CS 217 Data Management and Information Processing

OUTER JOINs and CROSS JOINs

Last Week: GROUP BY and JOINs

- Default type of JOIN is the **INNER JOIN**
- Combines rows from two tables using a **join predicate**, which usually specifies that two columns must be equal.
- Multiple JOINs can be combined
- Refer to columns as *table.column*
- Can use AS to give a table an alias for use in the statement
  - Do this when joining a table two or more times, to distinguish each copy of the table.

NATURAL JOIN

- A shorthand notation to make some JOINs shorter to express.
- NATURAL JOIN matches rows using whatever columns have identical names.

For example:

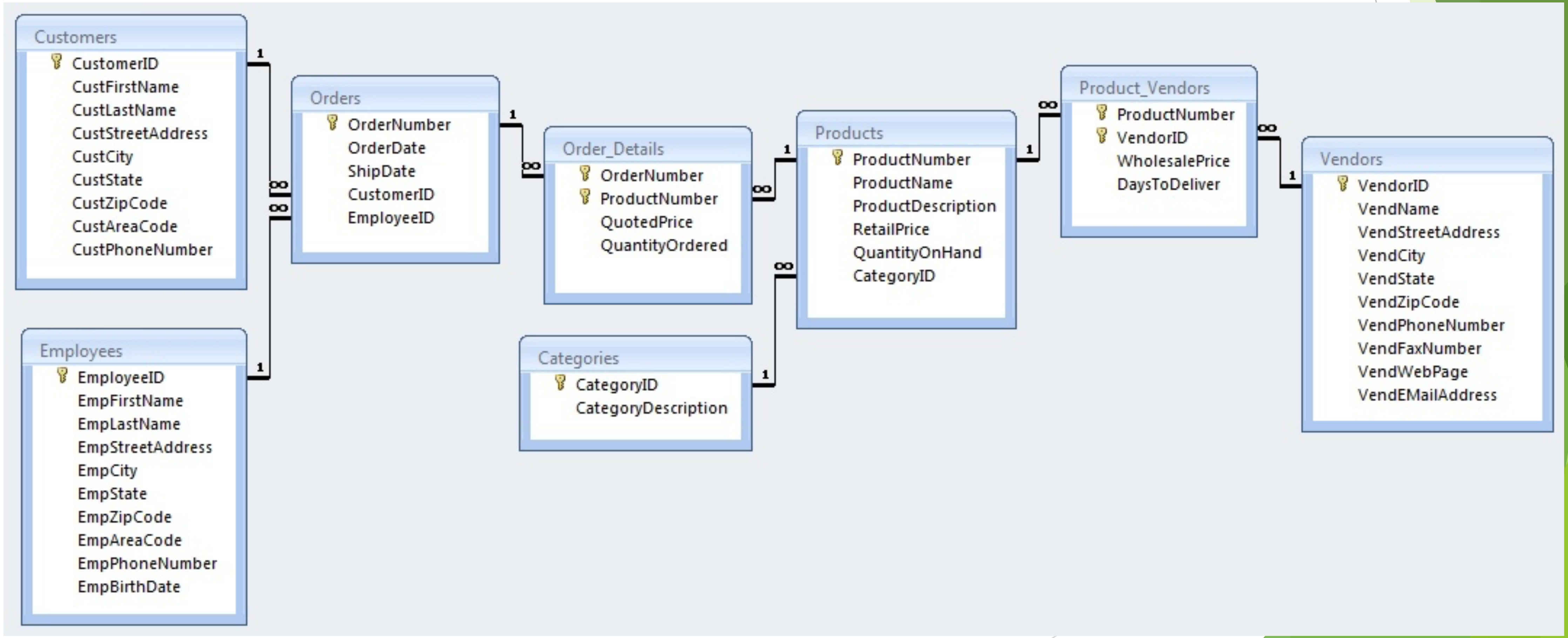
```
SELECT * FROM Orders JOIN Order_Details

ON Orders.OrderNumber=Order_Details.OrderNumber;
```

Very similar to:

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

Designing your data model NATURAL-ly

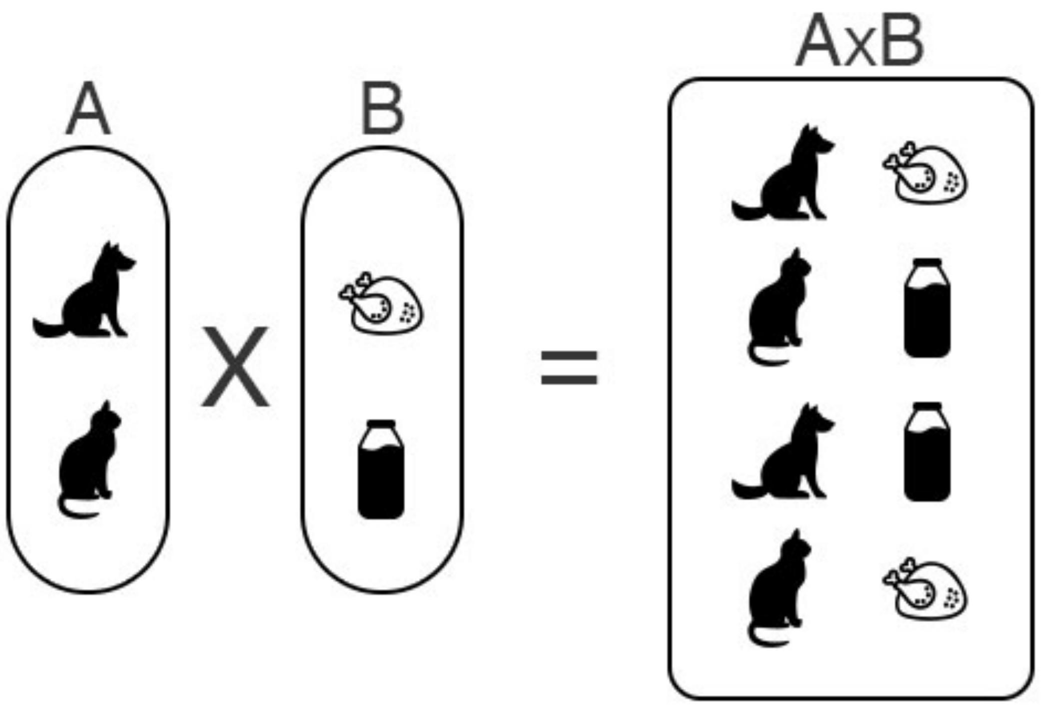
- Consistent column naming allows you to use NATURAL JOINs.



CROSS JOIN is like the cartesian product of two sets

- Take every element (row) of the first set (table) and combine it with every element of the second set.
- If first set has N elements and second set has M elements, then cartesian product has N·M elements.
- There is no “ON” expression to limit results:

  -
    ```
    SELECT Orders
    CROSS JOIN Order_Details;
    ```



Cartesian Product of Two Sets.

ON functions are exactly like WHERE

These two expressions are actually equivalent:

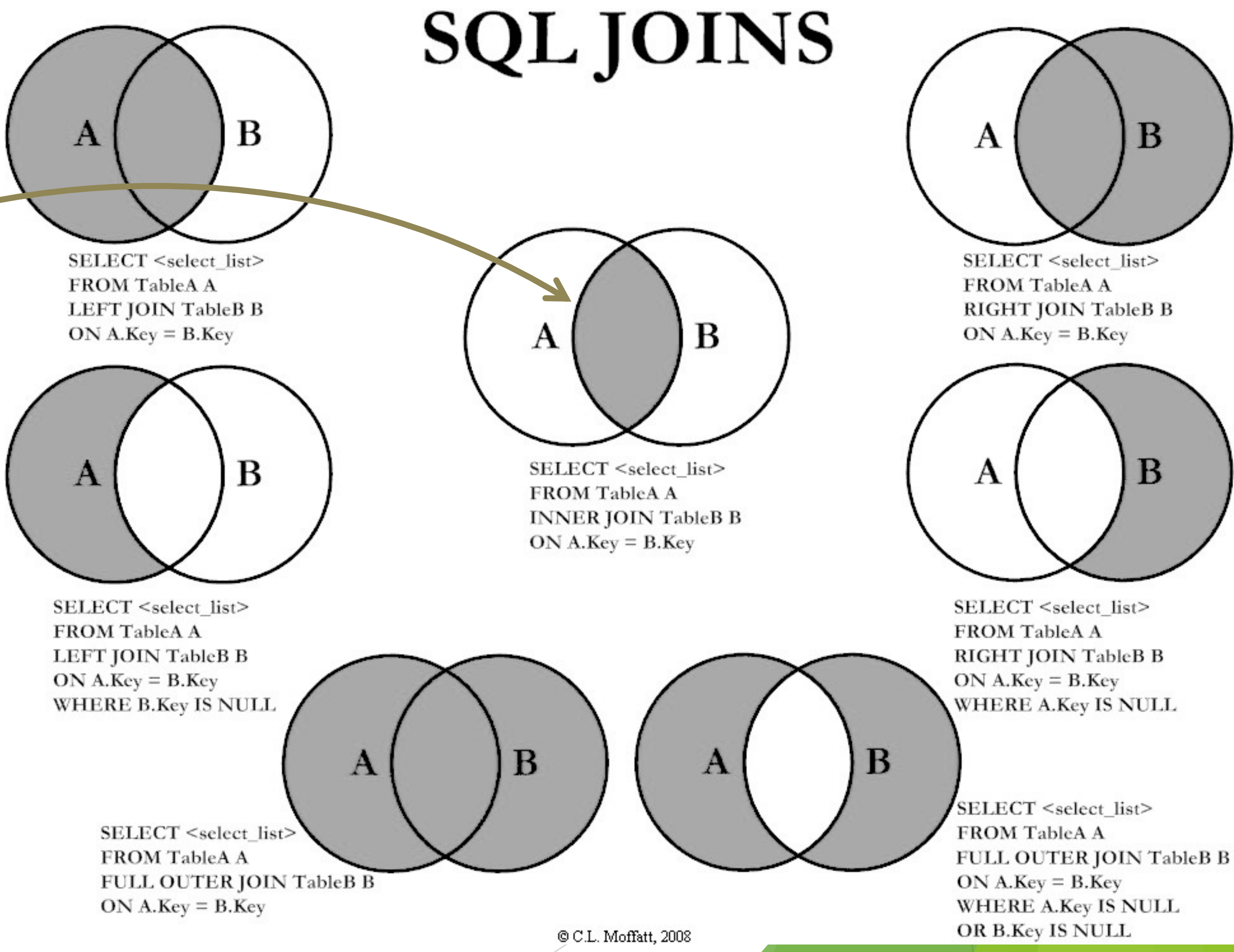
-
  ```
  SELECT * FROM Orders
  JOIN Order_Details
  ON Orders.OrderNumber=Order_details.OrderNumber;
  ```

-
  ```
  SELECT * FROM Orders
  CROSS JOIN Order_Details
  WHERE Orders.OrderNumber=Order_details.OrderNumber;
  ```

- However, using ON may be more efficient because it tells the DBMS to avoid building the full N·M cartesian product, and just match rows according to a rule.
- It also makes the join easier to think about, by separating the filtering and JOINing predicates.

Different JOINs

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



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

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

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

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

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

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

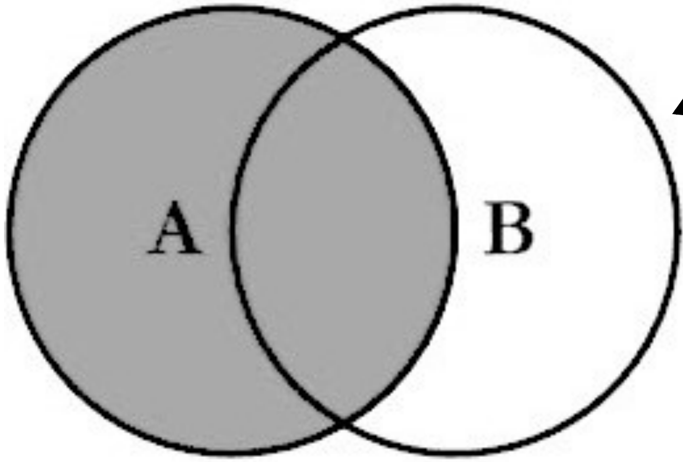
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

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

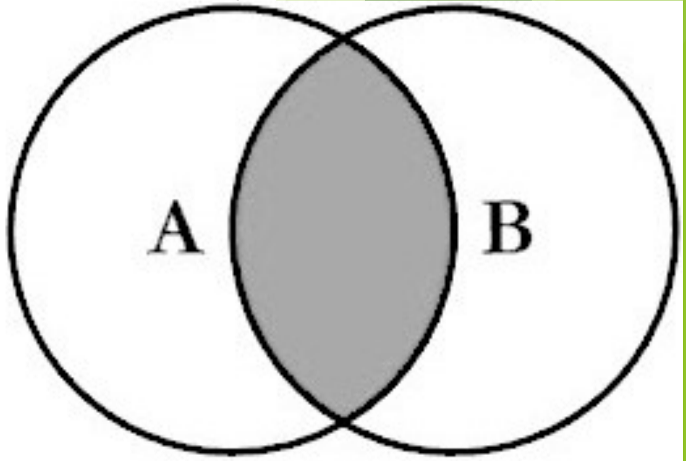
All rows from First table



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 *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**.

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 ON staff.departmentId=department.id

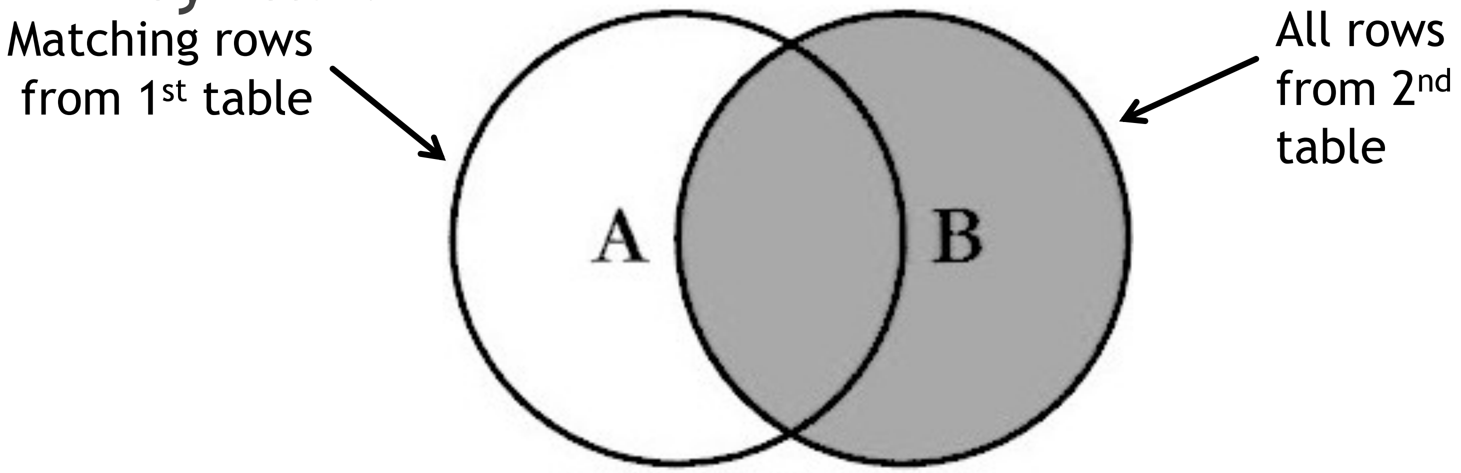
```
SELECT * FROM staff LEFT JOIN department ON staff.departmentId=department.id;
```

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

RIGHT JOIN is symmetrical to LEFT

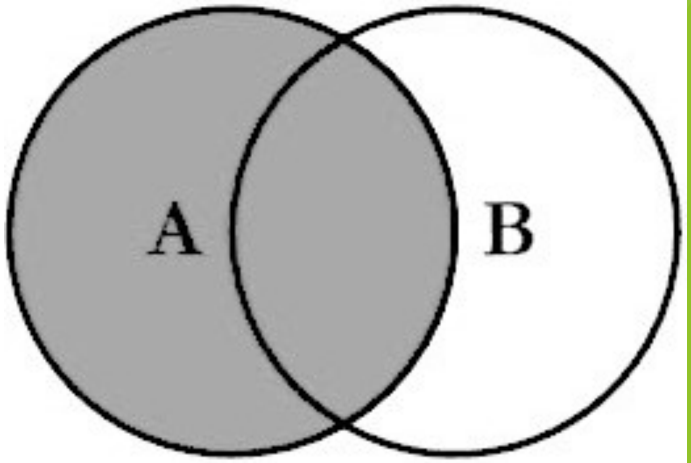
- Includes all rows from right table and matching rows from left table
- Reordering the tables makes a RIGHT JOIN a LEFT JOIN, so it is not necessary to use the RIGHT JOIN syntax.

RIGHT JOIN



```
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 GROUP BY

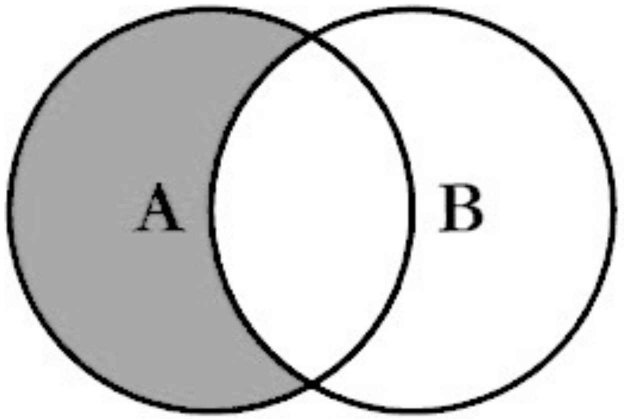
In ClassScheduling.slite, count the classes taught by each faculty member:

- If you want this report to include faculty members teaching zero classes, you must use LEFT JOIN:

```
SELECT StaffID,
    COUNT(ClassID) AS num_classes
  FROM Faculty NATURAL LEFT JOIN
Faculty_Classes
  GROUP BY StaffID;
```

- Note that "`COUNT(*)`" would return "1" for faculty members with no classes, because there would still be one unmatched row from the left table.

LEFT JOIN with exclusion

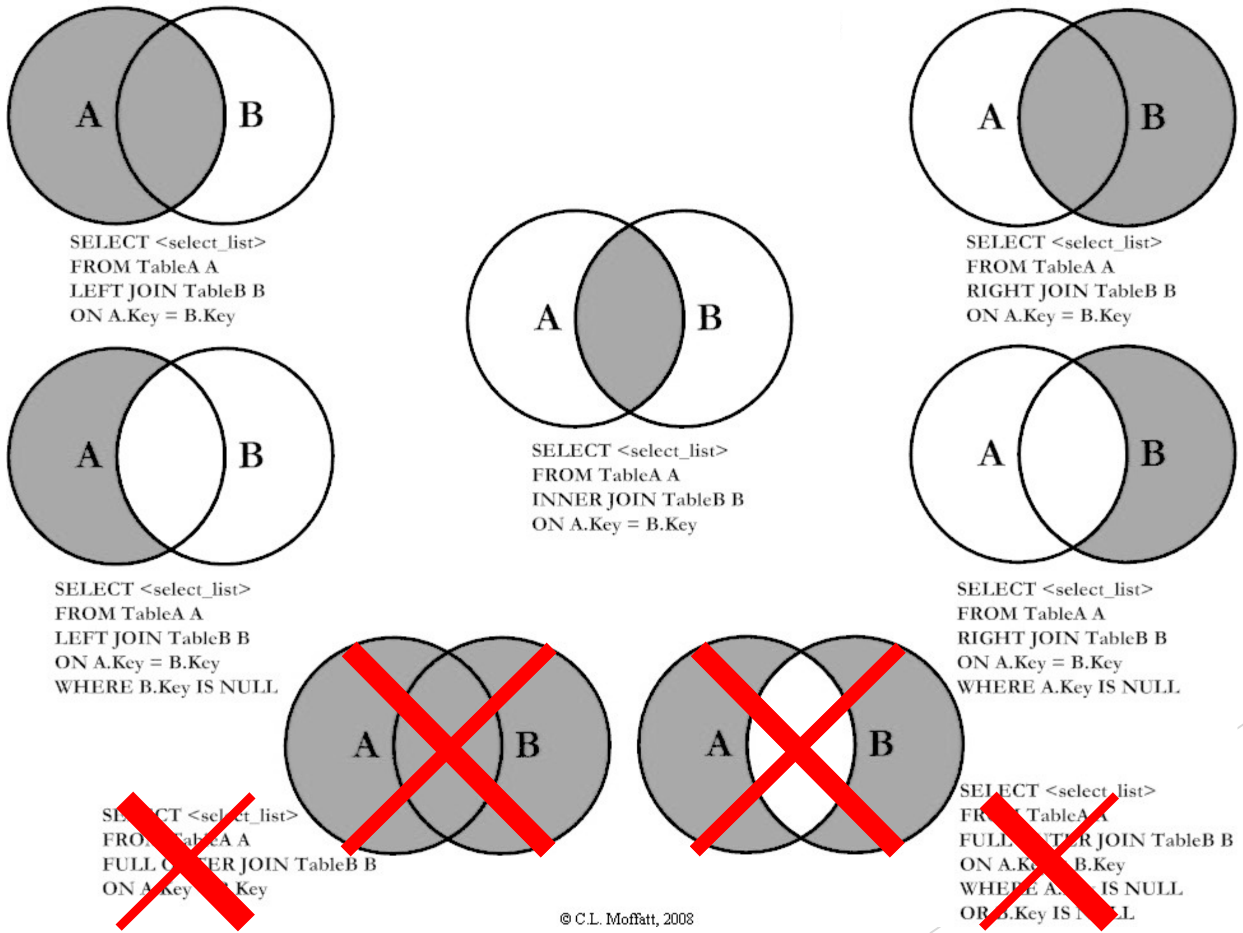


```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

- 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`**`;`

FULL OUTER JOINs are not available in MySQL or SQLite

- You can *emulate* `FULL OUTER JOIN` with the `UNION` of two queries.



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

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

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

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

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

~~SE CT <sel ct_list>~~
~~FRO abl A A~~
~~FULL O ER JOIN TableB B~~
~~ON A Key Key~~

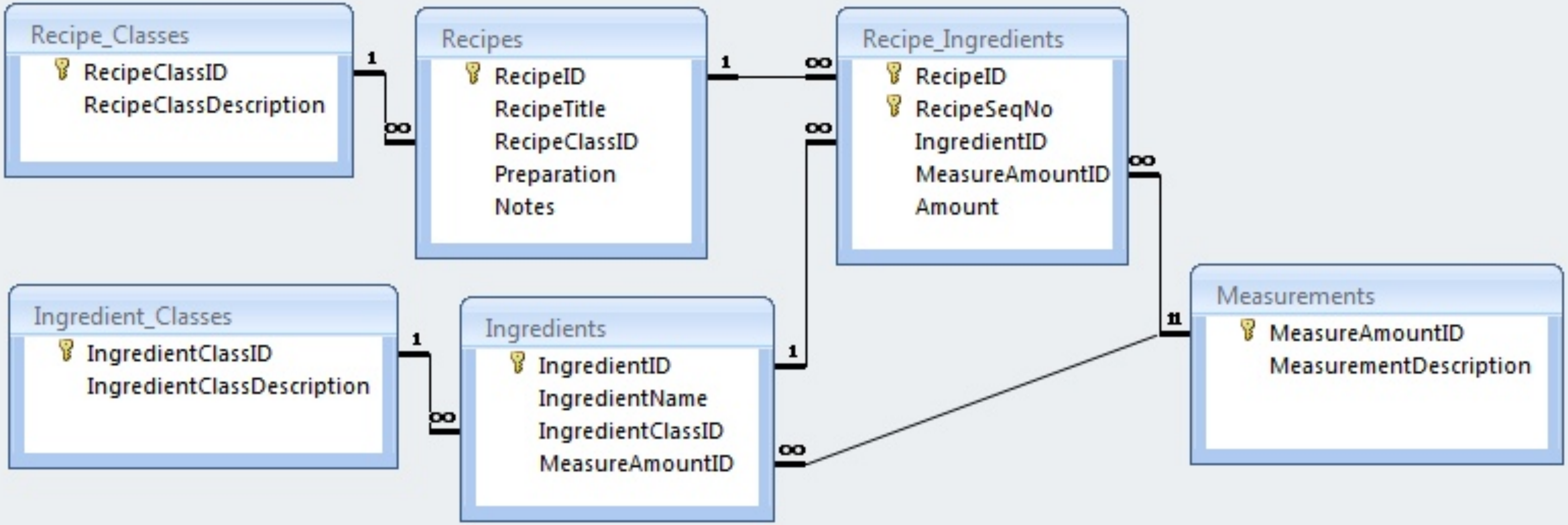
~~SELECT <select_list>~~
~~FR TableA A~~
~~FULL TER JOIN TableB B~~
~~ON A.K B.Key~~
~~WHE E A. IS NULL~~
~~OR B.Key IS LL~~

© C.L. Moffatt, 2008

Examples

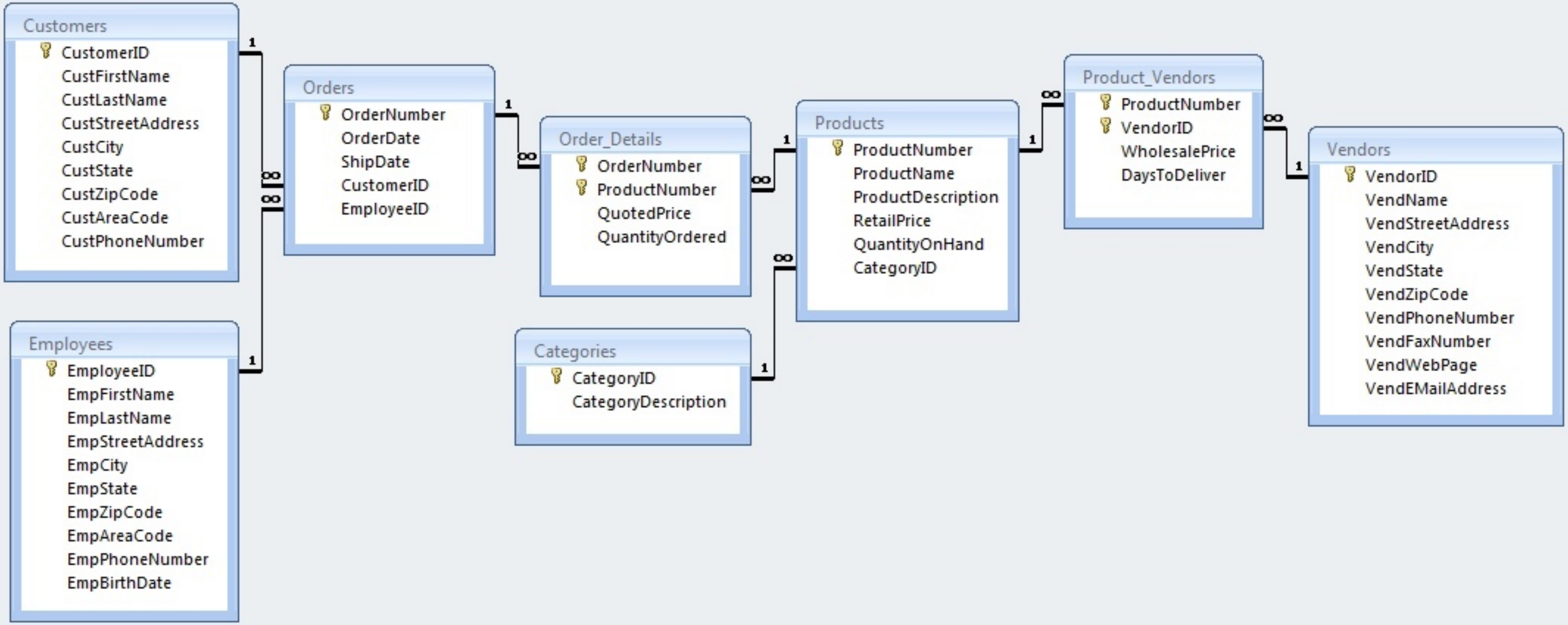
Recipes.sqlite: List the number of recipes in each category (RecipeClassID)

```
SELECT RecipeClassDescription, COUNT(RecipeID) AS RecipeCount
FROM Recipe_Classes LEFT NATURAL JOIN Recipes GROUP BY RecipeClassID;
```



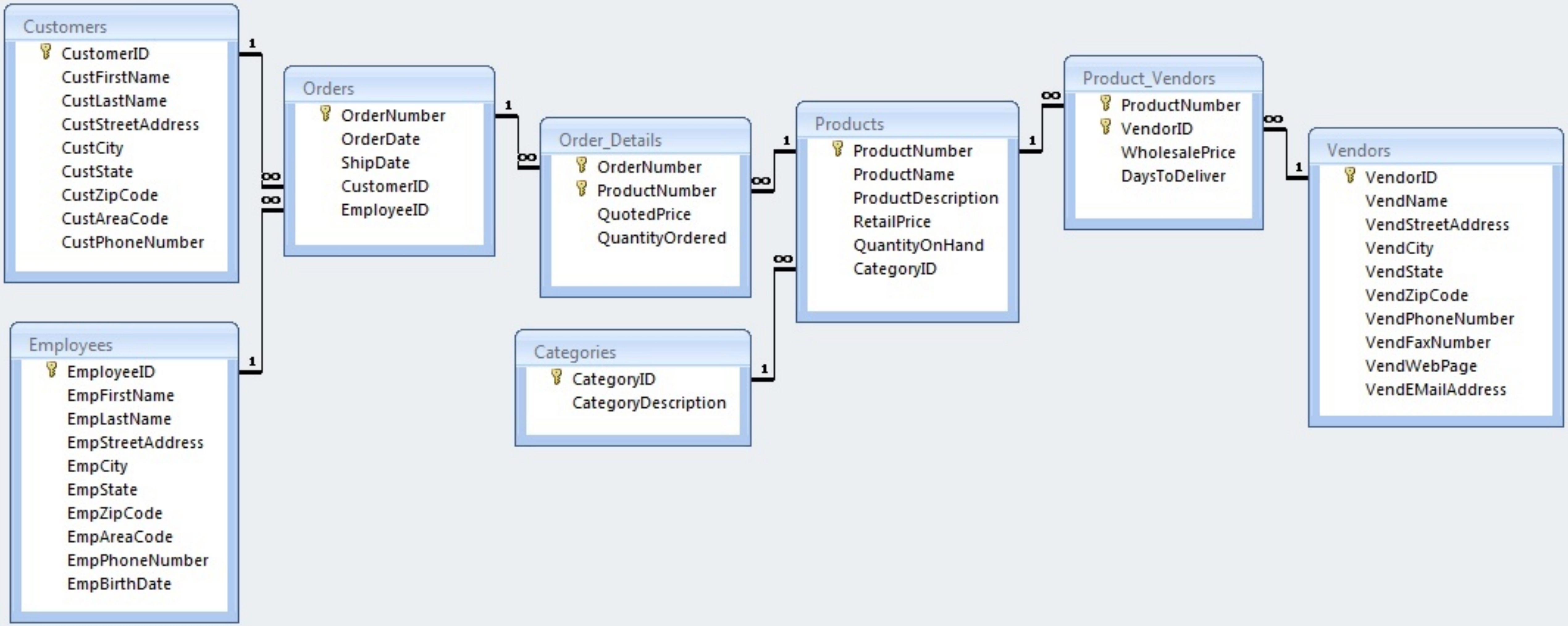
SalesOrders.sqlite: For all products, list any orders of that product and their dates.

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



Display customers who have no sales rep (employees) in the same ZIP Code.

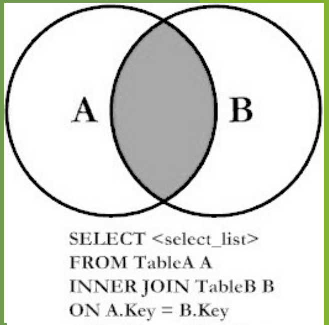
```
SELECT * FROM Customers LEFT JOIN Employees ON
CustZipCode=EmpZipCode WHERE EmpZipCode IS NULL;
```



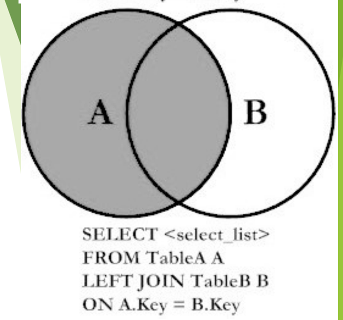
Recap

Introduced different types of JOINs:

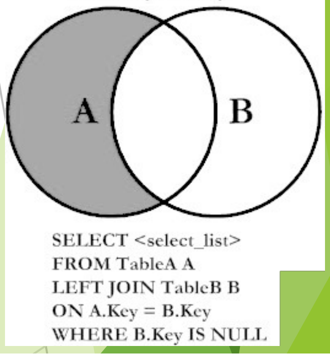
- **INNER (default)**: prints all pairs of rows (one from first table, one from second table) that satisfy the *JOIN predicate*.
- **LEFT**: same as INNER, but adds rows from LEFT table that never satisfied the JOIN predicate.
- **LEFT with exclusion**: only print rows form left table that never satisfied the JOIN predicate.
- **CROSS JOIN**: print the cartesian product, meaning all rows from the first table combined with all rows from the second table. There is no "ON" to match rows.



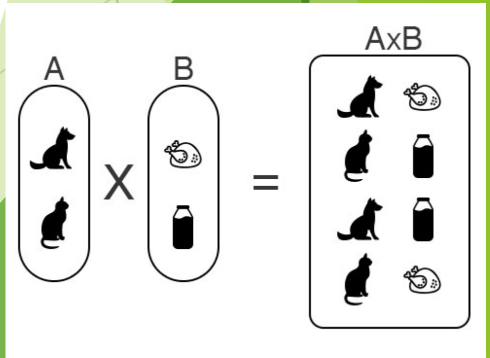
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



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



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



Cartesian Product of Two Sets.