Homework #7
CS486
Prof. Charles Winstead
CRN: 40876

Jiacheng Zhao
Jordan Witte
Resheet Schultz
Due: 04-06-2019

Exercise 1

Consider a database with two tables:
`HeroAction(HAName, Hero, Group, PlayerDeck)`
`Player(PlayerId, PlayerName)`
`HeroAction.PlayerDeck` *is a foreign key referencing* `Player.PlayerId`*.*
Suppose the following command is executed:
`Player P FULL OUTER JOIN HeroAction HA ON (P.PlayerId=HA.PlayerDeck)`

A) *Consider computing this join using a sort-merge join algorithm. Explain how to adapt the sort-merge algorithm to produce the additional rows needed for the outer join.*

`FULL OUTER JOIN` should output all of the merged rows plus all of the unmerged rows (but the unmerged rows will have a bunch of `NULL`s).

The merge join algo works by taking two sorted stacks. If the top element on each stack should be merged, it merges and outputs them, otherwise it pops one element off (based on their values) and skips to the next one. Instead of skipping, it should do a merge with an empty element (to produce the `NULL`s) and output the result.

B) *Now consider computing this join using a simple hash-join algorithm when* `Player` *will fit in memory. That is, it will be the inner input, which is inserted into a hash table. Explain how to adapt the hash-join algorithm to produce the additional rows needed for the outer join.*

The hash join algo works by converting the smaller table into a hash table where the `ON` argument is the key. Then for each row in the bigger table, the algo hashes into the table and checks if there's data there. If there is, it merges the results. This has the same structure as part **B** and is thus fixed in the same way by merging with an empty element if the hash table turns up empty.

Afterwards we also need to merge unmatched rows in the table with null rows. To do this, a second hashlist should be maintained. Each time a row in the table is matched, its sister in the list should be removed. Afterwards, each row in the list can be merged with a null row and outputted.fix

Exercise 2

A) *Determine the min and max salary values in the agent table, and the number of rows in that table.*

I retrieved the salary data with `SELECT MIN(A.salary) FROM agent A` and `SELECT MAX(A.salary) FROM age`
The min turned out to be 50008 and the max was 366962. The table size I got with `SELECT COUNT(A.salary) FR`
which ended up being 662.

B) *Give an estimate for the number of rows in agent with salary < 75000, assuming a uniform distribution of salaries between min and max salary. Explain how you derived your estimate.*

First I found the salary range by subtracting: $366962 - 50008 = 316954$. Then I similarly subtracted the target salary: $75000 - 50008 = 24992$. Then I simply divided the results: $24992/316954 = 0.0789$ and multiplied by the total number of rows $0.0789 * 662 = 52$ or about 52.

C) *Find the 25th, 50th and 75th percentile values for salaries in the agent table.*

I used the following command:

```
SELECT DISTINCT
percentile_cont (0.25) WITHIN GROUP
(ORDER BY A.salary ASC) as percentile_25,
percentile_cont (0.50) WITHIN GROUP
(ORDER BY A.salary ASC) as percentile_50,
percentile_cont (0.75) WITHIN GROUP
(ORDER BY A.salary ASC) as percentile_75
FROM agent A;
```

And the answer is:

percentile_25	percentile_50	percentile_75
54802	58627	89642.5

D) *Give an estimate of the number of rows in agent with salary < 75000, assuming in a uniform distribution in each quartile determined in c. Explain how you derived your estimate.*

First I rescaled the salary and percentile points, then combined them into a single value of dollars per percentile: $89642 - 58627 = 31015$ and $75 - 50 = 25$, which results in $\frac{31025}{25} = 1241$. Then I found how many dollars less we needed: $89642 - 75000 = 14642$. Divide by the conversion factor: $\frac{14642}{1241} \approx 12$. So we need 12 points less, which means that $75 - 12 = 63$ percentile, or 63% of the data is below the target salary. Since there's 662 rows, the answer is $662 * 0.63 \approx 417$.

E) *How many rows in agent actually have salary < 75000?*

This can be found with a simple command:
`SELECT COUNT(A.salary) FROM agent A WHERE A.salary<75000`
The answer is 427.

Exercise 3

A) *State in English what this query returns. Dont just paraphrase the SQL into words.*

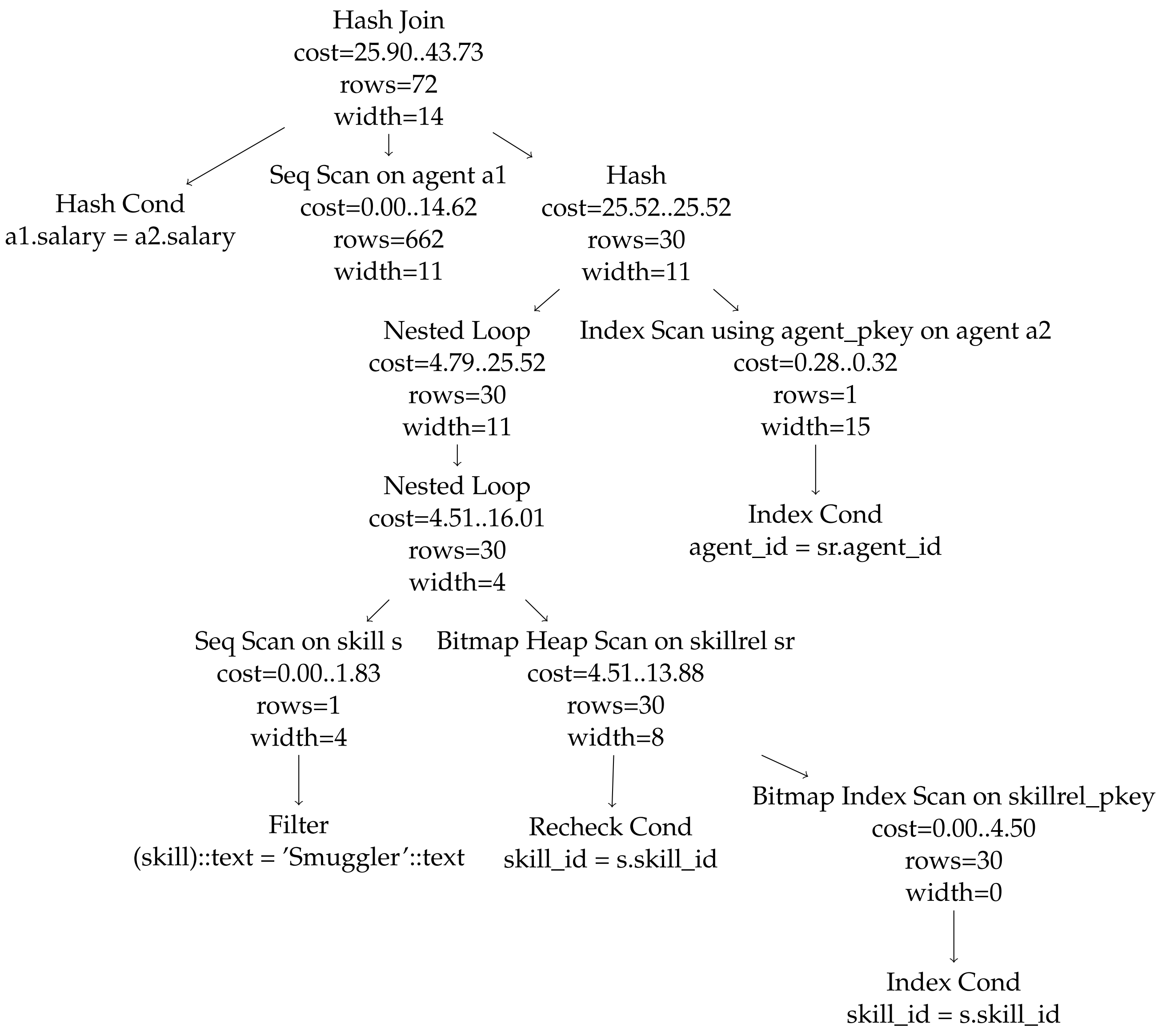
```
SELECT A2.last, A1.last
FROM agent A1,
agent A2 NATURAL JOIN skillrel SR NATURAL JOIN Skill S
WHERE A1.salary = A2.salary AND S.Skill = 'Smuggler';
```

List agents pairs who are paid the same amount where one of them is a smuggler.

B) *Draw the query tree for the SQL query in 3a.*

$$\begin{tikzcd}
 & \pi_{\text{A2.last,A1.last}} \arrow[d] & \\
 & \sigma_{A1.salary=A2.salary} \arrow[d] & \\
 & \sigma_{S.Skill='Smuggler'} \arrow[d] & \\
 & \bowtie_{agent\_id=agent\_id} \arrow[dl] \arrow[dr] & \\
\text{agent} & & \text{skillrel}
\end{tikzcd}$$

C) *Draw the query plan that Postgres chooses (which you can obtain with the* `EXPLAIN` *command) for the SQL query from 3a. Explain how Postgres executed the query and why.*

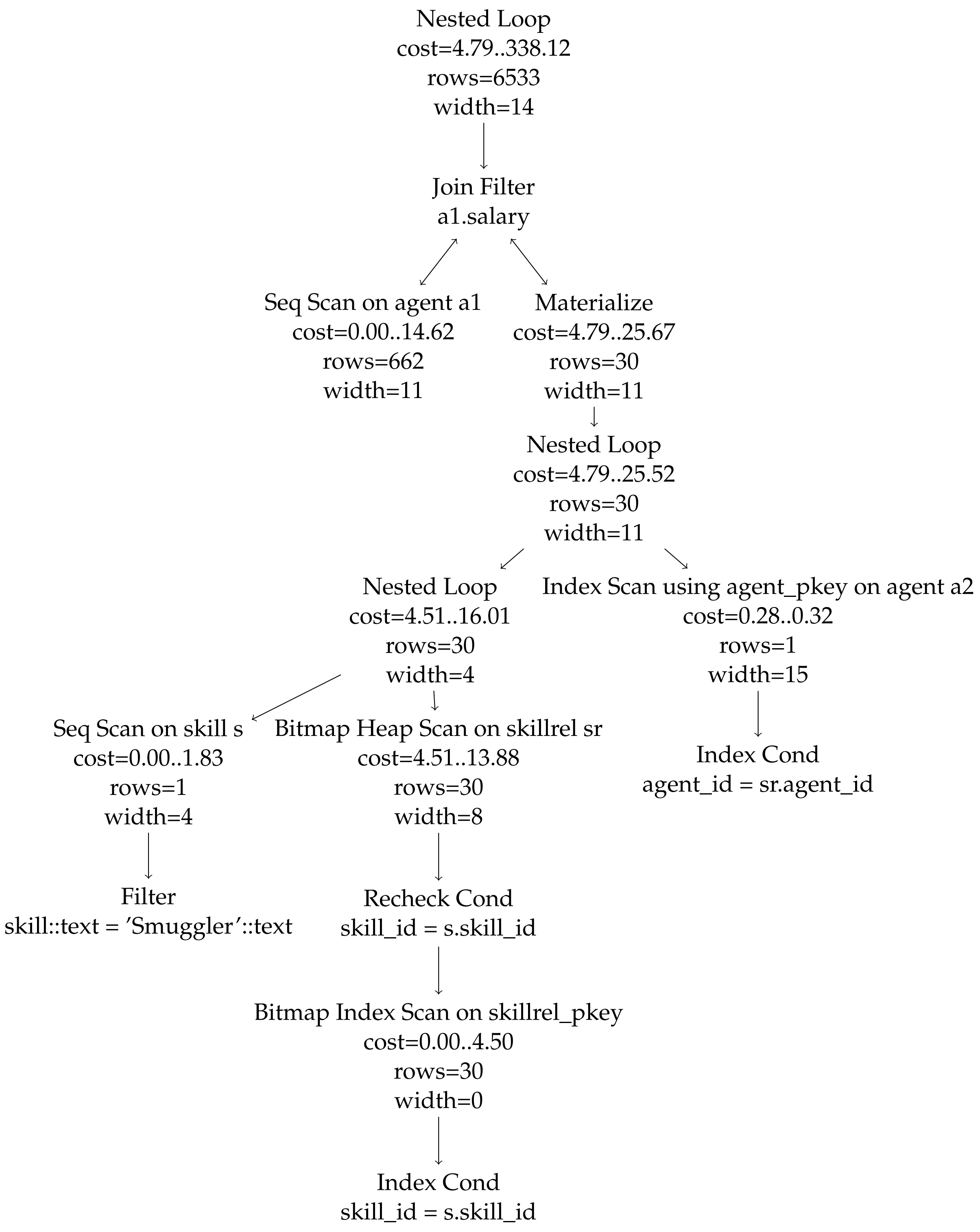

Hash Join cost=25.90..43.73 rows=72 width=14
Hash Cond a1.salary = a2.salary
Seq Scan on agent a1 cost=0.00..14.62 rows=662 width=11
Hash cost=25.52..25.52 rows=30 width=11
Nested Loop cost=4.79..25.52 rows=30 width=11
Index Scan using agent_pkey on agent a2 cost=0.28..0.32 rows=1 width=15
Nested Loop cost=4.51..16.01 rows=30 width=4
Index Cond agent_id = sr.agent_id
Seq Scan on skill s cost=0.00..1.83 rows=1 width=4
Bitmap Heap Scan on skillrel sr cost=4.51..13.88 rows=30 width=8
Filter (skill)::text = ′Smuggler′::text
Recheck Cond skill_id = s.skill_id
Bitmap Index Scan on skillrel_pkey cost=0.00..4.50 rows=30 width=0
Index Cond skill_id = s.skill_id

Summarizing the query tree from approximately the bottom up:

1. The `skill` table is filtered to only include smugglers and is joined with `skillrel` using a heap-scan.
2. This is combined in the inner nested loop with skillrel, filtering on `skill_id`. Bitmap scan the skillrel table to look up matching `skill_ids`.
3. Then, this is combined in the outer nested loop with `agent A2`. Index scall possible because the `agent` table has an index on its primary key.
4. This is combined with `A1` using a hash to match salary values, since no index exists for the salary column.

D) *For the following query, draw the query plan that Postgres chooses. Explain how Postgres executed the query and why.*

```
SELECT A2.last , A1.last
FROM agent A1,
agent A2 NATURAL JOIN skillrel SR NATURAL JOIN Skill S
WHERE A1.salary < A2.salary AND S.Skill = 'Smuggler';
```

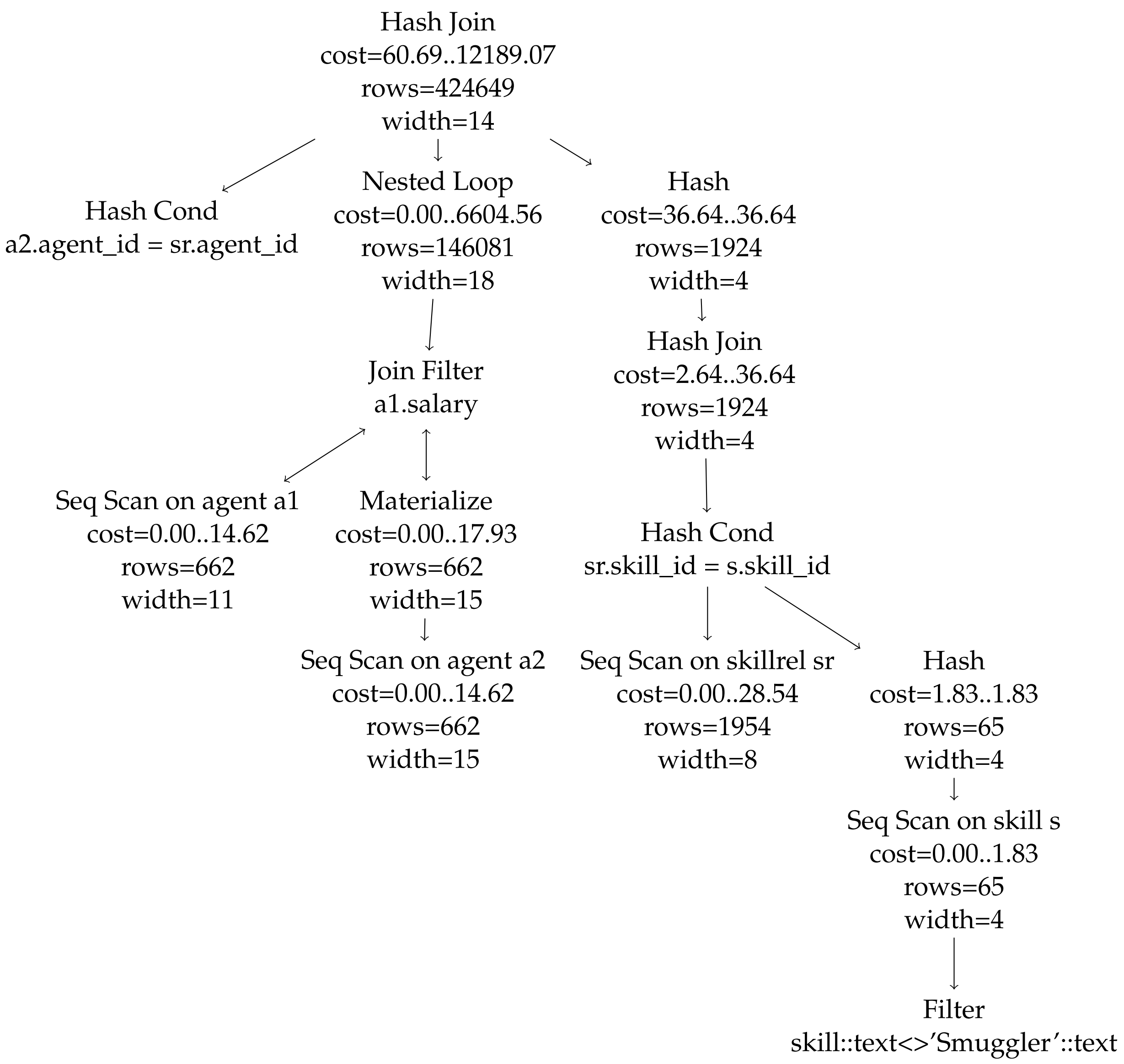

Nested Loop cost=4.79..338.12 rows=6533 width=14
Join Filter a1.salary
Seq Scan on agent a1 cost=0.00..14.62 rows=662 width=11
Materialize cost=4.79..25.67 rows=30 width=11
Nested Loop cost=4.79..25.52 rows=30 width=11
Nested Loop cost=4.51..16.01 rows=30 width=4
Index Scan using agent_pkey on agent a2 cost=0.28..0.32 rows=1 width=15
Index Cond agent_id = sr.agent_id
Seq Scan on skill s cost=0.00..1.83 rows=1 width=4
Filter skill::text = 'Smuggler'::text
Bitmap Heap Scan on skillrel sr cost=4.51..13.88 rows=30 width=8
Recheck Cond skill_id = s.skill_id
Bitmap Index Scan on skillrel_pkey cost=0.00..4.50 rows=30 width=0
Index Cond skill_id = s.skill_id

Very similar to part **3C**, except that:

1. An outer nested loop joins the two main tables because the filter operation cant be performed with a hash. The < operation requires that the join examine every pair of entries.
2. Materialize operation required because we iterate on the table using a nested loop, meaning the table cant be constructed and just hashed or something, but must be examined in full by the nested loop filter.

E) *For the following query, draw the query plan that Postgres chooses. Explain how Postgres executed the query and why.*

```sql
SELECT A2.last , A1.last
FROM agent A1,
agent A2 NATURAL JOIN skillrel SR NATURAL JOIN Skill S
WHERE A1.salary < A2.salary AND S.Skill <> 'Smuggler';
```



Hash Join cost=60.69..12189.07 rows=424649 width=14
Hash Cond a2.agent_id = sr.agent_id
Nested Loop cost=0.00..6604.56 rows=146081 width=18
Hash cost=36.64..36.64 rows=1924 width=4
Join Filter a1.salary
Hash Join cost=2.64..36.64 rows=1924 width=4
Seq Scan on agent a1 cost=0.00..14.62 rows=662 width=11
Materialize cost=0.00..17.93 rows=662 width=15
Hash Cond sr.skill_id = s.skill_id
Seq Scan on agent a2 cost=0.00..14.62 rows=662 width=15
Seq Scan on skillrel sr cost=0.00..28.54 rows=1954 width=8
Hash cost=1.83..1.83 rows=65 width=4
Seq Scan on skill s cost=0.00..1.83 rows=65 width=4
Filter skill::text<>'Smuggler'::text

A few major restructure differences compared to parts **3C** and **3D** are necessary:

1. `skill <> 'Smuggler'` results in a large (`skill NATURAL JOIN skillrel` table), so this is constructed with a hash join using the skills-not-matching smuggler as the inner (hashed) table.
2. `A1 JOIN A2` also requires a nested loop for the same reason as before; the < comparison requires an all-pairs search.
3. These tables are then hash-joined because rows are matched on the `a2.agent_id = sr.agent_id`.