Zürcher Hochschule
für Angewandte Wissenschaften

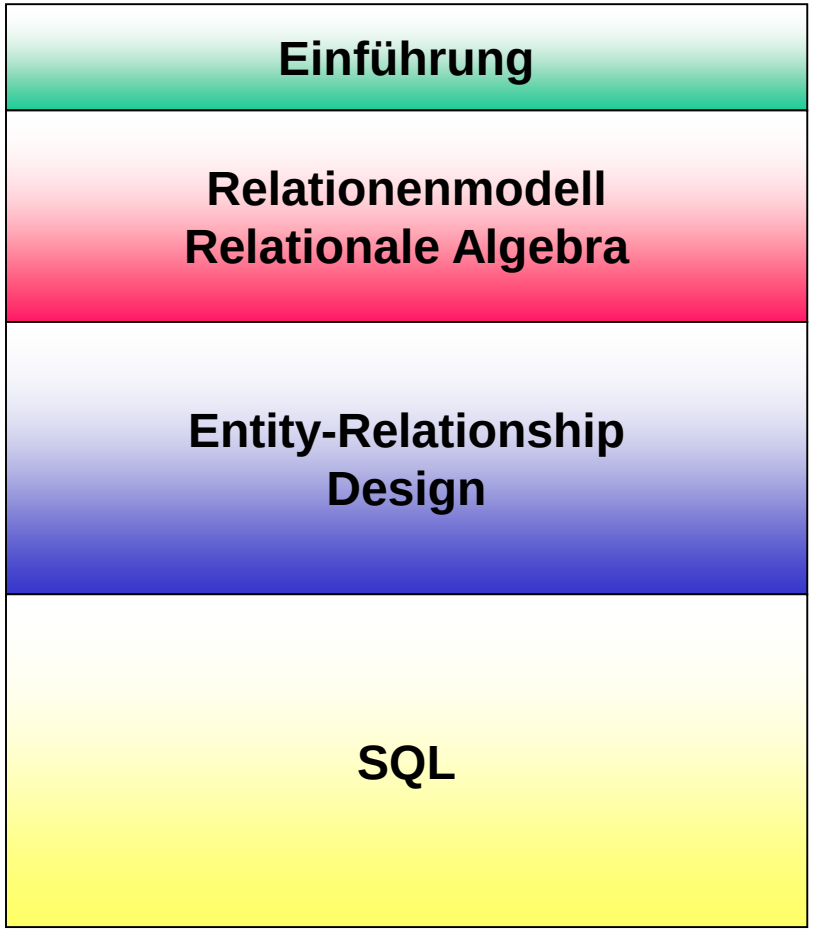
School of
Engineering
InIT Institut für angewandte
Informationstechnologie

DAB1 – Datenbanken 1

Dr. Daniel Aebi (aebd@zhaw.ch)

Lektion 13: SQL – DQL (Abschluss), DDL & DML (Ergänzungen)

Wo stehen wir?



"You are here"

Rückblick

- Aspekte von NULL's
- Aggregation
- Gruppierung, Selektion von Gruppen
- ANY / SOME, ALL

Lernziele Lektion 13

- Nochmals: Die dunkle Seite der Macht (NULL's) …
- Weitere Aspekte der SQL-SELECT-Anweisung kennen und anwenden können
- Das Konzept der Sichten verstehen
- Einige Ergänzungen von DDL-/DML-Anweisungen kennen

SOME/ANY-Operator

- Gegeben: R

r
10

S1

s
21
14
7

S2

s
10
20

- Was ergeben die beiden folgenden Queries für ein Resultat?

1.
```
SELECT r FROM R WHERE r > ANY (SELECT s FROM S1);
```

r
10

2.
```
SELECT r FROM R WHERE r > ANY (SELECT s FROM S2);
```

r

Die dunkle Seite der Macht: NULL's

- Mengen-Vergleiche:
  - `SELECT A FROM S WHERE A NOT IN (SELECT A FROM T)`

- Wenn S \ T eine leere Menge gibt, können wir nicht sagen, ob S ⊆ T, da die Query mit NULL's in T immer die leere Menge zurück gibt.
- Leere Mengen:
  - `SELECT SUM(Suppenpreis) FROM Restaurant WHERE 0=1;` → NULL nicht 0

- Matrix Checksumme:
  - `SELECT SUM(A) + SUM(B)` (→ 8 d.h. NULL = 0)
  - `SELECT SUM(A + B)` (→ 7 d.h. NULL ignoriert)

  → Ergibt nicht das Gleiche

A	B
1	NULL
3	4

← 1 + NULL = NULL

↑ NULL + 4 = 4

Die dunkle Seite der Macht: NULL's

- EXISTS und IN:
  - `WHERE S.A NOT IN (SELECT T.A FROM T)`
  - `WHERE NOT EXISTS (SELECT 1 FROM T WHERE T.A=S.A)`

  → Nicht das Gleiche, da Wahrheitswerte bei IN auch UNKNOWN sein können, jedoch nicht bei EXISTS.

- Durchschnittsberechung:
  - SUM und AVG ignorieren NULL, COUNT nicht
  - Durchschnitt mit AVG <> Durchschnitt mit SUM und COUNT

- Constraints
  - Constraints sind erfüllt, wenn immer diese TRUE oder UNKNOWN ergeben.

- Viele weitere Unterschiede je nach Datenbanksystem…

CASE

- Ermöglicht Fallunterscheidungen:

```
SELECT Bname, Bvorname, SUM(Frequenz) AS AnzahlBesuche,
CASE
   WHEN SUM(frequenz) > 10 THEN ' ist ein Säufer'
   ELSE ' trinkt nicht so viel'
END AS Status
FROM gast GROUP BY bname, bvorname
```

Allgemein:

```
"CASE"
   "WHEN" <Bedingung1> "THEN" <Wert1>
   {"WHEN" <Bedingung2> "THEN" <Wert2>}
   ["ELSE" <Wert3>]
"END"
```

Views – Sichten

- Oft müssen verschiedene Auswertungen auf derselben Datenbasis durchgeführt werden. Manchmal ist es angezeigt, die Struktur der Datenbank vor dem Benutzer zu verbergen.
- So könnte eine mögliche Grundlage sein, dass man Abfragen auf der Basis von Namen, Vornamen, Strasse, Geburtsdatum und Besuchs-frequenz von Gästen des Restaurants Ochsen (und nur von solchen Gästen!) machen will.
- Die Information ist in der Datenbank in zwei Tabellen (Struktur)
- Auswertung:

```
SELECT x.Name, x.Vorname, x.Strasse, x.Gebtag, y.Frequenz
FROM Besucher x, Gast y
WHERE x.Name = y.Bname AND x.Vorname = y.Bvorname
AND y.Rname = 'Ochsen';
```

Views – Sichten

- Diese Auswertung soll nun die Basis bilden für die weitere Arbeit:

```
CREATE VIEW Ochsengast AS
SELECT x.Name, x.Vorname, x.Strasse, x.Gebtag, y.Frequenz
FROM Besucher x, Gast y
WHERE x.Name = y.Bname AND x.Vorname = y.Bvorname
AND y.Rname = 'Ochsen';
```

→ «Eine neue Sicht auf bestehende Daten»

? Ist eine View gleichwertig zu einer «richtigen» Tabelle?

Views – Sichten

- Die Benutzer können nun Abfragen tätigen wie z.B.:

```
SELECT *
FROM Ochsengast
WHERE Vorname = 'Hans' AND Strasse = 'Bachweg';
```

? Wie sieht die entsprechende Abfrage ohne Views aus?

Views – Sichten

```
SELECT x.Name, x.Vorname, x.Strasse, x.Gebtag,
y.Frequenz
FROM Besucher x, Gast y
WHERE x.Name = y.Bname AND x.Vorname = y.Bvorname
AND y.Rname = 'Ochsen'
AND x.Vorname = 'Hans' AND x.Strasse = 'Bachweg';
```

Views – Sichten

- Views können auch geschachtelt werden:

```
CREATE VIEW MNamen AS
SELECT Name FROM Ochsengast WHERE Name LIKE 'M%';
```

DDL Revisited

- Wir haben nun genug gelernt, um einige weitere Spielarten von CREATE TABLE zu betrachten:

```
CREATE TABLE Besucher2 (LIKE Besucher)
```

- Erzeugt neue Tabelle Besucher2 mit den gleichen Attributen wie Besucher (also vereinigungskompatibel).
- Die beiden Tabellen sind vollständig entkoppelt, d.h., Änderungen in einer Tabelle wirken sich nicht auf die jeweilig andere aus.
- Per Default werden Constraints nicht übernommen.

DDL Revisited

- Erzeugen einer Tabelle mit Daten einer Query:

  "CREATE TABLE" <tableName> **"AS"** (<*query*>)

- Speichert die Resultate der Query in einer neuen Tabelle.

- Übernimmt keine Constraints.

- Beispiel:

```
CREATE TABLE Besucher3 AS (
   SELECT *
   FROM Besucher
   WHERE name LIKE 'M%');
```

DML Reloaded

- Einfügen ganzer Resultattabellen:

  "INSERT INTO" <tableName> **"("** <query> **")"**

- Fügt die Resultate der Abfrage in die Tabelle ein.

- Das Abfrageresultat muss die geeignete Anzahl Attribute und passende Domänen für die Attribute haben.

- Beispiel:

```
INSERT INTO Besucher2 (
  SELECT *
  FROM Besucher
  WHERE name LIKE '%a%');
```

DML Reloaded

- Modifizieren von Tupeln:

```
"UPDATE" <tableName>
"SET" <attributeName> "=" <attributeValue>
{"," <attributeName> "=" <attributeValue>}
["WHERE" <searchCondition>]
```

- Modifiziert die Tupel, welche durch die Suchbedingung selektiert werden
- <attributeValue> kann auch wieder eine query sein.

- Beispiel:

```
UPDATE KundeCH
SET Umsatz = (SELECT MAX(Umsatz) FROM KundeUSA)
WHERE Name = 'Meier';
```

DML Reloaded

- Gegeben sei folgende Tabelle:

Müller	Heinrich	Kirchweg	1945-03-01
Meier	Anna	Bachweg	1950-05-05
Schmid	Joseph	Bachweg	1960-10-03
Meier	Hanspeter	Dorfstrasse	1895-03-25

? Was könnte das Problem mit folgender Abfrage sein:

```
UPDATE Besucher
SET Vorname = 'Jim'
WHERE Name = 'Meier';
```

DML Reloaded

- Der Primärschlüssel ist {Name, Vorname}
- Wenn nun mehrere Einträge «ansprechen», wird der Primärschlüssel-Constraint verletzt:

  ERROR: duplicate key violates unique constraint «PK_Besucher»
- Analog kann es auch zu Problemen mit Fremdschlüssel-Constraints kommen (siehe frühere Bemerkungen zu CASCADE).

Views Reloaded

- Nach der Betrachtung der DML-Operationen kehren wir nochmals zu den Views zurück.
- Wir haben gesehen, dass beim Lesen (=Abfragen formulieren) eine View sich wie eine «echte» Tabelle verhält.
- Was passiert, wenn wir schreiben wollen?

```
UPDATE Ochsengast
SET Frequenz = 5
WHERE Name = 'Meier' AND Vorname = 'Hans'
```

Views Reloaded

- Könnte man grundsätzlich eindeutig umschreiben:

```
UPDATE Gast
SET    Frequenz = 5
WHERE  Bname = 'Meier' AND Bvorname = 'Hans' AND
       Rname = 'Ochsen'
```

- Wie sieht es aus mit:

```
DELETE FROM Ochsengast
WHERE Name = 'Meier' AND Vorname = 'Hans'
```

Geht das?

Views Reloaded

- Wir müssen uns fragen: Was ist gemeint?
- Soll der «Meier» aus Besucher und Gast verschwinden, mit allen seinen Informationen zu Restaurantbesuchen, oder ist etwas gemeint wie:

```
DELETE FROM Gast
WHERE Bname = 'Meier' AND Bvorname = 'Hans'
AND Rname = 'Ochsen'
```

- Es gibt auch Views, bei denen es keine Updates der zugrundeliegenden Tabellen gibt, die den Updatewunsch wiederspiegeln.

Views Reloaded

- Ob ein Update auf einer View überhaupt theoretisch möglich ist oder nicht, kann nicht per Algorithmus bestimmt werden.
- Es gibt ein paar Regeln, welche Views updatebar sein sollten (SQL92: wenn keine Joins, UNIONs, oder Aggregatfunktionen enthalten).
- Aber z.B. in PostgreSQL: «Currently, views are read only: the system will not allow an insert, update, or delete on a view»
- Alleine mit Views kommt man also nie durch (so interessant das für die Abstraktion wäre).

Und zum Schluss noch dies…

- Basis: LTP-Datenbank:
  - Lieferanten (L)
  - Teile (T)
  - Projekte (P)
  - Lieferungen (LTP)

Format	Constraints	Beschreibung
L (LNr, LName, Status, Stadt)	{LNr} Primary Key {LName} Unique	Lieferant
T (TNr, TName, Farbe, Gewicht, Stadt)	{TNr} Primary Key {TName} Unique	Teil
P (PNr, PName, Stadt)	{PNr} Primary Key {PName} Unique	Projekt
LTP (LNr, TNr, PNr, Menge)	{LNr} Foreign Key auf L {TNr} Foreign Key auf T {PNr} Foreign Key auf P {LNr, TNr, PNr} Unique	„Welcher Lieferant liefert welche Teile für welche Projekte in welcher Menge“. Eine Lieferung entspricht einer Zeile in LTP

Und zum Schluss noch dies…

- «Finde die Teilenummern aller Teile, welche an alle Projekte in Winterthur geliefert werden»
- Umformulieren in äquivalente Aussage («doppelte Verneinung»):
- «Gesucht sind Teile(nummern), für die gilt, dass kein Projekt in Winterthur existiert an das dieses Teil nicht geliefert wird»

Und zum Schluss noch dies…

- Gesucht sind Teilenummern, für die gilt:
  - Es existiert KEIN Projekt in Winterthur, für welches gilt:
  - Es existiert KEIN Eintrag in LTP mit dieser Teilenummer und dieser Projektnummer

```
SELECT TNr
FROM T
WHERE NOT EXISTS (
   SELECT 1
   FROM   P
   WHERE  P.Stadt = 'Winterthur' AND
   NOT EXISTS (
      SELECT 1
      FROM   LTP
      WHERE  LTP.PNr = P.PNr AND LTP.TNr = T.TNr
   )
);
```

Und zum Schluss noch dies…

- «Finde die Projektnummern aller Projekte, welche alle Teile geliefert bekommen, die Sulzer (irgendwohin) liefert»

- Projektnummern, für die gilt:
  - Es existiert KEIN Teil, für welches gilt:
  - Sulzer liefert dieses Teil UND
  - Es existiert KEIN Eintrag, dass dieses Teil IRGENDJEMAND an dieses Projekt liefert

Und zum Schluss noch dies…

```
SELECT P.PNr
FROM P
WHERE NOT EXISTS (
  SELECT 1
  FROM   LTP x, L
  WHERE  x.LNr = L.LNr AND L.LName = 'Sulzer' AND
  NOT EXISTS (
    SELECT 1
    FROM   LTP y
    WHERE  x.TNr = y.TNr AND y.PNr = P.PNr
  )
);
```

Und zum Schluss noch dies…

- Basis: Bier-Datenbank
- «Für welche Besucher gibt es ein Restaurant, das alle ihre Lieblingsbiere im Sortiment hat?»
- Umformen (gedanklich) in folgende Prädikate:

B(x):	«x ist Besucher»
R(x):	«x ist Restaurant»
L(x,y):	«y ist ein Lieblingsbier von x»
S(x,y):	«x hat y im Sortiment»

Wir suchen also x, für die gilt: $B(x) \wedge \exists y(R(y) \wedge \forall z(L(x,z) \rightarrow S(y,z)))$

Und zum Schluss noch dies…

- «Für welche Besucher gibt es ein Restaurant, das alle ihre Lieblingsbiere im Sortiment hat?»
- $B(x) \wedge \exists y(R(y) \wedge \forall z(L(x,z) \rightarrow S(y,z)))$
- muss umgeformt werden (da es in SQL keinen «Allquantor» gibt):
- Es gilt: $P \rightarrow Q$ ist gleichbedeutend mit $\neg(P \wedge \neg Q)$ (siehe Aussagenlogik)
- Wir suchen also: $B(x) \wedge \exists y(R(y) \wedge \neg\exists z(L(x,z) \wedge \neg S(y,z)))$

Und zum Schluss noch dies…

- «Für welche Besucher gibt es ein Restaurant, das alle ihre Lieblingsbiere im Sortiment hat?»
- Diesen Ausdruck kann man nun schrittweise in SQL umwandeln
- $B(x) \wedge \exists y(R(y) \wedge \neg\exists z(L(x,z) \wedge \neg S(y,z)))$

```
SELECT x.Name, x.Vorname FROM Besucher x
WHERE ∃y(R(y) ∧ ¬∃z(L(x,z) ∧ ¬S(y,z)))
```

Und zum Schluss noch dies…

- «Für welche Besucher gibt es ein Restaurant, das alle ihre Lieblingsbiere im Sortiment hat?»

```
SELECT x.Name, x.Vorname FROM Besucher x
WHERE ∃y(R(y) ∧ ¬∃z(L(x,z) ∧ ¬S(y,z)))
```

```
SELECT x.Name, x.Vorname FROM Besucher x
WHERE EXISTS (SELECT 1 FROM Restaurant y
  WHERE ¬∃z(L(x,z) ∧ ¬S(y,z)))
```

Und zum Schluss noch dies…

- «Für welche Besucher gibt es ein Restaurant, das alle ihre Lieblingsbiere im Sortiment hat?»

```
SELECT x.Name, x.Vorname FROM Besucher x
WHERE EXISTS (SELECT 1 FROM Restaurant y
  WHERE ¬∃z(L(x,z) ∧ ¬S(y,z)))
```

```
SELECT x.Name, x.Vorname FROM Besucher x
WHERE EXISTS (SELECT 1 FROM Restaurant y
  WHERE NOT EXISTS (SELECT 1 FROM Lieblingsbier z
    WHERE z.BName = x.Name AND z.BVorname = x.Vorname
    AND ¬S(y,z)))
```

Und zum Schluss noch dies…

- «Für welche Besucher gibt es ein Restaurant, das alle ihre Lieblingsbiere im Sortiment hat?»

```
SELECT x.Name, x.Vorname FROM Besucher x
WHERE EXISTS (SELECT 1 FROM Restaurant y
  WHERE NOT EXISTS (SELECT 1 FROM Lieblingsbier z
    WHERE z.BName = x.Name AND z.BVorname = x.Vorname
    AND NOT EXISTS (SELECT 1 FROM Sortiment t
      WHERE y.Name = t.RName
      AND z.BSorte = t.BSorte)))
```

Die Vielfalt von SQL

- Basierend auf der «LTP»-DB aus Chris Date: "An Introduction to Database Systems" soll abschliessend noch die Vielfalt von SQL illustriert werden.

- Geben Sie so viele verschiedene SQL-Formulierungen an wie möglich, für folgende Abfrage:

  «Gesucht sind die Namen aller Lieferanten, die Teil 'T2' (=TNr) liefern»

- Sie sollten mindestens vier verschiedene Varianten finden…
  - Zwei Varianten mit JOIN
  - Eine Variante mit EXISTS
  - Eine Variante mit ANY/IN

Die Vielfalt von SQL

? Mit Joins?

Die Vielfalt von SQL

Form 1:

```
SELECT DISTINCT L.LName
FROM L NATURAL JOIN LTP
WHERE LTP.TNr = 'T2';
```

Der Join erfolgt auf "LNr"

Wie kann der Join mit alternativen Formen geschrieben werden?

Die Vielfalt von SQL

Form 2:

```
SELECT DISTINCT L.LName
FROM L JOIN LTP ON L.LNr = LTP.LNr
WHERE LTP.TNr = 'T2';
```

Mit Join-Bedingung in ON-Klausel

Die Vielfalt von SQL

Form 3:

```
SELECT DISTINCT L.LName
FROM L INNER JOIN LTP ON L.LNr = LTP.LNr
WHERE LTP.TNr = 'T2';
```

Dasselbe wie Form 2, das "INNER" ist optional (als Gegensatz zu "OUTER")

Die Vielfalt von SQL

Form 4:

```
SELECT DISTINCT L.LName
FROM L INNER JOIN LTP ON L.LNr = LTP.LNr AND
LTP.TNr = 'T2';
```

Fragwürdig. Die Bedingung "LTP.TNr = 'T2'" ist eigentlich keine Join-Bedingung. Das "INNER" kann natürlich weggelassen werden ("Form 4b")

Die Vielfalt von SQL

Form 5:

```
SELECT DISTINCT L.LName
FROM L CROSS JOIN LTP
WHERE L.LNr = LTP.LNr AND LTP.TNr = 'T2';
```

Ebenso fragwürdig. Das "Cross Join" suggeriert, dass keine Join-Bedingung folgt, diese steckt aber in der WHERE-Klausel

Die Vielfalt von SQL

Form 6:

```
SELECT DISTINCT L.LName
FROM L JOIN LTP USING (LNr)
WHERE LTP.TNr = 'T2';
```

"Spielart" von Form 2. Die USING-Klausel entspricht "L.LNr = LTP.LNr" (eliminiert aber eine Spalte in der gejointen Tabelle!)

Kann natürlich auch mit "INNER" geschrieben werden (Form 6b)

Zumindest in MySQL müssen die Join-Attribute geklammert werden

Die Vielfalt von SQL

Form 7:

```
SELECT DISTINCT L.LName
FROM L, LTP
WHERE L.LNr = LTP.LNr AND LTP.TNr = 'T2';
```

"Klassische" Schreibform, sehr populär. Aber: Join- und andere Bedingungen sind gemischt.

Die Vielfalt von SQL

? Kann die Duplikatelimination alternativ umschrieben werden?

Die Vielfalt von SQL

Form 8:

```
SELECT L.LName
FROM L, LTP
WHERE L.LNr = LTP.LNr AND LTP.TNr = 'T2'
GROUP BY L.LName;
```

Fragwürdig. Das Gruppieren wird eigentlich nur für die Duplikatelimination «missbraucht», statt für weitere Berechnungen.

Kann natürlich mit allen bisherigen JOIN-Formen kombiniert werden (Formen 8b, 8c, 8d, 8e, 8f, 8g, 8h, 8i)

Die Vielfalt von SQL

? Können wir das Problem der Duplikatelimination ganz vermeinden?

Die Vielfalt von SQL

Form 9:

```
SELECT L.LName
FROM L
WHERE EXISTS(
    SELECT 1
    FROM LTP
    WHERE LTP.LNr = L.LNr AND LTP.TNr = 'T2');
```

Wir starten mit der Tabelle L. Dies ist eine Relation!

Statt einem Join greifen wir zur Existenzabfrage.

Die Vielfalt von SQL

Form 10:

```
SELECT L.LName
FROM L
WHERE 0 < (
    SELECT COUNT(*)
    FROM LTP
    WHERE LTP.LNr = L.LNr AND LTP.TNr = 'T2');
```

Fragwürdig. Wir haben das EXISTS durch eine Zählbedingung ersetzt.

EXISTS entspricht einem Zähler von grösser 0.

Die Vielfalt von SQL

Form 11:

```
SELECT L.LName
FROM L
WHERE L.Nr IN (
    SELECT LTP.LNr
    FROM LTP
    WHERE LTP.TNr = 'T2')
```

Alternative Sichtweise: Mengenorientiert. Die Lieferungsnummer muss in der Menge der zulässigen (d.h. 'T2' betreffenden) Lieferungen vorkommen.

Die Vielfalt von SQL

Form 12:

```
SELECT L.LName
FROM L
WHERE 'T2' IN (
    SELECT LTP.TNr
    FROM LTP
    WHERE LTP.LNr = L.LNr)
```

Umgekehrte Sichtweise. Es werden alle Teilenummern der zulässigen Lieferungen betrachtet, wobei die Lieferungen nun gemäss ihrer Lieferantennummer zulässig sind.

Die Vielfalt von SQL

Form 13:

```
SELECT L.LName
FROM L
WHERE L.Nr = ANY (
    SELECT LTP.LNr
    FROM LTP
    WHERE LTP.TNr = 'T2')
```

Spielart von Form 11. "IN" lässt sich auch umschreiben als "= ANY" (d.h., mindestens ein Tupel der betrachteten Menge ist gleich).

Dies gilt natürlich auch für Form 12 (Form 12b).

Die Vielfalt von SQL

Form 14:

```
SELECT L.LName
FROM L
WHERE L.Nr = SOME (
    SELECT LTP.LNr
    FROM LTP
    WHERE LTP.TNr = 'T2')
```

Spielart von Form 13. "SOME" ist ein Synonym von "ANY".

Das gleiche gilt auch wieder für Form 12b (Form 12c).

Die Vielfalt von SQL

Einige abschliessende Gedanken zum Beispiel:

- Die Liste ist alles andere als erschöpfend. Nicht alle Formen sind aber sinnvoll. Es soll nicht unser Ziel sein, möglichst absurde SQL-Statements zu schreiben!

- Wir haben triviale Änderungen vermieden
(wie z.B. "a AND b" $\rightarrow$ "b AND a" oder "0 < (...)" $\rightarrow$ "1 <= (...)")

- Trotzdem haben wir 14 Hauptformen und zahlreiche Nebenformen identifiziert! Und vor allem: das Beispiel ist extrem einfach!

- Die Vielfalt bei komplexeren Statements ist potentiell noch viel grösser!

Und weiter...

- Das nächste Mal: Rückblick, Abschluss