UNIVERSITÄT DES SAARLANDES

Saarbrücken, 11.06.2015
Information Systems Group

Vorlesung „Informationssysteme“

Vertiefung Kapitel 6: SQL und Verschachtelte Anfragen

Erik Buchmann (buchmann@cs.uni-saarland.de)

Foto: M. Strauch

Aus den Videos wissen Sie...

- ...wie eine SQL-Anfrage aufgebaut ist
  - Grundsätzliche Idee: lesbar wie ein englischer Satz
- ...dass es viele Möglichkeiten gibt, dasselbe in SQL auszudrücken
  - Was ist nun „*gutes*“ SQL?

- Vertiefung heute:
  - Vom Informationsbedürfnis zur SQL-Anfrage
  - Verschachtelte Anfragen auflösen

Foto: M. Strauch

Vom Informationsbedürfnis zur Anfrage

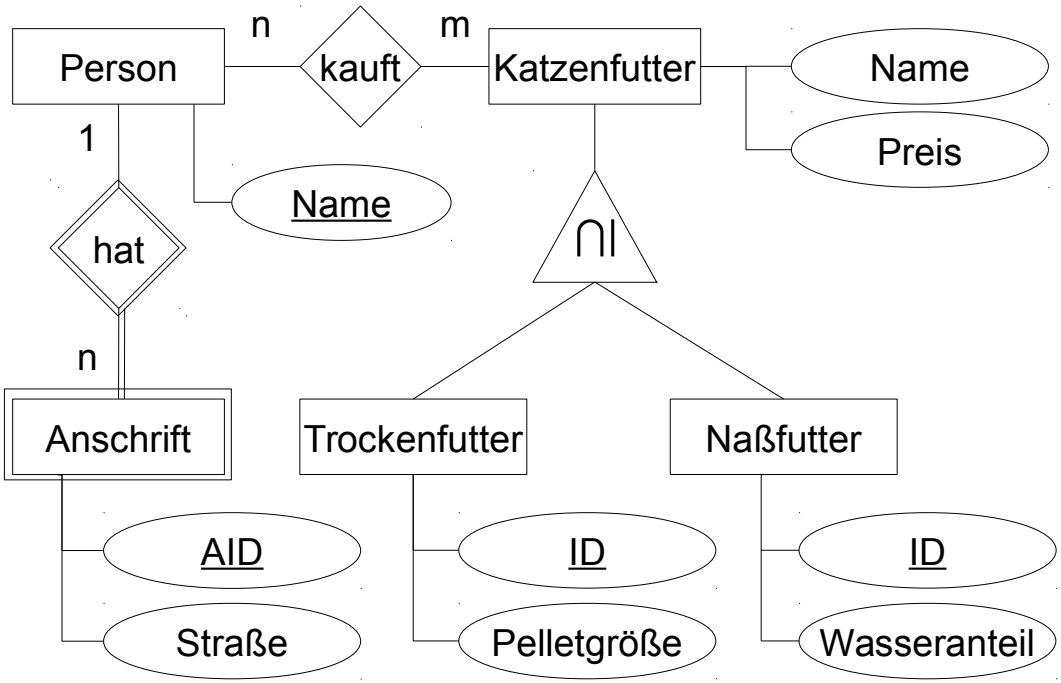
- Faustregel: (*fast identisch zur Relationenalgebra)*
  1) FROM: Verbund über alle Tabellen, die mit der Anfrage zu tun haben
     - Suche kürzesten „Weg“ über die Schlüsselbeziehungen zu allen Relationen, welche die Daten enthalten, die selektiert oder ausgegeben werden müssen
     - Vorzugsweise mit *Rel1* ***JOIN*** *Rel2* ***ON*** *Prädikat*
     - Mehrmals verwendete Relationen mit **AS** umbenennen
  2) ORDER BY/GROUP BY/HAVING: Sortiere und gruppiere falls erforderlich
     - Having erlaubt Selektion von Gruppen
  3) WHERE: Selektiere die relevanten Tupel
     - Alles mit Where-Prädikaten ausfiltern, was nicht in das Anfrageergebnis gehört
  4) SELECT: Projeziere die Attribute, die ausgegeben werden müssen
     - Ebenfalls hier: Berechnungen und Aggregationen durchführen

- Empfehlung: Bei mehr als einer Relation Tabellenpräfix *immer* angeben
  - SELECT Person.Name … statt SELECT Name
- Bei schwierigen Anfragen: verschachtelte Anfragen nutzen

Verschieden komplexe Beispiele

- Beispielanfragen
  - Welche Personen existieren in der DB?
  - Welches Futter ist billiger als 5 EUR?
  - Wo wohnt Jens?
  - Wer hat das Katzenfutter „Saarlands Bestes“ gekauft?
  - An welche Straßen wurde Naßfutter mit >80% Wasseranteil geliefert?

[Person]: { [Name] }
[Anschrift]: { [AID, Name, Straße] }
[Katzenfutter]: { [KID, Name, Preis] }
[Trockenfutter]: { [ID, Pelletgröße, KID] }
[Naßfutter]: { [ID, Wasseranteil, KID] }
[kauft]: {KID, Name}

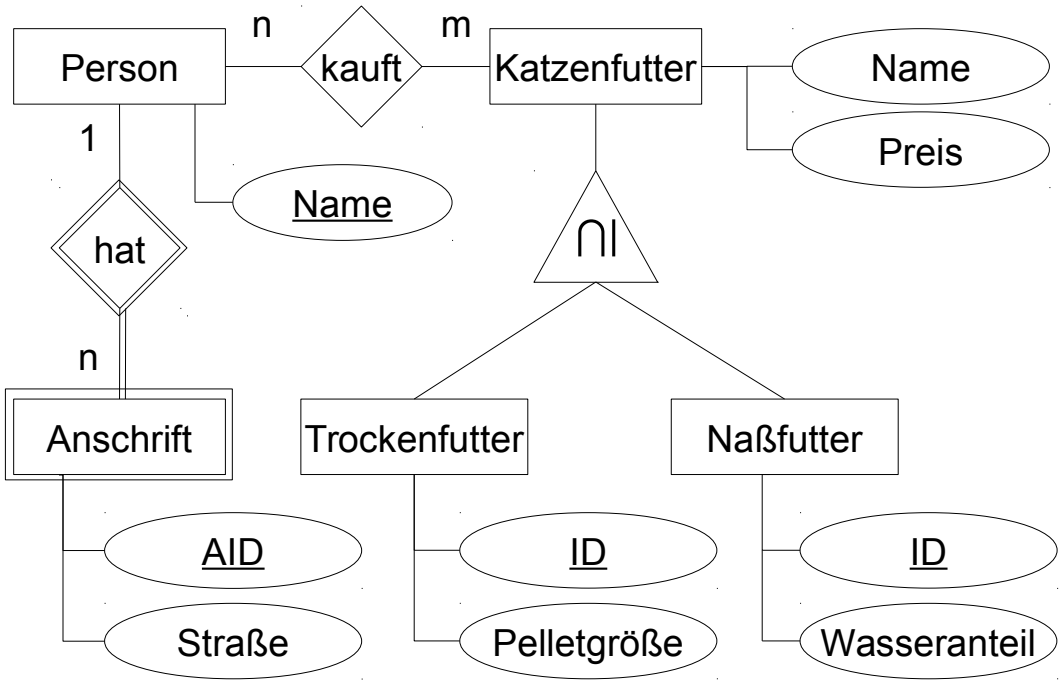


Beispiel 1

- Welche Personen existieren in der DB?
- Anfrage in SQL
  zu einfach ;-)

```
SELECT * FROM Person;
```

[Person]: { [Name] }
[Anschrift]: { [AID, Name, Straße] }
[Katzenfutter]: { [KID, Name, Preis] }
[Trockenfutter]: { [ID, Pelletgröße, KID] }
[Naßfutter]: { [ID, Wasseranteil, KID] }
[kauft]: {KID, Name}



Beispiel 2

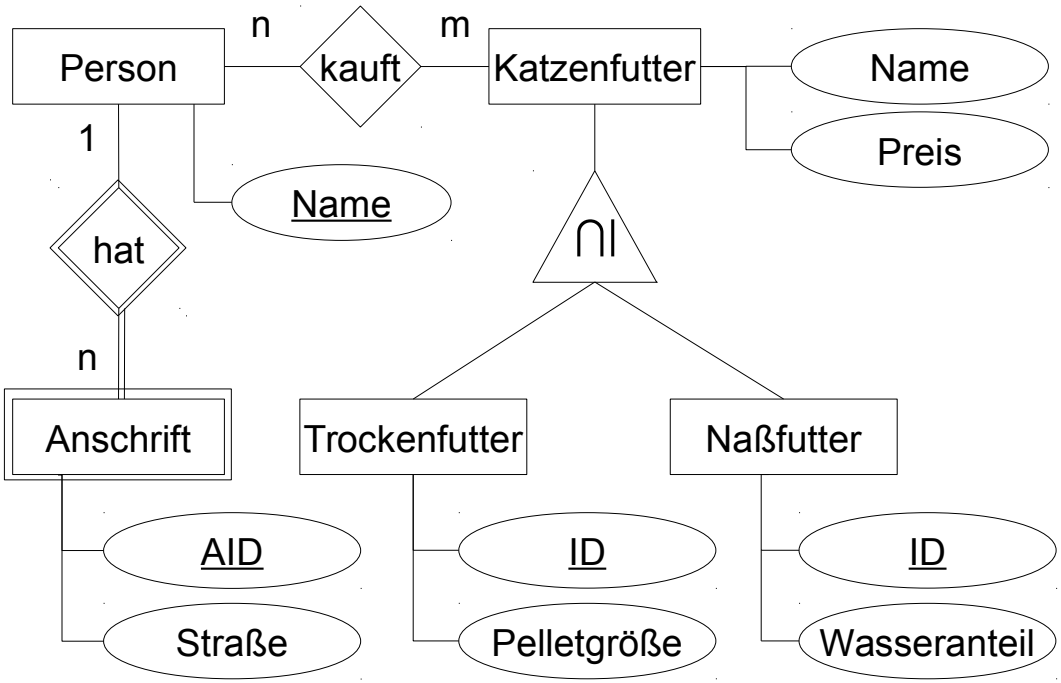
- Welches Futter ist billiger als 5 EUR?

- Anfrage in SQL
  1. beteiligte Relation suchen
  2. nach Preis selektieren
  3. Name ausfiltern

```
SELECT Name
FROM Katzenfutter
WHERE Preis < 5;
```

$\pi_{Name}(\sigma_{Preis<5}(Katzenfutter))$

[Person]: { [Name] }
[Anschrift]: { [AID, Name, Straße] }
[Katzenfutter]: { [KID, Name, Preis] }
[Trockenfutter]: { [ID, Pelletgröße, KID] }
[Naßfutter]: { [ID, Wasseranteil, KID] }
[kauft]: {KID, Name}



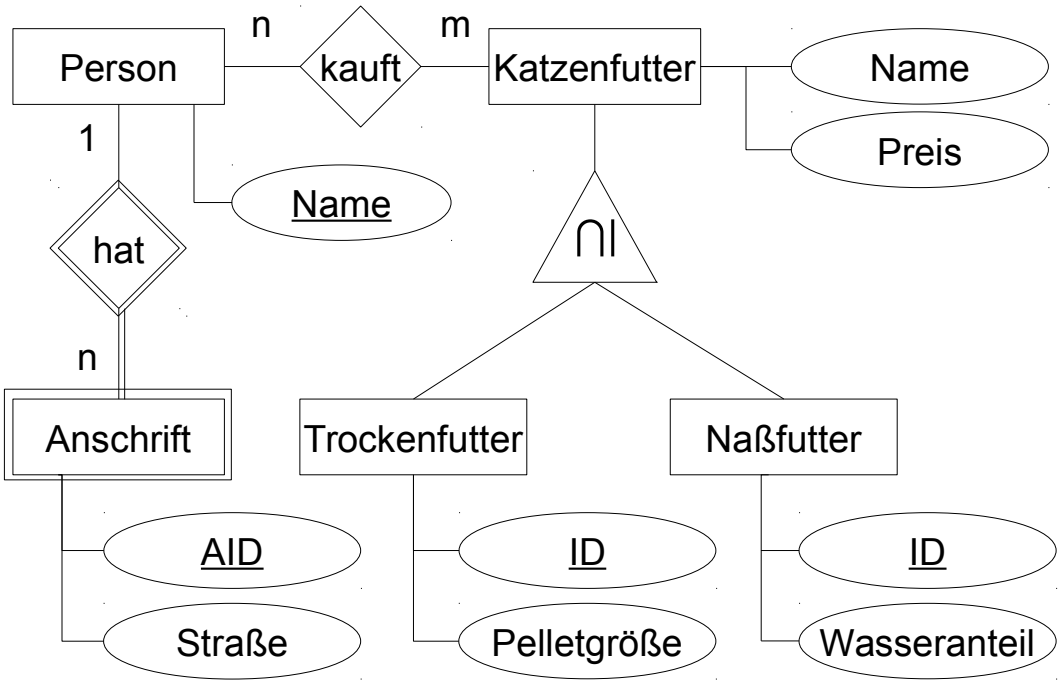
Beispiel 3

- Wo wohnt Jens?
- Anfrage in SQL
  1. „Weg“ von [Person] zu [Anschrift] suchen
     → *Name in [Anschrift] enthalten, d.h. [Person] wird nicht gebraucht*
  2. nach Jens selektieren
  3. Straße filtern

```
SELECT Straße
FROM Anschrift
WHERE Name = 'Jens';
```

$$\pi_{\text{Straße}}(\sigma_{\text{Name=Jens}}(\text{Anschrift}))$$

[Person]: { [Name] }
[Anschrift]: { [AID, Name, Straße] }
[Katzenfutter]: { [KID, Name, Preis] }
[Trockenfutter]: { [ID, Pelletgröße, KID] }
[Naßfutter]: { [ID, Wasseranteil, KID] }
[kauft]: {KID, Name}



Beispiel 4

- Wer hat das Katzenfutter „Saarlands Bestes“ gekauft?

[Person]: { [Name] }
[Anschrift]: { [AID, Name, Straße] }
[Katzenfutter]: { [KID, Name, Preis] }
[Trockenfutter]: { [ID, Pelletgröße, KID] }
[Naßfutter]: { [ID, Wasseranteil, KID] }
[kauft]: {KID, Name}

- Anfrage in SQL
  1. „Weg“ von [Katzenfutter] nach Name in [Person] suchen
  2. nach „Saarlands Bestes“ selektieren
  3. Projektion auf Namen

```
SELECT DISTINCT kauft.Name
FROM kauft JOIN Katzenfutter ON kauft.KID = Katzenfutter.KID
WHERE Katzenfutter.Name = 'Saarlands Bestes';
```

$$\pi_{\text{kauft.Name}}(\sigma_{\text{Katzenfutter.Name=“Saarlands Bestes“}}(\text{kauft} \rhd\lhd_{\text{kauft.KID=Katzenfutter.KID}} \text{Katzenfutter}))$$

Beispiel 5

- An welche Straßen wurde Naßfutter mit >80% Wasseranteil geliefert?

[Person]: { [Name] }
[Anschrift]: { [AID, Name, Straße] }
[Katzenfutter]: { [KID, Name, Preis] }
[Trockenfutter]: { [ID, Pelletgröße, KID] }
[Naßfutter]: { [ID, Wasseranteil, KID] }
[kauft]: {KID, Name}

- Anfrage in SQL
  1. „Weg“ von [Naßfutter] zu [Anschrift] suchen
  2. nach Wasseranteil selektieren
  3. Straße herausprojezieren

```
SELECT DISTINCT Anschrift.Straße
FROM Naßfutter JOIN kauft ON Naßfutter.kid = kauft.kid
               JOIN Anschrift ON kauft.Name = Anschrift.Name
WHERE Naßfutter.Wasseranteil >= 80;
```

$$\pi_{\text{Straße}}(\sigma_{\text{Wasseranteil} > 80}(\text{Naßfutter} \rhd\lhd \text{kauft} \rhd\lhd \text{Anschrift}))$$

Beispiel 5 mit verschiedenen Verbundtypen formuliert

- Natural Join

```
SELECT DISTINCT Anschrift.Straße
FROM Naßfutter NATURAL JOIN kauft NATURAL JOIN Anschrift
WHERE Naßfutter.Wasseranteil >= 80;
```

- Equi-Join

```
SELECT DISTINCT Anschrift.Straße
FROM Naßfutter JOIN kauft ON Naßfutter.kid = kauft.kid
               JOIN Anschrift ON kauft.name = Anschrift.Name
WHERE Naßfutter.Wasseranteil >= 80;
```

- Kreuzprodukt

```
SELECT DISTINCT Anschrift.Straße
FROM Nassfutter, kauft, Anschrift
WHERE   Nassfutter.kid = kauft.kid AND kauft.Name = Anschrift.Name
        AND Nassfutter.Wasseranteil >= 80;
```

Beispiel 5 mit verschachtelter Unteranfrage

- An welche Straßen wurde Naßfutter mit >80% Wasseranteil geliefert?

[Person]: { [Name] }
[Anschrift]: { [AID, Name, Straße] }
[Katzenfutter]: { [KID, Name, Preis] }
[Trockenfutter]: { [ID, Pelletgröße, KID] }
[Naßfutter]: { [ID, Wasseranteil, KID] }
[kauft]: {KID, Name}

1. Anfrage (innen): Wer hat Naßfutter mit >80% Wasseranteil gekauft?
2. Anfrage (außen): Wo wohnt diese Person?

- In SQL:

```
SELECT Anschrift.Straße
FROM Anschrift
WHERE Anschrift.Name IN (
            SELECT DISTINCT Name
            FROM Naßfutter NATURAL JOIN kauft
            WHERE Naßfutter.Wasseranteil >= 80);
```

Beispiel 5 mit korrelliert verschachtelter Unteranfrage

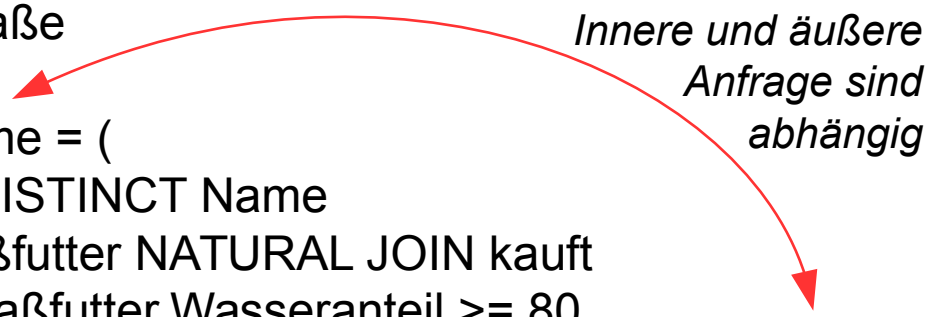
- An welche Straßen wurde Naßfutter mit >80% Wasseranteil geliefert?

[Person]: { [Name] }
[Anschrift]: { [AID, Name, Straße] }
[Katzenfutter]: { [KID, Name, Preis] }
[Trockenfutter]: { [ID, Pelletgröße, KID] }
[Naßfutter]: { [ID, Wasseranteil, KID] }
[kauft]: {KID, Name}

1. Anfrage (außen): Welche Personen gibt's in der Datenbank?
2. Anfrage (innen): Hat die Person (außen) das Naßfutter gekauft?

- In SQL:

```
SELECT Anschrift.Straße
FROM Anschrift
WHERE Anschrift.Name = (
            SELECT DISTINCT Name
            FROM Naßfutter NATURAL JOIN kauft
            WHERE Naßfutter.Wasseranteil >= 80
                        AND kauft.Name = Anschrift.Name);
```



Welche der Anfragen ist am besten?

- Was ist eigentlich „am besten“?
  - Besser zu schreiben?
  - Besser zu verstehen?
  - Besser zu debuggen?
  - Für das DBMS besser zu optimieren?
  - Für das DBMS schneller auszuführen?

- Gedanken dazu
  - Sehr (!) viele Möglichkeiten, dieselbe Anfrage zu formulieren.
  - Solange der Optimierer den schnellsten Anfrageplan finden kann, wähle etwas das gut zu schreiben/verstehen/debuggen ist.

- *Zentrale Frage: Was kann der Optimierer optimieren?*

Verschachtelte Anfragen

Foto: M. Strauch

Motivation

- Suche die Namen aller Personen, die sowohl Naßfutter als auch Trockenfutter gekauft haben

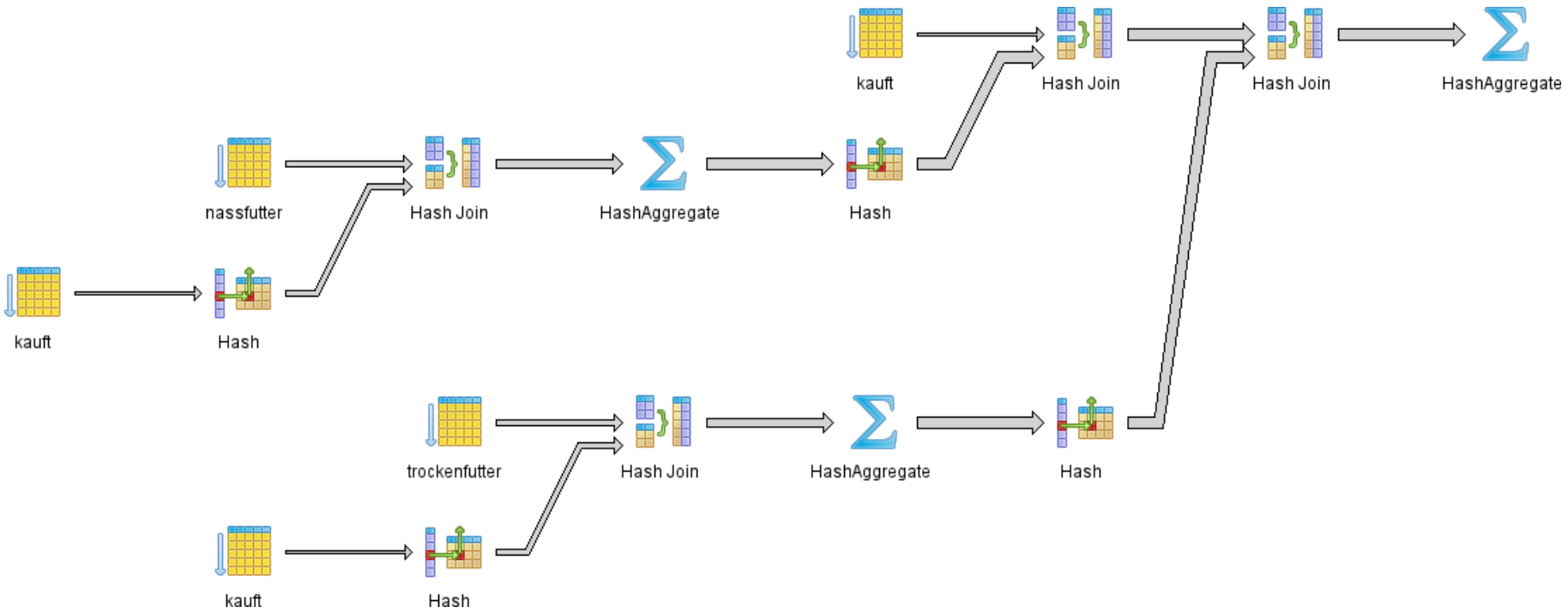
[Person]: { [Name] }
[Anschrift]: { [AID, Name, Straße] }
[Katzenfutter]: { [KID, Name, Preis] }
[Trockenfutter]: { [ID, Pelletgröße, KID] }
[Naßfutter]: { [ID, Wasseranteil, KID] }
[kauft]: {KID, Name}

- SQL-Anfrage mit korreliert verschachtelten Unteranfragen

```
SELECT DISTINCT name FROM kauft
WHERE name IN
        ( SELECT name FROM kauft JOIN trockenfutter
                ON kauft.kid = trockenfutter.kid )
AND name IN
        ( SELECT name FROM kauft JOIN Naßfutter
                ON kauft.kid = Naßfutter.kid );
```

Was macht der Optimierer daraus?

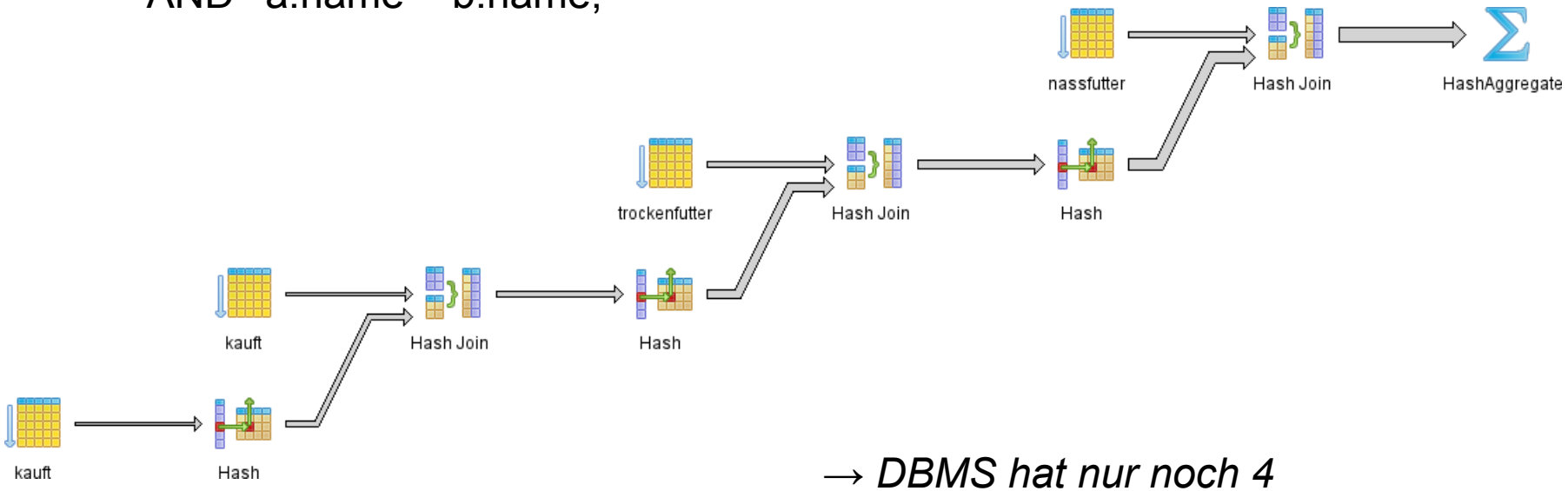
```
SELECT DISTINCT name FROM kauft
WHERE name IN ( SELECT name FROM kauft JOIN trockenfutter
                ON kauft.kid = trockenfutter.kid )
AND name IN ( SELECT name FROM kauft JOIN nassfutter
                ON kauft.kid = nassfutter.kid );
```



Gleiche Anfrage ohne Verschachtelung (1/2)

- Mit Kreuzprodukt und Selbstverbund über kauft

```
SELECT DISTINCT a.name
FROM kauft AS a, kauft AS b, trockenfutter, nassfutter
WHERE   a.kid = trockenfutter.kid
    AND   b.kid = nassfutter.kid
    AND   a.name = b.name;
```

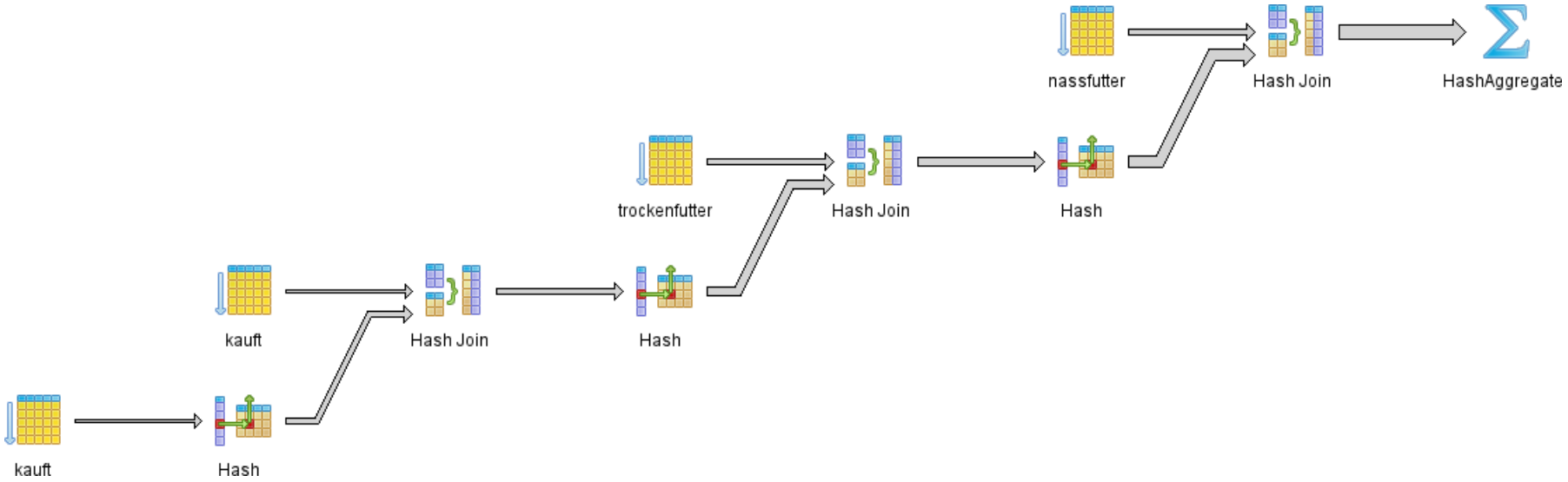


→ *DBMS hat nur noch 4 Relationen durchsucht und 3 Joins gebraucht!*

Gleiche Anfrage ohne Verschachtelung (2/2)

- Mit JOIN statt Kreuzprodukt, wieder Selbstverbund über kauft

```
SELECT DISTINCT a.name
FROM kauft AS a JOIN kauft AS b ON a.name = b.name
                JOIN trockenfutter ON a.kid = trockenfutter.kid
                JOIN nassfutter ON b.kid = nassfutter.kid;
```



Zwischenfazit

- DB-Optimierer kommt mit Verbund-Operationen sehr gut zurecht
- DB-Optimierer hat Probleme mit verschachtelten Unteranfragen
- Komplizierter „aussehende“ Anfragen können schneller sein

- Wie löst man verschachtelte Unteranfragen auf?
  - Gelten die Regeln zur Äquivalenzumformung aus der Relationenalgebra?
  - Gibt es einfache, verständliche Regeln zum „entschachteln“?

Äquivalenzumformung der Relationenalgebra

- Funktioniert ohne Einschränkung:
  - Kommutativität, Assoziativität
    - Alle Verbundoperationen und Kreuzprodukt
  - Kaskaden auflösen
    - Projektionen, Selektionen
  - Vertauschen
    - Projektion + Selektion
- Vertauschen unter Beachtung der Multimengensemantik
  - SELECT vs. SELECT DISTINCT
    - Projektion + Verbund
    - Selektion + Verbund
  - UNION ALL vs. UNION
    - Projektion + Vereinigung
    - Selektion + Vereinigung/Differenz

Verschachtelte Anfragen auflösen

Foto: M. Strauch

Verschachtelte Unteranfragen

- Ort der Verschachtelung
  - Verschachtelte Unteranfrage im SELECT-Teil
  - Verschachtelte Unteranfrage im FROM-Teil
  - Verschachtelte Unteranfrage im WHERE-Teil

- Art der Verschachtelung
  - Innere Anfrage und äußere Anfrage sind nicht korreliert
  - Innere Anfrage und äußere Anfrage sind korreliert

Verschachtelung im SELECT

- Unteranfrage muss einen einzelnen Wert zurückliefern!

```
SELECT [Attribut, …, ]
        (SELECT … FROM … WHERE...)
FROM [Relation,...]
WHERE [Prädikat,...]
```

- Beispiel (nicht korreliert)

```
SELECT anschrift.strasse, (SELECT count(*) FROM kauft)
FROM anschrift WHERE anschrift.name like 'E%';
```

- Beispiel (korreliert)

```
SELECT anschrift.strasse, (
        SELECT count(*) FROM kauft WHERE kauft.name = anschrift.name)
FROM anschrift WHERE anschrift.name like 'E%';
```

Verschachtelung im FROM

- Unteranfrage muss Tupelmenge zurückliefern!
  SELECT [Attribut, …,]
  FROM [Relation,...], [(JOIN)] **(SELECT … FROM … WHERE...)**
  WHERE [Prädikat,...]

- Beispiel (nicht korreliert)

```
SELECT *
FROM katzenfutter
        JOIN
                (SELECT * FROM trockenfutter WHERE pelletgroesse > 1) AS a
        ON katzenfutter.kid = a.kid;
```

- Korrelierte Unteranfragen im FROM-Teil sind nicht möglich
  - (zumindest in keinem DBMS das ich kenne)

Verschachtelung im WHERE: Vergleich

- Unteranfrage als Vergleich
  SELECT [Attribut, …,]
  FROM [Relation,...],
  WHERE [Prädikat,...] **Attribut (<,>, =,...) (SELECT … FROM … WHERE...)**

- Beispiel (nicht korreliert)
  SELECT * FROM katzenfutter
  WHERE preis < (SELECT AVG(preis) FROM katzenfutter);

- Beispiel (korreliert)
  SELECT *
  FROM katzenfutter AS A
  WHERE preis < (
  SELECT AVG(preis)
  FROM katzenfutter AS B
  WHERE A.kid != B.kid);

Verschachtelung im WHERE: Elementtest

- Unteranfrage als Elementest
  SELECT [Attribut, …,]
  FROM [Relation,...],
  WHERE [Prädikat,...,] **Attribut (NOT) IN (SELECT … FROM … WHERE...)**

- Beispiel (nicht korreliert)
  SELECT * FROM katzenfutter
  WHERE kid NOT IN (SELECT kid FROM trockenfutter);

- Beispiel von vorne (korreliert)

```
SELECT DISTINCT name FROM kauft
WHERE name IN
        ( SELECT name FROM kauft JOIN trockenfutter
                ON kauft.kid = trockenfutter.kid )
AND name IN
        ( SELECT name FROM kauft JOIN nassfutter
                ON kauft.kid = nassfutter.kid );
```

Verschachtelung im WHERE: Existenzquantor

- Unteranfrage mit Existenzquantor
  SELECT [Attribut, …,]
  FROM [Relation,...],
  WHERE [Prädikat,...,] **(NOT) EXISTS (SELECT … FROM … WHERE...)**

- Unkorrelierte Unteranfragen mit Existenzquantor sinnfrei aber möglich
  SELECT * FROM katzenfutter
  WHERE EXISTS
  (SELECT * FROM person);

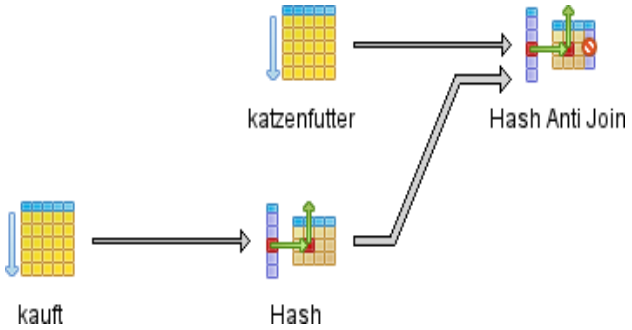
- Beispiel (korreliert)
  SELECT * FROM katzenfutter
  WHERE NOT EXISTS
  (SELECT * FROM kauft WHERE katzenfutter.kid = kauft.kid);

Wann Verschachtelung auflösen? (1/2)

- Aktuelle Optimierer können kanonisch auflösen
  - Korrelierten Unteranfragen mit maximal einer Relation im FROM-Teil
    daraus wird dann stets ein (Anti)-Hash-Join
  - Unkorrelierte Unteranfragen, Unteranfragen mit mehr als einer Relation, wenn sie max. ein Tupel pro Durchlauf der äußeren Anfrage zurückliefern
    daraus wird dann eine Hashtabelle und ein (Anti)-Hash-Join

- Beispiel von der letzten Folie

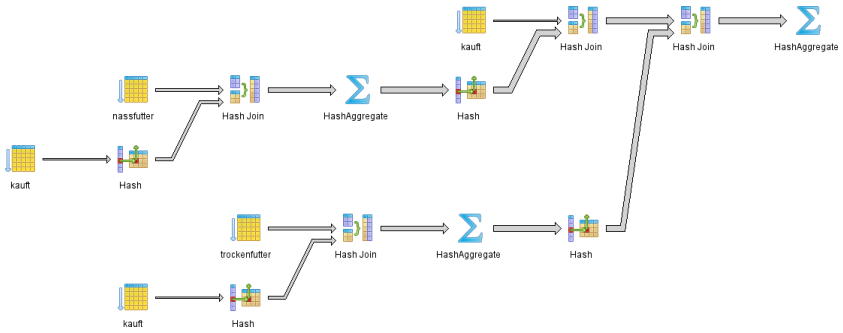
```
SELECT * FROM katzenfutter
WHERE NOT EXISTS
        (SELECT * FROM kauft WHERE katzenfutter.kid = kauft.kid);
```



Wann Verschachtelung auflösen? (2/2)

- Aktuelle Optimierer kommen (meist) nicht zurecht mit
  - allem anderen,
    sofern sie keine speziell für den Fall implementierte Regel kennen
    TPC-H Benchmark „erstaunlich“ gut durch Sonderregeln optimiert

- Siehe Beispiel von Folie 17



  - mehrere parallele verschachtelte Anfragen:
    „Wer hat sowohl Trocken- als auch Naßfutter gekauft?“

- **Allgemein: Ist Performanz kritisch, IMMER auflösen!**

Entschachteln mit Semi-Join (1/2)

- Kanonische Umformungsregel
  - Semi-Join von äußerer mit innerer Anfrage
  - **nur** für verschachtelte Anfragen ohne Aggregate oder Existenz-Quantor!

- Semi-Join bzw. Semi-Verbund:
  - Verbund, bei dem nur Attribute der linken Seite herausprojeziert werden
  - wenn [A]: {[a1,...an]} und [B]: {[b1,...bn]}, dann ist

    $A \ltimes_{Prädikat} B := \Pi_{a1,...,an}(A \bowtie_{Prädikat} B)$

A

a	b	c
1	2	3
2	4	6
3	6	4

B

a	d
1	4
2	3
7	9

SELECT a,b,c FROM A JOIN B ON A.a = B.a

$A \ltimes_{A.a=B.a} B$

a	b	c
1	2	3
2	4	6

Entschachteln mit Semi-Join (2/2)

- Transformation der inneren und äußeren Anfrage in Semi-Join
  1) Relationen der äußeren Anfrage → Linke Seite des Semi-Joins
  2) Relationen der inneren Anfrage → Rechte Seite des Semi-Joins
  3) Beziehung zwischen innerer und äußerer Anf. → Prädikat des Semi-Joins
  4) Beide Where-Prädikate zusammenführen
  5) Falls der Operator der Verschachtelung das erfordert, DISTINCT einfügen

- Allgemeine Vorgehensweise:
  SELECT [Attribut, …,]
  FROM [Relation,...]
  WHERE [Prädikat,...], **Attribut <Operator> (**
  SELECT [Attribut,...] FROM [Relation…] WHERE [Prädikat...])
- Wird zu:
  SELECT (**DISTINCT**) [Attribut, …,],
  FROM [Relation,...] **(JOIN .. ON | Kreuzprodukt) [Relation…]**
  WHERE [Prädikat,...] **AND [Prädikat...] AND Attribut <Operator'>** **Attribut**

Beispiel (nicht korrelierte Verschachtelung)

- Erste verschachtelte Anfrage von Beispiel 5:
  An welche Straßen wurde Naßfutter mit >80% Wasseranteil geliefert?

```
SELECT Anschrift.Strasse
FROM Anschrift
WHERE Anschrift.Name IN (
        SELECT Name
        FROM Nassfutter NATURAL JOIN kauft
        WHERE Nassfutter.Wasseranteil >= 80);
```

äußere Anfrage

innere Anfrage

```
SELECT DISTINCT Anschrift.Strasse
FROM Anschrift JOIN
        (Nassfutter NATURAL JOIN kauft) AS A
        ON Anschrift.Name = A.Name
WHERE A.Wasseranteil >= 80;
```

Beispiel (korrelierte Verschachtelung)

- Zweite verschachtelte Anfrage von Beispiel 5:
  An welche Straßen wurde Naßfutter mit >80% Wasseranteil geliefert?

```
SELECT Anschrift.Strasse
FROM Anschrift
WHERE Anschrift.Name = (
        SELECT DISTINCT Name
        FROM Nassfutter NATURAL JOIN kauft
        WHERE Nassfutter.Wasseranteil >= 80
                AND kauft.Name = Anschrift.Name);
```

äußere Anfrage

innere Anfrage

```
SELECT DISTINCT Anschrift.Strasse
FROM Anschrift JOIN
        (Nassfutter NATURAL JOIN kauft) AS A
        ON Anschrift.Name = A.Name
WHERE A.Wasseranteil >= 80;
```

Was wurde nicht gesagt?

- Neben Umformungsregel von verschachtelten Anfragen ohne Quantoren und Aggregate gibt's viele weitere Regeln

- Seit diesem Jahr:
  Ansatz zum Auflösen beliebiger (!) verschachtelter Anfragen
  - Idee dabei
    - spezifiziere einen *Dependent Join*-Operator in Relationenalgebra, der innere und äußere Anfragen beliebigen Typs verbindet
    - definiere Regeln zur Äquivalenzumformung für den Dependent Join
  - Paper: Thomas Neumann, Alfons Kemper:
    Unnesting Arbitrary Queries. BTW 2015

Zum Abschluss

Foto: M. Strauch

Wie geht es weiter?

- bis Montag, 15.06., 12 Uhr
  - Quiz: SQL Teil 2
- Dienstag, 16.06., GHH 12-14 Uhr: Tutoriumstermin
  - kurze Besprechung von Aufgabenblatt 6
  - nächstes Aufgabenblatt: SQL
    - Hinweis: Installieren Sie schon einmal PostgreSQL auf wenigstens einem Rechner pro Gruppe, und richten Sie einen Nutzer und eine Datenbank ein http://www.postgresql.org/
  - **Klausureinsicht**
    - ab ca. 13 Uhr - 15 Uhr, Raum 3.06 Geb. E1 1
- Donnerstag, 18.06.: Präsenztermin
  - SQL vertieft