Overview

The purpose of this class is to introduce the concepts of database normalization through the use of SQL joins. There is currently no direct assignment related to this lecture as the primary goal is to expose students to the process of creating relation based tables, understanding proper database design, and updating or fixing duplicate and error prone data through the use of SQL based joins.

Class 09 Lecture: Joins & Relations

- Topic 1 - Normalization & Joins
    - Database Normalization
        - Remove duplicate data.
        - Minimize database redesign.
        - Minimize modification anomalies.
    - Joins
        - Understanding Primary and Foreign Keys
            - Discuss how these keys are related and can be used for database normalization
        - Inner Join
            - Discuss syntax
            - Discuss concepts
        - High Level Overview of Other Join types
    - Web SQL
        - How can we populate tables efficiently?
            - Use .map on articles array to create SQLObjects to pass to webDB.execute
        - How to include foreign keys
            - Use subqueries as appropriate to get foreign keys
- Demo - How
    - How to normalize a database.
    - How to use foreign keys.
    - How to use subqueries.
    - How to perform an inner join.

Demo

Normalization

Breaking the article table into two.

First table is authors.

```
CREATE TABLE authors(
    id INTEGER PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL,
    url VARCHAR(255)
);
```

Insert Data into authors

```
INSERT INTO authors (name, url) VALUES ("Kevin Bacon", "https://bacon.com");
INSERT INTO authors (name, url) VALUES ("Meow Meow", "https://meow.com");
INSERT INTO authors (name, url) VALUES ("Zatarans", "https://cajun.com");
```

Second table is articles

```
CREATE TABLE articles(
    id INTEGER PRIMARY KEY,
    title VARCHAR(50) NOT NULL,
    authorId INTEGER NOT NULL REFERENCES authors(id),
    markdown TEXT NOT NULL,
    publishedOn DATETIME
);
```

Insert Data into articles

```
INSERT INTO articles (title, authorId, markdown, publishedOn) VALUES ("Bacon Ipsum", 1, "#markdown", "2015-06-12");
INSERT INTO articles (title, authorId, markdown, publishedOn) VALUES ("Six Degree", 1, "#markdown", "2016-06-10");
INSERT INTO articles (title, authorId, markdown, publishedOn) VALUES ("Cat Ipsum", 2, "#markdown", "2015-08-12");
INSERT INTO articles (title, authorId, markdown, publishedOn) VALUES ("Cajun Ipsum", 3, "#markdown", "2014-01-13");
INSERT INTO articles (title, authorId, markdown, publishedOn) VALUES ("If it fits", 3, "#markdown", "2015-06-19");
INSERT INTO articles (title, authorId, markdown, publishedOn) VALUES ("Why Grumpy?", 2, "#markdown", "2012-04-18");
```

Inner Join

```
SELECT
    articles.id,
    title,
    authors.name AS author,
    authors.url AS authorUrl,
    markdown,
    publishedOn
FROM articles
INNER JOIN authors
    ON articles.authorId = authors.id
ORDER BY publishedOn DESC;
```

Outer Join

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

- Left Outer Join
- Right Outer Join
- Full Outer Join

Left Outer Join

The left outer join returns a result table with the matched data of two tables then remaining rows of the left table and null for the right table's column.

```
SELECT *
FROM articles
LEFT OUTER JOIN authors
    ON articles.authorId = authors.id;
```

Right Outer Join

The right outer join returns a result table with the matched data of two tables then remaining rows of the right table and null for the left table's columns.

CANT DEMO

```
SELECT column-name-list
FROM table-name1
RIGHT OUTER JOIN table-name2
    ON table-name1.column-name = table-name2.column-name;
```

FULL OUTER JOIN

The full outer join returns a result table with the matched data of two table then remaining rows of both left table and then the right table.

```
SELECT column-name-list
FROM table-name1
FULL OUTER JOIN  table-name2
    ON table-name1.column-name = table-name2.column-name;
```

Multiple Data Table

Mulitple Tables: * courses * grades * students * teachers

DROP

```
DROP TABLE IF EXISTS courses;
DROP TABLE IF EXISTS grades;
DROP TABLE IF EXISTS students;
DROP TABLE IF EXISTS teachers;
```

CREATE

```
CREATE TABLE courses (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(32) DEFAULT NULL,
    teacher_id INT NOT NULL
);

CREATE TABLE grades (
    student_id INT NOT NULL,
    course_id INT NOT NULL,
    grade varchar(2) DEFAULT NULL
);

CREATE TABLE students (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(32) DEFAULT NULL,
    email VARCHAR(32) DEFAULT NULL,
    password VARCHAR(16) DEFAULT NULL
);

CREATE TABLE teachers (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(32) DEFAULT NULL
);
```

INSERT

```
INSERT INTO courses VALUES
    (10001, 'Computer Science 142', 1234),
    (10002, 'Computer Science 143', 5678),
    (10003, 'Computer Science 190M', 9012),
    (10004, 'Informatics 100', 1234);

INSERT INTO grades VALUES
    (123, 10001, 'B-'),
    (123, 10002, 'C'),
    (456, 10001, 'B+'),
    (888, 10002, 'A+'),
    (888, 10003, 'A+'),
    (404, 10004, 'D+'),
    (404, 10002, 'B'),
    (456, 10002, 'D-');

INSERT INTO students VALUES
    (123, 'Bart', 'bart@fox.com', 'bartman'),
    (404, 'Ralph', 'ralph@fox.com', 'catfood'),
    (456, 'Milhouse', 'milhouse@fox.com', 'fallout'),
    (888, 'Lisa', 'lisa@fox.com', 'vegan');

INSERT INTO teachers VALUES
    (1234, 'Krabappel'),
    (5678, 'Hoover'),
    (9012, 'Stepp');
```

Getting Values

```
-- SELECT students and grades, no explicit join
SELECT *
FROM students, grades
WHERE students.id = grades.student_id

-- SELECT students and grades, join
```

```
SELECT *
FROM students
JOIN grades
    ON grades.student_id = students.id

-- SELECT students and grades, inner join
SELECT *
FROM students
INNER JOIN grades
    ON grades.student_id = students.id

-- SELECT students and grades, inner join using alias
SELECT *
FROM students AS s
INNER JOIN grades AS g
    ON g.student_id = s.id

-- SELECT specifc columns from students and grades, inner join using alias
SELECT s.id
    , s.name
    , s.email
    , g.course_id
    , g.grade
FROM students AS s
INNER JOIN grades AS g
    ON g.student_id = s.id

-- Select from students, grades and courses
SELECT s.id
    , s.name
    , s.email
    , g.course_id
    , g.grade
FROM students AS s
INNER JOIN grades AS g
    ON g.student_id = s.id
INNER JOIN courses AS c
    ON c.id = g.course_id

-- Select from students, grades and courses, labelling all columns explicitly
SELECT s.id AS student_id
    , s.name AS student_name
    , s.email AS student_email
    , g.course_id AS course_id
    , g.grade AS course_grade
    , c.name AS course_name
FROM students AS s
INNER JOIN grades AS g
    ON g.student_id = s.id
INNER JOIN courses AS c
    ON c.id = g.course_id

-- Select from students, grades, courses and teachers, labelling all columns explicitly
SELECT s.id AS student_id
    , s.name AS student_name
    , s.email AS student_email
    , g.course_id AS course_id
    , g.grade AS course_grade
    , c.name AS course_name
    , t.name AS teacher_name
FROM students AS s
INNER JOIN grades AS g
    ON g.student_id = s.id
INNER JOIN courses AS c
    ON c.id = g.course_id
INNER JOIN teachers AS t
    ON t.id = c.teacher_id


INSERT INTO students VALUES (999, 'Nelson', 'nelson@fox.com', 'newton');


SELECT s.id AS student_id
    , s.name AS student_name
    , s.email AS student_email
    , g.course_id AS course_id
    , g.grade AS course_grade
    , c.name AS course_name
    , t.name AS teacher_name
FROM students AS s
LEFT OUTER JOIN grades AS g
    ON g.student_id = s.id
LEFT OUTER JOIN courses AS c
    ON c.id = g.course_id
LEFT OUTER JOIN teachers AS t
    ON t.id = c.teacher_id
```

The SQL subquery syntax

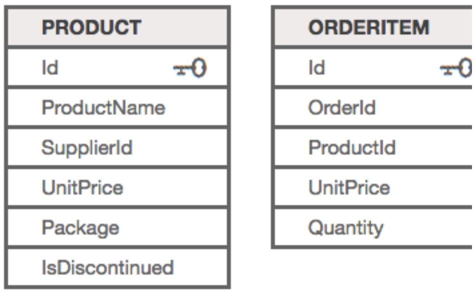
There is no general syntax; subqueries are regular queries placed inside parenthesis. Subqueries can be used in different ways and at different locations inside a query: Here is an subquery with the IN operator

```
SELECT column-names
FROM table-name1
WHERE value IN (SELECT column-name
               FROM table-name2
              WHERE condition)
```

Subqueries can also assign column values for each record:

```
SELECT  column1 = (SELECT column-name FROM table-name WHERE condition),
        column-names
FROM table-name
WHERE condition
```

SQL Subquery Examples



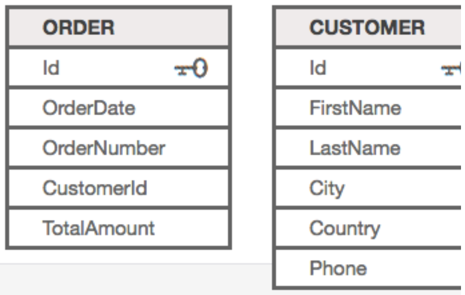
Problem: List products with order quantities greater than 100.

```sql
SELECT ProductName
  FROM Product
 WHERE Id IN (SELECT ProductId
                FROM OrderItem
               WHERE Quantity > 100)
```

Results: 12 records

PoductName
Guaraná Fantástica
Schoggi Schokolade
Chartreuse verte
Jack's New England Clam Chowder
Rogede sild
Manjimup Dried Apples
Perth Pasties
...

SQL Subquery Examples



Problem: List all customers with their total number of orders

```sql
SELECT FirstName, LastName,
       OrderCount = (SELECT COUNT(O.Id) FROM [Order] O WHERE O.CustomerId = C.Id)
  FROM Customer C
```

This is a **correlated subquery** because the subquery references the enclosing query (i.e. the C.Id in the WHERE clause).

Results: 91 records

FirstName	LastName	OrderCount
Maria	Anders	6
Ana	Trujillo	4
Antonio	Moreno	7
Thomas	Hardy	13
Christina	Berglund	18
Hanna	Moos	7
Frédérique	Citeaux	11
Martín	Sommer	3

...

Quick refactor functionality

Before

```javascript
Article.loadAll = function(rawData) {
    console.log(typeof rawData);
    rawData.sort(function(a,b) {
        return (new Date(b.publishedOn)) - (new Date(a.publishedOn));
    });

    rawData.forEach(function(ele) {
        Article.all.push(new Article(ele));
    });
}
```

After

```javascript
// DONE: Refactor to expect the raw data from the database, rather than localStorage.
Article.loadAll = function(rows) {
    Article.all = rows.map(function(ele) {
        return new Article(ele);
    });
};
```