Lecture 5 – OUTER JOINs and CROSS JOINs

Steve Tarzia
Spring 2019
Announcements

• HW2 is due on Monday.
Last Lecture

• Illustrated GROUP BY
• Introduced 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
• Must 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.
INNER JOIN review

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>room</th>
<th>departmentId</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Bob</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>Betsy</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>Fran</td>
<td>101</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>buildingId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Industrial Eng.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Computer Sci.</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Chemistry</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Physics</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>Materials Sci.</td>
<td>5</td>
</tr>
</tbody>
</table>

SELECT * FROM staff JOIN department ON staff.departmentId=department.id

In output,
• multiple matches leads to multiple rows.
• no matches leads to no rows
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;
```

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 reason to avoid generic column names like “id” or “name”
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 $N \cdot M$ elements.
- There is no “ON” expression to limit results:
  - `SELECT * FROM Orders CROSS JOIN Order_Details;`
ON functions 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’s 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.
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 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.
• 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 \( \text{ON staff.departmentId} = \text{department.id} \).
LEFT JOIN with Grouping

• When computing an \textit{aggregation} on a \textit{many-to-one} relationship, LEFT JOIN includes rows from the parent table with no children.

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:

\begin{verbatim}
SELECT StaffID, ClassID, COUNT(ClassID) AS num_classes
FROM Faculty NATURAL LEFT JOIN Faculty_Classes
GROUP BY StaffID;
\end{verbatim}

• Note that “\texttt{COUNT (\*)}” would return “1” for faculty members with no classes, because there would still be one unmatched row from the left table.
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.
LEFT JOIN with exclusion

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

```sql
SELECT * FROM Customers LEFT JOIN Employees ON CustZipCode = EmpZipCode WHERE EmpZipCode IS NULL;
```
Show me customers who have never ordered a Watch.

First solution uses EXCEPT (introduced later), second solution uses LEFT JOIN with exclusion:

• SELECT CustomerID FROM Customers EXCEPT
  SELECT CustomerID FROM Customers NATURAL JOIN Orders
  NATURAL JOIN Order_Details NATURAL JOIN Products
  WHERE ProductName LIKE "%Watch%" GROUP BY CustomerID;

• SELECT CustomerID FROM Customers LEFT JOIN
  (SELECT CustomerID AS watch_customer FROM Orders
  NATURAL JOIN Order_Details NATURAL JOIN Products
  WHERE ProductName LIKE "%Watch%" GROUP BY CustomerID)
  ON CustomerID=watch_customer
  WHERE Watch_customer IS NULL;
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
```
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;
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 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.