SQL für den alltäglichen Gebrauch

SQL ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen von darauf basierenden Datenbeständen.

SQL - Wikipedia

https://de.wikipedia.org/wiki/SQL

Agenda

- CREATE
  - Table
  - Index
  - View
  - Feldreferenz mit DDL
- SELECT
  - DISTINCT
  - Skalare Funktionen
  - Aggregatsfunktionen (count, sum, max, min, avg)
  - CASE (Fallunterscheidung)
  - FROM
    - Korrelationsnamen
    - geschachtelte Tabellenausdrücke
  - WHERE-Bedingung
  - ORDER BY
  - Sub-Query
  - Mengenoperatoren (UNION, EXCEPT, INTERSECT)
- JOIN
  - Join oder Inner Join
  - Left/Right Join oder Left/Right Outer Join
  - Left/Right Exception Join
  - Cross Join
- GROUP BY
  - ROLLUP
  - CUBE
  - RANK, DENSE_RANK
  - Grupping Sets
- HAVING
- CTE Temporäre Sichten (common table expression)
- CASTING und JOINING mit incompatiblen Keyfeldern

Agenda

- UPDATE
- Update mit “correlated Queries”

- DELETE
- Delete und Common Table Expression (CTE)
- Delete mit sub-Queries

- INSERT
  - mit VALUES
  - mit SELECT

- MERGE

- Rekursives SQL
- Rekursive Abfragen
- Datumsarithmetik
- Zusätzliche Infos
  - Begriffsklärung
  - Wo sind die SQL-Infos im System hinterlegt
  - Wie heißen die Felder in der Datei / Wo wird das Feld benutzt

Create table

```
create or replace Table KDSTAMP
( KDNR      dec (6)  generated always as Identity      ← "technischer" Schlüssel
            (Start with 100, increment by 10),         ← Start- und
                                                         Inkrementwert
  Name      char(35)     not NULL with default,
  Ort       char(35)     not NULL with default,
  ISOLand   char(2)      not NULL with default,
  AdrNr     dec (6)      not NULL with default,
  ErfD      date         not NULL with default,
  AenTS     timeStamp    not NULL
                         FOR EACH ROW ON UPDATE AS
                         ROW CHANGE TIMESTAMP        ← Änderungsdatum im
                                                         Datensatz wird
                                                         automatisch
                                                         fortgeschrieben
  ....
);
```

```
 Label on Table  KDSTAMP is 'Kundenstamm';
Label on Column KDSTAMP
( KDNR      is 'Kunden-          Nr',
  Name      is 'Name',
  Ort       is 'Ort',
  ISOLand   is 'Land-            Kenner',
  AdrNr     is 'Adress-          Nr',
  ErfD      is 'erfasst          am',
  AenTS     is 'geändert         am'
  ....
);
```

Index erstellen

```
CREATE UNIQUE INDEX KDSTAMIdx ON KDSTAMP (  KDNR );
```

View erstellen

```
create VIEW AuftgWert (Kunde, Name, Wert) as
select   k.kdnr, k.Name,
        (select  sum(ap.Preis)
           from  AUFPOSP ap
          where  ap.KDNR = k.KDNR) as Wert
  from  KDSTAMP k
 group by k.KDNR, k.Name   ;;

select * from AuftgWert;
```

Select

Im SELECT-Statement wird angegeben, wie das Ergebnis aussehen soll, d.h. welche Spalten die Ergebnistabelle haben soll, und gegebenenfalls wie diese benannt werden. Die einzelnen Spalten des Ergebnisses sind durch Kommata zu trennen, die Nachstellung eines nicht qualizierten Spaltennamen benennt die Spalte der Ergebnisrelation (um).

Durch * oder Tabelle.* werden alle Spalten (der benannten Tabelle) ins Ergebnis übernommen.

Es ist möglich, die Werte der Ergebnisspalten im Select-Teil durch einen Ausdruck berechnen zu lassen.

```
SELECT (distinct)..., ausdruck, ..., (select ...), ...
  FROM  tabelle , ... / (select ...)
WHERE ausdruck vgl ausdruck AND / OR ...
          ausdruck vgl ( select ... )
GROUP BY ausdruck, ...
HAVING bedingung
UNION / EXCEPT / INTERSECT
SELECT ...
...
ORDER BY spalte ASC / DESC, ...
FETCH FIRST anzahl ROWS ONLY
```

Mit der Distinct-Funktion in der Select-Anweisung können doppelte Zeilen eliminiert werden.

Skalarfunktionen

Eine *Skalarfunktion* führt eine Operation für einen Einzelwert aus und gibt einen anderen Einzelwert zurück. Im folgenden werden nur einige Beispiele für die Skalarfunktionen dargestellt:

abs	Gibt den absoluten Wert einer Zahl zurück.
hex	Gib die hexadezimale Darstellung eines Werts zurück.
length	Gibt die Anzahl der Byte in einem Argument zurück
year	Extrahiert den Abschnitt für das Jahr aus einem Wert für Datum und Uhrzeit.
trim	entfernt Leerzeichen (links und rechts)
trimr	entfernt rechte Leerzeichen
….	

Aggregatfunktionen

Eine *Aggregatfunktion* berechnet einen einzelnen Ergebniswert aus einer Gruppe von Werten.

count	zählt alle Zeilen
sum	stellt einen Gesamtwert bereit
max	stellt den Maximalwert bereit
min	stellt den kleinsten Wert bereit
avg	stellt den Durchschnittswert bereit. Die Durchschnitte von genauen numerischen Typen werden auf 6 Dezimalstellen berechnet

Case

CASE ermöglicht die Auswahl eines Ausdrucks / Wertes aufgrund des Ergebnisses einer Prüfung eines oder mehrerer Bedingungen.

CASE when {Spaltenname} then Ausdruck

when {Ausdruck } then Wert

else

END

```
select Vorname, Name,
       case Geschlecht when 'F' then 'Frau'
                       when 'M' then 'Mann'
                       when 'X' then 'Diverse'
                       else 'unbekannt'
       end
  from PERSONP;
```

```
select Vorname, Name,
    case when Geschlecht = 'F' then 'Frau'
         when Geschlecht = 'M' then 'Mann'
         when Geschlecht = 'X' then 'Diverse'
         else 'unbekannt'
     end
  from PERSONP;
```

From

Die FROM-Klausel legt eine „Zwischenergebnistabelle“ fest.

Wenn nur eine Datei (Tabelle) angegeben ist, ist die Zwischenergebnis-Tabelle einfach das Ergebnis dieser Table-Referenz. Wenn mehr als eine Table-Referenz angegeben wird, besteht die Zwischenergebnistabelle aus allen möglichen Kombinationsmöglichkeiten der Reihen des Ergebnisses jeder angegebenen Table-Referenz.

SELECT ..., ausdruck, ..., (select ...), ...

FROM tabelle , ... / (select ...)

Geschachtelte Tabellenausdrücke

Eine nested table expression bietet dem Anwender die Möglichkeit, in der FROM-Klausel eines SELECT-Statements nicht eine View oder Tabelle anzugeben, sondern ein weiteres SELECT-Statement.

Der nested table Select kann bis auf wenige Ausnahmen den vollen Funktionsumfang des normalen Select-Statements nutzen, darunter auch Subqueries, Joins und weitere nested tables. Nicht erlaubt sind dagegen ein UNION, UNION ALL sowie diverse Klauseln wie z.B. ORDER BY

Der nested table Select muss in Klammern gesetzt werden und von einem Korrelationsnamen gefolgt sein.

```
SELECT teilspalte , SUM (sp1) , AVG ( sp2 )
FROM
   ( SELECT SUBSTR ( spalte , 1 , 3 ) AS teilspalte , sp1 , sp2
   FROM tabelle
   WHERE bedingung ) AS corr
GROUP BY teilspalte ;
```

Korrelationsnamen

Ein Korrelationsname ist eine Kennung, mit der die einzelnen Verwendungsarten eines Objekts unterschieden werden.

Ein Korrelationsname kann in der Klausel FROM einer Abfrage und in der ersten Klausel einer Anweisung UPDATE oder DELETE definiert werden. Ihm kann eine Tabelle, eine Sicht oder ein verschachtelter Tabellenausdruck zugeordnet sein, jedoch nur in dem definierten Kontext.

Nachdem Sie einen Korrelationsnamen definiert haben, können Sie zur Qualifizierung des Objekts *ausschließlich* den Korrelationsnamen verwenden.

```
select *
 from Table kn
where rrn(kn) = 123;
```

Where

Mithilfe des **SQL WHERE-Befehls** werden in SQL Abfragen nur bestimmten Datensätze ausgewählt. Der **SQL WHERE-Befehl** funktioniert im Prinzip wie ein Filter, der es ermöglicht, nur Datensätze anzuzeigen, die bestimmte Kriterien erfüllen.

Soll ein **SQL** Statement eine bestimmte Bedingung erfüllen, muss eine **WHERE**-Bedingung eingebaut und erfüllt werden, damit die Abfrage eine Ergebnismenge liefern kann.

Nach WHERE kann eine beliebige Bedingung folgen. Wichtig sind hier Vergleichsoperatoren wie gleich (=), größer als (>), kleiner als (<), größer oder gleich (>=), kleiner oder gleich (>=) oder ungleich (<> bzw. !=).

Die Vergleichsoperatoren können mit AND und/oder OR kombiniert werden. Der Einsatz der Operatoren folgt der booleschen Algebra, die man aus dem Mathematikunterricht kennt.

Operatoren:

=, <>, !=, <, >, <=, >=, like, not like, is null, is not null, between, in, not in

Order by

Die Klausel ORDER BY wird als letzte Klausel der gesamten Anweisung SELECT angegeben. Bei den in dieser Klausel angegebenen Spalten kann es sich um Ausdrücke oder um beliebige Spalten der Tabelle handeln. Die Spaltennamen in der Klausel ORDER BY müssen nicht in der SELECT-Liste angegeben sein.

Die Zeilen können in aufsteigender oder in absteigender Reihenfolge sortiert werden. Hierzu geben Sie in der Klausel ORDER BY explizit ASC (=aufsteigend) oder DESC (=absteigend) an. Enthält die Klausel keine dieser Angaben, werden die Zeilen automatisch in aufsteigender Reihenfolge sortiert.

Sub-Query (sub-Select)

Ein sub-Query ist ein geschachteltes SQL-Statement (oder ein sub-Select), welches ein SELECT-Statement mit WHERE- und HAVING-Klausel (optional) beinhaltet.

```
select Column(s),
     ( select Column
        from TableSub s
       where s.Colum = p.Column)
 from TablePrim p
order by p.Column(s)
```

(INNER) Join

Ein SQL-**Join** verknüpft eine oder mehrere Spalten verschiedener Tabellen einer relationalen Datenbank, um das Ergebnis in Tabellenform auszugeben oder weiterzuverarbeiten.

Ein JOIN ermöglicht das Verknüpfen der Spalten mehrerer Tabellen, die gemeinsame Werte besitzen. Eine Tabelle kann ein Join auf sich selbst durchführen (SELF JOIN).

- meist benutzte JOIN-Anweisung
- liefert alle rows (Records) für die es eine Entsprechung (match) in Tabelle 1 und 2 lt. der JOIN-Bedingung gibt
- liefert die Werte für alle Columns (Felder)

```
select Column(s)
   from Table1 a                                  Datei 1
   Join   Table2 b                                Datei 2
      On a.Column1 = b.Column1 and                Join Bedingung
         a.Column2 = b.Column2
 where …..                                        Bedingung
```

Left (OUTER) Join

- Liefert die Werte aller rows (Records) der linken Tabelle und die Werte der „joined“ Tabelle, welche lt. Join Bedingung übereinstimmen
- Wenn eine row in der rechten Tabelle nicht gefunden wird, wird NULL zurückgeliefert
- NULL-Werte können überschrieben werden mit dem (z.B.) IFNULL-Operand

  (Alternativ kann für ifNULL der Operand VALUE oder COALESCE verwendet werden)

```
select Column(s)
   from Table1 a                                  Datei 1
   left (outer) Join Table2 b                     Datei 2
      On a.Column1 = b.Column1 and                Join Bedingung
         a.Column2 = b.Column2
 where …..                                        Bedingung
```

Right (OUTER) Join

- Liefert die Werte aller rows (Records) der rechten Tabelle und die Werte der „joined“ Tabelle, welche lt. Join Bedingung übereinstimmen
- Wenn eine row in der linken Tabelle nicht gefunden wird, wird NULL zurückgeliefert
- NULL-Werte können überschrieben werden mit dem (z.B.) IFNULL-Operand

  (Alternativ kann für ifNULL der Operand VALUE oder COALESCE verwendet werden)

Group by

Eine GROUP BY – Klausel gruppiert die Ergebnisse eines Query entsprechend der / den Gruppierungsanweisung/en.

select Column(s)

from Table1

where …..

***group by* Column(s)**

Mittels der CUBE- und ROLLUP-Funktion innerhalb der GROUP BY -Klausel können Zusammenfassungen für jede unterschiedliche Kombination der Gruppierungsspalten beschrieben werden.

…

***group by CUBE*(Column1, Column2, …)**

***group by ROLLUP*(Column1, Column2, …)**

Mittels der GRUPPING SET-Funktion innerhalb der GROUP BY -Klausel können Spalten zu Gruppenwechselbegriffen zusammengefasst werden.

…

group by grouping sets((Column1, Column2), (Column7, Column4))

RANK und DENSE_RANK

Mit der Funktion RANK und DENSE_RANK kann eine Reihenfolge (Ranking) für jedes Feld (column) im Resultset dargestellt werden.

```
select Column1, Column2, count(*),
      RANK() OVER (ORDER BY count(*) DESC) AS Rank,
      DENSE_RANK() OVER (ORDER BY count(*) DESC) AS DENSE_Rank
  from  Table1 a
    join  Table2 b
      using(Key-Column)
 group by Column1, Column2
```

Having

Die HAVING-Klausel legt eine Ergebnistabelle fest, die aus den Gruppen der Zwischentabelle besteht, für welche die Suchergebnisse gelten.

Beispiel

Nur Records anzeigen, welche lt. Bedingung mehr als 3x vorkommen:

```
select Column(s), count(*)
  from Table1 a
    join Table2 b
      on b.Column = a.Column and
           …….
 where Bedingung
 group by Column(s)
 having count(*)  > 3
order by 1, 3 desc, 2;
```

Temporäre Sichten (CTE)

Analog der Definition einer View können temporäre, nur für die nachfolgende SELECT-Anweisung benutzbare Sichten beschrieben werden.

Beispiel:

```
WITH TempFILE ( Column1, Column2…)          ← temporäre Ergebnistabelle
   as ( select Column1, Column2, …          ← SELECT (Quelle) für die temp. Ergebnistabelle
         from FILE
        group by Column1, Column2, …)
select *                                    ← Auswertung der temp.Ergebnistabelle
 fromTempFILE
where Bedingung
```

Update

Mit dem UPDATE Statement, können die Werte einer oder mehrerer Spalten (Column) in jedem Record (row) geändert werden, welche mit der WHERE-Bedingung übereinstimmen.

UPDATE table-name
    SET column-1 = value-1,
        column-2 = value-2, ...
WHERE search-condition ...

Einfacher Update mit Verwendung von „Jokern“ in der Where-Bedingung:

```
update ARTFARBP
  set Farbe = 'Bright Silver Metallic'
where ArtId    = 1001
    and FarbId = 21
    and Farbe like '%sil%metallic%';
```

Update mit „correlated Query“

```
update KDSTAMP k
set k.Ort =
   (select a.Ort
     from ADRSTAMP a
    where a.AdrNr = k.AdrNr
       and a.AdrTyp= 'KD'
       and a.Ort  <> ' ' ),
   k.ISOLand  =
   (select a.Land
     from ADRSTAMP a
    where a.AdrNr = k.AdrNr
       and a.AdrTyp= 'KD'
       and a.Ort  <> ' ' )
where exists
    (select a.Land
      from ADRSTAMP a
     where a.AdrNr = k.AdrNr
       and a.AdrTyp= 'KD'
       and a.Ort  <> ' '
       and a.Land  = 'US');
```

```
update KDSTAMP k
set (k.Ort, k.ISOLand) =
   (select a.Ort, a.Land
     from ADRSTAMP a
    where a.AdrNr = k.AdrNr
       and a.AdrTyp= 'KD'
       and a.Ort  <> ' ' )
where exists
    (select a.Land
      from ADRSTAMP a
     where a.AdrNr = k.AdrNr
       and a.AdrTyp= 'KD'
       and a.Ort  <> ' '
       and a.Land  = 'US');
```

Delete

Das DELETE-Statement wird benutzt zum Löschen einer oder mehrerer Records (rows) in einer Datei, basierend auf der/den WHERE-Bedingung/en.

Das Löschen eines Records (row) über einen View, löscht die Daten in der physische Datei (table).

delete from Table
where Column1 > Wert

Die WHERE-Bedingung kann – wie beim SELECT-Statement – einfache Ausdrücke, Mehrfachbedingungen, CASE-Konstrukte, Joker-Abfragen …… haben.

Delete mit CTE-Tabellen

Records aus Tabelle A in Abhängigkeit von Tabelle B löschen,
z.B. alle doppelten Sätze in einer Tabelle löschen

```
with Temp(AdrNr, Anzahl)
  as (select AdrNr, count(*)
        from KDSTAMP
       group by AdrNr
      having count(*) > 1)
delete from KDSTAMP k
where k.AdrNr in (select t.AdrNr
                        from Temp t);
```

Fehler SQL0199: Schlüsselwort DELETE nicht erwartet….
delete ist mit CTE (common table expression) nicht möglich:
(CTEs are part of an SELECT statement and not of an
DELETE statement.)

```
delete from KDSTAMP k
where (select count(*)
          from KDSTAMP t
         where t.AdrNr = k.AdrNr) > 1;

delete  from ADRSTAMP a
where  not exists (select *
                     from KDSTAMP k
                    where k.AdrNr = a.AdrNr);
```

Achtung bei „correlated Queries“:
Die SQL-Anweisung “crashed“, wenn das Ergebnis des Sub-Selects mehr als eine Zeile liefert.

Insert

Das INSERT-Statement fügt Records (rows) in eine Datei (table) oder eine Sicht (view) ein.

Ein INSERT in einen View fügt die Zeile in der phy. Datei an, auf der die View basiert.

Es gibt drei Formen zu diesem Statement:

- INSERT mit VALUES fügt eine einzelne Zeile (row) in eine Datei oder einen View
- INSERT mit SELECT fügt eine oder mehr Zeilen in eine Datei oder einen View unter Benutzung von Werten aus anderen Dateien, Views oder beidem
- INSERT mit FOR n ROWS fügt mehrere Zeilen in eine Datei oder einen View mit bereitgestellten oder referenzierten Werten. Diese Form des INSERT wird in SQL-Procedures unterstützt.

Insert

INSERT into ADRSTAMP

(AdrNr, AdrTyp, Name1, Name2, Land, Strasse, PLZS, Ort)

VALUES(1001, 'KD', 'Müller', 'Peter', 'DE', 'Hauptstr. 31', '63755', 'Alzenau'),

(1002, 'AD', 'Mayer', 'Erwin', 'DE', 'Gartenweg 17', *'63739', 'Aschaffenburg'),*

…… ;

INSERT into ADRSTAMP

(AdrTyp, Name1, Name2, Land, Strasse, PLZS, Ort)

SELECT 'AD', adr.NAME1, adr.Name2, adr.Land, adr.Strasse, adr.PLZ, adr.ORT

from Adresse adr

where upper(adr.Name1) like '%HUB%';

Merge

Das MERGE-Statement schreibt Daten einer Zieldatei (table) oder einer Sicht (view) mit den Werten einer Quelldatei (source) fort.

Zeilen (rows) der Zieldatei welche mit der Quelldatei (source) übereinstimmen, können fortgeschrieben (update) oder gelöscht (delete) werden, Zeile welche nicht in der Zieldatei existieren, können eingefügt werden.

(Update, delete und insert in einer View, ändern die Werte in der phy.Datei, auf welcher der View basiert.)

Merge (Beispiel)

```
MERGE into ADRSTAMP a
  using (select *
           from ADRESSBAY) as aBay
     on a.AdrNr = aBay.AdrNr
  when matched then
    update set a.Name1 = aBay.Name1,
               a.Name2 = aBay.Name2,
               a.Name3 = aBay.Name3,
               a.Land    = aBay.Land,
               a.Strasse = aBay.Strasse,
               a.PLZS    = aBay.PLZS,
               a.Ort     = aBay.Ort
  when not matched then
    insert  (AdrNr, AdrTyp, Name1, Name2, Name3, Land, Strasse, PLZS, Ort)
    values (aBay.AdrNr,   aBay.AdrTyp,
            aBay.Name1, aBay.Name2, aBay.Name3,
            aBay.Land,    aBay.Strasse,  aBay.PLZS, aBay.Ort)
  else ignore;
```

Rekursives SQL

Viele Anwendungen und Daten sind von Natur aus rekursiv. Beispiele solcher Anwendungen sind Stücklisten, Reservierungen, Reiseplanungen und Netzwerkplanungssysteme, wo Daten einer Ergebniszeile eine natürliche Verbindung mit Daten anderer Zeilen haben.

Rekursive Abfragen können mittels Definition von Recursive Common Table Expression (RCTE) oder einer Recursive View definiert werden.

Rekursive Abfragen

Eine rekursive Abfrage ist eine Abfrage, die mittels eines UNION ALL das initialisierende SELECT mit einem rekursiv benutzten SELECT verbindet, das eine direkte Referenz mit sich selbst in der FROM-Klausel beinhaltet.

Beispiel: File REC mit den Feldern sNum, wordNum, word

Insert into REC

```
VALUES  (1, 1, 'Das’),  (1, 2, 'ist’),
        (1, 3, 'ein’),   (1, 4, 'Beispiel'),
        (1, 5, 'für’),   (1, 6, 'rekursives'),
        (1, 7, 'SQL');
```

Rekursive Abfrage

```
with rQuery (sNum, wordNum, sentence)
  as ( select base.sNum, base.wordNum, base.word
         from REC base
       where wordNum = 1
       UNION ALL
        select t1.sNum, t1.wordNum, sentence || ' ' || t1.word
          from rQuery t0, REC t1
       where t0. sNum = t1. sNum
          and t0.wordNum + 1 = t1.wordNum )
select sentence
  from rQuery
 where wordNum = (select max(wordNum) from rQuery);
```

SNUM	WORDNUM	SENTENCE
1	7	Das ist ein Beispiel für rekursives SQL

Begriffsklärung

Bibliothek	Library	Collection/Schema
Datei	Physical File	Table
Log.Datei	Logical	View / Index
Datensatz	Record	Row
Feld	Field	Column

Wo sind die SQL-Infos im System hinterlegt:

SYSVIEWS	Views
SYSTRIGGER	Triggers
SYSPROCS	Procedures
SYSKEYS	Keys
SYSINDEXES	Indexes
SYSFUNCS	Functions
SYSCST	Constraints
SYSCOLUMNS	Columns