

# Unterabfragen (Subselects)

- Einwertig

Ergebnis ist ein Skalar

`SELECT COUNT(*) FROM tblArtikel`

- Mehrwertig

Rückgabewert ist ein Tupel

- Tabellenwertig

Sie gibt eine Relation zurück

`SELECT * FROM tblPers`

Eine einzige SQL-Anweisung kann aus mehreren Abfragen bestehen

- Bei Verschachtelung werden diese nacheinander ausgeführt:

```
SELECT * FROM PERS
```

```
WHERE GEHALT = (SELECT MAX(GEHALT)  
FROM PERS)
```

- Gleichwertig, dann parallele Ausführung:  

```
SELECT COUNT(*) FROM PERS + SELECT  
COUNT(*) FROM PERS1
```

- Lösung mit abhängigen Abfragen

- ◆ 

```
SELECT Artikelname, Einzelpreis
      FROM Artikel
     WHERE Einzelpreis >
          (
              SELECT AVG(Einzelpreis)
                FROM Artikel
          )
```

- Die äußere Abfrage ruft die innere Abfrage auf. Die innere Abfrage ermittelt den Durchschnittspreis und übergibt diesen an die äußere Abfrage, welche den Prozess steuert.
- Die innere Abfrage ist einwertig: Es gibt genau einen Durchschnittspreis
- Die äußere Abfrage ist tabellenwertig

## Unterabfragen .....

- werden meistens in Klammern geschrieben
- sollten wenn möglich rechts im Vergleichsausdruck stehen
- dürfen bei Single Row-Vergleichen (<, >, =, etc.) nur einwertig sein.
- dürfen bei IN, ANY und ALL mehrwertig sein.
- können mehrfach verschachtelt werden.
- werden ausgeführt von innen nach außen.
- müssen sich nicht auf dieselbe Tabelle wie die Masterabfrage beziehen.
- lassen sich oft anstelle von JOINS verwenden.

- Beispiel: Alle Mitarbeiter der Abteilungen Entwicklung und (?!?) Vertrieb
- Lösung (Statt **IN** ist auch = **ANY** möglich) :

```
SELECT ang.name
FROM ang
WHERE abt_nr IN (
    SELECT abt_nr
    FROM abt
    WHERE abt_name IN
        ('Vertrieb', 'Entwicklung')
)
```

- Unterabfrage ist mehrwertig: Beliebig viele Personen könnten zu einer der beiden Abteilungen gehören

- Beispiel:

Alle Mitarbeiter, die **nicht** in den Abteilungen Entwicklung **und** Vertrieb arbeiten

- Lösung 1:

```
SELECT name
FROM ang
WHERE abt_nr NOT IN (
    SELECT abt_nr
    FROM abt
    WHERE abt_name IN
        ('Vertrieb', 'Entwicklung')
)
```

- Lösung 2:

```
SELECT name
FROM ang
WHERE abt_nr != ALL (
    SELECT abt_nr
    FROM abt
    WHERE abt_name IN ('Vertrieb',
'Entwicklung')
)
```

- Also:

- ◆ **IN entspricht = ANY**
- ◆ **NOT IN entspricht != ALL**

- In der inneren Abfrage ist eine Spalte enthalten, die sich auf einen Wert aus der äußeren Abfrage bezieht
- Häufig in Verwendung mit EXISTS
  - ◆ EXISTS ist immer wahr, wenn es ein Ergebnis außer NULL gibt

- **Beispiel:** Zeige alle Angestellten, die an Projekten arbeiten
- **Lösung:**

```
SELECT name FROM ang
WHERE EXISTS (
    SELECT * FROM pro_ang
    INNER JOIN ang
    ON ang.a_nr = pro_ang.a_nr
)
```

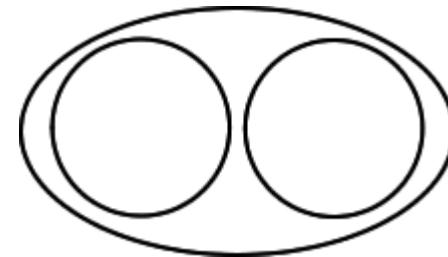
- Beispiel 2:  
Zeige alle Angestellten, die **nicht** an Projekten arbeiten
- Lösung:

```
SELECT name
FROM ang
WHERE NOT EXISTS (
    SELECT * FROM pro_ang
    WHERE ang.a_nr = pro_ang.a_nr
)
```
- Die innere Abfrage wird für **jede** Zeile der Tabelle ang in der äußeren Abfrage ausgeführt

**Situation:** Die Abfragen A und B mit identischer Anzahl Spalten und kompatiblen Datentypen werden parallel ausgeführt, ihre Ergebnisse sollen kombiniert werden.

- UNION: Alle Ergebnisse von Abfrage A oder Abfrage B (einschließendes Oder, Vereinigung,  $A \cup B$ )
- INTERSECT: *Ergebnisse von Abfrage A und Abfrage B* (Logisches Und, Schnittmenge,  $A \cap B$ )
- EXCEPT: *Ergebnisse von Abfrage A, aber nicht von Abfrage B* (Ausschlussbedingung, Mengendifferenz,  $A \setminus B$ )

Vereinigung enthält alle Ergebnisse jeder einzelnen Abfrage

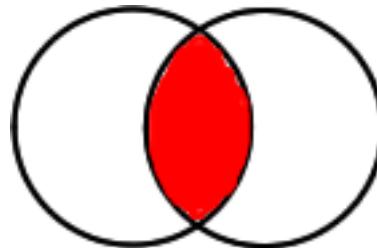


- ◆ UNION Operator
  - ✓ *UNION unterdrückt doppelte Zeilen (implizites DISTINCT)*
  - ✓ *UNION ALL zeigt auch doppelte Zeilen an*
  - ✓ *Syntax: Abfrage A UNION Abfrage B*  
*Abfrage A UNION ALL Abfrage B*

- Beispiel:  
Zeige alle Länder aus denen Kunden oder Lieferanten kommen

- Lösung:  
SELECT Land  
FROM Kunden  
**UNION**  
SELECT Land  
FROM Lieferanten  
ORDER BY Land

- Durchschnitt
  - ◆ Ergebnisse von Abfrage A und Abfrage B



- ◆ Lösung: **INTERSECT**
- ◆ Entspricht EXISTS bei verschachtelten Abfragen

- Beispiel:

Alle Länder, aus denen sowohl Kunden als auch Lieferanten kommen

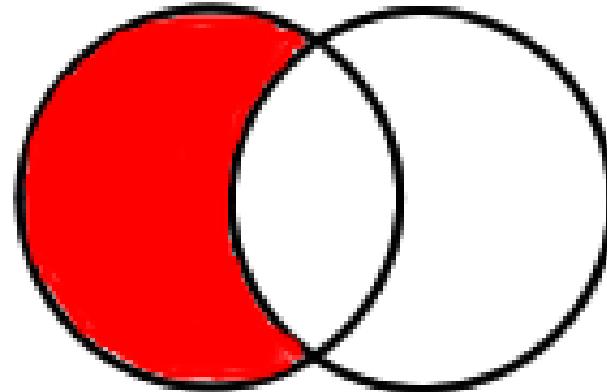
- Lösung:

```
SELECT Land  
FROM Kunden
```

**INTERSECT**

```
SELECT Land  
FROM Lieferanten  
ORDER BY Land
```

- Differenz
  - ◆ Alle Ergebnisse der ersten, aber **nicht** der zweiten Abfrage
  - ◆ Operator ist **EXCEPT**
  - ◆ Entspricht **NOT EXISTS** bei verschachtelten Abfragen
  - ◆ Achtung: Es kommt auf die Reihenfolge der Abfragen an!!!



## Mengenoperator EXCEPT

- Beispiel:  
Alle Länder, aus denen Kunden aber keine Lieferanten kommen
- Lösung:

```
SELECT Land
FROM Kunden
EXCEPT
SELECT Land
FROM Lieferanten
ORDER BY Land
```