MSiA-413 Introduction to Databases and Information Retrieval

Lecture 11
NATURAL and LEFT JOINs

Instructor: Nikos Hardavellas

Slides adapted from Steve Tarzia, C. L. Moffatt

Last Lecture

Showed how to create and modify data in SQL databases

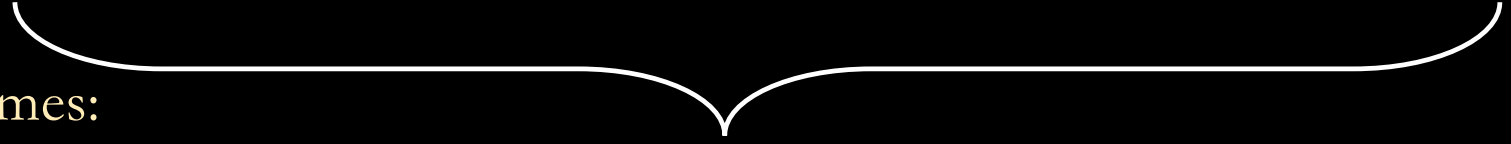
- `CREATE TABLE` defines columns, primary key, foreign keys
- `INSERT INTO` adds rows
  - Bulk loading
- `DELETE FROM` removes rows
- `UPDATE` changes column values for existing rows
- `ALTER TABLE` changes schema
- `CREATE INDEX` adds and index

NATURAL JOIN

- A shorthand notation to make some JOIN queries shorter to express
- NATURAL JOIN matches rows using **columns with identical names**

For example:

```
SELECT * FROM Orders JOIN Order_Details
  ON Orders.OrderNumber = Order_Details.OrderNumber;
```

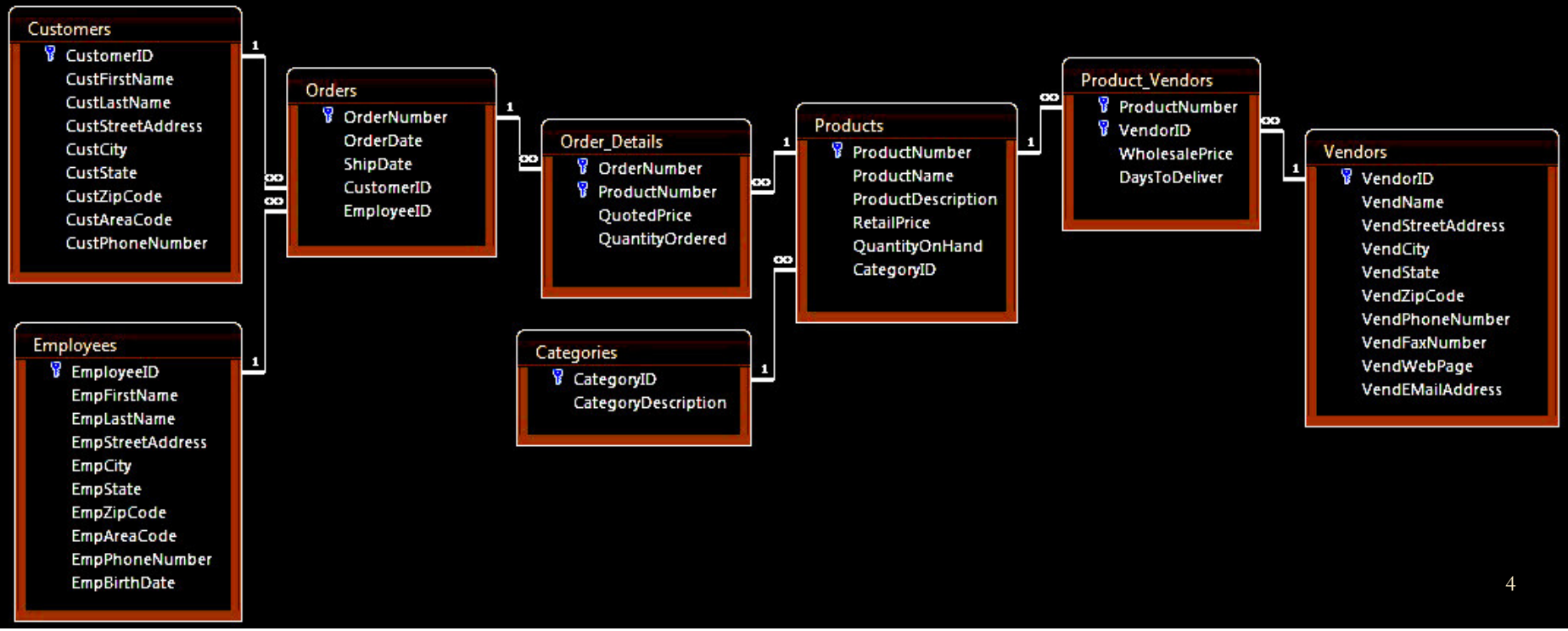


Becomes:

```
SELECT * FROM Orders NATURAL JOIN Order_Details;
```

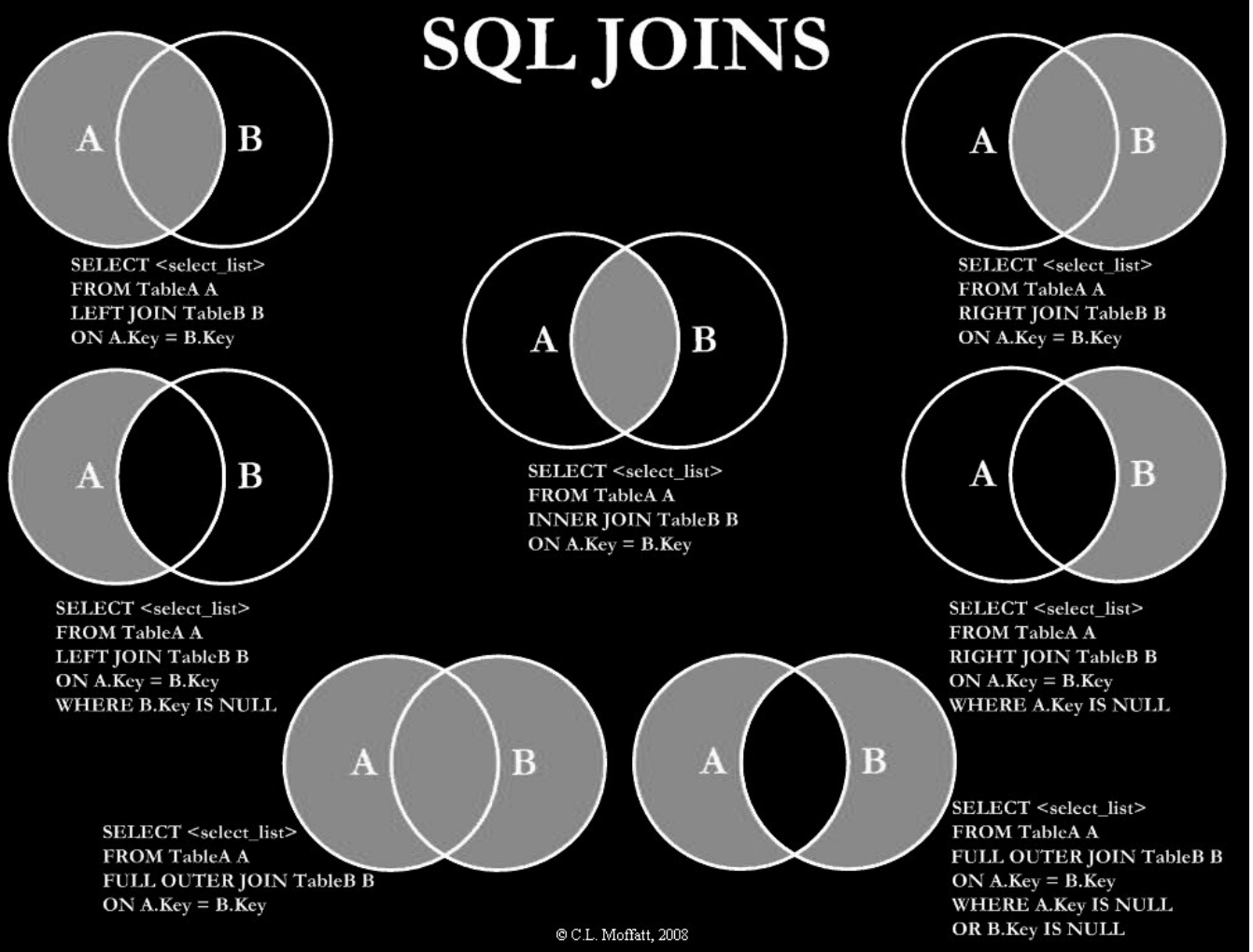
Designing your data model NATURAL-ly

- Consistent column naming allows you to use NATURAL JOINs
- This is a good reason to avoid generic column names like "id" or "name"
  - Be explicit; use "CustomerID" and "EmployeeID", etc, instead of just "id"



Different JOINs

- `INNER JOIN` constructs a table of all pairs of matching rows from two tables
  - `INNER` is the default
  - Useful for foreign keys
- However, there are many other ways to JOIN tables if you don't require matching



https://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

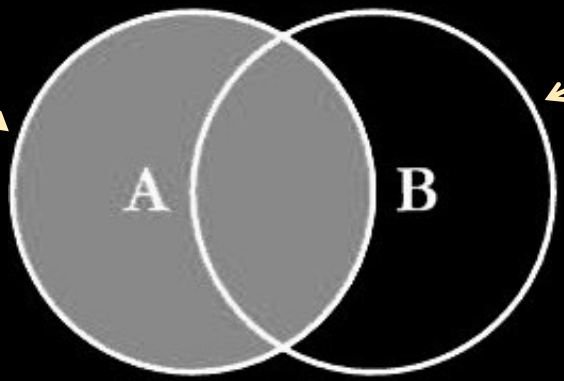
LEFT JOIN

- LEFT JOIN includes *all* rows in the first table (*left*-hand side) and just the matching rows in the second table (right-hand side)

LEFT JOIN

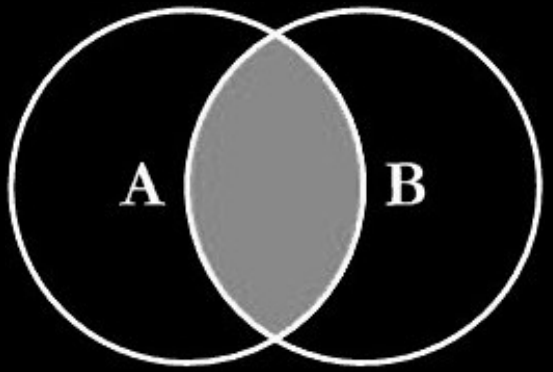
All rows from First table

Matching rows from Second table




```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

(standard) INNER JOIN




```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

LEFT JOIN output

- Like all JOINs, LEFT JOIN prints columns from the left table followed by columns from the right table
- However, with LEFT JOIN, some rows will have all *NULL* values in the right table columns, meaning that no match was found in the right table
- When to use LEFT JOIN?
  - To supplement a table with additional information that may be available for some rows, but not available for all the rows

LEFT JOIN example

staff			
id	***name***	***room***	***departmentId***
11	Bob	100	1
20	Betsy	100	*NULL*
21	Fran	101	1
22	Frank	102	99999
35	Sarah	200	5
40	Sam	10	7
54	Pat	102	2

department		
id	***name***	***buildingId***
1	Industrial Eng.	1
2	Computer Sci.	2
5	Physics	4
7	Materials Sci.	5

- Betsy and Frank have NULLs in the right half of the output because no matching department was found
- In other words, no pair of rows was found to satisfy the condition ON staff.departmentId = department.id

```
SELECT * FROM staff LEFT JOIN department ON staff.departmentId=department.id;
```

staff.***id***	staff.***name***	***staff.room***	staff.***departmentId***	department.***id***	department.***name***	department.***buildingId***
11	Bob	100	1	1	Industrial Eng.	1
20	Betsy	100	*NULL*	***NULL***	***NULL***	***NULL***
21	Fran	101	1	1	Industrial Eng.	1
22	Frank	102	99999	***NULL***	***NULL***	***NULL***
35	Sarah	200	5	5	Physics	4
40	Sam	10	7	7	Materials Sci.	5
54	Pat	102	2	2	Computer Sci.	2

LEFT JOIN example

staff			
id	***name***	***room***	***departmentId***
11	Bob	100	1
20	Betsy	100	*NULL*
21	Fran	101	1
22	Frank	102	99999
35	Sarah	200	5
40	Sam	10	7
54	Pat	102	2

department		
id	***name***	***buildingId***
1	Industrial Eng.	1
2	Computer Sci.	2
4	Physics	4
7	Materials Sci.	5

- Betsy and Frank have NULLs in the right half of the output because no matching department was found
- In other words, no pair of rows was found to satisfy the condition ON staff.departmentId = department.id

```
SELECT * FROM staff LEFT JOIN department ON staff.departmentId=department.id;
```

staff.***id***	staff.***name***	***staff.room***	staff.***departmentId***	department.***id***	department.***name***	department.***buildingId***
11	Bob	100	1	1	Industrial Eng.	1
20	Betsy	100	*NULL*	***NULL***	***NULL***	***NULL***
21	Fran	101	1	1	Industrial Eng.	1
22	Frank	102	99999	***NULL***	***NULL***	***NULL***
35	Sarah	200	5	5	Physics	4
40	Sam	10	7	7	Materials Sci.	5
54	Pat	102	2	2	Computer Sci.	2

LEFT JOIN example

staff			
id	***name***	***room***	***departmentId***
11	Bob	100	1
20	Betsy	100	*NULL*
21	Fran	101	1
22	Frank	102	99999
35	Sarah	200	5
40	Sam	10	7
54	Pat	102	2

department		
id	***name***	***buildingId***
1	Industrial Eng.	1
2	Computer Sci.	2
5	Physics	4
7	Materials Sci.	5

- Betsy and Frank have NULLs in the right half of the output because no matching department was found
- In other words, no pair of rows was found to satisfy the condition ON staff.departmentId = department.id

```
SELECT * FROM staff LEFT JOIN department ON staff.departmentId=department.id;
```

staff.***id***	staff.***name***	***staff.room***	staff.***departmentId***	department.***id***	department.***name***	department.***buildingId***
11	Bob	100	1	1	Industrial Eng.	1
20	Betsy	100	*NULL*	***NULL***	***NULL***	***NULL***
21	Fran	101	1	1	Industrial Eng.	1
22	Frank	102	99999	***NULL***	***NULL***	***NULL***
35	Sarah	200	5	5	Physics	4
40	Sam	10	7	7	Materials Sci.	5
54	Pat	102	2	2	Computer Sci.	2

LEFT JOIN example

staff			
id	***name***	***room***	***departmentId***
11	Bob	100	1
20	Betsy	100	*NULL*
21	Fran	101	1
22	Frank	102	99999
35	Sarah	200	5
40	Sam	10	7
54	Pat	102	2

department		
id	***name***	***buildingId***
1	Industrial Eng.	1
2	Computer Sci.	2
4	Physics	4
7	Materials Sci.	5

- Betsy and Frank have NULLs in the right half of the output because no matching department was found
- In other words, no pair of rows was found to satisfy the condition ON staff.departmentId = department.id

```
SELECT * FROM staff LEFT JOIN department ON staff.departmentId=department.id;
```

staff.***id***	staff.***name***	***staff.room***	staff.***departmentId***	department.***id***	department.***name***	department.***buildingId***
11	Bob	100	1	1	Industrial Eng.	1
20	Betsy	100	*NULL*	***NULL***	***NULL***	***NULL***
21	Fran	101	1	1	Industrial Eng.	1
22	Frank	102	99999	***NULL***	***NULL***	***NULL***
35	Sarah	200	5	5	Physics	4
40	Sam	10	7	7	Materials Sci.	5
54	Pat	102	2	2	Computer Sci.	2

LEFT JOIN with Grouping

- When computing an *aggregation* on a *many-to-one* relationship, LEFT JOIN includes rows from the parent table with no children
- In `SchoolScheduling.sqlite`, count the classes taught by each faculty member:
  - If you want this report to include faculty members teaching zero classes, you must use LEFT JOIN:

```
SELECT StaffID, SUM(ClassID IS NOT NULL) AS num_classes
FROM Faculty NATURAL LEFT JOIN Faculty_Classes
GROUP BY StaffID ORDER BY num_classes;
```

  - The conditional "`ClassID IS NOT NULL`" returns "`1`" when true, "`0`" when false. The sum of ones and zeros per faculty counts the number of classes that faculty teaches
  - Note that "`COUNT(*) AS num_classes`" would return "`1`" for faculty members with no classes, because the result table still includes every unmatched row from the left table

Aside: summing an indicator variable

- Two ways to count recipes with "fish" in description:
  - ```
    SELECT COUNT(*)
    FROM Recipes
    WHERE RecipeTitle LIKE "%fish%";
    ```
    - `WHERE` clause keeps just the rows matching "fish," then these rows are counted
  - ```
    SELECT SUM(RecipeTitle LIKE "%fish%")
    FROM Recipes;
    ```
    - A column is created for every recipe indicating whether it matches "fish" or not
    - The column's value will be `1` if it matches and `0` if not
    - Sum of all the ones and zeros will be the count of recipes with "fish" in description

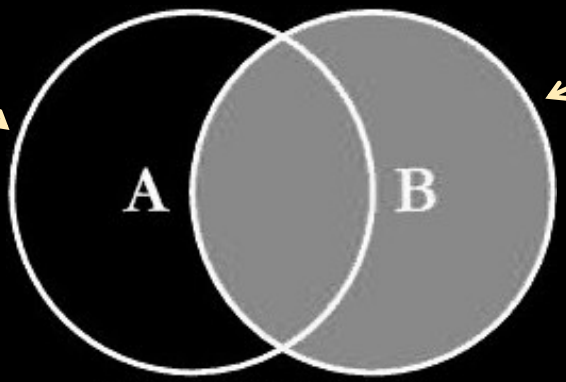
RIGHT JOIN is symmetrical to LEFT

- Includes all rows from right table and matching rows from left table
- Reordering the tables makes a RIGHT JOIN a LEFT JOIN, so it is not necessary to use the RIGHT JOIN syntax

RIGHT JOIN

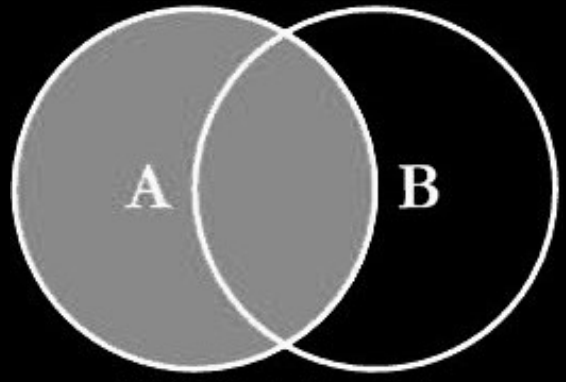
Matching rows from 1st table

All rows from 2nd table



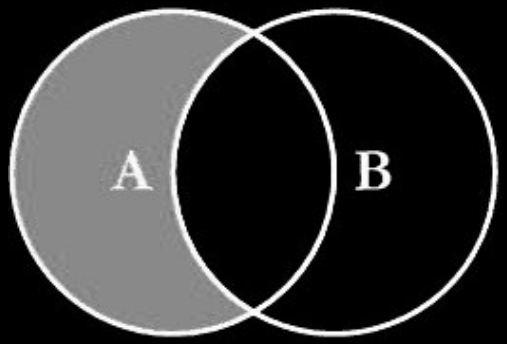

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

LEFT JOIN




```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

LEFT JOIN with exclusion




```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

- Includes rows from a table that *must not* match another table
- Useful for finding rows lacking something
- Just add a `WHERE` clause to look for *NULL* values in the right-hand side of the joined table
- For example, to determine which faculty members should be assigned a class:
  - ```
    SELECT *
    FROM Faculty NATURAL LEFT JOIN Faculty_Classes
    WHERE ClassID IS NULL;
    ```
- Which classrooms are unused?
  - ```
    SELECT *
    FROM Class_Rooms NATURAL LEFT JOIN Classes
    WHERE ClassID IS NULL;
    ```

Support Restrictions for JOINS in DB Engines

- FULL OUTER JOINs and RIGHT JOINs are not available in SQLite or MySQL
- You can *emulate* `FULL OUTER JOIN` with `UNION` (or `UNION ... EXCEPT`)

Note:

- `FULL JOIN`
- `OUTER JOIN`
- `FULL OUTER JOIN`

... all refer to the same thing

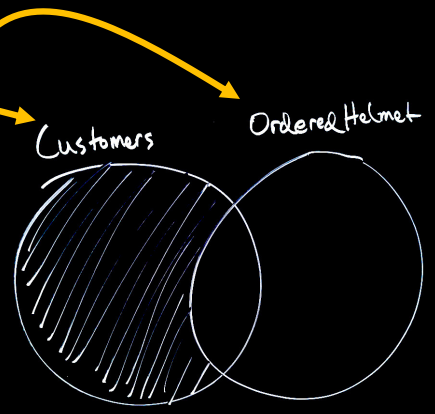
SalesOrders.sqlite: Which customers never ordered a helmet?

Solution 1 (using EXCEPT):

```
SELECT CustomerID FROM Customers
EXCEPT SELECT CustomerID FROM Customers
           NATURAL JOIN Orders
           NATURAL JOIN Order_Details
           NATURAL JOIN Products
       WHERE ProductName LIKE "%Helmet%";
```

Solution 2 (using LEFT JOIN with exclusion):

```
SELECT CustomerID
  FROM Customers
  LEFT JOIN (SELECT DISTINCT CustomerID
                        AS helmet_customer
             FROM Orders
                NATURAL JOIN Order_Details
                NATURAL JOIN Products
             WHERE ProductName LIKE "%Helmet%")
    ON CustomerID = helmet_customer
WHERE helmet_customer IS NULL;
```



Recipes.sqlite: "Display missing types of recipes"

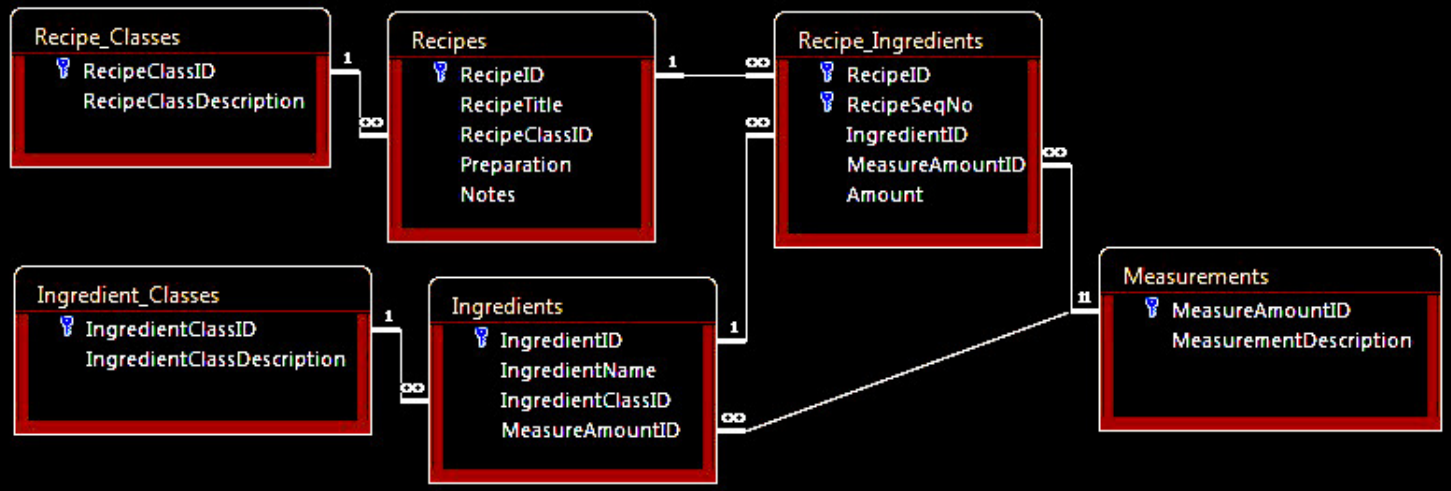
```
SELECT RecipeClassDescription
FROM Recipe_Classes LEFT NATURAL JOIN Recipes
GROUP BY RecipeClassID HAVING SUM(RecipeID IS NOT NULL) = 0;
```

or

```
SELECT RecipeClassDescription FROM Recipe_Classes
WHERE RecipeClassID NOT IN (SELECT DISTINCT RecipeClassID FROM Recipes);
```

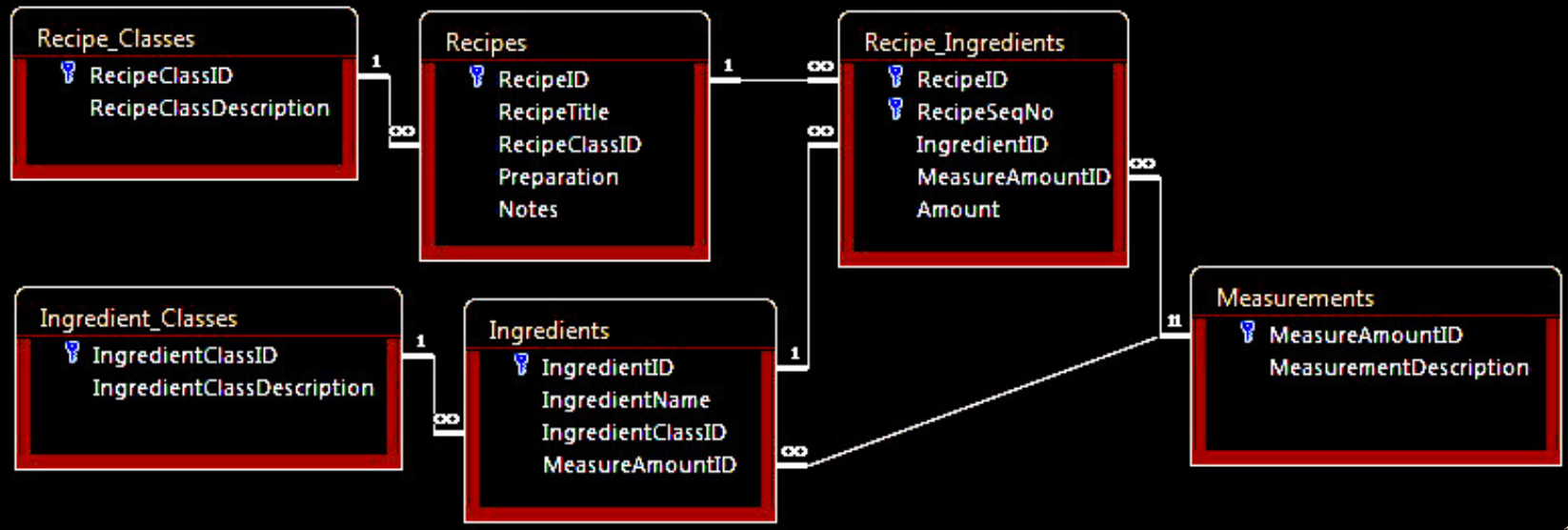
or

```
SELECT RecipeClassID FROM Recipe_Classes
EXCEPT SELECT DISTINCT RecipeClassID FROM Recipes;
```



Recipes.sqlite: "List the number of recipes in each category (RecipeClassID)"

```
SELECT RecipeClassDescription,
       SUM(RecipeID IS NOT NULL) AS RecipeCount
FROM Recipe_Classes
     LEFT NATURAL JOIN Recipes
GROUP BY RecipeClassID;
```

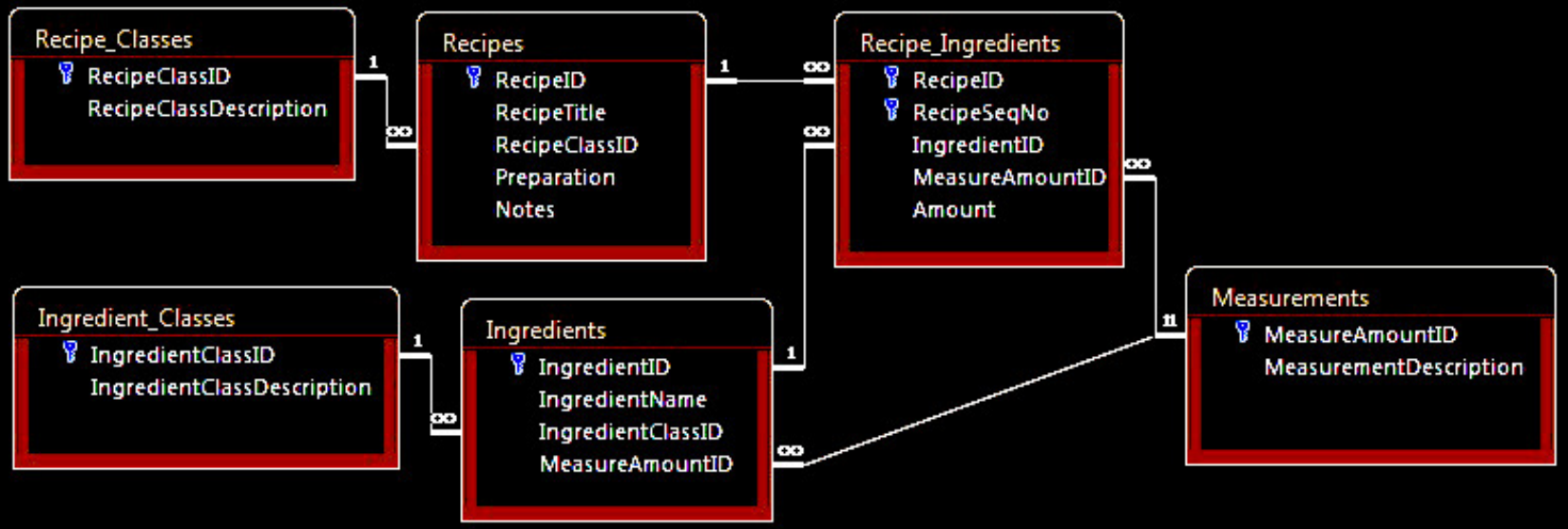


`Recipes.sqlite`: "Show me all ingredients and any recipes they are used in"

```
SELECT IngredientName, RecipeTitle FROM Ingredients
  LEFT NATURAL JOIN Recipe_Ingredients LEFT NATURAL JOIN Recipes;
```

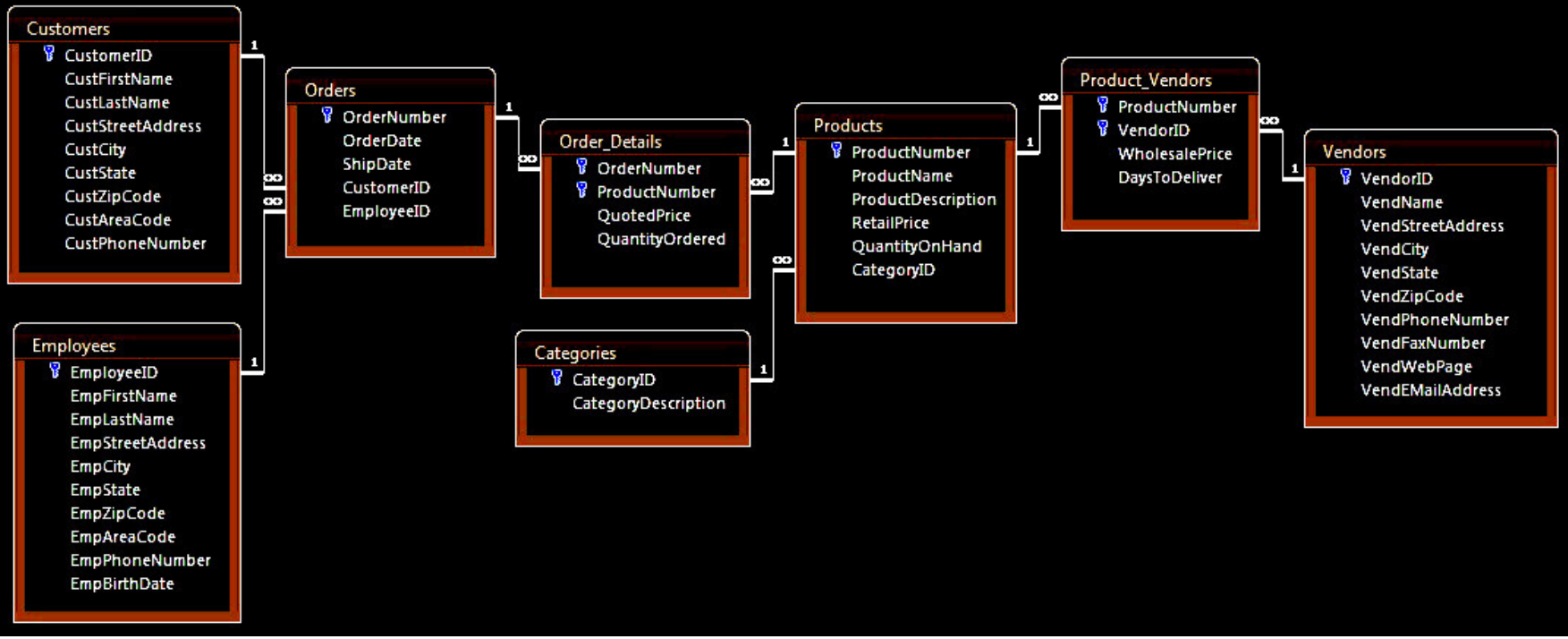
or, if only used ingredients are desired:

```
(Recipe_Ingredients NATURAL JOIN Recipes);
```



`SalesOrders.sqlite`: "Display customers who have no sales rep (employees) in the same ZIP Code"

- `SELECT * FROM Customers LEFT JOIN Employees ON CustZipCode=EmpZipCode WHERE EmpZipCode IS NULL;`
- `SELECT * FROM Customers WHERE CustZipCode IN (SELECT CustZipCode FROM Customers EXCEPT SELECT EmpZipCode FROM Employees);`



`SalesOrders.sqlite`: "List all products and the dates for any orders (of that product)"

```
SELECT Products.ProductNumber, ProductName, OrderDate
FROM Products
     LEFT NATURAL JOIN (Order_Details
                        NATURAL JOIN Orders);
```