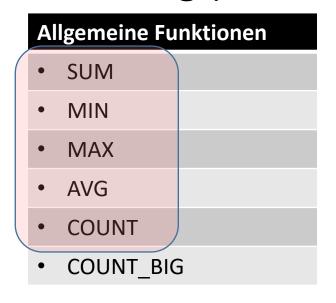
Aggregatfunktionen und Gruppierung

Aggregatfunktionen

- Geben einen Skalarwert (ohne Spaltennamen) zurück
- Ignorieren NULL-Werte (außer in COUNT(*))
- In SELECT-, HAVING- und ORDER BY-Klauseln einsetzbar
- Häufig (notwendigerweise) mit der GROUP BY-Klausel genutzt



Statistische Funktionen			
•	STDEV		
•	STDEVP		
•	VAR		
•	VARP		

Andere			
•	CHECKSUM_AGG		
•	GROUPING		
•	GROUPING_ID		

COUNT

- COUNT(<Spalte>) ignoriert NULL-Werte
- COUNT(*) zählt alle Zeilen
- Beispiel:

```
SELECT COUNT(DISTINCT City)
FROM dbo.Customers
WHERE City LIKE '%an%';
```

AVG

"Average": Durchschnitt

SUM

Summe

MIN/MAX

Die meisten Aggregatfunktionen ignorieren NULL-Werte, sie können aber zu falschen Ergebnissen führen (AVG)! => ISNULL oder COALESCE verwenden.

Minimum und Maximum

Gruppieren von Werten

- Verwenden der GROUP BY-Klausel für Ausgabezeilengruppen
- Filtern mit HAVING

Klausel	Ausdruck	Nutzen
SELECT	<spaltenliste></spaltenliste>	Legt Spaltenliste für Ergebnismenge fest
FROM	<tabellenname></tabellenname>	Sagt aus, welche Tabellen betrachtet werden
WHERE	<filterausdruck></filterausdruck>	Filtert unerwünschte Daten heraus
GROUP BY	<gruppierungskriterium></gruppierungskriterium>	Gemeinsame Spaltenwerte zusammenfassen
HAVING	<filterausdruck></filterausdruck>	Filtert unerwünschte Gruppen hinaus
ORDER BY	<sortierreihenfolge></sortierreihenfolge>	Sortiert nach Spalte(n)

GROUP BY I

- Wird GROUP BY in einer Abfrage verwendet, erfolgen alle folgenden Schritte im Zusammenhang mit den Gruppen und nicht mit den Quellzeilen
- HAVING, SELECT und ORDER BY müssen pro Gruppe einen einzelnen Wert zurückgeben
- Alle Spalten in SELECT, HAVING und ORDER BY müssen in der GROUP BY-Klausel vorkommen oder Eingaben für die Aggregatausdrücke sein

GROUP BY II

OrderID	EmployeeID	CustomerID
120	6	4
325	4	4
870	4	2
422	2	2
952	6	3

Workflow

WHERE CustomerID IN(2,4)

OrderID	EmployeeID	CustomerID
120	6	4
325	4	4
870	4	2
422	2	2

SELECT EmployeeID, COUNT(*) FROM dbo.Orders WHERE CustomerID IN (2,4) GROUP BY EmployeeID;

EmployeeID	
6	1
4	2
2	1

EmployeeID 6 **EmployeeID 4**

SELECT EmployeeID, COUNT(*)

EmployeeID 2

GROUP BY EmployeeID

GROUP BY III

- Aggregatfunktionen typischerweise in SELECT-Klausel
- Fassen pro Gruppe zusammen
- Können auf alle Spalten verweisen, nicht nur auf die in der GROUP BY-Klausel

```
SELECT ProductID, MAX(Quantity) AS Anzahl
FROM [Order Details]
GROUP BY ProductID
ORDER BY Anzahl DESC;
```

GROUP BY IV



ID	Artikel	Artikelgruppe	Preis	Anzahl
1	Radio	Elektro	39,-	3
2	Seife	Pflege	1,29	11
3	Brot	Lebensmittel	1,88	14
4	PC	Elektro	465,90	3
5	Pizza	Lebensmittel	2,99	11
6	Deo	Pflege	3,49	3
7	Bier	Lebensmittel	0,75	21

SELECT Artikelgruppe, SUM(Preis * Anzahl) AS Wert FROM Artikel

GROUP BY Artikelgruppe;

SELECT Anzahl, SUM(Preis * Anzahl) AS Wert
FROM Artikel
GROUP BY Anzahl;

SELECT	Artikelgruppe,	SUM(Preis	*	Anzahl)	AS	Wert
FROM Ar	rtikel;					

	Anzahl	Wert
1	3	1525,17
2	11	47,08
3	14	26,32
4	21	15,75

geht

← geht nicht, da nicht gruppiert wurde!

HAVING I

- Enthält Filterbedingungen, welche jede Gruppe erfüllen muss
- Wird nach der GROUP BY-Klausel verarbeitet
- Kein HAVING ohne GROUP BY!

```
SELECT CustomerID, COUNT(*) AS AnzahlBestellungen
FROM dbo.Orders
GROUP BY CustomerID
HAVING COUNT(*) > 10;
```

HAVING II

COUNT(*) und HAVING für typische Unternehmensprobleme

Beispiel: nur Kunden mit mehr als einer Bestellung anzeigen

```
SELECT o.CustomerID, COUNT(*) AS Anzahl
FROM dbo.Orders AS o
GROUP BY o.CustomerID
HAVING COUNT(*) > 1;
```

Beispiel: nur Produkte anzeigen, die in mindestens 10 Bestellungen vorkommen

```
SELECT od.ProductID, COUNT(*) AS Anzahl
FROM dbo.OrderDetails AS od
GROUP BY od.ProductID
HAVING COUNT(*) >= 10;
```

Unterschied HAVING und WHERE

- WHERE filtert Zeilen bevor Gruppen erstellt werden
 - > Welche Zeilen dürfen überhaupt verwendet werden?

- HAVING filtert Gruppen nach Kriterien
 - Welche Gruppen werden weitergegeben?