Aggregate Functions and Data Partitioning

Collections and aggregate functions

- A collection is a grouping of some variable number of data items (possibly zero)
- Usually the data items in a collection are of the same type
- Aggregate functions are functions that apply to collections, i.e., they consider all these data items in these collections
- Applied to a collection, an aggregate function returns a single value

Examples of collections

- sets, multisets, dictionaries (maps), relations
- vectors, lists, arrays, series
- data structures: stacks, queues, hash tables, trees, graphs

Aggregate functions on unordered collections

- We will restrict ourselves to aggregate functions on sets, bags, and relations
- COUNT (we will often use the notation |A| instead of COUNT(A))
- SUM, AVERAGE, MIN, and MAX
- COUNT({a,b,c}) = 3; COUNT({{a,c,c,b,b}}) = 6
- $SUM(\{1, 4, 7\}) = 12; SUM(\{\{1,1,1,1,1\}\}) = 5$
- $AVG(\{1,4,7\}) = 4$; $AVG(\{\{1,1,1,1,1\}\}) = 1$
- $MIN(\{1,4,7\}) = MIN(\{\{1,1,1,1,1\}\}) = 1$
- $MAX(\{1,4,7\}) = 7; MAX(\{\{1,1,1,1,1\}\}) = 1$
- MIN({John,Eric,Ann}) = Ann
- These aggregate functions are supported in SQL

Applications of aggregate functions

- Data analytics
- Formulating complex queries
- Facilitating efficient query evaluation

Counting the size of a set in SQL

- Let R(A₁,...,A_n) be a relation.
- Then |R| can be obtained as follows:

```
SELECT COUNT(*) FROM R r;
```

Alternatively,

```
SELECT COUNT(1) FROM R r;
```

• Of course we can restrict the COUNT function to apply to a subset of R by applying a WHERE clause.

COUNT examples

• "Find the number of courses in which the student with sid 10 is enrolled."

```
SELECT COUNT(*)
FROM Enroll E
WHERE E.sid = 's10';
```

 "Find the number of students who are not enrolled in any CS course."

```
SELECT COUNT(*)

FROM Student S

WHERE S.Sid NOT IN (SELECT E.Sid

FROM Enroll E, Course C

WHERE E.Cno = C.Cno AND C.Dept = 'CS');
```

COUNT Example

• Let R and S be two relations, then the following query will return $|R \times S| = |R||S|$, i.e. the size of the cartesian (cross) product of R and S.

SELECT COUNT(*) FROM R r, S s;

• $R \times S = \{(r,s) \mid r \in R \land s \in S\}$

COUNT DISTINCT

R

A	В
a	1
a	2
b	1

SELECT COUNT(r1.A) AS Total FROM R r1, R r2



SELECT COUNT(DISTINCT r1.A) AS Total FROM R r1, R r2



Total

2

Simulating COUNT with SUM

 The following SQL query uses the SUM aggregate function to determine the size of R, provided R ≠ Ø

```
SELECT SUM(1) FROM R r;
```

• The bag that is generated by the query is {{1,...,1}} containing as many 1's as there are tuples r in R

Example

- Consider the relation R
- |R| = 3

A	
a	
b	
С	

- COUNT applies to {{a,b,c}}
- SUM applies to {{1, 1, 1}}
- If you write the SQL query

the result will be $SUM(\{\{2,2,2\}\}) = 6$.

Caveat: empty collection

Consider the relation R

A

- |R| = 0
- COUNT applied to {} gives 0
- SUM applied to {} gives NULL

SELECT COUNT(1) FROM R r;



0

SELECT SUM(1) FROM R r;



NULL

MIN and MAX aggregate functions

- MIN returns the smallest data item in the bag to which it applies.
- MAX returns the largest data item in the bag to which it applies.
- Data items can come from any ordered basic domain: integer, float, text
- A more general MIN function can be simulated using <= ALL

```
SELECT DISTINCT r.A1,...,r.An

FROM R r

WHERE (r.A1,...,r.An) <= ALL (SELECT r1.A1,...,r1.An

FROM R r1);
```

- MAX can be simulated using >= ALL
- However, there is a problem if MIN (MAX) is applied to an empty set.

CAVEAT: aggregate functions on empty set

- Except for COUNT, SQL aggregate functions return a NULL value when applied to an empty set (or bag).
- Assume R is the empty relation
- Then SELECT MIN(r.A) AS smallest FROM R r
 returns the relation smallest
 NULL
- Howeverselect r.A AS smallest FROM R r WHERE r.A <= ALL(SELECT r1.A FROM R r1)
 returns the empty relation

Partitioning and counting

 "Determine for each student the number of courses taken by that student."

Enroll

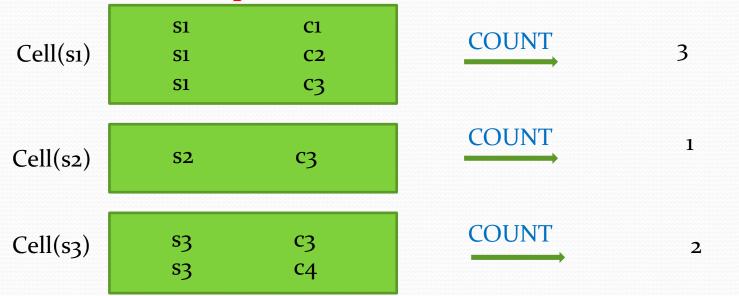
Sid	Cno
S1	C1
S2	C 1
S1	C2
s 3	c 3
s 3	C4
S1	c 3



Sid	No_Courses
S1	3
S2	1
s 3	2

Partition and map count function

- (1) First, Partition the Enroll table into cells (blocks) wherein each cell contains all the tuples that have a common sid value.
- (2) Next, Map the COUNT function over these cells.



Partition and map COUNT in SQL

- (1) The GROUP BY map COUNT method
- (2) The user-defined COUNT FUNCTION method
- (3) The SELECT COUNT-expression method

The GROUP BY map COUNT method

```
Map COUNT phase → SELECT E.Sid, COUNT(*) AS No_Courses FROM Enroll E

Partition phase → GROUP BY(E.Sid)
```

Partition phase: the GROUP BY operator places each tuple E into the cell identified by its E.Sid value

Map COUNT phase: the COUNT function is mapped over the cells identified by the different possible E.sid values

The user-defined COUNT FUNCTION method

 "Define a function with input parameter a student sid and as output the number of courses taken by that student.

```
CREATE FUNCTION NumberOfCourses (s TEXT) RETURNS bigint
AS $$

SELECT COUNT(*)

FROM Enroll E

WHERE E.Sid = s;
$$ LANGUAGE SQL;
```

Then execute the SQL query

Student

Sid \$1 \$2 \$3 \$4

Enroll

Lilion	
Sid	Cno
S1	C1
S1	C2
S1	c3
S2	c3
s 3	c3
s ₃	C 4

SELECT S.Sid, NumberOfCourses(S.Sid) AS No_Courses FROM Student S;

Sid	No_Courses
S1	3
S2	1
s 3	2
S 4	0

student s4 takes no courses

The SELECT COUNT-expression method

SELECT S.sid, (SELECT COUNT(E.Cno) AS NumberCourses
FROM Enroll E
WHERE E.Sid = S.Sid)
Partition phase
FROM Student S

- Observe that the subquery identified by S.Sid appears in the outer SELECT clause.
- The COUNT of the result of this subquery is then delivered as a value in the outer SELECT clause.
- Notice that this expression must appear between parentheses.
- The output of this query is the same as that on the previous slide.

Example query

- "Find the sid of each student who take the most courses."
- Using the GROUP BY method:

WITH

```
NumberOfCoursesbyStudent AS (SELECT E.Sid, COUNT(E.Cno) AS NumberOfCourses
FROM Enroll E
GROUP BY(E.Sid))
```

```
SELECT P.sid
FROM NumberOfCoursesbyStudent P
WHERE P.NumberOfCourses >= ALL (SELECT P1.NumberOfCourses
FROM NumberOfCoursesbyStudent P1);
```

Example query

- "Find the sid of each student who takes the most courses."
- Using the COUNT FUNCTION method:

Example query

- "Find the sid of each student who takes the most courses."
- Using the COUNT expression method:

```
SELECT S.Sid
FROM Student S
WHERE (SELECT COUNT(E.cno)
FROM Enroll E
WHERE E.sid = S.sid) >= ALL (SELECT (SELECT COUNT(E.cno))
FROM Enroll E
WHERE E.sid = S1.sid)
FROM Student S1);
```

The COUNT-bug of GROUP BY

• The result of the following 2 queries is the same. Notice that there is a bug since, if a student sid takes no courses, then (sid, 0) does not appear in the output.

SELECT E.Sid, COUNT(E.Cno) FROM Enroll E GROUP BY (E.Sid) SELECT S.Sid, Count(E.Cno) FROM Student S, Enroll E WHERE S.Sid = E.Sid GROUP BY(S.Sid)

• These two queries give the same result and exhibit the COUNT bug: the tuple (s4,0) does not appear in the result.

Fixing the COUNT-bug

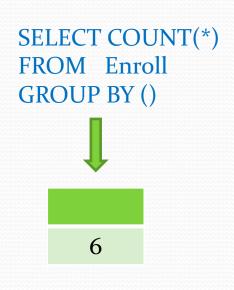
• To fix the COUNT-bug we need to add the (s,0) pair if student with sid s takes no courses. This can be done with the UNION operator.

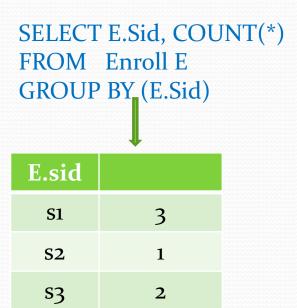
```
(SELECT E.Sid, COUNT(E.Cno) AS No_Courses
FROM Enroll E
GROUP BY (E.Sid))
UNION
(SELECT S.Sid, 0 AS No_Courses
FROM Student S
WHERE S.Sid NOT IN (SELECT E.Sid
FROM Enroll E))
```

Partitioning on different dimensions

Enroll

Sid	Cno
S 1	C1
S 1	C2
S 1	c3
S2	c3
s 3	c 3
s 3	C4





SELECT E.Sid, E.Cno, COUNT(*)
FROM Enroll E
GROUP BY (E.Sid, E.Cno)

E.Sid	E.Cno	
S 1	C1	1
S 1	C2	1
S 1	c3	1
S2	c3	1
s 3	c3	1
s 3	C4	1

What can appear in the GROUP BY clause?

• Answer: any valid expression over the tuples in the FROM clause.

```
SELECT ...
FROM R1 t1,...,Rn tn
WHERE ...
GROUP BY(expression(t1,...,tn))
```

 Partition: there will be as many cells in the partition as there are different values for expression(t1,...,tn)

Example: expressions in GROUP BY

S

X	Y
2	3
1	3
2	1
0	3

SELECT s.x + s.y AS sum, COUNT(*) as cell_size

FROM S s GROUP BY (s.x + s.y)

Notice that there are 4 tuples in R assigned to variable s but only 3 different s.x+s.y values: 3, 4, 5. Thus there are only 3 cells in the partition.

sum	cell_size
3	2
4	1
5	1

Example: expression in GROUP BY

Person

Pid	Age
рі	10
p 2	9
р3	12
р3	9

SELECT p.age > 10 AS OlderThanTen, COUNT(*)

FROM Person p

GROUP BY (p.age > 10);



OlderThanTen	Count
f	3
t	1

Restrictions on SELECT clause in GROUP BY query

 In a GROUP BY query, the SELECT clause may only contain aggregate expressions that returns a single value for each cell of the partition induced by the GROUP BY clause.

```
SELECT AggregateExpression(cell(expression(t1,...,tn)), ...
FROM R t1,...,tn
WHERE condition(t1,...,tn)
GROUP BY (expression(t1,...,tn))
```

Aggregate expressions in SELECT

elause

S

X	Y	
2	3	
1	3	
2	1	
0	3	

sum	sum_of_products
3	2 = (2 * 1 + 0 * 3)
4	3 = (1*3)
5	6 = (2*3)

Aggregate expressions in SELECT clause

 The following query will raise an error since s.x is not necessarily unique in a cell defined by s.x+s.y values

S

X	Y
2	3
1	3
2	1
0	3

SELECT s.x FROM S s GROUP BY (s.x+s.y)

Use case: some simple statistics

- Consider an experiment wherein we conduct a set of trials wherein in each trial we throw a pair of dice
- Further consider the random variable that returns, for each trial, the sum of the values that appear on the two dice
- We then want to determine the frequency of this sum to be a value between 2 and 12.
- Finally, we want to compute the expected value of the random variable.

Trials and random variable

Trials

Tid	Dice 1	Dice 2
t1	1	3
t2	2	3
t3	1	1
t4	1	6
t5	2	5
t6	1	6
t7	6	5
t8	6	1
t9	4	1

Random Variable

	10111
Tid	Dice 1 + Dice 2
tı	4
t2	5
t3	2
t4	7
t ₅	7
t6	7
t7	11
t8	7
t9	5

Notice that some Dice 1 + Dice 2 values occur more frequently than others

Frequency

 Suppose that we now want to determine the number of trials that have the same random variable value

SELECT t.Dice1 + t.Dice2 AS RV, COUNT(t.Tid) AS Frequency

FROM Trials t

GROUP BY (t.Dice1 + t.Dice2)

Notice that we get different frequencies across the different t.Dice1 + t.Dice2 values

RV	Frequency
2	1
4	1
5	2
7	4
11	1

Expected value of the random variable Dice1+Dice2

•
$$2 * \frac{1}{9} + 4 * \frac{1}{9} + 5 * \frac{2}{9} + 7 * \frac{4}{9} + 11 * \frac{1}{9} = 6.11...$$

- The theoretical value is 7
- We can get this expected value as follows:

```
SELECT SUM(Q.RV * Q.NTrials)/(SELECT COUNT(*)
FROM Trials) AS Expectation
FROM (SELECT t.Dice1 + t.Dice2 AS RV,
COUNT(t.Tid) AS NTrials
FROM Trials t
GROUP BY(t.Dice1 + t.Dice2)) AS Q
```

Expectation

6.11

The HAVING clause in GROUP BY queries

- The HAVING clause in a GROUP BY selects those cells from the partition induced by the GROUP BY clause that satisfy an Aggregate Condition.
- Only those cells are passed onto the SELECT clause.

```
SELECT AggregateExpression(Cell(expression(t1,...,tn)), ...
FROM R1 t1, ..., R tn
WHERE condition(t1,...,tn)
GROUP BY (expression(t1,...,tn))
HAVING AggregateCondition(Cell(expression(t1,...,tn)))
```

Example: HAVING clause

• "For each student who majors in CS determine the number of courses taken by that student, provided that this number is at least 2."

```
SELECT E.Sid, COUNT(E.Cno)

FROM Enroll E, Student S

WHERE E.Sid = S.Sid AND S.Major = 'CS'

GROUP BY (E.Sid)

HAVING COUNT(E.Cno) ≥ 2;
```

Simulating HAVING clause with user-defined functions in WHERE

- "For each student who majors in CS, determine the number of courses taken by that student, provided that this number is at least 3."
- The HAVING condition can be simulated in the WHERE clause with user-defined functions.

```
SELECT S.Sid AS Sid, NumberOfCourses(S.Sid)

FROM Student S

WHERE S.major = 'CS' AND NumberOfCourses(S.Sid) ≥ 3
```

Spreadsheet (Data Cube)

Sid	Cno
S1	C1
S1	C2
S1	с3
S2	C1
S2	C2
s ₃	C2
S 4	C1

	c1	c2	c3	sum(sid)	
s1	1	1	1	3	
s2	1	1	0	2	
s3	0	1	0	1	
s4	1	0	0	1	
sum(cno)	3	3	1		
				7	sum()

GROUPING sets

• It may be desirable to simultaneously generate different partitions and then apply an aggregation

Count

2

1

3

2

This is supported in SQL via GROUPING sets.

Sid	Cno	Sid	Cno
S1	C 1		
S1	C2	S1	*
51	CZ	S2	*
S2	C1	s3	*
S2	C2	*	C 1
		*	C2
s 3	C1		

SELECT Sid, Cno, COUNT(*)
FROM Enroll
GROUP BY
GROUPING SETS((Sid),(Cno))

CUBE operation

SELECT Sid, Cno, COUNT(*)

FROM Enroll

GROUP BY GROUPING SETS

((Sid,Cno),(Sid),(Cno),())

SELECT Sid, Cno, COUNT(*)
FROM Enroll
GROUP BY CUBE(Sid,Cno)

Sid	Cno	Sum
S1	C1	1
S 1	C2	1
S 1	с3	1
S2	C1	1
S2	C2	1
s ₃	C2	1
S 4	C1	1
S1	NULL	3
S2	NULL	2
s ₃	NULL	1
S 4	NULL	1
NULL	C1	3
NULL	C2	3
NULL	c3	1
NULL	NULL	7

WINDOW functions

- A window function performs a calculation across a set of tuples that are somehow related to the current tuple
- This is comparable to the type of calculation that can be done with an aggregate function
- But unlike regular aggregate functions, use of a window function does not cause tuples to become grouped into a single output tuple — the tuples retain their separate identities

Product

Name	Туре	Price
bag	accessory	30
footliner	socks	10
slippers	housewear	15
leggings	socks	7
pajamas	houseware	25
necklace	accessory	7
hat	accessory	15
watch	accessory	15

"Associate with each product the average price of all the products of that product's type."

SELECT name, type, price, AVG(price) OVER (PARTITION BY type) FROM product;

name	type	Price	Avg
bag	accessory	30	16.75
necklace	accessory	7	16.75
hat	accessory	15	16.75
watch	accessory	15	16.75
pijamas	housewear	25	20
slippers	housewear	15	20
footliner	socks	10	8.5
leggings	socks	7	8.5

Equivalent query

```
SELECT p.name, p.type, p.price,
(SELECT AVG(p1.price)
FROM product p1
WHERE p1.type = p.type)
FROM product p;
```

List the rank order of the price of each product among all the tuples of its type

SELECT name, type, price, rank() OVER (PARTITION BY type ORDER BY price) FROM Product;

name	type	price	Rank
necklace	accessory	7	1
watch	accessory	15	2
hat	accessory	15	2
bag	accessory	30	4
slippers	housewear	15	1
pijamas	housewear	25	2
leggings	sock	7	1
footliner	sock	10	2

Equivalent query

```
SELECT p.name, p.type, p.price,
(SELECT COUNT(1)

FROM product p1

WHERE p1.type = p.type AND

p1.price < p.price) + 1

FROM product p;
```