Report on Team Lab

Question A

- Create the database `Sailors`, `Boats` and `Reserves`. The code is provide below

```
create table Sailors (
    sid integer not null primary key,
    sname varchar(255),
    rating integer,
    age integer
);

create table Boats (
    bid integer not null primary key,
    bname varchar(255),
    color varchar(255)
);

create table Reserves(
    sid integer references Sailors(sid),
    bid integer references Boats(bid),
    day date primary key
);
```

Question B

- Populate the data base with huge numbers of data

```
-- retrieve from lab4 generating date with given range
DROP FUNCTION get_random_date(date,date);
CREATE OR REPLACE FUNCTION get_random_date(start_date date, end_date date) RETURNS Date AS
$BODY$
DECLARE  interval_days integer;  random_days integer;  random_date date;
BEGIN  interval_days := end_date -start_date;
random_days := random()*interval_days;
random_date := start_date + random_days;
RETURN random_date;END;
$BODY$  LANGUAGE plpgsql

-- Inserting 110 numbers of sailors
insert into sailors select generate_series(1,110),
(array['jack', 'Lubber','Bob','David'])[ceil(1 + random()*4)],
ceil(random()*(20-1)+10),ceil(random()*(30-1)+20);

-- Inserting 150 numbers of boats
insert into boats select generate_series(1,150),
(array['jack hammer','crowd','baki','hand'])[1+ random()*4],
(array['red','blue','green','orange'])[ceil(1+ random()*4)];

-- Inserting 110 numbers of reserves
insert into reserves select
generate_series(1,110),
```

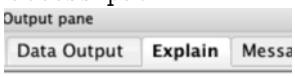
```
generate_series(1,110),
get_random_date(
    to_date('01 Jan 1980', 'DD Mon YYYY'),
    to_date('31 Dec 2017', 'DD Mon YYYY')
);
```

- For `access path of each query` is included in Question C and D

Query Questions

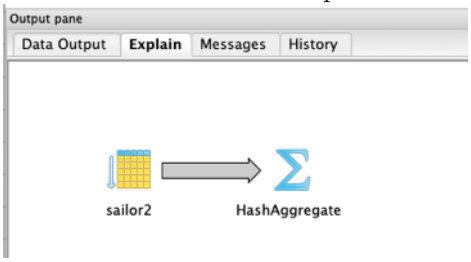
- Following include
  - `querys` from `question A`
  - `evaluation plan` from `quesstion B`
  - `access paths` from `question C`
  - `index improvement` from `question D`

1. Given question Find the names and ages of all sailors.
   - Query is `select s.age ,s.sname from sailors s;`
   - The `evaluation plan`
   - ```
     Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
     ```
   - The `access path`

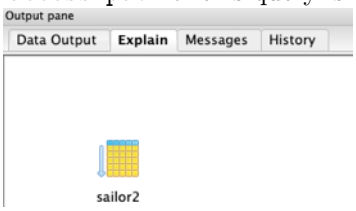
   - This would always require sequential search. This is already most optimized.
2. Given question Find the distinct names and ages of all sailors
   - Query is `select distinct s.age , s.sname from sailors s;`
   - The `evaluation plan`
   - ```
     Unique  (cost=5.83..6.93 rows=110 width=50)
        ->  Sort  (cost=5.83..6.10 rows=110 width=50)
              Sort Key: age, s.*, sname
        ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=50)
     ```
   - Instead optimizes on index we change the query to `group by` and change order of the `s.sname and s.age` since `s.sname` only have four types and `s.age` have 29 unique numbers.
   - Add index should make no performance differences

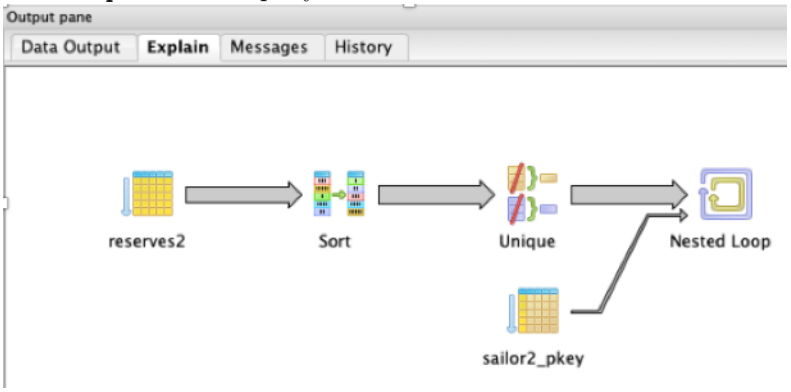
     

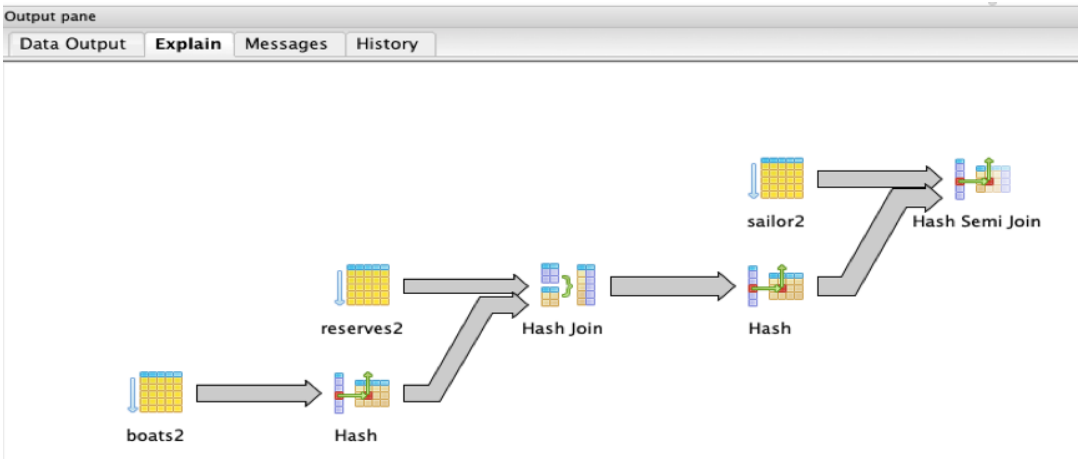
3. Given question Find all sailors with a rating above 7.
   - Query is `select * from sailors s where s.age > 7;`
   - The `evaluation plan`

- ```
  Seq Scan on sailors s  (cost=0.00..2.38 rows=110 width=17)
      Filter: (age > 7)
  ```
- Since is range comparison `tree index` can improve the optimization
  - `create index idx_sailors on sailors(age)`
- This is using the B+ tree property to retrieve range that is less than age.

```
Index Scan on sailors s  (cost=0.00..1.12 rows=110 width=17)
    Filter: (age > 7)
```

- The `access path` of this query is
  - 

4. Given question Find the names of sailors who have reserved boat number 103
   - Query is `sql select s.sname from sailors s join reserves r on s.sid = r.sid where r.bid = 103;`
   - The `evaluation plan`
   - ```
     Hash Join  (cost=2.39..4.91 rows=1 width=5)
         Hash Cond: (s.sid = r.sid)
         ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
         ->  Hash  (cost=2.38..2.38 rows=1 width=4)
               ->  Seq Scan on reserves r  (cost=0.00..2.38 rows=1 width=4)
                   Filter: (bid = 103)
     ```
   - According to question 2, the size of `sailors` and `reserves` are the same, the ordering doesn't matters
   - ```
     create index bid_reserves on reserves using hash(bid);
     create index sid_reserves on reserves using hash(sid)
     ```
     - Creating `Hash Index` on `reserves.sid` and `reserves.bid` for each sailors we do a **linear search** on each `reserves`. Additionaly, each reserves we also do a **linear search** with `bid==103`
   - ```
     Hash Join  (cost=2.39..4.91 rows=1 width=5)
         Hash Cond: (s.sid = r.sid)
         ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
         ->  Hash  (cost=2.38..2.38 rows=1 width=4)
               ->  Seq Scan on reserves r  (cost=0.00..2.38 rows=1 width=4)
                   Filter: (bid = 103)
     ```
   - The `access path` of this query is
     - 

5. Given question Find the names of sailors who have reserved a red boat
   - Query is
   - ```
     select s.sname from sailors s , reserves r , boats b
         where s.sid  = r.sid
         and  r.bid = b.bid
         and b.color = 'red';
     ```
   - The evaluation plan
   - ```
     Nested Loop  (cost=3.03..5.56 rows=1 width=5)
         ->  Hash Join  (cost=2.89..5.29 rows=1 width=4)
                 Hash Cond: (r.bid = b.bid)
                 ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=8)
                 ->  Hash  (cost=2.88..2.88 rows=1 width=4)
                     ->  Seq Scan on boats b  (cost=0.00..2.88 rows=1 width=4)
                             Filter: ((color)::text = 'red'::text)
         ->  Index Scan using sailors_pkey on sailors s  (cost=0.14..0.27 rows=1 width=9)
                 Index Cond: (sid = r.sid)
     ```
   - ```
     create index bid_reserves on reserves using hash(bid);
     create index sid_reserves on reserves using hash(sid);
     create index colors_boat on boats using hash(color);
     ```
     - Creating each index on reserves's foreign keys can improve looping from each red boats against each sailors and reserves.
     - Since boats has more sizes. We filtered by `red` boat then for each boat hash match against the reserves and reserves match against sailors
   - ```
     Nested Loop  (cost=3.03..5.56 rows=1 width=5)
     ->  Hash Join  (cost=2.89..5.29 rows=1 width=4)
             Hash Cond: (r.bid = b.bid)
             ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=8)
             ->  Hash  (cost=2.88..2.88 rows=1 width=4)
                 ->  Seq Scan on boats b  (cost=0.00..2.88 rows=1 width=4)
                         Filter: ((color)::text = 'red'::text)
     ->  Index Scan using sailors_pkey on sailors s  (cost=0.14..0.27 rows=1 width=9)
             Index Cond: (sid = r.sid)
     ```
   - The access path of this query is
     

6. Given question Find the colors of boats reserved by Lubber
   - Query is

```
select b.color from sailors s , reserves r , boats b
  where s.sid = r.sid and s.sname = 'Lubber';
```

- The evaluation plan

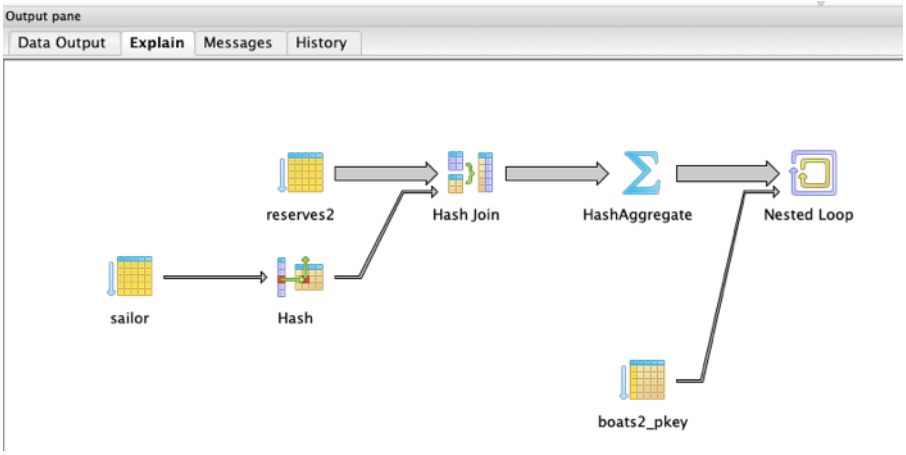
```
Nested Loop  (cost=2.69..54.52 rows=3750 width=6)
->  Seq Scan on boats b  (cost=0.00..2.50 rows=150 width=6)
->  Materialize  (cost=2.69..5.21 rows=25 width=0)
        ->  Hash Join  (cost=2.69..5.09 rows=25 width=0)
            Hash Cond: (r.sid = s.sid)
            ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=4)
            ->  Hash  (cost=2.38..2.38 rows=25 width=4)
                    ->  Seq Scan on sailors s  (cost=0.00..2.38 rows=25 width=4)
                        Filter: ((sname)::text = 'Lubber'::text)
```

```
create index bid_reserves on reserves using hash(bid);
create index sid_reserves on reserves using hash(sid);
create index name_sailors on sailors using hash(sname);
```

- two lines of `bid_reserves` and `sid_reserves` are for nested loop between three tables. Since the differences between previous question is filtering on `Lubber` sailors table. Therefore we used `name_sailors` index on sname.

```
Nested Loop  (cost=2.69..54.52 rows=3750 width=6)
->  Seq Scan on boats b  (cost=0.00..2.50 rows=150 width=6)
->  Materialize  (cost=2.69..5.21 rows=25 width=0)
        ->  Hash Join  (cost=2.69..5.09 rows=25 width=0)
            Hash Cond: (r.sid = s.sid)
            ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=4)
            ->  Hash  (cost=2.38..2.38 rows=25 width=4)
                    ->  Seq Scan on sailors s  (cost=0.00..2.38 rows=25 width=4)
                        Filter: ((sname)::text = 'Lubber'::text
```

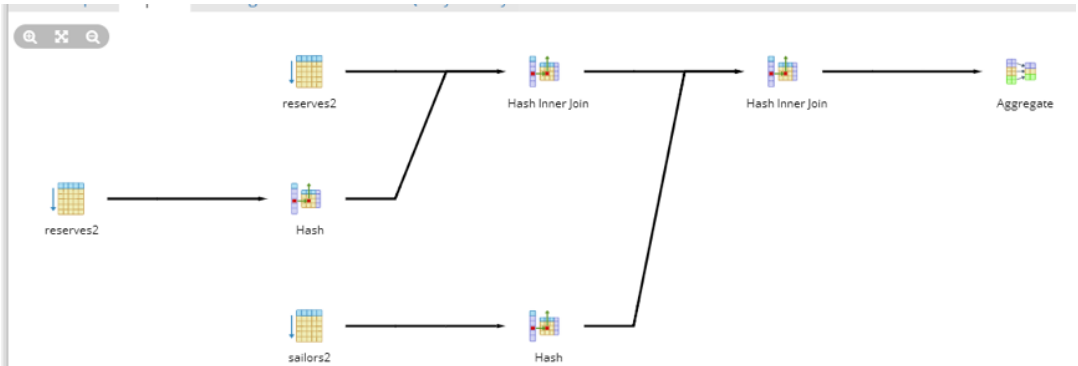
- The access path



7. Given question Find the names of sailors who have reserved at least one boat
   - Query is `select s.sname from sailors s , reserves r where s.sid = r.sid;`
   - The evaluation plan

```
Hash Join  (cost=3.48..5.87 rows=110 width=5)
   Hash Cond: (r.sid = s.sid)
   ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=4)
   ->  Hash  (cost=2.10..2.10 rows=110 width=9)
           ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
```

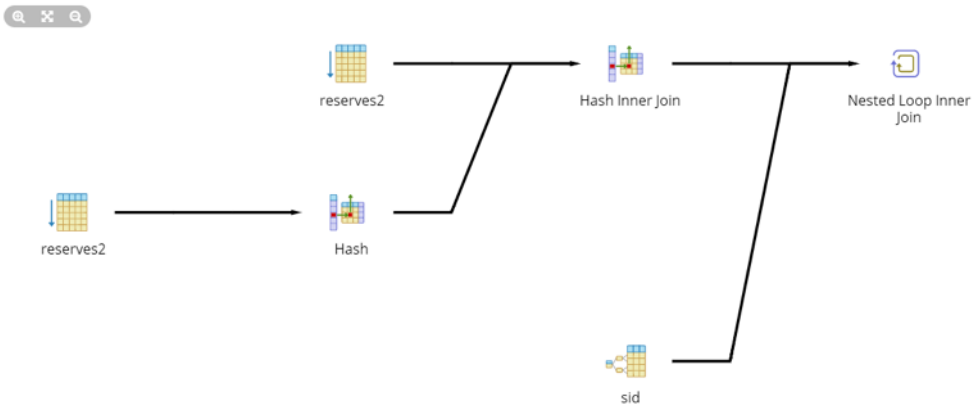
   - The access path is following



–

- Since both sizes of the table are the same. There the aranagement of the loop wouldn't influences the result.
- Creating index on joining would improve the performance
- ```
  create index sid_reserves on reserves using hash(sid);
  ```
- And performances shown below
- ```
  Hash Join  (cost=3.48..5.87 rows=110 width=5)
     Hash Cond: (r.sid = s.sid)
     ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=4)
     ->  Hash  (cost=2.10..2.10 rows=110 width=9)
           ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
  ```

8. Given question
   - Query is `sql select s.sid, rating +1 as increments from sailors s , reserves r1 , reserves r2 where s.sid = r1.sid and s.sid = r2.sid and r1.day = r2.day and r1.bid != r2.bid;`
   - The evaluation plan
   - ```
     Nested Loop  (cost=3.89..6.70 rows=1 width=8)
         ->  Hash Join  (cost=3.75..6.43 rows=1 width=8)
               Hash Cond: ((r1.sid = r2.sid) AND (r1.day = r2.day))
               Join Filter: (r1.bid <> r2.bid)
               ->  Seq Scan on reserves r1  (cost=0.00..2.10 rows=110 width=12)
               ->  Hash  (cost=2.10..2.10 rows=110 width=12)
                     ->  Seq Scan on reserves r2  (cost=0.00..2.10 rows=110 width=12)
         ->  Index Scan using sailors_pkey on sailors s  (cost=0.14..0.27 rows=1 width=8)
               Index Cond: (sid = r1.sid)
     ```
     ```



- The `access path`
- We would add `hash index` on `bid` , `day` , and `sailors.rating`. Since we evaluate the query with dismatch with its day.
- the inexs is

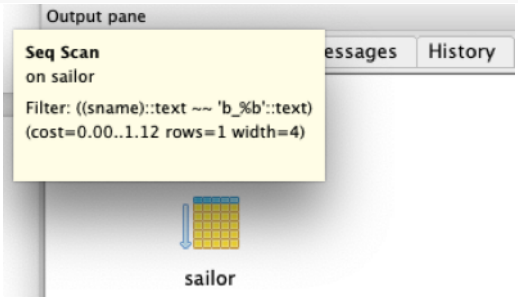
```
create index sid_reserves on reserves using hash(sid);
create index rating_sailors on sailors using hash(rating);
```

- This performance is

```
Nested Loop  (cost=3.89..6.70 rows=1 width=8)
   ->  Hash Join  (cost=3.75..6.43 rows=1 width=8)
         Hash Cond: ((r1.sid = r2.sid) AND (r1.day = r2.day))
         Join Filter: (r1.bid <> r2.bid)
         ->  Seq Scan on reserves r1  (cost=0.00..2.10 rows=110 width=12)
         ->  Hash  (cost=2.10..2.10 rows=110 width=12)
               ->  Seq Scan on reserves r2  (cost=0.00..2.10 rows=110 width=12)
   ->  Index Scan using i_sailor on sailors s  (cost=0.14..0.27 rows=1 width=8)
         Index Cond: (sid = r1.sid)
```

9. Given question Find the ages of sailors whose name begins and ends with B and has at least three characters
   - Query is `select s.age from sailors s where s.sname like '%b%' and length(s.sname) > 3;`
   - The `evaluation plan`

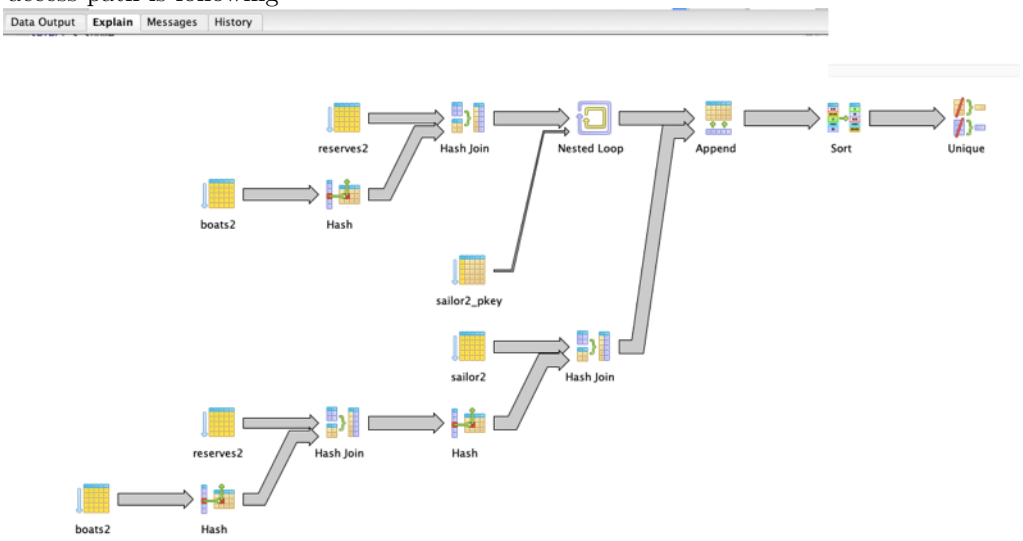
```
Seq Scan on sailors s  (cost=0.00..2.93 rows=19 width=4)
   Filter: (((sname)::text ~~ '%b%'::text) AND (length((sname)::text) > 3))
```



   - The `access path`
   - Take on the query on filtering length of the `sname` sequenctial scan is faster.
   - In comparison adding `create index len_sname on sailors(length(sname));` only gives following performance

```
Index Scan using len_sname on sailors s  (cost=0.14..8.88 rows=19 width=4)
   Index Cond: (length((sname)::text) > 3)
   Filter: ((sname)::text ~~ '%b%'::text)
```

10. Given question Find the names of sailors who have reserved a red or a green boat.
    - Query is `sql select s.sname from sailors s join reserves r on s.sid = r.sid join boats b on r.bid = b.bid where b.color = 'red' or b.color = 'green';`
    - The evaluation plan
    -
```
Hash Join  (cost=6.39..9.15 rows=25 width=5)
   Hash Cond: (s.sid = r.sid)
   ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
   ->  Hash  (cost=6.07..6.07 rows=25 width=4)
         ->  Hash Join  (cost=3.67..6.07 rows=25 width=4)
             Hash Cond: (r.bid = b.bid)
             ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=8)
             ->  Hash  (cost=3.25..3.25 rows=34 width=4)
                   ->  Seq Scan on boats b  (cost=0.00..3.25 rows=34 width=4)
                       Filter: (((color)::text = 'red'::text) OR ((color)::text = 'green
```
    - The access path is following



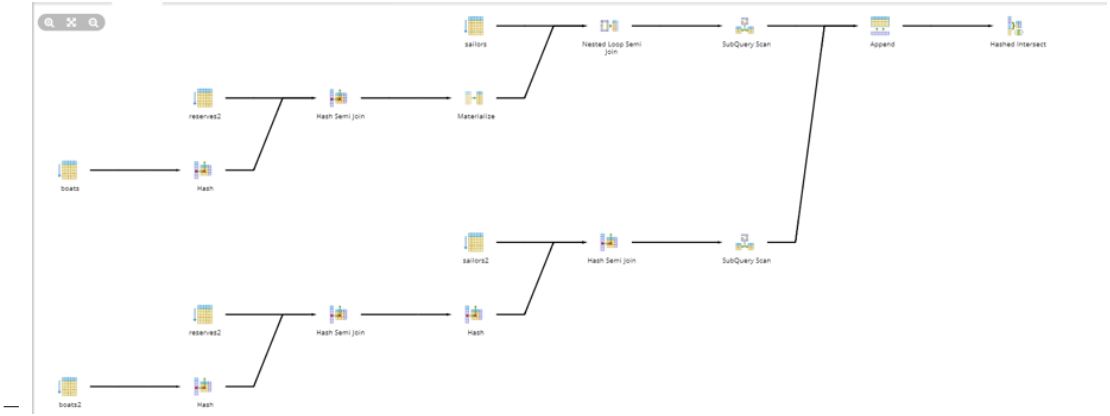
    - Creating `hash indexes` on `sid` and `bid` for joining. Inclusivly, adding `hash index` on boat's color.
    -
```
create index sid_reserves on reserves using hash(sid);
create index bid_reserves on reserves using hash(bid);
create index color_boats on boats using hash(color);
```
    - We do a Sequential Scan on boat's color filtering on `red` and `green`.
    -
```
Hash Join  (cost=6.39..9.15 rows=25 width=5)
   Hash Cond: (s.sid = r.sid)
   ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
   ->  Hash  (cost=6.07..6.07 rows=25 width=4)
         ->  Hash Join  (cost=3.67..6.07 rows=25 width=4)
             Hash Cond: (r.bid = b.bid)
             ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=8)
             ->  Hash  (cost=3.25..3.25 rows=34 width=4)
                   ->  Seq Scan on boats b  (cost=0.00..3.25 rows=34 width=4)
                       Filter: (((color)::text = 'red'::text) OR ((color)::text = 'green
```
11. Given question Find the names of sailors who have reserved both a red and a green boat.
    - Query is `sql select s.sname from sailors s , reserves r , boats b where s.sid = r.sid and ( (b.color = 'red' and`

r.bid = b.bid) or (b.color = 'green' and r.bid = b.bid)
);

- The evaluation plan

```
 Hash Join  (cost=6.39..9.15 rows=25 width=5)
      Hash Cond: (s.sid = r.sid)
      ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
      ->  Hash  (cost=6.07..6.07 rows=25 width=4)
              ->  Hash Join  (cost=3.67..6.07 rows=25 width=4)
                  Hash Cond: (r.bid = b.bid)
                  ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=8)
                  ->  Hash  (cost=3.25..3.25 rows=34 width=4)
                          ->  Seq Scan on boats b  (cost=0.00..3.25 rows=34 width=4)
                              Filter: (((color)::text = 'red'::text) OR ((color)::text = 'green
```

- The access path is following



- Since there is multiple conditional statement. We will rearange into left joined clauses and CNF. the query became:

```
  select s.sname from sailors s , reserves r , boats b
      where s.sid = r.sid
              and r.bid = b.bid
              and (b.color = 'red'
                      or b.color = 'green');
```

- Addding tree index on boat.color and hash index on reserves.sid and sailors.sid for joining sql create index sid_reserves on reserves using hash(sid); create index bid_reserves on reserves using hash(bid); create index color_boats on boats using hash(color);
- The performances

```
  Hash Join  (cost=6.39..9.15 rows=25 width=5)
      Hash Cond: (s.sid = r.sid)
      ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
      ->  Hash  (cost=6.07..6.07 rows=25 width=4)
              ->  Hash Join  (cost=3.67..6.07 rows=25 width=4)
                  Hash Cond: (r.bid = b.bid)
                  ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=8)
                  ->  Hash  (cost=3.25..3.25 rows=34 width=4)
                          ->  Seq Scan on boats b  (cost=0.00..3.25 rows=34 width=4)
                              Filter: (((color)::text = 'red'::text) OR ((color)::text = 'green
```

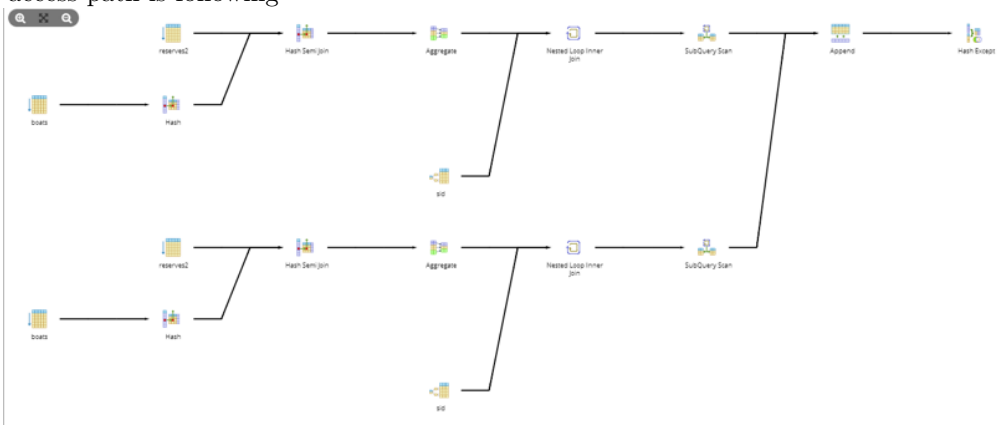
12. Given question Find the sids of all sailors who have reserved red boats but not green boats.
    - Query is sql select s.sname from sailors s , reserves r , boats b where s.sid = r.sid and r.bid = b.bid and b.color = 'red'

and not (b.color = 'green');

- The evaluation plan

```
Nested Loop  (cost=3.41..5.93 rows=1 width=5)
   ->  Hash Join  (cost=3.26..5.66 rows=1 width=4)
         Hash Cond: (r.bid = b.bid)
         ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=8)
         ->  Hash  (cost=3.25..3.25 rows=1 width=4)
             ->  Seq Scan on boats b  (cost=0.00..3.25 rows=1 width=4)
                   Filter: (((color)::text <> 'green'::text) AND ((color)::text = 'red'
   ->  Index Scan using sailors_pkey on sailors s  (cost=0.14..0.27 rows=1 width=9)
         Index Cond: (sid = r.sid)
```

- The access path is following



–

- Create `hash index` in boat's color and reserves both `sid` and `rid`, since reserves has less sizes than **boats** , we should do indexed nested loop with boats.
- Since boat's color is `red`, isn't green should also hold true
- So indexes we used is

```
create index sid_reserves on reserves using hash(sid);
create index bid_reserves on reserves using hash(bid);v
create index color_boat on boats using hash(color);
```

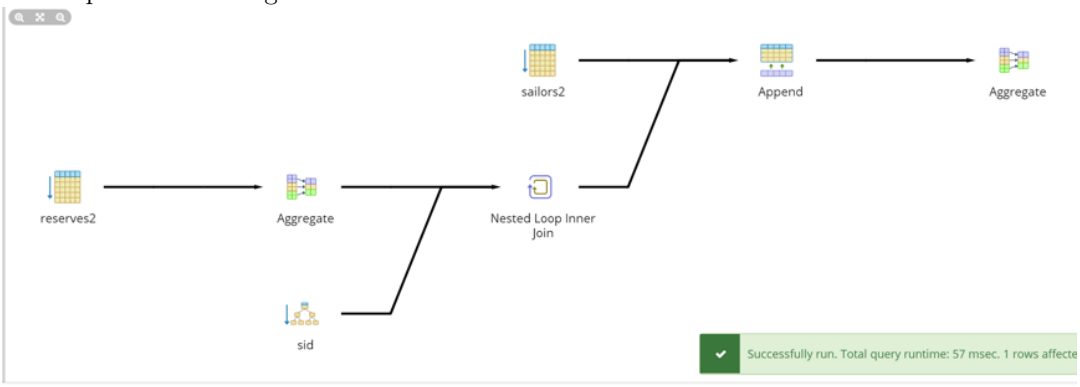
- We would also changed the query to

```
select s.sname from sailors s , reserves r , boats b
    where s.sid = r.sid
            and r.bid = b.bid
            and b.color = 'red'
```

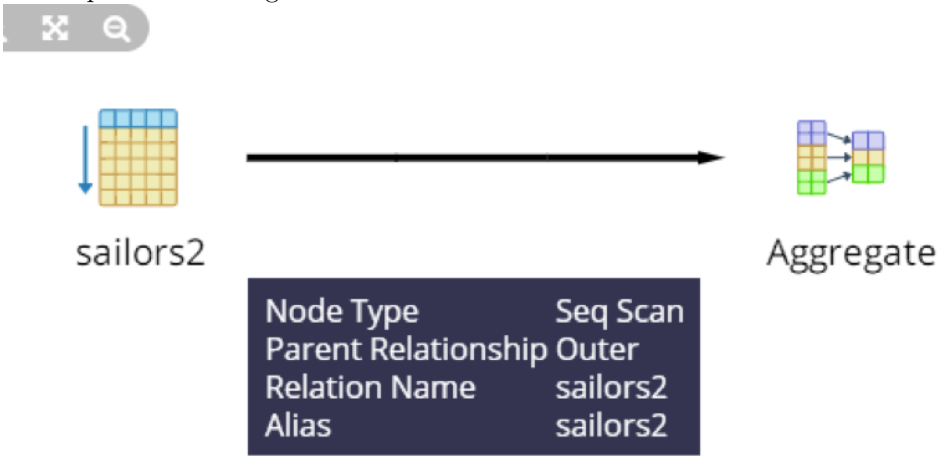
- The performance is

```
Nested Loop  (cost=3.03..5.56 rows=1 width=5)
   ->  Hash Join  (cost=2.89..5.29 rows=1 width=4)
         Hash Cond: (r.bid = b.bid)
         ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=8)
         ->  Hash  (cost=2.88..2.88 rows=1 width=4)
             ->  Seq Scan on boats b  (cost=0.00..2.88 rows=1 width=4)
                   Filter: ((color)::text = 'red'::text)
   ->  Index Scan using i_sailor on sailors s  (cost=0.14..0.27 rows=1 width=9)
         Index Cond: (sid = r.sid)
```

13. Given question Find all sids of sailors who have rating of 10 or reserved boat 104
    - Query is `sql  select s.sid from sailors s , reserves r  where s.sid = r.sid and (s.rating = 10 or r.bid = 124);`
    - The evaluation plan

- ```
  Nested Loop  (cost=0.14..37.37 rows=1 width=4)
    ->  Index Scan using i_sailor on sailors s  (cost=0.14..13.79 rows=110 width=8)
    ->  Index Scan using sid_reserves on reserves r  (cost=0.00..0.20 rows=1 width=8)
          Index Cond: (sid = s.sid)
          Filter: ((s.rating = 10) OR (bid = 124))
  ```
- The access path is following
  - 

- Create hash index on `sailors.rating` and `reserves.bid`. Since bid on reserves is buildin default.
- ```
  create index rating_sailors on sailors using hash(rating);
  ```
- Since the size of reserves and sailors is the same, we don't consider nested loop
- Performance
  - ```
    Hash Join  (cost=3.48..5.86 rows=1 width=4)
      Hash Cond: (r.sid = s.sid)
      Join Filter: ((s.rating = 10) OR (r.bid = 124))
      ->  Seq Scan on reserves r  (cost=0.00..2.10 rows=110 width=8)
      ->  Hash  (cost=2.10..2.10 rows=110 width=8)
            ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=8)
    ```

14. Given question Find the average age of all sailors.
    - Query is `select avg(s.age) from sailors s;`
    - The `evaluation plan`
    - ```
      Aggregate  (cost=2.38..2.39 rows=1 width=32)
        ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=4)
      ```
    - The access path is following
      - 

    - A index can be improvement is hash index

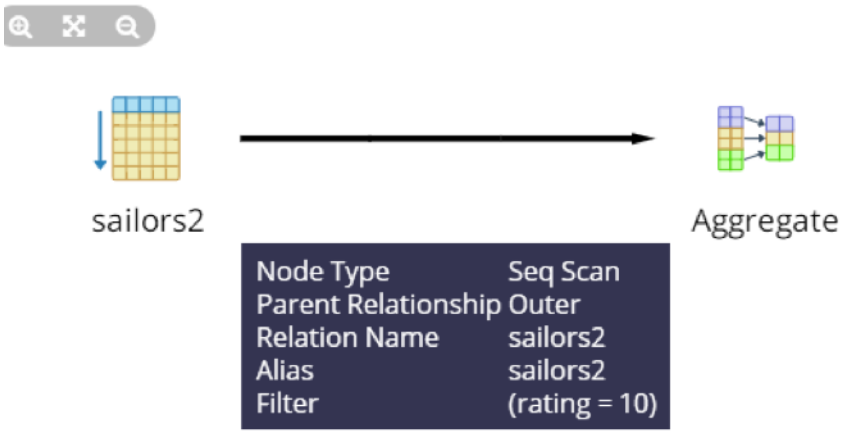
  - `create index sailors_age using hash(sailors.age)`
- Since `average` is an aggregate function, there is not thing more optimal than Seq Scan.
- The performance is

```
Aggregate  (cost=2.38..2.39 rows=1 width=32)
  ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=4)
```

15. Given question Find the average age of sailors with a rating of 10
    - Query is `select avg(s.age) from sailors s where s.rating = 10;`
    - The `evaluation plan`

```
Aggregate  (cost=2.38..2.39 rows=1 width=32)
  ->  Seq Scan on sailors s  (cost=0.00..2.38 rows=1 width=4)
        Filter: (rating = 10)
```

    - The access path is following



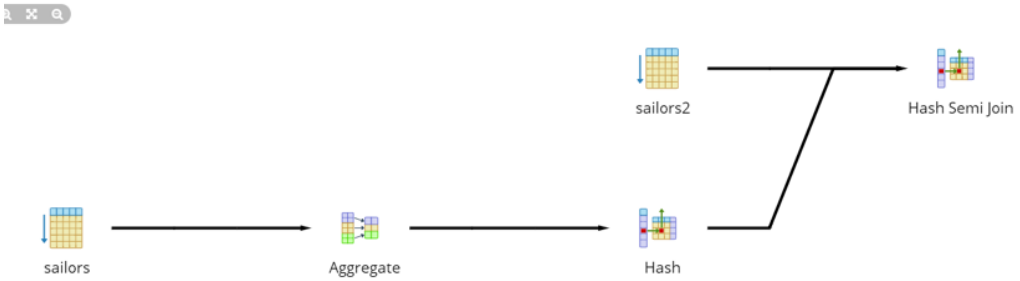
    -
    - Since we retrieve the equality on rating. We create a hash index in sailors's index we can improve the performance. On computing the average the performance is better in sequential scan.
    - `create index rating_sailors on sailors(rating)`

```
Aggregate  (cost=2.38..2.39 rows=1 width=32)
  ->  Seq Scan on sailors s  (cost=0.00..2.38 rows=1 width=4)
        Filter: (rating = 10)
```

16. Given question Find the name and age of the oldest sailor
    - Query is sql `select s.sname , s.age from sailors s where s.age in ( select max(s1.age) msa from sailors s1);`
    - The `evaluation plan`

```
Hash Join  (cost=0.30..2.73 rows=4 width=9)
  Hash Cond: (s.age = ($0))
  ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
  ->  Hash  (cost=0.29..0.29 rows=1 width=4)
        ->  Result  (cost=0.27..0.28 rows=1 width=4)
            InitPlan 1 (returns $0)
              ->  Limit  (cost=0.14..0.27 rows=1 width=4)
                    ->  Index Only Scan Backward using idx_sailor on sailors s1  (cost=0
                          Index Cond: (age IS NOT NULL)
```

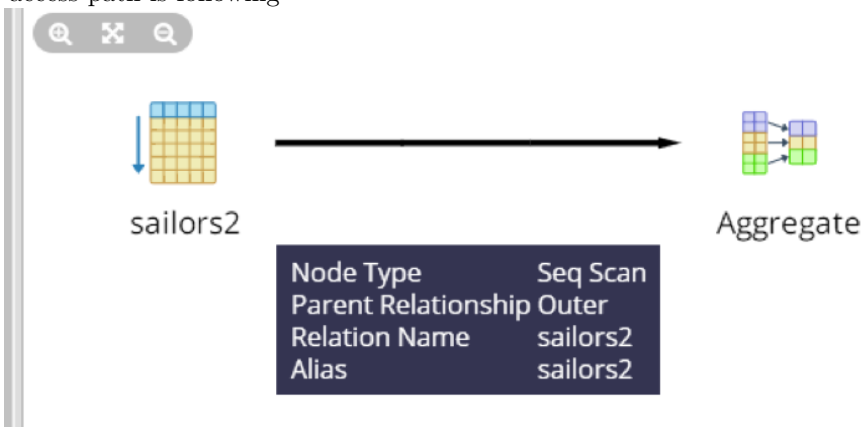
    - The access path is following



–

- Since there is nested query, we can re-represent the query into `group by .. having`. With additional hash index on age for faster retrieval on group by clasues. On matching we perform sequence scan
- ```
  select s.sname , s.age from sailors  s group by s.age , s.sname having s.age = max(s.age)
  ```
- The performance
- ```
  HashAggregate  (cost=2.92..3.21 rows=29 width=9)
  Group Key: age, sname
  Filter: (age = max(age))
  ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=9)
  ```

17. Given question Count the number of sailors
    - Query is `select count(s) from sailors s;`
    - The `evaluation plan`
    - ```
      Aggregate  (cost=2.38..2.38 rows=1 width=8)
        ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=41)
      ```
    - The access path is following

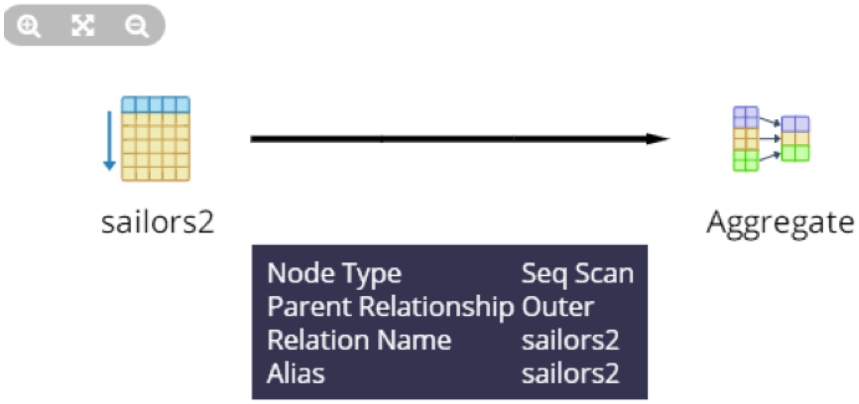


–

- We add no index on sailors. Since seuqential scan would be the most optimal. The following is the performance with index scanning.
- ```
  GroupAggregate  (cost=10000000005.83..10000000007.22 rows=29 width=9)
    Group Key: age, sname
    Filter: (age = max(age))
    ->  Sort  (cost=10000000005.83..10000000006.10 rows=110 width=9)
          Sort Key: age, sname
          ->  Seq Scan on sailors s  (cost=10000000000.00..10000000002.10 rows=110 width=9)
  ```

18. Given question Count the number of different sailor names.
    - Query is `select  count( distinct (s.sname)) from sailors s;`
    - The `evaluation plan`

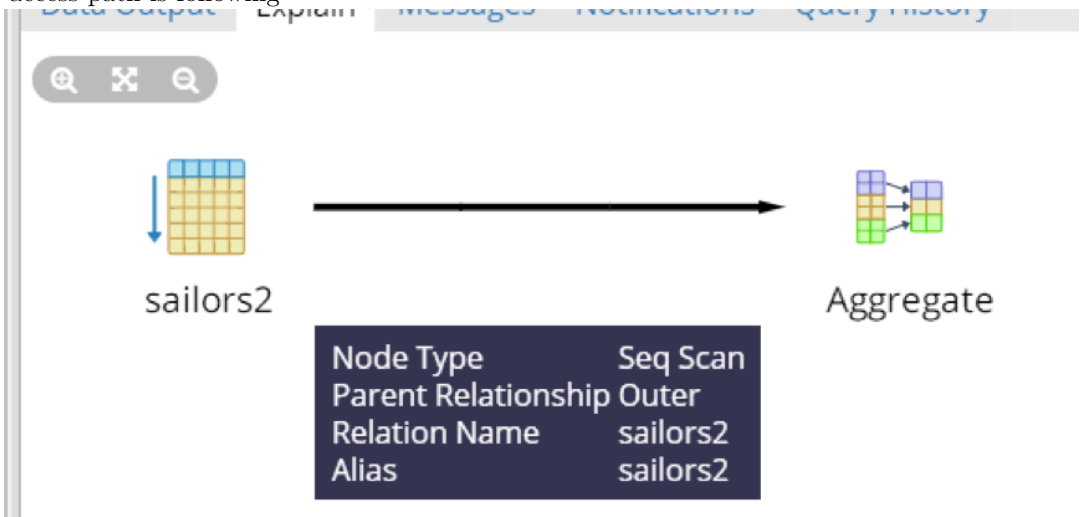
- ```
  Aggregate  (cost=2.38..2.38 rows=1 width=8)
    ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=5)
  ```
- The access path is following



–

- There might be no improvemtn done in adding index

19. Given question Find the age of the youngest sailor for each rating level.
    - Query is `sql select s.age, s.rating from sailors s where s.age in ( select min(s1.age) min_sa from sailors s1) order by s.rating;`
    - The `evaluation plan`
    - ```
      Sort  (cost=2.77..2.78 rows=4 width=8)
        Sort Key: s.rating
        ->  Hash Join  (cost=0.30..2.73 rows=4 width=8)
              Hash Cond: (s.age = ($0))
              ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=8)
              ->  Hash  (cost=0.29..0.29 rows=1 width=4)
                  ->  Result  (cost=0.27..0.28 rows=1 width=4)
                        InitPlan 1 (returns $0)
                        ->  Limit  (cost=0.14..0.27 rows=1 width=4)
                              ->  Index Only Scan using idx_sailor on sailors s1  (cost=0.
                                  Index Cond: (age IS NOT NULL)
      ```
    - The access path is following



–

- we redo the query into `group by` and `having` to minimizes the selection clause.
- The query is

```
select s.age, s.rating from sailors s
    group by s.age , s.rating having
        s.age = min(s.age)
    order by  s.rating
```

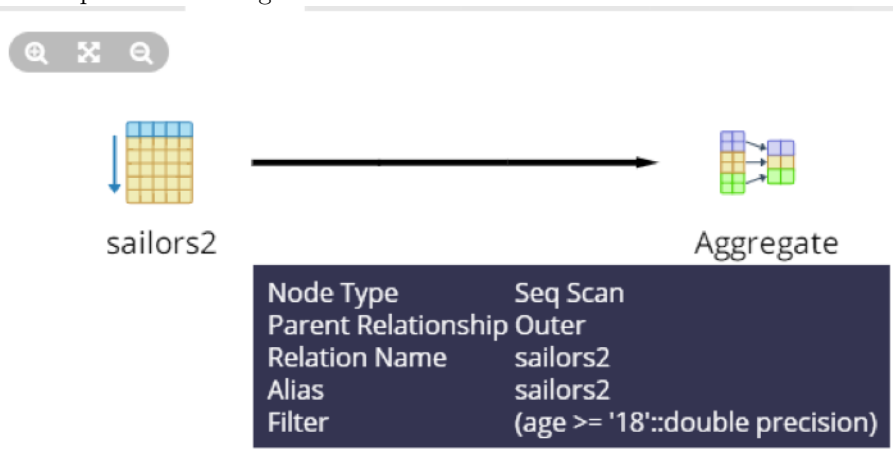
- Add index made no impact on the cost.
- The performance increase by remove additional limit hash join in th previous query.

```
Sort  (cost=3.92..3.99 rows=29 width=8)
   Sort Key: rating
   ->  HashAggregate  (cost=2.92..3.21 rows=29 width=8)
         Group Key: rating, age
         Filter: (age = min(age))
         ->  Seq Scan on sailors s  (cost=0.00..2.10 rows=110 width=8)
```

20. Given question Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 years old) for each rating level with at least two such sailors.
    - Query is `sql select s2.rating from ( select s.rating , min(s.age) as min_age from sailors s where s.age >= 18 group by s.rating having count(s.sid ) >= 2) as s2;`
    - The `evaluation plan`

```
Subquery Scan on s2  (cost=2.92..3.30 rows=19 width=4)
   ->  HashAggregate  (cost=2.92..3.11 rows=19 width=8)
         Group Key: s.rating
         Filter: (count(s.sid) >= 2)
         ->  Seq Scan on sailors s  (cost=0.00..2.38 rows=110 width=8)
             Filter: (age >= 18)
```

    - The access path is following



    –
    - Since the query is nest query. Such as subquery is un-needed.
    - Adding tree index on age would imrpove it.
      – `create index age_sailor on sailors(age)`
    - Replace the query into the following `sql select s.rating from sailors s where s.age >= 18 group by s.rating having count(s.sid) >= 2;`
    - The query would remove the nest subquery. The performance is `sql HashAggregate (cost=2.92..3.11 rows=19 width=4) Group Key: rating Filter: (count(sid) >= 2) -> Seq Scan on sailors s (cost=0.00..2.38 rows=110`

```
width=8)                    Filter: (age >= 18)
```