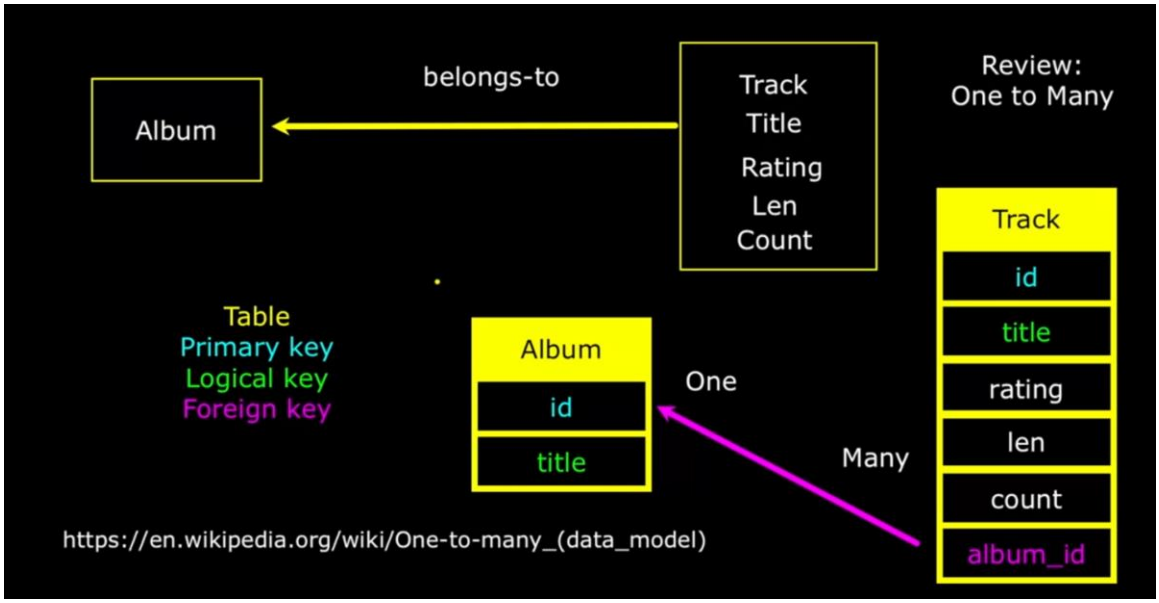
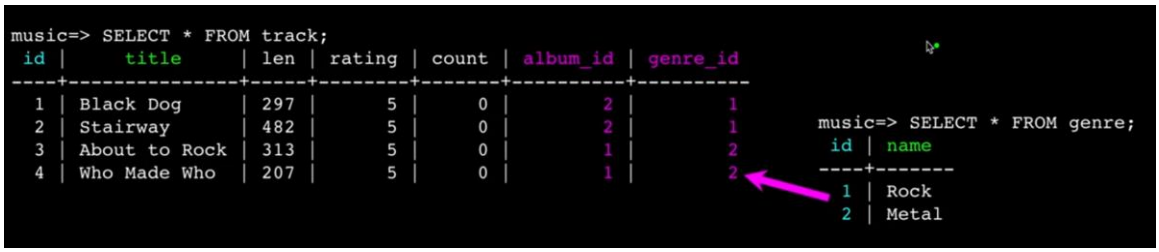
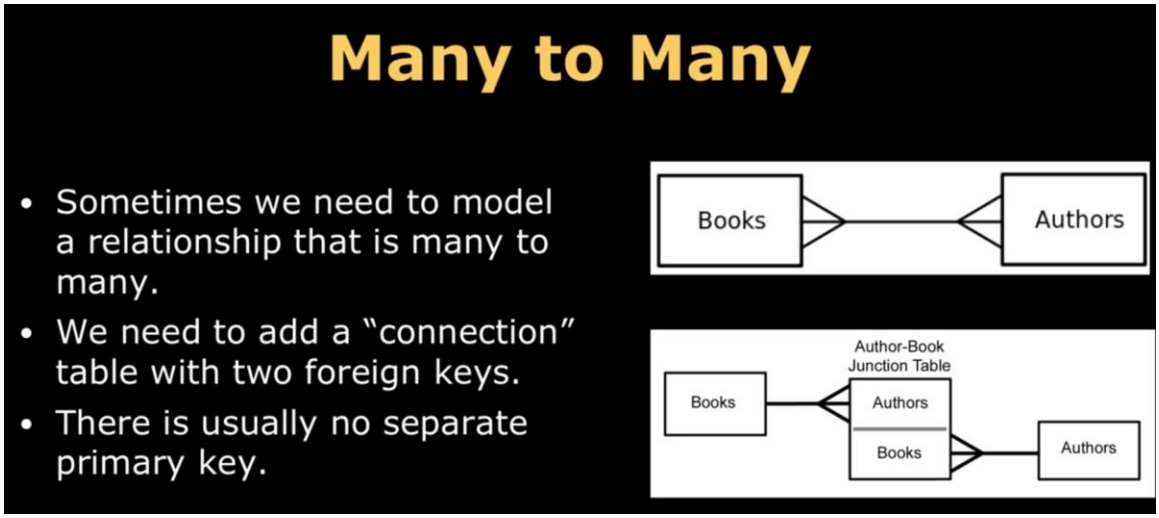
Many-to-Many Relationships

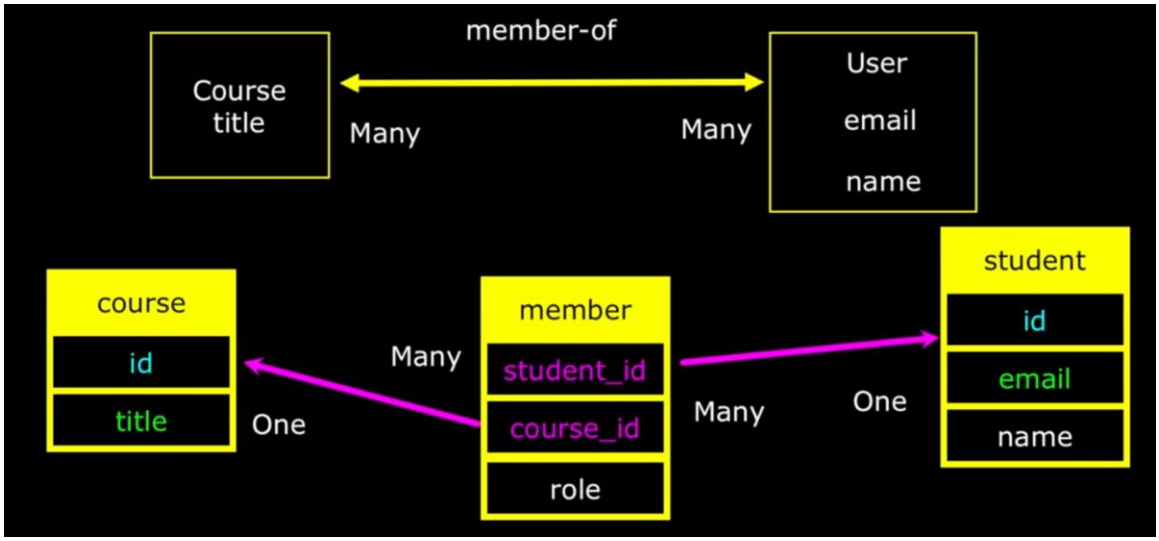




Notice how to draw the logical diagram for “Many to Many” relationship

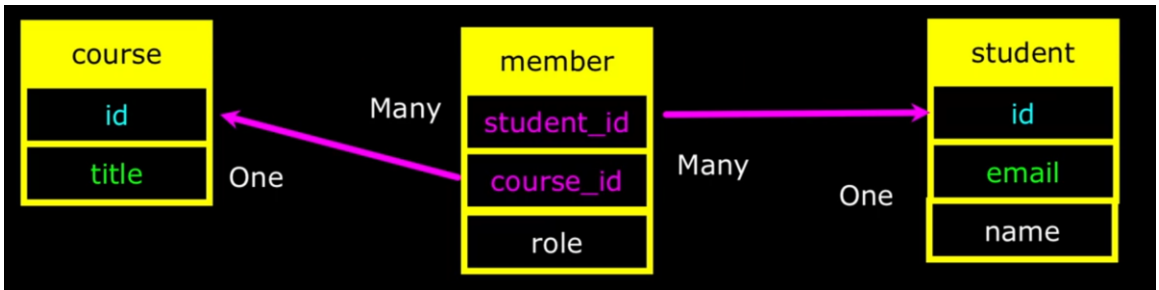


It is a good way to build a table for “many to many” relationships, primary key would be a combination of keys.



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

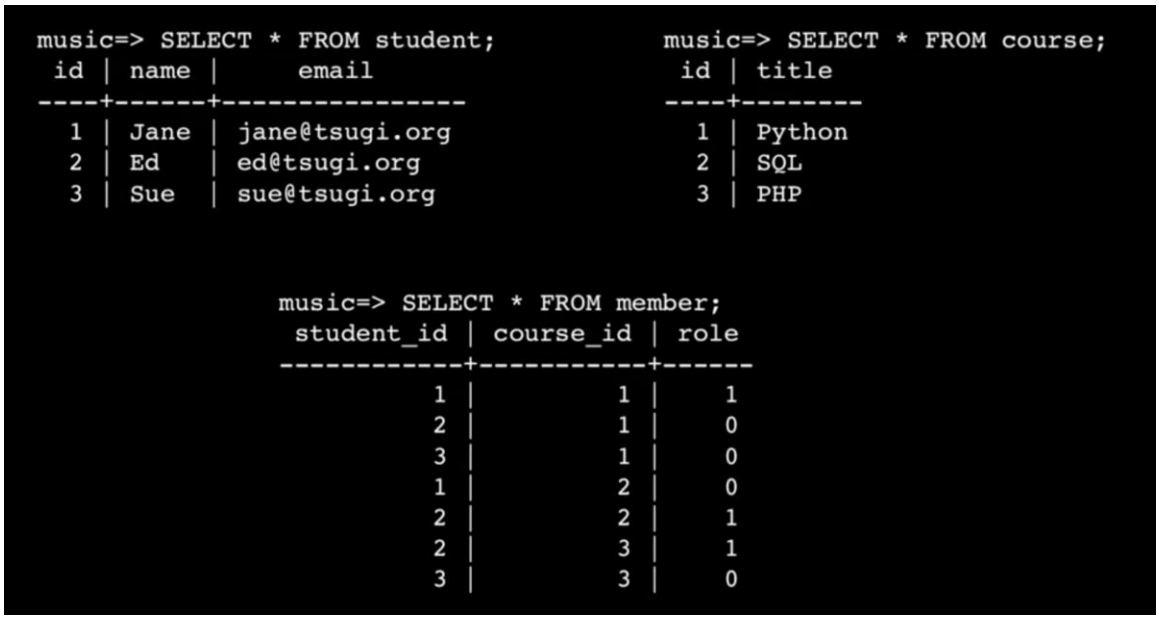
```
music=> INSERT INTO student (name, email) VALUES ('Jane', 'jane@tsugi.org');
music=> INSERT INTO student (name, email) VALUES ('Ed', 'ed@tsugi.org');
music=> INSERT INTO student (name, email) VALUES ('Sue', 'sue@tsugi.org');
music=> SELECT * FROM student;
 id | name |     email
----+------+---------------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

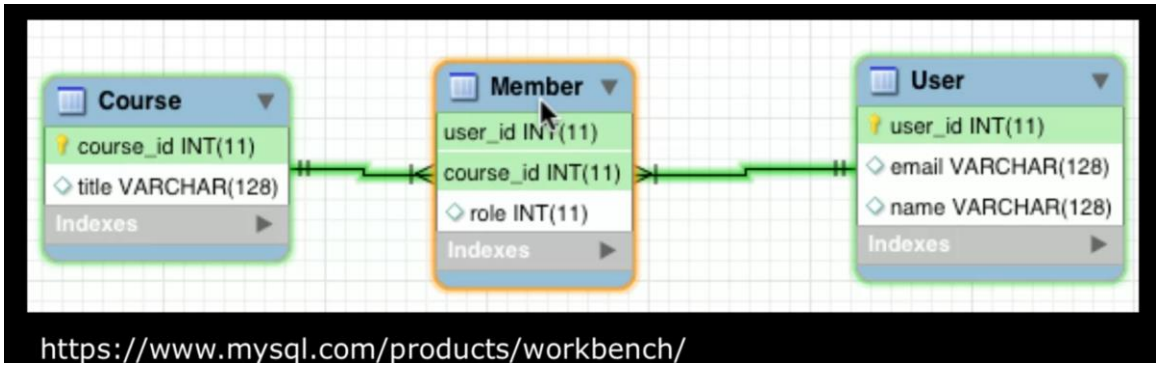
```
music=> SELECT * FROM course;
 id | title
----+--------
  1 | Python
  2 | SQL
  3 | PHP
```

```
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);
```



Play
Volume
13:15/15:12

```
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/

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.