SQL Aktionsabfragen

Lösche alle Kunden, die im letzten Jahr kein Ware bestellt haben.

Neue Waren werden in den Bestand aufgenommen.

Der Preis für Waren der Kategorie „Süßwaren“ erhöht sich um 10 %.

Aktionsabfragen

- Datensätze werden automatisiert in einer Tabelle verändert.
- Aktionsabfragen können auf alle oder gefilterte Datensätze angewendet werden.
- Die ausgeführte Aktion kann nicht auf Knopfdruck rückgängig gemacht werden.

Möglichkeiten

- Aktualisierung von Datensätzen. Änderung von Werten in Datenfeldern.
- Einfügung von neuen Datensätzen.
- Löschung von Datensätzen.

Hinweise

- Die zu ändernden Datensätze / Tabellen sollten vorher in Kopie gesichert werden.
- Mit Hilfe einer passenden SELECT-Anweisung kann die Aktion getestet werden.

Aktualisierung von allen Datensätzen

```
UPDATE employees
SET
      Title = 'Sales Agent';
```

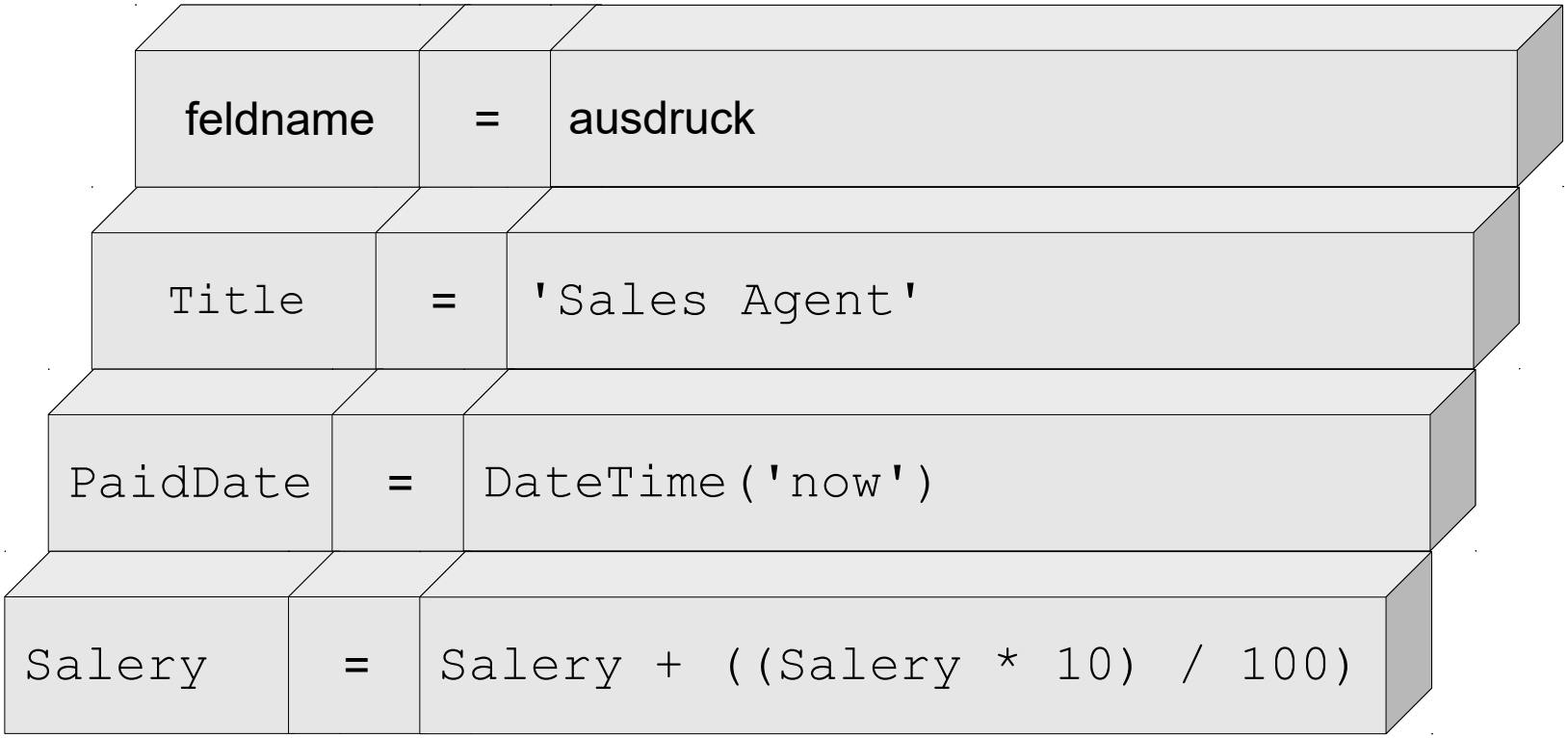
- Die Abfrage beginnt mit dem Schlüsselwort `UPDATE`.
- Dem Schlüsselwort folgt die zu ändernde Tabelle. In diesem Beispiel `employees`.
- Dem Schlüsselwort `SET` folgt eine Liste von Datenfeldern, deren Wert geändert werden soll.
- Standardmäßig werden alle Datensätze verändert.

Änderung des Wertes im Datenfeld

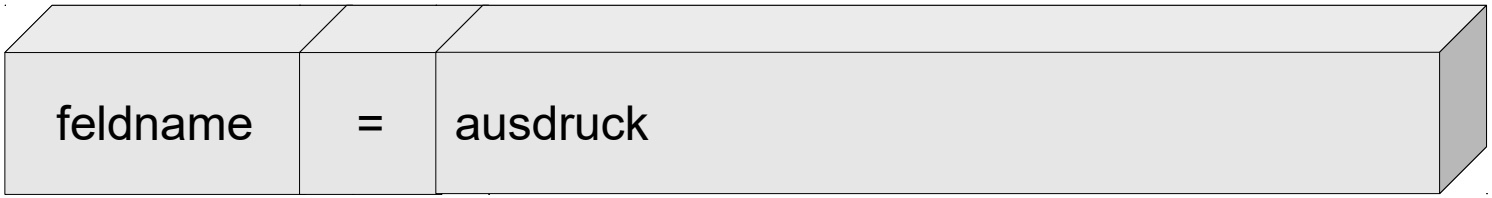
```
UPDATE employees
SET
      Title = 'Sales Agent';
```

- Der Name des zu ändernden Datenfeldes steht links vom Gleichheitszeichen. Die Bezeichnung darf nicht in der Form tabelle.feld angegeben werden.
- Rechts vom Gleichheitszeichen steht ein Wert, der dem Feld zugewiesen wird. Der neue Wert kann direkt oder aus dem gespeicherten Wert berechnet werden.

Neuberechnung von Attribut-Werten

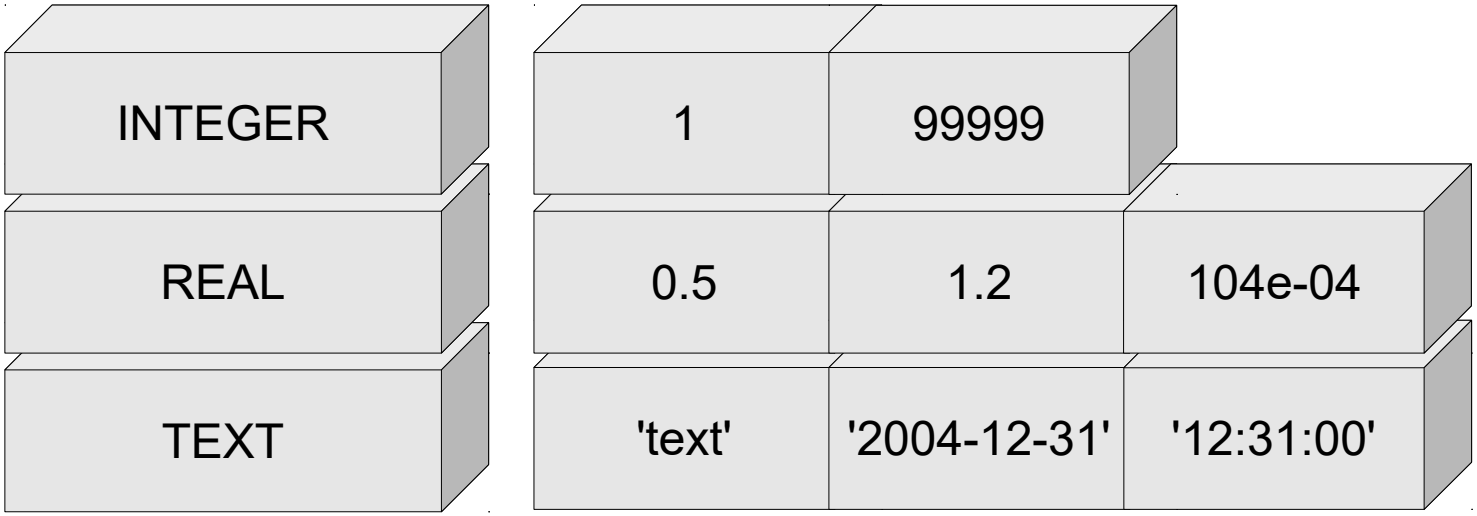


Welches Datenfeld wird geändert?

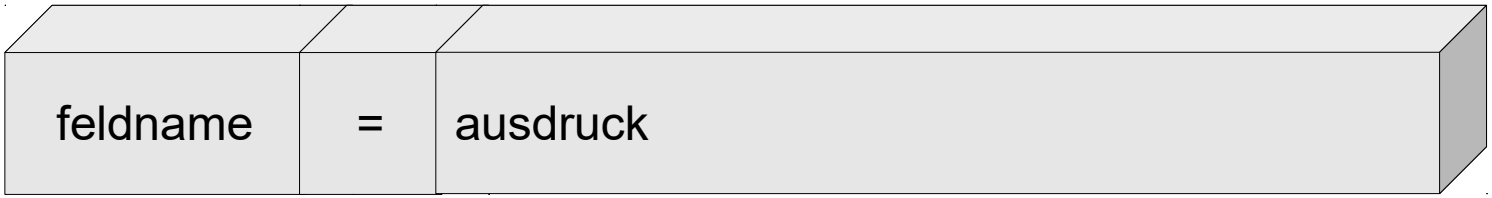


- Links vom Zuweisungsoperator wird das zu aktualisierende Feld angegeben.
- Das Datenfeld speichert in Abhängigkeit eines bestimmten Typs einen Wert.

Storage Class und deren Literale in SQLite



Neuer Wert



- Rechts vom Zuweisungsoperator wird der neue Wert angegeben oder berechnet.
- Statischen Wert (Literale) entsprechend des Datentyps können direkt als neuen Wert angegeben werden.
- Der neue Wert wird aus dem gespeicherten Wert in dem angegebenen Datenfeld berechnet.
- Der Wert wird mit Hilfe einer Funktion berechnet.

Änderung von mehreren Datenfeldern

```
UPDATE employees
SET
      Title = 'Sales Agent',
      Salery = 2000.00
WHERE (Title = 'Sales Support Agent');
```

- Dem Schlüsselwort `SET` folgt eine Liste von Datenfeldern, deren Wert aktualisiert werden soll.
- Die Zuweisungen an die Datenfelder werden durch ein Komma getrennt.

Aktualisierung von gefilterten Datensätzen

```
UPDATE employees
SET
      Title = 'Sales Agent'
WHERE (Title = 'Sales Support Agent');
```

- Mit Hilfe von `WHERE` können die Datensätze in Abhängigkeit der Bedingung eingeschränkt werden.
- Es werden nur Datensätze verändert, die die angegebene Bedingung erfüllen. Wenn keine Datensätze die Bedingung erfüllen, werden keine Daten geändert.

Test der Aktualisierungsabfrage

```
-- zu testende Abfrage
UPDATE employees
SET
      Salery = Salery + ((Salery * 10) / 100)
WHERE (Title = 'Sales Agent');
```

```
-- wird getestet mit
SELECT
  Salery,
  Salery + ((Salery * 10) / 100) AS SaleryNew
FROM employees
WHERE (Title = 'Sales Agent');
```

Einfügung von Datensätzen

```
INSERT INTO employees
(LastName, FirstName, HireDate, Address, City,
 PostalCode)

VALUES
('Smith', 'Kate', DateTime('now'),
 'Fisher Street 12', 'Calgary', 'T3B 1Y7')
```

- Die Aktionsabfrage beginnt mit dem Schlüsselwort `INSERT INTO`.

... in die Tabelle

```
INSERT INTO artists
VALUES
((SELECT Max(ArtistID) + 1 FROM artists),
  'Magtens Korridor');
```

- Dem Schlüsselwort `INSERT INTO` folgt der Tabellenname.
- Zwischen dem Schlüsselwort und dem Tabellenname muss ein Leerzeichen stehen.
- In diesem Beispiel wird in die Tabelle `artists` ein Datensatz eingefügt.

Neue Werte in die Datenfelder ..

```
INSERT INTO artists
VALUES
((SELECT Max(ArtistID) + 1 FROM artists),
  'Magtens Korridor');
```

```
INSERT INTO employees
(LastName, FirstName, HireDate, Address, City,
 PostalCode)

VALUES
('Smith', 'Kate', DateTime('now'),
 'Fisher Street 12', 'Calgary', 'T3B 1Y7')
```

Wo werden Daten eingefügt?

```
INSERT INTO employees
(LastName, FirstName, HireDate, Address, City,
 PostalCode)

VALUES
('Smith', 'Kate', DateTime('now'),
 'Fisher Street 12', 'Calgary', 'T3B 1Y7')
```

- Dem Schlüsselwort INSERT INTO folgt der Name einer Tabelle.
- In dieser Tabelle werden die Daten eingefügt. Daten werden immer nur in einer Tabelle eingefügt.

Hinzufügung der Daten in die Felder ..

```
INSERT INTO employees
(LastName, FirstName, HireDate, Address, City,
 PostalCode)
```

- Dem Tabellennamen folgt eine Feldliste. Die Feldliste wird durch die runden Klammer begrenzt.
- Die Feldnamen in der Liste werden durch ein Komma getrennt. In diese Felder werden neue Daten eingetragen.
- Die Feldliste kann weggelassen werden. Dann werden alle Datenfelder in der angegebenen Tabelle neu befüllt.

Füllen der Felder mit den Werten ...

```
INSERT INTO employees
(LastName, FirstName, HireDate, Address, City,
 PostalCode)

VALUES
('Smith', 'Kate', DateTime('now'),
 'Fisher Street 12', 'Calgary', 'T3B 1Y7')
```

- Dem Schlüsselwort `Values` folgt eine Liste von Werten für den neuen Datensatz.
- Die Liste wird durch die runden Klammern begrenzt. Die Werte in der Liste werden durch ein Komma getrennt.
- Die Anzahl der Listenelemente entspricht der Anzahl der Feldnamen in der Feldliste oder Tabelle.

Zuordnung der Werte

```
INSERT INTO employees
(LastName, FirstName, HireDate, Address, City,
 PostalCode)

VALUES
('Smith', 'Kate', DateTime('now'),
 'Fisher Street 12', 'Calgary', 'T3B 1Y7')
```

- Die Werte werden entsprechend der Position den Felder von links nach rechts zugeordnet.
- Der erste Wert wird dem ersten Feld zugeordnet und so weiter.

Beispiel

```
INSERT INTO artists
VALUES
(
  (SELECT Max(ArtistID) + 1 FROM artists),
  'Magtens Korridor'
);
```

```
CREATE TABLE `artists`(
`ArtistId` INTEGER NOT NULL PRIMARY KEY
           AUTOINCREMENT,
`Name`     NVARCHAR(120)
);
```

Regeln für die Angabe

- Für den Primärschlüssel in der Tabelle muss ein eindeutiger, nicht vorhandener Wert angegeben werden.
- Die Werte können entsprechend des Datentyps des zugeordneten Datenfeldes interpretiert werden.
- Für alle Felder, die keine NULL-Werte (`NOT NULL`) erlauben, muss ein Wert angegeben werden.

Berechnung des Primärschlüssels

```
INSERT INTO artists
VALUES
(
  (SELECT Max(ArtistID) + 1 FROM artists),
  'Magtens Korridor'
);
```

```
INSERT INTO artists
VALUES(
(SELECT (seq + 1) FROM sqlite_sequence
      WHERE (name = 'artists')),
'Pia Raug'
);
```

Nutzung von Max()

```
SELECT Max(ArtistID) + 1 FROM artists
```

- Die Aggregatfunktion `Max()` gibt den aktuell höchsten ID-Wert zurück.
- Der zurückgegebene Wert wird mit eins addiert.

Nutzung von Max()

```
SELECT (seq + 1) FROM sqlite_sequence
      WHERE (name = 'artists')
```

- Die Tabelle sqlite_sequence wird automatisiert von SQLite angelegt.
- In dieser Tabelle wird die momentan größte Zeilen-ID in dem Datenfeld `seq` für eine Tabelle gespeichert.
- Das Datenfeld `name` speichert die Bezeichnung der dazugehörigen Tabelle. Mit Hilfe der WHERE-Anweisung wird die gewünschte Tabelle gefiltert.

Kopieren von Datensätzen

```
INSERT INTO artists (ArtistId, Name)
SELECT ArtistId, Name FROM  artistsNew;
```

- Die Wertliste kann durch eine SQL-Anweisung ersetzt werden.
- Die Ergebnistabelle der SELECT-Anweisung wird in die entsprechenden Datenfelder kopiert. In diesem Beispiel wird aus der Quelltabelle `artistsNew` das Feld `ArtistID` in das Feld `ArtistID` der Zieltabelle `artists` kopiert und so weiter.

Löschen von allen Datensätzen

```
DELETE FROM tracks;
```

- Die Aktionsabfrage beginnt mit dem Schlüsselwort `DELETE`.
- Dem Schlüsselwort `FROM` folgt der Tabellenname aus dem die Datensätze gelöscht werden.
- In diesem Beispiel werden alle Datensätze ohne Rückfrage aus der Tabelle gelöscht.

Löschen von mehreren Datensätzen

```
DELETE FROM tracks
WHERE tracks.GenreId = 3;
```

- Die Aktionsabfrage beginnt mit dem Schlüsselwort `DELETE`.
- Dem Schlüsselwort `FROM` folgt der Tabellenname aus dem die Datensätze gelöscht werden.
- Mit Hilfe der WHERE-Klausel werden die zu löschenden Datensätze ausgewählt. Alle Datensätze, die der Bedingung entsprechen, werden gelöscht.

Löschen von mehreren Datensätzen

```
DELETE FROM tracks
WHERE tracks.GenreId = 3;
```

- Die Aktionsabfrage beginnt mit dem Schlüsselwort `DELETE`.
- Dem Schlüsselwort `FROM` folgt der Tabellenname aus dem die Datensätze gelöscht werden.
- Mit Hilfe der WHERE-Klausel werden die zu löschenden Datensätze ausgewählt. Alle Datensätze, die der Bedingung entsprechen, werden gelöscht.

Andere Möglichkeit

```
DELETE FROM artists

WHERE (artists.ArtistId IN (

  SELECT artists.ArtistId
  FROM artists
  OUTER LEFT JOIN albums
  ON (artists.ArtistId = albums.ArtistId)
  WHERE (albums.Title IS NULL)

));
```

Löschung in Abhängigkeit von SELECT

```
SELECT artists.ArtistId
FROM artists
OUTER LEFT JOIN albums
ON (artists.ArtistId = albums.ArtistId)
WHERE (albums.Title IS NULL)
```

- In der Ergebnistabelle wird der Primärschlüssel der Tabelle `artists` angezeigt.
- Es werden alle Schlüsselwerte aus der Tabelle `artists` angezeigt, egal ob der Schlüssel als Detailschlüssel in der Tabelle `albums` genutzt wird oder nicht.
- Das Ergebnis wird mit Hilfe von `WHERE` auf Schlüsselwerte eingeschränkt, die nicht in der Tabelle `albums` vorkommen.

Bedingung zur Löschung

```
WHERE (artists.ArtistId IN (
  SELECT artists.ArtistId
  FROM artists
  OUTER LEFT JOIN albums
  ON (artists.ArtistId = albums.ArtistId)
  WHERE (albums.Title IS NULL)

));
```

- Die Löschabfrage hat wiederum eine Bedingung. Es werden alle Datensätze gelöscht, deren Schlüsselwert in der Liste vorkommen.
- Die Werte werden mit Hilfe der SELECT-Anweisung ermittelt.

Mehrere Abfragen ausführen

```
DELETE FROM Quartalszahlen;

INSERT INTO
Quartalszahlen(ProduktName, Gesamtbestellmenge)
SELECT tracks.Name As Produktname,
coalesce(Sum(invoice_items.Quantity), 0)
As Gesamtbestellmenge
FROM tracks
LEFT OUTER JOIN invoice_items
ON (tracks.TrackId = invoice_items.TrackId)
GROUP BY tracks.Name
HAVING (invoice_items.InvoiceId IN (
SELECT invoices.InvoiceId
FROM invoices
WHERE (InvoiceDate BETWEEN '2009-01-01 00:00:00'
AND '2009-03-31 00:00:00')))
ORDER BY tracks.Name;
```

Arbeitsablauf

- Die SQL-Abfragen werden von oben nach unten abgearbeitet.
- Die SQL-Abfragen enden immer mit einem Semikolon.

1. Schritt

```
DELETE FROM Quartalszahlen;
```

- Die Tabelle `Quartalszahlen` wird vollständig gelöscht.
- Nach Ausführung der Aktionsabfrage enthält die Tabelle keine Datensätze.

2. Schritt

```
INSERT INTO
Quartalszahlen(ProduktName, Gesamtbestellmenge)
SELECT tracks.Name As Produktname, ...
```

- In die Tabelle `Quartalszahlen` werden Datensätze eingefügt.
- Die Datenfelder `ProduktName` und `Gesamtbestellmenge` werden mit Hilfe einer SQL-Anweisung befüllt.

Definition der Werte

```
SELECT tracks.Name As Produktname,
coalesce(Sum(invoice_items.Quantity), 0)
As Gesamtbestellmenge
FROM tracks
LEFT OUTER JOIN invoice_items
ON (tracks.TrackId = invoice_items.TrackId)
GROUP BY tracks.Name
HAVING (invoice_items.InvoiceId IN (
SELECT invoices.InvoiceId
FROM invoices
WHERE (InvoiceDate BETWEEN '2009-01-01 00:00:00'
AND '2009-03-31 00:00:00')))
ORDER BY tracks.Name;
```

Welche Daten werden genutzt?

```
SELECT tracks.Name As Produktname,
       coalesce(Sum(invoice_items.Quantity), 0)
          As Gesamtbestellmenge
```

- Aus der Tabelle `tracks` wird der Produktname angezeigt.
- Zu jedem Produktnamen wird die Gesamtbestellmenge mit Hilfe der Aggregatfunktion `Sum()` berechnet.
- Falls das Produkt nie bestellt wurde, wird als Wert 0 für das Datenfeld „Gesamtbestellmenge“ angezeigt.
- Beiden Datenfelder bekommen mit Hilfe von `As` ein passendes Label zugewiesen.

Wo sind die Daten abgelegt?

```
FROM tracks
LEFT OUTER JOIN invoice_items
ON (tracks.TrackId = invoice_items.TrackId)
```

- Es werden alle Datensätze aus der Tabelle `tracks` angezeigt.
- Falls vorhanden, werden die dazugehörigen Bestelldetails (`invoice_items`) angezeigt.
- Die Tabellen werden über den Primär- / Fremdschlüssel `TrackId` verbunden.

Zusammenfassung der Daten

```
GROUP BY tracks.Name
HAVING (invoice_items.InvoiceId IN (
SELECT invoices.InvoiceId
FROM invoices
WHERE (InvoiceDate BETWEEN '2009-01-01 00:00:00'
AND '2009-03-31 00:00:00')))
```

- Die Daten werden in Abhängigkeit des Feldes tracks.Name gruppiert.
- Es werden Gruppen von Produktnamen gebildet, die zwischen dem 01.01.2009 und 31.03.2009 bestellt wurden. Mit Hilfe einer Unterabfrage werden diese Produkte ermittelt.

Sortierung der Daten

```
ORDER BY tracks.Name;
```

- Die angezeigten Daten werden in Abhängigkeit des Feldes `tracks.Name` sortiert.