SQL
Informationen filtern und gruppieren

Alle Kunden, die
im Land x wohnen

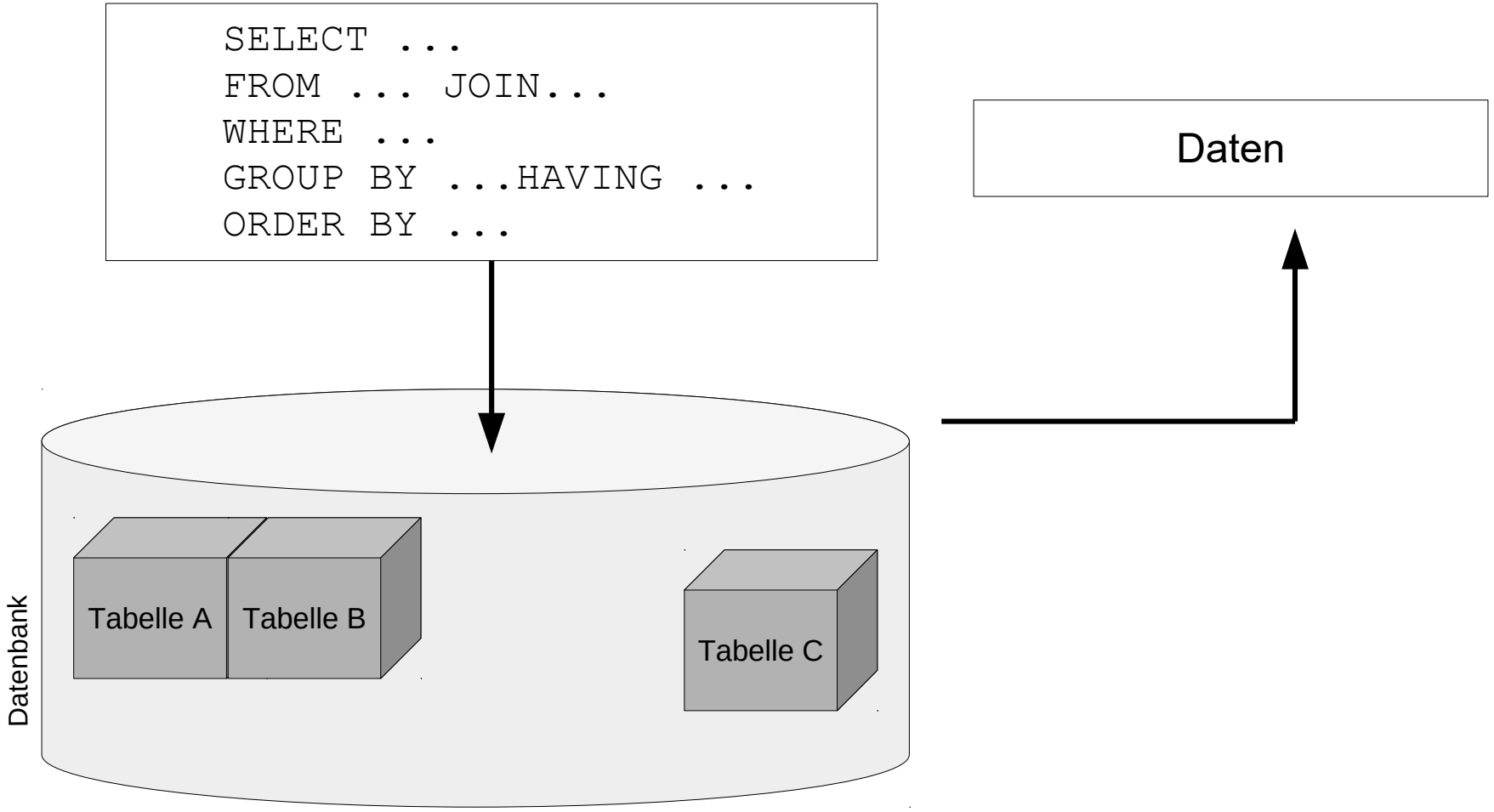
Alle Umsätze
größer
10.000, gruppiert nach den Städten

Bestellungen,
gruppiert nach den Kunden

Auswahlabfragen

- Beginn mit `SELECT`.
- Anzeige von allen oder ausgewählten Datenfeldern.
- Filterung der Daten mit Hilfe von Bedingungen.
- Gruppierung der Daten in Abhängigkeit von Bedingungen
- Aufsteigende oder absteigende Sortierung von Datenfeldern.

Arbeitsweise



Ergebnis einer Auswahlabfrage

- Speicherung in einer temporären Ergebnistabelle.
- Die Ergebnistabelle fasst die gewünschten Datenfelder zusammen. Die anzuzeigenden Datenfeldern können in verschiedenen Tabellen gespeichert sein.
- Die Daten werden in Abhängigkeit von Bedingungen gefiltert und gruppiert.

Informationstypen in SQL

- Alphanumerische Daten. Alle Zeichen eines Unicode-Zeichensatzes. Ziffern, Buchstaben, Satzzeichen etc.
- Numerische Daten. Ziffern aus denen neue Werte berechnet werden können.
- Datums- und Zeitwerte werden als alphanumerische Daten in SQLite dargestellt. In anderen Datenbanksystemen gibt es für diese Art von Daten eigene Datentypen.
- Binäre Daten, die von SQL nicht interpretiert werden können. Zum Beispiel Grafiken haben eine Struktur, die von SQL nicht interpretiert werden kann.

Filterung von Daten in SQL

```
SELECT [Feld], [Feld]

FROM [Tabelle]
INNER | LEFT OUTER JOIN [Tabelle]
ON [Primärschlüssel] = [Fremdschlüssel]

WHERE ([Bedingung])

ORDER BY [Feld] ASC|DESC, [Feld] ASC|DESC;
```

Beispiel

```
SELECT
    albums.Title,
    tracks.Milliseconds

FROM tracks
INNER JOIN albums
ON (tracks.AlbumId = albums.AlbumId)

WHERE (tracks.Milliseconds > 300000)

ORDER BY albums.Title, tracks.Milliseconds DESC;
```

Kriterien / Bedingungen

```
(Country Like 'Canada')
(Milliseconds > 300000)
(HireDate BETWEEN '2004-01-01' AND '2004-12-31')
```

- Ja / Nein-Fragen. Das Kriterium ist erfüllt (ja, wahr, true) oder nicht (nein, falsch, false).
- Wenn das Kriterium auf keinen Datensatz zutrifft, ist die Ergebnistabelle der SQL-Anweisung leer.
- Filtermuster, die mit Hilfe von Operatoren und Operanden zusammengestellt werden.
- Bedingungen können zu einem komplexen Kriterium verknüpft werden.

Operatoren

```
(Country Like 'Canada')
(Milliseconds > 300000)
(HireDate BETWEEN '2004-01-01' AND '2004-12-31')
(GenreID IN(7, 8, 12))
```

- Vergleichsoperatoren. Wie werden die Daten aus den Feldern mit dem Muster verglichen?
- Werte müssen innerhalb einer unteren und oberen Grenze liegen.
- Werte müssen Werte in einer Liste entsprechen.

Vergleichsoperatoren

ist ...	Operator
gleich	=
ungleich	<>
	!=
kleiner	<
kleiner gleich	<=
größer	>
größer gleich	>=

Operator „Like“

```
SELECT albums.Title,tracks.Milliseconds
FROM tracks
INNER JOIN albums
ON (tracks.AlbumId = albums.AlbumId)
WHERE (albums.Title LIKE 'H%')
ORDER BY albums.Title, tracks.Milliseconds DESC;
```

- Der Operator `LIKE` entspricht dem Gleichheitszeichen.
- Bei einem Vergleich mit dem Operator `LIKE` können einzelne Zeichen im Vergleichstext durch Wildcards ersetzt werden.
- Welche Wildcards genutzt werden können, ist abhängig vom gewählten Datenbanksystem.

Hinweise

- Zeichen werden in Abhängigkeit ihrer Zeichenkodierung verglichen.
- SQLite kann nur ASCII-Zeichen unabhängig von der Groß- und Kleinschreibung miteinander vergleichen.
- Unicode-Zeichen werden in Abhängigkeit der Groß- und Kleinschreibung verglichen.

Wildcards

```
WHERE(InvoiceDate LIKE '2009-01-0_ %');
```

- Wildcards können in beliebiger Kombination genutzt werden.
- Wildcards können an jeder beliebigen Position in dem Vergleichsmuster vorkommen.
- Das Prozentzeichen steht für kein, ein oder mehrere Zeichen.
- Der Unterstrich steht für ein beliebiges alphanumerisches oder numerisches Zeichen.

Logische Operatoren

ist ...	Operator
Negation	NOT([Bedingung])
UND	[Bedingung] AND [Bedingung]
ODER	[Bedingung] OR [Bedingung]

Sonstige Operatoren

ist ...	Operator
Liste von Vergleichswerten	[Feld] IN([Wert], [Wert], ...)
Zwischen … und ...	[Feld] BETWEEN [Wert] AND [Wert]

Operanden

```
(Country Like 'Canada')
(Milliseconds > 300000)
(HireDate BETWEEN '2004-01-01' AND '2004-12-31')
(GenreID IN(7, 8, 12))
```

- Feldnamen. Welche Daten sollen untersucht werden? Feldnamen werden links vom Operator genutzt.
- Statische Werte. Mit welchen Werten werden die Felder verglichen? Statische Werte werden als Literale bezeichnet. Statische Werte werden nur rechts vom Operator genutzt.

… vom Datentyp

- Kategorisierung von Daten in Abhängigkeit des Typs. Die Implementierung der verschiedenen Datentypen ist abhängig von dem genutzten Datenbanksystem.
- Wie können die Daten verarbeitet werden?
- Wie viel Platz wird für die Ablage der Informationen benötigt?

Storage Classes in SQLite

- SQLite hat keine Datentypen, sondern nur Storage Classes.
- Die Klassen beschreiben einen Datentyp allgemein.
- Informationen im Web: https://www.sqlite.org/datatype3.html.

Möglichkeiten

- `INTEGER`. Ganzzahl.
- `REAL`. Zahlen in diesem Format nähren sich immer einen Wert von doppelter Genauigkeit an. Gleitkommazahlen belegen 8 Byte (IEEE-Format).
- `TEXT`. String. Alphanumerische und numerische Zeichen im Format UTF-8, UTF-16.
- `BLOB`. Binary Large Object. Binäre Daten wie zum Beispiel Grafiken etc.
- `NULL`. Das Datenfeld hat einen undefinierten Wert.

Ganzzahlen

- Ganzzahlen sind vorzeichenbehaftet.
- Positive Ganzzahlen als Literale: 3, +13456 und so weiter.
- Negative Ganzzahlen als Literale: -5, -3456 und so weiter.
- Ein Datenfeld mit der Angabe `integer` oder `int` speichert in eine Ganzzahl in der Storage Class `INTEGER`.

Gleitkommazahl

- Zahlen mit Nachkommastellen. Als Dezimaltrennzeichen wird der Punkt genutzt.
- Zahlen, die sich einen bestimmten Wert nähren.
- Gleitkommazahlen sollten nicht in kaufmännischen Berechnungen genutzt werden.
- Zahlen wie zum Beispiel 3.5, 0.345667, 2.0e+24 und so weiter sind Gleitkommazahlen.
- Datenfelder mit der Angabe `float(12)` speichern Gleitkommazahlen in die Storage Class `REAL`.

Dezimalzahlen

- Zahlen mit Nachkommastellen. Als Dezimaltrennzeichen wird der Punkt genutzt.
- Zahlen, die für die Angabe von Preisen in Datenbanken genutzt werden.
- Zahlen wie zum Beispiel 3.55, 0.99 und so weiter sind Dezimalzahlen.
- Datenfelder mit der Angabe `numeric(10, 2)` speichern Dezimalzahlen in die Storage Class `REAL`.

Boolean

- Beantwortung von Ja- / Nein-Fragen.
- Interpretation in SQLite als Ganzzahl (`INTEGER`).
- Das Kriterium ist erfüllt: true, wahr, 1.
- Das Kriterium ist nicht erfüllt: false, falsch, 0.

Text

- Alphanumerische und numerische Zeichen.
- Zeichenketten.String.
- Als Text wird zum Beispiel die Länderbezeichnung „Canada“, die Postleitzahl „30159“ oder die Verpackungsangabe „24 - 12 oz bottles“ gespeichert.
- In Filterkriterien wird Text immer durch die Apostrophs begrenzt. Zum Beispiel: 'Canada', '30159'.
- Möglichkeiten: `nvarchar(160)`, `varchar(15)`, `text`.

Datums- und Zeitformate im SQL-Standard

- Datum (Date): '1968-01-09'. 'yyyy-mm-dd'
- Zeit (Time): '20:12:00 +01:00'. 'hh:mm:ss'. Die Stunden werden in einem 24-Stunden-Format angegeben. Eine Zeitverschiebung in Abhängigkeit der UTC (Universal Time Coordinate) kann angegeben werden. In diesem Beispiel wird die Sommerzeit berücksichtigt.
- Zeitstempel (Timestamp): '20:12:00.123 +01:00'. 'hh:mm:ss.dddd +|-hh:mm'. Die Bruchteile einer Sekunde wird mit Hilfe des Punktes an die Zeitangaben angehängt. Diese Angaben werden häufig für Log-Einträge genutzt.

Datums- und Zeitwerte in SQLite

- Eine Tabellenspalte mit dem Datentyp `datetime` oder `timestamp` wird in SQLite als ISO 8601 - String gespeichert.
- Der angegebene Datentyp wird in der Storage Class `TEXT` abgelegt.
- Datums- und Zeitangaben werden als Vergleichswerte in einer Bedingung in dem Typ „Text“ angegeben. Die Reihenfolge der Tages- und Monatsangaben ist abhängig von dem genutzten Format in dem zu filternden Datenfeld.

Beispiele für Operanden

- Die Angabe '1968-01-09 00:00:00' entspricht dem SQL-Standard. Das Datum wird in der Form yyyy-mm-dd angegeben.
- Die Angabe '8/14/1992 12:00:00 AM' entspricht nicht dem SQL-Standardformat. Das Datumsformat wird in der Form m/d/yyyy gespeichert. Bei einstelligen Monats- und Tagesangaben wird keine Null hinzugefügt. Zeitangaben werden im 12-Stunden-Format gespeichert.

Definierter Wert in einem Datenfeld

```
SELECT * FROM customers
WHERE (Company IS NOT NULL);
```

- Das Datenfeld `Company` ist (`IS`) definiert (`NOT NULL`).
- Die Information ist in dem Datensatz vorhanden. Das Attribut `Company` ist für das zu beschreibende Objekt gesetzt.
- In der Ergebnistabelle werden in diesem Beispiel alle Kunden angezeigt, denen eine Firma zugeordnet ist.

Undefinierter Wert in einem Datenfeld

```
SELECT * FROM customers
WHERE (Company IS NULL);
```

- Das Datenfeld `Company` ist (`IS`) nicht definiert (`NULL`).
- Die Information liegt für diesen Datensatz nicht vor. Ob der Attribut `Company` für das beschriebene Objekt existiert, ist nicht bekannt.
- SQLite zeigt in der Ergebnistabelle `NULL` an. Der Nutzer hat nie Eingaben in dieses Feld gemacht.

Leerer Wert

```
SELECT * FROM customers
WHERE (State = '');
```

- Das Datenfeld `State` ist leer (`''`).
- Die direkt aufeinanderfolgenden Apostrophen kennzeichnen eine leere Zeichenfolge. 0 oder 0.0 kennzeichnen bei Zahlen ein leeres Feld.
- Die Information ist momentan nicht vorhanden. Aber das Objekt besitzt dieses Attribut.
- SQLite zeigt in der Ergebnistabelle eine leere Zelle an. Der Nutzer hat in das Feld einen Wert eingetragen, aber diesen gelöscht.

Gruppierung von Daten in SQL

```
SELECT [Feld], [Feld]

FROM [Tabelle]
INNER | LEFT OUTER JOIN [Tabelle]
ON [Primärschlüssel] = [Fremdschlüssel]

WHERE ([Bedingung])

GROUP BY [Gruppenfeld]

ORDER BY [Feld] ASC|DESC, [Feld] ASC|DESC;
```

Beispiel

```
SELECT
  customers.Country, customers.City

FROM customers

GROUP BY customers.Country,customers.City

ORDER BY customers.Country,customers.City;
```

Funktionsweise

- Mit Hilfe des Schlüsselwortes `GROUP BY` werden beliebig viele Datenfelder in Abhängigkeit ihrer Informationen gruppiert.
- Daten werden in Abhängigkeit von einen oder mehreren Datenfelder zusammengefasst.
- Für jede Gruppe wird ein Label angezeigt.
- Verdichtung von großen Datenmengen.

Liste der zu gruppierenden Felder

```
SELECT customers.Country, customers.City

FROM customers

GROUP BY customers.Country,customers.City;
```

- Die Gruppierungsfelder werden durch ein Komma getrennt.
- Hinweis: Alle Felder in der Auswahlliste (`SELECT`) müssen zusammengefasst werden.

Wie werden die Felder gruppiert?

```
SELECT customers.Country, customers.City

FROM customers

GROUP BY customers.Country,customers.City;
```

- Die Datenfelder werden von links nach rechts gruppiert.
- Das erste Feld in der Liste wird zusammengefasst. In diesem Beispiel stellt das Feld `Country` die oberste und erste Gruppierungsebene dar.
- Das zweite Feld wird innerhalb der ersten Gruppierungsebene gruppiert. In diesem Beispiel wird das Feld `City` in Abhängigkeit des Feldes `Country` gruppiert.

Zusammenfassung von Daten

```
SELECT [Gruppenfeld], Aggregat([Feld])

FROM [Tabelle]
INNER | LEFT OUTER JOIN [Tabelle]
ON [Primärschlüssel] = [Fremdschlüssel]

WHERE ([Bedingung])

GROUP BY [Gruppenfeld]

ORDER BY [Feld] ASC|DESC, [Feld] ASC|DESC;
```

Beispiel

```
SELECT
  albums.Title,
  Sum(tracks.Milliseconds) AS LaengeAlbum

FROM tracks
INNER JOIN albums
ON (tracks.AlbumId = albums.AlbumId)

WHERE (tracks.Milliseconds > 300000)

GROUP BY albums.Title

ORDER BY Sum(tracks.Milliseconds) DESC;
```

Erläuterung

- Für die Gruppierung dürfen nur Datenfelder genutzt werden, die in der Datenquelle definiert sind.
- Gruppierungsfelder können angezeigt werden, müssen aber nicht. In dem Beispiel wird nach dem Feld `albums.Title` gruppiert. Die Werte des Feldes werden als Gruppenname angezeigt.
- Nicht-Gruppierungsfelder werden mit Hilfe von sogenannten Aggregatfunktionen zusammengefasst. Der Rückgabewert der Aggregatfunktion wird angezeigt. In diesem Beispiel wird das Feld `tracks.Milliseconds` für jedes Album summiert. Die Gesamtspieldauer eines Albums wird angezeigt.

Aggreatfunktionen

- Zusammenfassung von Datenfeldern in Abhängigkeit des zu gruppierenden Feldes.
- Einzelwerte in einer Gruppe werden zu einem Wert verdichtet. Durch die Verdichtung kommt es zu einem Informationsverlust.
- Berechnung der Gesamtanzahl, Summe, Mittelwert und so weiter über x Datensätze.

Arbeitsweise von Funktionen

```
COUNT(*)
COUNT(tracks.TrackId)
```

- Die Funktion wird mit Hilfe ihres Namens aufgerufen. In diesem Beispiel `COUNT`.
- In den runden Klammern, die direkt auf der Bezeichnung folgen, wird ein Parameter für die Übergabe an die Funktion definiert. Aggreatfunktionen wird immer der Name des zu verdichtenden Datenfeldes als Argument übergeben. Das Sternchen steht für alle Datensätze.
- Funktionen geben einen Wert zurück. Aggregatfunktionen geben den verdichtenden Wert zurück.

Anzahl der Datensätze in einer Tabelle

```
SELECT
  COUNT(*)
FROM albums;
```

Anzahl der Werte

```
SELECT
  albums.Title,
  COUNT(tracks.TrackId) AS AnzahlStuecke

FROM tracks
INNER JOIN albums
ON (tracks.AlbumId = albums.AlbumId)

WHERE (tracks.Milliseconds > 300000)

GROUP BY albums.Title

ORDER BY Count(tracks.TrackId) DESC;
```

Erläuterung

- Die Aggregatfunktion `Count(*)` zählt die Datensätze in einer Tabelle. Die Gesamtanzahl der Zeilen wird zurückgegeben.
- Die Funktion `Count([datenfeld])` bezieht sich auf die Werte in dem Datenfeld. Wie viele definierte Werte gibt es in dem Datenfeld? NULL-Werte werden nicht gezählt.

Gesamtsumme

```
SELECT
  albums.Title,
  Sum(tracks.Milliseconds) AS LaengeAlbum

FROM tracks
INNER JOIN albums
ON (tracks.AlbumId = albums.AlbumId)

WHERE (tracks.Milliseconds > 300000)

GROUP BY albums.Title

ORDER BY Sum(tracks.Milliseconds) DESC;
```

Kleinster Wert

```
SELECT
  albums.Title
  MIN(tracks.Milliseconds) AS Kuerzestes

FROM tracks
INNER JOIN albums
ON (tracks.AlbumId = albums.AlbumId)

GROUP BY albums.Title

ORDER BY albums.Title;
```

Größter Wert

```
SELECT
  albums.Title
  MAX(tracks.Milliseconds) AS Laengste

FROM tracks
INNER JOIN albums
ON (tracks.AlbumId = albums.AlbumId)

GROUP BY albums.Title

ORDER BY albums.Title;
```

Durchschnittswert

```
SELECT
  albums.Title
  AVG(tracks.Milliseconds) AS Durchschnitt

FROM tracks
INNER JOIN albums
ON (tracks.AlbumId = albums.AlbumId)

GROUP BY albums.Title

ORDER BY albums.Title;
```

Filterung in Abhängigkeit des Aggregatwertes

```
SELECT [Feld], Aggregat[Feld]

FROM [Tabelle]
INNER | LEFT OUTER JOIN [Tabelle]
ON [Primärschlüssel] = [Fremdschlüssel]

WHERE ([Bedingung])

GROUP BY [Gruppenfeld]
HAVING  ([Bedingung])

ORDER BY [Feld] ASC|DESC, [Feld] ASC|DESC;
```

Beispiel

```
SELECT
  albums.Title,
  Count(tracks.TrackId) AS AnzahlStuecke

FROM tracks
INNER JOIN albums
ON (tracks.AlbumId = albums.AlbumId)

WHERE (tracks.Milliseconds > 300000)

GROUP BY albums.Title
HAVING (Count(tracks.TrackId) >= 5)

ORDER BY Count(tracks.TrackId) DESC;
```

WHERE-Klausel

```
WHERE (tracks.Milliseconds > 300000)
```

- Mit Hilfe des Schlüsselwortes `WHERE` werden die Datensätze aus der Datenquelle in Abhängigkeit einer Bedingung gefiltert.
- Der Wert in einem Datenfeld muss die Filterbedingung erfüllen. Andernfalls wird der Datensatz nicht in der Ergebnistabelle angezeigt.
- Die Datensätze werden zuerst gefiltert. Die gefilterten Daten können anschließend gruppiert werden.

HAVING-Klausel

```
HAVING (Count(tracks.TrackId) >= 5)
```

- Die Datensätze werden zuerst gruppiert und dann mit Hilfe des Schlüsselwortes `HAVING` gefiltert.
- Das Schlüsselwort `HAVING` filtert mit Hilfe von Aggregatfunktionen verdichtete Datenfelder oder Gruppen.
- Wenn die aggregierten Daten, die Bedingung erfüllen, wird der dazugehörige Datensatz angezeigt.

Hinweise

```
HAVING (Count(tracks.TrackId) >= 5)
```

- Die Bedingungen der WHERE- und HAVING-Klausel sind gleich aufgebaut.
- Aggregierte Felder können nicht über einen Alias-Namen in der HAVING-Anweisung aufgerufen werden.