EECS-317 Data Management and Information Processing

Lecture 6 – Combining SELECTs, Advanced Predicates

Steve Tarzia

Spring 2019

Northwestern

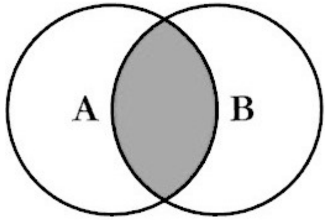
Announcements

- Second HW assignment due Monday night.
- HW1 solutions will be posted soon.
- Additional practice homework is posted in “files” section of Canvas. (Will not be graded.)

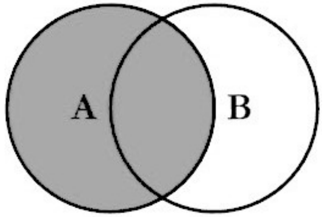
Last Lecture: OUTER and CROSS JOINs

Introduced different types of JOINs:

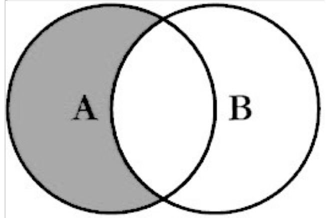
- **INNER** (default): prints all pairs of rows (one from first table, one from second table) that satisfy the *JOIN predicate.*
- **LEFT**: same as INNER, but adds rows from LEFT table that never satisfied the JOIN predicate.
- **LEFT with exclusion**: only print rows from left table that never satisfied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



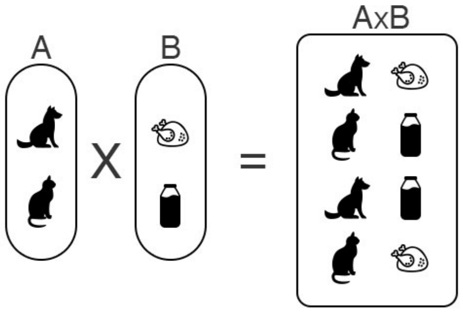
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



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

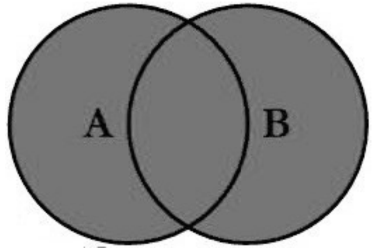


```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

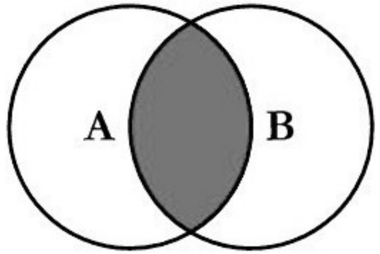


Cartesian Product of Two Sets.

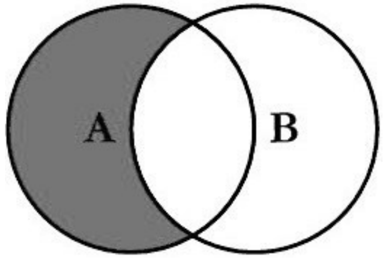
UNION, INTERSECT, and EXCEPT
are used to combine two SELECT statements



- **`UNION`** prints rows from *either of two* `SELECTs` (printing duplicates just once)



- **`INTERSECT`** prints rows *present in both* `SELECTs`

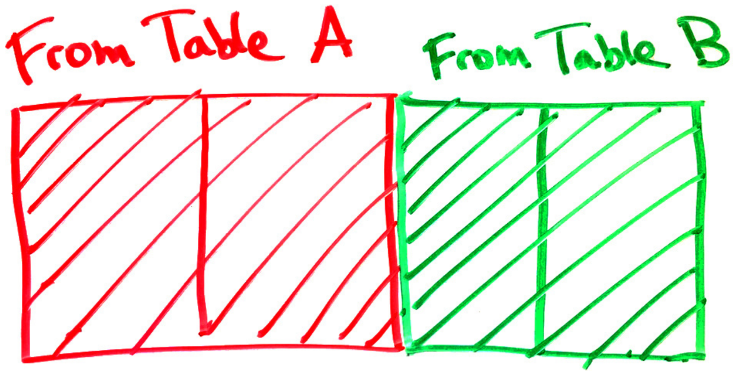


- **`EXCEPT`** prints rows *present in one* `SELECT` but *missing from another* `SELECT`

JOIN vs. **UNION**

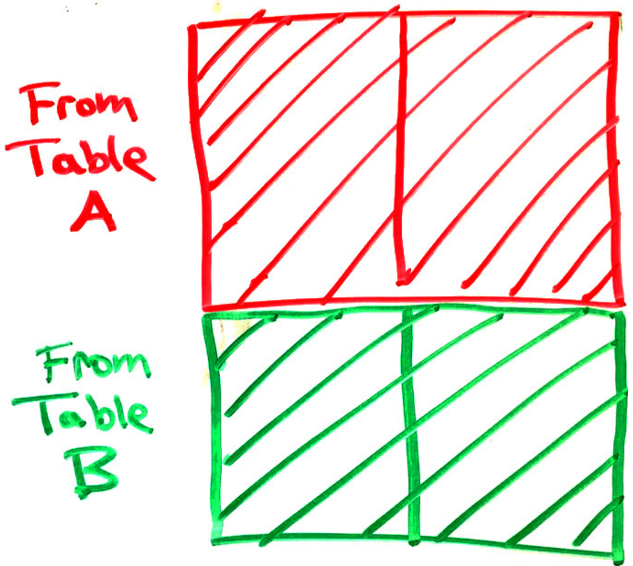
- `JOIN`s combine tables *horizontally*.
  - Match rows from two tables based on one or more columns matching.
  - Creates a wider set of rows, adding **columns** from both tables.

JOIN:



- `UNION`, `INTERSECT`, and `EXCEPT` combine result tables
  - Number & type of columns in the two result tables must match
  - Changes the number of **rows**, not columns

UNION:



Details of combining SELECTs

- UNION, INTERSECT, and EXCEPT all "combine" the results of two SELECT statements.
- UNION is the simplest, it just prints all rows from both:

```
SELECT … UNION SELECT …
```

- Duplicates are printed just once.
- Each SELECT statement gets data from a *different set of tables*, otherwise it would be easier to just use a WHERE clause (with AND).
- Left and right SELECT queries must return the same number of columns, and the matching columns must have compatible data types.
- For example: list names of all the Customers and Employees:

```
SELECT CustFirstName FROM Customers
  UNION SELECT EmpFirstName FROM Employees
```

Misuses of UNION, INTERSECT, and EXCEPT

Two SELECTs are not necessary if you can get an answer from just one virtual table.

```
SELECT * FROM Staff WHERE name="Jane”
   UNION SELECT * FROM Staff WHERE name="John”;
```

simplify to:

```
SELECT * FROM Staff WHERE name="Jane" OR name="John”;
```

```
SELECT * FROM Student_Schedules NATURAL JOIN Students
 EXCEPT
  SELECT * FROM Student_Schedules NATURAL JOIN Students
    WHERE Grade IS NULL;
```

simplify to:

```
 SELECT * FROM Student_Schedules NATURAL JOIN Students
   WHERE Grade IS NOT NULL;
```

“Display missing types of recipes” (1 row)

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

Predicates in more detail

- `WHERE` & `HAVING` filter rows according to conditions called *predicates*.
- Any of the following can be combined, like an algebraic expression:
  - Binary operations (used between two things):

    `=   !=   >   <   >=   <=   LIKE   AND   OR   REGEXP` ←*(coming soon!)*

    `+   -   *   /   ||   %   <<   >>   &   |`
  - `NOT …`
  - `… IS NULL,   … IS NOT NULL`
  - `… BETWEEN … AND …`
  - `… IN (…,…,…)`
  - `(…)`
- Can also use all of the above in the columns we print out, and inside aggregations like `SUM, MIN, MAX, AVG`

Summing an indicator variable

Two ways to count recipes with “salsa” in description:

- `SELECT COUNT(*) FROM Recipes WHERE` **`RecipeTitle LIKE "%salsa%”;`**
  - `WHERE` clause keeps just the rows matching “salsa,” then these rows are counted.
- `SELECT SUM(`**`RecipeTitle LIKE "%salsa%"`**`) FROM Recipes;`
  - A column is created for every recipe indicating whether its title matches “salsa” or not.
  - 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 matching recipes.
- First approach is easier to understand, but second is shorter.

CASE conditional

- Many programming languages have **`if`** … **`then`** … **`else`** … expressions.
- SQL's equivalent is `CASE`:

`CASE WHEN … THEN … ELSE … END`

- Condition after `WHEN` is checked for true/false (1/0)
  - If the condition is true, then the expression after `THEN` is used
  - Otherwise (if the condition is false), then the expression after `ELSE` is used

- For example, print *firstName* for children or *Mr/Ms lastName* for adults:

```
SELECT CASE WHEN age<18 THEN firstName ELSE
   (CASE WHEN gender="male" THEN "Mr. " ELSE "Ms. " END
    || lastName) END FROM people;
```

CASE in more detail

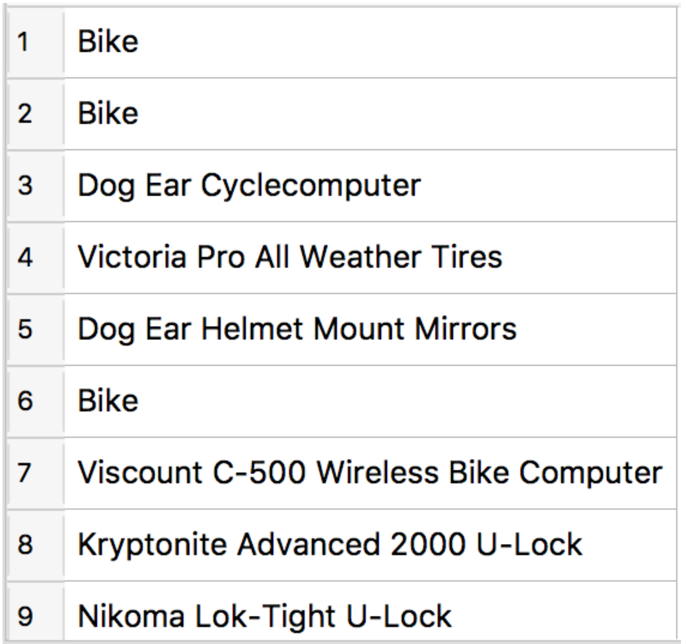
WHEN condition is tested for every row, giving *true* or *false*

```
SELECT CASE WHEN CategoryID=2
       THEN  "Bike"
       ELSE  ProductName  END FROM Products;
```

If condition is *true* then use the first value.

If condition is *false* then use the second value.

Output:

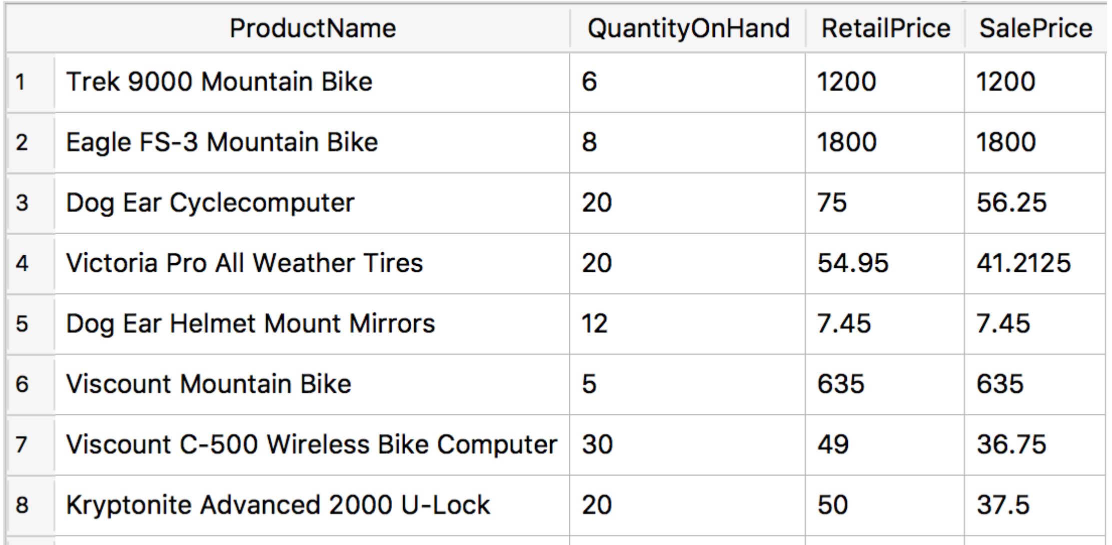


1	Bike
2	Bike
3	Dog Ear Cyclecomputer
4	Victoria Pro All Weather Tires
5	Dog Ear Helmet Mount Mirrors
6	Bike
7	Viscount C-500 Wireless Bike Computer
8	Kryptonite Advanced 2000 U-Lock
9	Nikoma Lok-Tight U-Lock

Another CASE example

- Let’s say we want to print “sale prices” for products that are overstocked. Any products with 20 or more items in stock are discounted 25%, but other products remain at regular retail price.

```
SELECT ProductName, QuantityOnHand, RetailPrice,
CASE WHEN QuantityOnHand >= 20 THEN
0.75*RetailPrice ELSE RetailPrice END AS SalePrice
FROM Products;
```



	ProductName	QuantityOnHand	RetailPrice	SalePrice
1	Trek 9000 Mountain Bike	6	1200	1200
2	Eagle FS-3 Mountain Bike	8	1800	1800
3	Dog Ear Cyclecomputer	20	75	56.25
4	Victoria Pro All Weather Tires	20	54.95	41.2125
5	Dog Ear Helmet Mount Mirrors	12	7.45	7.45
6	Viscount Mountain Bike	5	635	635
7	Viscount C-500 Wireless Bike Computer	30	49	36.75
8	Kryptonite Advanced 2000 U-Lock	20	50	37.5

CASE can also be used in filters

Print customers named “Martin” but refer to the first name in the friendly state of California and the last name elsewhere.

```
SELECT * FROM Customers WHERE CASE WHEN CustState = "CA" THEN
CustFirstName ELSE CustLastName END = "Martin";
```

Incidentally, this is equivalent to:

```
SELECT * FROM Customers WHERE
   (CustState = "CA" AND CustFirstName = "Martin")
  OR (CustState != "CA" AND CustLastName = "Martin");
```

Tell me if each recipe is vegetarian, and if not, then name the meat ingredient.

Print a different message for veg/meat recipes

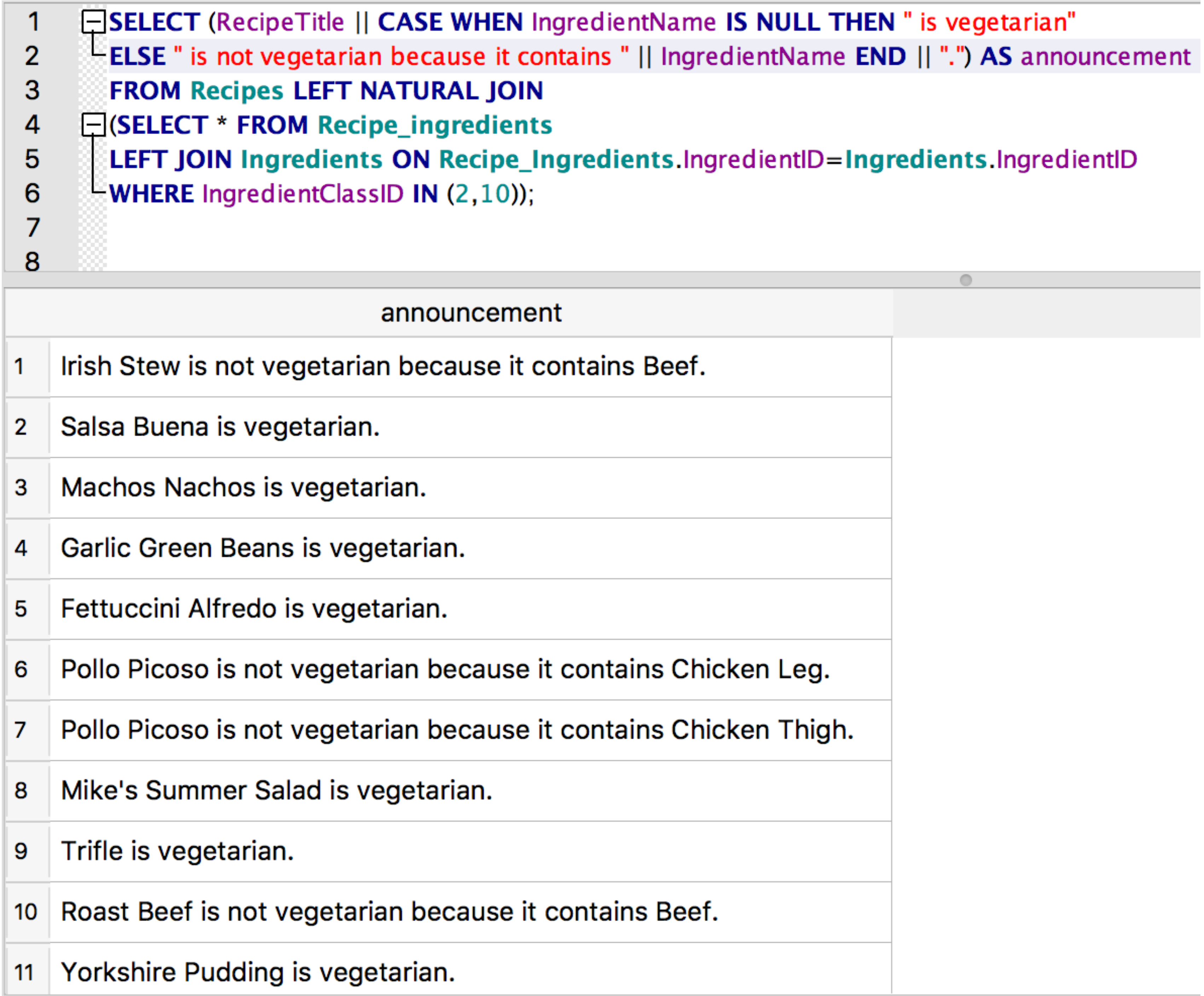
```
SELECT (RecipeTitle ||
  CASE WHEN IngredientName IS NULL THEN " is vegetarian"
  ELSE " is not vegetarian because it contains "
       || IngredientName END || ".") AS announcement
FROM Recipes LEFT NATURAL JOIN
(SELECT * FROM Recipe_Ingredients
  LEFT JOIN Ingredients ON
  Recipe_Ingredients.IngredientID=Ingredients.IngredientID
  WHERE IngredientClassID IN (2,10));
```

`LEFT JOIN` with a table printing only the meat/seafood recipe steps

Meat or seafood

*Note that a `NATURAL JOIN` cannot be used between Recipe_Ingredients and Ingredients because they have two columns in common (IngredientID and MeasureAmountID) and MeasureAmountID does not always match.

The result:



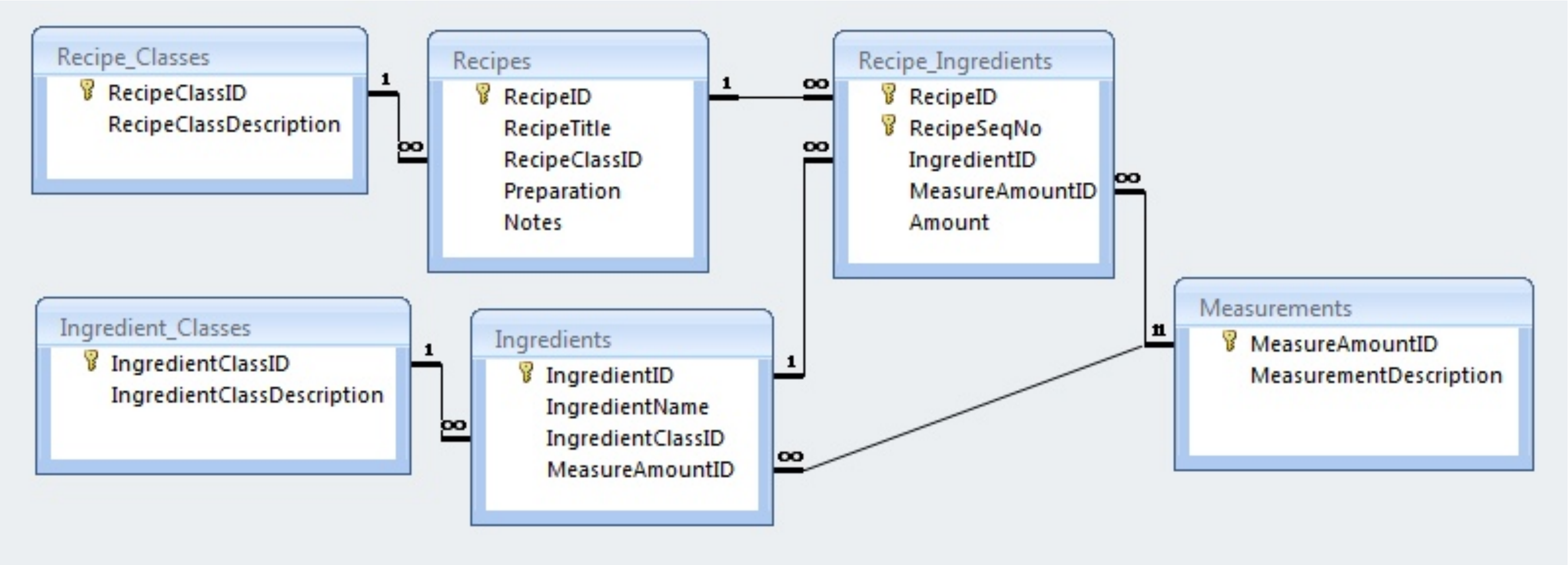
```
SELECT (RecipeTitle || CASE WHEN IngredientName IS NULL THEN " is vegetarian"
ELSE " is not vegetarian because it contains " || IngredientName END || ".") AS announcement
FROM Recipes LEFT NATURAL JOIN
(SELECT * FROM Recipe_ingredients
LEFT JOIN Ingredients ON Recipe_Ingredients.IngredientID=Ingredients.IngredientID
WHERE IngredientClassID IN (2,10));
```

	announcement
1	Irish Stew is not vegetarian because it contains Beef.
2	Salsa Buena is vegetarian.
3	Machos Nachos is vegetarian.
4	Garlic Green Beans is vegetarian.
5	Fettuccini Alfredo is vegetarian.
6	Pollo Picoso is not vegetarian because it contains Chicken Leg.
7	Pollo Picoso is not vegetarian because it contains Chicken Thigh.
8	Mike's Summer Salad is vegetarian.
9	Trifle is vegetarian.
10	Roast Beef is not vegetarian because it contains Beef.
11	Yorkshire Pudding is vegetarian.

Could change the query to eliminate this duplication.

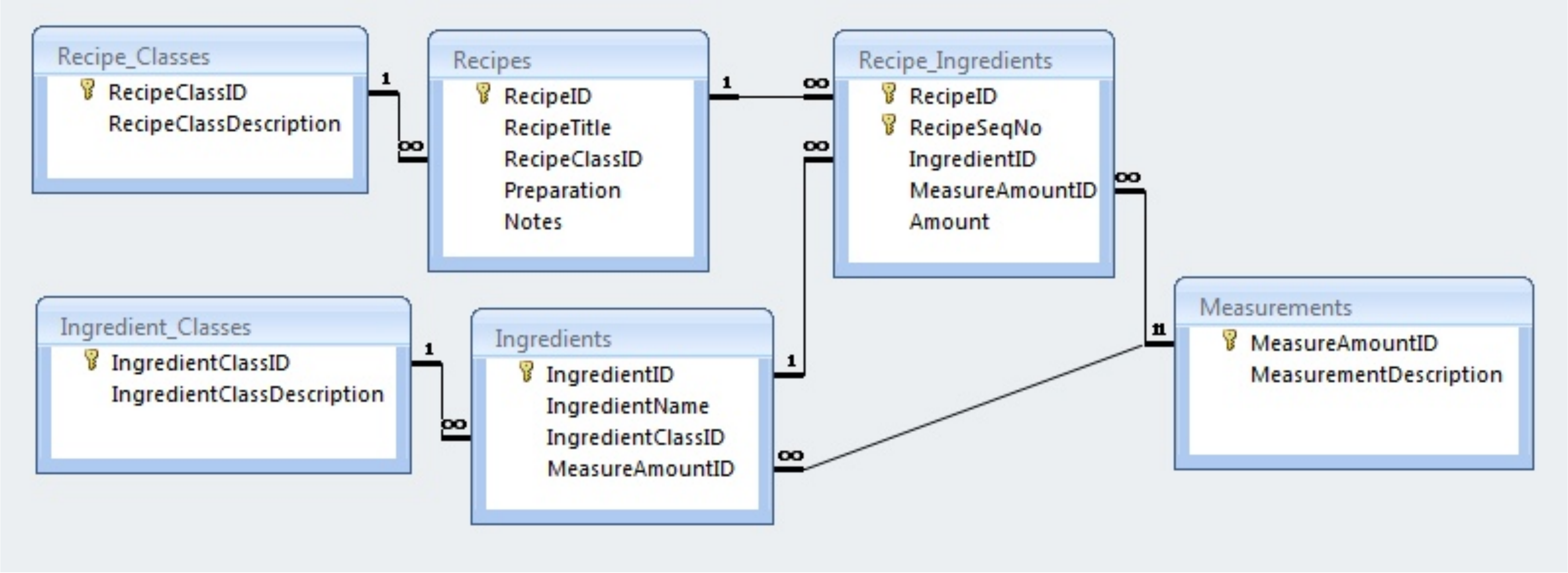
Recipes: Print every pair of recipes and the number of ingredients they share in common

```
SELECT r1.RecipeTitle, r2.RecipeTitle,
COUNT(i2.IngredientID) AS common_ingredients
FROM
  Recipes AS r1 CROSS JOIN Recipes AS r2
  JOIN Recipe_Ingredients AS i1 ON r1.RecipeID = i1.RecipeID
  LEFT JOIN Recipe_Ingredients AS i2 ON
    r2.RecipeID = i2.RecipeID AND i1.IngredientID=i2.IngredientID
GROUP BY r1.RecipeID, r2.RecipeID
HAVING r1.RecipeID < r2.RecipeID
ORDER BY common_ingredients DESC;
```



“Show me all ingredients and any recipes they’re used in” (108 rows)

```
SELECT IngredientName, RecipeTitle FROM Ingredients
LEFT JOIN Recipe_Ingredients
ON Ingredients.IngredientID=Recipe_Ingredients.IngredientID
LEFT NATURAL JOIN Recipes;
```



Recap

`UNION`, `INTERSECT`, and `EXCEPT`

- Used to combine two `SELECT` statements.
- Combines results table *vertically* (rather than horizontally for JOINs)
- Necessary when answer requires two different (virtual) tables.

- Discussed more advanced uses of predicates.
  - Summing an indicator variable.
- Introduced `CASE` statement which chooses between two different options depending on some condition in the row.