Database Management
Aggregate Functions

By: Meyer Tanuan (2022), Glenn Paulley (2015), John Mckay (2011)

Aggregate Functions

- Aggregate functions are used to count rows and summarize (typically) numeric data in a database
- Aggregate functions can appear in:
  - the column list of a SELECT statement
  - A HAVING clause (later in this unit)
  - An ORDER BY clause
- Aggregate functions differ from scalar functions
  - Scalar functions (e.g. ABS or SUBSTRING): one value in, one value returned
  - Aggregate functions: many values in, one value returned

The five basic aggregate functions

- Basic aggregate functions:
  - COUNT
  - AVG
  - SUM
  - MIN
  - MAX
- Each supports the ALL or DISTINCT keyword
  - ALL – compute the function over all non-NULL input values
    - AVG(ALL X)
  - DISTINCT – compute the result considering only unique non-NULL values
    - COUNT(DISTINCT X)

Aggregate Functions: COUNT

- COUNT always returns a value
  - COUNT(*) will return 0 (zero) if the input is empty
- All aggregate functions except COUNT(*) ignore NULL values in their input
  - For example, AVG(X) considers only non-null values of X
  - Aggregation functions other than COUNT will return NULL if they have all NULL values in their input

COUNT()

- COUNT(*) returns a count of the rows in a table
  - Result not related to nullability of any expression – merely a count of the rows in the input
- COUNT() always returns a value
  - If the input is empty, the result of COUNT(*) is 0 (zero)
- COUNT(expression)
  - Counts the number of non-null values in the input
- COUNT(DISTINCT expression)
  - Counts the number of distinct (duplicate-free) values

COUNT(*)

- COUNT(*) counts all rows unconditionally

```
SELECT COUNT( * ) AS [COUNT( * )]
FROM Person
```

```
COUNT(  *  )
------------
434
```

COUNT(expression)

- COUNT(expression) counts all rows with non-null values of expression

SELECT COUNT(fax) AS [COUNT(fax)]

FROM Employee

```
COUNT( fax )
---------------
4
```

COUNT(ALL)

- COUNT(ALL column) counts all rows with non-null values for column
  - ALL is the default and is usually omitted

SELECT COUNT(ALL fax) AS [COUNT(ALL fax)]

FROM Employee

```
COUNT( ALL fax )
--------------------
4
```

COUNT(DISTINCT) with WHERE

- Naturally, you can use a WHERE clause to restrict the rows being counted

```
SELECT COUNT( DISTINCT fax ) AS [COUNT( DISTINCT fax ) with WHERE]
FROM Employee
WHERE schoolCode = 'TAP' ;
```

```
COUNT( DISTINCT fax ) with WHERE
--------------------------------
1
```

AVG() – Compute the average

- AVG() is used to compute the average value of an expression

```
SELECT '$' + CONVERT(CHAR(7), CAST(AVG(amount) AS money),1)
    AS 'Average amount'
FROM InvoiceItem
```

```
Average amount
--------------
$ 457.94
```

MIN() – Find the minimum value

- The MIN() aggregate function is used to find the minimum value from a set of values

```
SELECT '$' + CONVERT(CHAR(7), CAST(MIN(amount) AS money),1)
    AS [Minimum Price]
FROM InvoiceItem
```

```
Minimum Price
-------------
$ 9.00
```

MAX() – Find the maximum value

- The MAX() aggregate function is used to find the maximum value from a set of values

```
SELECT '$' + CONVERT(CHAR(9), CAST(MAX(amount) AS money),1)
    AS [Maximum Price]
FROM InvoiceItem
```

```
Maximum Price
-------------
$ 3,380.00
```

CONESTOGA
Connect Life and Learning

SUM() – Compute the sum

- The SUM(expression) aggregate function is used to compute the sum from all non-null values of expression

```
SELECT '$' + CONVERT(CHAR(9), CAST(SUM(amount) AS money),1)
    AS [Sum of all Items]
FROM InvoiceItem
```

```
Sum of all Items
----------------
$ 5,037.32
```

CONESTOGA
Connect Life and Learning

Aggregates with WHERE

- You can use a WHERE clause to restrict the input rows to the computation of AVG(), MIN(), MAX(), and SUM() as well

```
SELECT '$' + CONVERT(CHAR(10), CAST(AVG(amount) AS money),1)
    AS [Avg of all items]
FROM InvoiceItem
WHERE item LIKE '%CSI%'
```

```
Avg of all items
----------------
$ 210.63
```

GROUP BY and HAVING

CONESTOGA
Connect Life and Learning

GROUP BY

- GROUP BY, in combination with aggregate functions, can be used to group rows and summarize the results of aggregate functions for each group
- Once GROUP BY is used, the expressions in a query's SELECT list are restricted to:
  - Expressions specified in the GROUP BY clause
  - Aggregate functions (or other computations based on their results)

What does GROUP BY do?

- GROUP BY partitions the input rows (FROM and WHERE clauses) into groups that have the identical set of values for the GROUP BY expressions specified in the query
- Once all of the input rows are partitioned, the server then computes the result of the aggregate functions for each partition
- Each partition then generates a single row in the output

Using GROUP BY

```
SELECT studentNumber
     , SUM(amount)
FROM Payment
GROUP BY studentNumber
```

```
studentNumber
------------ --------------------
1114453      1000.00
1335314      10000.00
2286425      1000.00
2642726      13159.00
2826147      1000.00
3244449      5291.44
3868247      13159.00
6644710      1159.00
7252620      5291.44
7677479      2000.00
7681752      1000.00
7826662      6791.44
8431710      5291.44
8588766      1000.00
8866782      3000.00
```

What forms a GROUP?

- Groups are formed from unique values of the combination of grouping columns in the GROUP BY clause
  - For the purposes of comparing column or expression values, NULL is semantically equivalent to NULL, as in SELECT DISTINCT
- GROUP BY X
  - Groups formed based on unique values of X
- GROUP BY X, Y
  - Groups formed based on unique combinations of values of X and Y for the same row

Empty input with GROUP BY

- If a query's intermediate result is empty, and that query contains GROUP BY, then the query also returns the *empty set* (i.e., 0 rows affected)

```
SELECT sessionCode, AVG(amount) AS AverageAmount
FROM Invoice
WHERE amount > 0
GROUP BY sessionCode
```

```
sessionCode AverageAmount
----------- --------------------
(0 rows affected)
```

Empty input without GROUP BY

- For queries that involve aggregation but don’t have a GROUP BY clause, then the query does not return an empty set (i.e., 1 row affected)

```
SELECT AVG(amount) AS AverageAmount
FROM Invoice
WHERE amount > 0
```

```
AverageAmount
---------------------
NULL
(1 row affected)
```

HAVING

- HAVING can be used to limit the results that appear in groups created with GROUP BY
- HAVING is similar to a WHERE clause
  - WHERE restricts the rows created by the FROM clause
  - HAVING restricts the groups created from the GROUP BY clause
- HAVING is often used with aggregate functions
  - You cannot use an aggregate function in a WHERE clause because at the point of applying the WHERE conditions, the groups have yet to be formed

GROUP BY and HAVING

- Determine the SUM of payments for specific students but only for students who have paid over more than one payment

```
SELECT studentNumber
    , '$' + CONVERT( CHAR(12), CAST( SUM(amount) AS money ), 1 )
    AS [Invoice Total]
FROM Payment
GROUP BY studentNumber
HAVING COUNT(*) > 1
```

GROUP BY and HAVING Result

studentNumber	Invoice Total
2642726	$ 13,159.00
7677479	$ 2,000.00
7826662	$ 6,791.44
8431710	$ 5,291.44
8866782	$ 3,000.00

ORDER OF CLAUSES

- If a query contains a WHERE clause, you must put the WHERE clause before GROUP BY and HAVING
- ORDER BY follows GROUP BY and the optional HAVING clause

```
SELECT studentNumber
    , '$' + CONVERT( CHAR(12), CAST( SUM(amount) AS money ), 1 )
    AS [Invoice Total]
FROM Payment
WHERE studentNumber <> 8431710
GROUP BY studentNumber
HAVING COUNT(*) > 1
ORDER BY 2 DESC;
```

GROUP BY, HAVING, WHERE and ORDER BY Result

studentNumber	Invoice Total
2642726	$ 13,159.00
7826662	$ 6,791.44
8866782	$ 3,000.00
7677479	$ 2,000.00

Limitation: Expressions with GROUP BY

- One cannot specify an expression in a query's SELECT list that is NOT an aggregate function unless the identical expression appears in the query's GROUP BY clause
- For example:
  - SELECT X, SUM(Y) FROM T GROUP BY X – permitted
  - SELECT X, SUM(Y) FROM T – illegal
  - SELECT X, Y, SUM(Z) FROM T GROUP BY X – illegal
  - SELECT X, SUM(Z) FROM T GROUP BY X, Y – permitted

Example: GROUP BY WITH ROLLUP

- The ROLLUP operator in a query adds a final summary row to the result (SQL Server 2008 or later)
- Select all persons grouping them by first letter of their last name and provide a count of the number of persons with the same letter of their last name (e.g., 24 unique first letters)

```
SELECT LEFT(LastName,1), COUNT(LEFT(LastName,1))
FROM Person
GROUP BY LEFT(LastName,1) WITH ROLLUP
```

Example: Add search condition (7 letters dropped)

- Select all international students grouping them by first letter of their last name and provide a count of the number of students with each first letter (e.g., 17 unique first letters)

```
SELECT LEFT(LastName,1), COUNT(LEFT(LastName,1))
FROM Student s JOIN Person p
    ON s.number = p.number
WHERE isInternational=1
GROUP BY (LEFT(LastName,1)) WITH ROLLUP
```

Example: OUTER JOIN and join table expression

- To show all the original 24 unique first letters (some with zero)
  - RIGHT OUTER JOIN
  - Use join table expression (move search condition from WHERE clause to ON clause)

```
SELECT LEFT(LastName,1), COUNT(s.number)
FROM Student s RIGHT OUTER JOIN Person p
    ON (s.number = p.number AND isInternational=1)
GROUP BY (LEFT(LastName,1)) WITH ROLLUP
```