Datenbanken

Informatik, ICS und als Wahlfach

7. Erweiterte Abfragen mit SQL: Unterabfragen und JOINs

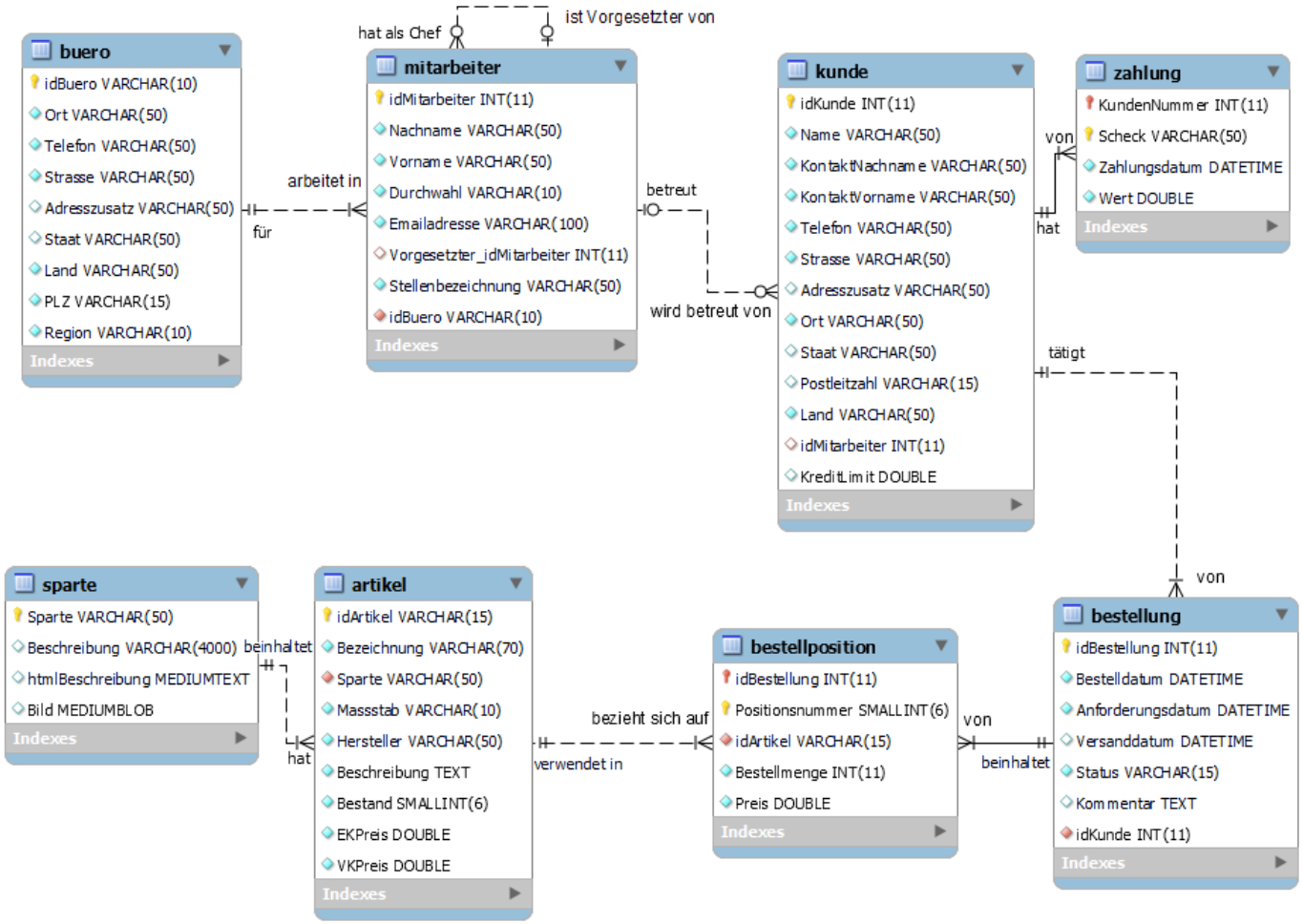
Prof. Dr. Markus Goldstein

SoSe 2022

Inhalt

Beispieldatenbank

- „Classic Models“: siehe MySQL-Server



Definition Unterabfrage

- SQL Statements können eingebettete SELECTs enthalten
  → **Unterabfrage** (bzw **Subselect**)
- Subselects sind möglich in
  - `WHERE` und `HAVING` Klauseln
  - Spaltenliste des äußeren `SELECT`
  - `INSERT`, `UPDATE` und `DELETE` Statements (s.u.)
- Beispiel: Mitarbeiter in Büro „London“ (Büro Nr. 7)

```
SELECT *
FROM Mitarbeiter
WHERE idBuero = 7;
```

- Besser wäre „London“ in der Abfrage zu verwenden …
  → Alternative?

Unterabfrage im WHERE-Statement

- Beispiel: Mitarbeiter in Büro „London“ (Büro Nr. 7)

```
SELECT *
FROM Mitarbeiter
WHERE idBuero = ( SELECT idBuero
          FROM Buero
          WHERE Ort = 'London');
```

- Zuerst wird innere Abfrage ausgeführt, dann die äußere Abfrage
- **Doppelte** Anfrage an die DB
  - Verlängert die Abfragezeit
  - Nicht ideal – diese Abfrage lässt sich auch anders formulieren (später mehr: JOIN)
  - Manchmal sind Unterabfragen aber nicht vermeidbar
- **Regel: Unterabfragen so sparsam wie möglich einsetzen**

Unterabfrage mit Aggregat

- Beispiel: Differenz zum durchschnittlichen Kreditlimit eines Kunden
- Gesucht ist eine Auflistung von Kundennummer, Kundenname, Kreditlimit und Differenz zum durchschnittlichen Kreditlimit aller Kunden, deren Limit überdurchschnittlich hoch ist.

```
SELECT idKunde, Kreditlimit, Kreditlimit -
  (SELECT AVG(Kreditlimit) FROM Kunde) AS WertDiff
 FROM Kunde
 WHERE Kreditlimit > ( SELECT AVG(Kreditlimit)
                       FROM Kunde );
```

Unterabfrage mit Aggregat

- Achtung:
  `WHERE Kreditlimit > AVG(Kreditlimit)` wäre falsch!

- Erklärungsversuch durch Berechnen der Unterabfragen
  - Unterabfrage ermittelt Durchschnittswert → 67659.02
  - Dann in äußeren `SELECT` Einkäufe suchen, die teurer waren → äquivalente Abfrage hier:

```
SELECT idKunde, Kreditlimit,
Kreditlimit - 67659.02 AS WertDiff
FROM Kunde
WHERE Kreditlimit > 67659.02;
```

Regeln für Unterabfragen

- `ORDER BY` Klauseln
  - **nicht** in Unterabfragen
  - nur in äußerstem `SELECT`-Statement

- Spaltenliste der Unterabfrage
  - nur **ein einziger** Spaltenname als Ergebnis
  - Ausnahme: `EXISTS` (später mehr)

- Bei Vergleichen: Unterabfrage steht **rechts**

Geschachtelte Unterabfragen

- Beispiel:
  Ermittle alle Bestellungen deren Kunden von Herrn „Bott“ betreut werden

```
SELECT * FROM Bestellung
WHERE idKunde IN (SELECT idKunde
            FROM Kunde
            WHERE idMitarbeiter = (
                SELECT idMitarbeiter
                FROM Mitarbeiter
                WHERE Nachname = 'Bott'));
```

- Verwendung von `IN`
  - immer, wenn Ergebnis der Unterabfrage mehrwertig
  - hier sinnvolle Verwendung des Mengenvergleichs!
- Auch hier gilt: Abfrage besser als JOIN (später)

Geschachtelte Unterabfragen: IN, ANY, ALL

Mengenvergleich „enthalten“: `IN`

Andere Mengenvergleiche: `ANY` und `ALL`

- Voraussetzung für `ANY` und `ALL`: Unterabfrage erzeugt eine einzige numerische Spalte
- `ALL`
  - Bedingung wahr, wenn für alle Werte des Ergebnisses der Unterabfrage richtig
- `ANY` (manchmal auch `SOME`)
  - Bedingung wahr, wenn für irgendeinen Wert des Ergebnisses der Unterabfrage richtig
- Leeres Ergebnis der Unterabfrage
  - `ALL` → wahr
  - `ANY` → falsch

Geschachtelte Unterabfragen: IN, ANY, ALL

Beispiel für `ANY`

- Alle Zahlungen deren Wert größer ist als irgendeine Zahlung die im Jahr 2004 getätigt wurde.

```
SELECT * FROM Zahlung
WHERE Wert > ANY
    (SELECT Wert FROM Zahlung
      WHERE YEAR(Zahlungsdatum) = 2004);
```

- Ergebnis der inneren Abfrage ist
  {1676.14, …, 116208.40}
- Äußere Abfrage wählt die aus, die höher waren, als **einer** dieser Beträge (also alle > 1676.14 [hier das Minimum])

Geschachtelte Unterabfragen: IN, ANY, ALL

Beispiel für `ALL`

- Alle Zahlungen deren Wert größer ist als alle Zahlungen die im Jahr 2004 getätigt wurde.

```
SELECT * FROM Zahlung
WHERE Wert > ALL
    (SELECT Wert FROM Zahlung
      WHERE YEAR(Zahlungsdatum) = 2004);
```

- Ergebnis der inneren Abfrage ist wieder {1676.14, …, 116208.40}
- Äußere Abfrage wählt die aus, die teurer waren, als **alle** dieser Beträge (also > 116208.40 [hier das Maximum])

Geschachtelte Unterabfragen: IN, ANY, ALL

Alternative Abfragen für `ANY` und `ALL`:

- `ANY`

```
SELECT * FROM Zahlung
WHERE Wert > (SELECT MIN(Wert) FROM Zahlung
                WHERE YEAR(Zahlungsdatum) = 2004);
```

- `ALL`

```
SELECT * FROM Zahlung
WHERE Wert > (SELECT MAX(Wert) FROM Zahlung
                WHERE YEAR(Zahlungsdatum) = 2004);
```

Inhalt

Abfragen über mehrere Tabellen (Verbund)

- Ergebnisspalten
  - aus einer Tabelle → Unterabfrage möglich
  - aus mehreren Tabellen → `JOIN` zwangsweise notwendig
- Erinnerung: Wir vermeiden Unterabfragen, wenn möglich → `JOIN` bevorzugt, da i.d.R. schneller
- `JOIN`
  - `FROM`-Klausel enthält mehrere Tabellennamen, durch Komma getrennt
  - `WHERE`-Klausel mit Vergleich der Joinspalten
  - **Alternativ** mit Schlüsselwort `JOIN` und Bedingung `ON` (moderneres SQL)
- Häufigster Fall: Verbund über Fremdschlüssel
- **Aliastrennung**: Leerzeichen oder Schlüsselwort `AS`

Einfacher JOIN (alte Schreibweise)

- Kunden und Mitarbeiter
  - Auflistung aller Kunden und den passenden betreuenden Mitarbeitern

```
SELECT Name, Nachname, Vorname
 FROM Kunde, Mitarbeiter
 WHERE Kunde.idMitarbeiter =
       Mitarbeiter.idMitarbeiter;
```

- Vorgehen
  - Nur Kunden mit zugeordnetem Mitarbeiter erscheinen im Ergebnis
  - Kunden ohne Mitarbeiter (`NULL`) erscheinen nicht
  - Verbund hier über Fremdschlüssel

Einfacher JOIN: Alternativen

- Verwendung von Tabellen-Alias
  - Erleichtern Schreiben von Abfragen
  - Erlauben leichtere Qualifikation von Spalten (z.B. k.idKunde (eindeutig!) statt nur idKunde (zweideutig, da in beiden Tabellen)
  - AS kann weggelassen werden (Kunde `AS` k)

```
SELECT k.Name, m.Nachname, m.Vorname
 FROM Kunde k, Mitarbeiter m
 WHERE k.idMitarbeiter = m.idMitarbeiter;
```

- Alternative mit Schlüsselwörtern `JOIN` und `ON`

```
SELECT k.Name, m.Nachname, m.Vorname
 FROM Kunde k JOIN Mitarbeiter m
  ON k.idMitarbeiter = m.idMitarbeiter;
```

  - **Wir verwenden diese modernere Schreibweise!**

Einfacher JOIN: neue Schreibweise

- Falls Sie mit der „alten Schreibweise“ vertraut sein sollten, versuchen Sie bitte ab jetzt die „neue Schreibweise“ (nach SQL/92) zu verwenden: `[INNER] JOIN … ON …`
- Vorteile
  - `WHERE` Klausel klar getrennt von den `ON`-Bedingungen
  - Ein zusammenhängendes `JOIN … ON …` besser lesbar bei vielen Tab.

```
SELECT a.A, b.B, c.C
 FROM TabA a
 JOIN TabB b ON a.fk = b.pk
 JOIN TabC c ON b.fk = c.pk
 WHERE a.A = 0;
```

JOIN von TabA und TabB mit FK/PK in einer Zeile

WHERE nicht mit ON gemischt...

```
SELECT a.A, b.B, c.C
 FROM TabA a, TabB b, TabC c
 WHERE a.fk = b.pk AND b.fk = c.pk AND a.A = 0;
```

  - **Wir verwenden diese modernere Schreibweise!**

Einfacher JOIN: Beobachtung

- `JOIN` mit *

```
SELECT *
FROM Kunde k JOIN Mitarbeiter m
ON k.idMitarbeiter = m.idMitarbeiter;
```

- `IdMitarbeiter` doppelt im Ergebnis (aus beiden Tabellen)

	Adresszusatz	Ort	Staat	Postleitzahl	Land	idMitarbeiter	KreditLimit	idMitarbeiter	Nachname	Vorname	Durchwahl	Emailadresse
ona St.	NULL	San Rafael	CA	97562	USA	1165	210500	1165	Jennings	Leslie	x3291	ljennings@classicmod
th Pendale Street	NULL	San Francisco	CA	94217	USA	1165	64600	1165	Jennings	Leslie	x3291	ljennings@classicmod
th Circle	NULL	Burlingame	CA	94217	USA	1165	84600	1165	Jennings	Leslie	x3291	ljennings@classicmod
ona St.	NULL	San Francisco	CA	94217	USA	1165	105000	1165	Jennings	Leslie	x3291	ljennings@classicmod
ile Ln.	NULL	San Jose	CA	94217	USA	1165	77600	1165	Jennings	Leslie	x3291	ljennings@classicmod

  - Eigentlich unnötig …
  - Abhilfe: `NATURAL JOIN`

NATURAL JOIN

- Konzept: „Natürlicher“ Join
  - Verwendung aller gemeinsamen Attribute (**Attributsnamen**)
  - Vergleich auf Gleichheit
  - Zumeist: Abgleich von Fremd- und Primärschlüsselattributen

- wegen Häufigkeit eigenes Schlüsselwort : `NATURAL JOIN`

```
SELECT k.Name, b.Bestelldatum
FROM Kunde k NATURAL JOIN Bestellung b;
```

- Bei Verwendung von * tauchen gleiche Namen nur einmal im Ergebnis auf (→ FK/PK-Verbund)

- `NATURAL JOIN` funktioniert nur, wenn Benennungen passen (z.B. Spalte „Name“ in Kunde und Mitarbeiter → Problem!)

Sortieren eines JOINs

- Beispiel: Bestellungen des Kunden `Mini Wheels Co.`
- Auflistung von Bestelldatum, Anforderungsdatum und Status, sortiert nach Aktualität

```
SELECT b.Bestelldatum, b.Anforderungsdatum,
       b.Status AS Bestellstatus
 FROM Kunde k JOIN Bestellung b
  ON k.idKunde = b.idKunde
 WHERE k.Name = 'Mini Wheels Co.'
 ORDER BY Bestelldatum, Anforderungsdatum,
          Bestellstatus DESC;
```

- Sortierung erfolgt gemäß Spaltennamen der Ausgabe (Bestelldatum, Anforderungsdatum, Bestellstatus)

JOINs über mehrere Tabellen

- Beispiel: Bestellte Artikel des Kunden `Mini Wheels Co.`
  - Auflistung von Bestelldatum, Bestellmenge und Artikelbezeichnung

```
SELECT b.Bestelldatum, p.Bestellmenge,
       a.Bezeichnung
  FROM Kunde k ,Bestellung b, Bestellposition p,
       Artikel a
 WHERE k.idKunde = b.idKunde
   AND k.Name = 'Mini Wheels Co.'
   AND p.idBestellung = b.idBestellung
   AND p.idArtikel = a.idArtikel
 ORDER BY Bezeichnung, Bestellmenge;
```

- Hier in „alter“ Schreibweise (schwer zu Lesen)
- Sortierung erfolgt gemäß Spaltennamen der Ausgabe (Bezeichnung, Bestellmenge)

JOINs über mehrere Tabellen

- Beispiel: Bestellte Artikel des Kunden `Mini Wheels Co.`
  - Auflistung von Bestelldatum, Bestellmenge und Artikelbezeichnung

```
SELECT b.Bestelldatum, p.Bestellmenge,
       a.Bezeichnung
  FROM Kunde k JOIN Bestellung b ON k.idKunde =
       b.idKunde
  JOIN Bestellposition p
    ON p.idBestellung = b.idBestellung
  JOIN Artikel a ON p.idArtikel = a.idArtikel
 WHERE k.Name = 'Mini Wheels Co.'
 ORDER BY Bezeichnung, Bestellmenge;
```

- Hier in „**neuer**“ Schreibweise
- Sortierung erfolgt gemäß Spaltennamen der Ausgabe (Bezeichnung, Bestellmenge)

JOINs über mehrere Tabellen und GROUP BY

- Bestellmenge je Artikel und Bestelldatum des Kunden `Mini Wheels Co.`
  - Auflistung von Bestelldatum, Artikel und Bestellmenge der Bestellung

```
SELECT b.Bestelldatum, SUM(p.Bestellmenge) AS
       SUM_Menge, a.Bezeichnung
  FROM Kunde k JOIN Bestellung b
    ON k.idKunde = b.idKunde
 JOIN Bestellposition p
    ON p.idBestellung = b.idBestellung
 JOIN Artikel a ON p.idArtikel = a.idArtikel
 WHERE k.Name = 'Mini Wheels Co.'
 GROUP BY b.Bestelldatum, a.Bezeichnung
 ORDER BY Bestelldatum, Bezeichnung, SUM_Menge;
```

- Gruppierung erfolgt gemäß orig. Spaltennamen

Theorie: Auswertung eines JOINs

- Brute Force Ansatz
  - `FROM/JOIN` Klausel auswerten
  - Jede Zeile der ersten Tabelle mit jeder Zeile aus der zweiten verbinden (Kreuzprodukt) → **Sehr großes Ergebnis**
  - bei vorhandener `WHERE/ON` Klausel Bedingung für jede Zeile im Kreuzprodukt testen → Tabelle wird wieder kleiner (FK == PK)
  - nur Attribute aus Spaltenliste verwenden
  - bei `DISTINCT` Duplikate eliminieren
  - bei `ORDER BY` sortieren
- **Query Optimizer** verbessert Ansatz
  - Abfragen werden oft „umgeschrieben“ (z.B. `WHERE` zuerst)
  - Die sehr großen Kreuzprodukt-Tabellen werden nicht vollständig erzeugt

CROSS JOIN: Kreuzprodukt

- Vollständiges Kreuzprodukt auch in SQL erzeugbar

```
SELECT * FROM Mitarbeiter CROSS JOIN Buero;
```

  - Vorsicht!

- Alternativ: `JOIN` ohne `WHERE`/`ON`-Klausel

```
SELECT * FROM Mitarbeiter, Buero;
```

```
SELECT * FROM Mitarbeiter JOIN Buero;
```

- Das Beispiel zeigt: Nie die `WHERE`/`ON`-Klausel vergesen
- Anstatt `CROSS JOIN` kann man auch nur `JOIN` schreiben.

VIEWs: Virtuelle Tabellen

- Virtuelle Tabellen
  - Benannte Abfragen
  - Verwendung wie Basistabellen
  - SQL DDL zum Anlegen:

```
CREATE VIEW Viewname AS ( SELECT Statement );
```

- Bsp: Alle Büros und Mitarbeiter als `VIEW`

```
CREATE VIEW MitarbeiterBuero AS (
 SELECT m.Nachname, m.Vorname, b.Strasse, b.Ort
 FROM Buero b JOIN Mitarbeiter m
 ON b.idBuero = m.idBuero);
```

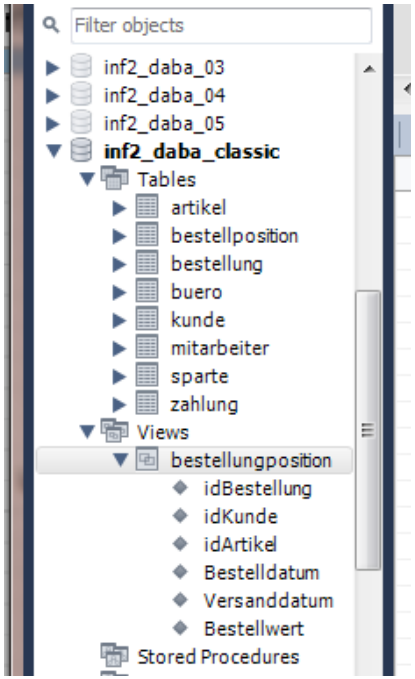
VIEWs: Virtuelle Tabellen

- VIEWs können dann wie normale Tabellen verwendet werden

```
SELECT * FROM MitarbeiterBuero;
```

- VIEWs sind keine echten Tabellen, sondern werden „on-the-fly“ generiert
- D.h. keine Kopie der Daten
- VIEWs führen auch nicht zu schnelleren Antwortzeiten
- In CASE-Tools oftmals getrennt von Tabellen

→



VIEWs: Virtuelle Tabellen

- Beispiel: Alle Bestellungen mit Bestellwert

```
CREATE VIEW BestellungPosition AS (
SELECT b.idBestellung, b.idKunde, bp.idArtikel,
b.Bestelldatum, b.Versanddatum,
      bp.Preis *  bp.Bestellmenge AS Bestellwert
   FROM Bestellung b JOIN Bestellposition bp
   ON b.idBestellung = bp.idBestellung);
```

idBestellung	idKunde	idArtikel	Bestelldatum	Versanddatum	Bestellwert
10100	363	S24 3969	2003-01-06 00:00:00	2003-01-10 00:00:00	1729.21
10100	363	S18 2248	2003-01-06 00:00:00	2003-01-10 00:00:00	2754.5
10100	363	S18 1749	2003-01-06 00:00:00	2003-01-10 00:00:00	4080
10100	363	S18 4409	2003-01-06 00:00:00	2003-01-10 00:00:00	1660.12
10101	128	S18 2795	2003-01-09 00:00:00	2003-01-11 00:00:00	4343.56
10101	128	S24 2022	2003-01-09 00:00:00	2003-01-11 00:00:00	2040.1000000000001
10101	128	S24 1937	2003-01-09 00:00:00	2003-01-11 00:00:00	1463.8500000000001
10101	128	S18 2325	2003-01-09 00:00:00	2003-01-11 00:00:00	2701.5
10102	181	S18 1367	2003-01-10 00:00:00	2003-01-14 00:00:00	1768.3300000000002
10102	181	S18 1342	2003-01-10 00:00:00	2003-01-14 00:00:00	3726.45
10103	121	S24 2300	2003-01-29 00:00:00	2003-02-02 00:00:00	3864.2400000000002

Exkurs: Materialized VIEWs

- `VIEWs` sind nur virtuelle Tabellen
- Es können aber auch Tabellen als Kopie erzeugt werden
  → `MATERIALIZED VIEW`
- **Vorsicht**
  - Datenintegrität geht verloren (Daten doppelt)
  - `UPDATEs` an den Originaltabellen werden nicht automatisch kopiert
    → Inkonsistenz!
- Wird in Ausnahmefällen als „Cache“ benutzt
  - Syntax

```
CREATE TABLE BestellungPosition AS (
SELECT … JOIN … );
```

  - Abfragen schneller
- Einige RDBMS (nicht MySQL) unterstützen Update-Strategien

Aufgaben

Bitte bearbeiten Sie jetzt die Aufgaben in Moodle zum Kapitel 7.

- **Teil A**

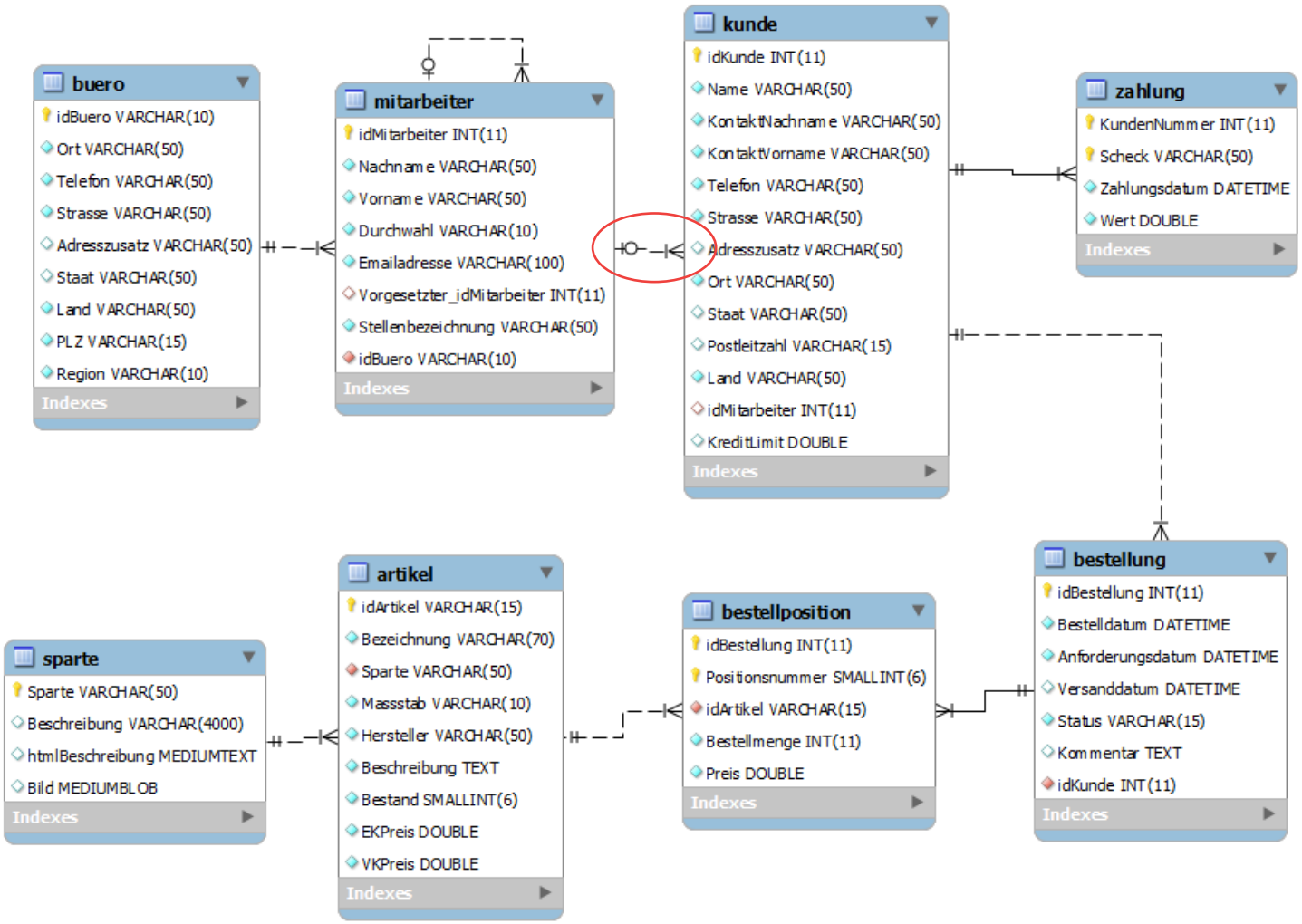
Inhalt

Äußere JOINs

- Innerer `JOIN`:
  - Zeile ohne Entsprechung im `ON` fehlt komplett im Ergebnis
  - Anstatt `JOIN` kann auch `INNER JOIN` geschrieben werden
- Äußere Joins (engl. outer join):
  - Zeile mit Entsprechung im `ON` erscheint (wie inner join)
  - Zeile ohne Entsprechung in der anderen Tabelle erscheint auch
  - Fehlende Werte sind `NULL`
  - Nicht mit `CROSS JOIN` verwechseln
- Varianten:
  - `LEFT JOIN`: linke Tabelle wird komplett ausgegeben
  - `RIGHT JOIN`: rechte Tabelle wird komplett ausgegeben
  - `FULL JOIN`: beide Tabellen werden komplett ausgegeben

Beispieldatenbank

- „Classic Models“: siehe MySQL-Server



Äußere JOINs

- Äußere `JOINs` spielen vor allen bei Optionalität eine wichtige Rolle
  - Ausgabe auch der nicht-referenzierten Kunden!
- Manche Äußere `JOINs` nicht in jedem RDBMS verfügbar
  - `RIGHT JOIN` fehlt in Sqlite
  - `FULL JOIN` fehlt in MySQL (Alternative später)
- Am häufigsten: `LEFT JOIN`
  - Listet auch Einträge aus der linken Tabelle, für die es keinen passenden Eintrag in der rechten Tabelle gibt
  - Beispiel nächste Folie...

Äußere JOINs

- Beispiel(e): Mitarbeiter mit Kunden bzw. Büros verbinden

```
-- Mitarbeiter in Boston
SELECT m.Nachname, m.Vorname, b.Ort
FROM Mitarbeiter m JOIN Buero b
ON m.idBuero = b.idBuero
WHERE b.Ort = 'Boston';
-- Kunden des Mitarbeiters
SELECT m.Nachname, m.Vorname, k.Name, k.Ort
FROM Mitarbeiter m JOIN Kunde k
ON m.idMitarbeiter = k.idMitarbeiter;
```

- Einfacher JOIN:
  Alle Zeilen mit Übereinstimmungen werden ausgegeben
  → Kunden, die keinen Mitarbeiter zugeteilt sind, fehlen

Äußere JOINs: LEFT JOIN

- Beispiel(e): Alle Kunden, auch die ohne Mitarbeiter

```
-- Alle Kunden, falls vorhanden auch Betreuer
SELECT m.Nachname, m.Vorname, k.Name, k.Ort
 FROM Kunde k LEFT JOIN Mitarbeiter m
 ON m.idMitarbeiter = k.idMitarbeiter;

-- Nur Kunden, die keinen Betreuer haben
SELECT m.Nachname, m.Vorname, k.Name, k.Ort
 FROM Kunde k LEFT JOIN Mitarbeiter m
 ON m.idMitarbeiter = k.idMitarbeiter
 WHERE k.idMitarbeiter IS NULL;
```

Nachname	Vorname	Name	Ort
NULL	NULL	Asian Shopping Network,...	Singapore
Hernandez	Gerard	Mini Caravv	Strasbourg
Nishi	Mami	King Kong Collectables, Co.	Central Hong...
Gerard	Martin	Enaco Distributors	Barcelona
Thompson	Leslie	Boards & Toys Co.	Glendale
NULL	NULL	NatÃ¼rlich Autos	Cunewalde
Castillo	Pamela	Heintze Collectables	Ã...rhus
Tseng	Foon Yue	OuÃ©bec Home Shoppin...	MontrÃ©al
NULL	NULL	ANG Resellers	Madrid

Äußere JOINs: RIGHT JOIN

- Beispiel(e): Alle Mitarbeiter, auch die ohne Kunden

```
-- Alle Mitarbeiter, falls vorhanden auch Kunde
SELECT m.Nachname, m.Vorname, k.Name, k.Ort
 FROM Kunde k RIGHT JOIN Mitarbeiter m
 ON m.idMitarbeiter = k.idMitarbeiter;

-- Nur Mitarbeiter ohne Kunden
SELECT m.Nachname, m.Vorname, k.Name, k.Ort
 FROM Kunde k RIGHT JOIN Mitarbeiter m
 ON m.idMitarbeiter = k.idMitarbeiter
 WHERE k.idMitarbeiter IS NULL;
```

Nachname	Vorname	Name	Ort
Murphy	Diane	NULL	NULL
Patterson	Mary	NULL	NULL
Firrelli	Jeff	NULL	NULL
Patterson	William	NULL	NULL
Bondur	Gerard	NULL	NULL
Bow	Anthony	NULL	NULL
King	Tom	NULL	NULL
Kato	Yoshimi	NULL	NULL

Existenzabfrage: EXISTS

- Immer in Kombination mit Unterabfrage
  - dürfen beliebig viele Spalten enthalten
  - i.A. spezifiziert als `(SELECT * FROM ...)`
- Bewertung in „wahr“ und „falsch“
- `EXISTS`
  - wahr, wenn Unterabfrage nicht leeres Ergebnis liefert
  - falsch, wenn kein Treffer in Unterabfrage
- `NOT EXISTS` ist Gegenteil von `EXISTS`

Existenzabfrage: EXISTS

- Beispiel: Mitarbeiter mit Kunden in Boston

```
SELECT * FROM Mitarbeiter m
WHERE EXISTS (
    SELECT * FROM Kunde k
    WHERE m.idMitarbeiter = k.idMitarbeiter
    AND Ort = 'Boston');
```

- Dies ist eine andere (kompliziertere) Schreibweise wie ein JOIN:

```
SELECT m.* FROM Mitarbeiter m
JOIN Kunde k ON m.idMitarbeiter = k.idMitarbeiter
WHERE Ort = 'Boston';
```

Existenzabfrage: EXISTS

Beispiel: Mitarbeiter mit Kunden in Boston

- Anmerkungen:
  - Bedingung `m.idMitarbeiter = k.idMitarbeiter` notwendig!
- Ohne Bedingung:
  - Innere Abfrage wahr, da es Kunden in Boston gibt
  - äußere Abfrage wäre dann:

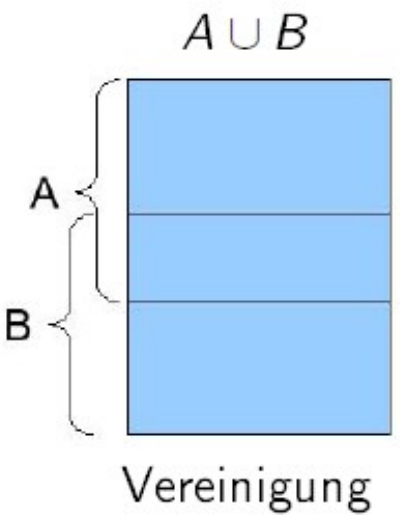
```
SELECT *
 FROM Mitarbeiter m
 WHERE TRUE
```

- alle Mitarbeiter werden ausgegeben (→ Kein JOIN mehr)
- Wirkung der Bedingung: Frage, ob bestimmter Mitarbeiter (m.idMitarbeiter) Kunden in Boston betreut

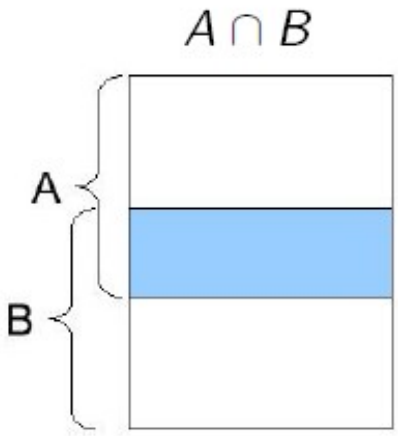
Inhalt

Mengenoperationen

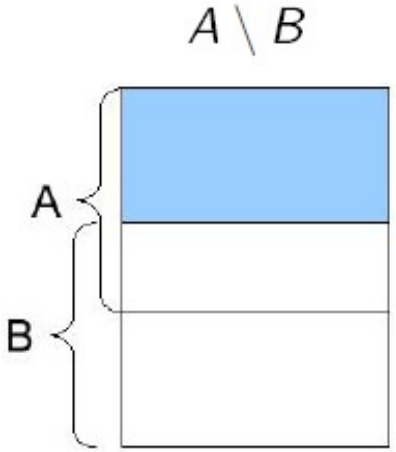
- Veranschaulichung



Vereinigung



Schnittmenge



Differenz

Mengenoperationen

Mengenoperationen mit SQL

- Schlüsselwort `UNION`
  - Vereinigungsmenge A ∪ B
  - bei Tabellen: alle Datensätze aus A und B aneinanderhängen
- Schlüsselwort `INTERSECT`
  - Schnittmenge A ∩ B
  - bei Tabellen: alle Datensätze, die in A und B vorkommen
- Schlüsselwort `EXCEPT`
  - Differenzmenge A \ B
  - bei Tabellen: alle Datensätze aus A, die aber nicht B vorkommen
- Ergebnis aus mehreren Abfragen zusammengesetzt

Mengenoperationen

- **Voraussetzung**: Vereinigungskompatibilität, d. h. die Struktur der Abfrageergebnisse muss übereinstimmen
  - Gleiche Anzahl von Spalten
  - Datentypen müssen passen

- Ersatzformulierungen:
  - `INTERSECT` und `EXCEPT` im Standard vorgesehen, aber nicht immer vom DBMS implementiert
  - → Alternativformulierungen nötig
- Beispiel für `UNION`:

```
SELECT CONCAT(Vorname,' ',Nachname) AS Name FROM Mitarbeiter
UNION
SELECT Name FROM Kunde;
```

Mengenoperationen: UNION

- Weiteres Beispiel für `UNION`:
  - Mitarbeiter mit Kunden aus verschiedenen Orten
  - Als (schlechter) Ersatz für `OR`

```
SELECT m.* FROM Mitarbeiter m NATURAL JOIN Kunde k
 WHERE k.Ort = 'Boston'
UNION
SELECT m.* FROM Mitarbeiter m NATURAL JOIN Kunde k
 WHERE k.Ort = 'San Francisco';
```

- Bewertung
  - Die Abfrage mit `OR` in der `WHERE`-Klausel ist schneller, da die Tabelle nur einmal durchlaufen wird

Mengenoperationen: UNION

- Verwendung von `UNION` um einen `FULL JOIN` zu erhalten
  - `FULL JOIN` nicht in MySQL unterstützt
  - `FULL JOIN = LEFT JOIN + RIGHT JOIN`

```
SELECT m1.Nachname AS Mitarbeiter, m2.Nachname AS Vorgesetzter
 FROM Mitarbeiter m1
 LEFT JOIN Mitarbeiter m2
  ON m1.Vorgesetzter_idMitarbeiter = m2.idMitarbeiter
UNION
SELECT m1.Nachname AS Mitarbeiter, m2.Nachname AS Vorgesetzter
 FROM Mitarbeiter m1
 RIGHT JOIN Mitarbeiter m2
  ON m1.Vorgesetzter_idMitarbeiter = m2.idMitarbeiter
```

- **Self Join** aufgrund des rekursiven Beziehungstyps (`A JOIN A`)
- Ersatz für `FULL JOIN`: Anzeigen aller Mitarbeiter inkl. Vorgesetzte, und Mitarbeiter ohne Vorgesetzte (oberer Teil) und Vorgesetzte (Personen), die keine Mitarbeiter haben (unten)

Mengenoperationen: UNION

- Erweitertes Beispiel: Zeige alle Bestellungen von allen Kunden und die Gesamtsumme an
  - `VIEW` als Hilfstabelle

```
CREATE VIEW BestellungKunde AS(
  SELECT k.Name, b.idBestellung, b.Bestelldatum,
  ROUND(SUM(Bestellmenge * Preis),2) AS
  Bestellwert
  FROM Bestellung b JOIN Bestellposition bp
   ON b.idBestellung = bp.idBestellung
  JOIN Kunde k
   ON k.idKunde = b.idKunde
  GROUP BY b.idKunde, b.idBestellung
);
```

Mengenoperationen: UNION

- Erweitertes Beispiel: Zeige alle Bestellungen von allen Kunden und die Gesamtsumme an
  - Bestellungen von Lyon Souveniers mit Zusammenfassung

```
SELECT Name, Bestelldatum, Bestellwert
 FROM BestellungKunde
 WHERE Name = 'Lyon Souveniers'
UNION
SELECT 'Summe','Alle', SUM(Bestellwert)
 FROM BestellungKunde
 WHERE Name = 'Lyon Souveniers';
```

  - Vereinigungskompatibilität: Literale im zweiten SELECT

Mengenoperationen: INTERSECT

- Aufgabe
  - Welche Kunden haben sowohl einen „Alfa Romeo GTA“ als auch einen „Harley Davidson Ultimate Chopper“ gekauft?

```
SELECT k.idKunde, k.Name
 FROM Kunde k
 JOIN Bestellung b ON k.idKunde = b.idKunde
 JOIN Bestellposition bp ON b.idBestellung = bp.idBestellung
 JOIN Artikel a ON bp.idArtikel = a.idArtikel
 WHERE a.Bezeichnung = '1972 Alfa Romeo GTA'
INTERSECT
SELECT k.idKunde, k.Name
 FROM Kunde k
 JOIN Bestellung b ON k.idKunde = b.idKunde
 JOIN Bestellposition bp ON b.idBestellung = bp.idBestellung
 JOIN Artikel a ON bp.idArtikel = a.idArtikel
 WHERE a.Bezeichnung = '1969 Harley Davidson Ultimate
                       Chopper';
```

  - `INTERSECT` in MySQL nicht implementiert :-(

Mengenoperationen: INTERSECT

- Aufgabe: Lösung mit Unterabfrage und IN
  - Welche Kunden haben sowohl einen „Alfa Romeo GTA“ als auch einen „Harley Davidson Ultimate Chopper“ gekauft?

```
SELECT k.idKunde, k.Name
 FROM Kunde k
 JOIN Bestellung b ON k.idKunde = b.idKunde
 JOIN Bestellposition bp ON b.idBestellung = bp.idBestellung
 JOIN Artikel a ON bp.idArtikel = a.idArtikel
 WHERE a.Bezeichnung = '1972 Alfa Romeo GTA'
  AND k.idKunde IN (
   SELECT k.idKunde
    FROM Kunde k
    JOIN Bestellung b ON k.idKunde = b.idKunde
    JOIN Bestellposition bp ON b.idBestellung = bp.idBestellung
    JOIN Artikel a ON bp.idArtikel = a.idArtikel
    WHERE a.Bezeichnung = '1969 Harley Davidson Ultimate
                          Chopper'
  );
```

  - `INTERSECT` als Unterabfrage formuliert

Mengenoperationen: EXCEPT

- Aufgabe:
  - Welche Kunden haben einen „Alfa Romeo GTA“ aber keinen „Harley Davidson Ultimate Chopper“ gekauft?

```
SELECT k.idKunde, k.Name
 FROM Kunde k
 JOIN Bestellung b ON k.idKunde = b.idKunde
 JOIN Bestellposition bp ON b.idBestellung = bp.idBestellung
 JOIN Artikel a ON bp.idArtikel = a.idArtikel
 WHERE a.Bezeichnung = '1972 Alfa Romeo GTA'
EXCEPT
SELECT k.idKunde, k.Name
 FROM Kunde k
 JOIN Bestellung b ON k.idKunde = b.idKunde
 JOIN Bestellposition bp ON b.idBestellung = bp.idBestellung
 JOIN Artikel a ON bp.idArtikel = a.idArtikel
 WHERE a.Bezeichnung = '1969 Harley Davidson Ultimate
                       Chopper';
```

  - `EXCEPT` nicht in MySQL implementiert :-(

Mengenoperationen: EXCEPT

- Aufgabe: Lösung mit Unterabfrage und NOT IN
  - Welche Kunden haben einen „Alfa Romeo GTA“ aber keinen „Harley Davidson Ultimate Chopper“ gekauft?

```
SELECT k.idKunde, k.Name
 FROM Kunde k
 JOIN Bestellung b ON k.idKunde = b.idKunde
 JOIN Bestellposition bp ON b.idBestellung = bp.idBestellung
 JOIN Artikel a ON bp.idArtikel = a.idArtikel
 WHERE a.Bezeichnung = '1972 Alfa Romeo GTA'
  AND k.idKunde NOT IN (
   SELECT k.idKunde
    FROM Kunde k
    JOIN Bestellung b ON k.idKunde = b.idKunde
    JOIN Bestellposition bp ON b.idBestellung = bp.idBestellung
    JOIN Artikel a ON bp.idArtikel = a.idArtikel
    WHERE a.Bezeichnung = '1969 Harley Davidson Ultimate
                          Chopper'
  );
```

  - `EXCEPT` als Unterabfrage und Mengenoperation formuliert

Aufgaben

Bitte bearbeiten Sie jetzt die Aufgaben in Moodle zum Kapitel 7.

- **Teil B**