SQL
Mathematische Berechnungen. Funktionen.

Welchen Bestellwert haben die einzelnen Bestellposten?

Wie viel Zeit liegt zwischen dem Bestelldatum und dem Versanddatum?

Wie ist der durchschnittliche Bestellwert?

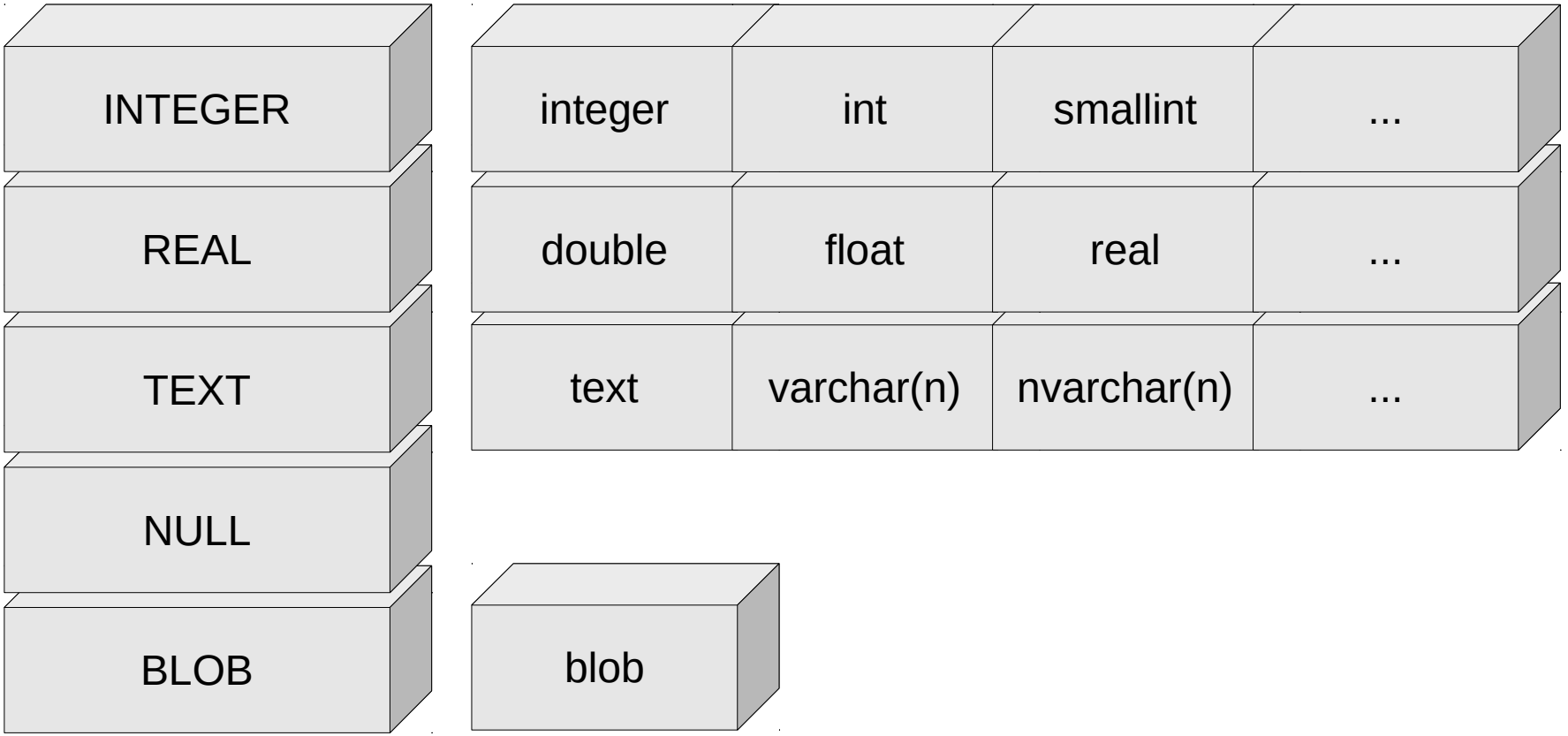
Ausdruck

- Nach bestimmten Regeln werden Operanden und Operatoren zusammengesetzt.
- Berechnung eines Wertes mit Hilfe von Operanden und Operatoren.
- Ausdrücke können geklammert werden. Die runden Klammern erhöhen die Lesbarkeit bei komplexen Ausdrücken und verändern die Rangfolge der Operatoren.
- Jeder Feldname in einer SQL-Anweisung kann fast immer durch ein Ausdruck ersetzt werden.

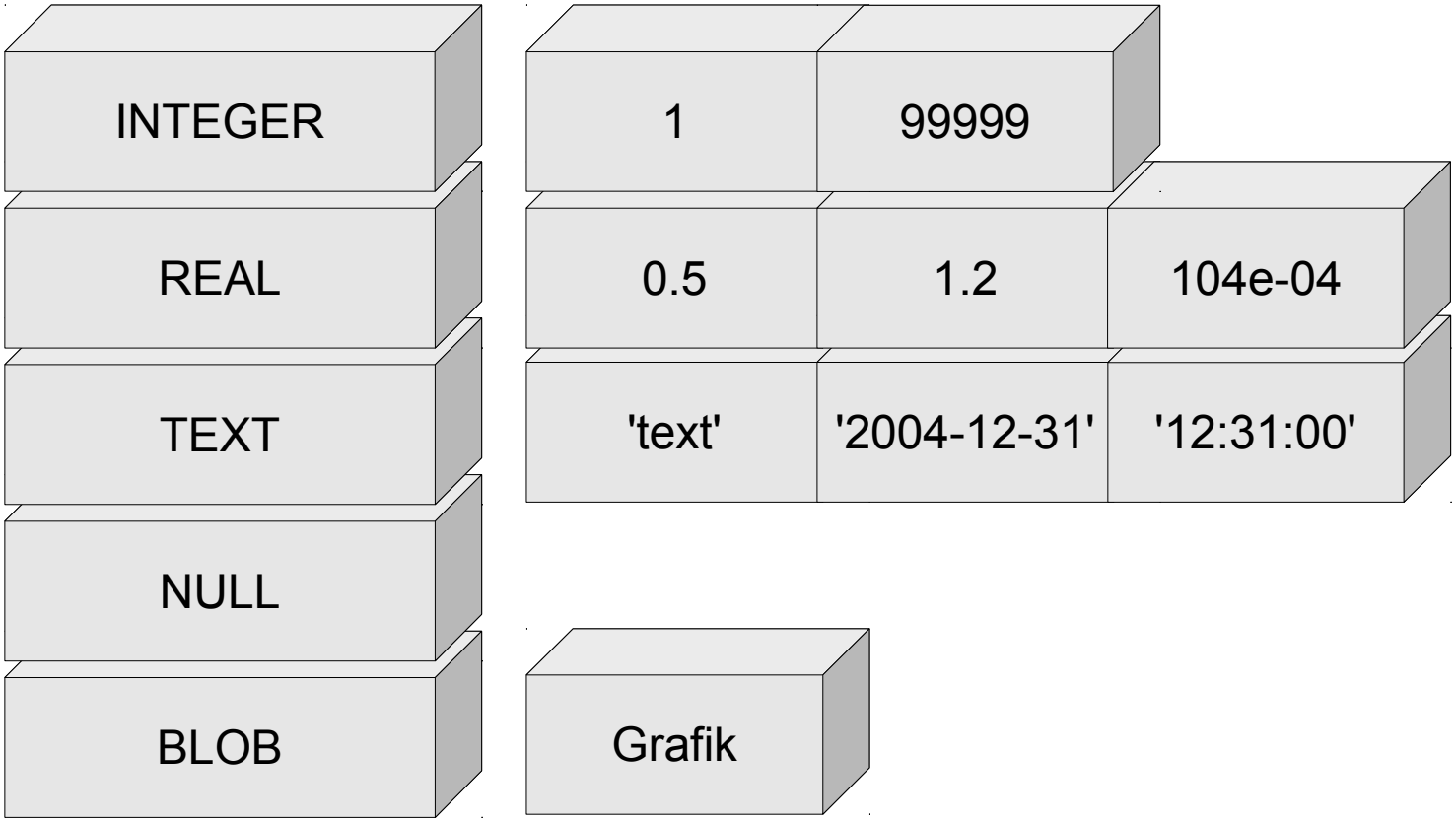
Operanden

- Datenfelder. Die, in einem Datenfeld gespeicherte Information wird in Ausdrücken als dynamischer Wert genutzt.
- Literale. Statische Werte, die direkt in der SQL-Anweisung stehen. Direkte Darstellung von Zahlen, Zeichenfolgen etc. im Ausdruck.

Storage Classes und Datentypen in SQlite



Beispiele


INTEGER
REAL
TEXT
NULL
BLOB
1
99999
0.5
1.2
104e-04
'text'
'2004-12-31'
'12:31:00'
Grafik

Operatoren

- Regeln zur Bildung von neuen Operanden.
- Vorschriften zur Bildung von Ausdrücken aus mehreren Operanden.

Operatoren in SQL

- Arithmetische Operatoren.
- Vergleichsoperatoren.
- Logische Operatoren.

Vergleichsoperatoren

- Vergleich von zwei Werten.
- Entspricht der, in einem Datenfeld gespeicherte Wert, dem Muster?
- Ein Vergleich gibt immer einen boolschen Wert zurück. Falls der Vergleich stimmt, wird true (wahr) zurückgegeben. Andernfalls wird false (falsch) zurückgegeben.
- Vergleichsoperatoren werden in WHERE- oder HAVING-Anweisungen genutzt.

… in SQL

	Operator
gleich	[Feld] = [Wert]
	[Feld] LIKE [Text]
ungleich	[Feld] <> [Wert]
	[Feld] != [Wert]
kleiner	[Feld] < [Wert]
kleiner gleich	[Feld] <= [Wert]
größer	[Feld] > [Wert]
größer gleich	[Feld] >= [Wert]

Logische Operatoren

- Verknüpfung von Ausdrücken, die einen boolschen Wert zurückgeben.
- Verknüpfung von Bedingungen, die in in WHERE- oder HAVING-Anweisungen genutzt werden.

Logische Operatoren

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

Sonstige Operatoren

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

Arithmetische Operatoren

- Berechnung eines Wertes mit Hilfe eines Ausdrucks.
- Erstellung von neuen Informationen aus den, in Datenfeldern gespeicherten Werten.
- Der berechnete Wert wird in der Ergebnistabelle angezeigt.

… in SQL

	Operator	
Addition	[Wert] + [Wert]	5 + 4 = 9
Subtraktion	[Wert] - [Wert]	5 – 4 = 1
Multiplikation	[Wert] * [Wert]	5 * 4 = 20
Division	[Wert] / [Wert]	18 / 7 = 2.57
Modula	[Wert] % [Wert]	18 % 7 = 4
Potenz	[Basis] ^ [Exponent]	2^3 = 8

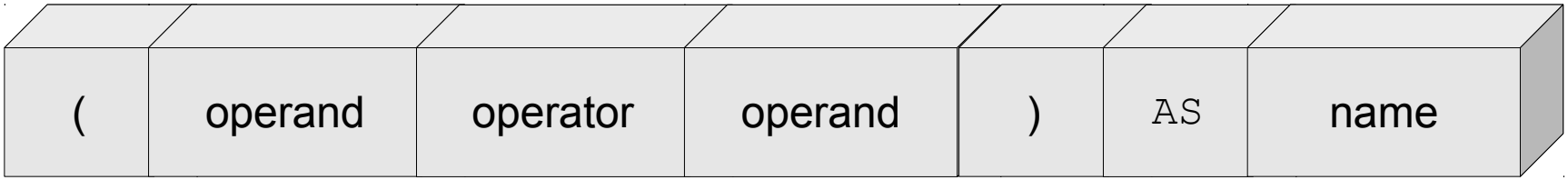
- Hinweis: Eine Division durch Null ist nicht erlaubt.

Beispiele

```
SELECT
  UnitPrice, Quantity, Discount,
  ((UnitPrice * Quantity) - Discount) AS itemPrice
FROM OrderDetails;
```

```
SELECT
  ProductName,UnitsInStock, UnitsOnOrder
FROM Products
WHERE (UnitsOnOrder - UnitsInStock) > 0;
```

Mathematische Ausdrücke



- Der berechnete Wert wird in der Ergebnistabelle der Auswahlabfrage angezeigt.
- Nutzung in der Auswahlliste, in Bedingungen oder zum Sortieren der Daten.
- In der Anzeige kann mit Hilfe des Schlüsselwortes `AS` ein Alias für die Berechnung angezeigt werden.

Hinweise

- Es gilt die Punkt- vor Strichrechnung.
- Die Operatoren werden entsprechend ihrer Rangfolge ausgewertet. Die Rangfolge der Operatoren ist nicht standardisiert.
- Komplizierte Ausdrücke können geklammert werden.

Alias-Namen

- Der Alias-Name sollte den berechneten Wert widerspiegeln.
- Der Alias-Name ist nur in der SQL-Anweisung gültig, in der die Bezeichnung definiert ist.
- Namen, die Sonderzeichen etc. enthalten, müssen durch die Anführungsstriche begrenzt werden.
- Alias-Namen in der Auswahlliste können nicht in Bedingungen oder als Sortierung für die Daten genutzt werden.

Verknüpfung von Text

```
SELECT
   CompanyName,

   ContactTitle || ' "' || ContactFirstName || ' ' ||
   ContactLastName || '"'
   AS ContactName,

   Address || ' ' || PostalCode || ' ' || City
   AS PostAddress

FROM Customers;
```

Verknüpfungsoperator

- Der Verknüpfungsoperator für Text ist in SQL nicht standardisiert. Der Operator ist abhängig vom verwendeten Datenbanksystem.
- Der zu verknüpfende Text kann in einem Datenfeld stehen oder als Literal direkt in die Anweisung geschrieben werden.

Textliterale

- Beginn und Ende mit dem Apostroph.
- Wenn als Begrenzer das Apostroph genutzt wird, darf kein Apostroph im Literal vorkommen.

Verknüpfung von Text und Zahlen

```
SELECT
Shippers.CompanyName,
Orders.OrderID || '_' || Orders.EmployeeID,
Orders.OrderDate,
Orders.Freight,
Orders.ShipName
FROM Orders INNER JOIN Shippers
ON (Shippers.ShipperID = Orders.ShipVia)
ORDER BY Shippers.CompanyName;
```

- Zahlen, die mit Text verknüpft werden, werden automatisiert umgewandelt.

Konvertierung von Werten

```
SELECT

Products.ProductName || ': ' ||
CAST(
 ((OrderDetails.UnitPrice * Quantity) - Discount)
 AS TEXT)
AS ausgabe

FROM OrderDetails
INNER JOIN Products

ON(OrderDetails.ProductID = Products.ProductID)
```

Konvertierung zu …

```
CAST(3.45 AS TEXT)
```

- Der CAST-Ausdruck wird genutzt, um ein Wert in eine beliebige Storage Class von SQLite zu konvertieren.
- Links vom Schlüsselwort AS steht der zu konvertierende Wert.
- Rechts vom Schlüsselwort steht der Name der Storage Class.

Neue Zeile (new Line)

```
SELECT
  CompanyName,

  Address || ' ' || PostalCode || char(10) || City
  AS PostAddress

FROM Customers;
```

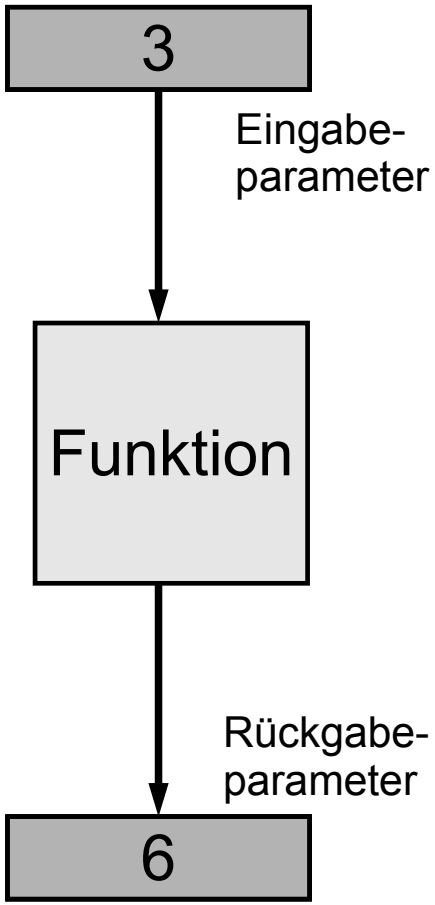
```
SELECT
  CompanyName,

  Address || ' ' || PostalCode || x'0a' || City
  AS PostAddress

FROM Customers;
```

Erläuterung

- Das Zeichen „Zeilenvorschub (Linefeed)“ wird mit der Dezimalzahl 10 oder der Hexadezimalzahl 0A kodiert.
- Der Funktion `char()` wird die dezimale Kodierung eines Unicode-Zeichens übergeben.
- Zeichenkodierungen, die hexadezimal angegeben werden, können mit dem Präfix `x` kennzeichnet werden.

Funktionen

- Stück Code, welches eine bestimmte Aktion abbildet.
- SQL-Funktionen sind eine Blackbox für den Nutzer. Der Nutzer weiß wie die Funktion genutzt werden kann, aber nicht wie diese in implementiert ist.
- Funktionen können Parameter übergeben werden. Diese Parameter werden in der Funktion verarbeitet.
- Eine Funktion kann entsprechend der implementierten Aktivität einen Wert zurückgeben.



Beispiel

```
SELECT
    Sum(invoices.Total) AS GesamtBestellsumme,
    strftime('%Y', invoices.InvoiceDate) AS Jahr

FROM invoices

GROUP BY (strftime('%Y', invoices.InvoiceDate));
```

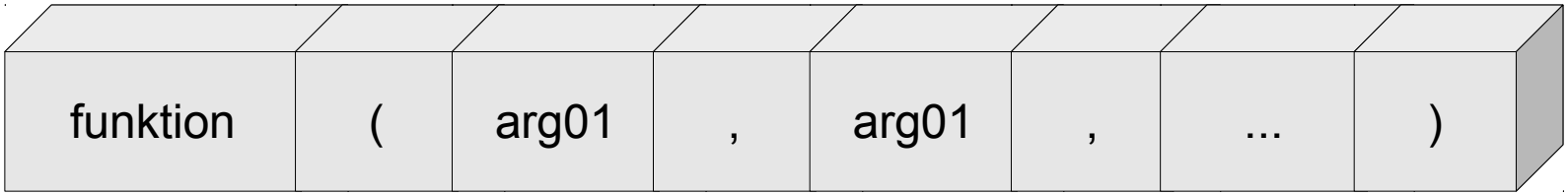
Nutzung von Funktionen

- Jede, in SQL definierte Funktion, wird mit Hilfe ihres Namens aufgerufen. Dem Namen der Funktionen folgen zwei runde Klammern. Die Klammern können leer sein.
- Eine Funktion kann anstatt eines Feldnamens aufgerufen werden.
- Funktionen werden in der Feldliste in einer Auswahlabfrage genutzt.
- Funktionen können einen Wert zurückliefern, der in Bedingungen genutzt wird.

Hinweise

- Die, in SQL implementieren Funktionen sind abhängig vom verwendeten Datenbanksystem.
- Funktionen, die in SQLite implementiert sind, werden auf der Seite https://www.sqlite.org/lang_corefunc.html gelistet.

Aufbau



- Der Name der Funktion beschreibt die Aktion, die erledigt wird.
- Dem Namen der Funktion folgen runde Klammern. Die runden Klammern begrenzen die Parameterliste.
- Die Parameterliste kann leer sein. Die Liste kann beliebig viele Parameter enthalten.
- Die Parameter in der Liste werden durch ein Komma getrennt.

Mathematische Funktionen

```
SELECT
  UnitPrice, Quantity, Discount,

  ROUND(((UnitPrice * Quantity) - Discount), 2)
  AS itemPrice

FROM OrderDetails;
```

Round()

```
ROUND(((UnitPrice * Quantity) - Discount), 2)
```

- Mit Hilfe der runden Klammern wird die Parameterliste begrenzt.
- Der Funktion werden in diesem Beispiel zwei Parameter übergeben. Die Parameter werden durch ein Komma getrennt.
- Die Funktion gibt den gerundeten Wert zurück.

Parameter der Funktion Round()

```
ROUND(((UnitPrice * Quantity) - Discount), 2)
```

- Der erste Parameter definiert die zu rundende Gleitkomma- oder Dezimalzahl. Die Zahl kann wie in diesem Beispiel berechnet werden oder in einem Datenfeld stehen.
- Der zweite Parameter legt die Anzahl der Nachkommastellen fest. Falls keine Angaben gemacht werden, wird standardmäßig auf 0 Nachkommastellen gerundet.

Text-Funktionen

```
SELECT PhotoPath
FROM Employees

WHERE NOT(
  (substr(PhotoPath, 1, length('http:'))
    LIKE 'http:')
  );
```

Länge eines Textes

```
length('http:')
```

- Als Parameter wird der Funktion ein Wert vom Datentyp `TEXT` übergeben. Es kann der Name eines Datenfeldes oder ein Literal genutzt werden.
- Die Funktion gibt die Anzahl der Zeichen zurück.

Teilstring

```
substr(PhotoPath, 1, length('http:'))
```

- Die Funktion gibt einen Teilstring aus einem Wert vom Datentyp `TEXT` zurück.
- Der erste Parameter definiert einen Text.
- Der zweite Parameter definiert eine Startposition. Ab dieser Position beginnt der Teilstring im ersten Parameter.
- Der zweite Parameter ist optional. Die Ende-Position des Teilstrings wird definiert. Falls keine Ende-Position definiert ist, werden alle Zeichen bis zum Ende des ersten Parameters zurückgegeben.

Datums- und Zeitfunktion

```
SELECT
    Sum(invoices.Total) AS GesamtBestellsumme,
    strftime('%Y', invoices.InvoiceDate) AS Jahr

FROM invoices

GROUP BY (strftime('%Y', invoices.InvoiceDate));
```

Erläuterung

- Die Funktion `strftime('%Y', invoices.InvoiceDate)` gibt einen formatierten Datums- und Zeitwert zurück.
- Der erste Parameter definiert einen Formatstring. In Abhängigkeit des Musters wird der zweite Parameter formatiert.
- Der zweite Parameter legt den zu formatierenden Datums- und Zeitwert fest.

Formatstring

- Platzhalter und Literale als Text. Der Text wird durch ein Apostroph am Beginn und Ende definiert.
- Platzhalter in dem String beginnen mit einem Prozentzeichen. Eine Liste der möglichen Platzhalter siehe unter https://www.sqlite.org/lang_datefunc.html.