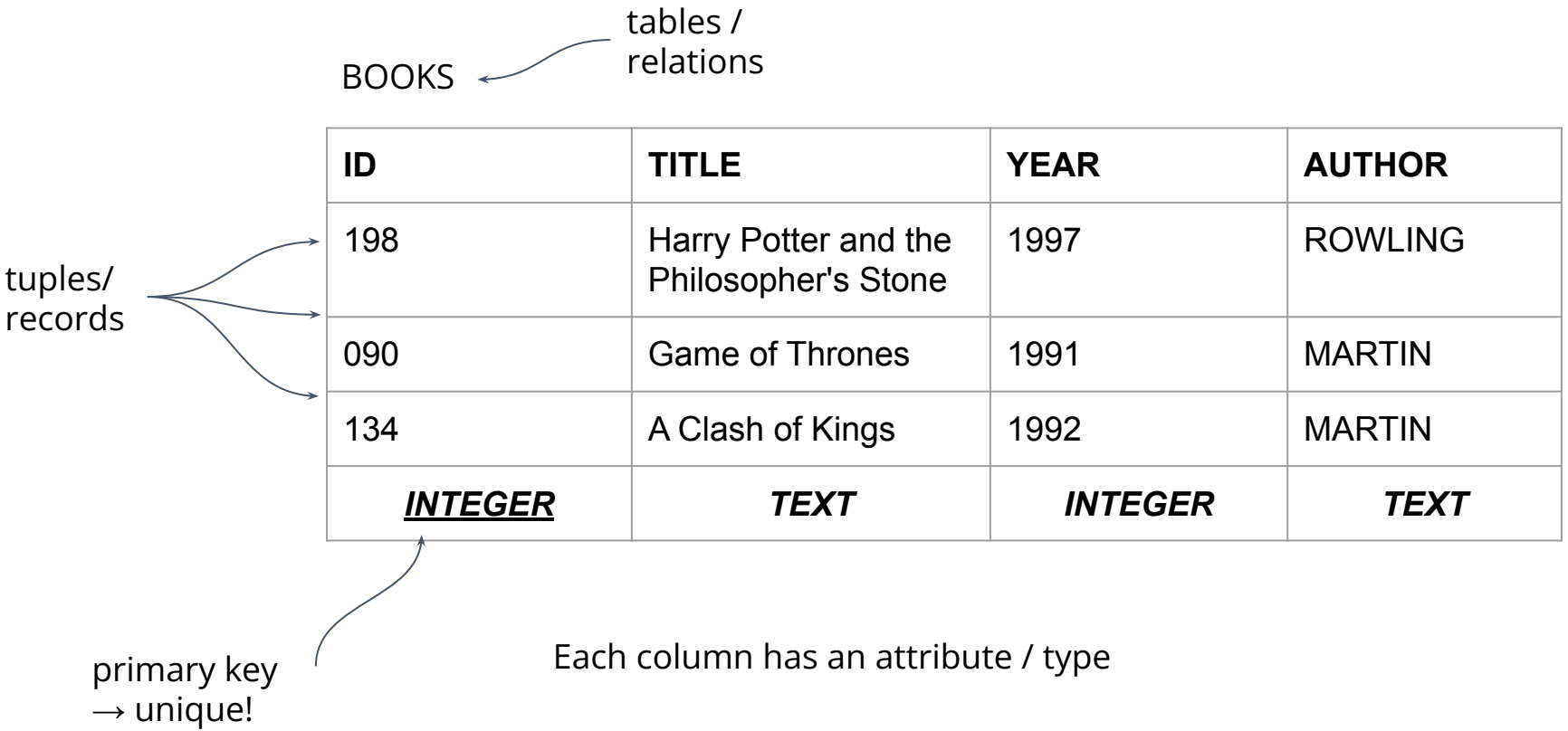
COS 316 Precept #5: SQL

Relational Database

- What is a *relational* database?
  - Present data as a collection of tables
  - Use “relation” operators to manipulate data across tables
- A table represents one “entity type” / “class”
- 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



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

"*Foreign key*" relation

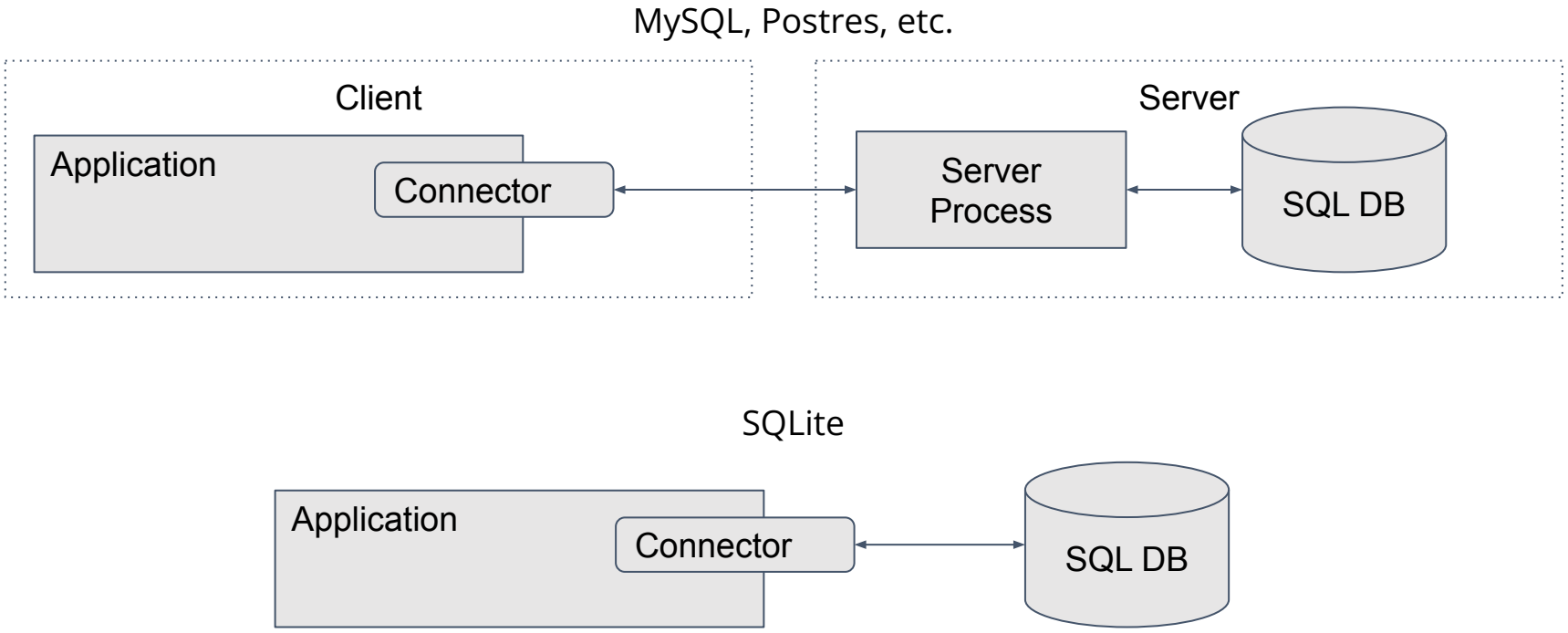
AUTHORS

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 setup, database administration, resource overheads
  - features: self-contained, serverless, zero-configuration, transactional

RDBMS Architectures



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

→ SQLite has *flexible* typing:

An INTEGER column can store TEXT, etc.

Advice: don't mix types!

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

Using SQLite - Setup

```
> cd <Precepts repo>
> git pull                    # update with precept5
> cd precept5
> go build .
> (go get github.com/mattn/go-sqlite3)
```

Using SQLite - Locally

- SQLite 3 should already be installed on OS X
- SQLite Installation: https://www.sqlite.org/download.html
- Optional: download DB Browser for SQLite

  https://sqlitebrowser.org/dl

Exercise Dataset

MovieLens: https://grouplens.org/datasets/movielens/

→ Already in an SQLite database in the Precepts repository!

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

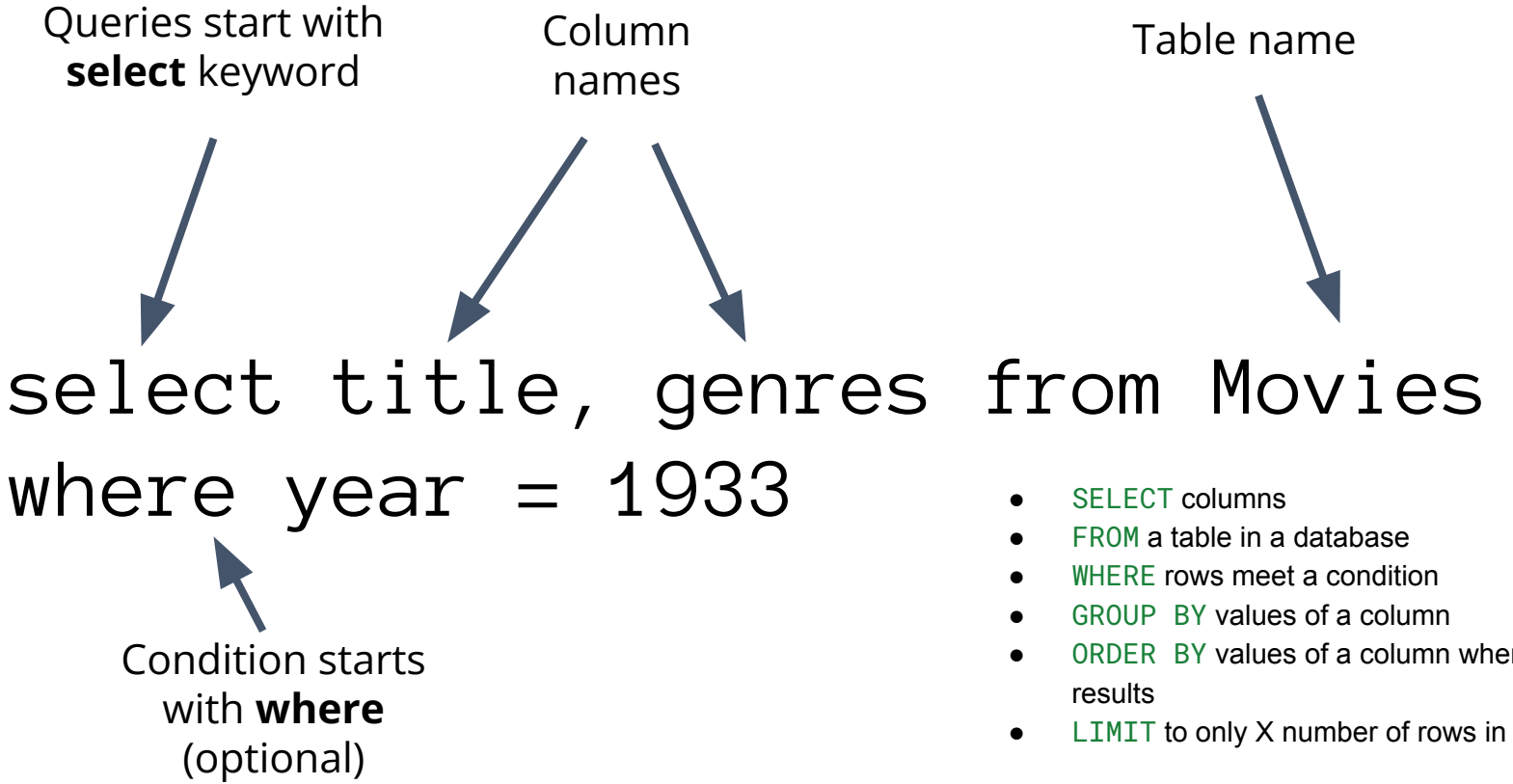
```
db, err := sql.Open("sqlite3",
                    "file:MovieLens.db")
```

- 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
`nil`	`null`
`int`	`integer`
`int64`	`integer`
`float64`	`real`
`bool`	`integer`
`[]byte`	`blob`
`string`	`text`
`time.Time`	`timestamp/datetime`

What is an SQL query?



- `SELECT` columns
- `FROM` a table in a database
- `WHERE` rows meet a condition
- `GROUP BY` values of a column
- `ORDER BY` values of a column when displaying results
- `LIMIT` to only X number of rows in resulting table

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

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)`

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)`