

Datenbanksysteme

SQL DQL

Jan Haase

2024

Abschnitt 7

Wiederholung: Kategorien von SQL-Befehlen

- **DQL (Data Query Language)**
Abfrage und Zusammenstellung von Daten

- SELECT

- **DML (Data Manipulation Language)**
Umgang mit Tabelleninhalten

- INSERT
- UPDATE
- DELETE

Externe
Sicht



- **DDL (Data Definition Language)**
Erstellen und Ändern von Datenbanken und Tabellen

- CREATE
- ALTER
- DROP

Konzeptionelle
Ebene



- **DAL (Data Administration Language)**
 - TCL (Transaction Control Language)
 - DCL (Data Control Language)

Interne Ebene

Wiederholung: Grundstruktur SQL-DQL-Anweisungen

- **Befehl:** SELECT
- **Feldselektion:** „*“ oder Aufzählung der Attribute
- **Tabellenselektion:** FROM mit Tabellennamen
Verknüpfung mehrerer Tabellen durch kartesisches Produkt/JOIN
- optional:
 - **Klauseln:** WHERE mit Operatoren
(=, <>, <, >, <=, >=, LIKE, BETWEEN, IN)
 - **Verknüpfung der Klauseln:**
Logische Operatoren AND, OR und NOT.
Klammernsetzung ist möglich.
 - **Sortieren der Ergebnisse:** ORDER BY
Richtung mit ASC - aufsteigend und DESC - absteigend
- Erstes Beispiel: **SELECT * FROM Rechnung**
WHERE Rechnungsnummer > 2
AND Kunde LIKE 'Eastwood%'
ORDER BY Rechnungsdatum DESC

Wiederholung - Relationen Algebra

- Theoretische Grundlage relationaler Datenbanken

- Anfragen formulieren
- Informationen zusammenstellen

- **Klassische Operationen**

- Vereinigung



- Durchschnitt



- Differenz

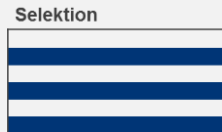


Unterabfragen:

- UNION
- INTERSECT
- EXCEPT | MINUS

- **Spezielle Operationen**

- Selektion

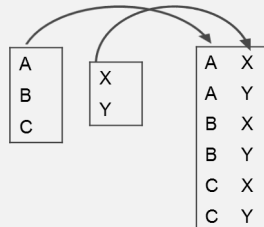


- Projektion

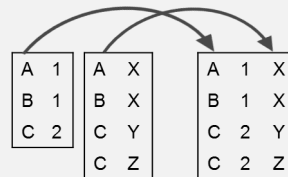


Einfache SQL-Abfragen:
SELECT ... WHERE...

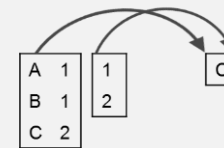
- Kartesisches Produkt



- Natural Join



- Division



JOINS

Themenübersicht

- Grundbegriffe und Datenbankentwurf
- Entity-Relationship-Modelle
- Relationales Datenbankmodell
- Normalisierung
- Arbeiten mit relationalen Datenbanken (SQL)
 - **DQL (Data Query Language)**
 - Einfache SQL-Abfragen
 - NULL-Werte
 - ➔ **Unterabfragen**
 - JOINS
 - GROUP BY
 - DML
 - DDL
 - DAL

SQL Unterabfragen

- Eine Unterabfrage kann als **innerer Anfrageausdruck** in der
 - SELECT-Klausel
 - FROM-Klausel
 - WHERE-Klausel
 - WITH-Klausel oder
 - HAVING-Klauselgenutzt werden.
- Anzahl der Verschachtelungen unbegrenzt
- Leere Unterabfragen (= 0 Zeile) liefern NULL
- Beispiel: „Alle Städte aus Deutschland“, wobei angenommen wird, dass wir den Code für Deutschland nicht kennen:

```
SELECT * FROM city
      WHERE country = (SELECT code FROM country
                      WHERE Name = 'Germany' );
```

SQL Unterabfragen: IN OPERATOR

- Wenn man das Ergebnis einer Unterabfrage bereits als Ergebnismenge angeben möchte, kann hierfür der **IN**-Operator verwendet werden:

```
SELECT * FROM city  
WHERE country IN ('D', 'A', 'CH');
```

SQL Unterabfragen: EXISTS OPERATOR

- Beispiel: Alle Länder, die eine Stadt mit mehr als 1.000.000 Einwohner haben:

```
SELECT country.Name FROM country
WHERE EXISTS (SELECT * FROM city
              WHERE city.Population>1000000
              AND city.Country=country.Code) ;
```

- Die Unterabfrage mit **EXISTS** liefert ein Ergebnis, wenn der innerhalb der EXISTS-Klammer stehende Ausdruck eine wahre Aussage enthält. Analog gilt NOT EXISTS. Dann liefert die Abfrage ein Ergebnis, wenn der in der EXISTS-Klammer stehende Ausdruck eine falsche Aussage enthält.
Probieren Sie es aus!

SQL Unterabfragen: UNION OPERATOR

- **Vereinigung** der Relationen-Algebra
- Mit dem Befehl **UNION** können zwei Abfragen verbunden werden.
- Tabellen müssen vereinigungskonform sein.
- Beispiel: Gebe den Namen aller Länder aus, die eine Area größer als 100.000 oder mehr als 1 Million Einwohner haben.



```
(SELECT Name FROM Country WHERE Area > 100000)
```

```
UNION
```

```
(SELECT Name FROM Country WHERE Population >  
1000000) ;
```

SQL Unterabfragen: INTERSECT

- **Schnittmenge** der Relationen Algebra
- Tabellen müssen vereinigungskonform sein.
- Beispiel: Gebe den Namen aller Länder aus, die eine Area größer als 100.000 und mehr als 1 Million Einwohner haben.



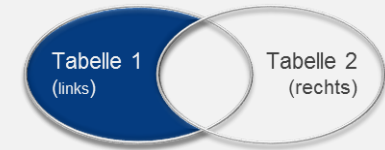
```
(SELECT Name FROM Country WHERE Area > 100000)
```

```
INTERSECT
```

```
(SELECT Name FROM Country WHERE Population >  
1000000) ;
```

SQL Unterabfragen: EXCEPT | MINUS

- **Differenz** der Relationen Algebra
- Tabellen müssen vereinigungskonform sein.

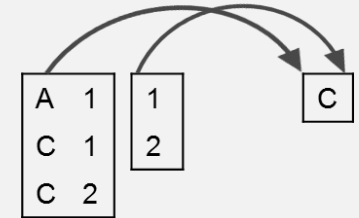


- Beispiel: Gebe den Namen aller Länder aus, die eine Area größer als 100.000 aber nicht mehr als 1 Million Einwohner haben.

```
(SELECT Name FROM Country WHERE Area > 100000)
MINUS
(SELECT Name FROM Country WHERE Population >
1000000) ;
```

SQL Unterabfragen: Division

- Die **Division** ist im SQL-Standard nicht enthalten
- Der "All-Quantor" wird gemäß Prädikatenlehre über die doppelte Verneinung abgebildet.
- Beispiel: "Die Verkäufer, die alle Produkte verkauft haben" wird zu "Die Verkäufer, für die es kein Produkt gibt, dass sie nicht verkauft haben"



```
SELECT * FROM Verkaeuer WHERE NOT EXISTS
  (SELECT PNr FROM Produkt WHERE NOT EXISTS
    (SELECT PNr FROM Verkauf
      WHERE Verkauf.PNr = Produkt.PNr
      AND Verkauf.VNr = Verkaeuer.VNr) ) ;
```

Verkaeuer		Verkauf			Produkt			
VNr	Name	Nr	VNr	PNr	VNr	Name	VNr	Name
1	Udo	1	1	1	1	Hose	1	Udo
2	Uwe	2	1	2	2	Rock		
3	Ulf	3	2	2	3	Anzug		
		4	1	3				



VNr	Name
1	Udo

Übungsaufgaben Unterabfragen (07.1)

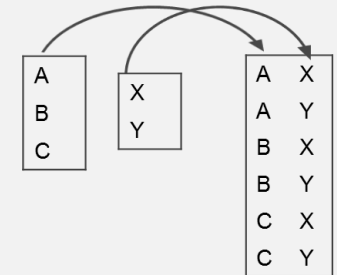
- a) Geben Sie die Namen aller Länder in Europa aus.
- b) Geben Sie den Namen aller Städte mit mehr als 500.000 Einwohnern in Europa aus.
- c) Ermitteln Sie die einwohnermäßig fünf größten Städte in Europa.
- d) Geben Sie den Namen aller Nachbarländer von Deutschland aus.
- e) Ermitteln Sie alle Länder, die eine größere Fläche als China haben.
- f) Geben Sie die Kürzel aller Länder an, die in Europa und Asien liegen.
- g) Geben Sie alle Länder an, von denen die Datenbank weiß, dass es einen Ort südlich des Äquators in diesem Land gibt.
- h) (Zum Knobeln): Welche Länder sind in mindestens einer Organisation Mitglied, in der auch Deutschland (Code 'D' genügt) Mitglied ist?

Themenübersicht

- Grundbegriffe und Datenbankentwurf
- Entity-Relationship-Modelle
- Relationales Datenbankmodell
- Normalisierung
- Arbeiten mit relationalen Datenbanken (SQL)
 - **DQL (Data Query Language)**
 - Einfache SQL-Abfragen
 - NULL-Werte
 - Unterabfragen
 -  **JOINS**
 - GROUP BY
 - DML
 - DDL
 - DAL

SQL-JOINS: CROSS JOIN

- Der **CROSS JOIN** bildet das Kreuzprodukt (kartesische Produkt, siehe Relationen Algebra). Doppelte Datensätze werden im Gegensatz zu allen folgenden Joins nicht entfernt.



SELECT *
FROM Verkäufe CROSS JOIN Produktliste

SELECT *
FROM Verkäufe, Produktliste – implizite, sehr verbreitete Schreibweise!

Verkäufe

Verkäufer	Produkt	Käufer
Meier	Hose	Schmidt
Müller	Rock	Schmidt
Meier	Hose	Schulz

Produktliste

Produkt	Preis	Klasse
Hose	100	B
Rock	200	A



Verkäufe × *Produktliste*

Verkäufer	Produkt	Käufer	Produkt	Preis	Klasse
Meier	Hose	Schmidt	Hose	100	B
Meier	Hose	Schmidt	Rock	200	A
Müller	Rock	Schmidt	Hose	100	B
Müller	Rock	Schmidt	Rock	200	A
Meier	Hose	Schulz	Hose	100	B
Meier	Hose	Schulz	Rock	200	A

SQL-JOINS: INNER JOIN

- Beim **INNER JOIN** werden erweiternd zum Kreuzprodukt nur solche Tupel ausgewählt, die in einer Beziehung zueinander stehen.
 - Verbindet Datensätze aus zwei Tabellen, sobald ein gemeinsames Feld dieselben Werte enthält.
 - WHERE-Bedingung: definiert weitere Ergebnis-Selektionen
 - ON-Klausel definiert die verbindenden Spalten

```
SELECT *  
FROM Verkaeuer INNER JOIN Verkauf  
    ON Verkaeuer.VNr = Verkauf.VNr
```


SQL-JOINS: INNER JOIN

```
SELECT *
FROM Verkaeuer INNER JOIN Verkauf
    ON Verkaeuer.VNr = Verkauf.VNr
```

- Alternativ

```
SELECT *
FROM Verkaeuer JOIN Verkauf
    ON Verkaeuer.VNr = Verkauf.VNr
```

```
SELECT *
FROM Verkaeuer, Verkauf
WHERE Verkaeuer.Vnr = Verkauf.Vnr;
```

Kreuzprodukt mit
WHERE-Bedingung

Verkaeuer		Verkauf		
VNr	Name	Nr	VNr	Produkt
1	Udo	1	1	Hose
2	Uwe	2	1	Rock
3	Ulf	3	2	Schal
		4		Anzug




VNr	Name	Nr	VNr	Produkt
1	Udo	1	1	Hose
1	Udo	2	1	Rock
2	Uwe	3	2	Schal

SQL-JOINS: LEFT OUTER JOIN, LEFT JOIN

- Mit einem LEFT OUTER JOIN wird eine sogenannte linke Inklusionsverknüpfung erstellt. Linke Inklusionsverknüpfungen schließen alle Datensätze aus der ersten (linken) Tabelle ein, auch wenn keine entsprechenden Werte für Datensätze in der zweiten Tabelle existieren.

```
SELECT *
FROM Verkaeuer LEFT JOIN Verkauf
ON Verkaeuer.VNr = Verkauf.VNr
```

Verkaeuer		Verkauf							
VNr	Name	Nr	VNr	Produkt					
1	Udo	1	1	Hose					
2	Uwe	2	1	Rock					
3	Ulf	3	2	Schal					
		4		Anzug					




VNr	Name	Nr	VNr	Produkt
1	Udo	1	1	Hose
1	Udo	2	1	Rock
2	Uwe	3	2	Schal
3	Ulf			

SQL – JOINS: RIGHT OUTER JOIN, RIGHT JOIN

- Mit einem RIGHT OUTER JOIN wird eine sogenannte rechte Inklusionsverknüpfung erstellt. Rechte Inklusionsverknüpfungen schließen alle Datensätze aus der zweiten (rechten) Tabelle ein, auch wenn keine entsprechenden Werte für Datensätze in der ersten Tabelle existieren.

```
SELECT *
FROM Verkaeuer RIGHT JOIN Verkauf
ON Verkaeuer.VNr = Verkauf.VNr
```

Verkaeuer		Verkauf							
VNr	Name	Nr	VNr	Produkt					
1	Udo	1	1	Hose					
2	Uwe	2	1	Rock					
3	Ulf	3	2	Schal					
		4		Anzug					



VNr	Name	Nr	VNr	Produkt
1	Udo	1	1	Hose
1	Udo	2	1	Rock
2	Uwe	3	2	Schal
		4		Anzug

SQL – JOINS: FULL OUTER JOIN

- Der FULL OUTER JOIN ist eine Kombination von LEFT OUTER JOIN und RIGHT OUTER JOIN.

```
SELECT *
FROM Verkaeuer FULL JOIN Verkauf
ON Verkaeuer.VNr = Verkauf.VNr
```

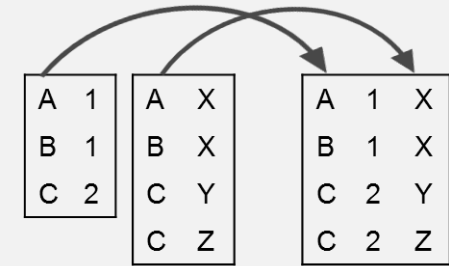
Verkaeuer		Verkauf		
VNr	Name	Nr	VNr	Produkt
1	Udo	1	1	Hose
2	Uwe	2	1	Rock
3	Ulf	3	2	Schal
		4		Anzug



VNr	Name	Nr	VNr	Produkt
1	Udo	1	1	Hose
1	Udo	2	1	Rock
2	Uwe	3	2	Schal
3	Ulf			
		4		Anzug

SQL – JOINS: NATURAL JOIN


- Der **NATURAL JOIN** verbindet die Tabellen automatisch über alle gleichnamige Spalten und entfernt doppelte Spalten.
- Ebenso gibt es den NATURAL LEFT JOIN, NATURAL RIGHT JOIN und den NATURAL FULL JOIN, die die Ergebnismengen gemäß den vorherigen JOINS mit NULL auffüllen.



SELECT *

FROM Verkaeuer NATURAL FULL JOIN Verkauf

<i>Verkaeuer</i>		<i>Verkauf</i>						
VNr	Name	Nr	VNr	Produkt				
1	Udo	1	1	Hose				
2	Uwe	2	1	Rock				
3	Ulf	3	2	Schal				
		4		Anzug				



VNr	Name	Nr	Produkt
1	Udo	1	Hose
1	Udo	2	Rock
2	Uwe	3	Schal
3	Ulf		
		4	Anzug

Übungsaufgabe Notenverwaltung

- Formulieren Sie die folgenden Textzeilen jeweils als SQL-Abfragen

- Geben Sie die Namen der Studierenden aus, die eine Prüfung im Fach "Wahl1" gemacht haben.
- Geben Sie den Titel der Veranstaltung und die zugehörige Note für alle Prüfungen, die Simson gemacht hat, aus.
- Geben Sie eine Liste aller Titel von Veranstaltungen mit den bisher in den Prüfungen erreichten Noten (Ausgabe: Titel, Note) aus.
- Geben Sie die Anzahl der Studierenden aus, die bereits eine Prüfung im Fach "Datenbanken" gemacht haben.
- Geben Sie die Namen aller Dozenten aus, die mindestens zwei Veranstaltungen anbieten.
- Geben Sie die Durchschnittsnote für alle Fächer zusammen aus, die von Hinz unterrichtet wurden.

Student

<u>MatrNr</u>	Name
42	Simson
43	Millhus
44	Monz

Pruefung

<u>MatrNr</u>	<u>Fach</u>	Note
42	Wahl1	3,0
42	DB	1,7
43	Wahl2	4,0
43	DB	1,3
44	Wahl1	5,0

Veranstaltung

<u>Kürzel</u>	Titel	Dozent
Wahl1	Controlling	Hinz
Wahl2	Java	Hinz
DB	Datenbanken	Kunz

aus: Kleuker, Grundkurs Datenbankentwicklung, S. 171

Weitere Übungsaufgabe zu Joins (Aufgabe 07.2) (1/2)

- Für viele Abfragen benötigt man mehrere Tabellen.
- Beispiel: Wir wollen aus der Mondial-Datenbank die Namen aller Länder herausfinden, die in Europa liegen. Die Zuordnung Land ↔ Kontinent erfolgt über die Tabelle "Encompasses":

Encompasses

Country	Continent	Percentage
A	Europe	100
AFG	Asia	100
AG	America	100

- Problem: Hier ist nur der Primärschlüssel der Tabelle "Country", also das Kürzel angegeben. Um nun den Namen herauszufinden, müssen wir beide Tabellen miteinander verknüpfen:

```
SELECT country.name
FROM country, encompasses
WHERE country.code = encompasses.country
      AND encompasses.continent = 'Europe';
```

Weitere Übungsaufgabe zu Joins (Aufgabe 07.2) (2/2)

- Aufgaben:

- Ermitteln Sie den Ländernamen, den Namen der zugehörigen Hauptstadt und die Einwohnerzahl dieser Hauptstadt aller Länder in Amerika sortiert nach Ländernamen.
- Geben Sie die Namen aller Länder aus, deren Hauptstadt weniger als 500.000 Einwohner hat.

Hinweis: Hierfür wird auch die Tabelle "City" benötigt:

City

Name	Country	Province	Population	Longitude	Latitude
Aachen	D	Nordrhein Westfalen	247113		
Aalborg	DK	Denmark	113865	10	57
Aarau	CH	AG			

Themenübersicht

- Grundbegriffe und Datenbankentwurf
- Entity-Relationship-Modelle
- Relationales Datenbankmodell
- Normalisierung
- Arbeiten mit relationalen Datenbanken (SQL)
 - **DQL (Data Query Language)**
 - Einfache SQL-Abfragen
 - NULL-Werte
 - Unterabfragen
 - JOINS
 - ➡ **GROUP BY**
 - DML
 - DDL
 - DAL

GROUP BY

- Aggregatsfunktionen (Min, Max, Avg,...) wurden bereits vorgestellt.
- Das Zusammenfassen bestimmter Teilmengen der Tupel einer Relation zu Gruppen erlaubt nun weitergehende Auswertungen. Damit ist es nun z.B. möglich, mit nur einer SQL-Abfrage auszugeben, wie viele Einwohner die gespeicherten Städte eines Landes haben.
- Die GROUP BY – Gruppierung wird an die bestehende SELECT-Abfrage drangehängt. Zum Beispiel:

```
SELECT City.Country, SUM(City.Population)
FROM City
GROUP BY City.Country
```

GROUP BY

- Weiteres Beispiel: Gebe die Länder mit der durchschnittlichen Einwohnerzahl in den aufgeführten Städten aus.

```
SELECT Country.name, AVG(City.population)
FROM Country, City
WHERE County.code = City.Country
GROUP BY Country.name
```

GROUP BY mit Bedingung (HAVING)

- Wenn beispielsweise die zuvor aufgeführte Abfrage "Gebe die Länder mit der durchschnittlichen Einwohnerzahl in den aufgeführten Städten aus" noch nachträglich selektiert werden soll, da nur die Länder ausgegeben werden sollen, deren Städte durchschnittlich weniger als 10.000 Einwohner haben, benutzen wir die HAVING-Klausel:

```
SELECT Country.name, AVG(City.population)
FROM Country, City
WHERE County.code = City.Country
GROUP BY Country.name
HAVING AVG(City.population) < 10000;
```

Auswertungsreihenfolge

- Relevant für Ergebnis und Zwischenergebnisdatensmengen

Schlüsselwort	Auswertungsreihenfolge	Inhalt
SELECT	6	Attribute, Aggregatsfunktionen
FROM	1	Liste von Tabellen, deren Kreuzprodukt betrachtet wird
WHERE	2	Boolesche Bedingung, zur Auswertung von Zeilen der Tabellen (optional)
GROUP BY	3	Liste von Attributen, nach denen gruppiert wird (optional)
HAVING	4	Boolesche Bedingung mit Attributen aus der GROUP-BY-Zeile oder Aggregatsfunktionen zur Auswahl von Gruppen (optional)
ORDER BY	5	Attribute (oder Aggregatsfunktionen bei GROUP-BY) für Sortierreihenfolge bei der Ausgabe (optional)

Übungsaufgabe GROUP BY 1 (Aufgabe 07.3)

- Warum ist die folgende SQL-Anfrage fehlerhaft?
Was ist die intuitive Begründung für diesen konkreten Fall?
Was ist die technische Begründung für den allgemeinen Fall?

```
SELECT Country, Population  
FROM City  
GROUP BY Country;
```

Übungsaufgabe GROUP BY 2 (Aufgabe 07.4)

- a) Welche Länder sind Mitglied in mehr als 70 Organisationen?
- b) Geben Sie für jede Organisation (Name) die Summe der Einwohner der Mitgliedstaaten an (kleinste zuerst).
- c) Geben Sie für alle Länder, für die mehr als 100 Städte in der Datenbank eingetragen sind, die durchschnittliche Einwohnerzahl dieser Städte an.
- d) Geben Sie alle Städtenamen aus, die mindestens dreimal in der Datenbank vorkommen
- e) Geben Sie pro Land die Anzahl der Städte aus, für die keine Werte für Longitude und Latitude eingetragen sind.

Übungsaufgabe GROUP BY 3 (Aufgabe 07.5)

Probieren Sie die folgenden SQL-Anfragen aus und erläutern Sie deren Bedeutung:

- a)

```
SELECT Country, AVG (Population)
FROM City
GROUP BY Country;
```
- b)

```
SELECT Country, COUNT (*) AS n
FROM City
GROUP BY Country
ORDER BY n DESC;
```
- c)

```
SELECT Country, MAX (Population)
FROM City
GROUP BY Country
HAVING Count (*) >= 2;
```


Zusammenfassung DQL II

- SQL Unterabfragen
 - Allgemein
 - IN
 - EXISTS
 - UNION
 - INTERSECT
 - DIVISION - Doppelte Verneinung
- JOINS
 - CROSS JOIN
 - INNER JOIN
 - LEFT OUTER JOIN, LEFT JOIN
 - RIGHT OUTER JOIN, RIGHT JOIN
 - NATURAL JOIN
- GROUP BY