

Introduction to SQL

Afternoon Lecture - April 5, 2017

The logo for Galvanize, featuring a stylized 'g' icon followed by the word 'galvanize' in a lowercase, sans-serif font. The 'g' icon is composed of several small circles, and the text is in a light blue color.

Afternoon Objectives

- Clarify two things: Aliases and SELECT DISTINCT
- Build on understanding of JOINS
 - Joining more than two tables in one query
 - Joining to the same table multiple times (hint: aliases are key)
 - Joining to subqueries
- Learn to use temporary tables

Aliases

- In Postgres, Aliases can NOT be used in WHERE or HAVING clauses
- Aliases can be used in GROUP BY clauses

```
SELECT
    type AS meal_type,
    AVG(price) AS avg_price
FROM
    meals
WHERE
    type != 'french'
GROUP BY
    meal_type
HAVING
    AVG(price) > 2
```

SELECT DISTINCT

TABLE(S)

cars

make	model	category
Ford	Explorer	SUV
Ford	Focus	Sedan
Ford	Taurus	Sedan
Ford	Excursion	SUV
Ford	Expedition	SUV
Toyota	4Runner	SUV
Toyota	Highlander	SUV
Toyota	Camry	Sedan

QUERY

```
SELECT DISTINCT
  make
FROM
  cars;
```

```
SELECT DISTINCT
  make,
  category
FROM
  cars;
```

OUTPUT

make
Ford
Toyota

make	category
Ford	SUV
Ford	Sedan
Toyota	SUV
Toyota	Sedan

Queries with Multiple JOIN Clauses

Recall the original hypothetical table that we used as the basis for a 3-table database:

purch_id	cust_name	cust_state	description	price	date
1	Kayla	CO	skis	\$300	10/30
2	Kayla	CO	goggles	\$75	11/14
3	Erich	CO	snowboard	\$400	11/18
4	Adam	NY	skis	\$300	12/11
5	Frank	AZ	skis	\$300	12/19
6	Adam	NY	goggles	\$75	12/24

Queries with Multiple JOIN Clauses (cont'd)

How would we combine the tables below in *a single query* to re-form the original table?

customers

cust_id	cust_name	cust_state
1	Kayla	CO
2	Erich	CO
3	Adam	NY
4	Frank	AZ

products

prod_id	description	price
1	skis	300
2	goggles	75
3	snowboard	400

purchases

purch_id	cust_id	prod_id	date
1	1	1	10/30
2	1	2	11/14
3	2	3	11/18
4	3	1	12/11
5	4	1	12/19
6	3	2	12/24

Queries with Multiple JOIN Clauses (cont'd)

Recall that the first part evaluating of any query is to form a product of all tables based on the FROM and JOIN clauses.


```
SELECT ...  
FROM purchases AS p
```

p.purch_id	p.cust_id	p.prod_id	p.date
1	1	1	10/30
2	1	2	11/14
3	2	3	11/18
4	3	1	12/11
5	4	1	12/19
6	3	2	12/24

Queries with Multiple JOIN Clauses (cont'd)

Recall that the first part evaluating of any query is to form a product of all tables based on the FROM and JOIN clauses.

```
SELECT ...  
FROM purchases AS p  
LEFT OUTER JOIN customers AS c ON p.cust_id = c.cust_id
```



p.purch_id	p.cust_id	p.prod_id	p.date	c.cust_id	c.cust_name	c.cust_state
1	1	1	10/30	1	Kayla	CO
2	1	2	11/14	1	Kayla	CO
3	2	3	11/18	2	Erich	CO
4	3	1	12/11	3	Adam	NY
5	4	1	12/19	4	Frank	AZ
6	3	2	12/24	3	Adam	NY

Queries with Multiple JOIN Clauses (cont'd)

Recall that the first part evaluating of any query is to form a product of all tables based on the FROM and JOIN clauses.

```
SELECT ...  
FROM purchases AS p  
LEFT OUTER JOIN customers AS c ON p.cust_id = c.cust_id  
LEFT OUTER JOIN products AS pr ON p.prod_id = pr.prod_id
```

p.purch_id	p.cust_id	p.prod_id	p.date	c.cust_id	c.cust_name	c.cust_state	pr.prod_id	pr.description	pr.price
1	1	1	10/30	1	Kayla	CO	1	skis	300
2	1	2	11/14	1	Kayla	CO	2	goggles	75
3	2	3	11/18	2	Erich	CO	3	snowboard	400
4	3	1	12/11	3	Adam	NY	1	skis	300
5	4	1	12/19	4	Frank	AZ	1	skis	300
6	3	2	12/24	3	Adam	NY	2	goggles	75

Queries with Multiple JOIN Clauses (cont'd)

Then we specify which columns we want to keep,
and we have our answer.

```
SELECT
    p.purch_id,
    c.cust_name,
    c.cust_state,
    pr.description,
    pr.price,
    p.date
FROM
    purchases AS p
LEFT OUTER JOIN
    customers AS c
        ON p.cust_id = c.cust_id
LEFT OUTER JOIN
    products AS pr
        ON p.prod_id = pr.prod_id;
```

p.purch_id	c.cust_name	c.cust_state	pr.description	pr.price	p.date
1	Kayla	CO	skis	300	10/30
2	Kayla	CO	goggles	75	11/14
3	Erich	CO	snowboard	400	11/18
4	Adam	NY	skis	300	12/11
5	Frank	AZ	skis	300	12/19
6	Adam	NY	goggles	75	12/24

call_history

caller_id	receiver_id	date
3	4	10/30
2	4	11/14
3	2	11/18
4	1	12/11
2	3	12/19

customers

id	name
1	Kayla
2	Erich
3	Adam
4	Frank

Joining to the Same Table Twice

QUERY

```
SELECT
    caller.name AS caller_name,
    receiver.name AS receiver_name,
    ch.date
FROM
    call_history AS ch
LEFT OUTER JOIN
    customers AS caller
ON
    ch.caller_id = caller.id
LEFT OUTER JOIN
    customers AS receiver
ON
    ch.receiver_id = receiver.id;
```

OUTPUT

Who called whom?

caller_name	receiver_name	date
Adam	Frank	10/30
Erich	Frank	11/14
Adam	Erich	11/18
Frank	Adam	12/11
5	NULL	12/19

Using different aliases for the same table allows us to JOIN to that table multiple times ON different fields.

call_history

caller_id	receiver_id	date
3	4	10/30
2	4	11/14
3	2	11/18
4	1	12/11
2	3	12/19

customers

id	name
1	Kayla
2	Erich
3	Adam
4	Frank

Joining to the Same Table Twice

QUERY

```

SELECT
    customers.name,
    calls_made.total_calls
FROM
    customers
LEFT OUTER JOIN
    (SELECT
        caller_id,
        count(*) AS total_calls
    FROM call_history
    GROUP BY caller_id
    ) AS calls_made
ON
    customers.id = calls_made.caller_id;

```

OUTPUT

How many calls did each person make?

name	total_calls
Kayla	NULL
Erich	2
Adam	2
Frank	1

Again, aliasing a subquery allows us to refer to it after creation (in ON clause).

call_history

caller_id	receiver_id	date
3	4	10/30
2	4	11/14
3	2	11/18
4	1	12/11
2	3	12/19

customers

id	name
1	Kayla
2	Erich
3	Adam
4	Frank

Another way: Using Temp. Tables

QUERY

```
WITH calls_made AS
  (SELECT
    caller_id,
    count(*) AS total_calls
  FROM call_history
  GROUP BY caller_id)
```

```
SELECT
  customers.name,
  calls_made.total_calls
FROM
  customers
LEFT OUTER JOIN
  calls_made
ON
  customers.id = calls_made.caller_id;
```

OUTPUT

How many calls did each person make?

name	total_calls
Kayla	NULL
Erich	2
Adam	2
Frank	1

A single temporary table can be used in place of multiple identical subqueries.

Subquery vs Temp Table vs Create/Drop Table

All three approaches yield the same results. The best one might depend on how many times you will reference newTable.

```
SELECT
    newTable.col1,
    newTable.col2
FROM
    (SELECT
        col1,
        col2,
        col3
    FROM
        anotherTable
    ) AS newTable;
```

```
WITH newTable AS
    (SELECT
        col1,
        col2,
        col3
    FROM
        anotherTable)

SELECT
    newTable.col1,
    newTable.col2
FROM
    newTable;
```

```
CREATE TABLE newTable AS
    (SELECT
        col1,
        col2,
        col3
    FROM
        anotherTable);

SELECT
    newTable.col1,
    newTable.col2
FROM
    newTable;

DROP TABLE newTable;
```

Afternoon Objectives

- Build on understanding of JOINS
 - Joining more than two tables in one query
 - Joining to the same table multiple times (hint: aliases are key)
 - Joining to subqueries
- Learn to use temporary tables