Fall 2017 - W205 – Storing and Retrieving Data
Week 5 Live Class Session Agenda
Kevin R. Crook

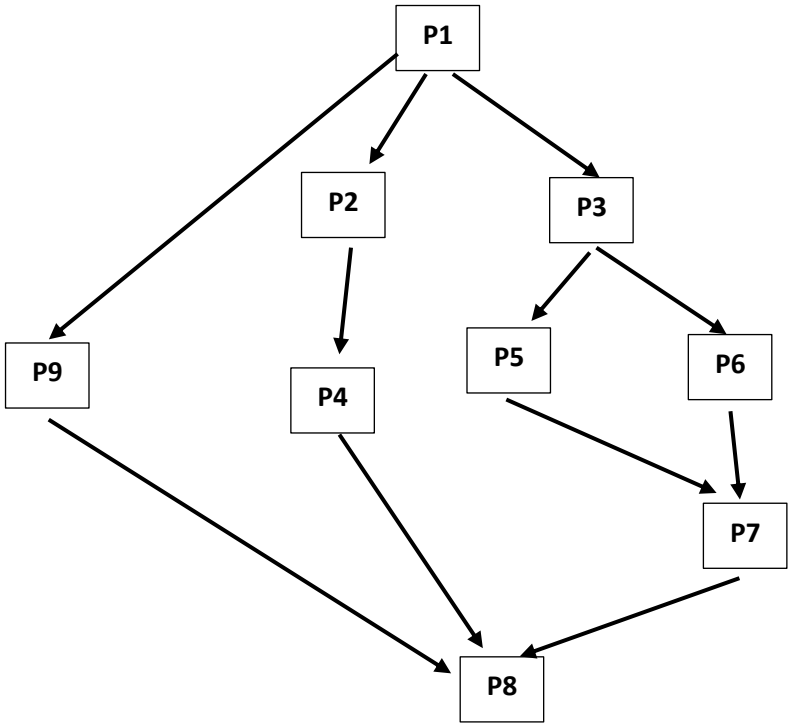
- Schedule
  - Lab 3 – due Tuesday, 10/3/2017 at 11:59 pm
  - Lab 4 – due Tuesday, 10/10/2017 at 11:59 pm
  - Asynchronous for next week
    - Unit 5 – Data Ingestion: Storage and Maintenance
  - Exercise 1
    - Due Tuesday 10/31/2017 at 11:59 pm
    - Suggestions:
      - Be reading documentation to familiarize yourself with the data set, focus on the 5 csv files we will be using
      - After you finish Lab 3, you will have the Hive skills to create schema on read definitions for these files
      - Try to get Hive schema on reads defined for the 5 files as soon as you can, so you can query the data to explore it
      - Explore and understand the data so you can design your analytics layers to answer the 4 questions

- Today in class
  - Finish up last week's break out exercise (we didn't get to the ERDs)
  - Instructor led exercise
    - DAGs
    - SQL execution plans
    - SQL – review the basics using PostgreSQL

(next page)

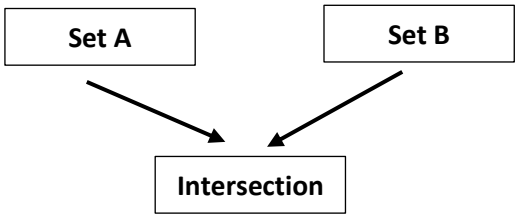
"Hard Skills" - Instructor Led Exercise
DAGs, SQL Execution Plans, SQL basics using PostgreSQL

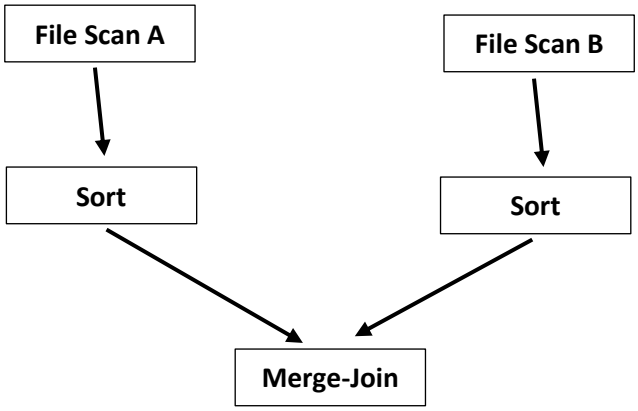
- DAG – Directed Acyclic Graph
  - Graph = vertices + edges
  - Directed Graph = Digraph = edges have directions
  - Acyclic = no cycles, cannot visit a vertex twice
  - Parallel Processing (and extended to Massively Parallel Processing – MPP)
    - vertex represents a process
    - edges show process dependencies
      - what processes must complete before a process can run
      - what processes can run at the same time
    - acyclic
      - no deadlock
      - no starvation
      - subject to the constraint that locks are properly taken and released
      - will eventually complete (under normal conditions - if no stack traces, etc)



- SQL execution plans
    - SQL – declarative (or functional) programming language
    - SQL execution plans – translate declarative into procedural
        - DAGs
            - Special fixed DAGs
            - Drawn upside down
            - No arrows on edges
            - Binary
        - Rule Based
            - One plan – fixed rubrics
        - Cost Based
            - Multiple plans – take one with lowest cost

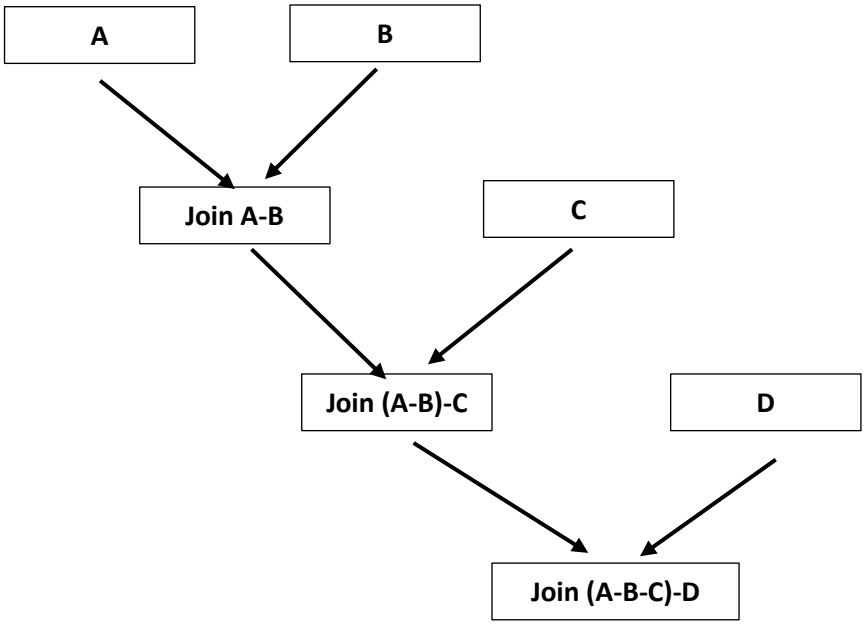
Graefe's paper, page 79

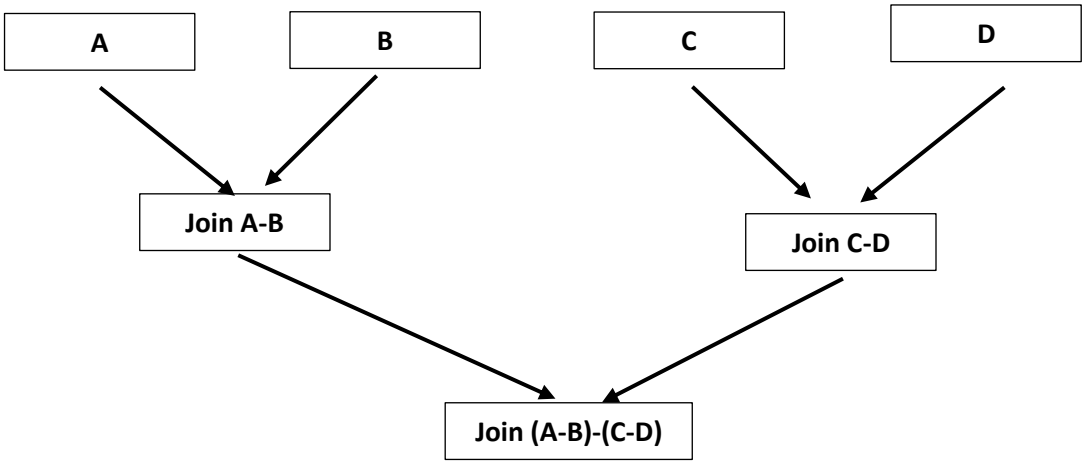




(next page)

Graefe's paper, page 81



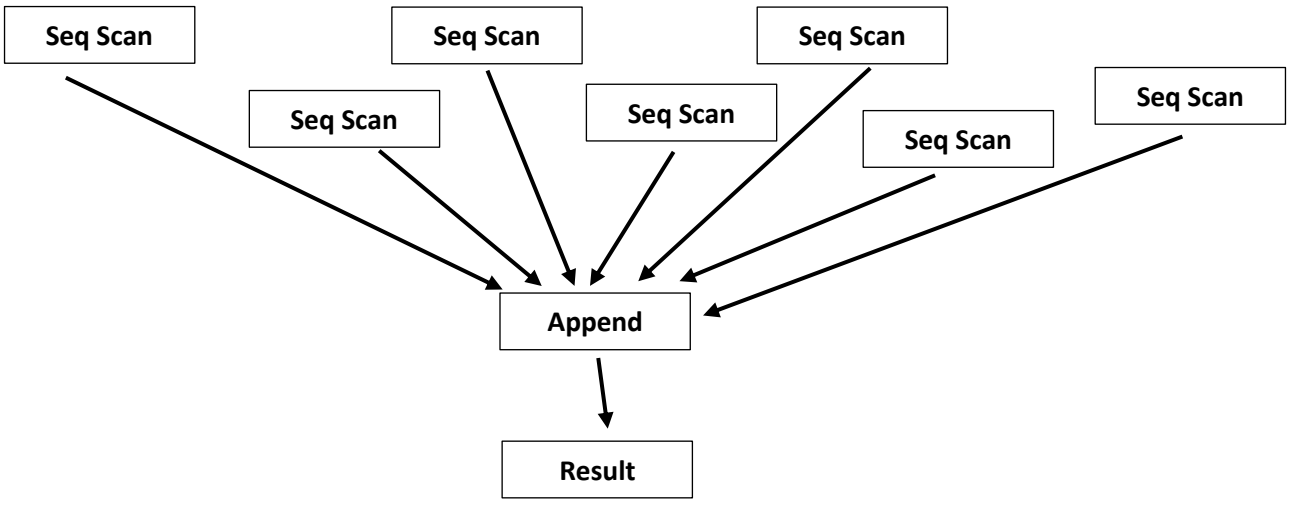


(next page)

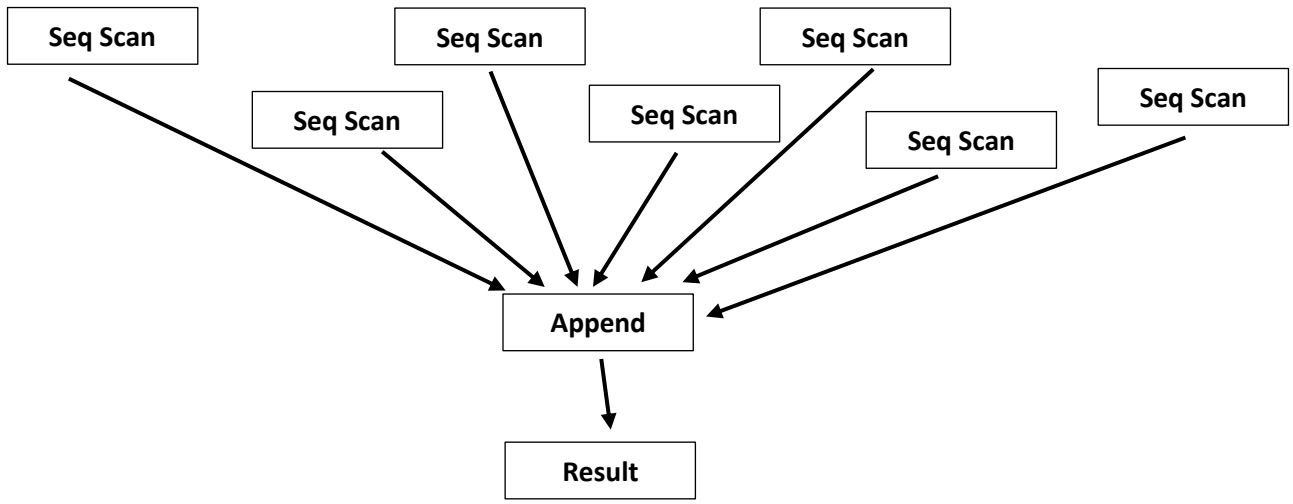
Lab 5 Queries, Execution Plans, and DAGs

```
EXPLAIN SELECT customer_id, first_name, last_name FROM customer;
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on customer  (cost=0.00..14.99 rows=599 width=17)
(1 row)
```


```
explain SELECT customer_id,
        amount,
        payment_date
FROM payment
WHERE amount <= 1
   OR amount >= 8;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Result  (cost=0.00..420.63 rows=5178 width=19)
   ->  Append  (cost=0.00..420.63 rows=5178 width=19)
         ->  Seq Scan on payment  (cost=0.00..29.95 rows=739 width=21)
               Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
         ->  Seq Scan on payment_p2007_01 payment  (cost=0.00..26.36 rows=266 width=18)
               Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
         ->  Seq Scan on payment_p2007_02 payment  (cost=0.00..51.68 rows=531 width=18)
               Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
         ->  Seq Scan on payment_p2007_03 payment  (cost=0.00..126.66 rows=1268 width=18)
               Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
         ->  Seq Scan on payment_p2007_04 payment  (cost=0.00..151.31 rows=1557 width=18)
               Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
         ->  Seq Scan on payment_p2007_05 payment  (cost=0.00..4.73 rows=78 width=17)
               Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
         ->  Seq Scan on payment_p2007_06 payment  (cost=0.00..29.95 rows=739 width=21)
               Filter: ((amount <= 1::numeric) OR (amount >= 8::numeric))
(16 rows)
```

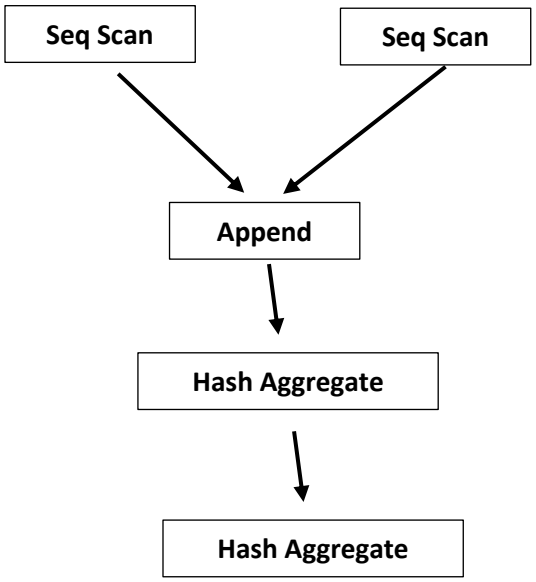


```
explain SELECT customer_id,
       payment_id,
       amount
FROM payment
WHERE amount BETWEEN 5 AND 9;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Result  (cost=0.00..420.63 rows=3600 width=14)
   ->  Append  (cost=0.00..420.63 rows=3600 width=14)
         ->  Seq Scan on payment  (cost=0.00..29.95 rows=7 width=17)
               Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
         ->  Seq Scan on payment_p2007_01 payment  (cost=0.00..26.36 rows=242 width=14)
               Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
         ->  Seq Scan on payment_p2007_02 payment  (cost=0.00..51.68 rows=506 width=14)
               Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
         ->  Seq Scan on payment_p2007_03 payment  (cost=0.00..126.66 rows=1290 width=14)
               Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
         ->  Seq Scan on payment_p2007_04 payment  (cost=0.00..151.31 rows=1535 width=14)
               Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
         ->  Seq Scan on payment_p2007_05 payment  (cost=0.00..4.73 rows=13 width=13)
               Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
         ->  Seq Scan on payment_p2007_06 payment  (cost=0.00..29.95 rows=7 width=17)
               Filter: ((amount >= 5::numeric) AND (amount <= 9::numeric))
(16 rows)
```



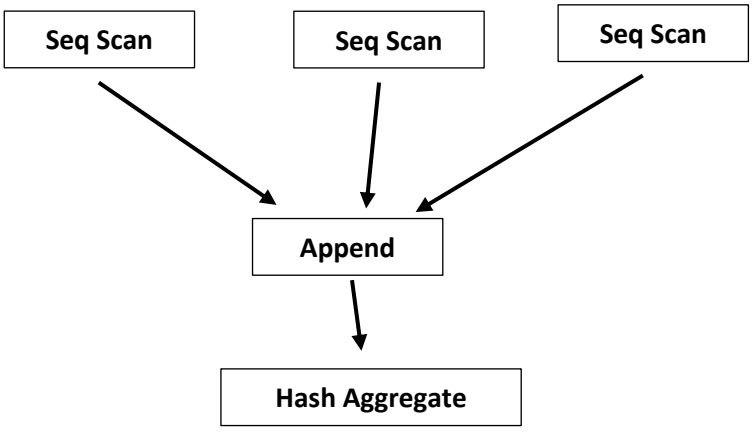
(next page)

```
EXPLAIN SELECT u.customer_id,
        sum(u.amount)
FROM
   ( SELECT *
    FROM payment_p2007_01
    UNION SELECT *
    FROM payment_p2007_02) u
WHERE u.payment_date <= '2007-02-01 00:00:00'::TIMESTAMP WITHOUT time ZONE
GROUP BY u.customer_id ;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------
-
 HashAggregate  (cost=127.26..129.76 rows=200 width=13)
   ->  HashAggregate  (cost=98.31..109.89 rows=1158 width=28)
         ->  Append  (cost=0.00..80.94 rows=1158 width=28)
               ->  Seq Scan on payment_p2007_01  (cost=0.00..23.46 rows=1157 width=28)
                     Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
               ->  Seq Scan on payment_p2007_02  (cost=0.00..45.90 rows=1 width=28)
                     Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
(7 rows)
```



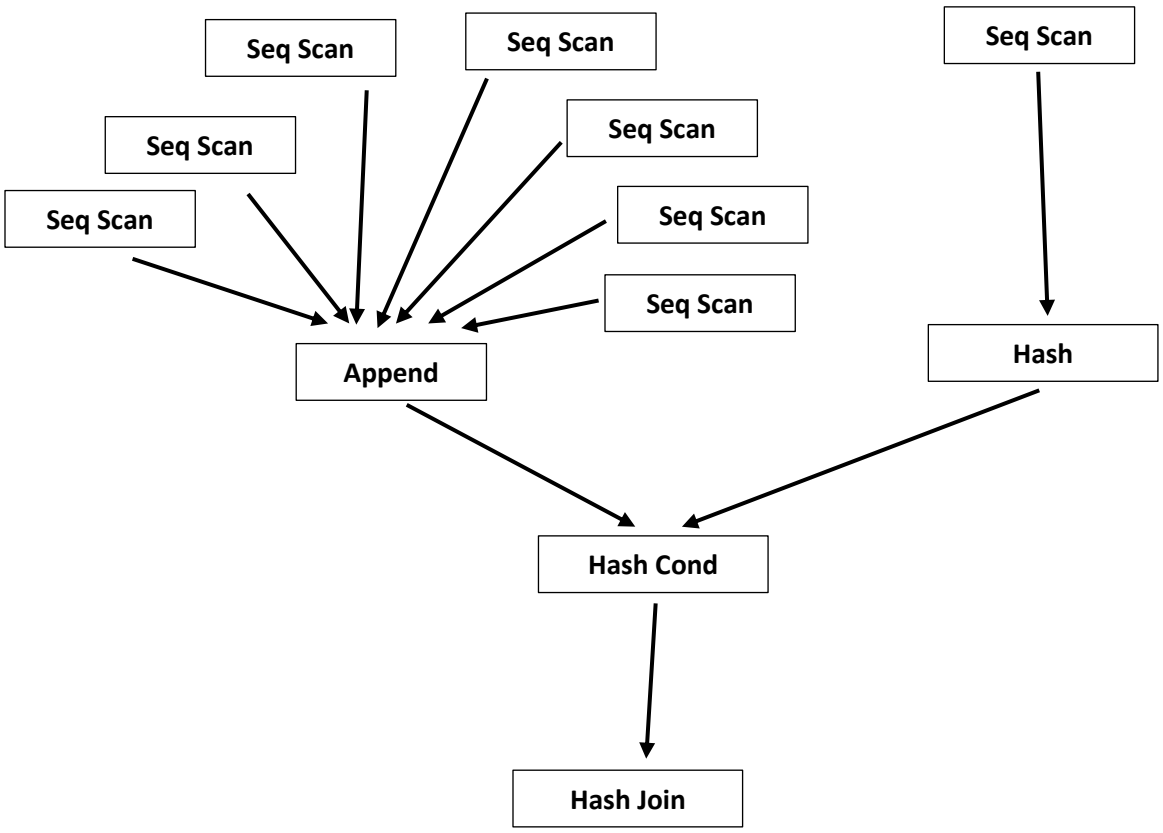
(next page)

```
EXPLAIN SELECT customer_id,
sum(amount)
FROM payment
WHERE payment_date <= '2007-02-01 00:00:00'::TIMESTAMP WITHOUT time ZONE
GROUP BY customer_id ;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------
 HashAggregate  (cost=103.99..106.49 rows=200 width=11)
   ->  Append  (cost=0.00..95.99 rows=1601 width=11)
         ->  Seq Scan on payment  (cost=0.00..26.62 rows=443 width=13)
               Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
         ->  Seq Scan on payment_p2007_01 payment  (cost=0.00..23.46 rows=1157 width=10)
               Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
         ->  Seq Scan on payment_p2007_02 payment  (cost=0.00..45.90 rows=1 width=10)
               Filter: (payment_date <= '2007-02-01 00:00:00'::timestamp without time zone)
(8 rows)
```



(next page)

```
EXPLAIN SELECT customer.customer_id,
        first_name,
        last_name,
        email,
        amount,
        payment_date
FROM customer
INNER JOIN payment ON payment.customer_id = customer.customer_id;
                                      QUERY PLAN
-------------------------------------------------------------------------------------------
 Hash Join  (cost=22.48..606.82 rows=18709 width=65)
   Hash Cond: (public.payment.customer_id = customer.customer_id)
   ->  Append  (cost=0.00..327.09 rows=18709 width=18)
         ->  Seq Scan on payment  (cost=0.00..23.30 rows=1330 width=21)
         ->  Seq Scan on payment_p2007_01 payment  (cost=0.00..20.57 rows=1157 width=18)
         ->  Seq Scan on payment_p2007_02 payment  (cost=0.00..40.12 rows=2312 width=18)
         ->  Seq Scan on payment_p2007_03 payment  (cost=0.00..98.44 rows=5644 width=18)
         ->  Seq Scan on payment_p2007_04 payment  (cost=0.00..117.54 rows=6754 width=18)
         ->  Seq Scan on payment_p2007_05 payment  (cost=0.00..3.82 rows=182 width=17)
         ->  Seq Scan on payment_p2007_06 payment  (cost=0.00..23.30 rows=1330 width=21)
   ->  Hash  (cost=14.99..14.99 rows=599 width=49)
         ->  Seq Scan on customer  (cost=0.00..14.99 rows=599 width=49)
(12 rows)
```



- SQL – review the basics using PostgreSQL