Zürcher Hochschule
für Angewandte Wissenschaften

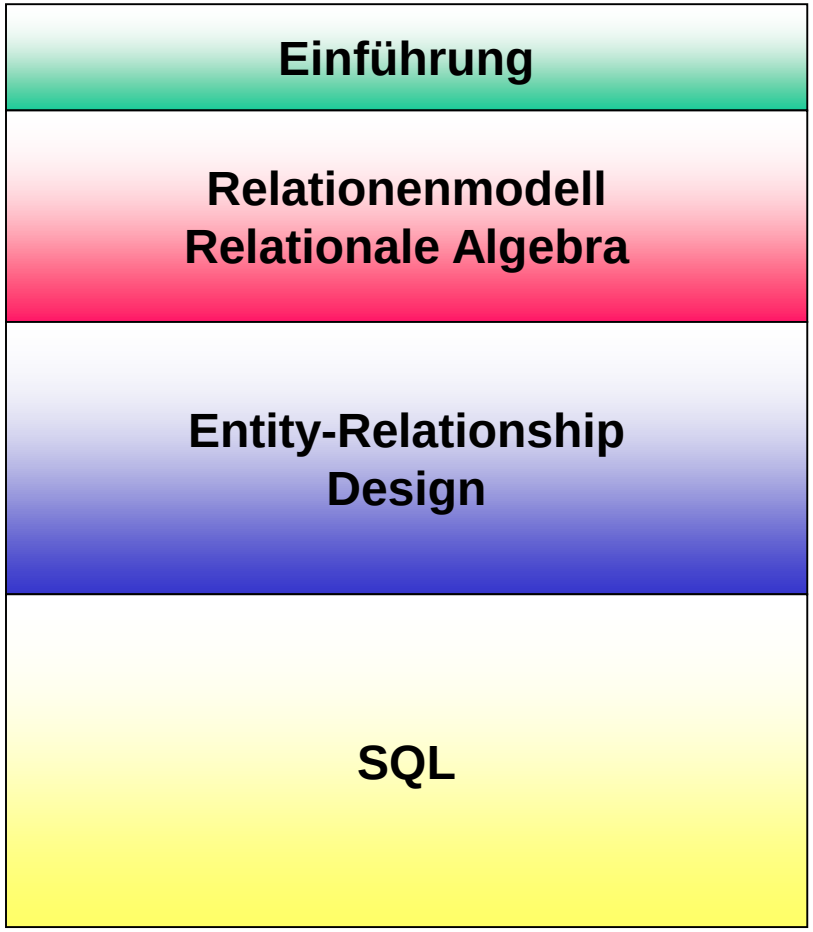
School of
Engineering
InIT Institut für angewandte
Informationstechnologie

DAB1 – Datenbanken 1

Dr. Daniel Aebi (aebd@zhaw.ch)

Lektion 12: SQL – DQL (Fortsetzung), NULLs

Wo stehen wir?



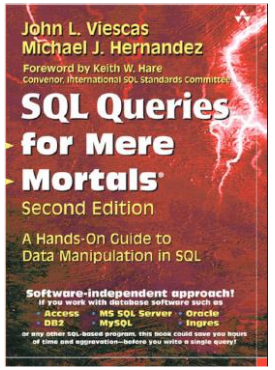
"You are here"

SQL: Buchempfehlung

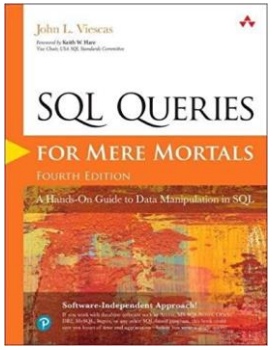
- Es existiert eine Vielzahl von SQL-Lehrbüchern. Selber entscheiden!
- Aufpassen auf «Dialekt» (ein Lehrbuch zu T-SQL nützt nicht sehr viel beim Einsatz einer Oracle-DB…).

- Vorschlag als Ergänzung zum Vorlesungsstoff:

  (2. Auflage ist gratis, haben Sie bereits im Praktikum 1 vom USB-Stick des Dozenten kopiert).



- Aktuelle 4. Auflage:
  - J. Viescas: SQL Queries for Mere Mortals. Addison-Wesley. 4. Auflage, ISBN 978-0134858333



Rückblick

- Abbildung ER-Schema → SQL
- Probleme beim direkten Abbilden (ohne Beziehungstypen als Tabellen)
- Grundaufbau der SELECT-Anweisung:

SELECT …	→ welche Attribute (entspricht Projektion)
FROM …	→ aus welchen Tabellen (eine oder Verbund mehrerer)
WHERE …	→ gemäss welchen Kriterien (entspricht Selektion)
GROUP BY …	→ wie zusammengefasst
ORDER BY …	→ wie geordnet

Lernziele Lektion 12

- Eigenheiten von NULL's kennen.
- Weitere Aspekte der SQL-SELECT-Anweisung kennen und anwenden können.

NULL's

NULL's

- Was ist das Ausgangsproblem? → Fehlende Information!

- Was heisst das? Wir brauchen in der Praxis Möglichkeiten, um verschiedene Situationen abzudecken:
  - Ein Wert ist unbekannt
  - Ein Wert ist nicht anwendbar
  - Ein Wert existiert nicht
  - Ein Wert ist nicht definiert
  - Ein Wert ist die leere Menge
  - … (Achtung: das ist nicht einfach alles dasselbe!)

NULL's

- Für ALLE diese Situationen wird in SQL (nicht im Relationenmodell!) mit dem «Platzhalter» NULL gearbeitet (man spricht oft – zu unrecht – von NULL-Werten, NULL ist aber kein Wert).

- «NULL» steht für «unbekannt» (und kann deshalb keinen Wert haben).

- Was macht denn nun die Sache so problematisch?
  → Die Frage, wie mit diesem «unbekannt» umgegangen werden soll.

- Bisher (d.h. in der relationalen Algebra) hatten wir es mit Prädikaten zu tun, die entweder zu «wahr» oder «falsch» ausgewertet werden (→ zweiwertige Logik).

NULL's

- Bisher (2VL, two-valued-logic): zwei Wahrheitswerte, wahr und falsch mit folgenden Operationen:

  AND ("logisches und", $\wedge$)
  OR ("logisches oder", $\vee$)
  NOT ("logisches nicht", $\neg$)

- Mit folgender Bedeutung ("Wahrheitstabelle"):

A	B	$\neg A$	$A \wedge B$	$A \vee B$
W	W	F	W	W
W	F	F	F	W
F	W	W	F	W
F	F	W	F	F

NULL's

- Mit NULL's gibt es drei Zustände (3VL): wahr, falsch und «unbekannt». Mit folgender Bedeutung («Wahrheitstabellen»):

a und b			
a \ b	f	u	w
f	f	f	f
u	f	u	u
w	f	u	w

a oder b			
a \ b	f	u	w
f	f	u	w
u	u	u	w
w	w	w	w

nicht a	
a	¬ *a*
f	w
u	u
w	f

- Interessanter Gedanke: Sei x eine Variable für einen Wahrheitswert. Dann heisst (umgangssprachlich) x ist nicht 'wahr', dass x entweder 'falsch' oder 'unbekannt' ist (in der 3VL)

- Aber (Logik): $x = \neg w \rightarrow x = f$ (x ist NICHT 'wahr' heisst x = 'falsch')

NULL's

- Wir haben zwei Probleme:
  1. 3VL ist an sich problematisch (siehe vorherige Überlegung) und auch schwer zu verstehen
  2. 3VL ist in SQL unsauber implementiert!

- Was ist denn so schlimm? → Ein paar Beispiele

Wer es genauer wissen möchte dem sei folgendes Video empfohlen:

Chris Date talks about "The Problem of Missing Information"
http://www.youtube.com/watch?v=kU-MXf2TsPE

NULL's

- Es sei eine einfache Kontotabelle gegeben:

```
CREATE TABLE Konto (
  KontoNr      integer       NOT NULL PRIMARY KEY,
  Bezeichnung  varchar(100)  NOT NULL,
  Ueberzogen   char(1)           NULL,
  Saldo        decimal(10,2)     NULL);
```

- Mit folgenden Daten:

KontoNr	Bezeichnung	Ueberzogen	Saldo
10	Kassa	N	100.00
20	Post	N	200.00
30	Bank	NULL	NULL
40	Debitoren	J	300.00

NULL's

- Wie viele Kontos haben wir?

```
SELECT COUNT(*)           FROM Konto;
SELECT COUNT(KontoNr)     FROM Konto;
SELECT COUNT(Ueberzogen) FROM Konto;
```

- Wir wollen alles:

```
SELECT * FROM Konto;

SELECT * FROM Konto
WHERE Ueberzogen = 'N'
UNION
SELECT * FROM Konto
WHERE Ueberzogen <> 'N';

analog:

SELECT * FROM Konto
WHERE (Ueberzogen = 'N') OR NOT (Ueberzogen = 'N');
```

KontoNr	Bezeichnung	Ueberzogen	Saldo
10	Kassa	N	100.00
20	Post	N	200.00
30	Bank	NULL	NULL
40	Debitoren	J	300.00

KontoNr	Bezeichnung	Ueberzogen	Saldo
10	Kassa	N	100.00
20	Post	N	200.00
40	Debitoren	J	300.00

NULL's

- WHERE-Klauseln liefern in SQL nur Tupel deren Selektionsprädikat auf «wahr» ausgewertet wird!

- Wir wollen den Gesamtsaldo:

```
SELECT SUM(Saldo) AS SaldoTotal FROM Konto;
```

SaldoTotal
600.00

KontoNr	Bezeichnung	Ueberzogen	Saldo
10	Kassa	N	100.00
20	Post	N	200.00
30	Bank	NULL	NULL
40	Debitoren	J	300.00

- Warum ist eigentlich 100 + 200 + 300 + unbekannt = 600???

- Wir wollen den Durchschnitt:

```
SELECT AVG(Saldo) AS SaldoDurchschnitt FROM Konto;
```

SaldoDurchschnitt
200.000000

- Warum? Wir haben doch 4 Kontos… (inkonsistent zur Summe oben…)

NULL's

- Wir wollen die Ausgabe sortieren:

```
SELECT * FROM Konto ORDER BY Ueberzogen;
```

KontoNr	Bezeichnung	Ueberzogen	Saldo
30	Bank	NULL	NULL
40	Debitoren	J	300.00
10	Kassa	N	100.00
20	Post	N	200.00

- Auf anderen Systemen ergibt das:

KontoNr	Bezeichnung	Ueberzogen	Saldo
40	Debitoren	J	300.00
10	Kassa	N	100.00
20	Post	N	200.00
30	Bank	NULL	NULL

- Viele weitere Probleme:
  - String-Konkatenation (Bsp.: Surname ¦¦ Middlename ¦¦ Lastname)
  - Gruppierung
  - Vergleiche mit: A IS [NOT] NULL, aber: UPDATE … SET A = NULL
  - …

NULL's – Lösungsansätze (Beispiele)

- Vermeiden (mindestens in Schlüsselattributen, in PK zwingend)
- Vorgabewerte verwenden:
  - `CREATE TABLE T(A integer NULL DEFAULT -1)`
- In Abfragen "umwandeln" in Defaultwert:
  - `SELECT COALESCE(A,'n/a') FROM …` (ISNULL, IFNULL, NVL, …)
  - `SELECT CASE A WHEN NULL THEN 0 ELSE A END AS …`
- In Abfragen berücksichtigen («3VL-Fragen stellen»):
  - `SELECT … FROM … WHERE A = 5 OR A IS NULL`
- Andere (komplexere!) Datenstruktur:
  - Handorgel (EAV)
  - …

Wenn NULL's nötig/vorhanden … → AUFPASSEN!!

Subquery: Aggregatfunktionen ohne Gruppierung

- Unterschied zwischen COUNT(*) und COUNT(<attributName>):
  - COUNT(<attributName>) zählt nur diejenigen Tupel, bei denen der Wert des Attributs nicht NULL ist
  - COUNT(*) zählt alle Tupel (es gibt kein Tupel, bei dem alle Attribute gleichzeitig NULL sein können)

- Weitere Option:

```
SELECT COUNT (DISTINCT TITEL)
FROM CD;
```

  zählt die Anzahl verschiedener Titel (falls es zufällig verschiedene CDs mit demselben Titel geben sollte)

- Aggregatfunktionen ohne Gruppierung liefern eine Tabelle mit einem Tupel und einem Attribut (also ein Attributwert). Das ist aber eine richtige Tabelle, die weiterverarbeitet werden kann. Kann überall dort eingesetzt werden wo ein einzelner Wert erwartet wird.

Subquery: Aggregatfunktionen mit Gruppierung

- Beispiel CDShop: Anzahl Bestellungen pro Kunde

```
SELECT kdNr, COUNT(*) AS AnzahlBestellungen
FROM Kaufhistorie
GROUP BY kdNr;
```

Liefert eine Tabelle mit einem Tupel pro kdNr, d.h. ein Tupel pro Kunde

- Komplizierteres Beispiel: Bestellumsatz pro Kunde, unabhängig von der einzelnen Bestellung

```
SELECT kdNr, SUM(menge * preis) AS Umsatz
FROM Bestellposition JOIN Kaufhistorie ON bestNr = bNr
GROUP BY kdNr;
```

Subquery: Aggregatfunktionen mit Gruppierung

Ausschliessen einzelner Gruppen aus dem Resultat:

Beispiel CDShop: Bestellumsatz pro Kunde, falls der Umsatz Fr. 500.- übersteigt

```
SELECT kdNr, SUM(menge * preis) AS Umsatz
FROM Bestellposition JOIN KaufHistorie ON bestNr = bNr
GROUP BY kdNr
HAVING SUM(menge * preis)[1] > 500;
```

[1] SUM(menge * preis) wird zu Umsatz umbenannt. Das ist eine der letzten Aktionen, die das RDBMS in einer SQL-Abfrage vornimmt. Innerhalb der Abfrage muss das Attribut deshalb noch mit dem ursprünglichen Namen angesprochen werden.
→ Reihenfolge der Abarbeitung

Subquery: Aggregatfunktionen mit Gruppierung

- Reihenfolge, in der SQL eine Abfrage bearbeitet:
  1. FROM
  2. WHERE
  3. GROUP BY
  4. HAVING
  5. SELECT
  6. ORDER BY

- Diese Reihenfolge ist wichtig, um den Unterschied zwischen WHERE und HAVING zu verstehen:
  - WHERE wirkt auf jedes Tupel der Datenquelle, unabhängig von einer allfälligen Gruppierung
  - Gruppiert wird die durch WHERE bereinigte Datenquelle
  - HAVING schliesst anschliessend an die Gruppierung ganze Gruppen aus

Subquery: Aggregatfunktionen mit Gruppierung

Beispiel Company, um den Unterschied zu zeigen:

Liste aller Abteilungen, welche mindestens 3 Angestellte beschäftigen, die mehr als 35000.- verdienen

```
SELECT dno, COUNT(*) AS Anzahl
FROM Employee JOIN WorksFor ON ssn = wssn
WHERE salary > 35000
GROUP BY dno
HAVING COUNT(*) > 2;
```

Schritt für Schritt:

```
1) SELECT wssn, salary, dno
   FROM Employee JOIN WorksFor ON ssn = wssn
   ORDER BY dno[1];
```

[1] nach Abteilungsnummer aufsteigend sortiert (der besseren Lesbarkeit halber)

Subquery: Aggregatfunktionen mit Gruppierung

wssn	salary	dno
888665555	55000	1
111111111	41000	4
987654321	43000	4
987987987	25000	4
999887777	25000	4
111223333	22500	5
123123123	21000	5
123456789	30000	5
333221111	26000	5
333445555	40000	5
453453453	25000	5
666884444	38000	5
432156789	39000	7
776655443	57000	7
890321654	120000	7

Subquery: Aggregatfunktionen mit Gruppierung


```
2) SELECT wssn, salary, dno
   FROM Employee JOIN WorksFor ON ssn = wssn
   WHERE salary > 35000
   ORDER BY dno;
```

wssn	salary	dno
888665555	55000	1
111111111	41000	4
987654321	43000	4
333445555	40000	5
666884444	38000	5
432156789	39000	7
776655443	57000	7
890321654	120000	7

Subquery: Aggregatfunktionen mit Gruppierung


```
3) SELECT dno, COUNT(*) AS Anzahl
   FROM Employee JOIN WorksFor ON ssn = wssn
   WHERE salary > 35000
   GROUP BY dno
   ORDER BY dno;
```

dno	Anzahl
1	1
4	2
5	2
7	3

Man erkennt gut, dass nach der WHERE-Klausel gruppiert wurde

Subquery: Aggregatfunktionen mit Gruppierung

- Letzter Schritt: schliesse die Gruppen (=Abteilungen) aus, welche nicht mindestens 3 Angestellte zählen

```
4) SELECT dno, COUNT(*) AS Anzahl
   FROM Employee JOIN WorksFor ON ssn = wssn
   WHERE salary > 35000
   GROUP BY dno
   HAVING COUNT(*) > 2;
```

dno	Anzahl
7	3

IN-Prädikat

- Das "IN" Prädikat hat zwei Grundformen:
  "IN <Werteliste>" und "IN <Subquery>".

- Erste Form:

```
SELECT Name, Vorname, Strasse, Gebtag
FROM Besucher
WHERE Name IN ('Meier', 'Müller', 'Sonderegger')
```

- Zweite Form:

```
SELECT Name, Strasse, Wirtsname
FROM Restaurant
WHERE Name IN (SELECT Rname
               FROM Sortiment
               WHERE Bsorte = 'Sorte1' AND AnLager > 0)
```

IN-Prädikat

- In manchen Systemen (Teil des SQL92-Standards!) ist auch eine allgemeinere Form möglich (auch in MySQL):

```
SELECT Name, Vorname, Strasse, Gebtag
FROM Besucher
WHERE (Name, Vorname) IN (SELECT Bname, Bvorname
                          FROM Gast
                          WHERE Rname='Ochsen');
```

Negierter IN-Operator

- Achtung bei negiertem IN-Operator!
- Beispiel Vorname, Name aller Angestellten, welche keine Angehörigen desselben Geschlechts haben:

```
SELECT fName, lName
FROM Employee
WHERE (ssn, sex) NOT IN
                  (SELECT Dependent.ssn, Dependent.sex
                   FROM Dependent
                   WHERE Dependent.ssn = Employee.ssn);
```

- Liefert nicht nur die Angestellten, welche eben keine Angehörigen desselben Geschlechts haben, sondern auch jene, welche überhaupt keine Angehörigen haben. War das gemeint??

ALL, SOME/ANY-Operator

- Das Prädikat "ALL" kann zusammen mit einem Vergleichsoperator (=, <>, <, <=, >, >=) bei Subqueries eingesetzt werden.
- Beispiel:

```
SELECT *
FROM Gast
WHERE Frequenz > ALL (SELECT Frequenz
                      FROM Gast
                      WHERE Rname='Ochsen');
```

- Kann scheinbar auch folgendermassen geschrieben werden:

```
SELECT *
FROM Gast g1
WHERE NOT EXISTS(SELECT 1
         FROM Gast g2
         WHERE Rname='Ochsen'
         AND g2.Frequenz >= g1.Frequenz)
```

ALL, SOME/ANY-Operator

- Die beiden Formen sind bei Vorkommen von NULL nicht identisch.

  Beispiel: Wenn mindestens eine Frequenz eines Gasts des Restaurants Ochsen "NULL" ist, liefert die Form mit ALL nichts.

- Die Form mit EXISTS liefert hingegen für NULL-Werte “false", d.h. bei NOT EXISTS "true" → Alle Gäste mit Frequenzen oberhalb der bekannten Frequenzen, sowie Gäste mit Frequenzen "NULL" werden gelistet!

ALL, SOME/ANY-Operator

- Auch das Prädikat «ANY» kann zusammen mit einem Vergleichsoperator (=, <>, <, <=, >, >=) bei Subqueries eingesetzt werden. SOME ist ein Synonym für ANY.

- Beispiel:

```
SELECT *
FROM Gast
WHERE Frequenz > ANY(SELECT Frequenz
                     FROM Gast
                     WHERE Rname='Ochsen');
```

- = ANY entspricht IN

Weitere JOINS

- Wir haben bisher vor allem den "NATURAL JOIN" und den "Theta-JOIN" besprochen.
- Es stehen uns bekanntlich weitere Formen des JOINS zur Verfügung:
- "CROSS JOIN“ (Produkt):

```
SELECT * FROM a CROSS JOIN b;
```

- Oder auch (üblichere Form):

```
SELECT * FROM a, b;
```

OUTER JOINS

- In NATURAL-JOINs und Theta-JOINS zweier Bags r und s sind Tupel aus r, welche keine Entsprechung in s haben gemäss der Join-Kriterien, unsichtbar ($\rightarrow$ "fallen heraus")

- OUTER JOINs sollen auch "nonmatching"-Tupel berücksichtigen

OUTER JOINS

- [NATURAL] LEFT OUTER JOIN:
- r ⟕ s
- Alle Tupel, die bei einem normalen Join "erzeugt" werden, plus die weiteren Tupel aus r, mit NULLs ergänzt.
- Ohne NATURAL: Mit entsprechenden JOIN-Bedingungen.

OUTER JOINS

Gast

Bname	Bvorname	Rname	Frequenz
Zarro	Darween	Löwen	9
Meier	Hans	Bären	5
Meier	Hans	Löwen	4
Meier	Hans	Löwen	4
Anderegg	Ursula	Bären	1

Lieblingsbier

Bname	Bvorname	Bsorte	Bewertung	Literprowoche
Zarro	Darween	Potatsaft	10	7
Meier	Hans	Hopfdrink	4	2
Meier	Hans	Hopfdrink	4	2
Meier	Hans	Malzdrink	3	1

```
SELECT * FROM Gast NATURAL LEFT OUTER JOIN Lieblingsbier;
```

(unvollständiger) Auszug aus Resultat:

Bname	Bvorname	Rname	Frequenz	Bsorte	Bewertung	Literprowoche
Zarro	Darween	Löwen	9	Potatsaft	10	7
Meier	Hans	Bären	5	Hopfdrink	4	2
Meier	Hans	Löwen	4	Hopfdrink	4	2
Meier	Hans	Löwen	4	Malzdrink	3	1
Anderegg	Ursula	Bären	1	null	null	null

OUTER JOINS

- [NATURAL] RIGHT OUTER JOIN:
- r ⟖ s
- Alle Tupel, die bei einem normalen Join "erzeugt" werden, plus die weiteren Tupel aus s, mit NULLs ergänzt.
- Ohne NATURAL: Mit entsprechenden JOIN-Bedingungen.

OUTER JOINS

- Beispiel:

```
SELECT *
FROM Gast NATURAL RIGHT OUTER JOIN Lieblingsbier;
```

- Auszug aus Resultat:

Bname	Bvorname	Rname	Frequenz	Bsorte	Bewertung	Literprowoche
Meier	Anna	Löwen	6	Hopfdrink	8	3
Meier	Anna	Ochsen	1	Hopfdrink	8	3
Schmid	Joseph	null	null	Potatsaft	7	5
Müller	Heinrich	Bären	0	Hopfdrink	10	7
Müller	Heinrich	Rössli	0	Hopfdrink	10	7

OUTER JOINS

- [NATURAL] FULL OUTER JOIN:
- r ⟗ s
- Alle Tupel, die bei einem normalen Join "erzeugt" werden, plus die weiteren Tupel aus r und s, mit NULLs ergänzt.
- Ohne NATURAL: Mit entsprechenden JOIN-Bedingungen.

OUTER JOINS

- Beispiel (geht nicht in MySQL):

```
SELECT *
FROM Gast NATURAL FULL OUTER JOIN Lieblingsbier;
```

- Auszug aus Resultat:

Bname	Bvorname	Rname	Frequenz	Bsorte	Bewertung	Literprowoche
Meier	Anna	Löwen	6	Hopfdrink	8	3
Meier	Anna	Ochsen	1	Hopfdrink	8	3
Schmid	Joseph	null	null	Potatsaft	7	5
Meier	Hans	Löwen	4	Malzdrink	3	1
Anderegg	Ursula	Bären	1	null	null	null

OUTER JOINS

- Outer Joins können gebraucht werden, um "nicht-vorhandene" Information zu suchen:

```
SELECT Bname, Bvorname
FROM Gast NATURAL LEFT OUTER JOIN Lieblingsbier
WHERE Bsorte IS NULL;
```

- (Erinnerung: Nicht "= NULL"!)

OUTER JOINS – Bemerkungen

- OUTER JOINs sind nicht immer effizient
- Es existieren verschiedene (wilde) OUTER JOIN-Syntaxvarianten
- Es werden NULLs eingeführt, eigentlich nur der Darstellung willen

→ OUTER JOINs mit Vorsicht geniessen

SQL Beispiele

- Ein paar weitere SQL-Beispiele:

- Eine Liste aller Besucher mit Name, Vorname und Geburtstag, sowie Angabe, wieviel Liter desjenigen Bieres, welches sie mit der Note 1 bewertet haben, sie pro Woche trinken, sowie der Name des Bieres (Bem.: 1 ist die Bestnote!)

? Woher wird Information benötigt?

SQL Beispiele

- Wir brauchen Information aus den Tabellen 'Besucher' und 'Lieblingsbier‘, beides sind Relationen, da sie je einen Schlüssel haben

- Zusammenhang via Fremdschlüssel <Bname,Bvorname> in 'Lieblingsbier' → Join

? Wie kann der entsprechende Join geschrieben werden?

SQL Beispiele

```
SELECT *
FROM Besucher b, Lieblingsbier l
WHERE b.Name = l.BName AND b.Vorname = l.BVorname
```

Oder (besser):

```
SELECT *
FROM Besucher b
JOIN Lieblingsbier l
ON b.Name = l.Bname AND b.Vorname = l.Bvorname
```

- Ist das Resultat noch eine Relation?

SQL Beispiele

- Der Join zweier Relationen ist wieder eine Relation
- Wir müssen nun die Bedingung mit der Bewertung einbringen:

? Wie lässt sich diese Bedingung hinzufügen?

SQL Beispiele

```
SELECT *
FROM Besucher b, Lieblingsbier l
WHERE b.Name = l.BName AND b.Vorname = l.Bvorname
AND l.Bewertung = 1
```

bzw.:

```
SELECT *
FROM Besucher b JOIN Lieblingsbier l ON b.Name = l.Bname
AND b.Vorname = l.Bvorname
WHERE l.Bewertung = 1
```

? Ist das Resultat noch eine Relation?

SQL Beispiele

- Das Resultat der Selektion einer Relation ist ebenfalls eine Relation. Wir müssen nun noch abschliessend die Projektion anwenden:

```
SELECT Name, Vorname, Gebtag, Bsorte, Literprowoche
FROM Besucher b
JOIN Lieblingsbier l
ON b.Name=l.Bname AND b.Vorname=l.Bvorname
WHERE l.Bewertung=1
```

? Ist das Resultat nun immer noch eine Relation? Warum ja/nein?

SQL Beispiele

- Nach der Projektion könnte ein Bag entstanden sein. Aber:

- <Name, Vorname> ist in Besucher eindeutig → nur eine Gebtag-Angabe pro Kombination
- <Bname, BVorname, Bsorte> ist in Lieblingsbier eindeutig → nur eine Literprowoche-Angabe pro Kombination

→ Resultat ist Relation, Duplikatelimination ist nicht nötig!

? Was passiert, wenn wir stattdessen nach den Bieren mit der Bestnote pro Besucher fragen (statt Note="1")? Kleiner = besser

SQL Beispiele

- Wir müssen die Bedingung "l.Bewertung=1" durch etwas anderes ersetzen. Das "betrachtete" Tupel muss die Eigenschaft haben, dass kein anderes Tupel existiert, das zum gleichen Besucher gehört, und das eine bessere Note verzeichnet hat:

```
SELECT Name, Vorname, Gebtag, Bsorte, Literprowoche
FROM Besucher b, Lieblingsbier l
WHERE b.Name = l.Bname AND b.Vorname = l.BVorname
   AND NOT EXISTS (SELECT 1
          FROM Lieblingsbier l2
          WHERE l.Bname = l2.Bname
          AND l.BVorname = l2.BVorname
          AND l2.Bewertung < l.Bewertung)
```

- Kann pro Besucher mehr als eine Zeile auftreten? Ist das Resultat eine Relation?

SQL Beispiele

- Ein weiteres SQL-Beispiel:

- Gesucht sind alle Restaurants mit Name, Strasse und Suppenpreis, welche mindestens die Biersorten 'Sorte1' und 'Sorte2' im Sortiment haben

? Woher stammt die gesuchte Information?

SQL Beispiele

- Wir brauchen Felder aus der Relation Restaurant

- Lösungsansatz («Pseudo-SQL»):

```
SELECT Name, Strasse, Suppenpreis
FROM Restaurant AS r
WHERE Bedingung(r)
```

- Die Selektionsbedingung Bedingung(r) muss aussagen, dass das betrachtete Restaurant mindestens beide Sorten hat

? Wie können wir die Bedingung ausdrücken?

SQL Beispiele

- Für eine einzelne Sorte:

  «Es existiert ein Eintrag in Sortiment mit dem Namen des Restaurants und der gewünschten Biersorte»

- Für zwei Sorten:

  «Es existiert ein Eintrag in Sortiment mit dem Namen des Restaurants und der gewünschten Sorte 1 UND es existiert ein Eintrag in Sortiment mit dem Namen des Restaurants und der gewünschten Sorte 2»

? In SQL?

SQL Beispiele

```
SELECT r.Name, r.Strasse, r.Suppenpreis
FROM Restaurant AS r
WHERE EXISTS(SELECT 1
      FROM Sortiment AS x
      WHERE x.Rname = r.Name AND x.Bsorte = 'Sorte 1')
AND EXISTS(SELECT 1
      FROM Sortiment AS x
      WHERE x.Rname = r.Name AND x.Bsorte = 'Sorte 2');
```

? Warum enthält das obige Statement zweimal die Variable "x"?

SQL Beispiele

- Die beiden «gebundenen Variablen» x stören sich nicht: sie haben je einen eigenen Scope

- Anders, wenn wir umformulieren: «Es existieren ZWEI Einträge in Sortiment, mit dem gewünschten Restaurant und den jeweils gewünschten Biersorten»

? In SQL?

SQL Beispiele

```
SELECT r.Name, r.Strasse, r.Suppenpreis
FROM Restaurant AS r
WHERE EXISTS(SELECT 1
      FROM Sortiment AS x, Sortiment AS y
      WHERE x.Rname = r.Name AND x.Bsorte='Sorte 1'
      AND y.Rname=r.Name AND y.Bsorte='Sorte 2')
```

Zwei unabhängige EXISTS sind in einem Kreuzprodukt untergebracht worden

? Können wir diese Idee weiterentwickeln, und ganz ohne EXISTS auskommen?

SQL Beispiele

Wir untersuchen das Kreuzprodukt r x s x s

Wir sind also interessiert an:

$\pi_{r.Name,\ r.Strasse,\ r.Suppenpreis}(\sigma(_{x.Rname=r.Name \wedge x.Bsorte='Sorte1' \wedge y.Rname = r.Name \wedge y.Bsorte='Sorte2'}(r \times (s\ AS\ x) \times (s\ AS\ y)))$

? Wird das Resultat eine Relation sein?

SQL Beispiele

- Das Kreuzprodukt ist eine Relation. Ebenso das Resultat nach der Selektion. Wie sieht es nach der Projektion aus?

- r.Name ist Schlüssel in r
- {x.Rname, x.Bsorte} ist Schlüssel in (s AS x)
- {y.Rname, y.Bsorte} ist Schlüssel in (s AS y)

- Da r.Name=x.Rname=y.Rname und x.Bsorte='Sorte1' und y.Bsorte='Sorte2' ist {r.Name,x.Rname,x.Bsorte,y.Rname,y.Bsorte} unique, d.h., jedes r.Name kommt nur einmal vor.
- Daher ist das Resultat eine Relation

? Wie sieht das fertige Statement in SQL aus?

SQL Beispiele

```
SELECT
  r.Name, r.Strasse, r.Suppenpreis
FROM
  Restaurant AS r,
  Sortiment AS x,
  Sortiment AS y
WHERE
  x.Rname = r.Name AND
  x.Bsorte = 'Sorte1' AND
  y.Rname = r.Name AND
  y.Bsorte = 'Sorte2'
```

SQL – DQL: Hörsaalübung (geleitet)

- Starten Sie MySQL-Workbench und formulieren Sie in der Datenbank `classicmodels` folgende Abfragen. Ziehen Sie ggf. die Dokumentation (`MySQLSampleDatabaseSchema.pdf`) der Datenbank bei:

1. Gesucht sind die Kundennummer, der Kundenname sowie die Anzahl Bestellungen pro Kunde

2. Gesucht sind die Bestellnummern und der Bestellwert derjenigen Bestellungen, bei denen dieser Bestellwert mehr als 60000 beträgt

SQL – DQL: Lösungen

Diskutiert im Unterricht. Machen Sie Ihre eigenen Notizen.

Und weiter...

- Das nächste Mal: SQL (Queries, Fortsetzung)