CSCD 327: Relational Database Systems

Aggregate functions

Instructor: Dr. Dan Li

Aggregate Functions

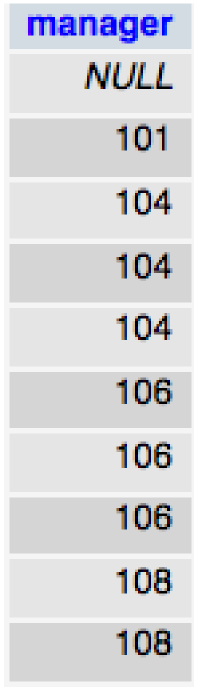
- Take a collection of values from a column as input and return a single value.
- Average: **avg()**
- Minimum: **min()**
- Maximum: **max()**
- Total: **sum()**
- Count: **count(), count(*)**

Simple Aggregate Functions

- *What are the average quota and average sales of our salespeople?*
  SELECT **AVG**(QUOTA), **AVG**(SALES)
  FROM SALESREPS;
- *What are the total quotas and sales for all salespeople?*
  SELECT **SUM**(QUOTA), **SUM**(SALES)
  FROM SALESREPS;
- *What are the smallest and largest assigned quotas?*
  SELECT **MIN**(QUOTA), **MAX**(QUOTA)
  FROM SALESREPS ;
- *What is the total of the orders taken by Bill Adams?*
  SELECT SUM(AMOUNT)
  FROM ORDERS, SALESREPS
  WHERE NAME = 'Bill Adams' AND REP = EMPL_NUM;
- *What is the best sales performance of any salesperson?*
  SELECT MAX(**100 * (SALES/QUOTA)**)
  FROM SALESREPS;

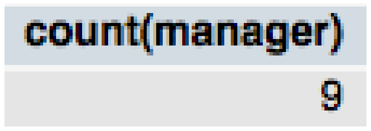
COUNT()

SELECT manager
FROM salesreps



manager
NULL
101
104
104
104
106
106
106
108
108

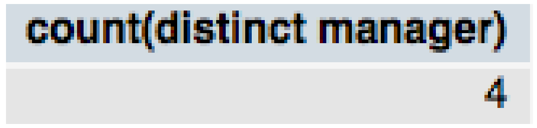
SELECT count(manager)
FROM salesreps;



count(manager)
9

NULL values are ignored.

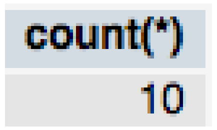
SELECT count(**DISTINCT** manager)
FROM salesreps



count(distinct manager)
4

Duplicates are eliminated.

SELECT count(*****)
FROM salesreps



count(*)
10

Returns the total number of rows.

NULL Values in Aggregate Functions

- In general, except for COUNT, aggregate functions ignore null values.
- If every data item in the column is NULL, then the SUM(), AVG(), MIN(), and MAX() column functions return a NULL value; the COUNT(*column*) function returns a value of zero.
- If no data items are in the column (that is, the column is empty), then the SUM(), AVG(), MIN(), and MAX() column functions return a NULL value; the COUNT() function returns a value of zero.
- The COUNT(*****) counts rows and does not depend on the presence or absence of NULL values in a column. If there are no rows, it returns a value of zero.

GROUP BY Clause

- *What is the average order size?*

```
SELECT AVG(AMOUNT)
  FROM ORDERS;
```

- *What is the average order size **for each salesperson**?*

```
SELECT REP, AVG(AMOUNT)
  FROM ORDERS
 GROUP BY REP;
```

Step 1: SQL divides the orders into groups of orders, with one group for each salesperson.
Step 2: For each group, SQL computes the average value of the AMOUNT column for all of the rows in the group and generates a single, summary row of query results.

More Grouped Queries

- *What is the range of assigned quotas in each office?*

```
SELECT REP_OFFICE, MIN(QUOTA), MAX(QUOTA)
  FROM SALESREPS
 GROUP BY REP_OFFICE;
```

- *How many salespeople are assigned to each office?*

```
SELECT REP_OFFICE, COUNT(*)
  FROM SALESREPS
 GROUP BY REP_OFFICE;
```

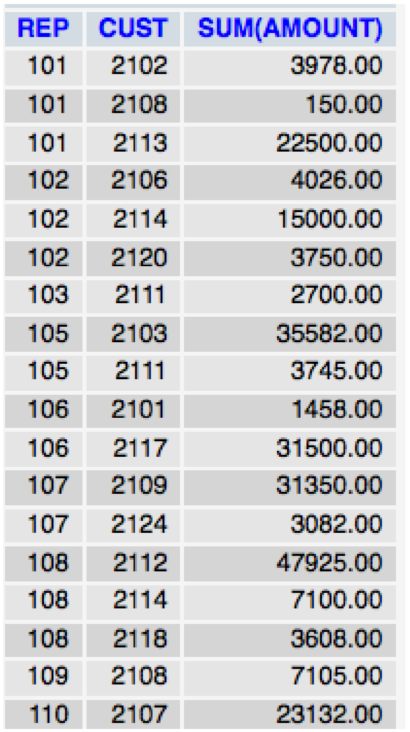
- *How many different customers are served by each salesperson?*

```
SELECT COUNT(DISTINCT CUST_NUM), 'customers for salesrep', CUST_REP
  FROM CUSTOMERS
 GROUP BY CUST_REP;
```

Multiple Grouping Columns

- *Calculate the total orders **for each customer of each salesperson**.*

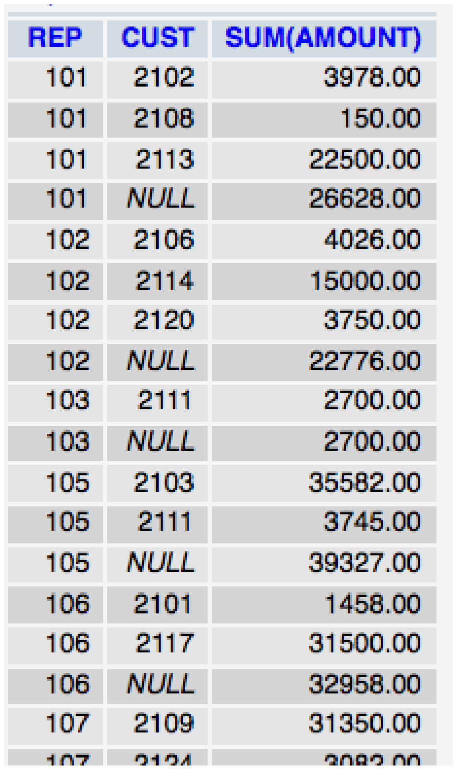
```
SELECT REP, CUST, SUM(AMOUNT)
  FROM ORDERS
 GROUP BY REP, CUST;
```



REP	CUST	SUM(AMOUNT)
101	2102	3978.00
101	2108	150.00
101	2113	22500.00
102	2106	4026.00
102	2114	15000.00
102	2120	3750.00
103	2111	2700.00
105	2103	35582.00
105	2111	3745.00
106	2101	1458.00
106	2117	31500.00
107	2109	31350.00
107	2124	3082.00
108	2112	47925.00
108	2114	7100.00
108	2118	3608.00
109	2108	7105.00
110	2107	23132.00

- *Calculate the total orders for each customer of each **salesperson with subtotals for each salesperson**.*

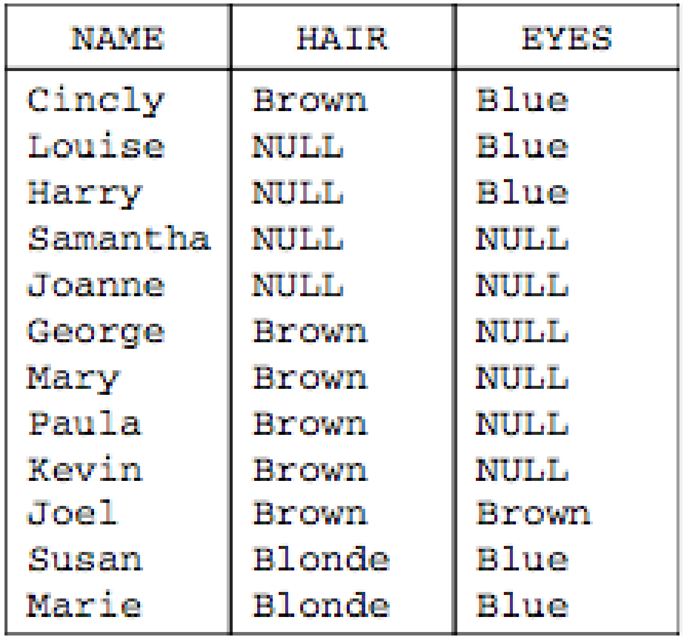
```
SELECT REP, CUST, SUM(AMOUNT)
  FROM ORDERS
GROUP BY REP, CUST WITH ROLLUP;
```



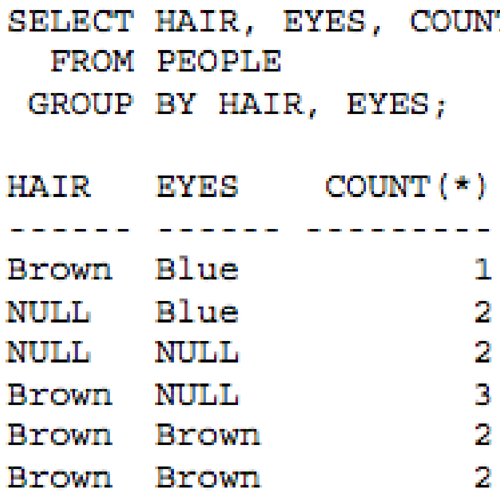
REP	CUST	SUM(AMOUNT)
101	2102	3978.00
101	2108	150.00
101	2113	22500.00
101	*NULL*	26628.00
102	2106	4026.00
102	2114	15000.00
102	2120	3750.00
102	*NULL*	22776.00
103	2111	2700.00
103	*NULL*	2700.00
105	2103	35582.00
105	2111	3745.00
105	*NULL*	39327.00
106	2101	1458.00
106	2117	31500.00
106	*NULL*	32958.00
107	2109	31350.00
107	2124	3082.00

NULL Values in GROUP BY

- The ANSI/ISO SQL standard considers two NULL values to be equal for purposes of the GROUP BY clause.
  - If two rows have NULLs in the same grouping columns and identical values in all of their non-NULL grouping columns, they are grouped together into the same row group.



NAME	HAIR	EYES
Cincly	Brown	Blue
Louise	NULL	Blue
Harry	NULL	Blue
Samantha	NULL	NULL
Joanne	NULL	NULL
George	Brown	NULL
Mary	Brown	NULL
Paula	Brown	NULL
Kevin	Brown	NULL
Joel	Brown	Brown
Susan	Blonde	Blue
Marie	Blonde	Blue

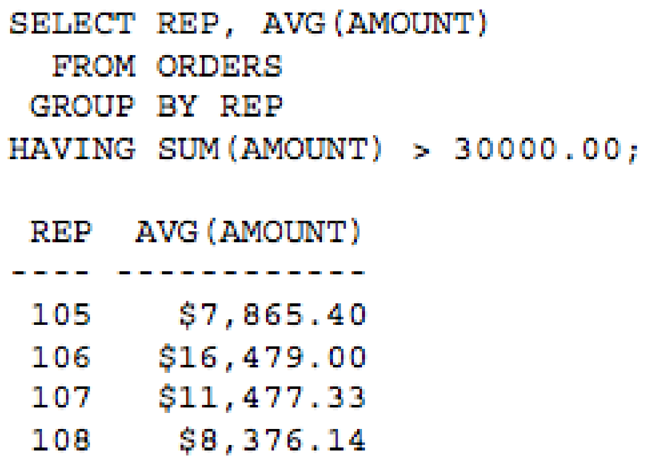


```
SELECT HAIR, EYES, COUNT(*)
  FROM PEOPLE
 GROUP BY HAIR, EYES;

HAIR   EYES    COUNT(*)
------ ------ ---------
Brown  Blue           1
NULL   Blue           2
NULL   NULL           2
Brown  NULL           3
Brown  Brown          2
Brown  Brown          2
```

Although this behavior of NULLs in grouping columns is clearly specified in the ANSI/ ISO standard, it is not implemented in all SQL dialects.

GROUP BY … HAVING …

- WHERE clause can be used to select and reject the **individual rows** that participate in a query
- The HAVING clause can be used to select and reject **row groups**.
  - The HAVING clause specifies a search condition for groups.
- *What is the average order size for each salesperson whose orders total more than $30,000?*



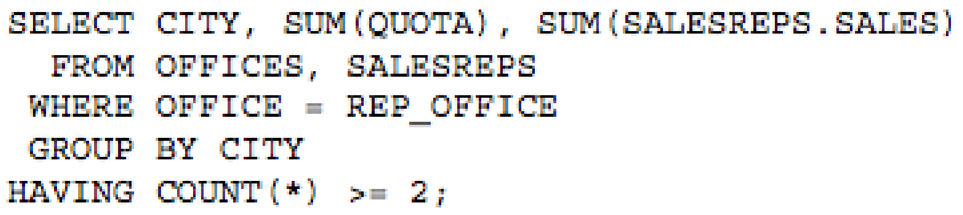
```
SELECT REP, AVG(AMOUNT)
  FROM ORDERS
 GROUP BY REP
HAVING SUM(AMOUNT) > 30000.00;

 REP  AVG(AMOUNT)
---- ------------
 105    $7,865.40
 106   $16,479.00
 107   $11,477.33
 108    $8,376.14
```

SQL Query Processing

1. FROM clause is first applied.
2. Then the WHERE clause is applied to retains those rows for which the search condition is TRUE.
3. Next, the GROUP BY clause is applied to arrange table into groups.
4. Then the HAVING clause is applied to retain those groups for which the search condition is TRUE.
5. Next, the SELECT is applied to list a subset of columns.
6. Finally, the ORDER BY is applied to sort the query results.

Query Processing Further Illustrated

- *For each office with two or more people, compute the total quota and total sales for all salespeople who work in the office.*



```
SELECT CITY, SUM(QUOTA), SUM(SALESREPS.SALES)
  FROM OFFICES, SALESREPS
 WHERE OFFICE = REP_OFFICE
 GROUP BY CITY
HAVING COUNT(*) >= 2;
```

1. Joins the OFFICES and SALESREPS tables to find the city where each salesperson works.
2. Groups the resulting rows by office.
3. Eliminates groups with two or fewer rows—these represent offices that don’t meet the HAVING clause criterion.
4. Calculates the total quota and total sales for each group.

- **ONCE AGAIN**, the WHERE clause is applied to *individual rows,* so the expressions it contains must be computable for individual rows. The HAVING clause is applied to *row groups,* so the expressions it contains must be computable for a group of rows.

NULL Values in HAVING

- If the search condition is TRUE, the row group is retained, and it contributes a summary row to the query results.
- If the search condition is FALSE, the row group is discarded, and it does not contribute a summary row to the query results.
- If the search condition is NULL, the row group is discarded, and it does not contribute a summary row to the query results.

Relational Algebra - Aggregation

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G_1, G_2, \ldots, G_n} \mathcal{G}_{F_1(A_1), F_2(A_2), \ldots, F_n(A_n)}(E)$$

  E is any relational-algebra expression
  - *G_1, G_2 …, G_n* is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

- Note: Some books/articles use γ instead of $\mathcal{G}$ (Calligraphic G)

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

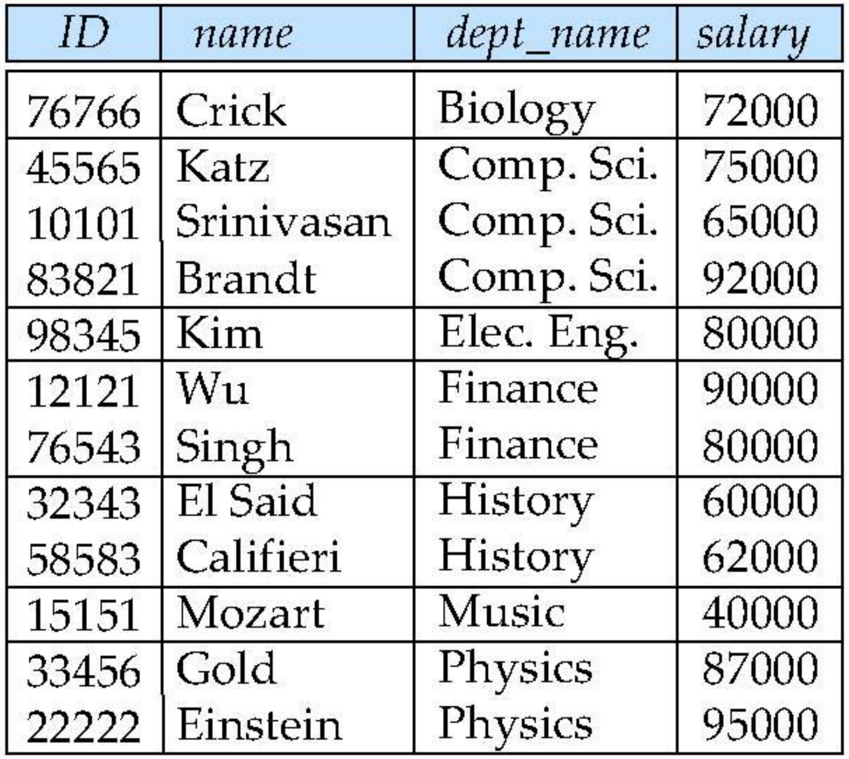
- $\mathcal{G}_{\textbf{sum(c)}}\,(\mathrm{r})$

sum(*c*)
27

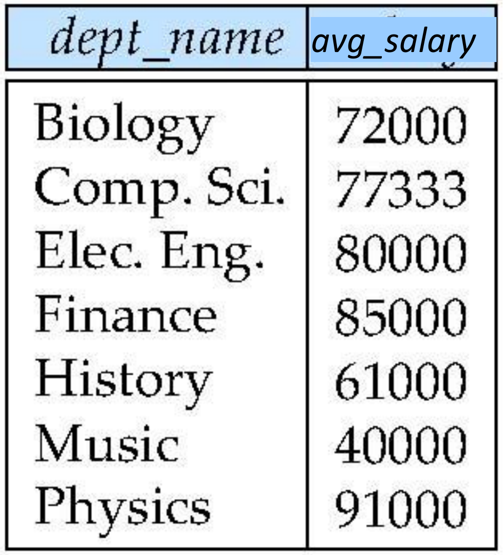
Aggregate Operation – Example

- Find the average salary in each department

$${}_{dept\_name}\mathcal{G}_{\ \mathbf{avg}(salary)}\ (instructor)$$



ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000



dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

$_{dept\_name}\mathcal{G}_{\mathbf{avg}(salary)\ \mathbf{as}\ avg\_sal}(instructor)$

Exercise

- *Find the course sections taught by more than one instructor.*
  - RA: Without using any aggregate function

$$\Pi_{course\_id, section\_id, year, semester}(\sigma_{ID <> ID2}(takes \bowtie \rho_{takes1(ID2, course\_id, section\_id, year, semester)}(takes)))$$

    - Should change takes into teaches, and also change section_id into sec_id

  - RA: Using an aggregate function

$$\sigma_{instrcnt > 1}\ ({}_{course\_id, section\_id, year, semester}\mathcal{G}_{\mathbf{count(*)\ as}\ instrcnt}(teaches))$$

  - SQL:

Select course_id, sec_id, year, semester, count(*)
From teaches
Group by course_id, sec_id, year, semester
Having count(*) > 1 ;