Module 9

Grouping and Aggregating Data

Zyead Ahmed.
Aspiring To Learning Data Engineer.

Module Overview

- Using Aggregate Functions
- Using the GROUP BY Clause
- Filtering Groups with HAVING

Lesson 1: Using Aggregate Functions

- Working with Aggregate Functions
- Built-In Aggregate Functions
- Using DISTINCT with Aggregate Functions
- Using Aggregate Functions with NULL
- Demonstration: Using Aggregate Functions

Working with Aggregate Functions

- Aggregate functions:
  - Return a scalar value (with no column name)
  - Ignore NULLs except in COUNT(*)
  - Can be used in
    - SELECT, HAVING, and ORDER BY clauses
  - Frequently used with GROUP BY clause

```
SELECT      AVG(unitprice) AS avg_price,
            MIN(qty)AS min_qty,
            MAX(discount) AS max_discount
FROM Sales.OrderDetails;
```

```
avg_price min_qty max_discount
--------- ------- ------------
26.2185   1         0.250
```

Built-In Aggregate Functions

Common

- SUM
- MIN
- MAX
- AVG
- COUNT
- COUNT_BIG

Statistical

- STDEV
- STDEVP
- VAR
- VARP

Other

- CHECKSUM_AGG
- GROUPING
- GROUPING_ID

- This lesson will only cover common aggregate functions. For more information on other built-in aggregate functions, see Books Online.

Using DISTINCT with Aggregate Functions

- Use DISTINCT with aggregate functions to summarize only unique values
- DISTINCT aggregates eliminate duplicate values, not rows (unlike SELECT DISTINCT)
- Compare (with partial results):

```
SELECT empid, YEAR(orderdate) AS orderyear,
      COUNT(custid) AS all_custs,
      COUNT(DISTINCT custid) AS unique_custs
FROM Sales.Orders
GROUP BY empid, YEAR(orderdate);
```

```
empid       orderyear   all_custs   unique_custs
----------- ----------- ----------- ------------
1           2006        26          22
1           2007        55          40
1           2008        42          32
2           2006        16          15
```

Using Aggregate Functions with NULL

- Most aggregate functions ignore NULL
  - COUNT(<column>) ignores NULL
  - COUNT(*) counts all rows
- NULL may produce incorrect results (such as use of AVG)
- Use ISNULL or COALESCE to replace NULLs before aggregating

```
SELECT
      AVG(c2) AS AvgWithNULLs,
      AVG(COALESCE(c2,0)) AS AvgWithNULLReplace
FROM dbo.t2;
```

Demonstration: Using Aggregate Functions

In this demonstration, you will see how to:

- Use built-in aggregate functions

Lesson 2: Using the GROUP BY Clause

- Using the GROUP BY Clause
- GROUP BY and the Logical Order of Operations
- GROUP BY Workflow
- Using GROUP BY with Aggregate Functions
- Demonstration: Using GROUP BY

Using the GROUP BY Clause

- GROUP BY creates groups for output rows, according to a unique combination of values specified in the GROUP BY clause

```
SELECT <select_list>
FROM <table_source>
WHERE <search_condition>
GROUP BY <group_by_list>;
```

- GROUP BY calculates a summary value for aggregate functions in subsequent phases

```
SELECT empid, COUNT(*) AS cnt
FROM Sales.Orders
GROUP BY empid;
```

- Detail rows are “lost” after GROUP BY clause is processed

GROUP BY and the Logical Order of Operations

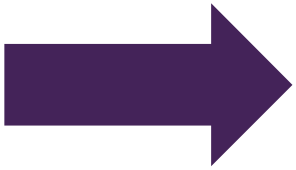
Logical Order	Phase	Comments
5	SELECT	
1	FROM	
2	WHERE	
3	GROUP BY	Creates groups
4	HAVING	Operates on groups
6	ORDER BY	

- If a query uses GROUP BY, all subsequent phases operate on the groups, not source rows
- HAVING, SELECT, and ORDER BY must return a single value per group
- All columns in SELECT, HAVING, and ORDER BY must appear in GROUP BY clause or be inputs to aggregate expressions

GROUP BY Workflow

```
SELECT orderid, empid, custid
FROM Sales.Orders;
```

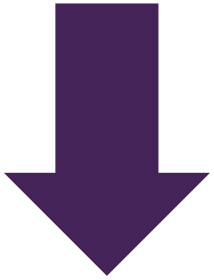
orderid	empid	custid
10643	6	1
10692	4	1
10926	4	2
10625	3	2
10365	3	3



```
WHERE
custid IN(1,2)
```

orderid	empid	custid
10643	6	1
10692	4	1
10926	4	2
10625	3	2

```
GROUP BY empid
```



empid	COUNT(*)
6	1
4	2
3	1

Using GROUP BY with Aggregate Functions

- Aggregate functions are commonly used in SELECT clause, summarize per group:

```
SELECT custid, COUNT(*) AS cnt
FROM Sales.Orders
GROUP BY custid;
```

- Aggregate functions may refer to any columns, not just those in GROUP BY clause

```
SELECT productid, MAX(qty) AS largest_order
FROM Sales.OrderDetails
GROUP BY productid;
```

Demonstration: Using GROUP BY

In this demonstration, you will see how to:

- Use the GROUP BY clause

Lesson 3: Filtering Groups with HAVING

- Filtering Grouped Data Using the HAVING Clause
- Compare HAVING to WHERE
- Demonstration: Filtering Groups with HAVING

Filtering Grouped Data Using the HAVING Clause

- HAVING clause provides a search condition that each group must satisfy
- HAVING clause is processed after GROUP BY

```
SELECT custid, COUNT(*) AS count_orders
FROM Sales.Orders
GROUP BY custid
HAVING COUNT(*) > 10;
```

Compare HAVING to WHERE

- Using a COUNT(*) expression in HAVING clause is useful to solve common business problems:
- Show only customers that have placed more than one order:

```
SELECT c.custid, COUNT(*) AS cnt
FROM Sales.Customers AS c
    JOIN Sales.Orders AS o ON c.custid = o.custid
GROUP BY c.custid
HAVING COUNT(*) > 1;
```

- Show only products that appear on 10 or more orders:

```
SELECT p.productid, COUNT(*) AS cnt
FROM Production.Products AS p JOIN Sales.OrderDetails
    AS od ON p.productid = od.productid
GROUP BY p.productid
HAVING COUNT(*) >= 10;
```

Demonstration: Filtering Groups with HAVING

In this demonstration, you will see how to:

- Filter grouped data using the HAVING clause