Relational Database Design

Charles Severance
www.pg4e.com

http://www.pg4e.com/lectures/02-Database-Design-Many-to-Many.txt

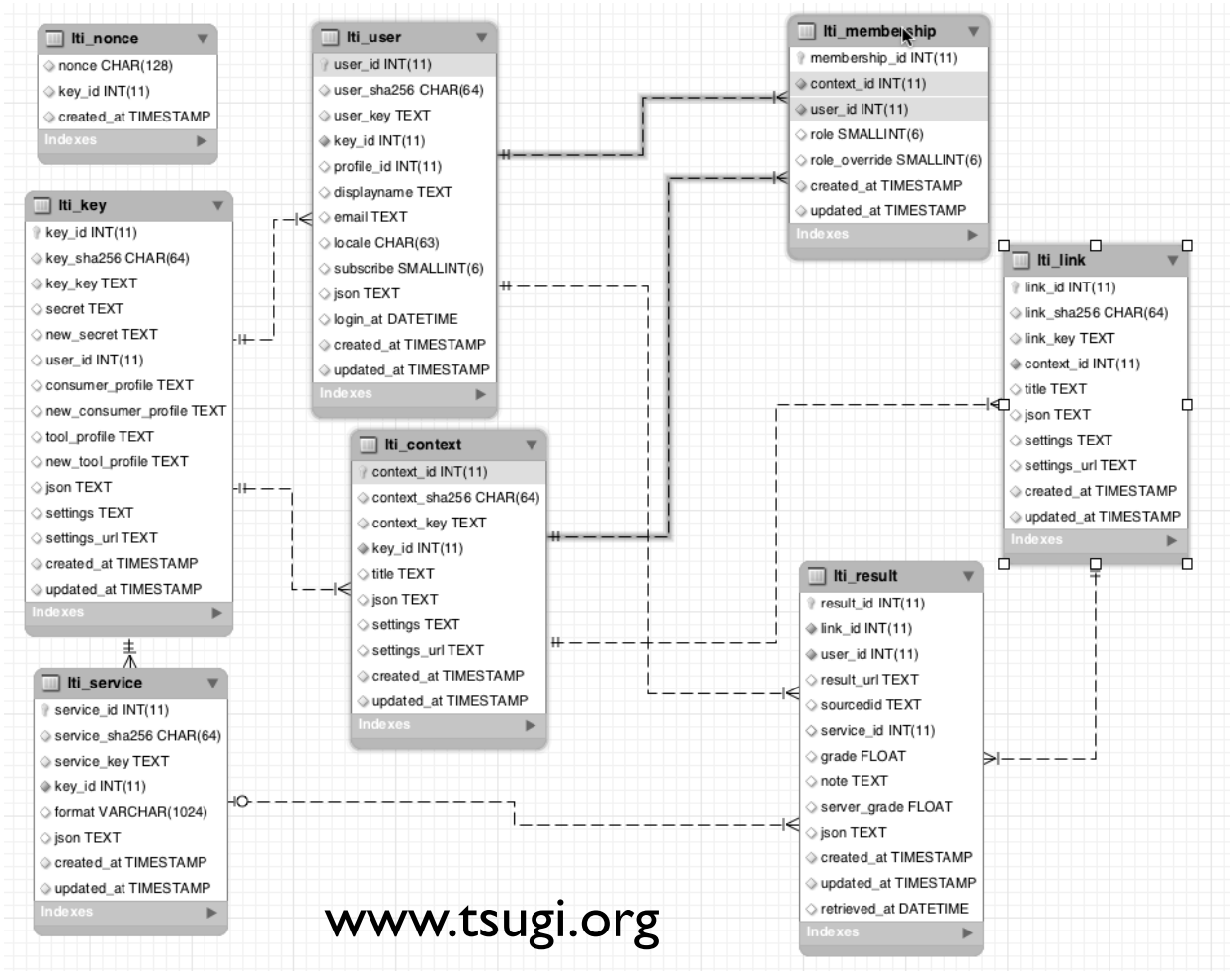


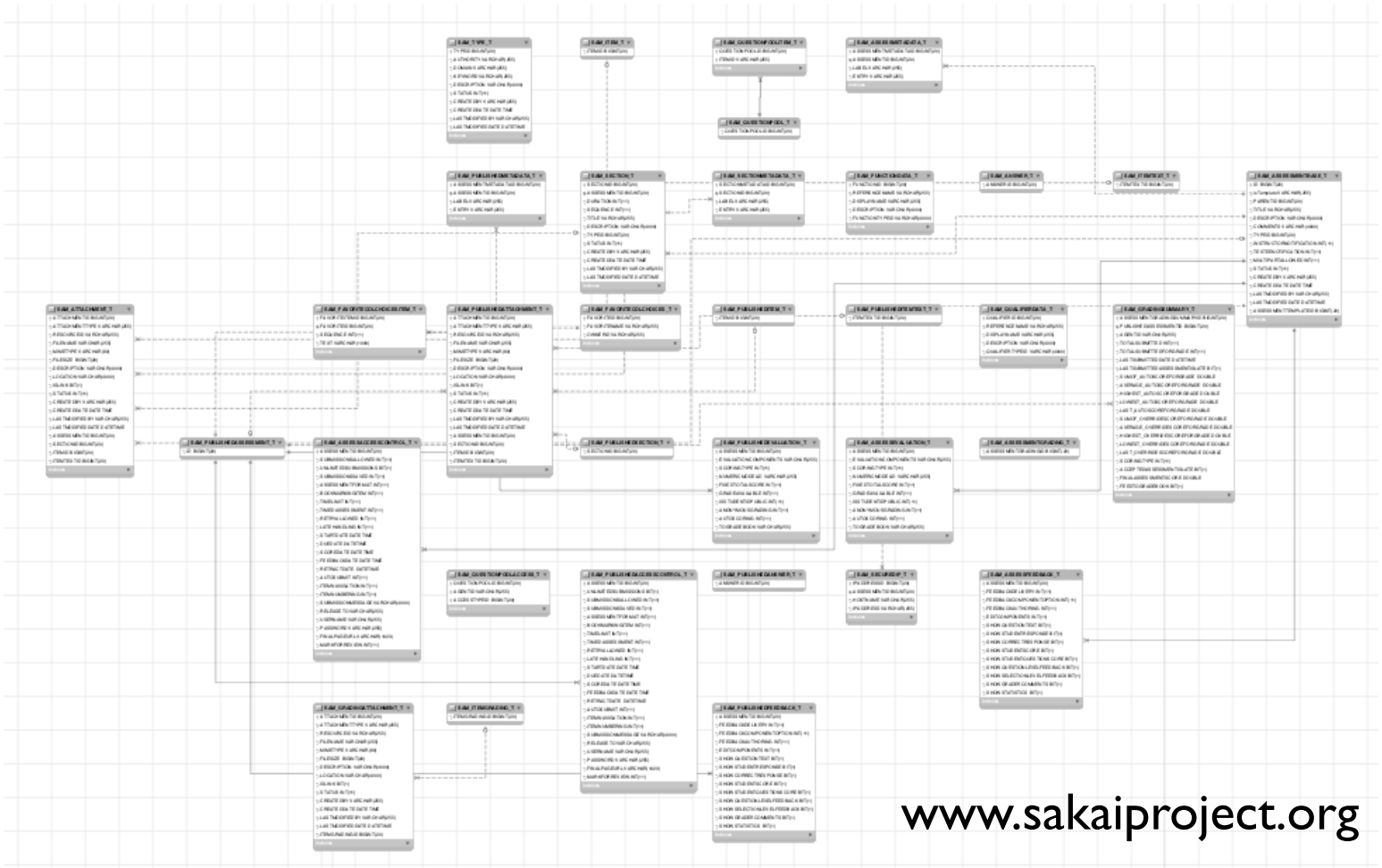
Relational Database Design

http://en.wikipedia.org/wiki/Relational_model

Database Design

- Database design is an art form of its own with particular skills and experience.
- Our goal is to avoid the really bad mistakes and design clean and easily understood databases.
- Others may performance tune things later.
- Database design starts with a picture...


lti_nonce
nonce CHAR(128)
key_id INT(11)
created_at TIMESTAMP
Indexes
lti_user
user_id INT(11)
user_sha256 CHAR(64)
user_key TEXT
key_id INT(11)
profile_id INT(11)
displayname TEXT
email TEXT
locale CHAR(63)
subscribe SMALLINT(6)
json TEXT
login_at DATETIME
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_membership
membership_id INT(11)
context_id INT(11)
user_id INT(11)
role SMALLINT(6)
role_override SMALLINT(6)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_key
key_id INT(11)
key_sha256 CHAR(64)
key_key TEXT
secret TEXT
new_secret TEXT
user_id INT(11)
consumer_profile TEXT
new_consumer_profile TEXT
tool_profile TEXT
new_tool_profile TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_link
link_id INT(11)
link_sha256 CHAR(64)
link_key TEXT
context_id INT(11)
title TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_context
context_id INT(11)
context_sha256 CHAR(64)
context_key TEXT
key_id INT(11)
title TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_result
result_id INT(11)
link_id INT(11)
user_id INT(11)
result_url TEXT
sourcedid TEXT
service_id INT(11)
grade FLOAT
note TEXT
server_grade FLOAT
json TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
retrieved_at DATETIME
Indexes
lti_service
service_id INT(11)
service_sha256 CHAR(64)
service_key TEXT
key_id INT(11)
format VARCHAR(1024)
json TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
www.tsugi.org



www.sakaiproject.org

Building a Data Model

- Drawing a picture of the data objects for our application and then figuring out how to represent the objects and their relationships
- Basic Rule: Don't put the same string data in twice - use a relationship instead
- When there is one thing in the "real world" there should only be one copy of that thing in the database

Track	Len	Artist	Album	Genre	Rating	Count
Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18
Tin Man	3:30	America	Greatest Hits	Easy Listen...	★★★★★	23
Sister Golden Hair	3:22	America	Greatest Hits	Easy Listen...	★★★★★	24
Track 01	4:22	Billy Price	Danger Zone	Blues/R&B	★★★★★	26
Track 02	2:45	Billy Price	Danger Zone	Blues/R&B	★★★★★	18
Track 03	3:26	Billy Price	Danger Zone	Blues/R&B	★★★★★	22
Track 04	4:17	Billy Price	Danger Zone	Blues/R&B	★★★★★	18
Track 05	3:50	Billy Price	Danger Zone	Blues/R&B	★★★★★	21
War Pigs/Luke's Wall	7:58	Black Sabbath	Paranoid	Metal	★★★★★	25
Paranoid	2:53	Black Sabbath	Paranoid	Metal	★★★★★	22
Planet Caravan	4:35	Black Sabbath	Paranoid	Metal	★★★★★	25
Iron Man	5:59	Black Sabbath	Paranoid	Metal	★★★★★	26
Electric Funeral	4:53	Black Sabbath	Paranoid	Metal	★★★★★	22
Hand of Doom	7:10	Black Sabbath	Paranoid	Metal	★★★★★	23
Rat Salad	2:30	Black Sabbath	Paranoid	Metal	★★★★★	31
Jack the Stripper/Fairies Wear ...	6:14	Black Sabbath	Paranoid	Metal	★★★★★	24
Bomb Squad (TECH)	3:28	Brent	Brent's Album			1
clay techno	4:36	Brent	Brent's Album			2
Heavy	3:08	Brent	Brent's Album			1
Hi metal man	4:20	Brent	Brent's Album			1
Mistro	2:58	Brent	Brent's Album			1

For each “piece of info”...

- Is the column an object or an attribute of another object?
- Once we define objects, we need to define the relationships between objects.

Len

Album

Genre

Artist

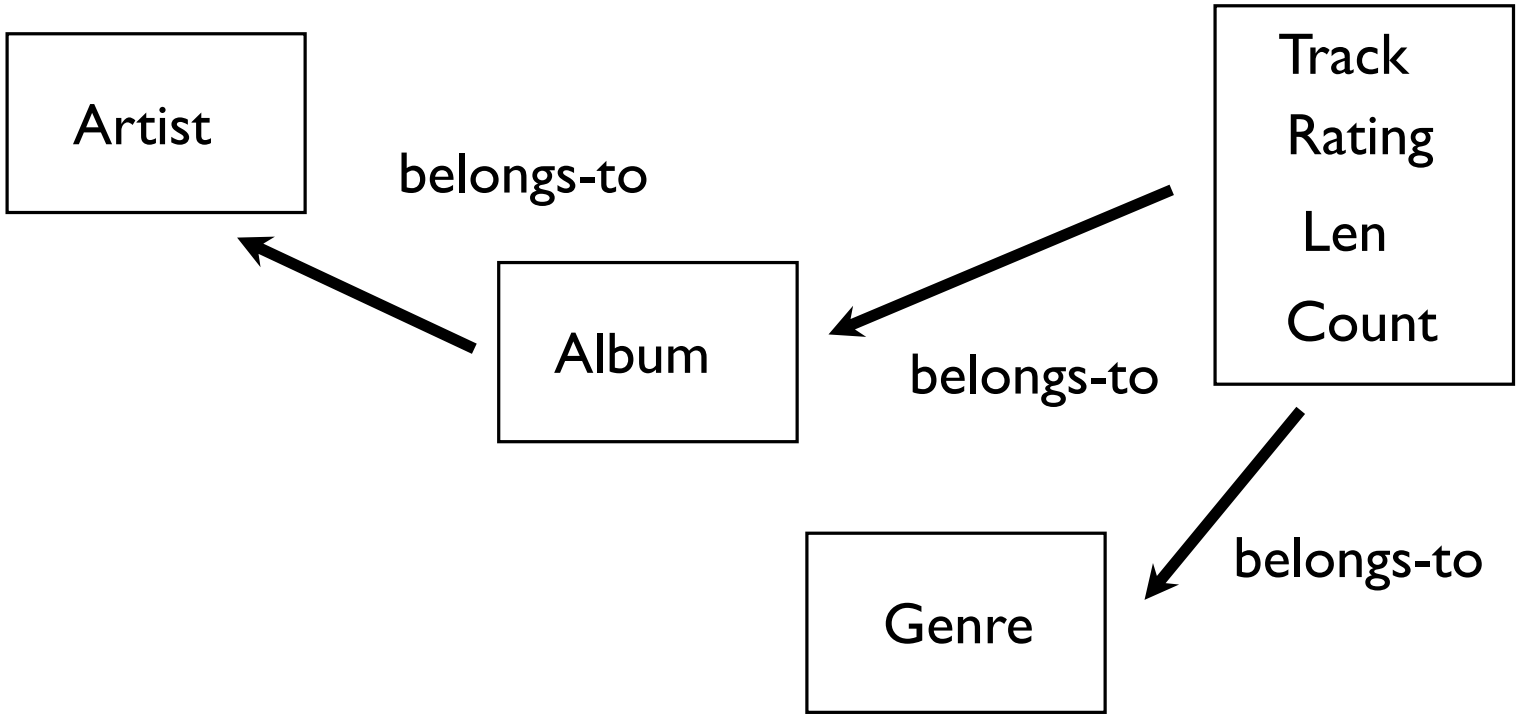
Rating

Track

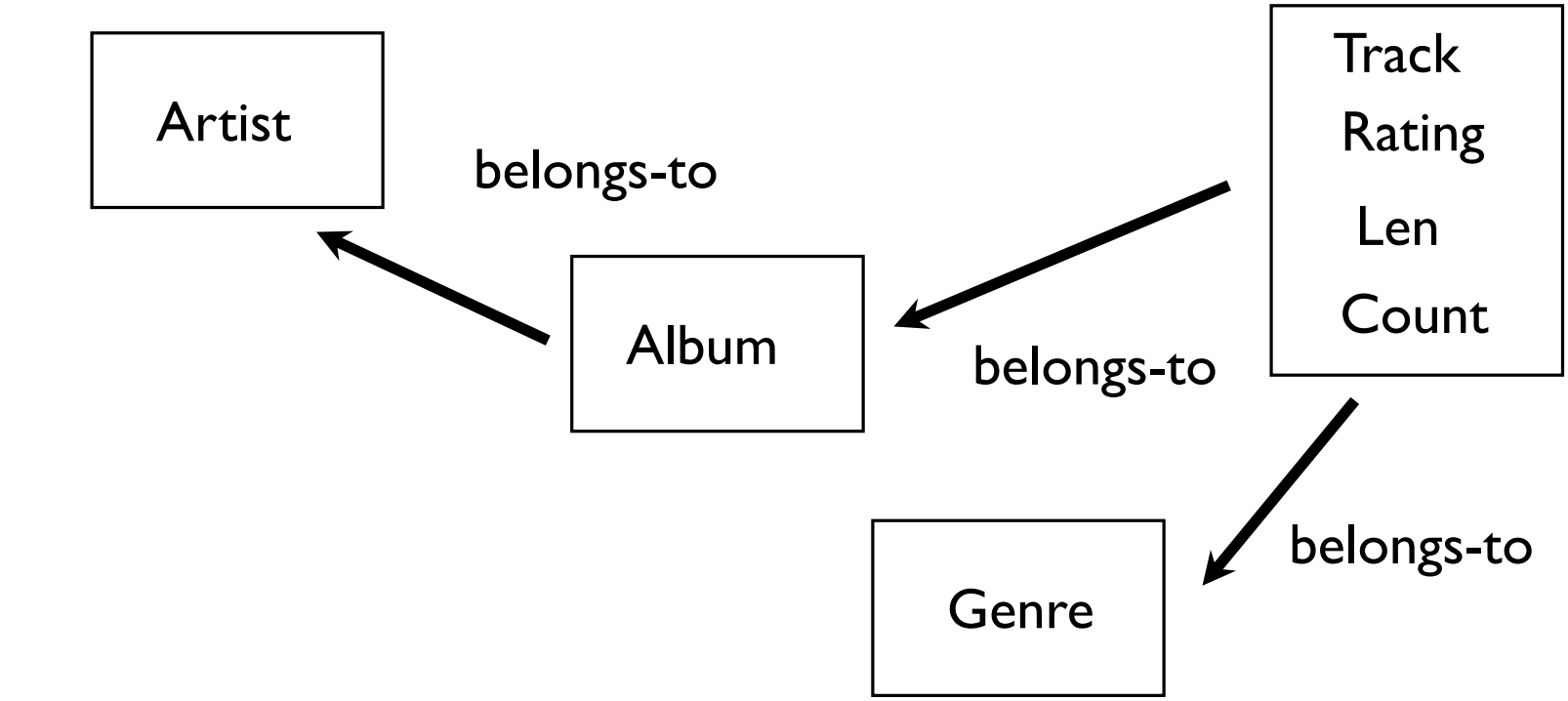
Count

Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18

Track
Album
Artist
Genre
Rating
Len
Count



Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We …	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M…	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen…	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen…	★★★★★	18



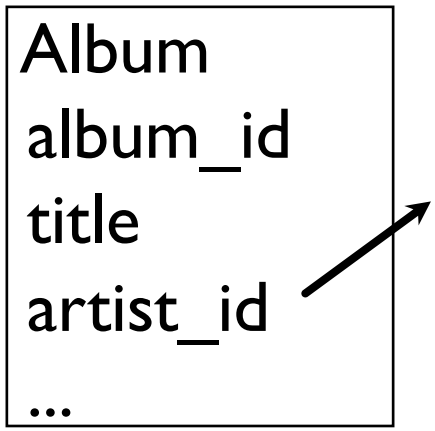
Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We …	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M…	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen…	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen…	★★★★★	18

Key Terminology

Finding our way around....

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



Primary Key Rules

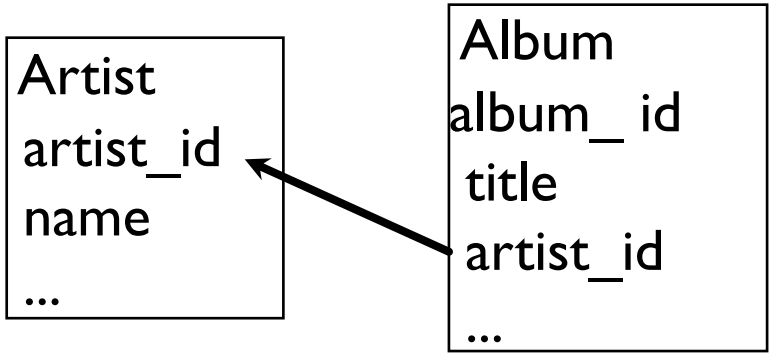
Best practices:

- 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
user_id
email
password
name
created_at
modified_at
login_at

Foreign Keys

- A foreign key is when a table has a column containing a key that 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.



Normalization and Foreign Keys



Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We …	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M…	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen…	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen…	★★★★★	18

We want to keep track of which band is the "creator" of each music track...
What album does this song "belong to"?

Which album is this song related to?

Database Normalization (3NF)

There is *tons* of database theory - way too much to understand without excessive predicate calculus

- Do not replicate data. Instead, reference data. Point at data.
- Use integers for keys and for references.
- Add a special “key” column to each table, which you will make references to.

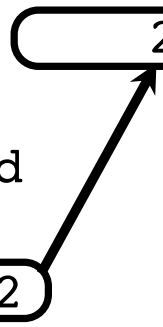
http://en.wikipedia.org/wiki/Database_normalization

Integer Reference Pattern

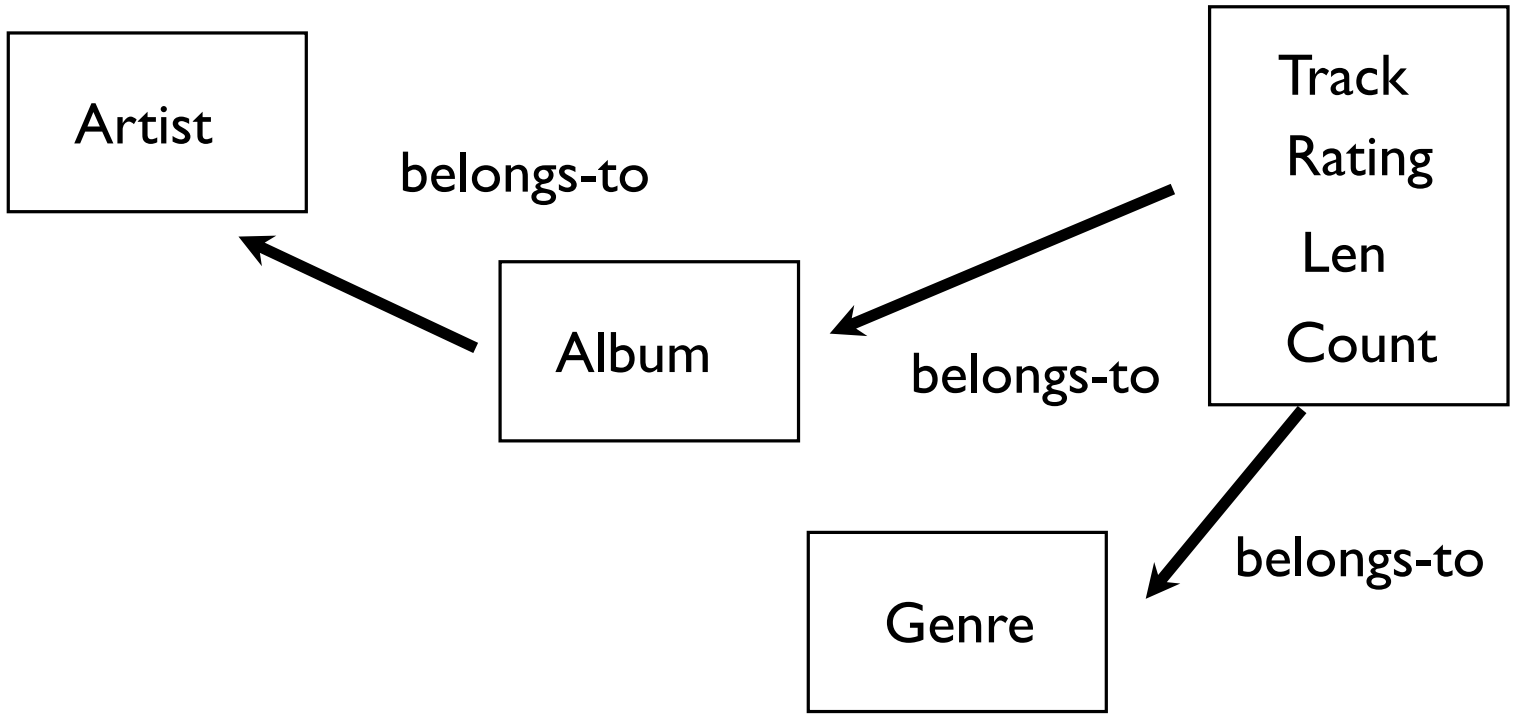
We use integer columns in one table to reference (or look up) rows in another table.

```
music=> SELECT * FROM artist;
 id |     name
----+-------------
  1 | Led Zeppelin
  2 | AC/DC
```

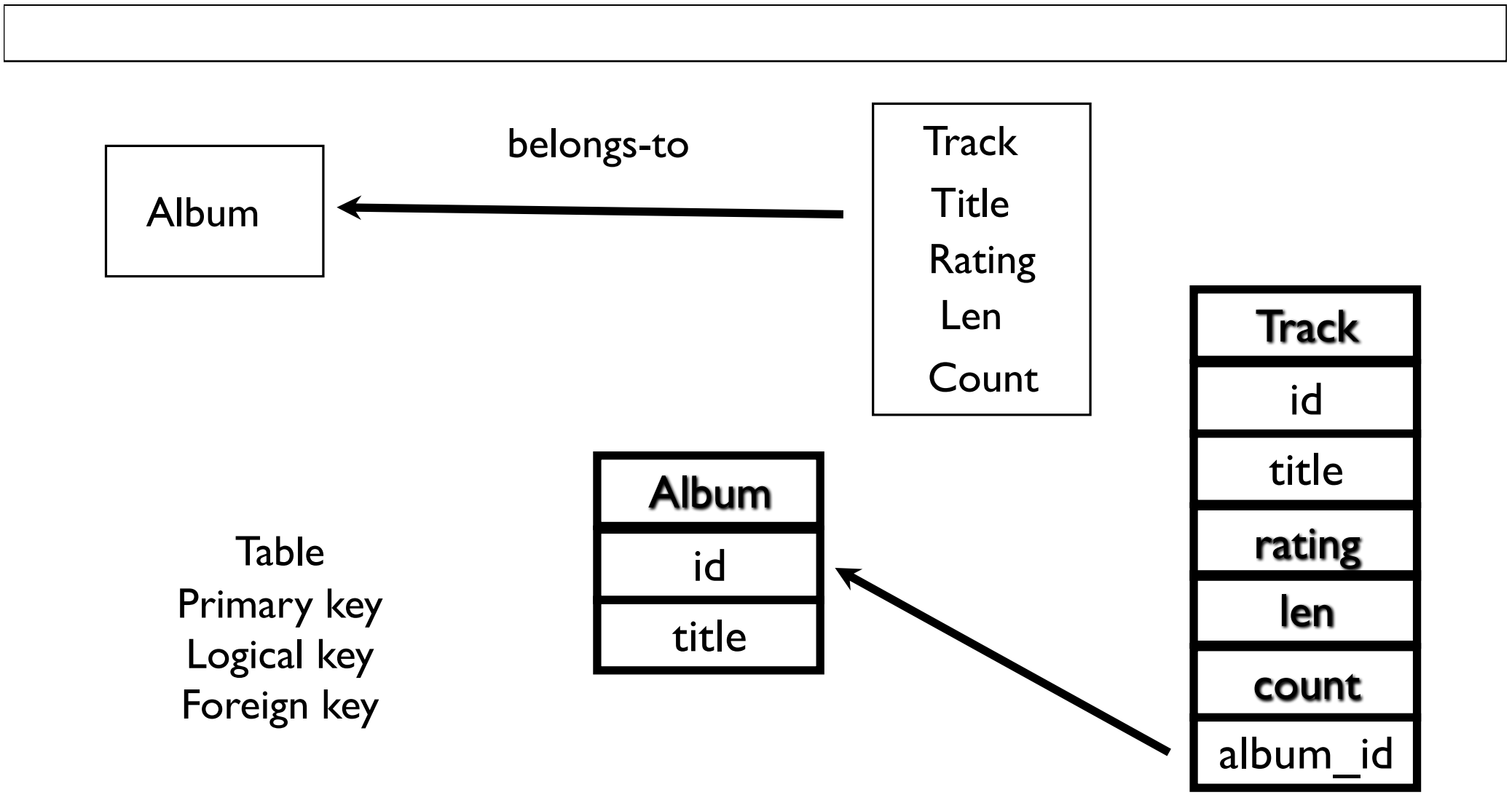
```
music=> SELECT * FROM album;
 id |     title      | artist_id
----+----------------+-----------
  1 | Who Made Who   |         2
  2 | IV             |         1
```

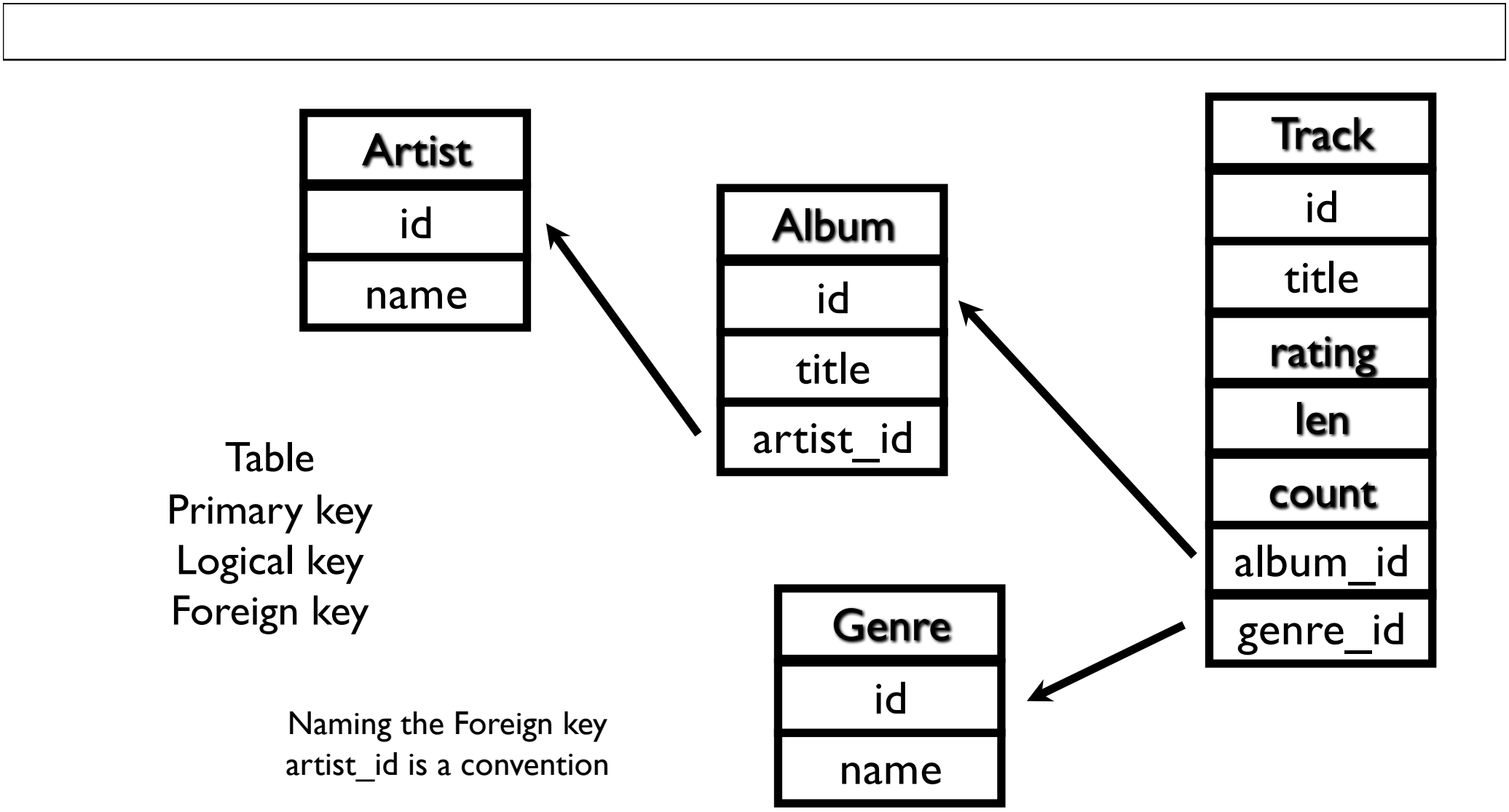


Building a Physical Data Schema



Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We …	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M…	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen…	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen…	★★★★★	18


Album
belongs-to
Track
Title
Rating
Len
Count
Track
id
title
rating
len
count
album_id
Album
id
title
Table
Primary key
Logical key
Foreign key


Artist
id
name
Album
id
title
artist_id
Track
id
title
rating
len
count
album_id
genre_id
Genre
id
name
Table
Primary key
Logical key
Foreign key
Naming the Foreign key
artist_id is a convention

Creating our Music Database

```
sudo -u postgres psql postgres

postgres=# CREATE DATABASE music
    WITH OWNER 'pg4e' ENCODING 'UTF8';
CREATE DATABASE
postgres=#
```

```
CREATE TABLE artist (
  id SERIAL,
  name VARCHAR(128) UNIQUE,
  PRIMARY_KEY(id)
);

CREATE TABLE album (
  id SERIAL,
  title VARCHAR(128) UNIQUE,
  artist_id INTEGER REFERENCES artist(id) ON DELETE CASCADE,
  PRIMARY KEY(id)
);
```

```
CREATE TABLE genre (
  id SERIAL,
  name VARCHAR(128) UNIQUE,
  PRIMARY_KEY(id)
);

CREATE TABLE track (
  id SERIAL,
  title VARCHAR(128),
  len INTEGER,
  rating INTEGER,
  count INTEGER,
  album_id INTEGER REFERENCES genre(id) ON DELETE CASCADE,
  genre_id INTEGER REFERENCES album(id) ON DELETE CASCADE,
  UNIQUE(title, album_id),
  PRIMARY KEY(id)
);
```

```
music=> \d track
                                  Table "public.track"
  Column  |          Type          |                      Modifiers
----------+------------------------+-----------------------------------------------
 id       | integer                | not null default nextval('track_id_seq'::regclass)
 title    | character varying(128) |
 len      | integer                |
 rating   | integer                |
 count    | integer                |
 album_id | integer                |
 genre_id | integer                |
Indexes:
    "track_pkey" PRIMARY KEY, btree (id)
    "track_title_album_id_key" UNIQUE CONSTRAINT, btree (title, album_id)
Foreign-key constraints:
    "track_album_id_fkey" FOREIGN KEY (album_id) REFERENCES album(id) ON DELETE CASCADE
    "track_genre_id_fkey" FOREIGN KEY (genre_id) REFERENCES genre(id) ON DELETE CASCADE

music=>
```

```
music=> INSERT INTO artist (name) VALUES ('Led Zeppelin');
INSERT 0 1
music=> INSERT INTO artist (name) VALUES ('AC/DC');
INSERT 0 1
music=> SELECT * FROM artist;
 id |    name
----+------------
  1 | Led Zeppelin
  2 | AC/DC
(2 rows)

music=>
```

```
music=> INSERT INTO album (title, artist_id) VALUES ('Who Made Who', 2);
INSERT 0 1
music=> INSERT INTO album (title, artist_id) VALUES ('IV', 1);
INSERT 0 1
music=> SELECT * FROM album;
 id |    title     | artist_id
----+-------------+----------
  1 | Who Made Who |         2
 2 | IV           |         1
(2 rows)
```

```
music=> INSERT INTO genre (name) VALUES ('Rock');
INSERT 0 1
music=> INSERT INTO genre (name) VALUES ('Metal');
INSERT 0 1
music=> SELECT * FROM genre;
 id | name
----+------
  1 | Rock
  2 | Metal
(2 rows)
```

```
music=> INSERT INTO track (title, rating, len, count, album_id, genre_id)
music->     VALUES ('Black Dog', 5, 297, 0, 2, 1) ;
INSERT 0 1
music=> INSERT INTO track (title, rating, len, count, album_id, genre_id)
music->     VALUES ('Stairway', 5, 482, 0, 2, 1) ;
INSERT 0 1
music=> INSERT INTO track (title, rating, len, count, album_id, genre_id)
music->     VALUES ('About to Rock', 5, 313, 0, 1, 2) ;
INSERT 0 1
music=> INSERT INTO track (title, rating, len, count, album_id, genre_id)
music->     VALUES ('Who Made Who', 5, 207, 0, 1, 2) ;
INSERT 0 1
music=> SELECT * FROM track;
 id |     title     | len | rating | count | album_id | genre_id
----+---------------+-----+--------+-------+----------+----------
  1 | Black Dog     | 297 |      5 |     0 |        2 |        1
  2 | Stairway      | 482 |      5 |     0 |        2 |        1
  3 | About to Rock | 313 |      5 |     0 |        1 |        2
  4 | Who Made Who  | 207 |      5 |     0 |        1 |        2
(4 rows)
```

```
music=> SELECT * FROM track;
 id |     title      | len | rating | count | album_id | genre_id
----+----------------+-----+--------+-------+----------+----------
  1 | Black Dog      | 297 |      5 |     0 |        2 |        1
  2 | Stairway       | 482 |      5 |     0 |        2 |        1
  3 | About to Rock  | 313 |      5 |     0 |        1 |        2
  4 | Who Made Who   | 207 |      5 |     0 |        1 |        2
```

```
music=> SELECT * FROM album;
 id |    title     | artist_id
----+--------------+-----------
  1 | Who Made Who |         2
 2 | IV           |         1
```

```
music=> SELECT * FROM genre;
 id | name
----+------
  1 | Rock
  2 | Metal
```

```
music=> SELECT * FROM artist;
 id |     name
----+-------------
  1 | Led Zeppelin
  2 | AC/DC
```

We Have Relationships!

Using Join Across Tables

http://en.wikipedia.org/wiki/Join_(SQL)

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.

```
music=> SELECT * FROM album;
 id |    title     | artist_id
----+--------------+----------
  1 | Who Made Who |         2
 2 | IV           |         1
```

```
music=> SELECT * FROM artist;
 id |    name
----+------------
  1 | Led Zeppelin
 2 | AC/DC
```



```
music=> SELECT album.title, artist.name
music->     FROM album JOIN artist
music->     ON album.artist_id = artist.id;
    title     |    name
--------------+-------------
 Who Made Who | AC/DC
 IV           | Led Zeppelin
```

What we want to see
The tables that hold the data
How the tables are linked

```
music=> SELECT * FROM album;
 id |    title     | artist_id
----+--------------+-----------
  1 | Who Made Who |         2
  2 | IV           |         1
```

```
music=> SELECT * FROM artist;
 id |    name
----+------------
  1 | Led Zeppelin
  2 | AC/DC
```

```
music=> SELECT album.title, album.artist_id, artist.id, artist.name
music->     FROM album INNER JOIN artist ON album.artist_id = artist.id;
    title     | artist_id | id |    name
--------------+-----------+----+------------
 Who Made Who |         2 |  2 | AC/DC
 IV           |         1 |  1 | Led Zepplin
```

```
music=> SELECT track.title, track.genre_id, genre.id, genre.name
music->     FROM track CROSS JOIN genre;
     title      | genre_id | id | name
----------------+---------+----+------
 Black Dog      |         1 |  1 | Rock
 Stairway       |         1 |  1 | Rock
 About to Rock  |         2 |  1 | Rock
 Who Made Who   |         2 |  1 | Rock
 Black Dog      |         1 |  2 | Metal
 Stairway       |         1 |  2 | Metal
 About to Rock  |         2 |  2 | Metal
 Who Made Who   |         2 |  2 | Metal
```

```
music=> SELECT * FROM track;
 id |     title      | len | rating | count | album_id | genre_id
----+----------------+-----+--------+-------+----------+----------
  1 | Black Dog      | 297 |      5 |     0 |        2 |        1
  2 | Stairway       | 482 |      5 |     0 |        2 |        1
  3 | About to Rock  | 313 |      5 |     0 |        1 |        2
  4 | Who Made Who   | 207 |      5 |     0 |        1 |        2
```

```
music=> SELECT * FROM genre;
 id | name
----+------
  1 | Rock
  2 | Metal
```

```
music=> SELECT track.title, genre.name
music->     FROM track JOIN genre
music->     ON track.genre_id = genre.id;
     title      | name
----------------+-------
 Black Dog      | Rock
 Stairway       | Rock
 About to Rock  | Metal
 Who Made Who   | Metal
```

It Can Get Complex...

```
music=> SELECT track.title, artist.name, album.title, genre.name
music-> FROM track
music->     JOIN genre ON track.genre_id = genre.id
music->     JOIN album ON track.album_id = album.id
music->     JOIN artist ON album.artist_id = artist.id;

     title      |    name       |    title      | genre
----------------+---------------+---------------+-------
 Black Dog      | Led Zeppelin | IV             | Rock
 Stairway       | Led Zeppelin | IV             | Rock
 About to Rock | AC/DC         | Who Made Who  | Metal
 Who Made Who   | AC/DC         | Who Made Who  | Metal
```

Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18
Tin Man	3:30	America	Greatest Hits	Easy Listen...	★★★★★	23
Sister Golden Hair	3:22	America	Greatest Hits	Easy Listen...	★★★★★	24
Track 01	4:22	Billy Price	Danger Zone	Blues/R&B	★★★★★	26
Track 02	2:45	Billy Price	Danger Zone	Blues/R&B	★★★★★	18
Track 03	3:26	Billy Price	Danger Zone	Blues/R&B	★★★★★	22
Track 04	4:17	Billy Price	Danger Zone	Blues/R&B	★★★★★	18
Track 05	3:50	Billy Price	Danger Zone	Blues/R&B	★★★★★	21
War Pigs/Luke's Wall						
Paranoid						
Planet Caravan						
Iron Man						
Electric Funeral						
Hand of Doom						
Rat Salad						
Jack the Stripper/Fair						
Bomb Squad (TECH)						
clay techno						
Heavy						
Hi metal man	4:20	Brent	Brent's Album			1
Mistro	2:58	Brent	Brent's Album			1

title	name	title	name
Black Dog	Led Zeppelin	IV	Rock
Stairway	Led Zeppelin	IV	Rock
About to Rock	AC/DC	Who Made Who	Metal
Who Made Who	AC/DC	Who Made Who	Metal

ON DELETE CASCADE

Child

```
music=> SELECT * FROM track;
 id |     title      | len | rating | count | album_id | genre_id
----+----------------+-----+--------+-------+----------+----------
  1 | Black Dog      | 297 |      5 |     0 |        2 |        1
  2 | Stairway       | 482 |      5 |     0 |        2 |        1
  3 | About to Rock  | 313 |      5 |     0 |        1 |        2
  4 | Who Made Who   | 207 |      5 |     0 |        1 |        2
```

```
music=> SELECT * FROM genre;
 id | name
----+------
  1 | Rock
2 | Metal
```

Parent

We are telling Postgres to
"clean up" broken references

```
DELETE FROM Genre WHERE name = 'Metal'
```

ON DELETE CASCADE

```
music=> SELECT * FROM track;
 id |     title     | len | rating | count | album_id | genre_id
----+---------------+-----+--------+-------+----------+----------
  1 | Black Dog     | 297 |      5 |     0 |        2 |        1
  2 | Stairway      | 482 |      5 |     0 |        2 |        1
  3 | About to Rock | 313 |      5 |     0 |        1 |        2
  4 | Who Made Who  | 207 |      5 |     0 |        1 |        2
(4 rows)

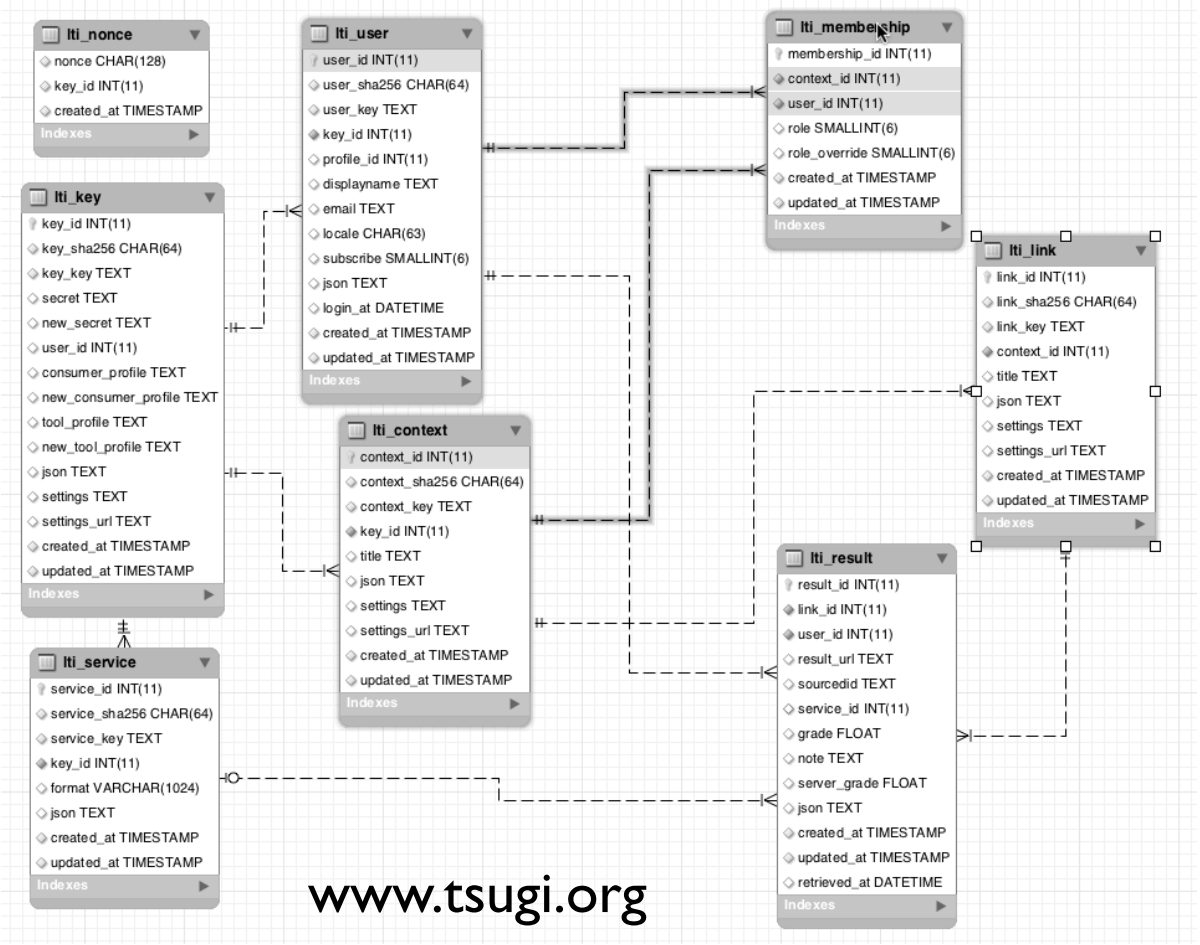
music=> DELETE FROM genre WHERE name='Metal';
DELETE 1
music=> SELECT * FROM track;
 id |   title   | len | rating | count | album_id | genre_id
----+-----------+-----+--------+-------+----------+----------
  1 | Black Dog | 297 |      5 |     0 |        2 |        1
  2 | Stairway  | 482 |      5 |     0 |        2 |        1
(2 rows)
```

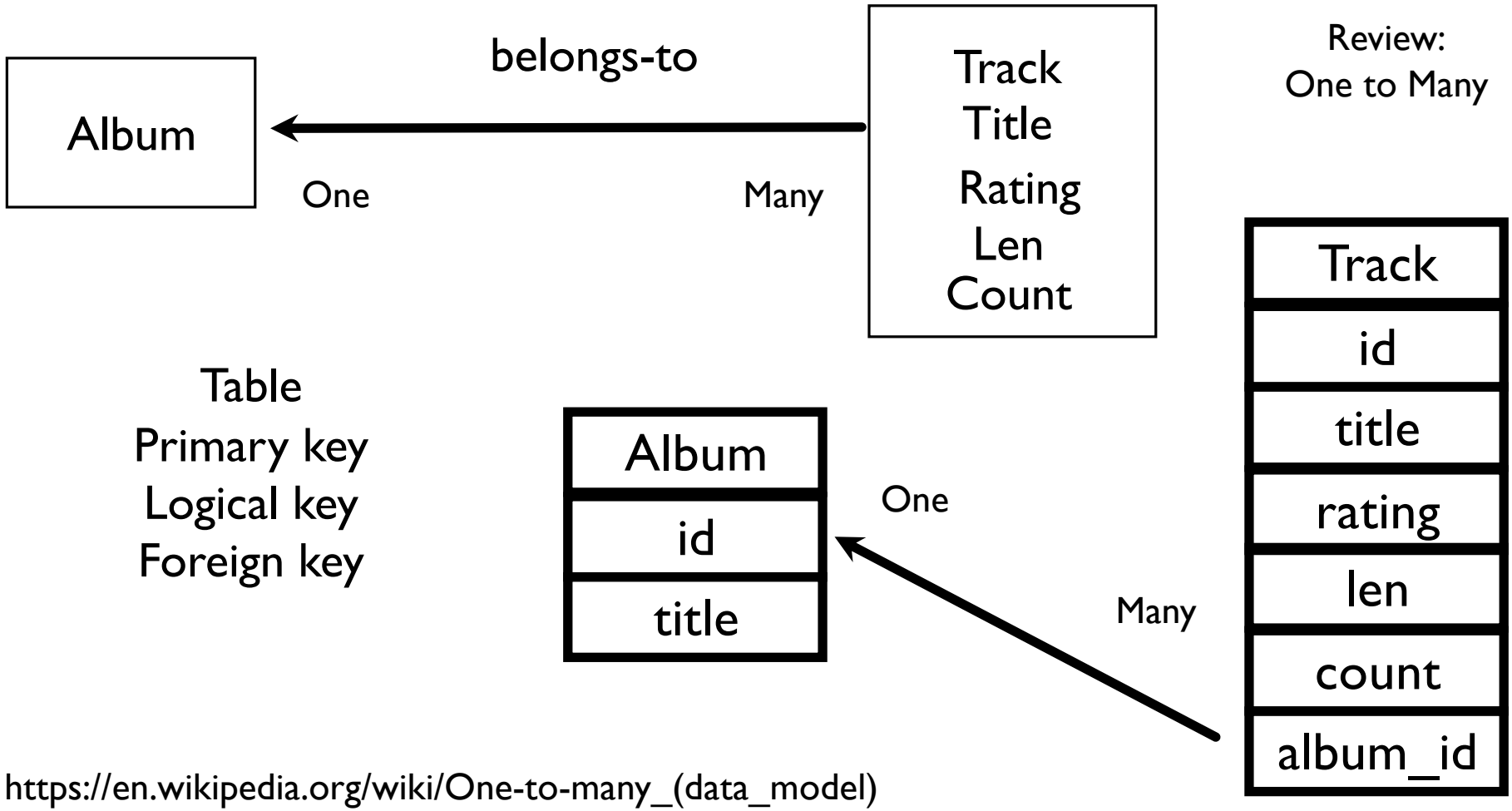
ON DELETE Choices

- Default / RESTRICT – Don't allow changes that break the constraint
- CASCADE – Adjust child rows by removing or updating to maintain consistency
- SET NULL – Set the foreign key columns in the child rows to null

http://stackoverflow.com/questions/1027656/what-is-mysqls-default-on-delete-behavior

Many-to-Many Relationships


lti_nonce
nonce CHAR(128)
key_id INT(11)
created_at TIMESTAMP
Indexes
lti_user
user_id INT(11)
user_sha256 CHAR(64)
user_key TEXT
key_id INT(11)
profile_id INT(11)
displayname TEXT
email TEXT
locale CHAR(63)
subscribe SMALLINT(6)
json TEXT
login_at DATETIME
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_membership
membership_id INT(11)
context_id INT(11)
user_id INT(11)
role SMALLINT(6)
role_override SMALLINT(6)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_key
key_id INT(11)
key_sha256 CHAR(64)
key_key TEXT
secret TEXT
new_secret TEXT
user_id INT(11)
consumer_profile TEXT
new_consumer_profile TEXT
tool_profile TEXT
new_tool_profile TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_link
link_id INT(11)
link_sha256 CHAR(64)
link_key TEXT
context_id INT(11)
title TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_context
context_id INT(11)
context_sha256 CHAR(64)
context_key TEXT
key_id INT(11)
title TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_result
result_id INT(11)
link_id INT(11)
user_id INT(11)
result_url TEXT
sourcedid TEXT
service_id INT(11)
grade FLOAT
note TEXT
server_grade FLOAT
json TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
retrieved_at DATETIME
Indexes
lti_service
service_id INT(11)
service_sha256 CHAR(64)
service_key TEXT
key_id INT(11)
format VARCHAR(1024)
json TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
www.tsugi.org



https://en.wikipedia.org/wiki/One-to-many_(data_model)

```
music=> SELECT * FROM track;
 id |     title      | len | rating | count | album_id | genre_id
----+----------------+-----+--------+-------+----------+----------
  1 | Black Dog      | 297 |      5 |     0 |        2 |        1
  2 | Stairway       | 482 |      5 |     0 |        2 |        1
  3 | About to Rock  | 313 |      5 |     0 |        1 |        2
  4 | Who Made Who   | 207 |      5 |     0 |        1 |        2
```

One

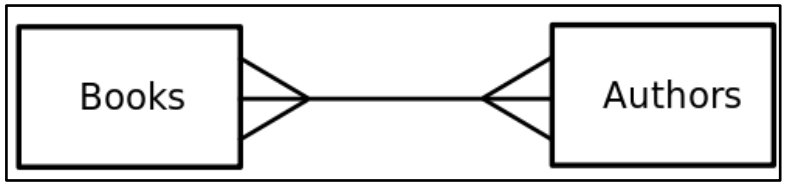
Many

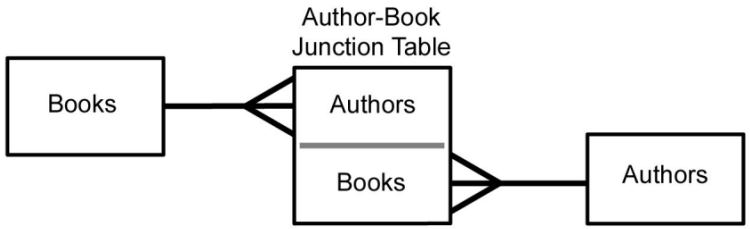
```
music=> SELECT * FROM genre;
 id | name
----+------
  1 | Rock
  2 | Metal
```

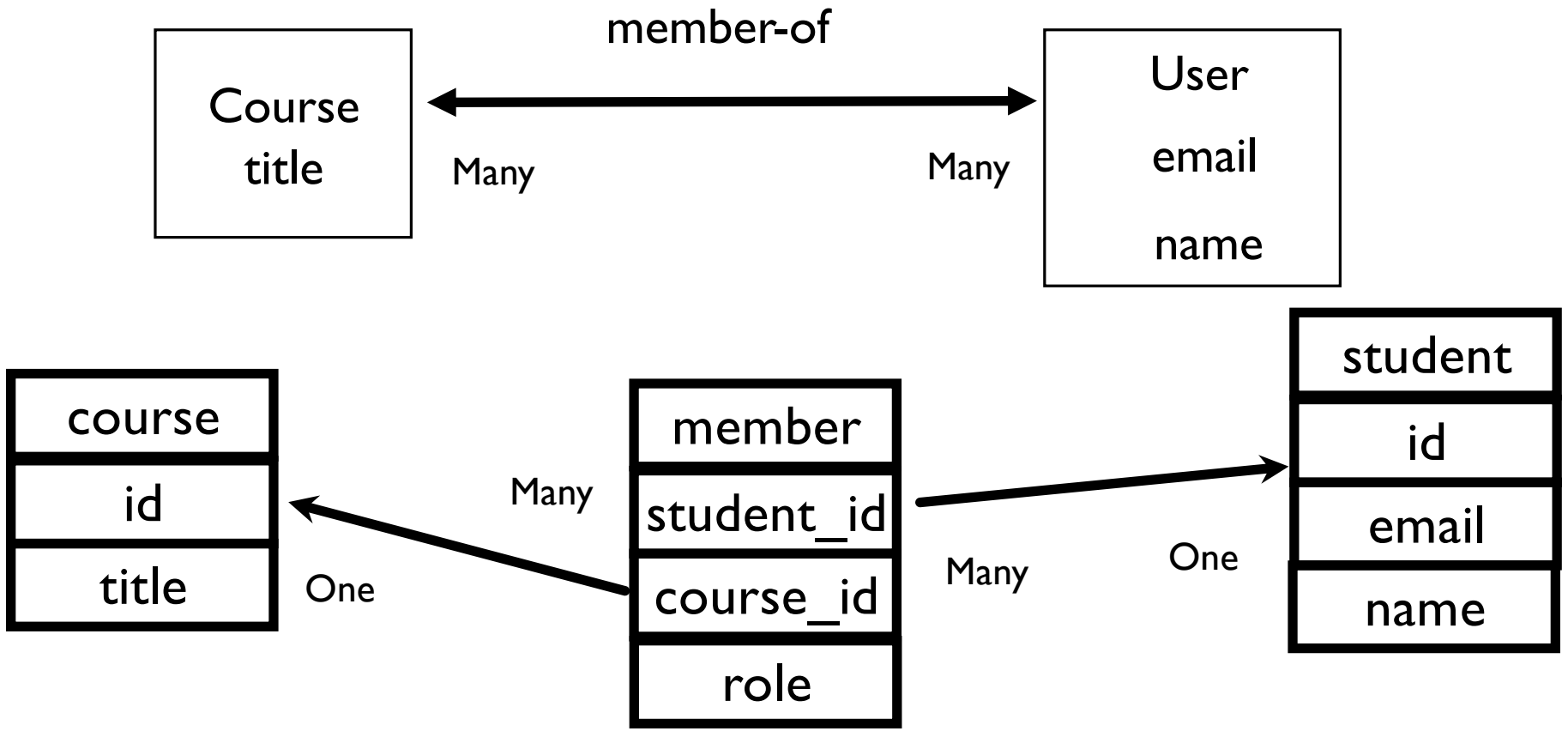
https://en.wikipedia.org/wiki/One-to-many_(data_model)

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.





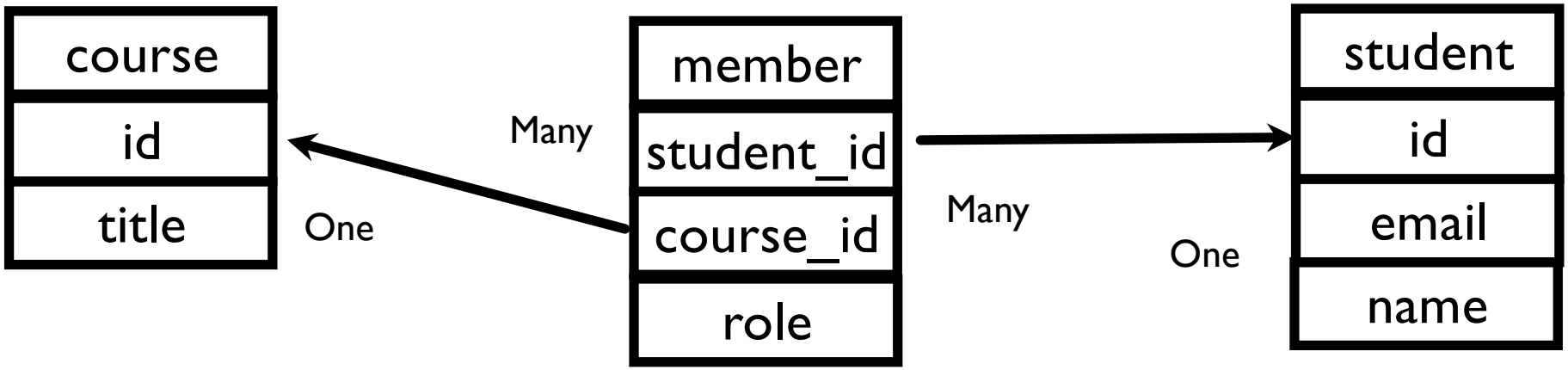


https://en.wikipedia.org/wiki/Many-to-many_(data_model)

Start with a Fresh Database

```
CREATE TABLE student (
  id SERIAL,
  name VARCHAR(128),
  email VARCHAR(128) UNIQUE,
  PRIMARY KEY(id)
);

CREATE TABLE course (
  id SERIAL,
  title VARCHAR(128) UNIQUE,
  PRIMARY KEY(id)
);
```




```
CREATE TABLE member (
    student_id INTEGER REFERENCES student(id) ON DELETE CASCADE,
    course_id  INTEGER REFERENCES course(id) ON DELETE CASCADE,
    role       INTEGER,
    PRIMARY KEY (student_id, course_id)
);
```

Insert Users and Courses

```
music=> INSERT INTO student (name, email) VALUES ('Jane', 'jane@tsugi.org');
music=> INSERT INTO student (name, email) VALUES ('Ed', 'ed@tsugi.org');
music=> INSERT INTO student (name, email) VALUES ('Sue', 'sue@tsugi.org');
music=> SELECT * FROM student;
 id | name |     email
----+------+----------------
  1 | Jane | jane@tsugi.org
  2 | Ed   | ed@tsugi.org
  3 | Sue  | sue@tsugi.org

music=> INSERT INTO course (title) VALUES ('Python');
music=> INSERT INTO course (title) VALUES ('SQL');
music=> INSERT INTO course (title) VALUES ('PHP');
music=> SELECT * FROM COURSE;
 id | title
----+--------
  1 | Python
  2 | SQL
  3 | PHP
```

Insert Memberships

```
music=> SELECT * FROM student;
 id | name |     email
----+------+---------------
  1 | Jane | jane@tsugi.org
  2 | Ed   | ed@tsugi.org
  3 | Sue  | sue@tsugi.org
```

```
music=> SELECT * FROM course;
 id | title
----+--------
  1 | Python
  2 | SQL
  3 | PHP
```

```
INSERT INTO member (student_id, course_id, role) VALUES (1, 1, 1);
INSERT INTO member (student_id, course_id, role) VALUES (2, 1, 0);
INSERT INTO member (student_id, course_id, role) VALUES (3, 1, 0);

INSERT INTO member (student_id, course_id, role) VALUES (1, 2, 0);
INSERT INTO member (student_id, course_id, role) VALUES (2, 2, 1);

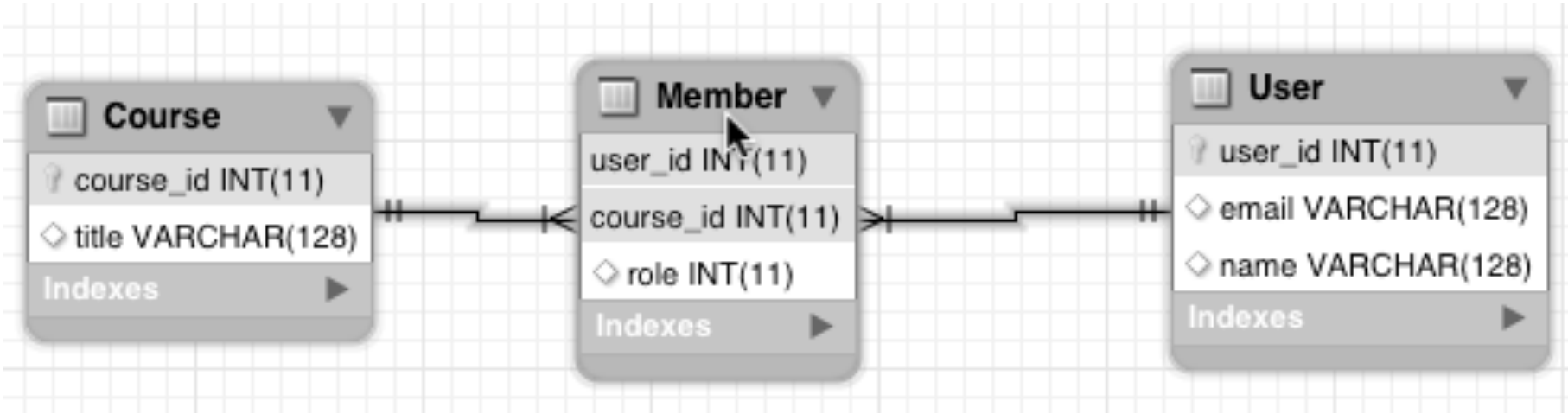
INSERT INTO member (student_id, course_id, role) VALUES (2, 3, 1);
INSERT INTO member (student_id, course_id, role) VALUES (3, 3, 0);
```

```
music=> SELECT * FROM student;
 id | name |     email
----+------+----------------
  1 | Jane | jane@tsugi.org
  2 | Ed   | ed@tsugi.org
  3 | Sue  | sue@tsugi.org
```

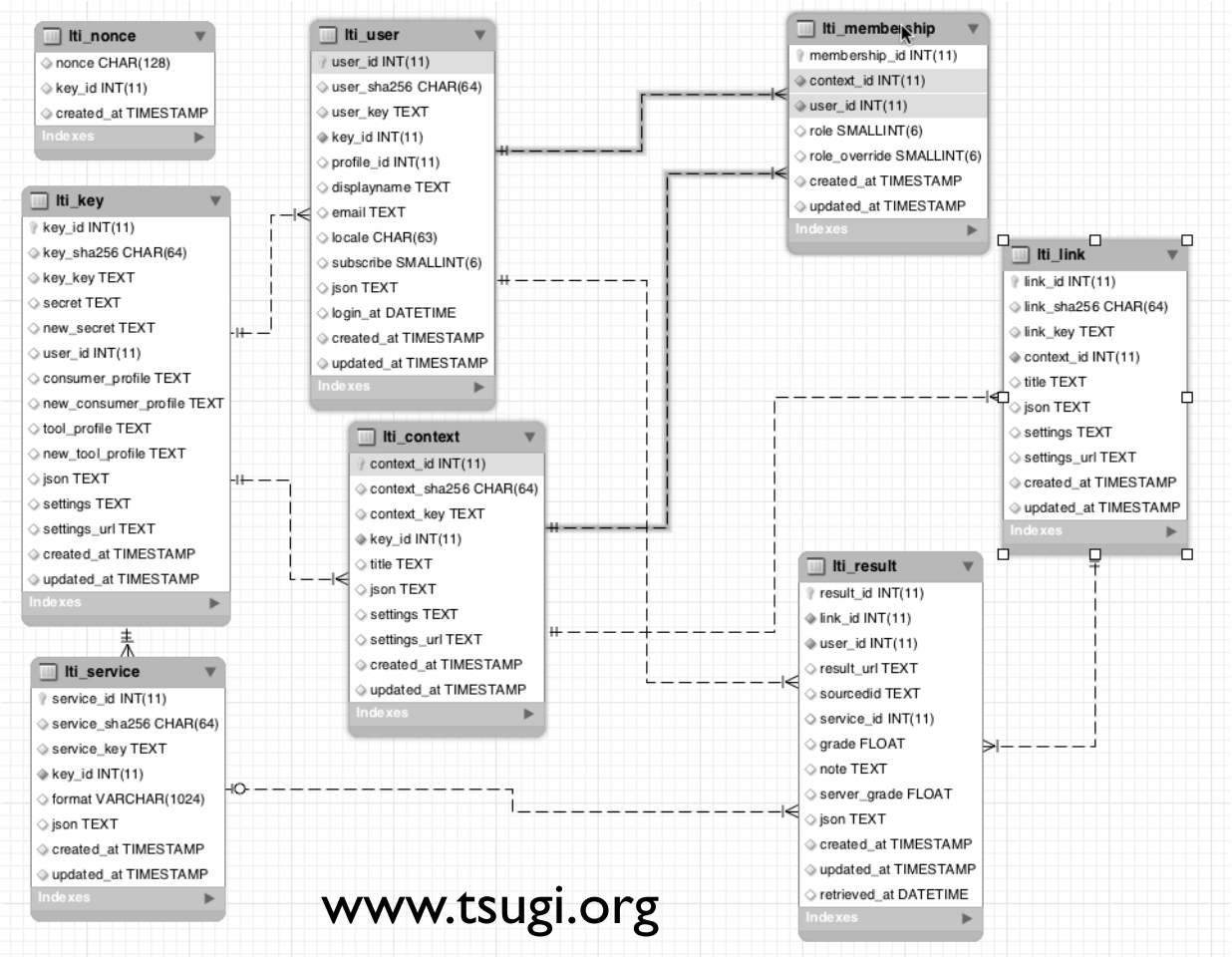
```
music=> SELECT * FROM course;
 id | title
----+--------
  1 | Python
  2 | SQL
  3 | PHP
```

```
music=> SELECT * FROM member;
 student_id | course_id | role
------------+-----------+------
          1 |         1 |    1
          2 |         1 |    0
          3 |         1 |    0
          1 |         2 |    0
          2 |         2 |    1
          2 |         3 |    1
          3 |         3 |    0
```

```
music=> SELECT student.name, member.role, course.title
music-> FROM student
music-> JOIN member ON member.student_id = student.id
music-> JOIN course ON member.course_id = course. id
music-> ORDER BY course.title, member.role DESC,
student.name;
 name | role | title
------+------+-------
 Ed   |    1 | PHP
 Sue  |    0 | PHP
 Jane |    1 | Python
 Ed   |    0 | Python
 Sue  |    0 | Python
 Ed   |    1 | SQL
 Jane |    0 | SQL
(7 rows)
```



https://www.mysql.com/products/workbench/


lti_nonce
nonce CHAR(128)
key_id INT(11)
created_at TIMESTAMP
Indexes
lti_user
user_id INT(11)
user_sha256 CHAR(64)
user_key TEXT
key_id INT(11)
profile_id INT(11)
displayname TEXT
email TEXT
locale CHAR(63)
subscribe SMALLINT(6)
json TEXT
login_at DATETIME
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_membership
membership_id INT(11)
context_id INT(11)
user_id INT(11)
role SMALLINT(6)
role_override SMALLINT(6)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_key
key_id INT(11)
key_sha256 CHAR(64)
key_key TEXT
secret TEXT
new_secret TEXT
user_id INT(11)
consumer_profile TEXT
new_consumer_profile TEXT
tool_profile TEXT
new_tool_profile TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_link
link_id INT(11)
link_sha256 CHAR(64)
link_key TEXT
context_id INT(11)
title TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_context
context_id INT(11)
context_sha256 CHAR(64)
context_key TEXT
key_id INT(11)
title TEXT
json TEXT
settings TEXT
settings_url TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
lti_result
result_id INT(11)
link_id INT(11)
user_id INT(11)
result_url TEXT
sourcedid TEXT
service_id INT(11)
grade FLOAT
note TEXT
server_grade FLOAT
json TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
retrieved_at DATETIME
Indexes
lti_service
service_id INT(11)
service_sha256 CHAR(64)
service_key TEXT
key_id INT(11)
format VARCHAR(1024)
json TEXT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
www.tsugi.org

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.

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 that must 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) as part of www.pg4e.com 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 and Translators here including names and dates

Continue new Contributors and Translators here