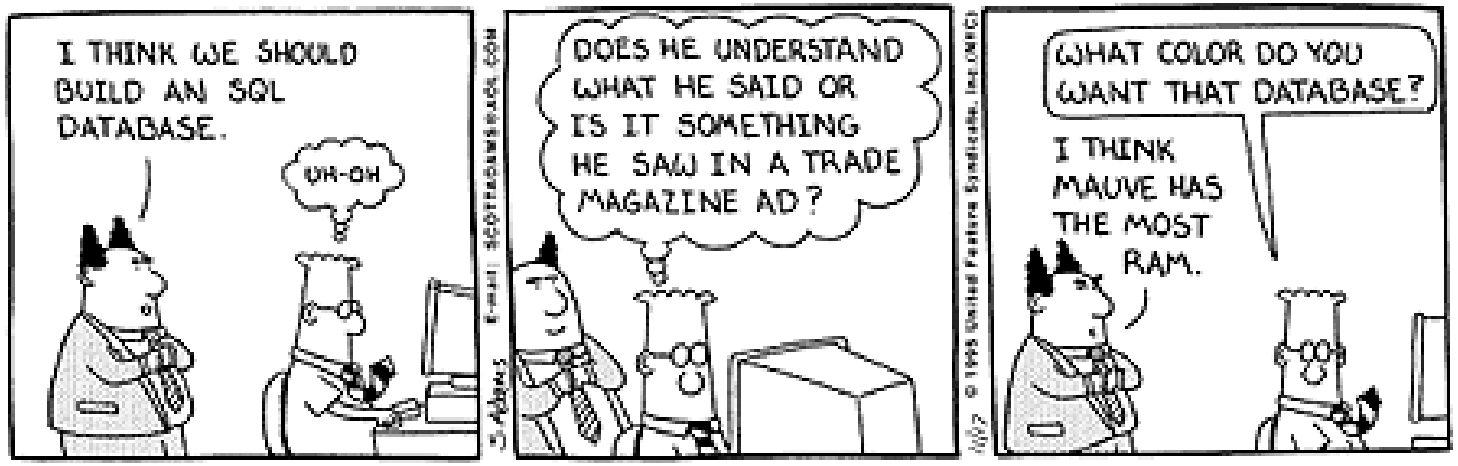
SQL: The Query Language
Part IV
Aggregate Queries

R&G - Chapter 5



Simple and Nested SQL Queries

- **Simple SQL Queries**

  SELECT A_1, A_2, ..., A_n
  FROM r_1, r_2, ..., r_m
  WHERE P

- **Nested SQL queries**

  SELECT A_1, A_2, ..., A_n
  FROM r_1, r_2, ..., r_m
  WHERE (NOT)IN|(NOT) EXISTS|Op ANY/ALL
  (SELECT A_1, A_2, ..., A_n
  FROM r_1, r_2, ..., r_m
  WHERE P)

Today's plan

- **Aggregate queries**

Aggregate Function

- **Input**: a collection of values.
- **Output**: a single value (aggregated result).
- Types of aggregate functions in SQL
  - *SUM*: returns the sum of all non-NULL values a set.
  - *AVG*: calculates the average of non-NULL values in a set.
  - *MIN*: returns the lowest value (minimum) in a set of non-NULL values.
  - *MAX*: returns the highest value (maximum) in a set of non-NULL values.
  - *COUNT*: returns the number of rows in a group, including rows with NULL values.
- Used ONLY in the SELECT and HAVING clauses (HAVING clause will be covered later today).

Examples of Aggregate Functions

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

single column

```
SELECT  COUNT (*)
FROM  Sailors S
```

```
SELECT  AVG (S.age)
FROM  Sailors S
WHERE  S.rating=10
```

```
SELECT COUNT (DISTINCT S.rating)
FROM  Sailors S
WHERE  S.sname='Bob'
```

Input to Aggregate Function

- Types of input data
  - SUM and AVG
    - Only accept numerical values as input.
  - MIN , MAX and COUNT
    - Accept both numerical and categorical values at input.

- Null values in input

Function	Consider Null values?
MIN	Only non-null values
MAX	Only non-null values
AVG	Only non-null values
SUM	Only non-null values
COUNT	Including null values

Dealing with Duplicates in Input

- Two options:
  - *DISTINCT*: consider only distinct values in the input
  - *ALL*: consider all values including duplicates.
  - By default: *ALL*

Example: SELECT COUNT(DISTINCT column_name)

Example of DISTINCT and ALL

sno	salary
SL100	30000
SL101	10000
SL102	10000
SL103	20000

Staff

Query 1:

```
SELECT  AVG(DISTINCT salary) AS avg_sal
FROM  Staff;
```

Result:

avg_sal
20000

= (30K+10K+20K)/3

Query 2:

```
SELECT  AVG(ALL salary) AS avg_sal
FROM  Staff;
```

Result:

avg_sal
17500

= (30K+10K+10K+20K)/4

Example of Aggregate Queries

- **Schema**
  - Boats (bid, bname, color)
  - Sailors(sid, sname, rating, age)
  - Reserves(sid, bid, day)

Query: Find name and age of the oldest sailor(s)

```
SELECT S.sname, S.age
FROM   Sailors S
WHERE  S.age = (SELECT  MAX (S2.age)
                FROM  Sailors S2);
```

Solution 1

Example of Aggregate Queries

- **Schema**
  - Boats (bid, bname, color)
  - Sailors(sid, sname, rating, age)
  - Reserves(sid, bid, day)

Query: Find name and age of the oldest sailor(s)

```
SELECT   S.sname, MAX(S.age) as maxage
FROM     Sailors S;
```

Solution 2

Question: is Solution 2 correct?

Answer

Query: Find name and age of the oldest sailor(s)

```
SELECT S.sname, MAX(S.age)
FROM   Sailors S;
```

- This solution is wrong!
  - MySQL: return the name of the 1st sailor, and the maximum age of all sailors
  - SQL server and Oracle: return error message.

- **Rule**: For aggregate queries, DO NOT put non-aggregate attributes and aggregate functions together in SELECT clause if no GROUP BY clause is used (GROUP BY clause will be explained next).

GROUP BY and HAVING

- Sometimes, we want to ask for groups of records.
- The query description normally starts with "For each…".
- Consider the query: *For each rating level, find the age of the youngest sailor.*
  - If we know that rating values go from 1 to 10
    - We can write 10 queries that look like this (!):

```
For i = 1, 2, ... , 10:   SELECT  MIN (S.age)
                          FROM  Sailors S
                          WHERE  S.rating = i
```

Rating	Min_age
7	25
8	22
9	30
10	28

  - However, this solution does not work in general
    - There can be more than 10 ratings. Should we write a query for each rating separately?
    - Even worse, we don't know how many rating levels exist, and what the rating values for these levels are. How can we write the queries?

Solution: GROUP BY Clause

- The GROUP BY statement groups rows of the same values into groups.
- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

Non-aggregate attributes

```
SELECT  att1, att2, ... attk, aggregate_function
FROM    tables
[WHERE  qualification]
GROUP BY att1, att2, ... attn;
```

  - The *grouping attributes* $att_1, att_2, \ldots att_n$ should correspond to the grouping criteria XXX as asked in the query "for each XXX".
  - If there is any non-aggregate attribute in SELECT clause, these attributes $att_1, att_2, \ldots att_k$ must appear in GROUP BY clause (k<=n)
  - GROUP BY clause is always placed after WHERE clause if it is present.
    - If there is no WHERE clause, then GROUP BY clause is placed after FROM clause

Example of GROUP BY

- Schema:
  - Sailors(sid, sname, age, rating)
- Query: For each rating level, find the age of the youngest sailor.

```
SELECT    rating, MIN(age) as Min_age
FROM      Sailors
GROUP BY  rating;
```

Rating	Min_age
7	18
8	19
9	24
10	28

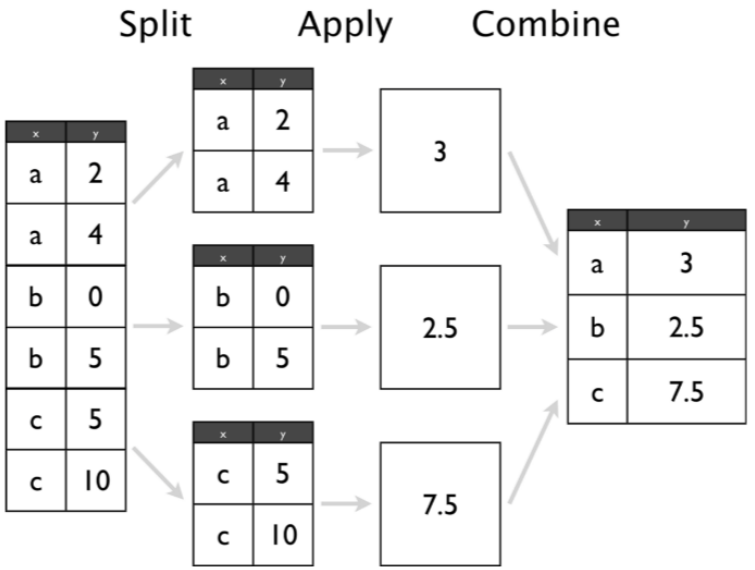
Output

Evaluation of GROUP BY

- **Split-apply-combine** strategy
  - *Split* phase: divides the groups by the values on the grouping attributes.
  - *Apply* phase: applies the aggregate function and generates a single value.
  - *Combine* phase: For each group, combines its grouping value with aggregated results.
- **Example:**

Query:

```
SELECT    X,   AVG(Y)
FROM      T
GROUP BY   X;
```



More Examples of GROUP BY

Schema:

Sailors(sid, sname, age, rating)

Q1: For each rating level, find the average age of the sailors

```
SELECT   AVG(age) AS avg-age
FROM     Sailors
GROUP BY  rating;
```

Q2: For each rating level, find the age of the youngest sailor who are older than 18

```
SELECT   MIN(age) AS min-age
FROM     Sailors
WHERE    age > 18
GROUP BY  rating;
```

GROUP BY with Join

- **Schema**
  - Boats (bid, bname, color)
  - Sailors(sid, sname, rating, age)
  - Reserves(sid, bid, day)

Query: *For each* red boat, return its id and number of reservations.

```
SELECT    B.bid, COUNT(*) AS numres
FROM      Boats B NATURAL JOIN Reserves R
WHERE     B.color='red'
GROUP BY  B.bid;
```

Bid	numres
100	10
104	2
…	…

HAVING Clause

- HAVING clause is a conditional clause with GROUP BY clause.
  - It filters groups based on GROUP BY results.
- HAVING requires a GROUP BY clause to be present
- Attributes in *group-qualification* of HAVING clause must be either an aggregate op or appear in $att_1, att_2, \ldots att_n$ of GROUP BY clause

```
SELECT   att1, att2, ... attk, aggrega_function
FROM     tables
[WHERE   qualification]
GROUP BY att1, att2, ... attn
HAVING   group-qualification;
```

Evaluation of HAVING clause

Query: for each department, return its average employee salary if it is more than $3,000;

Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

SELECT *DeptID, AVG(Salary)*
FROM *Employee*
GROUP BY *DeptID;*

GROUP BY Employee Table using DeptID

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

SELECT *DeptID, AVG(Salary)*
FROM *Employee*
GROUP BY *DeptID*
HAVING *AVG(Salary) > 3000;*

HAVING

DeptID	AVG(Salary)
2	4000.00
3	4250.00

Example of HAVING Clause

Schema:

Sailors(sid, sname, age, rating)

Query: For each rating group that has more than 10 sailors, find its lowest rating.

```
SELECT  MIN(rating) AS min-rating
FROM  Sailors
GROUP BY  rating
HAVING  COUNT (*) > 10;
```

HAVING VS. WHERE Clauses

- WHERE clause filters the individual records, while HAVING Clause filters the whole group.
  - Any condition on the aggregate funtions (e.g., avg, min, max) should be specified in HAVING clause;
  - Any condition without aggregation should be specified in WHERE clause
- WHERE clause is applied before GROUP BY, while HAVING clause is applied after GROUP BY.

Example: HAVING VS. WHERE Clauses

Query: for each red boat, find its bid and the number of reservations.

Solution 1:

```
SELECT B.bid, COUNT(*) AS rCount
FROM Boats B NATURAL JOIN Reserves R
WHERE B.color='red'
GROUP BY B.bid;
```

Solution 2:

```
SELECT B.bid, COUNT(*) AS rCount
FROM Boats B NATURAL JOIN Reserves R
GROUP BY B.bid
HAVING B.color='red';
```

Which solution is correct?

- B.color='red' is specified on non-aggregate values.
- Specify it in WHERE clause not HAVING clause

Use GROUP BY and HAVING Clauses for Division operator

Query: Find the name of sailors who've reserved all boats.

- Hint: for each sailor, check whether the number of distinct boats he/she has reserved = the total number of (distinct) boats.

```
SELECT S.sname
FROM   Sailors S NATURAL JOIN reserves R
GROUP  BY S.sname, S.sid
HAVING COUNT(DISTINCT R.bid)=
                (SELECT COUNT (*) FROM Boats);
```

Questions:

1. Can we use *sid only* in GROUP BY clause?
2. Can we use *sname only* in GROUP BY clause?
3. Can we remove DISTINCT from HAVING clause, assuming the same sailor can reserve the same boat at different days (i.e., the key of the Reserves table is (sid, bid, day))?