SeQueL 2 – Queries - JOIN in Now!

by Clark Anderson

Please JOIN me in the long awaited continuation of the adventures in Structured Query Language. Last time in "**SQL Select Queries - the Beginning**", we explored a lot of ways to select records in one table. It is more fun when you can join together data from more than one table.

Are you hungry for some bagels? I baked up a big batch.

BagelsBaked Table

BakedID	BagelID	Name	Quantity
1	1	Plain	360
2	4	Sesame Seed	240
3	7	Poppy Seed	240
4	2	Egg	240

I also received some orders.

BagelsOrdered Table

OrderID	BagelID	Name	Quantity
1	1	Plain	240
2	2	Egg	120
3	5	Whole Wheat	120
4	3	Everything	60

I can join the information in these tables together with a query. The simplest query is an INNER JOIN.
I "Only included rows where the joined fields from both tables were equal". The values in the Name fields are identical.

```
SELECT BagelsBaked.Name, BagelsBaked.Quantity, BagelsOrdered.Quantity, BagelsOrdered.Name
FROM BagelsBaked INNER JOIN BagelsOrdered ON BagelsBaked.Name = BagelsOrdered.Name;
```

BagelsBaked		BagelsOrdered	
Name	Quantity	Quantity	Name
Egg	240	120	Egg
Plain	360	240	Plain

You can see that some of the bagels baked and some of the bagels ordered are not shown. Their Names do not match.

In the next query, I "Include ALL records from "BagelsBaked" and only those records from BagelsOrdered" where the joined fields are equal". This is a LEFT JOIN. In the Design View of Access the arrow is pointing from the Name field in the table, that has ALL of its records included, to the Name field in the other table. I guess you could say it is pointing from its primary data source to its secondary data reference.

```
SELECT BagelsBaked.Name, BagelsBaked.Quantity, BagelsOrdered.Quantity, BagelsOrdered.Name
FROM BagelsBaked LEFT JOIN BagelsOrdered ON BagelsBaked.Name = BagelsOrdered.Name;
```

BagelsBaked		BagelsOrdered	
Name	Quantity	Quantity	Name
Egg	240	120	Egg
Plain	360	240	Plain
Poppy Seed	240		
Sesame Seed	240		

The LEFT JOIN query results include bagels that have been baked but not ordered. It is worth noting that all of the records, specified by the field on the LEFT side of the SQL phrase ON BagelsBaked.Name = BagelsOrdered.Name, are included.

The next query does this the other way. A RIGHT JOIN “Includes ALL records from “BagelsOrdered” and only those records from “BagelsBaked” where the joined fields are equal”. In this case the Design View shows an Arrow pointing from the Name field in the “BagelsOrdered” table. ALL of its records are included.

SELECT BagelsBaked.Name, BagelsBaked.Quantity, BagelsOrdered.Quantity, BagelsOrdered.Name
FROM BagelsBaked RIGHT JOIN BagelsOrdered ON BagelsBaked.Name = BagelsOrdered.Name;

BagelsBaked		BagelsOrdered	
Name	Quantity	Quantity	Name
Egg	240	120	Egg
		60	Everything
Plain	360	240	Plain
		120	Whole Wheat

In the RIGHT JOIN query results, all of the bagels ordered are listed. In other words, all records are included for the field specified on the RIGHT side of the phrase (ON BagelsBaked.Name = BagelsOrdered.Name).

I have made these observations when using the Access Query Design View:

1) While dragging the mouse from a field in one table to a corresponding field in another table, The resulting SQL statement lists the table dragged from, first (e.g.: FROM BagelsBaked INNER JOIN BagelsOrdered).

2) Also, the Field dragged from, is listed first (e.g.: ON BagelsBaked.Name = BagelsOrdered.Name).

This makes sense. You are dragging FROM the source table to JOIN with another table.

My most frequent type of JOIN is with a lookup table, which can serve as a centralized repository of names that several tables can refer to.

LUBagel Table

BagelID	Name
1	Plain
2	Egg
3	Everything

4	Sesame Seed
5	Whole Wheat
6	Cinnamon Raisin
7	Poppy Seed

When I was preparing for this article, I started with the lookup table and added the Baked and Ordered tables. I was using JOINs to reference the lookup table, but that would have been too complicated to start with. I built a couple of UPDATE queries to fill the bagel names in the Baked and Ordered tables from the names with matching BagelIDs in LUBagel.

UPDATE LUBagel RIGHT JOIN BagelsBaked ON LUBagel.BagelID = BagelsBaked.BagelID SET BagelsBaked.Name = LUBagel.Name;

UPDATE LUBagel RIGHT JOIN BagelsOrdered ON LUBagel.BagelID = BagelsOrdered.BagelID SET BagelsOrdered.Name = LUBagel.Name;

In these queries I am using the BagelID to identify specific records in the tables. It is important to notice that the JOIN statement has two parts, joined tables: (LUBagel RIGHT JOIN BagelsOrdered) and joined fields: (ON LUBagel.BagelID = BagelsOrdered.BagelID).

```
SELECT LUBagel.Name, BagelsBaked.Quantity, BagelsOrdered.Quantity, LUBagel_1.Name
FROM LUBagel AS LUBagel_1 RIGHT JOIN (LUBagel RIGHT JOIN (BagelsBaked RIGHT JOIN BagelsOrdered ON
BagelsBaked.BagelID = BagelsOrdered.BagelID) ON LUBagel.BagelID = BagelsBaked.BagelID) ON LUBagel_1.BagelID
= BagelsOrdered.BagelID
ORDER BY LUBagel.Name;
```

Let me break this up a little:

```
SELECT LUBagel.Name, BagelsBaked.Quantity, BagelsOrdered.Quantity, LUBagel_1.Name FROM
LUBagel AS LUBagel_1 RIGHT JOIN (
LUBagel RIGHT JOIN (
BagelsBaked RIGHT JOIN BagelsOrdered
ON BagelsBaked.BagelID = BagelsOrdered.BagelID
) ON LUBagel.BagelID = BagelsBaked.BagelID
) ON LUBagel_1.BagelID = BagelsOrdered.BagelID
ORDER BY LUBagel.Name;
```

This query selects the same data records as the RIGHT JOIN query above. It demonstrates that JOINs can be nested. The first, JOINs a copy of the LUBagel table as a reference for the BagelsBaked table. The copy is aliased with the LUBagel AS LUBagel_1 phrase. The second, JOINs LUBagel table as a reference for the BagelsOrdered table. The third, JOINs the BagelsBaked table to the BagelsOrdered table.

LUBagel_1	Baked	Ordered	LUBagel
Name	Quantity	Quantity	Name
Egg	240	120	Egg
		60	Everything
Plain	360	240	Plain
		120	Whole Wheat

I believe this will give you something useful to experiment with until our next adventure in SQL.