SQL: Structured Query Language
Grouping Queries: Example

How Grouping works

Consider two relations R(A,B,C) and S(B,D) (all attributes are integer). Consider the following instances of tables R and S:

R

A	B	C
1	2	2
1	4	4
2	2	5
3	4	8
2	4	1
3	4	12

S

B	D
2	1
4	7

Consider the following SQL query

```
SELECT R.A, SUM(R.B), AVG(R.C), COUNT(*), MIN(S.D)
FROM  R, S
WHERE R.B = S.B and
      R.C < 10
GROUP BY R.A
HAVING COUNT(*) > 1;
```

This query is evaluated as follows.

Step 1. FROM Clause: Cartesian Product. First, the FROM clause is evaluated. It is convenient to view thist step as creation of a cartesian product of all tables referenced in the FROM clause. Here, this leads to computation of $R \times S$:

R×S

R.A	R.B	R.C	S.B	S.D
1	2	2	2	1
1	4	4	2	1
2	2	5	2	1
3	4	8	2	1
2	4	1	2	1
3	4	12	2	1
1	2	2	4	7
1	4	4	4	7
2	2	5	4	7
3	4	8	4	7
2	4	1	4	7
3	4	12	4	7

Step 2: `WHERE` Clause: Join and selection. On this step, each tuple in the cartesian product constructed on the previous step is evaluated against the conditions specified in the `WHERE` clause.

Here, the first condition, `R.B=S.B` specifies an equijoin between `R` and `S`, while the second condition, `R.C < 10` specifies a selection. In the table below (on the right), tuples satisfying the first condition are marked in *italics*, while tuples satisfying the second condition are in **bold**. The result of this operation is shown on the left.

R×S

R.A	R.B	R.C	S.B	S.D
1	***2***	***2***	***2***	***1***
1	**4**	**4**	**2**	**1**
2	***2***	***5***	***2***	***1***
3	**4**	**8**	**2**	**1**
2	**4**	**1**	**2**	**1**
3	4	12	2	1
1	**2**	**2**	**4**	**7**
1	***4***	***4***	***4***	***7***
2	**2**	**5**	**4**	**7**
3	***4***	***8***	***4***	***7***
2	***4***	***1***	***4***	***7***
3	*4*	*12*	*4*	*7*

$\Longrightarrow$

$\sigma_{R.C<10}(\mathtt{R} \bowtie_{R.B=S.B} \mathtt{S})$

R.A	R.B	R.C	S.B	S.D
1	2	2	2	1
2	2	5	2	1
1	4	4	4	7
3	4	8	4	7
2	4	1	4	7

Step 3: `GROUP BY` clause: grouping. The `GROUP BY` clause causes the transformation from the space of attributes defined by $R \times S$ to the space of attributes that consists of

- All attributes mentioned in the `GROUP BY` clause.
- All aggregates of the attrtibutes NOT mentioned in the `GROUP BY` clause.
- `COUNT(*)`.

We can illustrate it in two steps. First, we reorder tuples in the $\sigma_{R.C<10}(\mathtt{R} \bowtie_{R.B=S.B} \mathtt{S})$ and identify groups:

$\sigma_{R.C<10}$(R $\bowtie_{R.B=S.B}$ S)

R.A	R.B	R.C	S.B	S.D
1	2	2	2	1
2	2	5	2	1
1	4	4	4	7
3	4	8	4	7
2	4	1	4	7

$\Longrightarrow$

$\sigma_{R.C<10}$(R $\bowtie_{R.B=S.B}$ S)

R.A	R.B	R.C	S.B	S.D
1	2	2	2	1
	4	4	4	7
2	2	5	2	1
	4	1	4	7
3	4	8	4	7

Next, we replace each of the attributes R.B, R.C, S.B, S.C with five columns representing the aggregates COUNT(DISTINCT), SUM(), AVG(), MIN() and MAX() for each column. We also add one more attribute, COUNT(*) to the list. We then populate the new columns with the corresponding values:

$\sigma_{R.C<10}$(R $\bowtie_{R.B=S.B}$ S)

R.A	R.B	R.C	S.B	S.D
1	2	2	2	1
	4	4	4	7
2	2	5	2	1
	4	1	4	7
3	4	8	4	7

$\Longrightarrow$

$\gamma_{R.A}(\sigma_{R.C<10}$(R $\bowtie$ S))

R.A	COUNT(*)	C(R)	A(B)	S(B)	m(B)	M(B)	C(C)	A(C)	S(C)	m(C)	M(C)	C(D)	A(D)	S(D)	m(D)	M(D)
1	2	2	3	6	2	4	2	3	6	2	4	2	4	8	1	7
2	2	2	3	6	2	4	2	3	6	1	5	2	4	8	1	7
3	1	1	4	4	4	4	1	8	8	8	8	1	7	7	7	7

(Note: C(<Att>) stands for COUNT(DISTINCT <Att>), S(Att) is for SUM(<Att>), A(<Att>) is for AVG(<Att>), m(<Att>) is for MIN(<Att>), and M(<Att>) is for MAX(<Att>). Also, to save space, we moved to the natural join of R and S and removed S.B from the table.)

Step 4. HAVING clause: selection of groups. On this step, the conditions specified in the HAVING clause of the query are checked for each tuple in the result of the grouping.

In our query, the HAVING clause has an atomic condition COUNT(*)> 1. First two groups satisfy it, while the second - does not, which results in the following table:

$\sigma_{\texttt{COUNT(*)}>1}(\gamma_{R.A}(\sigma_{R.C<10}$(R $\bowtie$ S)))

R.A	COUNT(*)	C(R)	A(B)	S(B)	m(B)	M(B)	C(C)	A(C)	S(C)	m(C)	M(C)	C(D)	A(D)	S(D)	m(D)	M(D)
1	2	2	3	6	2	4	2	3	6	2	4	2	4	8	1	7
2	2	2	3	6	2	4	2	3	6	1	5	2	4	8	1	7

Step 5. SELECT clause: projection. On the final step, projection takes place. In our example, only the group identifier, R.A and three aggregate attributes (one for each column form R$\bowtie$S) are kept, all other columns are removed. The result of the entire SELECT statemnet is

R.A	SUM(R.B	AVG(R.C)	MIN(S.D)
1	6	3	1
2	6	3	1

Comments

Please, note the following. The particular example is deisgned to illustrate on what data SQL query processors execute SELECT statements. However, we note that not all of the tables discussed in this handout are *materialized*, i.e., explicitly computed. In particular:

- query processors almost never materialize cartesian products if there is a join operation present. Joins are computed directly.

- the table shown at the end of Step 3 is not fully materialized. In particular, only the columns mentioned either in `SELECT` or `HAVING` clauses will be materialized in addition to the columns defining the group. In our example, the materialized table would have columns `(R.A, COUNT(*), SUM(R.B), AVG(R.C), MIN(S.D))`.