MSDS 597

Lecture 4

JOIN

- Combine (merge) data from two or more tables
- Uses a common key between the two tables to combine rows across tables

Syntax:

```
SELECT
        s.*,
        c.*
FROM sales s
JOIN customers c
    ON s.customer_id = c.customer_id
```

sales

customer_id	product_id	amount
1	987	25.49
2	876	123.25

customers

customer_id	name	address
1	Greg	123 Fake St.
2	Sally	135 Fake St.

customer_id	product_id	amount	customer_id	name	address
1	987	25.49	1	Greg	123 Fake St.
2	876	123.25	2	Sally	135 Fake St.

LEFT JOIN

chemistry

student_id	grade
1	95
2	90
3	85

physics

student_id	grade
1	93
2	91
4	75

```
SELECT
    *
FROM chemistry c
LEFT JOIN physics p
   ON c.student_id = p.student_id
```

All rows from the “left” side table and only the matching rows from the “right” side table

student_id	grade	student_id	grade
1	95	1	93
2	90	2	91
3	85		

RIGHT JOIN

chemistry

student_id	grade
1	95
2	90
3	85

physics

student_id	grade
1	93
2	91
4	75

```
SELECT
        *
FROM chemistry c
RIGHT JOIN physics p
    ON c.student_id = p.student_id
```

All rows from the “right” side table and only the matching rows from the “left” side table

student_id	grade	student_id	grade
1	95	1	93
2	90	2	91
		4	75

INNER JOIN

chemistry

student_id	grade
1	95
2	90
3	85

physics

student_id	grade
1	93
2	91
4	75

```
SELECT
        *
FROM chemistry c
INNER JOIN physics p
    ON c.student_id = p.student_id
```

student_id	grade	student_id	grade
1	95	1	93
2	90	2	91

Only the matching rows from both tables

JOIN is equivalent to INNER JOIN

Can act as a filter, similar to WHERE clause

FULL OUTER JOIN

chemistry

student_id	grade
1	95
2	90
3	85

physics

student_id	grade
1	93
2	91
4	75

```
SELECT
        *
FROM chemistry c
FULL OUTER JOIN physics p
    ON c.student_id = p.student_id
```

All rows from both tables

student_id	grade	student_id	grade
1	95	1	93
2	90	2	91
3	85		
		4	75

COALESCE

Returns the first non-NULL value of the provided values

a	b	COALESCE(a, b)
apple	banana	apple
NULL	banana	banana
apple	NULL	apple
NULL	NULL	NULL

Use cases

- consolidating values when performing JOINs
- setting a “default” value when you see a NULL

CROSS JOIN

chemistry

student_id	grade
1	95
2	90
3	85

physics

student_id	grade
1	93
2	91
4	75

```
SELECT
        *
FROM chemistry c
CROSS JOIN physics p;
```

Returns every possible combination of the rows

student_id	grade	student_id	grade
1	95	1	93
1	95	2	91
1	95	4	75
2	90	1	93
...	...	...	...

SELF JOIN

employees

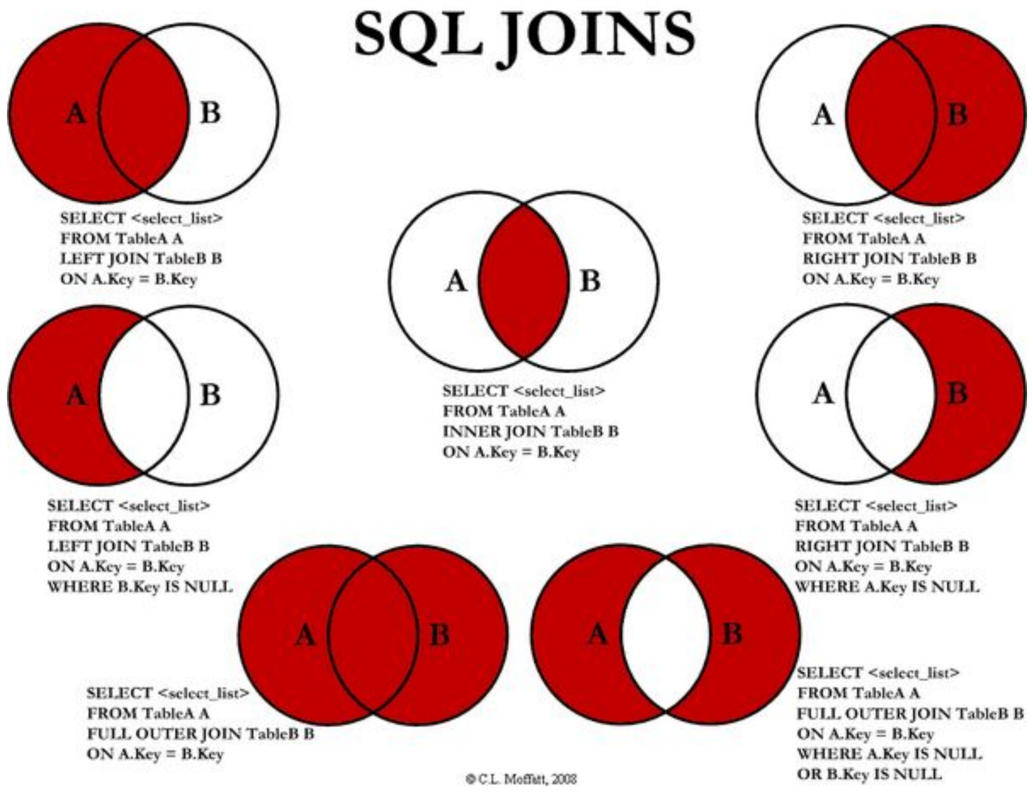
employee_id	name	manager_id
1	Todd	3
2	Henry	3
3	Mary	4
4	Justin	

```
SELECT
        emp.employee_id,
        emp.name,
        mgr.name AS manager_name
FROM employees emp
LEFT JOIN employees mgr
    ON emp.manager_id = mgr.employee_id
```

employee_id	name	manager_name
1	Todd	Mary
2	Henry	Mary
3	Mary	Justin
4	Justin	

In some situations, you may want to join a table with itself. This is called a self join. It's not a different type of join logic, but rather a reference to the fact that both sides of the join are the same table.

Different Types of JOINs



```
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
```

```
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
```

USING

If the columns that we are joining on have the **same name**, then we can use **USING** rather than **ON** to specify the join condition, as a short hand way of writing out the join conditions.

This produces an output similar to if a COALESCE was used on the key column.

```
SELECT
        *
FROM chemistry c
FULL OUTER JOIN physics p
USING (student_id)
```

=

```
SELECT
        COALESCE(c.student_id, p.student_id) AS student_id,
        c.grade,
        p.grade
FROM chemistry c
FULL OUTER JOIN physics p
ON c.student_id = p.student_id
```

Keys

Primary Key

- A column (or columns) that **uniquely** identify a row in a table
  - Must be unique in the table (e.g. student_id in a students table, or tx_id in a transaction table)
  - Must not be NULL
  - In Postgres (and other SQL variants), you can add a primary key constraint on column(s) in the table, which will enforce the **uniqueness** and ensure **non-NULL** values in that column(s)

Foreign Key

- A column in a table which references a column (e.g. a primary key) of another table
  - Ex: product_id in a transaction table where product_id is the primary key of a products table
  - In Postgres (and other SQL variants), you can add a foreign key constraint on column(s) in a table, which enforce that values in that column(s) are **valid values** from the referenced table and are **non-NULL**

Best practices when using JOIN

- Understand the structure of the tables that you are joining on.
    - Is there a primary key on either of the tables? What is it?
    - Which column(s) joins with which column(s)?
- Ensure that the column(s) you are joining on are unique in at least one of the tables.
    - Otherwise you end up with a many-to-many join, which lead to generally undesired and unintended results
- Alias your tables with meaningful abbreviations for readability and understandability