Tutorium 7 – Datenbanksysteme

14.11.2022 – Finn Kapitza

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

1. Wiederholung – Erweiterte Anfragen in SQL

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Unterabfragen in SQL in „FROM“

- In FROM können Unterabfragen verwendet werden
- Ergebnis der Unterabfrage ist wie eine Relation
- **WICHTIG**: Ergebnis braucht ein „alias“: (SELECT … FROM …) **as** xy
- Alias darf nicht der Name einer existierenden Relation/Attributes/SQL-Befehls sein (z.B. (select … From …) as count)

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Unterabfragen in SQL in „WHERE“

- Deutlich gängiger sind Unterabfragen in der WHERE Klausel
- Attribute können mit Ergebnis der Unterabfrage verglichen werden
- Nachbildung des Existenzquantors möglich (aber nicht des Allquantors)

  -> Allquantor kann durch Negation in Existenzquantor umgewandelt werden

Unterabfragen in SQL in „WHERE“ – Vergleich mit EINEM Wert

WHERE ausdruck θ (**SELECT** … **FROM** …) ($\theta \in \{<, \leq, \geq, >, =, \neq\}$)

Aufpassen: Unterabfrage darf nur **einen einzelnen** Wert (ein Attribut von einem Tupel) liefern. Das Ergebnis der Unterabfrage wird mit dem Ausdruck verglichen

Bsp.:

```
SELECT tnr
FROM LTP
WHERE lnr = (SELECT lnr FROM L WHERE lname = "MEIER");
```

Unterabfragen in SQL in „WHERE“ – Attribut in Liste enthalten

WHERE ausdruck **[NOT] IN** (**SELECT** … **FROM** …)

Ausdruck gleicht keinem bzw. Mindestens einem Wert der Unterabfrage

Aufpassen: Unterabfrage darf mehrere Tupel aber nur ein Attribut enthalten

Bsp.:

```
SELECT DISTINCT tnr
FROM LTP
WHERE lnr IN (SELECT lnr FROM L WHERE lname LIKE "m%");
```

Unterabfragen in SQL in „WHERE“ – Vergleich mit mindestens einem Wert muss stimmen

WHERE ausdruck θ **ANY | SOME** (**SELECT** … **FROM** …)

Bedingung muss für mindestens einen Wert der Unterabfrage erfüllt sein

< ANY: weniger als das Maximum

> ANY: mehr als das Minimum

= ANY: die selbe Bedeutung wie **IN**

```
SELECT tname
FROM T
WHERE gewicht < ANY (SELECT gewicht FROM T)
```

Unterabfragen in SQL in „WHERE“ – Vergleich mit allen Werten muss stimmen

WHERE ausdruck θ **ALL** (**SELECT** … **FROM** …)

Bedingung muss für alle Werte der Unterabfrage erfüllt sein

< **ALL**: weniger als das Minimum

<= **ALL**: weniger oder gleich dem Minimum -> Minimum bestimmen

>= **ALL**: mehr oder gleich dem Maximum -> Maximum bestimmen

```
SELECT tname, gewicht
FROM T
WHERE gewicht <= ALL (SELECT gewicht FROM T)
```

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Unterabfragen in SQL in „WHERE“ – Existenzquantor simulieren

WHERE [NOT] EXISTS (SELECT … FROM …)

Bedingung ist erfüllt, genau dann wenn Ergebnis der Subquery [leer] nicht leer ist

Meistens in Folge mit Koppelung aus Relationen aus der Außenabfrage

```
SELECT *
FROM P as p1
WHERE EXISTS (SELECT * FROM LTP ltp1 WHERE p1.pnr = ltp1.pnr AND ltp1.tnr = "T1")
```

-> Alle Projektinformationen wo das Teil T1 geliefert wird

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Sortierung der Ergebnisrelation

ORDER BY attribut_1 [, attribute_2, …] **ASC | DESC**

Legt die aufsteigende / absteigende Reihenfolge fest, in der die Ergebniszeilen ausgegeben werden

-> wenn sich zwei Tupel in *attribut_1* Gleichen werden sie nach *attribute_2* sortiert

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Gruppierung in SQL

- Wird benutzt um Informationen über Gruppen in der Relation zu erhalten wie z.B.:
  - **AVG(**x**)** - Mittelwert
  - **COUNT(**x**)** - Anzahl der nicht-NULL Werte
  - **COUNT(DISTINCT** x**)** - Anzahl der verschiedenen nicht-NULL Werte
  - **MAX(**x**)** - Maximum
  - **MIN(**x**)** - Minimum
  - **SUM(**x**)** - Summe aller Werte

- Diese Funktionen nennt man **Aggregatsfunktionen**

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Gruppierung in SQL

- … **GROUP BY** attribute_1 [, attribute_2, …]
  - Teilt die Zeilen einer Tabelle in Gruppen auf
  - WICHTIG: **Alle** Attribute in der SELECT-Liste, die keine Aggregatfunktion sind, müssen in der GROUP BY-Liste enthalten sein

- … **HAVING** ausdruck
  - In der WHERE-Klausel können keine Aggregatsfunktionen verwendet werden
  - Wenn man Bedingungen auf Gruppen definieren will, benutzt die HAVING-Klausel

```
SELECT lnr, lname, SUM(menge) as teile_gesamt
FROM LTP NATURAL JOIN L
GROUP BY lnr, lname
HAVING teile_gesamt >= 1000;
```

Gruppierung in SQL

select A, **sum**(D)
from ... **where** ...
group by A, B

1. Schritt:

from/where

A	B	C	D
1	2	3	4
1	2	4	5
2	3	3	4
3	3	4	5
3	3	6	7

2. Schritt:

Gruppenbildung

A	B	C	D
1	2	3 4	4 5
2	3	3	4
3	3	4 6	5 7

3. Schritt:

Aggregation

A	B	sum(D)	max(C)
1	2	9	4
2	3	4	3
3	3	12	6

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Aufgabe 7.1

- Folgende Anfragen sollen in SQL formuliert werden
- Benutzt wird das Möbel Schema (vgl. Blatt 4)

```
Kunde (kund_nr, kund_name, adresse, ort, plz)
Personal (pers_nr, nachname, vorname, einsatz, vorgesetzt, gehalt)
Verkauf (auftr_nr, bestelldatum, pers_nr, kund_nr)
Inventar (art_nr, art_bez, lagerbest, lagerort, preis)
Auftragsposten (auftr_nr, art_nr, menge)
```

- A) und B) sollen jeweils einmal mit Join-Operationen und einmal nur mit Unterabfragen (kein Join, kein Kreuzprodukt) formuliert werden

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Aufgabe 7.1.a – Joins: Finden Sie Nummern und Bezeichnungen aller Artikel, deren Preis entweder dem Gehalt von „Roswita Hartinger“ oder „Margot Winter“ entspricht

```
SELECT DISTINCT art_nr, art_bez

FROM Inventar JOIN Personal ON preis = gehalt

WHERE (vorname=“Roswita” AND nachname=“Hartinger”) OR
      (vorname=“Margot”  AND nachname=“Winter”);
```

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Aufgabe 7.1.a – Unterabfrage: Finden Sie Nummern und Bezeichnungen aller Artikel, deren Preis entweder dem Gehalt von „Roswita Hartinger“ oder „Margot Winter“ entspricht

```
SELECT DISTINCT art_nr, art_bez
FROM Inventar

WHERE preis IN (
        SELECT gehalt
        FROM Personal
        WHERE (vorname=“Roswita” AND nachname=“Hartinger”) OR
                (vorname=“Margot”  AND nachname=“Winter”)
);
```

Aufgabe 7.1.b – Joins: Geben Sie alle Kundennamen an, die am 24.07.2019 etwas von einem Mitarbeiter mit dem Einsatzort Hamburg gekauft haben

```
SELECT DISTINCT kund_name

FROM Kunde NATURAL JOIN Verkauf NATURAL JOIN Personal

WHERE bestelldatum = “24.07.2019” AND einsatz = “Hamburg”;
```

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Aufgabe 7.1.b – Unterabfrage: Geben Sie alle Kundennamen an, die am 24.07.2019 etwas von einem Mitarbeiter mit dem Einsatzort Hamburg gekauft haben

```
SELECT DISTINCT kund_name
FROM Kunde
WHERE kund_nr IN (
        SELECT kund_nr
        FROM Verkauf
        WHERE bestelldatum = “24.07.2019 AND pers_nr IN (
                SELECT pers_nr
                FROM Personal
                WHERE einsatz = “Hamburg
        )
);
```

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Aufgabe 7.1.c – Erzeugen Sie eine Liste aller Mitarbeiter Vornamen, Nachnamen und Gehalt. Absteigend nach Gehalt. Bei gleichem Gehalt zunächst aufsteigend nach Nachnamen und dann nach Vornamen sortiert.

```
SELECT vorname, nachname, gehalt

FROM Personal

ORDER BY gehalt DESC, nachname ASC, vorname ASC;
```

Aufgabe 7.1.d – Bestimmen Sie die Artikelnummer, Artikelbezeichnungen und Preise des Inventars, die den niedrigsten Preis aufweisen

Option 1:

```
SELECT DISTINCT art_nr, art_bez, preis

FROM Inventar

WHERE preis <= ALL (
        SELECT preis
        FROM Inventar
);
```

Aufgabe 7.1.d – Bestimmen Sie die Artikelnummer, Artikelbezeichnungen und Preise des Inventars, die den niedrigsten Preis aufweisen

Option 2:

```
SELECT DISTINCT art_nr, art_bez, preis

FROM Inventar

WHERE preis = (
        SELECT MIN(preis)
        FROM Inventar
);
```

Aufgabe 7.1.d – Bestimmen Sie die Artikelnummer, Artikelbezeichnungen und Preise des Inventars, die den niedrigsten Preis aufweisen

Option 3:

```
SELECT DISTINCT art_nr, art_bez, preis

FROM Inventar I1

WHERE NOT EXISTS (
        SELECT *
        FROM Inventar I2
        WHERE I1.preis > I2.preis
);
```

Aufgabe 7.1.e – Finden Sie die Artikelnummern, die von mindestens zwei unterschiedlichen Kunden mit Wohnsitz in Stuttgart gekauft wurden

SELECT art_nr

FROM Auftragsposten **NATURAL JOIN** Verkauf **NATURAL JOIN** Kunde

WHERE Kunde.ort = “Stuttgart”

GROUP BY art_nr

HAVING COUNT(**DISTINCT** kund_nr) >= 2

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Aufgabe 7.2 – Anfragen in SQL

- Formulieren Sie folgende Anfragen in der Datenbanksprache SQL

- Bei komplizierten Anfragen kann es hilfreich sein, diese im Tupelkalkül zu formulieren und dann in SQL umzuwandeln

Aufgabe 7.2.a – Finden Sie die Nummern und Namen aller Kunden, die noch nie etwas gekauft haben

$$\{[k.kund\_nr, k.kund\_name]Kunde(k) \wedge \neg\exists v \in Verkauf : v.kund\_nr = k.kund\_nr\}$$

```
SELECT kund_nr, kund_name
FROM Kunde k
WHERE NOT EXISTS (
        SELECT *
        FROM Verkauf v
        WHERE v.kund_nr = k.kund_nr
);
```

Aufgabe 7.2.b – Finden Sie die Nummern und Nachnamen aller Angestellten (Personal), welche allen Kunden mit Wohnsitz in Landshut bereits etwas verkauft haben

$$\{[p.pers\_nr, p.nachname] | Personal(p) \wedge \forall k \in Kunde: k.ort = \acute{}Landshut\acute{} \Rightarrow \exists v \in Verkauf : v.pers\_nr = p.pers\_nr \wedge v.kund\_nr = k.kund\_nr\}$$

=

$$\{[p.pers\_nr, p.nachname] | Personal(p) \wedge \forall k \in Kunde: k.ort \neq \acute{}Landshut\acute{} \vee \exists v \in Verkauf : v.pers\_nr = p.pers\_nr \wedge v.kund\_nr = k.kund\_nr\}$$

=

$$\{[p.pers\_nr, p.nachname] | Personal(p) \wedge \neg\exists k \in Kunde: (k.ort = \acute{}Landshut\acute{} \wedge \neg\exists v \in Verkauf : v.pers\_nr = p.pers\_nr \wedge v.kund\_nr = k.kund\_nr)\}$$

Aufgabe 7.2.b – Finden Sie die Nummern und Nachnamen aller Angestellten (Personal), welche allen Kunden mit Wohnsitz in Landshut bereits etwas verkauft haben

$$\{[p.pers\_nr, p.nachname] | Personal(p) \wedge \neg\exists k \in Kunde: (k.ort = \text{´}Landshut\text{´} \wedge \neg\exists v \in Verkauf: v.pers\_nr = p.pers\_nr \wedge v.kund\_nr = k.kund\_nr)\}$$

```
SELECT pers_nr, nachname FROM Personal P
WHERE NOT EXISTS (
        SELECT * FROM Kunde K
        WHERE K.ort = “Landshut” AND NOT EXISTS (
                SELECT * FROM Verkauf V
                WHERE V.pers_nr = P.pers_nr AND V.kund_nr = K.kund_nr
        )
);
```

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Finn Kapitza – Finn.Kapitza@campus.lmu.de