Polytechnic University of Turin

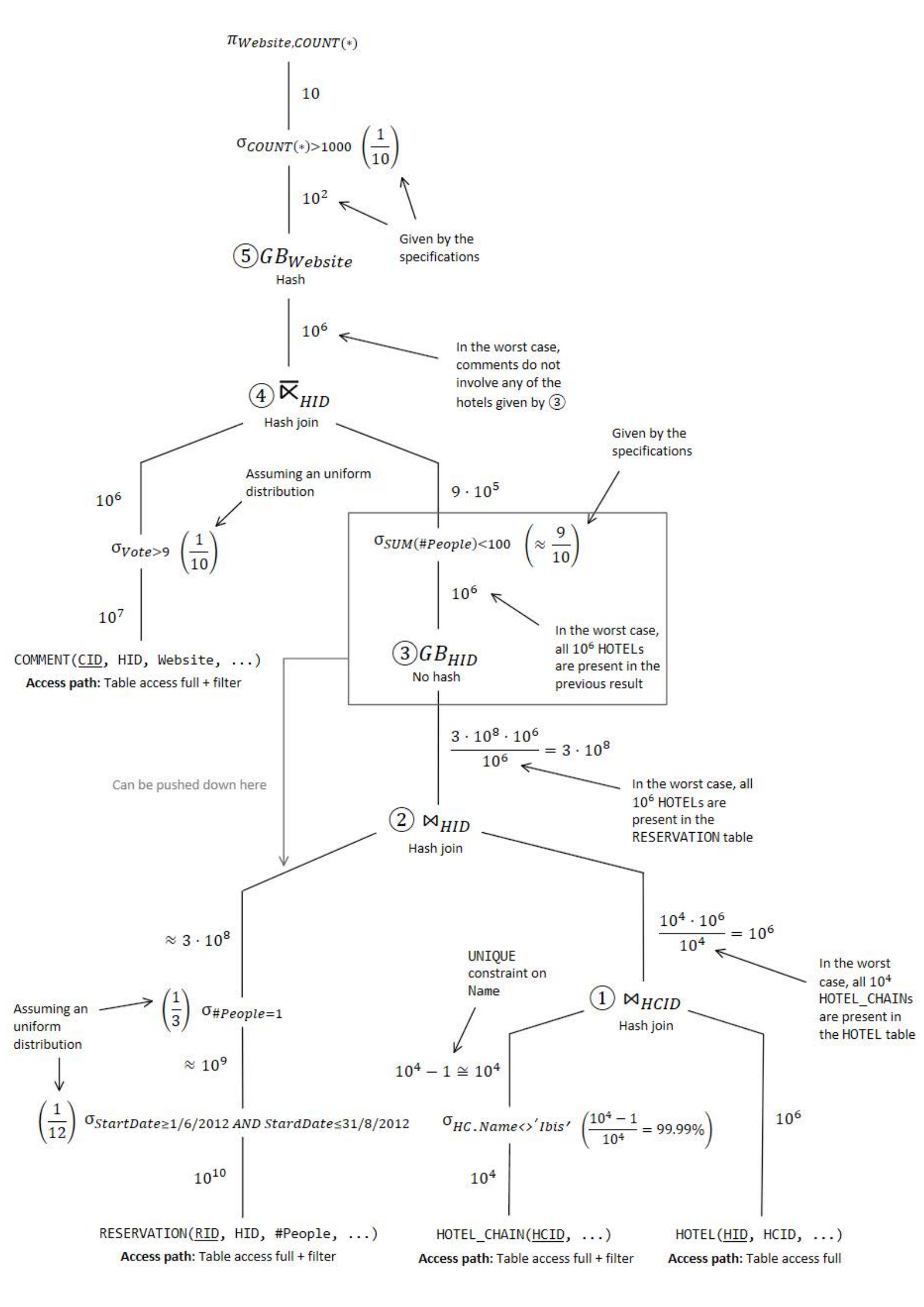
Master of Science in Computer Engineering

Database Management Systems' first homework

Marco Micera

Academic Year 2017-2018

Contents


$\pi_{Website,COUNT(*)}$
10
$\sigma_{COUNT(*)>1000}$ $\left(\frac{1}{10}\right)$
10^2
Given by the specifications
⑤ $GB_{Website}$
Hash
10^6
In the worst case, comments do not involve any of the hotels given by ③
④ $\overline{\ltimes}_{HID}$
Hash join
Given by the specifications
Assuming an uniform distribution
10^6
$9 \cdot 10^5$
$\sigma_{Vote>9}$ $\left(\frac{1}{10}\right)$
$\sigma_{SUM(\#People)<100}$ $\left(\approx \frac{9}{10}\right)$
10^7
10^6
COMMENT(CID, HID, Website, ...)
Access path: Table access full + filter
③ GB_{HID}
No hash
In the worst case, all 10^6 HOTELs are present in the previous result
$\frac{3 \cdot 10^8 \cdot 10^6}{10^6} = 3 \cdot 10^8$
In the worst case, all 10^6 HOTELs are present in the RESERVATION table
Can be pushed down here
② $\bowtie_{HID}$
Hash join
$\approx 3 \cdot 10^8$
$\frac{10^4 \cdot 10^6}{10^4} = 10^6$
In the worst case, all 10^4 HOTEL_CHAINs are present in the HOTEL table
UNIQUE constraint on Name
Assuming an uniform distribution
$\left(\frac{1}{3}\right)$ $\sigma_{\#People=1}$
① $\bowtie_{HCID}$
Hash join
$\approx 10^9$
$10^4 - 1 \cong 10^4$
$\left(\frac{1}{12}\right)$ $\sigma_{StartDate\geq 1/6/2012\ AND\ StardDate\leq 31/8/2012}$
$\sigma_{HC.Name<>'Ibis'}$ $\left(\frac{10^4-1}{10^4} = 99.99\%\right)$
10^6
10^{10}
10^4
RESERVATION(RID, HID, #People, ...)
Access path: Table access full + filter
HOTEL_CHAIN(HCID, ...)
Access path: Table access full + filter
HOTEL(HID, HCID, ...)
Access path: Table access full

2 Access path

- For the `HOTEL` table: Table access full
- For the `HOTEL_CHAIN` table: Table access full + filter
- For the `RESERVATION` table: Table access full + filter
- For the `COMMENT` table: Table access full + filter

3 Inner query `JOIN` order

There are two possible orders by which the inner query double-`JOIN` can be performed.

- Performing the `JOIN` operation between the `RESERVATION` table and the `HOTEL` table first would involve respectively $3 \cdot 10^8$ and 10^6 tuples, and then $3 \cdot 10^8$ and 10^4 tuples for the subsequent `JOIN` between the previous result and the `HOTEL_CHAIN` table
- By instead performing the `JOIN` operation between the `HOTEL_CHAIN` table and the `HOTEL` table first, there will be involved respectively just 10^4 and 10^6 tuples, and then 10^6 and $3 \cdot 41^8$ tuples for the second `JOIN` between the previous result (1) and the `RESERVATION` table.

That is why the second order has been chosen.

4 `JOIN` and `GROUP BY` discussion

`JOIN` and `GROUP BY` operations have been numbered sequentially, as previously shown in the algebraic query tree: hence, this list will follow that notation.

(1) `JOIN` on `HCID` between `HOTEL_CHAIN` and `HOTEL`

- Merge join
  - No because both tables would need to be sorted, and they are too big (more than 10^3 tuples each)
  - There are no following operations that could benefit from this table sorting
- Nested loop join
  - Inner table: `HOTEL_CHAIN` (smallest one)
  - Outer table: `HOTEL`
  - No because both tables are too big (more than 10^3 tuples each)
- Hash join
  - Yes because both tables are big (more than 10^3 tuples each)
  - It does not support any following `GROUP BY` operation

(2) `JOIN` on `HID` between `RESERVATION` and (1)

- Merge join

  - No because both tables would need to be sorted, and they are too big (more than 10^3 tuples each)
  - There is a following `GROUP BY` operation that could benefit from this table sorting, but an hash join is more efficient and it also supports the following `GROUP BY` operation
- Nested loop join
  - Inner table: (1)
  - Outer table: `RESERVATION`
  - No because both tables are too big (more than 10^3 tuples each)
- Hash join
  - Yes because both tables are too big (more than 10^3 tuples each)
  - It supports the following `GROUP BY` operation

(3) `GROUP BY HID` on (2)

- No hash, already provided by the (2)nd `JOIN` operation

(4) Anti-semi `JOIN` on `HID` between `COMMENT` and (3)

- Merge join
  - No because both tables would need to be sorted, and they are too big (more than 10^3 tuples each)
  - The following `GROUP BY` operation could not benefit from this table sorting as the `GROUP BY` attribute (`Website`) is different from this `JOIN` attribute (`HID`)
- Nested loop join
  - Inner table: (3)
  - Outer table: `COMMENT`
  - No because both tables are too big (more than 10^3 tuples each)
- Hash join
  - Yes because both tables are too big (more than 10^3 tuples each)
  - The following `GROUP BY` operation cannot benefit from this table hashing as the `GROUP BY` attribute (`Website`) is different from this `JOIN` attribute (`HID`)

(5) `GROUP BY Website` on (4)

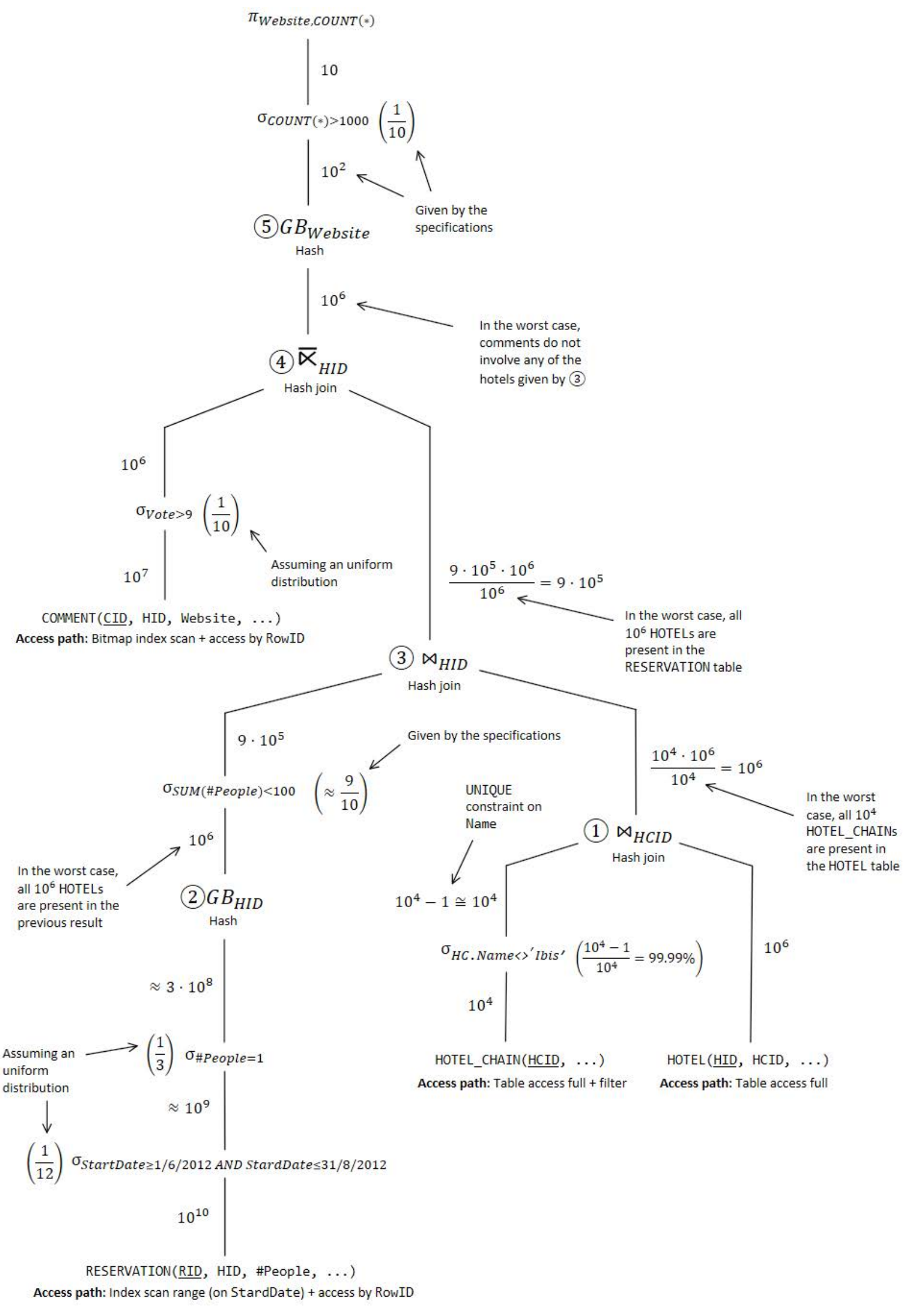
- Hash, as this `GROUP BY` attribute (`Website`) is different from the previous `JOIN` attribute (`HID`)

5 Indexes

- For the `HOTEL` table:
  - No index is needed for this query since all table rows must be accessed
- For the `HOTEL_CHAIN` table:
  - No index is needed for this query since the selection discards one row only, hence forcing the Buffer Manager to basically read the whole table
- For the `RESERVATION` table:
  - Secondary B^+-tree index on `StartDate`
    - Index range scan
    - Yes since selectivity is high enough ($\frac{1}{12} < \frac{1}{10}$)
  - Secondary B^+-tree or hash index on `#People`
    - No since selectivity is too low ($\frac{1}{3}$)
  - Secondary B^+-tree composite index on `StartDate` and `#People`
    - Yes since selectivity is high ($\frac{1}{12} \cdot \frac{1}{3} = \frac{1}{36} < \frac{1}{10}$)
    - However, no because a reduction factor of $\frac{1}{36}$ is not so much better than $\frac{1}{12}$, and the expensive index maintenance is not worth the higher selectivity
- For the `COMMENT` table:
  - Bitmap index on `Vote`
    - Yes because it's a low cardinality attribute
    - Yes because selectivity is high enough ($\frac{1}{10}$)

6 `GROUP BY` push down

The first `GROUP BY` operation on `HID` (previously denoted with (3)) can be pushed down on the `RESERVATION` branch. The algebraic query tree becomes as follows:


$\pi_{Website,COUNT(*)}$
10
$\sigma_{COUNT(*)>1000}$ $\left(\frac{1}{10}\right)$
10^2
Given by the specifications
⑤ $GB_{Website}$
Hash
10^6
In the worst case, comments do not involve any of the hotels given by ③
④ $\overline{\ltimes}_{HID}$
Hash join
10^6
$\sigma_{Vote>9}$ $\left(\frac{1}{10}\right)$
Assuming an uniform distribution
10^7
COMMENT(CID, HID, Website, ...)
Access path: Bitmap index scan + access by RowID
$\frac{9 \cdot 10^5 \cdot 10^6}{10^6} = 9 \cdot 10^5$
In the worst case, all 10^6 HOTELs are present in the RESERVATION table
③ $\bowtie_{HID}$
Hash join
$9 \cdot 10^5$
Given by the specifications
$\sigma_{SUM(\#People)<100}$ $\left(\approx \frac{9}{10}\right)$
10^6
In the worst case, all 10^6 HOTELs are present in the previous result
② GB_{HID}
Hash
$\approx 3 \cdot 10^8$
Assuming an uniform distribution
$\left(\frac{1}{3}\right)$ $\sigma_{\#People=1}$
$\approx 10^9$
$\left(\frac{1}{12}\right)$ $\sigma_{StartDate\geq 1/6/2012\ AND\ StardDate\leq 31/8/2012}$
10^{10}
RESERVATION(RID, HID, #People, ...)
Access path: Index scan range (on StardDate) + access by RowID
$\frac{10^4 \cdot 10^6}{10^4} = 10^6$
In the worst case, all 10^4 HOTEL_CHAINs are present in the HOTEL table
UNIQUE constraint on Name
① $\bowtie_{HCID}$
Hash join
$10^4 - 1 \cong 10^4$
$\sigma_{HC.Name<>'Ibis'}$ $\left(\frac{10^4-1}{10^4} = 99.99\%\right)$
10^4
HOTEL_CHAIN(HCID, ...)
Access path: Table access full + filter
10^6
HOTEL(HID, HCID, ...)
Access path: Table access full

Pushing down the `GROUP BY HID` operation brings an advantage since the `JOIN` on `HID` left branch's cardinality drops from $3 \cdot 10^8$ to $9 \cdot 10^5$, making the `JOIN` operation faster.
Note that since the order of `GROUP BY` and `JOIN` operations has changed, also the numeration has, so the following `JOIN` and `GROUP BY` discussion will follow this new notation.

6.1 `GROUP BY` and `JOIN` rediscussion

(1) `JOIN` on `HCID` between `HOTEL_CHAIN` and `HOTEL`

- Merge join
  - No because both tables would need to be sorted, and they are too big (more than 10^3 tuples each)
  - There are no following operations that could benefit from this table sorting
- Nested loop join
  - Inner table: `HOTEL_CHAIN` (smallest one)
  - Outer table: `HOTEL`
  - No because both tables are too big (more than 10^3 tuples each)
- Hash join
  - Yes because both tables are big (more than 10^3 tuples each)
  - It does not support any following `GROUP BY` operation

(2) `GROUP BY HID` on `RESERVATION`

- Hash-based: sorting would be expensive since the table is too big ($3 \cdot 10^8$ rows)

(3) `JOIN` on `HID` between (1) and (2)

- Merge join
  - No because both tables would need to be sorted, and they are too big (more than 10^3 tuples each)
  - It would help the following anti-semi `JOIN` (4) if it also would be a merge one
- Nested loop join
  - Inner table: (2)
  - Outer table: (1)
  - No because both tables are too big (more than 10^3 tuples each)
- Hash join
  - Yes because both tables are too big (more than 10^3 tuples each)

④ Anti-semi `JOIN` on `HID` between `COMMENT` and ③

- Merge join
  - No because both tables would need to be sorted, and they are too big (more than 10^3 tuples each)
  - The following `GROUP BY` operation could not benefit from this table sorting as the `GROUP BY` attribute (`Website`) is different from this `JOIN` attribute (`HID`)
- Nested loop join
  - Inner table: ③
  - Outer table: `COMMENT`
  - No because both tables are too big (more than 10^3 tuples each)
- Hash join
  - Yes because both tables are too big (more than 10^3 tuples each)
  - The following `GROUP BY` operation cannot benefit from this table hashing as the `GROUP BY` attribute (`Website`) is different from this `JOIN` attribute (`HID`)

⑤ `GROUP BY Website` on ④

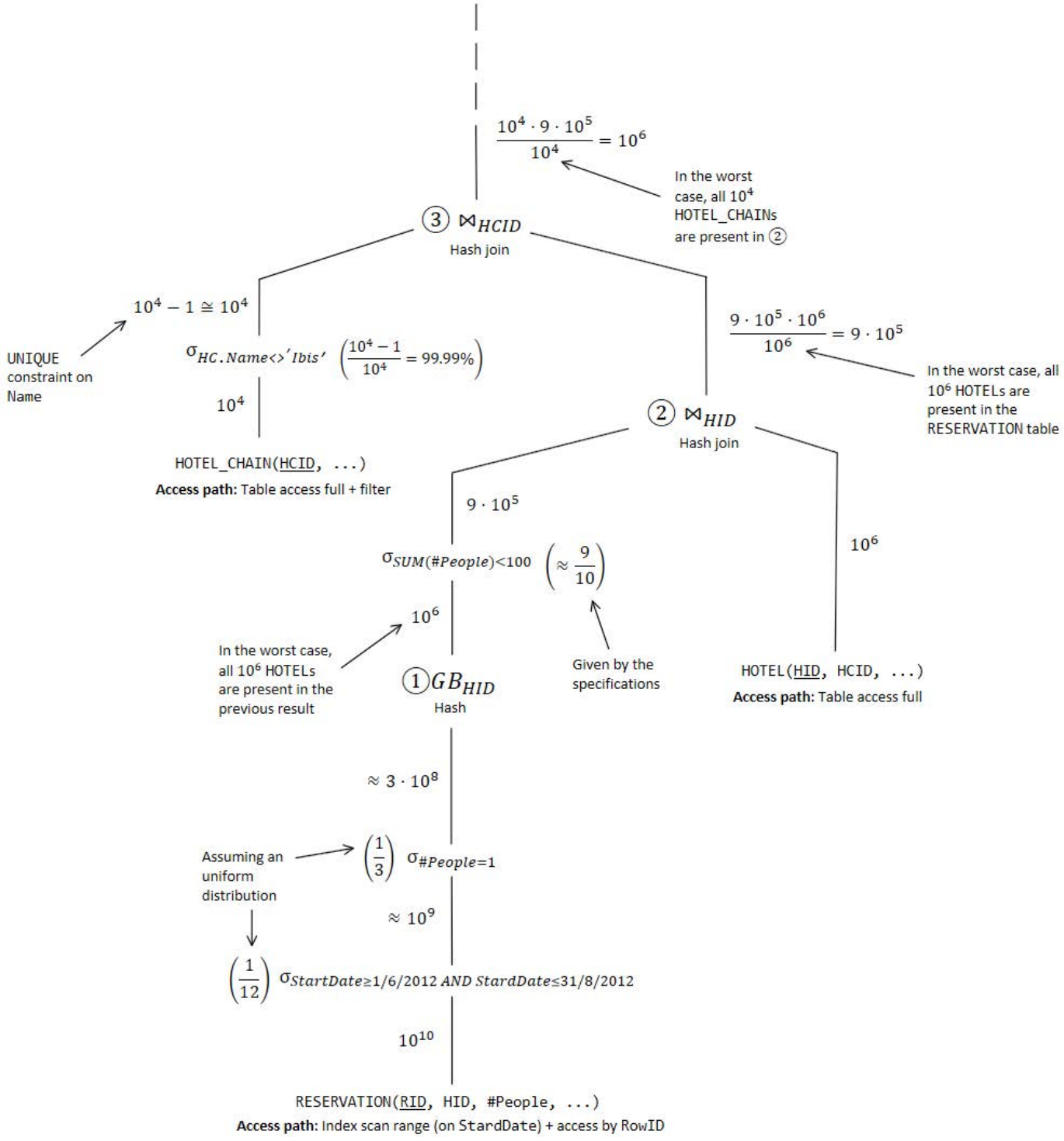
- Hash, as this `GROUP BY` attribute (`Website`) is different from the previous `JOIN` attribute (`HID`)

6.2 Inner query `JOIN` order

Having pushed down the `GROUP BY` operation in the algebraic query tree, it is now necessary to re-evaluate the inner query `JOIN` order in order to obtain the best overall performance.
Two options, as before:

- Performing the `JOIN` operation between the `HOTEL_CHAIN` table and the `HOTEL` table first would involve respectively 10^4 and 10^6 tuples, and then 10^6 and $9 \cdot 10^5$ tuples for the subsequent `JOIN` between the previous result and the `RESERVATION` table
- By instead performing the `JOIN` operation between the `RESERVATION` table and the `HOTEL` table first, there will be involved respectively just $9 \cdot 10^5$ and 10^6 tuples, and then $9 \cdot 10^5$ and 10^4 tuples for the second `JOIN` between the previous result and the `HOTEL_CHAIN` table.

It is now clear why the inner query `JOIN` order has been changed according to the second point of this list.
The modified section of the algebraic query tree is reported in the next page.



7 Access path with indexes

In order to exploit the secondary physical structures introduced in section 5, access paths have been changed in the following way:

- For the `HOTEL` table: Table access full
- For the `HOTEL_CHAIN` table: Table access full + filter
- For the `RESERVATION` table: Index range scan (on `StartDate`) + access by `RowID`
- For the `COMMENT` table: Bitmap index scan + access by `RowID`