



IT Systems Engineering | Universität Potsdam

Datenmanagement mit SQL

Mengenoperationen

Prof. Felix Naumann

Motivation

2

- Hauptstärke der Relationalen Algebra ist die Kombination von Relationen
- Erst mit mehreren Relationen sind viele interessante Anfragen möglich.
- Nennung der beteiligten Relationen in der FROM Klausel

Kreuzprodukt

3

- Alle Paare aus Tupeln der beteiligten Relationen
 - `SELECT *`
`FROM Schauspieler CROSS JOIN Film`
 - `SELECT *`
`FROM Schauspieler, Film`
- Selten verwendet
- Grundbaustein für Joins

Mengenoperationen in SQL

4

- Vereinigung: **UNION**
- Schnittmenge: **INTERSECT**
- Differenz: **EXCEPT**
- Mengenoperationen nur zwischen geklammerten Anfrageergebnissen
- Anfrageergebnisse müssen gleiche Schemata haben
 - Gleiche Attributnamen
 - Gleiche Datentypen
- **Schauspieler(Name, Adresse, Geschlecht, Geburtstag)**
- **Manager(Name, Adresse, ManagerID, Gehalt)**

Schnittmenge: INTERSECT

5

- Entspricht dem logischen „und“
- ```
(SELECT Name, Adresse FROM Schauspieler)
 INTERSECT
(SELECT Name, Adresse FROM Manager) ;
```
- ```
(SELECT Name, Adresse  
      FROM Schauspieler  
      WHERE Geschlecht = 'F')  
      INTERSECT  
(SELECT Name, Adresse  
      FROM Manager  
      WHERE Gehalt > 1.000.000)
```
- Multimengen-Semantik: **INTERSECT ALL**

Vereinigung: UNION

6

- Entspricht dem logischen „oder“
- ```
(SELECT Name, Adresse FROM Schauspieler)
 UNION
(SELECT Name, Adresse FROM Manager) ;
```
- Multimenge: UNION ALL
  - Beliebt, da schnell
  - Verwenden falls
    - ◇ Semantik egal
    - ◇ Multimengensemantik erwünscht
    - ◇ Mengeneigenschaft von Input und Output bekannt

# Differenz: EXCEPT

7

- Auch **MINUS**
- ```
(SELECT Titel, Jahr  
FROM Film)  
EXCEPT  
(SELECT FilmTitel AS Titel, FilmJahr AS Jahr  
FROM spielt_in)
```
- Multimenge: **EXCEPT ALL**

Klammerung

8

```
SELECT *  
FROM  
(  
    (SELECT A FROM R)  
    INTERSECT  
    (SELECT * FROM  
        (SELECT A FROM S)  
        UNION  
        (SELECT A FROM T)  
    )  
)
```


Zusammenfassung der Semantik

9

R1	R2	UNION ALL	UNION	EXCEPT ALL	EXCEPT	INTER- SECT ALL	INTER- SECT
1	1	1	1	1	2	1	1
1	1	1	2	2	5	1	3
1	3	1	3	2		3	4
2	3	1	4	2		4	
2	3	1	5	4			
2	3	2		5			
3	4	2					
4		2					
4		3					
5		3					
		3					
		3					
		3					
		4					
		4					
		4					
		5					

<http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/topi/c/com.ibm.db2.luw.sql.ref.doc/doc/r0000877.html>

Datenmanagement mit SQL

Der Join

Prof. Felix Naumann

Kreuzprodukt und Join

2

- `Film(Titel, Jahr, Länge, inFarbe, StudioName, ProduzentID)`
- `Manager(Name, Adresse, ManagerID, Gehalt)`

- `SELECT Name`
`FROM Film, Manager`
`WHERE Titel = 'Star Wars'`
`AND ProduzentID = ManagerID;`

- Semantik
 - Betrachte jedes Tupelpaar der Relationen **Film** und **Manager**.
 - Wende Bedingung der WHERE Klausel auf jedes Tupelpaar an
 - Falls Bedingung erfüllt, produziere ein Ergebnistupel.
- Kreuzprodukt gefolgt von Selektion: Join

Kreuzprodukt und Join

3

- `Film(Titel, Jahr, Länge, inFarbe, StudioName, ProduzentID)`
- `Manager(Name, Adresse, ManagerID, Gehalt)`

```

■ SELECT Name
  FROM   Film, Manager
 WHERE  Titel = 'Star Wars'
 AND    ProduzentID = ManagerID;

```

Kreuzprodukt

Selektionsbedingung

Joinbedingung

- Semantik
 - Betrachte jedes Tupelpaar der Relationen **Film** und **Manager**.
 - Wende Bedingung der WHERE Klausel auf jedes Tupelpaar an
 - Falls Bedingung erfüllt, produziere ein Ergebnistupel.
- Kreuzprodukt gefolgt von Selektion: Join

Uneindeutige Attributnamen

4

- **Schauspieler**(Name, Adresse, Geschlecht, Geburtstag)
- **Manager**(Name, Adresse, ManagerID, Gehalt)

Bei gleichen Attributnamen aus mehreren beteiligten Relationen:

- Relationenname als Präfix:
 - **SELECT** Schauspieler.Name, Manager.Name
FROM Schauspieler, Manager
WHERE Schauspieler.Adresse = Manager.Adresse;
- Präfix ist auch erlaubt wenn Attributname eindeutig ist.
 - Erleichtert das Lesen von SQL Anfragen

Tupelvariablen

5

- Zur eindeutigen Kennzeichnung von Tupeln beteiligter Relationen
 - „Alias“ einer Relation
 - Insbesondere: Bei der mehrfachen Verwendung einer Relation in einer Anfrage

- Gesucht: Schauspieler, die zusammen leben

```

□ SELECT Star1.Name, Star2.Name
   FROM   Schauspieler Star1, Schauspieler Star2
   WHERE  Star1.Adresse = Star2.Adresse

```

Äquivalent zu
Schauspieler AS Star2

- Auch sinnvoll als abkürzenden Schreibweise

```

□ SELECT S.Name, M.Name
   FROM   Schauspieler S, Manager M
   WHERE  S.Adresse = M.Adresse;

```

- Ohne explizites Angeben einer Tupelvariablen wird der Relationenname als Tupelvariable verwendet.

Tupelvariablen – Selfjoin

6

Name	Adresse	Geschlecht	Geburt
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	M	8/8/88
Brad Pitt	123 Maple St., Hollywood	M	7/7/77

- `SELECT Star1.Name, Star2.Name`
`FROM Schauspieler Star1, Schauspieler Star2`
`WHERE Star1.Adresse = Star2.Adresse;`
- `SELECT Star1.Name, Star2.Name`
`FROM Schauspieler Star1, Schauspieler Star2`
`WHERE Star1.Adresse = Star2.Adresse`
`AND Star1.Name <> Star2.Name;`
- `SELECT Star1.Name, Star2.Name`
`FROM Schauspieler Star1, Schauspieler Star2`
`WHERE Star1.Adresse = Star2.Adresse`
`AND Star1.Name < Star2.Name;`

Star1.Name	Star2.Name
Carrie Fisher	Carrie Fisher
Carrie Fisher	Brad Pitt
Brad Pitt	Carrie Fisher
Brad Pitt	Brad Pitt
Mark Hamill	Mark Hamill

Star1.Name	Star2.Name
Carrie Fisher	Brad Pitt
Brad Pitt	Carrie Fisher

Star1.Name	Star2.Name
Brad Pitt	Carrie Fisher

Interpretation von Anfragen

7

- Gegeben drei Relationen: $R(A)$, $S(A)$ und $T(A)$
- Gesucht: $R \cap (S \cup T)$ ($= (R \cap S) \cup (R \cap T)$)

```

□ SELECT R.A
  FROM   R, S, T
  WHERE  R.A = S.A
  OR     R.A = T.A;
```

- Problemfall: T ist leer, hat also kein Tupel
- Vermeintliches Resultat: $R \cap S$
- Tatsächliches Resultat: leere Menge
 - Ausführung als drei geschachtelte Schleifen

```

SELECT *
FROM
(
  (SELECT A FROM R)
  INTERSECT
  (SELECT * FROM
    (SELECT A FROM S)
    UNION
    (SELECT A FROM T)
  )
)
```


Joins

8

Man kann Joins auch auf andere Weise ausdrücken.

- Geschmacksfrage
- **Film CROSS JOIN spielt_in**
 - Kreuzprodukt
 - Doppelte Attributnamen werden mit Präfix der Relation aufgelöst
- **Film JOIN spielt_in**
 - ON Titel = FilmTitel AND Jahr = FilmJahr
 - Theta-Join
 - ```
SELECT Titel, Jahr, Länge, inFarbe, StudioName,
 ProduzentID, SchauspielerName
FROM Film JOIN spielt_in ON Titel = FilmTitel
 AND Jahr = FilmJahr;
```
  - Eliminiert redundante Attribute FilmTitel und FilmJahr
- **Schauspieler NATURAL JOIN Manager**
  - Natural Join; Eliminiert redundante Attribute

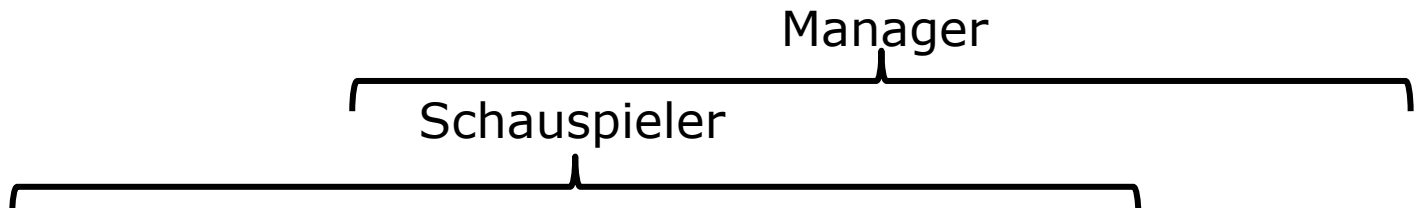
# Outer Joins

9

- Schauspieler(Name, Adresse, Geschlecht, Geburtstag)
- Manager(Name, Adresse, ManagerID, Gehalt)
- Schauspieler, die zugleich Manager sind
  - **SELECT Name, Adresse, Geburtstag, Gehalt**  
**FROM Schauspieler NATURAL INNER JOIN Manager**
- Schauspieler und gegebenenfalls ihre Managerinfo
  - **...FROM Schauspieler NATURAL LEFT OUTER JOIN Manager**
  - Gehalt bleibt gegebenenfalls NULL
- Manager und gegebenenfalls ihre Schauspielerinfo
  - **...FROM Schauspieler NATURAL RIGHT OUTER JOIN Manager**
  - Geburtstag bleibt gegebenenfalls NULL
- Alle Schauspieler und Manager
  - **...FROM Schauspieler NATURAL FULL OUTER JOIN Manager**
  - Geburtstag oder Gehalt bleiben gegebenenfalls leer
  - Unterschied zu UNION: Nur eine Zeile pro Person

# Outer Joins

10



| Geburtstag  | Name              | Adresse  | Gehalt      |
|-------------|-------------------|----------|-------------|
| 1.2.1960    | Mark Hamill       | LA       | <i>NULL</i> |
| 27.5.1969   | Carrie Fischer    | New York | <i>NULL</i> |
| 11.12.1940  | Alec Guinness     | London   | <i>NULL</i> |
| 22.3.1981   | Ben Affleck       | Boston   | 5Mio        |
| 23.8.1973   | Quentin Tarantino | Berlin   | 10Mio       |
| <i>NULL</i> | George Lukas      | San Jose | 100Mio      |
| <i>NULL</i> | Steven Spielberg  | LA       | 500Mio      |

# Outer Joins

11

- Alle Filme mit Schauspielern
  - `Film JOIN spielt_in`  
`ON Titel = FilmTitel AND Jahr = FilmJahr;`
  
- Alle Filme
  - `Film FULL OUTER JOIN spielt_in`  
`ON Titel = FilmTitel AND Jahr = FilmJahr;`
  
- Alle Filme, gegebenenfalls mit Schauspieler
  - `Film LEFT OUTER JOIN spielt_in`  
`ON Titel = FilmTitel AND Jahr = FilmJahr;`

Datenmanagement mit SQL

Geschachtelte Anfragen: Skalare Subanfragen

Prof. Felix Naumann

# Motivation

2

Eine Anfrage kann Teil einer anderen Anfrage sein.

- Theoretisch beliebig tiefe Schachtelung

Drei Varianten

1. Subanfrage erzeugt einen einzigen Wert, der in der WHERE-Klausel mit einem anderen Wert verglichen werden kann.
2. Subanfrage erzeugt eine Relation, die auf verschiedene Weise in WHERE-Klausel verwendet werden kann.
3. Subanfrage erzeugt eine Relation, die in der FROM Klausel verwendet werden kann.
  - Wie jede normale Relation

# Skalare Subanfragen

3

Allgemeine Anfragen produzieren Relationen.

- Mit mehreren Attributen
  - Zugriff auf ein bestimmtes Attribut ist möglich
- i.A. mit mehreren Tupeln
- Manchmal (garantiert) nur ein Tupel und Projektion auf nur ein Attribut
  - „Skalare Anfrage“
  - Verwendung wie eine Konstante möglich

**SUM**

17

# Skalare Subanfragen

4

- `Manager(Name, Adresse, ManagerID, Gehalt)`
- `Film(Titel, Jahr, Länge, inFarbe, StudioName, ProduzentID)`
- Gesucht: Produzent von Star Wars
  - `SELECT Name`  
`FROM Film, Manager`  
`WHERE Titel = 'Star Wars' AND Jahr = '1977'`  
`AND ProduzentID = ManagerID;`
- Oder aber
  - `SELECT Name`  
`FROM Manager`  
`WHERE ManagerID =`  
`( SELECT ProduzentID`  
`FROM Film`  
`WHERE Titel = 'Star Wars' AND Jahr = '1977' );`
- DBMS erwartet **genau ein** Tupel als Ergebnis der Teilanfrage
  - Falls kein Tupel: Laufzeitfehler
  - Falls mehr als ein Tupel: Laufzeitfehler



# Skalare Subanfragen – Beispiel

5

*Abteilungen, deren durchschnittliche Bonuszahlungen höher sind als deren durchschnittliches Gehalt.*

```
SELECT a.Name, a.Standort
FROM Abteilung a
WHERE (SELECT AVG(bonus)
 FROM personal p
 WHERE a.AbtID = p.AbtID)
 >
 (SELECT AVG(gehalt)
 FROM personal p
 WHERE a.AbtID = p.AbtID)
```

# Skalare Subanfragen – Beispiel

6

- *Alle Potsdamer Abteilungen mit ihrem Maximalgehalt.*

```

□ SELECT a.AbtID, a.Name,
 (SELECT MAX(Gehalt)
 FROM Personal p
 WHERE a.AbtID = p.AbtID) AS maxGehalt
FROM Abteilung a
WHERE a.Ort = 'Potsdam'

```

- Anmerkung: Auch Abteilungen ohne Mitarbeiter erscheinen im Ergebnis.

- Nicht so in der folgenden Anfrage:

```

□ SELECT a.AbtID, a.Name, MAX(p.Gehalt) AS maxGehalt
FROM Abteilung a, Personal p
WHERE a.Ort = 'Potsdam'
AND a.AbtID = p.AbtID
GROUP BY a.AbtID, a.Name

```

Datenmanagement mit SQL

Geschachtelte Anfragen:  
Bedingungen mit Relationen

Prof. Felix Naumann

# Bedingungen mit Relationen

2

Bestimmte SQL Operatoren auf Relationen erzeugen Boole'sche Werte

- **EXISTS R**

- TRUE, falls R nicht leer

- **x IN R**

- TRUE falls x gleich einem Wert in R ist (R hat nur ein Attribut)
- Verallgemeinerung auf Tupel später
- **x NOT IN R**: TRUE falls x keinem Wert in R gleicht

- **x > ALL R**

- TRUE falls x größer als jeder Wert in R ist (R hat nur ein Attribut)
- Alternativ: **<, >, <=, >=, <>, =**
- **x <> ALL R**: Entspricht **x NOT IN R** bzw. auch **NOT(x in R)**

- **x > ANY R**

- TRUE falls x größer als mindestens ein Wert in R ist (R hat nur ein Attribut)
- Alternativ: **<, >, <=, >=, <>, =**
- **x = ANY R**: Entspricht **x IN R**
- Alternativer Befehl: **SOME**

- Negation mit **NOT(...)** ist immer möglich.

# EXISTS Beispiele

3

## ■ *Die ISBNs aller ausgeliehenen Bücher*

```
□ SELECT ISBN
 FROM BuchExemplar
 WHERE EXISTS
 (SELECT *
 FROM Ausleihe
 WHERE Ausleihe.Inventarnr = BuchExemplar.Inventarnr)
```

# EXISTS Beispiele

4

- *Lehrstuhlbezeichnungen der Professoren, die alle von ihnen gelesenen Vorlesungen auch schon einmal geprüft haben.*
- *bzw. Lehrstuhlbezeichnungen von Professoren, so dass keine von diesem gelesene Vorlesung existiert, die von ihm nicht geprüft wurde.*

```

□ SELECT Lehrstuhlbezeichnung
 FROM Prof
 WHERE NOT EXISTS
 (SELECT *
 FROM Liest
 WHERE Liest.PANr = Prof.PANr
 AND NOT EXISTS (SELECT *
 FROM Prüft
 WHERE Prüft.PANr = Prof.PANr
 AND Prüft.VL_NR = Liest.VL_NR)
)

```

# IN Beispiele

5

- *Eine Auswahl an Büchern*

- `SELECT Titel`  
`FROM Bücher`  
`WHERE ISBN IN`  
`( '3898644006', '1608452204', '0130319953' )`

- *Matrikel der Studenten, die zumindest einen Prüfer gemeinsam mit dem Studenten der Matrikel ,123456' haben*

- `SELECT Matrikel`  
`FROM Prüft`  
`WHERE Prüfer IN ( SELECT Prüfer`  
`FROM Prüft`  
`WHERE Matrikel = '123456' )`

# IN Beispiele

6

- *Nachnamen aller Professoren, die schon einmal eine 1,0 vergeben haben.*

```
□ SELECT Nachname
 FROM Person
 WHERE 1.0 IN (SELECT Note
 FROM Prüft
 WHERE Prüfer = Person.Prüfer)
```

- Achtung: Korrelierte Subanfrage – siehe nächste Lerneinheit



# ALL und ANY Beispiele

7

- *Die schlechteste Note des Studenten mit Matrikel 123456*

```

□ SELECT Note
 FROM Prüft
 WHERE Matrikel = '123456'
 AND Note >= ALL (SELECT Note
 FROM Prüft
 WHERE Matrikel = '123456')

```

- *All Studenten, die mindestens eine Prüfung absolvierten*

```

□ SELECT Name, Matrikel
 FROM Student
 WHERE Matrikel = ANY (SELECT Matrikel
 FROM Prüft)

```

# Bedingungen mit Tupeln

8

## Verallgemeinerung von **IN**, **ALL** und **ANY** auf Tupel

### ■ **t IN R**

- TRUE falls t ein Tupel in R ist (mehr als ein Attribut möglich)
- Setzt gleiche Schemata voraus
- Setzt gleiche Reihenfolge der Attribute voraus

### ■ **t > ALL R**

- TRUE falls t größer als jedes Tupel in R ist
- Vergleiche in Standardreihenfolge der Attribute

### ■ **t <> ANY R**

- TRUE falls R mindestens ein Tupel hat, das ungleich t ist

# Bedingungen mit Tupeln

9

## ■ *Namen von Produzenten von Filmen mit Harrison Ford*

```

□ SELECT Name
 FROM Manager
 WHERE ManagerID IN
 (SELECT ProduzentID
 FROM Film
 WHERE (Titel, Jahr) IN
 (SELECT FilmTitel AS Titel, FilmJahr AS Jahr
 FROM spielt_in
 WHERE SchauspielerName = 'Harrison Ford'
)
);

```

## ■ Analyse am besten von innen nach außen

## ■ Alternative Formulierung

```

□ SELECT Name
 FROM Manager, Film, spielt_in
 WHERE ManagerID = ProduzentID
 AND Titel = FilmTitel
 AND Jahr = FilmJahr
 AND SchauspielerName = 'Harrison Ford';

```

# Subanfragen in FROM-Klausel

10

Bisher: Nur Subanfragen in WHERE-Klausel

- Anstelle einfacher Relation steht eine geklammerte Subanfrage
- Es muss ein Alias vergeben werden.

□ **SELECT M.Name**

```
FROM Manager M, (SELECT ProduzentID AS ID
 FROM Film, spielt_in
 WHERE Titel = FilmTitel
 AND Jahr = FilmJahr
 AND Schauspieler = 'Harrison Ford')
 Produzent
WHERE M.ManagerID = Produzent.ID;
```

Datenmanagement mit SQL

Korrelierte Subanfragen

Prof. Felix Naumann

# Korrelierte Subanfragen

2

Bisher: Subanfragen einmalig ausführen und das Ergebnis weiterverwenden

- Korrelierte Subanfragen werden mehrfach ausgeführt, einmal pro Bindung der korrelierten Variable der äußeren Anfrage
- *Alle mehrfachen Filme mit Ausnahme der jeweils jüngsten Ausgabe*

```

□ SELECT Titel, Jahr
 FROM Film Alt
 WHERE Jahr < ANY
 (SELECT Jahr
 FROM Film
 WHERE Titel = Alt.Titel);

```

Scope: Attributnamen gehören zunächst zur Tupelvariablen der aktuellen Anfrage. Sonst: Suche von innen nach außen.

- Ausführung der Subanfrage für jedes Tupel in Filme

# Korrelierte Subanfragen – Beispiel

3

Unkorreliert:

*Name und Gehalt aller Mitarbeiter  
in Potsdam*

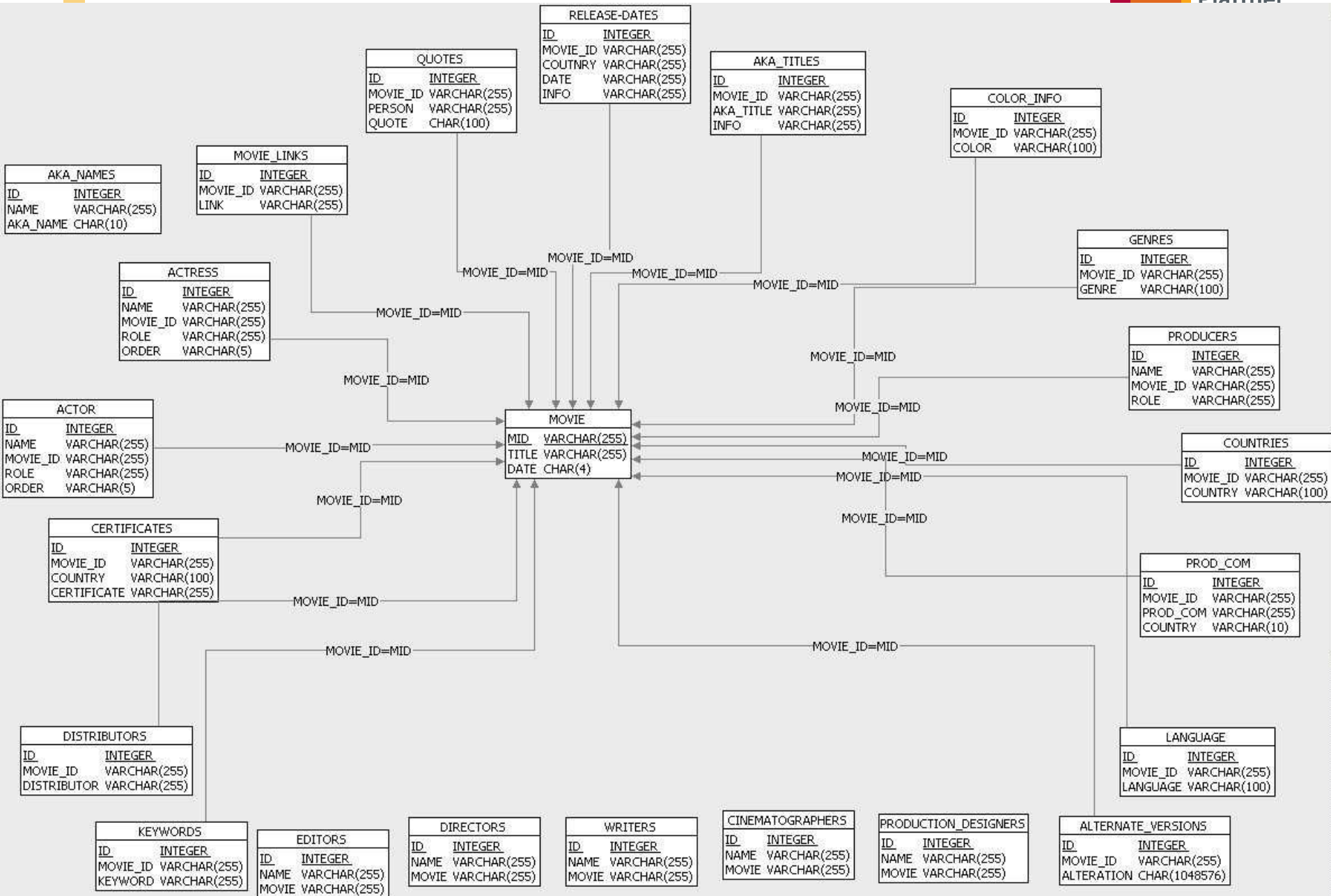
```
SELECT Name, Gehalt
FROM Personal p
WHERE AbtID IN
 (SELECT AbtID
 FROM Abteilung
 WHERE Ort = 'Potsdam')
```

Korreliert:

*Name und Gehalt aller  
Mitarbeiter, deren Gehalt höher  
als 10% des Abteilungsbudgets  
ist.*

```
SELECT Name, Gehalt
FROM Personal p
WHERE Gehalt >
 (SELECT 0.1*Budget
 FROM Abteilung a
 WHERE a.AbtID = p.AbtID)
```

# IMDB Datenbankschema





# Entschachtelung mit WITH

5

- *Durchschnittliche Anzahl an Schauspielern pro Film*

- **WITH**

```
m AS (SELECT count(*) AS ZahlMovies FROM imdb.movie),
actress AS (SELECT count(*) AS ZahlActress FROM imdb.actress),
actor AS (SELECT count(*) AS ZahlActor FROM imdb.actor),
actors AS (SELECT (ZahlActress + ZahlActor) AS GesamtActors
 FROM actress, actor)
SELECT DOUBLE(actors.GesamtActors) / DOUBLE(m.ZahlMovies)
FROM m, actors
```

- Ergebnis: 8,7

# Entschachtelung mit WITH

6

- *Durchschnittliche Anzahl an Filmen pro Schauspieler*

- **WITH**

```
actors AS (SELECT * FROM imdb.actor UNION
 SELECT * FROM imdb.actress),
counts AS (SELECT name, count(movie_id) AS m
 FROM actors GROUP BY name)
SELECT AVG(DOUBLE(m)) FROM counts
```

- Ergebnis: 4,2

Datenmanagement mit SQL

Duplikateliminierung

Prof. Felix Naumann

# Duplikateliminierung

2

Relationale DBMS verwenden i.d.R. Multimengensemantik, nicht Mengensemantik.

- Duplikate entstehen durch

- Einfügen von Duplikaten in Basisrelation
- Veränderung von Tupeln in Basisrelation
- Projektion in Anfragen
- Durch Subanfragen (**UNION**)
- Vermehrung von Duplikaten durch Kreuzprodukt

- Duplikateliminierung

- **SELECT DISTINCT Attributnamen**
- Kosten sind hoch: Sortierung oder hashing

- *Alle Filme, in denen mindestens ein Schauspieler mitspielt*

- **SELECT DISTINCT Titel, Jahr**  
**FROM spielt\_in**

# Duplikateliminierung

3

```
■ SELECT ManagerID, Name
 FROM Manager
 WHERE ManagerID IN
 (SELECT ProduzentID
 FROM Film
 WHERE (Titel, Jahr) IN
 (SELECT FilmTitel,
 FilmJahr
 FROM spielt_in
 WHERE SchauspielerName =
 'Harrison Ford'
)
);
```

```
■ SELECT ManagerID, Name
 FROM Manager, Film, spielt_in
 WHERE ManagerID = ProduzentID
 AND Titel = FilmTitel
 AND Jahr = FilmJahr
 AND SchauspielerName =
 'Harrison Ford';
```

```
■ SELECT DISTINCT ManagerID, Name
 FROM Manager, Film, spielt_in
 WHERE ManagerID = ProduzentID
 AND Titel = FilmTitel
 AND Jahr = FilmJahr
 AND SchauspielerName =
 'Harrison Ford';
```

## ■ Mengenoperationen in SQL entfernen Duplikate

- **UNION, INTERSECT, EXCEPT**

- wandeln Multimengen in Mengen um und verwenden Mengensemantik

- Solche Duplikateliminierung verhindern durch **ALL**

  - ◇ **(SELECT Titel, Jahr, FROM Film)**

    - UNION ALL**

    - (SELECT FilmTitel AS Titel, FilmJahr AS Jahr FROM spielt\_in);**

  - ◇ Film mit drei Schauspielern erscheint also 4 Mal im Ergebnis

- **R INTERSECT ALL S**

- **R EXCEPT ALL S**

Datenmanagement mit SQL

Gruppierung und Aggregation

Prof. Felix Naumann

# Aggregation

2

- Standardaggregationsoperatoren
  - **SUM, AVG, MIN, MAX, COUNT**
  - Angewendet auf einzelne Attribute in der FROM-Klausel
- Typische weitere Aggregationsoperatoren
  - **VAR, STDDEV**
- **COUNT (\*)** zählt Anzahl der Tupel
  - in der Relation, die durch die FROM und WHERE Klauseln definiert wird.
- Kombination mit **DISTINCT**
  - **COUNT(DISTINCT Jahr)**
  - **SUM(DISTINCT Gehalt)**



# Aggregation – Beispiele

3

- `SELECT AVG(Gehalt)  
FROM Manager;`
- `SELECT COUNT(*)  
FROM spielt_in;`
- `SELECT COUNT(Schauspieler)  
FROM spielt_in;`
- `SELECT COUNT(DISTINCT Schauspieler)  
FROM spielt_in  
WHERE Jahr = 1990;`

# Gruppierung, Aggregation und NULL

4

- NULL wird bei Aggregation ignoriert.
  - Trägt also nicht zu SUM, AVG oder COUNT bei.
  - Ist nicht MIN oder MAX
  - Anzahl Tupel: `SELECT COUNT(*) FROM spielt_in;`
  - Anzahl nicht-NULL Werte:  
`SELECT COUNT(Schauspieler) FROM spielt_in;`
- NULL ist ein eigener Gruppierungswert
  - Es gibt also z.B. die NULL-Gruppe
  - `SELECT A, COUNT(B) FROM R GROUP BY A;`  
   ◇ Ergebnis: (NULL, 0)
  - `SELECT A, SUM(B) FROM R GROUP BY A;`  
   ◇ Ergebnis: (NULL, NULL)

| A    | B    |
|------|------|
| NULL | NULL |

# Gruppierung

5

- Gruppierung mittels **GROUP BY** nach der WHERE-Klausel
- **SELECT StudioName, SUM(Länge)**  
**FROM Film**  
**GROUP BY StudioName**
- In SELECT-Klausel zwei „Sorten“ von Attributen
  1. Gruppierungsattribute
  2. Aggregierte Attribute
    - Nicht-aggregierte Werte der SELECT-Klausel müssen in der GROUP BY-Klausel erscheinen.
    - Keine der beiden Sorten muss erscheinen.
- **SELECT StudioName**  
**FROM Film**  
**GROUP BY StudioName**
- **SELECT SUM(Länge)**  
**FROM Film**  
**GROUP BY StudioName**

# Gruppierung

6

Gruppierung bei mehreren Relationen wird am Schluss durchgeführt.

```
■ SELECT Name, SUM(Länge)
 FROM Manager, Film
 WHERE ManagerID = ProduzentID
 GROUP BY Name
```

■ Reihenfolge der Ausführung

- FROM-Klausel
- WHERE-Klausel
- GROUP BY-Klausel
- SELECT-Klausel

# Gruppierung

7

- Einschränkung der Ergebnismenge nach der Gruppierung durch HAVING
  - Einschränkung der Ergebnismenge
    - ◇ `SELECT Name, SUM(Länge)`  
`FROM Manager, Film`  
`WHERE ManagerID = ProduzentID`  
`AND Gehalt > 1000000`  
`GROUP BY Name`
    - ◇ `SELECT Name, SUM(Länge)`  
`FROM Manager, Film`  
`WHERE ManagerID = ProduzentID`  
`GROUP BY Name`  
`HAVING SUM(Länge) > 1000`
    - ◇ `SELECT Name, SUM(Länge)`  
`FROM Manager, Film`  
`WHERE ManagerID = ProduzentID`  
`GROUP BY Name`  
`HAVING SUM(Länge) > 1000`
  - Aggregationen in HAVING Klausel beziehen sich nur auf aktuelle Gruppe.
  - Nur Gruppierungsattribute dürfen unaggregiert in HAVING Klausel erscheinen (wie bei SELECT-Klausel).

# Zusammenfassung SQL

8

## Grundbausteine einer SQL Anfrage (mit empfohlener Lesereihenfolge)

- **4. SELECT**
- **1. FROM**
- **2. WHERE**
- **3. GROUP BY**
- **5. HAVING**
- **6. ORDER BY**
  
- **SELECT ... FROM ...** sind Pflicht.
  - Ausnahme: z.B. **SELECT 7 + 3**
- **HAVING** darf nur in Kombination mit **GROUP BY** erscheinen.

Datenmanagement mit SQL

Sichten

Prof. Felix Naumann

# Virtuelle Relationen

2

- Relationen aus **CREATE TABLE** Ausdrücken existieren tatsächlich (materialisiert, physisch) in der Datenbank.
  - Persistenz
  - Updates sind möglich
  
- Die Daten aus Sichten (*views*) existieren nur virtuell.
  - Sichten entsprechen Anfragen, denen man einen Namen gibt. Sie wirken wie physische Relationen.
  - Updates sind nur manchmal möglich.



# Sichten in SQL

3

- **CREATE VIEW** *Name AS Anfrage*
- **CREATE VIEW** **ParamountFilme AS**  
    **SELECT** Titel, Jahr  
    **FROM** Film  
    **WHERE** StudioName = 'Paramount';
  - Auch mehr als eine Relation möglich!
- Bedeutung einer Anfrage an die Sicht
  1. Ausführung der Anfrage aus der Sichtdefinition
  2. Die ursprüngliche Anfrage verwendet dann das Ergebnis als Relation.
- Daten der Sicht ändern sich mit der Änderung der zugrundeliegenden Relationen.
- Entfernen der Sicht: **DROP VIEW** **ParamountFilme**
  - Basisdaten bleiben unverändert.

# Anfragen an Sichten

4

- **CREATE VIEW ParamountFilme AS**  
    **SELECT Titel, Jahr**  
    **FROM Film**  
    **WHERE StudioName = 'Paramount';**
- **SELECT Titel**  
    **FROM ParamountFilme**  
    **WHERE Jahr = 1979;**
- Umwandlung der ursprünglichen Anfrage in eine Anfrage an Basisrelationen
  - **SELECT Titel**  
    **FROM Film**  
    **WHERE StudioName = 'Paramount' AND Jahr = 1979;**
  - Übersetzung durch DBMS
- Anfrage zugleich an Sichten und Basisrelationen möglich
  - **SELECT DISTINCT Schauspieler**  
    **FROM ParamountFilme, spielt\_in**  
    **WHERE Titel = FilmTitel AND Jahr = FilmJahr;**

# Anfragen an Sichten

5

- `Film(Titel, Jahr, Länge, inFarbe, StudioName, ProduzentID)`
- `Manager(Name, Adresse, ManagerID, Gehalt)`
- `CREATE VIEW FilmeProduzenten AS`  
    `SELECT Titel, Name`  
    `FROM Film, Manager`  
    `WHERE ProduzentID = ManagerID;`
- Anfrage
  - `SELECT Name`  
    `FROM FilmeProduzenten`  
    `WHERE Titel = 'Gone with the Wind'`
- Bedeutung
  - `SELECT Name`  
    `FROM Film, Manager`  
    `WHERE ProduzentID = ManagerID`  
    `AND Titel = 'Gone with the Wind';`

# Attributumbenennung mittels Sichten

6

- Nebenbei: Umbenennung von Attributen

- `CREATE VIEW FilmeProduzenten(FilmTitel,  
Produzentennamen) AS  
SELECT Titel, Name  
FROM Film, Manager  
WHERE ProduzentID = ManagerID;`

- Oder auch: Sicht einfach nur zur Umbenennung

- `CREATE VIEW Movie(title, year, length, inColor,  
studio, producerID) AS  
SELECT *  
FROM Film;`

# Diskussion

7

## Vorteile

- Vereinfachung von Anfragen
- Strukturierung der Datenbank
- Logische Datenunabhängigkeit
  - Sichten stabil bei Änderungen der Datenbankstruktur
  - Sichtdefinitionen müssen gegebenenfalls angepasst werden
  - Stabilität nicht bei jeder Änderung
- Beschränkung von Zugriffen (Datenschutz)
- Optimierung durch materialisierte Sichten

## Probleme

- Automatische Anfragetransformation schwierig
- Änderungen auf Sichten
- Updatepropagierung für materialisierte Sichten

# Materialisierte Sichten

8

- Viele Anfragen an eine Datenbank wiederholen sich häufig
  - Business Reports, Bilanzen, Umsätze
  - Bestellungsplanung, Produktionsplanung
  - Kennzahlenberechnung
- Viele Anfragen sind Variationen mit gemeinsamem Kern
- Idee: Einmaliges Berechnen der Anfrage als Sicht
  - Automatische, transparente Verwendung in folgenden Anfragen
  - Materialisierte Sicht (*materialized view*, MV)

# MV – Themen und Probleme

9

- Wahl der Sichten zur Materialisierung
  - MVs kosten: Platz und Aktualisierungsaufwand
  - Wahl der optimalen MVs hängt von Workload ab
  - Auswahl der „optimalen“ Menge von MVs
- Automatische Aktualisierung
  - Aktualisierung bei Änderungen der Basisrelationen
  - U.U. schwierig: Aggregate, Joins, Outer-Joins, ...
- Automatische Verwendung von MV
  - Umschreiben der Anfrage notwendig
  - Schwierigkeit hängt von Komplexität der Anfrage / Views ab
  - Algorithmen zur transparenten und kostenoptimalen Verwendung der materialisierten Sichten

# Zusammenfassung

10

- Die Anfragesprache SQL
- Der SFW Block
- Subanfragen
  - In FROM und WHERE
  - EXISTS, IN, ALL, ANY
- Mengenoperationen
  - UNION, INTERSECT, EXCEPT
- Joins und Outerjoins
- Nullwerte
- Mengen vs. Multimengen
- Gruppierung und Aggregation
  - MIN, MAX, COUNT
  - GROUP BY, HAVING
- Datenbankveränderungen
  - INSERT, UPDATE, DELETE
- Schemata und Datentypen
  - CREATE TABLE
  - ALTER TABLE
- Indizes
- Sichten
  - Updates und Anfragen
  - Materialisierte Sichten