“SQL in 10 minutes” by Ben Forta, Ch.12, 13

Presented by Mya Bakhova

Chapter 12

A JOIN is a method to combine information from 2 or more tables.

1. In base R `merge` (base), in dplyr `left_join`, `right_join`, `inner_join`, `outer_join`.
2. In Python we have Pandas `concat` and `merge` methods.
3. About Relative DB principles: There is detailed system of permissions in a SQL database. A data scientist who only needs to write queries and does not have permissions to change anything can survive without Relative DB principle knowledge, although it helps to optimize your work.
4. Tableau and Power BI provide graphical interface for Join methods

An example of a JOIN

```
SELECT vend_name, prod_name, prod_price

FROM vendors, products

WHERE vendor.vend_id=product.vend_id
```

1. In the `FROM` clause we have 2 tables
2. `WHERE` condition is for columns from different tables
3. Fully qualified column names
4. Each product name will be paired with its vendor name if exists, and each vendor name will be paired with a corresponding product name if any.

More about joins in SQL

5. Without column pairing in `WHERE` we get cartesian product (cross join)

6. `INNER JOIN` pairs existing values from both tables (equijoin)

7. Using `ON` clause

A more typical join format states a join type explicitly.

```
SELECT vend_name, prod_name, prod_price
FROM vendors
INNER JOIN product
ON vendor.vend_id=product.vend_id
```

Summary and Warnings

- Joins allow to optimize DB storage
- Join queries create a cornucopia of tables
- Join method skills are crucial for data download from DB

Warnings

1. Joins are resource intensive
2. Maximum number of tables in a join in some DB
3. DB specific syntax

Chapter 13. Creating advanced joins

Recall Aliases

- To name a calculated field
- To shorten a table name in fully qualified column names

A query from previous chapter:

```
SELECT vendors.vend_name, products.prod_name,
products.prod_price

FROM vendors, products

WHERE vendor.vend_id=product.vend_id
```

Chapter 13. Creating advanced joins

Recall Aliases

- To name a calculated field
- To shorten a table name in fully qualified column names

A query with aliases from previous chapter:

```
SELECT v.vend_name, p.prod_name, p.prod_price

FROM vendors AS v, products AS p

WHERE v.vend_id = p.vend_id
```

Warning: Drop AS for tables in Oracle.

Joining 3 Tables

```
SELECT C.cust_name, OI.cust_contract

FROM Customers as C, Orders as O, OrderItems as OI

WHERE C.cust_id = O.cust_id

AND OI.order_num = O.order_num

AND prod_id= ‘RGAN01’;
```

Self Joins

Find customer contacts who work for the same company as Jim Jones.

Solution with a subquery:

```
SELECT cust_id, cust_name, cust_contact

FROM Customers

WHERE cust_name = (SELECT cust_name
                   FROM Customers
                   WHERE cust_contact = 'Jim Jones';
```

Self Joins

Find customer contacts which work for the same company as Jim Jones.

Solution with a self-join:

```
SELECT c1.cust_id, c1.cust_name, c1.cust_contact
FROM Customers AS c1, Customers AS c2
WHERE c1.cust_name = c2.cust_name
AND c2.cust_contact = 'Jim Jones'
```

Remark: Self joins might be quicker than subqueries.

Natural Joins

A natural join finds same columns in different joined tables and returns only one. For this ask only for distinct columns.

```
SELECT C.*, O.order_num, O.order_date, OI.prod_id,
OI.quantity, OI.item_price

FROM Customers AS C, Orders AS O, OrderItems AS OI

WHERE C.cust_id = O.cust_id,

AND OI.order_num = O.order_num

And OI.prod_id = 'RGAN01'
```

Outer Joins

Recall: Inner join.

Sometimes you want to include rows from one table which do not have related rows in another table.

- Display all product names with corresponding vendor names, even if a vendor names is missing.
- Calculate average sale sizes, taking into account customers which have not placed an order.

Left Outer Join

All customers with their orders, even when a customer did not place an order yet:

```
SELECT C.cust_id, O.order_num

FROM Customer AS C

LEFT OUTER JOIN Orders AS O

ON C.cust_id = O.cust_id
```

Right Outer Join

All customers with their orders, even if a customer did not place an order yet:

```
SELECT C.cust_id, O.order_num

FROM Customer AS C

RIGHT OUTER JOIN Orders AS O

ON C.cust_id = O.cust_id
```

Remark: no right outer join in SQLite

Using Joins with Aggregate Functions, Example 1

We can summarize data with joins, too! Let us count placed orders for each customer.

```
SELECT C.cust_id,  COUNT(O.order_num) AS num_ord

FROM Customers as C

INNER JOIN Orders as O

ON C.cust_id = O.cust_id

GROUP BY C.cust_id;
```

Using Joins with Aggregate Functions, Example 2

Count all customer orders even if a customer did not placed one yet.

```
SELECT C.cust_id,  COUNT(O.order_num) AS num_ord

FROM Customers as C

LEFT OUTER JOIN Orders as O

ON C.cust_id = O.cust_id

GROUP BY C.cust_id;
```

Recommendations on working with joins

1. Pay careful attention to the type of join.
2. Check your DBMS documentation for exact syntax with joins (and other clauses).
3. Make sure to sure to use correct join condition.
4. Make sure to provide a join condition unless you want a Cartesian product.
5. You may include multiple tables in a join and even have different join types for each. Although this is legal and often useful, make sure you test each join separately before testing them together. This will make troubleshooting far simpler.