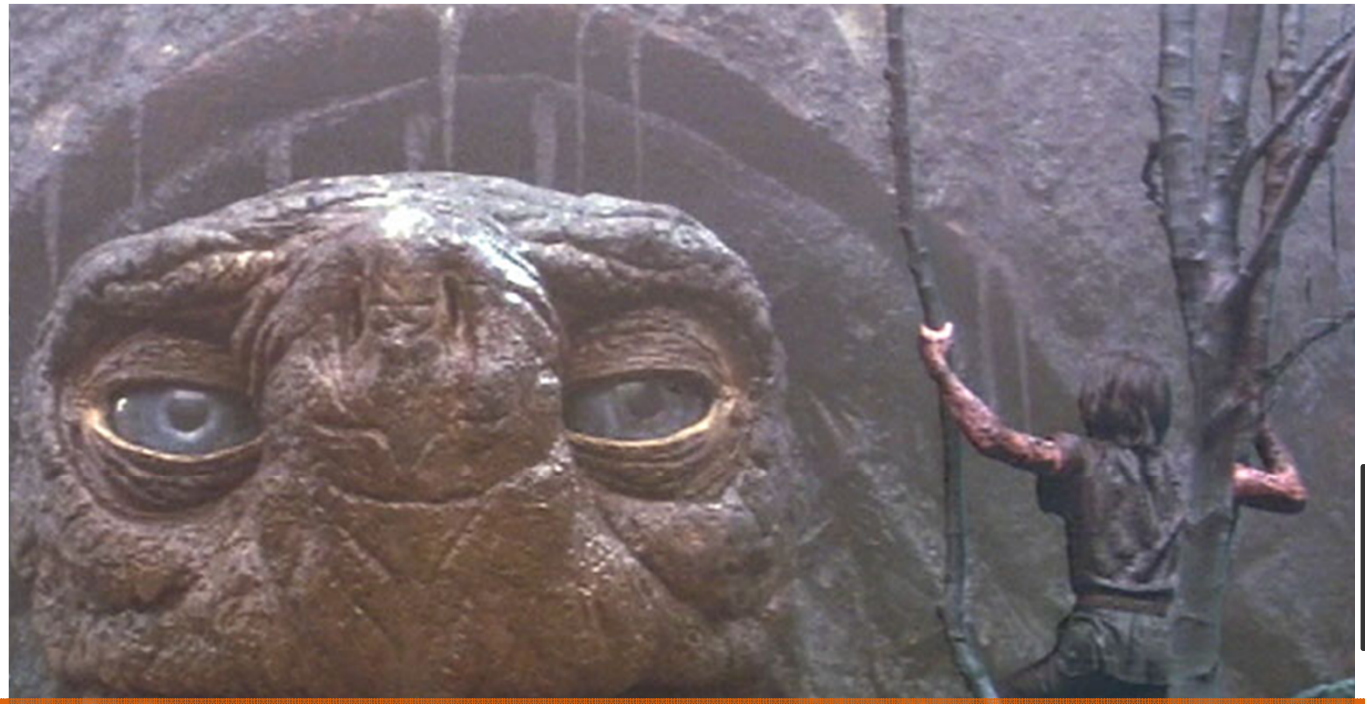


Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Folien basierend auf Thorsten Papenbrock

Übung Datenbanksysteme I
SQL

Leon Bornemann
F-2.06, Campus II
Hasso Plattner Institut

Diverses

- In eigener Sache: Stellenausschreibung Werkstudent
  - Mit Programmiererfahrung in Java oder Scala
  - Zur Unterstützung von Forschungstätigkeiten
  - Link: https://hpi.de//naumann/people/open-positions.html (unten)
  - Bei Interesse Mail an mich mit kurzem Lebenslauf/Notenübersicht
- Nächstes Übungsblatt
  - Vollständig Lesen!
  - Status Installation und Datenimport?
- Verschiebung des letzten und vorletzten Übungstermins
  - Vermutlich beides um 2 Wochen nach hinten

Merkblatt SQL

Grundbausteine:

SELECT <Attributliste> (= Schema der Ergebnisrelation; * für alle Attribute)
FROM <Relationenliste> (Relationen aus denen die Tupel stammen)
WHERE <Bedingungen>; (Bedingungen an die Daten; Verknüpfung über Schlüsselwort **AND**)

SELECT:
- Entspricht Projektion π der relationalen Algebra
- Umbenennung: SELECT Titel AS Filmtitel
- Arithmetik: SELECT Länge * 3.1415 AS LängeMalPi
- Konstanten: SELECT ‚Herr' AS Titel
- Duplikateliminierung δ: SELECT DISTINCT Titel

FROM:
- Entspricht Kreuzprodukt $\times$ der relationalen Algebra (falls mehrere Relationen gewählt)

WHERE:
- Entspricht Selektion σ der relationalen Algebra
- WHERE-Teil der Anfrage ist optional
- Operatoren: =, <>, >, <, <=, >=, LIKE, NOT, ANY, ALL, EXISTS, IN, ...
- Kann Kreuzprodukt des FROM-Teils zum Join machen:
  WHERE Person.ID = Mensch.ID (Natürlicher Join $\bowtie$)
  WHERE Person.Name = Mensch.Vorname (Theta Join $\bowtie_\theta$)

Komplexe Anfragen:

SELECT <Attributliste>
FROM <Relationenliste>
WHERE <Bedingungen>
GROUP BY <Gruppierungsattribute>
HAVING <Bedingungen auf Gruppierungsattribute>
ORDER BY <Attributliste>;

GROUP BY ... HAVING:
- Entspricht Gruppierung γ der relationalen Algebra
- Aggregationsoperatoren für SELECT-Statement: AVG(<Attribut>)
  COUNT(<Attribut>)
  SUM(<Attribut>)
- HAVING entspricht einer Selektion nach der Gruppierung

ORDER BY:
- Entspricht Sortierung τ der relationalen Algebra
- Sortiert das Ergebnis der Anfrage entsprechend der Attributliste
- Aufsteigend: ORDER BY Vorname, Nachname ASC
- Absteigend: ORDER BY Vorname, Nachname DESC

Datendefinition: Data Definition Language (DDL)

CREATE TABLE <Tabellenname>**(**<Attributliste mit Datentypen>**);**
- Aufgabe: neue Tabelle erstellen
- Datentypen mit Länge n bzw. m: CHAR(n), VARCHAR(n), BIT(n), DECIMAL(n,m)
- Datentypen mit impliziter Länge: INT, FLOAT
- Datentypen für Objekte: CLOB, BLOB
- Datentypen für Zeiten: TIME, DATE, TIMESTAMP
- Bsp.:
```
CREATE TABLE Schauspieler (
    Name CHAR(30),
    Adresse VARCHAR(255),
    Geschlecht CHAR(1),
    Geburtstag DATE );
```
- Nebenbedingungen:
  - Primärschlüssel: PRIMARY KEY
  - Eindeutigkeit: UNIQUE
  - Default-Werte: DEFAULT <Defaultwert>
  - Nicht-Null: NOT NULL
  - Fremdschlüssel: FOREIGN KEY (<Attributliste>) REFERENCES <Tabellenname>(<Attributliste>)
  - Weitere: CHECK, CREATE TRIGGER, CREATE ASSERTION, ...
  Bsp.:
```
CREATE TABLE Schauspieler (
    SchauspielerNummer INT PRIMARY KEY,
    Name CHAR(30) NOT NULL,
    Adresse VARCHAR(255) NOT NULL UNIQUE,
    Geschlecht CHAR(1),
    Geburtstag DATE DEFAULT DATE ‚0000-00-00',
    FOREIGN KEY (Adresse) REFERENCES Haus(Adresse) );
```

DROP TABLE <Tabellenname>;
- Aufgabe: bestehende Tabelle löschen
- Bsp.: DROP TABLE Schauspieler;

ALTER TABLE <Tabellenname> <Aktion>;
- Aufgabe: bestehende Tabelle ändern
- ADD: - Attribut hinzufügen
  - Bsp.: ALTER TABLE Schauspieler ADD Telefon CHAR(16)
- DROP: - Attribut löschen
  - Bsp.: ALTER TABLE Schauspieler DROP Geburtstag

CREATE INDEX <Indexname> **ON** <Tabellenname>**(**<Attributliste des neuen Index>**);**
- Aufgabe: Index erstellen

DROP INDEX <Indexname>;
- Aufgabe: Index löschen

Datenbearbeitung: Data Modelling Language (DML)

INSERT INTO <Tabellenname>**(**<Attributliste>**) VALUES (**<Attributliste>**);**
- Aufgabe: Tupel einfügen
- Bsp.: INSERT INTO Studio(Name, Nummer) VALUES (‚Pixa', 34);
- Ergebnis einer Anfrage für Einfügen nutzen:
  Bsp.:
```
INSERT INTO Studios(Name)
    SELECT DISTINCT StudioName
    FROM  Film
    WHERE StudioName NOT IN
        (SELECT Name
        FROM Studios);
```

DELETE FROM <Tabellenname> **WHERE** <Bedingung>;
- Aufgabe: Tupel löschen
- Bsp.: DELETE FROM Studio WHERE Name='Pixa';

UPDATE <Tabellenname> **SET** <Zuweisung> **WHERE** <Bedingung>;
- Aufgabe: Attributwerte ändern
- Bsp.: UPDATE Studio SET Name='Pixa' WHERE Name='Pi';

Bulk insert: IMPORT, LOAD, ... (→ DBMS spezifisch)

Mengenoperationen

(<Anfrage>) **UNION** (<Anfrage>) (Liefert Vereinigung „$\cup$" der beiden Ergebnismengen)

(<Anfrage>) **EXCEPT** (<Anfrage>) (Liefert Differenz „$-$"der beiden Ergebnismengen)

(<Anfrage>) **INTERSECT** (<Anfrage>) (Liefert Schnittmenge „$\cap$" der beiden Ergebnismengen)

- UNION, EXCEPT und INTERSECT nutzen Mengensemantik (→eliminieren Duplikate)
- UNION ALL, EXCEPT ALL und INTERSECT ALL nutzen Multimengensemantik (→erhalten Duplikate)

Join-Varianten

1.) Kreuzprodukt mit Bedingung:
```
SELECT *
FROM <Join-Relation1>, <Join-Relation2>
WHERE <Join-Attribut1> = <Join-Attribut2>;
```
2.) Schlüsselwort:
```
<Tabellenname> CROSS JOIN <Tabellenname>
<Tabellenname> NATURAL JOIN <Tabellenname>
<Tabellenname> NATURAL INNER JOIN <Tabellenname>
<Tabellenname> NATURAL LEFT OUTER JOIN <Tabellenname>
<Tabellenname> NATURAL RIGHT OUTER JOIN <Tabellenname>
<Tabellenname> NATURAL FULL OUTER JOIN <Tabellenname>
```

Sichten

CREATE VIEW <Sichtname> **AS** <Anfrage>;
- Aufgabe: Erstelle eine Sicht für die gegebene SQL-Anfrage

Übersicht
SQL

SELECT <Attributliste>
FROM <Relationenliste>
WHERE <Bedingungen>
GROUP BY <Gruppierungsattribute>
HAVING <Bedingungen auf Gruppierungsattribute>
ORDER BY <Attributliste>**;**

Weitere Schlüsselwörter:

- **DISTINCT, AS, JOIN**
- **AND, OR**
- **MIN, MAX, AVG, SUM, COUNT**
- **NOT, IN, LIKE, ANY, ALL, EXISTS**
- **UNION, EXCEPT, INTERSECT**
- …

Übersicht:

Relationale Übungsschemata

Modellnummer

Prozessorgeschwindigkeit [MHz]

Product(maker, <u>model</u>, type)

- Beispieltupel: `('Lenovo', 1005, 'pc')`

Festplattengröße [GB]

PC(<u>model</u>, speed, ram, hd, rd, price)

- Beispieltupel: `(1005, 1000, 128, 20, '12xDVD', 1499)`

Geschwindigkeit und Typ des Laufwerks

Laptop(<u>model</u>, speed, ram, hd, screen, price)

- Beispieltupel: `(2008, 650, 64, 10, 12.1, 1249)`

Bildschirm-auflösung

Printer(<u>model</u>, color, type, price)

- Beispieltupel: `(3005, 'y', 'bubble', 200)`

Typ (laser, ink-jet, bubble)

Preis [Euro]

Aufgabe:
Deutsch → SQL

Product(maker, <u>model</u>, type)

- Beispieltupel: `('Lenovo', 1005, 'pc')`

PC(<u>model</u>, speed, ram, hd, rd, price)

- Beispieltupel: `(1005, 1000, 128, 20, '12xDVD', 1499)`

Laptop(<u>model</u>, speed, ram, hd, screen, price)

- Beispieltupel: `(2008, 650, 64, 10, 12.1, 1249)`

Printer(<u>model</u>, color, type, price)

- Beispieltupel: `(3005, true, 'bubble', 200)`

Anfrage: „Welche PC-Modelle haben eine Geschwindigkeit von mindestens 1000 MHz?“

$$\pi_{model}(\sigma_{speed \geq 1000}(PC))$$

Lösung:
Deutsch → SQL

```
SELECT model
FROM PC
WHERE speed >= 1000;
```

Anfrage: „Welche PC-Modelle haben eine Geschwindigkeit von mindestens 1000 MHz?“

$$\pi_{model}(\sigma_{speed \geq 1000}(PC))$$

Aufgabe:
Deutsch → SQL

Product(maker, <u>model</u>, type)

- Beispieltupel: `('Lenovo', 1005, 'pc')`

PC(<u>model</u>, speed, ram, hd, rd, price)

- Beispieltupel: `(1005, 1000, 128, 20, '12xDVD', 1499)`

Laptop(<u>model</u>, speed, ram, hd, screen, price)

- Beispieltupel: `(2008, 650, 64, 10, 12.1, 1249)`

Printer(<u>model</u>, color, type, price)

- Beispieltupel: `(3005, true, 'bubble', 200)`

Anfrage: „Welche Hersteller bauen Laptops mit einer Harddisk von mindestens 10 GB Größe?“

$$\pi_{maker}(\sigma_{hd \geq 10}(Product \bowtie Laptop))$$

Lösung:

Deutsch → SQL

a)
```
SELECT maker
FROM Product, Laptop
WHERE Product.model = Laptop.model
AND hd >= 10;
```

b)
```
SELECT maker
FROM Product NATURAL JOIN Laptop
WHERE hd >= 10;
```

Anfrage: „Welche Hersteller bauen Laptops mit einer Harddisk von mindestens 10 GB Größe?“

$$\pi_{maker}(\sigma_{hd \geq 10}(Product \bowtie Laptop))$$

Aufgabe:
Deutsch → SQL

Product(maker, <u>model</u>, type)

- Beispieltupel: `('Lenovo', 1005, 'pc')`

PC(<u>model</u>, speed, ram, hd, rd, price)

- Beispieltupel: `(1005, 1000, 128, 20, '12xDVD', 1499)`

Laptop(<u>model</u>, speed, ram, hd, screen, price)

- Beispieltupel: `(2008, 650, 64, 10, 12.1, 1249)`

Printer(<u>model</u>, color, type, price)

- Beispieltupel: `(3005, true, 'bubble', 200)`

Anfrage: „Finde die Modellnummern aller Farblaserdrucker."

$$\pi_{model}(\sigma_{color=true \wedge type='laser'}(Printer))$$

Lösung:

Deutsch → SQL

```
SELECT model
FROM Printer
WHERE color=true
AND type='laser';
```

Anfrage: „Finde die Modellnummern aller Farblaserdrucker.“

$$\pi_{model}(\sigma_{color=true \wedge type=`laser`}(Printer))$$

Aufgabe:
Deutsch → SQL

Product(maker, <u>model</u>, type)

- Beispieltupel: `('Lenovo', 1005, 'pc')`

PC(<u>model</u>, speed, ram, hd, rd, price)

- Beispieltupel: `(1005, 1000, 128, 20, '12xDVD', 1499)`

Laptop(<u>model</u>, speed, ram, hd, screen, price)

- Beispieltupel: `(2008, 650, 64, 10, 12.1, 1249)`

Printer(<u>model</u>, color, type, price)

- Beispieltupel: `(3005, true, 'bubble', 200)`

Anfrage: „Finde die Modellnummer und den Preis aller Produkte (jeden Typs), die von Hersteller 'Apple' gebaut werden."

$$\pi_{model,price}(\sigma_{maker='Apple'}(Product \bowtie (\pi_{model,price}(PC) \cup \pi_{model,price}(Laptop) \cup \pi_{model,price}(Printer))))$$

Lösung:

Deutsch → SQL

```
    (SELECT Product.model, price
     FROM Product JOIN PC ON Product.model = PC.model
     WHERE maker = 'Apple')
UNION
    (SELECT Product.model, price
     FROM Product JOIN Laptop ON Product.model = Laptop.model
     WHERE maker = 'Apple')
UNION
    (SELECT Product.model, price
     FROM Product JOIN Printer ON Product.model = Printer.model
     WHERE maker = 'Apple');
```

Anfrage: „Finde die Modellnummer und den Preis aller Produkte (jeden Typs), die von Hersteller ‘Apple‘ gebaut werden.“

$$\pi_{model,price}(\sigma_{maker=\text{‘Apple‘}}(Product \bowtie (\pi_{model,price}(PC) \cup \pi_{model,price}(Laptop) \cup \pi_{model,price}(Printer))))$$

Lösung:

Deutsch → SQL

```
WITH Price AS (
    (SELECT model, price FROM PC) UNION
    (SELECT model, price FROM Laptop) UNION
    (SELECT model, price FROM Printer))
SELECT Price.*
FROM Product NATURAL JOIN Price
WHERE maker = 'Apple';
```

Anfrage: „Finde die Modellnummer und den Preis aller Produkte (jeden Typs), die von Hersteller ‘Apple‘ gebaut werden.“

$$\pi_{model,price}(\sigma_{maker=\text{‘Apple‘}}(Product \bowtie (\pi_{model,price}(PC) \cup \pi_{model,price}(Laptop) \cup \pi_{model,price}(Printer))))$$

Aufgabe:
Deutsch → SQL

Product(maker, <u>model</u>, type)

- Beispieltupel: `('Lenovo', 1005, 'pc')`

PC(<u>model</u>, speed, ram, hd, rd, price)

- Beispieltupel: `(1005, 1000, 128, 20, '12xDVD', 1499)`

Laptop(<u>model</u>, speed, ram, hd, screen, price)

- Beispieltupel: `(2008, 650, 64, 10, 12.1, 1249)`

Printer(<u>model</u>, color, type, price)

- Beispieltupel: `(3005, true, 'bubble', 200)`

Anfrage: „Finde alle Hersteller, die Laptops, aber keine PCs herstellen."

$$\delta(\pi_{maker}(Product \bowtie Laptop)) - \delta(\pi_{maker}(Product \bowtie PC))$$

Lösung:

Deutsch → SQL

```
(SELECT DISTINCT maker
 FROM Product, Laptop
 WHERE Product.model = Laptop.model)
EXCEPT
(SELECT DISTINCT maker
 FROM Product, PC
 WHERE Product.model = PC.model);
```

DISTINCTs sind nicht nötig wegen Mengensemantik von **EXCEPT**! Für Multimengen: **EXCEPT ALL**

Anfrage: „Finde alle Hersteller, die Laptops, aber keine PCs herstellen."

$$\delta(\pi_{maker}(Product \bowtie Laptop)) - \delta(\pi_{maker}(Product \bowtie PC))$$

Aufgabe:
Deutsch → SQL

Product(maker, <u>model</u>, type)

- Beispieltupel: `('Lenovo', 1005, 'pc')`

PC(<u>model</u>, speed, ram, hd, rd, price)

- Beispieltupel: `(1005, 1000, 128, 20, '12xDVD', 1499)`

Laptop(<u>model</u>, speed, ram, hd, screen, price)

- Beispieltupel: `(2008, 650, 64, 10, 12.1, 1249)`

Printer(<u>model</u>, color, type, price)

- Beispieltupel: `(3005, true, 'bubble', 200)`

Anfrage: „Finde alle Harddisk-Größen, die in mehr als zwei PCs vorkommen."

$$\pi_{hd}(\sigma_{Anzahl>2}(\gamma_{hd,count(model) \rightarrow Anzahl}(PC)))$$

Lösung:
Deutsch → SQL

```
SELECT hd
FROM PC
GROUP BY hd
HAVING COUNT(model) > 2;
```

Anfrage: „Finde alle Harddisk-Größen, die in mehr als zwei PCs vorkommen.“

$$\pi_{hd}(\sigma_{Anzahl>2}(\gamma_{hd,count(model) \rightarrow Anzahl}(PC)))$$

Aufgabe:
Deutsch → SQL

Product(maker, <u>model</u>, type)

- Beispieltupel: `('Lenovo', 1005, 'pc')`

PC(<u>model</u>, speed, ram, hd, rd, price)

- Beispieltupel: `(1005, 1000, 128, 20, '12xDVD', 1499)`

Laptop(<u>model</u>, speed, ram, hd, screen, price)

- Beispieltupel: `(2008, 650, 64, 10, 12.1, 1249)`

Printer(<u>model</u>, color, type, price)

- Beispieltupel: `(3005, true, 'bubble', 200)`

Anfrage: „Finde alle Paare von PCs mit gleicher Festplatten- und Hauptspeichergröße. Vermeide dabei doppelte Paare.“

$$(\rho_{PC1}(PC)) \bowtie_{PC1.HD=PC2.HD \wedge PC1.RAM=PC2.RAM \wedge PC1.model<PC2.model} (\rho_{PC2}(PC))$$

Lösung:
Deutsch → SQL

```
SELECT *
FROM PC PC1, PC PC2
WHERE PC1.hd = PC2.hd
AND PC1.ram = PC2.ram
AND PC1.model < PC2.model;
```

Anfrage: „Finde alle Paare von PCs mit gleicher Festplatten- und Hauptspeichergröße. Vermeide dabei doppelte Paare."

$$(\rho_{PC1}(PC)) \bowtie_{PC1.HD=PC2.HD \wedge PC1.RAM=PC2.RAM \wedge PC1.model<PC2.model} (\rho_{PC2}(PC))$$

Aufgabe:
Deutsch → SQL

Product(maker, <u>model</u>, type)

- Beispieltupel: `('Lenovo', 1005, 'pc')`

PC(<u>model</u>, speed, ram, hd, rd, price)

- Beispieltupel: `(1005, 1000, 128, 20, '12xDVD', 1499)`

Laptop(<u>model</u>, speed, ram, hd, screen, price)

- Beispieltupel: `(2008, 650, 64, 10, 12.1, 1249)`

Printer(<u>model</u>, color, type, price)

- Beispieltupel: `(3005, true, 'bubble', 200)`

Anfrage: „Finde alle Hersteller, die mindestens einen Laptop und einen PC mit derselben Prozessorgeschwindigkeit anbieten."

$$\pi_{D.a}(\sigma_{A.b=B.b \wedge A.a=C.g \wedge B.a=D.g \wedge C.a=D.a}(\rho_{A(a,b,c,d,e,f)}(\text{Laptop}) \times \rho_{B(a,b,c,d,e,f)}(\text{PC}) \times \rho_{C(a,g,h)}(\text{Product}) \times \rho_{D(a,g,h)}(\text{Product})))$$

Lösung:

Deutsch → SQL

```
SELECT D.a
FROM
 (SELECT model AS a, speed AS b, ram AS c, hd AS d, screen AS e, price AS f FROM Laptop) A,
 (SELECT model AS a, speed AS b, ram AS c, hd AS d, rd AS e, price AS f FROM PC) B,
 (SELECT maker AS a, model AS g, type AS h FROM Product) C,
 (SELECT maker AS a, model AS g, type AS h FROM Product) D
WHERE A.b = B.b
AND A.a = C.g
AND B.a = D.g
AND C.a = D.a;
```

Anfrage: „Finde alle Hersteller, die mindestens einen Laptop und einen PC mit derselben Prozessorgeschwindigkeit anbieten."

$$\pi_{D.a}(\sigma_{A.b=B.b \wedge A.a=C.g \wedge B.a=D.g \wedge C.a=D.a}(\rho_{A(a,b,c,d,e,f)}(Laptop) \times \rho_{B(a,b,c,d,e,f)}(PC) \times \rho_{C(a,g,h)}(Product) \times \rho_{D(a,g,h)}(Product)))$$

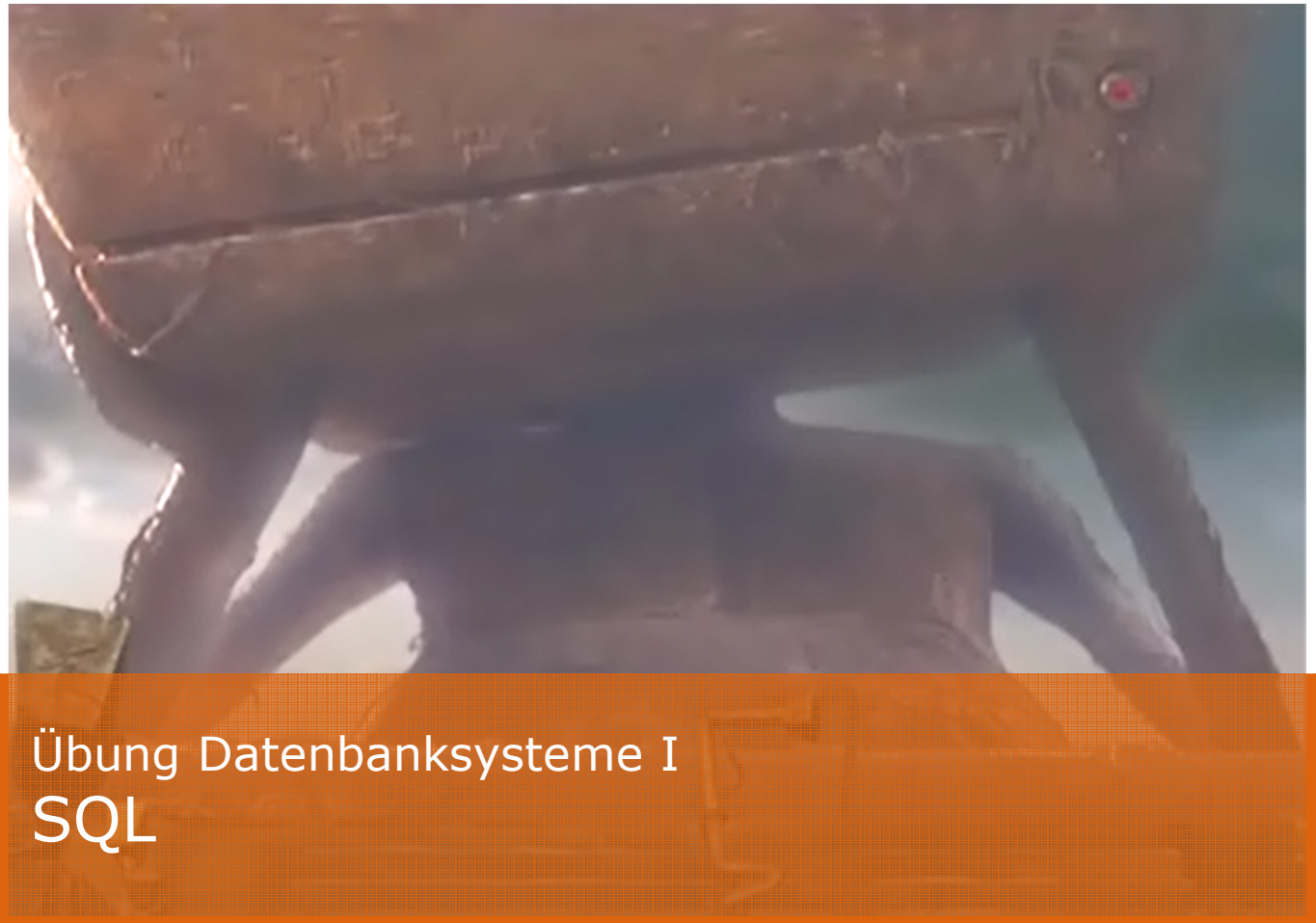
Lösung:

Deutsch → SQL

```
SELECT D.maker
FROM Laptop A, PC B, Product C, Product D
WHERE A.speed = B.speed
AND A.model = C.model
AND B.model = D.model
AND C.maker = D.maker;
```

Anfrage: „Finde alle Hersteller, die mindestens einen Laptop und einen PC mit derselben Prozessorgeschwindigkeit anbieten."

$$\pi_{D.a}(\sigma_{A.b=B.b \wedge A.a=C.g \wedge B.a=D.g \wedge C.a=D.a}(\rho_{A(a,b,c,d,e,f)}(Laptop) \times \rho_{B(a,b,c,d,e,f)}(PC) \times \rho_{C(a,g,h)}(Product) \times \rho_{D(a,g,h)}(Product)))$$



Übung Datenbanksysteme I
SQL

Leon Bornemann
F-2.06, Campus II
Hasso Plattner Institut