Relational Databases and SQL, Continued
Representing Relationships in a Database
Database Normalization (3NF)

- There is *tons* of database theory - way too much to understand without excessive predicate calculus
- Do not replicate data - reference data - point at data
- Use integers for keys and for references
- Add a special “key” column to each table which we will make references to. By convention, many programmers call this column “id”

Three Kinds of Keys

- **Primary key** - generally an integer auto-increment field

- **Logical key** - What the outside world uses for lookup

- **Foreign key** - generally an integer key pointing to a row in another table
Key Rules

Best practices

- Never use your **logical key** as the **primary key**

- **Logical keys** can and do change, albeit slowly

- **Relationships** that are based on matching string fields are less efficient than integers

User id

login

password

name

e-mail

created_at

modified_at

login_at
Foreign Keys

- A **foreign key** is when a table has a column that contains a key which points to the **primary key** of another table.

- When all primary keys are integers, then all foreign keys are integers - this is good - very good
Relationship Building (in tables)
<table>
<thead>
<tr>
<th>Track</th>
<th>Rating</th>
<th>Artist</th>
<th>Album</th>
<th>Genre</th>
<th>Len</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hells Bells</td>
<td>5:13</td>
<td>AC/DC</td>
<td>Who Made Who</td>
<td>Rock</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>Shake Your Foundations</td>
<td>3:54</td>
<td>AC/DC</td>
<td>Who Made Who</td>
<td>Rock</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>Chase the Ace</td>
<td>3:01</td>
<td>AC/DC</td>
<td>Who Made Who</td>
<td>Rock</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>For Those About To Rock (We ...</td>
<td>5:54</td>
<td>AC/DC</td>
<td>Who Made Who</td>
<td>Rock</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>Dúlamán</td>
<td>3:43</td>
<td>Altan</td>
<td>Natural Wonders M...</td>
<td>New Age</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Rode Across the Desert</td>
<td>4:10</td>
<td>America</td>
<td>Greatest Hits</td>
<td>Easy Listen...</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>Now You Are Gone</td>
<td>3:08</td>
<td>America</td>
<td>Greatest Hits</td>
<td>Easy Listen...</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Tip Mane</td>
<td>3:20</td>
<td>America</td>
<td>Greatest Hits</td>
<td>Easy Listen...</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>
Album

belongs-to

Track
Title
Rating
Len
Count

Table
Primary key
Logical key
Foreign key

Album
id
title

Track
id
title
rating
len
count
album_id
Table
Primary key
Logical key
Foreign key

Naming FK artist_id is a convention
Using Join Across Tables

Relational Power

- By removing the replicated data and replacing it with references to a single copy of each bit of data we build a “web” of information that the relational database can read through very quickly - even for very large amounts of data.

- Often when you want some data it comes from a number of tables linked by these foreign keys.
The JOIN Operation

• The JOIN operation links across several tables as part of a select operation

• You must tell the JOIN how to use the keys that make the connection between the tables using an ON clause
select Album.title, Artist.name from Album join Artist on Album.artist_id = Artist.id

What we want to see
The tables that hold the data
How the tables are linked
```
select Album.title, Album.artist_id, Artist.id, Artist.name
from Album join Artist on Album.artist_id = Artist.id
```
SELECT Track.title, Track.genre_id, Genre.id, Genre.name
FROM Track JOIN Genre

Joining two tables without an ON clause gives all possible combinations of rows.
select Track.title, Genre.name from Track join Genre on Track.genre_id = Genre.id

What we want to see

The tables that hold the data

How the tables are linked
select Track.title, Artist.name, Album.title, Genre.name
from Track join Genre join Album join Artist on
Track.genre_id = Genre.id and Track.album_id = Album.id and Album.artist_id = Artist.id

What we want to see
The tables which hold the data
How the tables are linked
<table>
<thead>
<tr>
<th>title</th>
<th>name</th>
<th>genre</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hells Bells</td>
<td>AC/DC</td>
<td>Rock</td>
<td>5:13</td>
</tr>
<tr>
<td>Shake Your Foundations</td>
<td>AC/DC</td>
<td>Rock</td>
<td>3:54</td>
</tr>
<tr>
<td>Chase the Ace</td>
<td>AC/DC</td>
<td>Rock</td>
<td>3:01</td>
</tr>
<tr>
<td>For Those About To Rock</td>
<td>AC/DC</td>
<td>Rock</td>
<td>5:54</td>
</tr>
<tr>
<td>Dúlamán</td>
<td>Altan</td>
<td>New Age</td>
<td>3:43</td>
</tr>
<tr>
<td>Rode Across the Desert</td>
<td>America</td>
<td>Easy Listen</td>
<td>4:10</td>
</tr>
<tr>
<td>Now You Are Gone</td>
<td>America</td>
<td>Easy Listen</td>
<td>3:08</td>
</tr>
<tr>
<td>Tin Man</td>
<td>America</td>
<td>Easy Listen</td>
<td>3:30</td>
</tr>
<tr>
<td>Sister Golden Hair</td>
<td>America</td>
<td>Easy Listen</td>
<td>3:22</td>
</tr>
<tr>
<td>Track 01</td>
<td>Billy Price</td>
<td>Danger Zone</td>
<td>4:22</td>
</tr>
<tr>
<td>Track 02</td>
<td>Billy Price</td>
<td>Danger Zone</td>
<td>2:45</td>
</tr>
<tr>
<td>Track 03</td>
<td>Billy Price</td>
<td>Danger Zone</td>
<td>3:26</td>
</tr>
<tr>
<td>Track 04</td>
<td>Billy Price</td>
<td>Danger Zone</td>
<td>4:17</td>
</tr>
<tr>
<td>Track 05</td>
<td>Billy Price</td>
<td>Danger Zone</td>
<td>3:50</td>
</tr>
<tr>
<td>War Pigs/Luke's Wall</td>
<td>Black Sabbath</td>
<td>PA</td>
<td>7:58</td>
</tr>
<tr>
<td>Paranoid</td>
<td>Black Sabbath</td>
<td>PA</td>
<td>2:53</td>
</tr>
<tr>
<td>Planet Caravan</td>
<td>Black Sabbath</td>
<td>PA</td>
<td>4:35</td>
</tr>
<tr>
<td>Iron Man</td>
<td>Black Sabbath</td>
<td>PA</td>
<td>5:59</td>
</tr>
<tr>
<td>Electric Funeral</td>
<td>Black Sabbath</td>
<td>PA</td>
<td>4:53</td>
</tr>
<tr>
<td>Hand of Doom</td>
<td>Black Sabbath</td>
<td>PA</td>
<td>7:10</td>
</tr>
<tr>
<td>Rat Salad</td>
<td>Black Sabbath</td>
<td>PA</td>
<td>2:30</td>
</tr>
<tr>
<td>Jack the Stripper/Fairies Wear ...</td>
<td>Black Sabbath</td>
<td>PA</td>
<td>6:14</td>
</tr>
<tr>
<td>Bomb Squad (TECH)</td>
<td>Brent</td>
<td>Brent's Album</td>
<td>3:28</td>
</tr>
<tr>
<td>clay techno</td>
<td>Brent</td>
<td>Brent's Album</td>
<td>4:36</td>
</tr>
<tr>
<td>Heavy</td>
<td>Brent</td>
<td>Brent's Album</td>
<td>3:08</td>
</tr>
<tr>
<td>Hi metal man</td>
<td>Brent</td>
<td>Brent's Album</td>
<td>4:20</td>
</tr>
<tr>
<td>Mistro</td>
<td>Brent</td>
<td>Brent's Album</td>
<td>2:58</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>title</th>
<th>name</th>
<th>genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Dog</td>
<td>Led Zeppelin</td>
<td>Rock</td>
</tr>
<tr>
<td>Stairway</td>
<td>Led Zeppelin</td>
<td>Rock</td>
</tr>
<tr>
<td>About to Rock</td>
<td>AC/DC</td>
<td>Metal</td>
</tr>
<tr>
<td>Who Made Who</td>
<td>AC/DC</td>
<td>Metal</td>
</tr>
</tbody>
</table>
Using Join Across Tables

Code Sample
tracks.py
https://www.py4e.com/code3.zip
Chapter 15
Many-To-Many Relationships

https://en.wikipedia.org/wiki/Many-to-many_(data_model)
The diagram illustrates a one-to-many relationship between an `Album` and a `Track`. The `Album` table has an `id` and a `title` column, while the `Track` table includes `id`, `title`, `rating`, `len`, `count`, and `album_id` columns. The `belongs-to` relationship is indicated by an arrow from `Album` to `Track`, showing that each album can have many tracks, but each track belongs to only one album.

The diagram also includes references to database tables and foreign keys, with links to Wikipedia for further reading on the topic of one-to-many relations in data modeling.
Many to Many

- Sometimes we need to model a relationship that is many-to-many
- We need to add a "connection" table with two foreign keys
- There is usually no separate primary key

https://en.wikipedia.org/wiki/Many-to-many_(data_model)
Many-to-many relationship between Course and User:

- Course
  - id
  - title
- Member
  - user_id
  - course_id
- User
  - id
  - name
  - email

https://en.wikipedia.org/wiki/Many-to-many_(data_model)
CREATE TABLE User (  id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  name TEXT UNIQUE,  email TEXT )

CREATE TABLE Course (  id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  title TEXT UNIQUE )

CREATE TABLE Member (  user_id INTEGER,  course_id INTEGER,  role INTEGER,  PRIMARY KEY (user_id, course_id) )

Start with a Fresh Database
CREATE TABLE Course {
  id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
  title TEXT
}

CREATE TABLE Member {
  user_id INTEGER,
  course_id INTEGER,
  PRIMARY KEY (user_id, course_id)
}

CREATE TABLE User {
  id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
  name TEXT,
  email TEXT
}

CREATE TABLE sqlite_sequence {
  name, seq
}

CREATE INDEX sqlite_autoindex_Course_1 ON Course (id);
CREATE INDEX sqlite_autoindex_Member_1 ON Member (user_id, course_id);
CREATE INDEX sqlite_autoindex_User_1 ON User (id, name, email);
Insert Users and Courses

```
INSERT INTO User (name, email) VALUES ('Jane', 'jane@tsugi.org');
INSERT INTO User (name, email) VALUES ('Ed', 'ed@tsugi.org');
INSERT INTO User (name, email) VALUES ('Sue', 'sue@tsugi.org');

INSERT INTO Course (title) VALUES ('Python');
INSERT INTO Course (title) VALUES ('SQL');
INSERT INTO Course (title) VALUES ('PHP');
```
INSERT INTO Member (user_id, course_id, role) VALUES (1, 1, 1);
INSERT INTO Member (user_id, course_id, role) VALUES (2, 1, 0);
INSERT INTO Member (user_id, course_id, role) VALUES (3, 1, 0);
INSERT INTO Member (user_id, course_id, role) VALUES (1, 2, 0);
INSERT INTO Member (user_id, course_id, role) VALUES (2, 2, 1);
INSERT INTO Member (user_id, course_id, role) VALUES (2, 3, 1);
INSERT INTO Member (user_id, course_id, role) VALUES (3, 3, 0);
### Table: Member

<table>
<thead>
<tr>
<th>user_id</th>
<th>course_id</th>
<th>role</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
SELECT User.name, Member.role, Course.title
FROM User JOIN Member JOIN Course
ON Member.user_id = User.id AND
Member.course_id = Course.id
ORDER BY Course.title, Member.role DESC, User.name
Many-To-Many Relationships

https://en.wikipedia.org/wiki/Many-to-many_(data_model)
Complexity Enables Speed

• Complexity makes speed possible and allows you to get very fast results as the data size grows.

• By normalizing the data and linking it with integer keys, the overall amount of data which the relational database must scan is far lower than if the data were simply flattened out.

• It might seem like a tradeoff - spend some time designing your database so it continues to be fast when your application is a success.
Additional SQL Topics

- **Indexes** improve access performance for things like string fields
- **Constraints** on data - (cannot be NULL, etc..)
- **Transactions** - allow SQL operations to be grouped and done as a unit
Summary

• Relational databases allow us to **scale** to very large amounts of data

• The key is to have **one copy of any data** element and use relations and joins to link the data to multiple places

• This greatly **reduces the amount of data which much be scanned** when doing complex operations across large amounts of data

• Database and SQL design is a bit of an **art form**
Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors here