CODE: Working With Foreign Keys

Section 12, Lecture 198

```
-- Creating the customers and orders tables

CREATE TABLE customers(
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(100)
);
CREATE TABLE orders(
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE,
    amount DECIMAL(8,2),
    customer_id INT,
    FOREIGN KEY(customer_id) REFERENCES customers(id)
);
```

```
-- Inserting some customers and orders

INSERT INTO customers (first_name, last_name, email)
VALUES ('Boy', 'George', 'george@gmail.com'),
       ('George', 'Michael', 'gm@gmail.com'),
       ('David', 'Bowie', 'david@gmail.com'),
       ('Blue', 'Steele', 'blue@gmail.com'),
       ('Bette', 'Davis', 'bette@aol.com');

INSERT INTO orders (order_date, amount, customer_id)
VALUES ('2016/02/10', 99.99, 1),
       ('2017/11/11', 35.50, 1),
       ('2014/12/12', 800.67, 2),
       ('2015/01/03', 12.50, 2),
       ('1999/04/11', 450.25, 5);
```

```
-- This INSERT fails because of our fk constraint.  No user with id: 98

INSERT INTO orders (order_date, amount, customer_id)
VALUES ('2016/06/06', 33.67, 98);
```

CODE: Cross Joins

Section 12, Lecture 200

```
-- Finding Orders Placed By George: 2 Step Process

SELECT id FROM customers WHERE last_name='George';
SELECT * FROM orders WHERE customer_id = 1;

-- Finding Orders Placed By George: Using a subquery

SELECT * FROM orders WHERE customer_id =
    (
        SELECT id FROM customers
        WHERE last_name='George'
    );

-- Cross Join Craziness

SELECT * FROM customers, orders;
```

CODE: Inner Joins

Section 12, Lecture 202

```
-- IMPLICIT INNER JOIN

SELECT * FROM customers, orders
WHERE customers.id = orders.customer_id;
```

```
-- IMPLICIT INNER JOIN

SELECT first_name, last_name, order_date, amount
FROM customers, orders
    WHERE customers.id = orders.customer_id;
```

```
-- EXPLICIT INNER JOINS

SELECT * FROM customers
JOIN orders
    ON customers.id = orders.customer_id;

SELECT first_name, last_name, order_date, amount
FROM customers
JOIN orders
    ON customers.id = orders.customer_id;

SELECT *
FROM orders
JOIN customers
    ON customers.id = orders.customer_id;
```

```
-- ARBITRARY JOIN - meaningless, but still possible

SELECT * FROM customers
JOIN orders ON customers.id = orders.id;
```

CODE: Left Joins

Section 12, Lecture 204

-- Getting Fancier (Inner Joins Still)

```
SELECT first_name, last_name, order_date, amount
FROM customers
JOIN orders
    ON customers.id = orders.customer_id
ORDER BY order_date;

SELECT
    first_name,
    last_name,
    SUM(amount) AS total_spent
FROM customers
JOIN orders
    ON customers.id = orders.customer_id
GROUP BY orders.customer_id
ORDER BY total_spent DESC;
```

-- LEFT JOINS

```
SELECT * FROM customers
LEFT JOIN orders
    ON customers.id = orders.customer_id;

SELECT first_name, last_name, order_date, amount
FROM customers
LEFT JOIN orders
    ON customers.id = orders.customer_id;

SELECT
    first_name,
    last_name,
    IFNULL(SUM(amount), 0) AS total_spent
FROM customers
LEFT JOIN orders
    ON customers.id = orders.customer_id
GROUP BY customers.id
ORDER BY total_spent;
```

CODE: Right Joins Part 1

Section 12, Lecture 206

-- OUR FIRST RIGHT JOIN (seems the same as a left join?)

```
SELECT * FROM customers
RIGHT JOIN orders
   ON customers.id = orders.customer_id;
```

-- ALTERING OUR SCHEMA to allow for a better example (optional)

```
CREATE TABLE customers(
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(100)
);
CREATE TABLE orders(
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE,
    amount DECIMAL(8,2),
    customer_id INT
);
```

-- INSERTING NEW DATA (no longer bound by foreign key constraint)

```
INSERT INTO customers (first_name, last_name, email)
VALUES ('Boy', 'George', 'george@gmail.com'),
       ('George', 'Michael', 'gm@gmail.com'),
       ('David', 'Bowie', 'david@gmail.com'),
       ('Blue', 'Steele', 'blue@gmail.com'),
       ('Bette', 'Davis', 'bette@aol.com');

INSERT INTO orders (order_date, amount, customer_id)
VALUES ('2016/02/10', 99.99, 1),
       ('2017/11/11', 35.50, 1),
       ('2014/12/12', 800.67, 2),
       ('2015/01/03', 12.50, 2),
       ('1999/04/11', 450.25, 5);

INSERT INTO orders (order_date, amount, customer_id) VALUES
('2017/11/05', 23.45, 45),
(CURDATE(), 777.77, 109);
```

CODE: Right Joins Part 2

Section 12, Lecture 208

```
--A MORE COMPLEX RIGHT JOIN



SELECT
    IFNULL(first_name,'MISSING') AS first,
    IFNULL(last_name,'USER') as last,
    order_date,
    amount,
    SUM(amount)
FROM customers
RIGHT JOIN orders
    ON customers.id = orders.customer_id
GROUP BY first_name, last_name;
```

```
-- WORKING WITH ON DELETE CASCADE

CREATE TABLE customers(
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(100)
);

CREATE TABLE orders(
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE,
    amount DECIMAL(8,2),
    customer_id INT,
    FOREIGN KEY(customer_id)
        REFERENCES customers(id)
        ON DELETE CASCADE
);


INSERT INTO customers (first_name, last_name, email)
VALUES ('Boy', 'George', 'george@gmail.com'),
       ('George', 'Michael', 'gm@gmail.com'),
       ('David', 'Bowie', 'david@gmail.com'),
       ('Blue', 'Steele', 'blue@gmail.com'),
       ('Bette', 'Davis', 'bette@aol.com');

INSERT INTO orders (order_date, amount, customer_id)
VALUES ('2016/02/10', 99.99, 1),
       ('2017/11/11', 35.50, 1),
       ('2014/12/12', 800.67, 2),
       ('2015/01/03', 12.50, 2),
       ('1999/04/11', 450.25, 5);
```

CODE: Right and Left Joins FAQ

Section 12, Lecture 210

```
SELECT * FROM customers
LEFT JOIN orders
    ON customers.id = orders.customer_id;

SELECT * FROM orders
RIGHT JOIN customers
    ON customers.id = orders.customer_id;

SELECT * FROM orders
LEFT JOIN customers
    ON customers.id = orders.customer_id;

SELECT * FROM customers
RIGHT JOIN orders
    ON customers.id = orders.customer_id;
```

CODE: Our First Joins Exercise

Section 12, Lecture 213

-- The Schema

```
CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100)
);


CREATE TABLE papers (
    title VARCHAR(100),
    grade INT,
    student_id INT,
    FOREIGN KEY (student_id)
        REFERENCES students(id)
        ON DELETE CASCADE
);
```

-- The Starter Data

```
INSERT INTO students (first_name) VALUES
('Caleb'),
('Samantha'),
('Raj'),
('Carlos'),
('Lisa');

INSERT INTO papers (student_id, title, grade ) VALUES
(1, 'My First Book Report', 60),
(1, 'My Second Book Report', 75),
(2, 'Russian Lit Through The Ages', 94),
(2, 'De Montaigne and The Art of The Essay', 98),
(4, 'Borges and Magical Realism', 89);
```

CODE: Our First Joins Exercise SOLUTION PT. 2

Section 12, Lecture 215

-- EXERCISE 1

```
SELECT first_name, title, grade
FROM students
INNER JOIN papers
    ON students.id = papers.student_id
ORDER BY grade DESC;
```

-- ALT SOLUTION

```
SELECT first_name, title, grade
FROM students
RIGHT JOIN papers
    ON students.id = papers.student_id
ORDER BY grade DESC;
```

-- PROBLEM 2

```
SELECT first_name, title, grade
FROM students
LEFT JOIN papers
    ON students.id = papers.student_id;
```

-- PROBLEM 3

```
SELECT
    first_name,
    IFNULL(title, 'MISSING'),
    IFNULL(grade, 0)
FROM students
LEFT JOIN papers
    ON students.id = papers.student_id;
```

-- PROBLEM 4

```
SELECT
    first_name,
    IFNULL(AVG(grade), 0) AS average
FROM students
LEFT JOIN papers
    ON students.id = papers.student_id
GROUP BY students.id
ORDER BY average DESC;
```

-- PROBLEM 5

```
SELECT first_name,
       Ifnull(Avg(grade), 0) AS average,
       CASE
         WHEN Avg(grade) IS NULL THEN 'FAILING'
         WHEN Avg(grade) >= 75 THEN 'PASSING'
         ELSE 'FAILING'
       end                   AS passing_status
FROM   students
       LEFT JOIN papers
              ON students.id = papers.student_id
GROUP  BY students.id
ORDER  BY average DESC;
```