05. Querying Relations

[PM Jat, DAIICT, Gandhinagar]

Aggregate operations on Relations

Given a relation, we perform some operations over all tuples or grouped tuples

eno	ename	dob	gender	salary	super_eno	dno
105	Jayesh	10-11-1967	M	55000		1
102	Sanna	23-07-1982	F	35000	105	5
106	Vijay	20-06-1971	M	53000	105	4
101	Sanjay	09-11-1955	M	70000		
108	Priya	19-07-1978	F	45000	106	4
104	Ramesh	15-09-1972	M	38000	106	5
103	Kalpesh	31-07-1982	F	25000	102	5
107	Ahmad	29-03-1979	M	25000	106	4
111	Kirit	08-12-1985	M	40000	102	5

Aggregation operations are basically counting or summing of values of an attribute of a relation. Following are common aggregation operations – COUNT, SUM, AVG, MAX, MIN.

Aggregation can be computed over all tuples or on grouped tuples of relations.

Following are examples of queries that require performing aggregate operations, and thus are called aggregate queries -

- Find out the total salary we pay to all employees.
- Find out the total number of employees, the total number of supervisors, and so
- Find out the department-wise salary statistics, i.e., sum, avg, max, min
- Find out employees who are drawing less than average salary,
- Find out the average salary paid to managers,
- And so forth

Aggregation operations are expressed using script F (F) operator

$\mathcal{F}_{\text{<funtion-list>}}(r)$

Examples

$\mathcal{F}_{\text{COUNT(ENO), AVG(SALARY)}}(\text{employee})$

Written in SQL as

```
SELECT count(ENO), avg(salary) FROM employee;
```

The result of this operation will be a single tuple having two columns.

Another example

$\mathcal{F}_{\text{COUNT(ENO), MAX(SALARY), MIN(SALARY), AVG(SALARY)}}(\text{employee})$

```
SELECT count(ENO), max(salary), min(salary), avg(salary) FROM employee;
```

Aggregation over grouped tuples

In this case, tuples of operand relation are grouped based on some attributes(s) value(s). We call these attributes grouping attributes.

For every distinct value of grouping attribute(s), there will be a group, and then an aggregation operation is computed for each group.

<grouping-attributes> $\mathcal{F}$ <funtion-list>(r)

Example:

DNO $\mathcal{F}$ COUNT(ENO), SUM(SALARY) (EMPLOYEE)

eno	ename	dob	gender	salary	super_eno	dno
105	Jayesh	10-11-1967	M	55000		1
102	Sanna	23-07-1982	F	35000	105	5
106	Vijay	20-06-1971	M	53000	105	4
101	Sanjay	09-11-1955	M	70000		
108	Priya	19-07-1978	F	45000	106	4
104	Ramesh	15-09-1972	M	38000	106	5
103	Kalpesh	31-07-1982	F	25000	102	5
107	Ahmad	29-03-1979	M	25000	106	4
111	Kirit	08-12-1985	M	40000	102	5

DNO $\mathcal{F}$ COUNT(ENO), SUM(SALARY) (EMPLOYEE)

eno	ename	dob	gender	salary	superv	dno
101	Sanjay	09-11-1955	M	70000	null	null

dno	count	sum
null	1	70000

eno	ename	dob	gender	salary	superv	dno
105	Jayesh	10-11-1967	M	55000	null	1

dno	count	sum
1	1	55000

eno	ename	dob	gender	salary	superv	dno
106	Vijay	20-06-1971	M	53000	105	4
108	Priya	19-07-1978	F	45000	106	4
107	Ahmad	29-03-1979	M	25000	106	4

dno	count	sum
4	3	123000

eno	ename	dob	gender	salary	superv	dno
102	Sanna	23-07-1982	F	35000	105	5
104	Ramesh	15-09-1972	M	38000	106	5
103	Kalpesh	31-07-1982	F	25000	102	5
111	Kirit	08-12-1985	M	40000	102	5

dno	count	sum
5	4	138000

dno	count	sum
null	1	70000
5	4	138000
4	3	123000
1	1	55000

SQL:

```
SELECT dno, count(ENO), sum(salary) FROM employee GROUP BY dno;
```

Another example

DNO, GENDER $\mathcal{F}$COUNT(ENO) , AVG(SALARY) (EMPLOYEE)

```
SELECT dno, gender, count(ENO), avg(salary)
       FROM employee GROUP BY dno, gender;
```

Renaming of operations in aggregation

We can use renaming in aggregate operation,

For example, an aggregate query-

$$_{DNO}\mathcal{F}_{COUNT(ENO) \rightarrow No\_of\_Emps,\ AVG(SALARY) \rightarrow AVG\_SAL}(EMPLOYEE)$$

In SQL, we write as following –

```
SELECT dno, count(ENO) AS No_of_Emps, avg(salary) AS avg_sal
       FROM employee
       GROUP BY dno;
```

NULL's in Aggregation

- NULL never contributes to sum, average, or count, and can never be the minimum or maximum of a column.
- But if there are no non-NULL values in a column, then the result of the aggregation is NULL.

Examples

```
SELECT count(*) FROM employee; -- counts all row, * is used for counting rows!
SELECT count(dno) FROM employee; -- counts occurrence of dno values (excludes NULL)
SELECT count(DISTINCT dno) FROM employee; --counts of distinct values for dno
SELECT count(super_eno) FROM employee;
SELECT count(DISTINCT super_eno) FROM employee;
SELECT min(salary), max(salary), avg(salary) FROM employee;
```

Exercise: Can you figure out, why the following references (in red) are invalid?

```
SELECT dno, ENO, avg(salary) AS avg_sal FROM employee GROUP BY dno;

SELECT dno, avg(salary) AS avg_sal FROM employee;
```

HAVING clause

HAVING is used to specify restrict over result of aggregation

For example:

```
SELECT dno, avg(salary) AS avg_sal
            FROM employee GROUP BY dno
            HAVING avg(salary) > 50000;
```

Algebraically, equivalent to

$$r1 \leftarrow {}_{DNO}\mathcal{F}_{AVG(SALARY) \rightarrow AVG\_SAL}(EMPLOYEE)$$

$$result \leftarrow \sigma_{avg(sal) > 50000}(r1)$$

Note that you cannot use avg_sal instead of avg(salary) in HAVING clause, because renaming is done at the time of projection.

Semantics of SQL SELECT statement

```
SELECT <attrib and/or function-list> (5)
FROM <relation-expression> (1)
[WHERE <condition>] (2)
[GROUP BY <grouping attribute(s)>] (3)
[HAVING <group-filter-condition>] (4)
[ORDER BY <attrib-list>]; (6)
```

r1 <- <relational-expression>
r2 <- $\sigma_{\text{<where-condition>}}(r1)$
r3 <- $_{\text{<group-attributes>}}F_{\text{<aggregate-operation>}}(r2)$
r4 <- $\sigma_{\text{<group-filter-condition>}}(r3)$
r5 <- $\pi_{\text{<attrib-list>}}(r4)$

- Result of FROM and WHERE is given to GROUP BY operation
- GROUP BY operation computes *aggregated values* for each group value, and gives you one tuple for each group value.
- **group-filter-conditions** in HAVING are used to apply restriction over result of GROUP BY operation
- Finally, project is applied as defined in SELECT clause of SELECT statement.