

SQL

Grundlagen

Abfragen auf mehreren Tabellen



A word cloud of SQL keywords in blue text. The words are arranged in a dense, overlapping manner. The largest words are 'ROLLBACK', 'COMMIT', 'DATABASE', 'INDEX', 'VIEW', 'SELECT', 'OUTER', 'TABLE', 'JOIN', 'RIGHT', and 'OUTER'. Other visible words include 'RECURSION', 'DDL', 'RDBMS', 'DELETE', 'VALUES', 'DEFAULT', 'CREATE', 'LEFT', 'INNER', 'SAVEPOINT', 'EXPLAIN', 'PLAN', 'DROP', 'TRIGGER', 'TRANSACTION', 'DML', 'NUMERIC', 'ISOLATION', 'LEVEL', 'DATA', 'DEFINITION', 'LANGUAGE', 'DEADLOCK', 'INSERT', 'JOIN', 'EXTRACT', 'INNER', 'EXPLAIN', 'PLAN', 'DROP', 'TRIGGER', 'TRANSACTION', 'DML', 'NUMERIC', 'ISOLATION', 'LEVEL', 'DATA', 'DEFINITION', 'LANGUAGE', 'DEADLOCK'.

Verwendung der Unterlagen



- Bitte beachte, dass die vorliegenden Unterlagen als Begleitmaterial für die Schulung erstellt worden sind. Sie sind daher nur eingeschränkt zum Selbststudium geeignet.
- Die meisten der vorgestellten Konzepte, Anweisungen, Syntaxdiagramme u.ä. bieten noch weitere als die hier vorgestellten Möglichkeiten. Die Darstellung ist im Wesentlichen auf den Umfang der Grundlagenschulung eingeschränkt.

Voraussetzungen

- Grundkenntnisse Datenbanken (z.B. aus Teil 1 der Schulung)
- Grundkenntnisse SQL Anweisungen (DML – z.B. aus Teil 2 und 3 der Schulung)
- Für die Übungsdatenbank: Mac/PC mit aktueller Java-Version

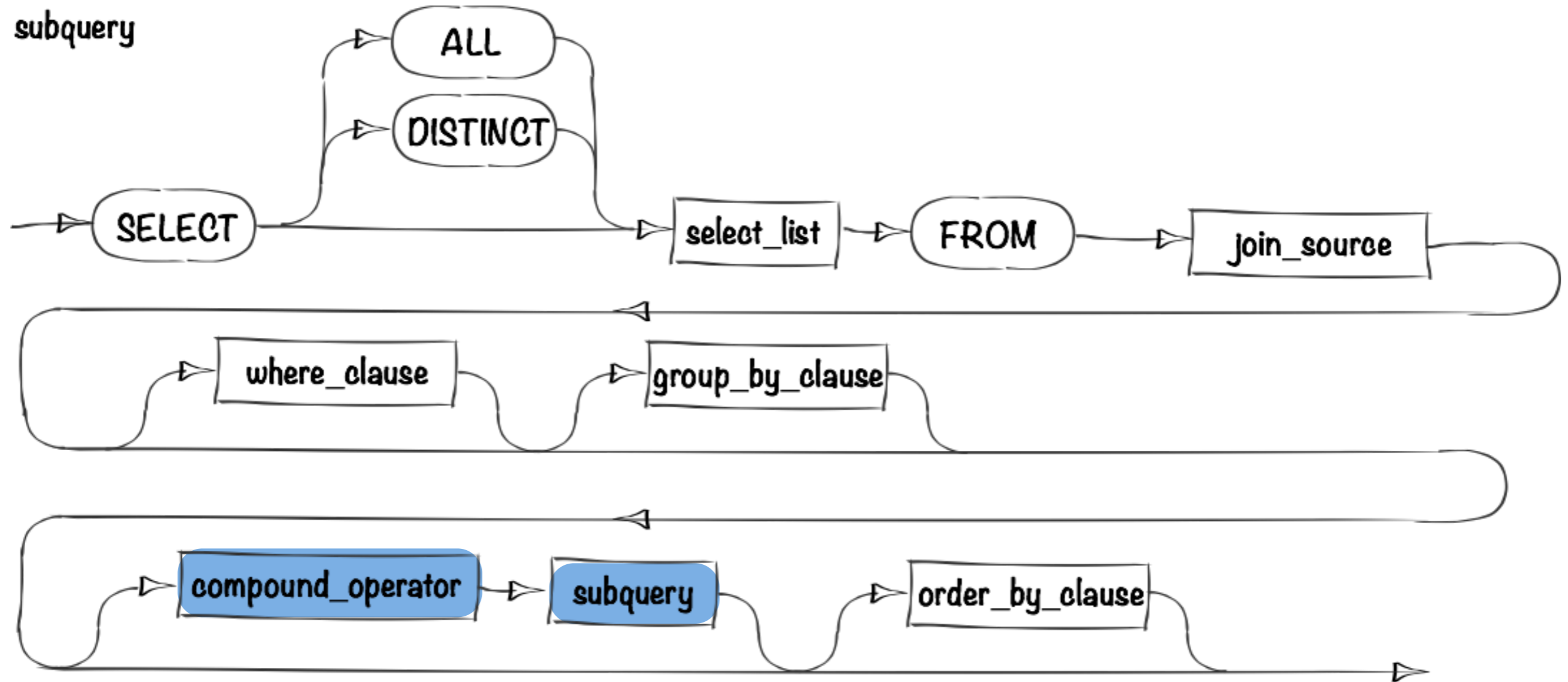
Übersicht

- Mengenoperationen in SQL
- Joins
- Unterabfragen

Mengenoperationen

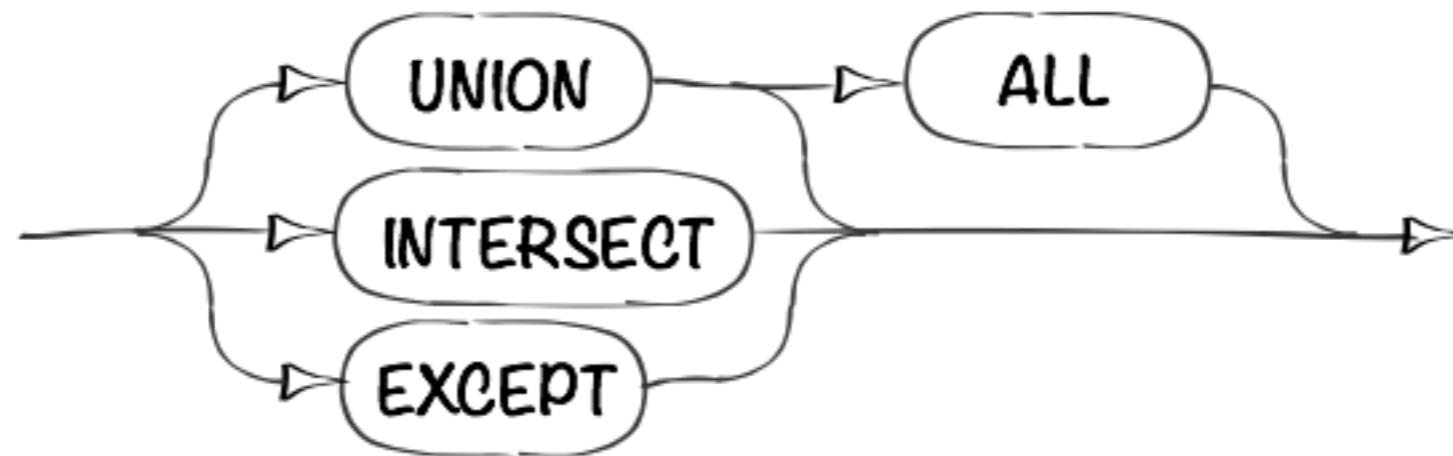
- Mengenoperatoren verbinden die Ergebnisse jeweils zweier **SELECT**-Anweisungen zu einer gemeinsamen Ergebnistabelle.
- Mengenoperatoren sind **UNION**, **INTERSECT** und **EXCEPT**.
- Mehrere (auch verschiedene) Mengenoperatoren können in einer Anweisung kombiniert werden.
- Durch Klammern können Prioritäten gesetzt werden.

Mengenoperationen – Syntax



Mengenoperationen – Syntax

`compound_operator`



Mengenoperationen – Beispiel

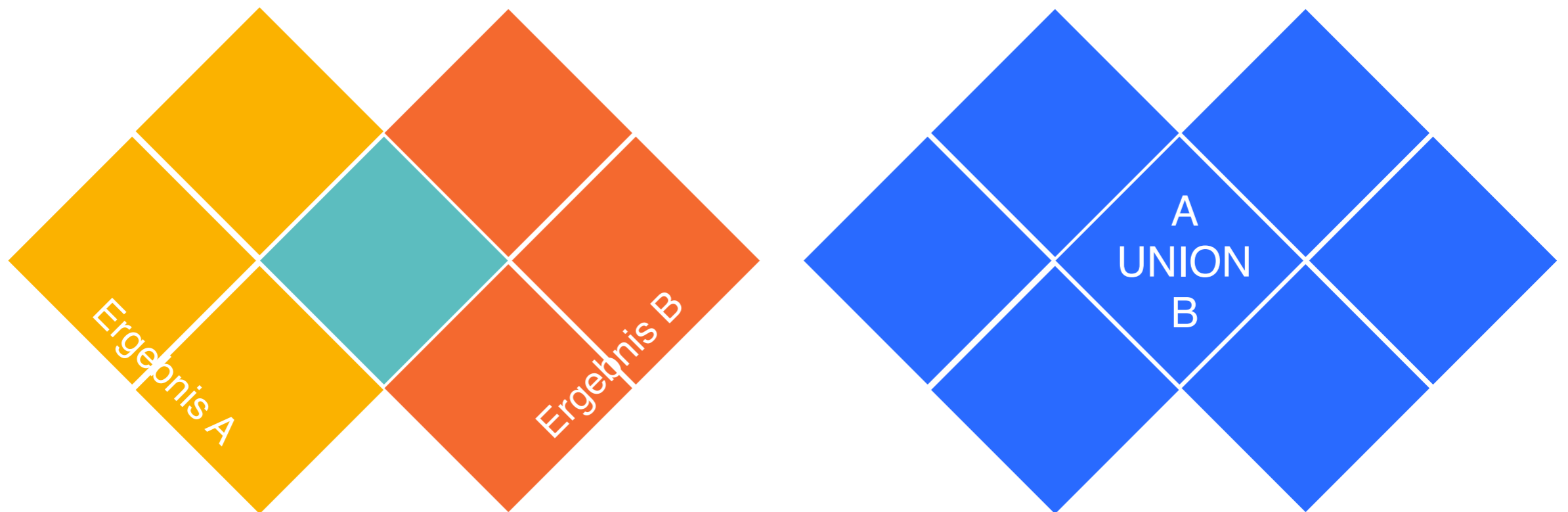
- Beispiel
 - `select name from ort`
`union`
`select name from person;`

Mengenoperationen – Randbedingungen

- Die einzelnen, mit Mengenoperatoren verknüpfte **SELECT**-Anweisungen dürfen keine **ORDER BY** Klausel enthalten (nur eine abschließende Sortierung für die Gesamtanweisung).
- Die Spaltenangaben der verknüpften **SELECT**-Anweisungen müssen in Anzahl und Datentyp übereinstimmen.
- Die Spaltennamen spielen keine Rolle.

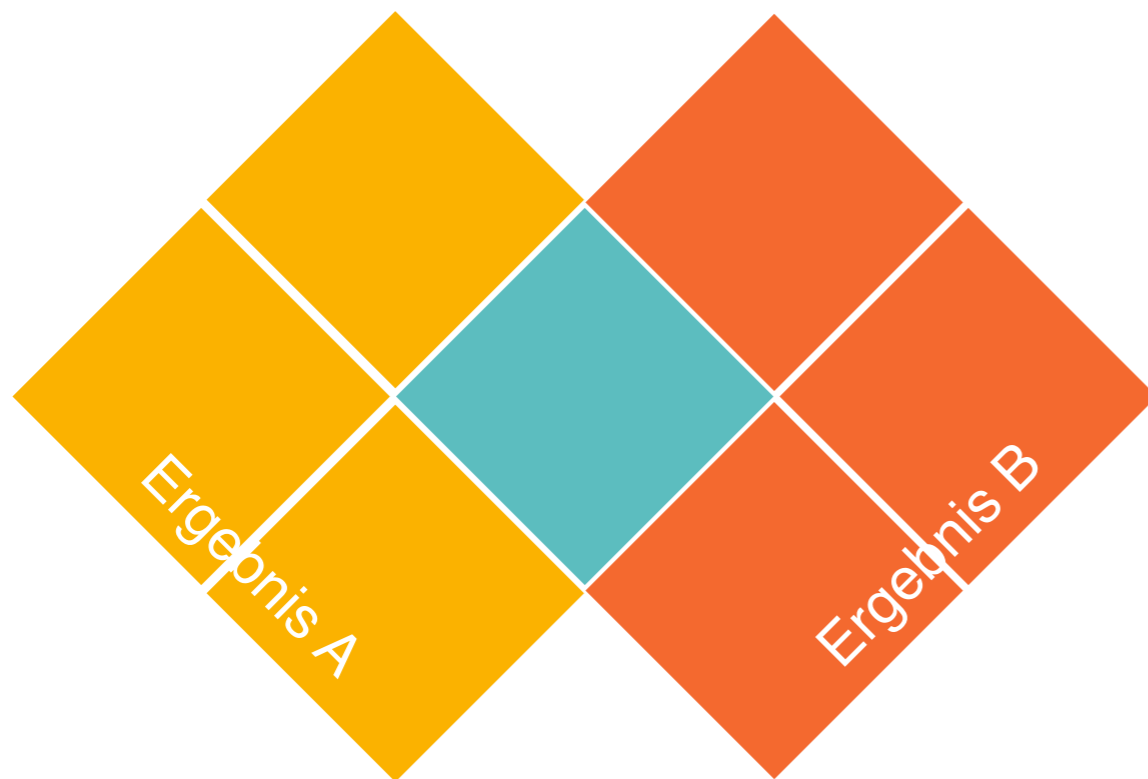
Vereinigungsmenge – UNION

- **UNION** entspricht der Vereinigungsmenge
- Doppelte Zeilen werden nur einmal ausgegeben



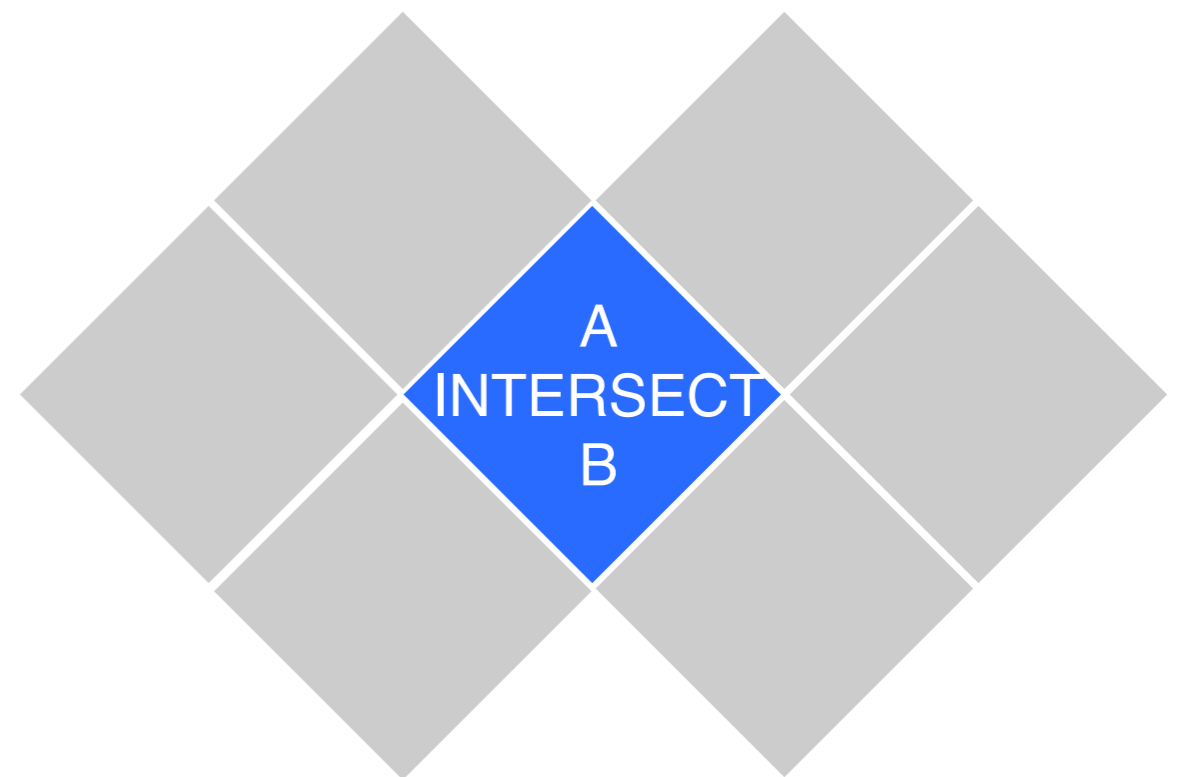
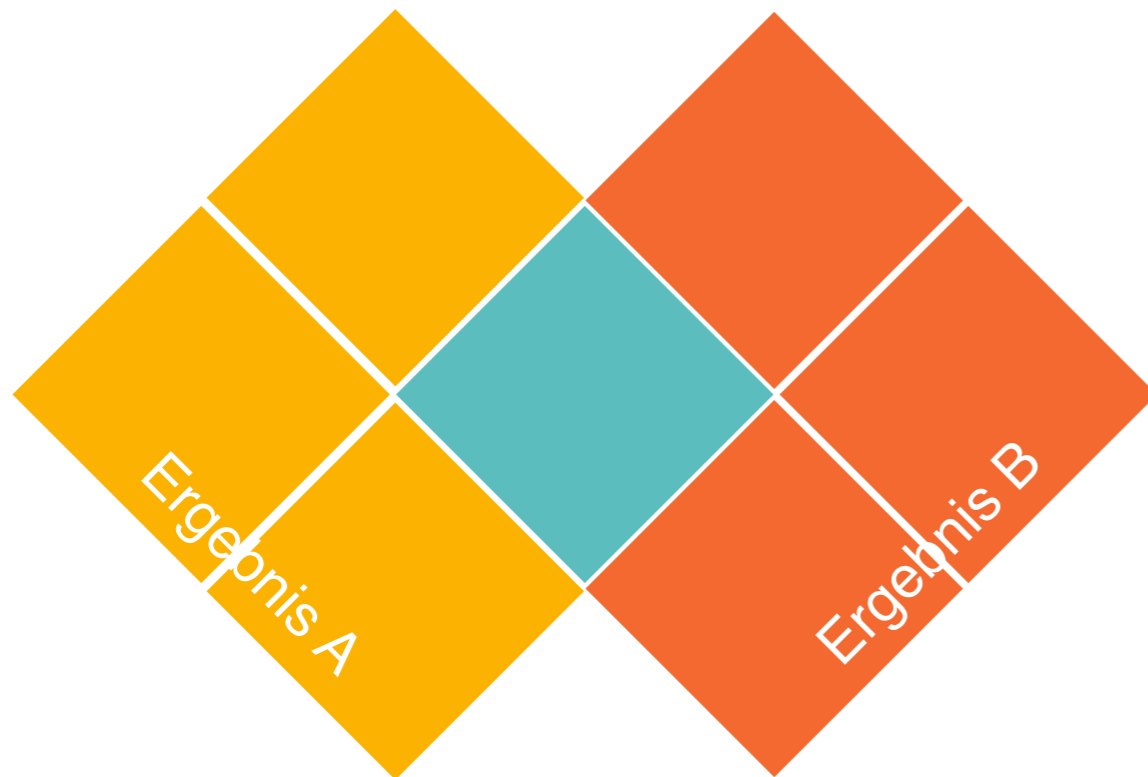
Vereinigungsmenge – UNION ALL

- Die Variante **UNION ALL** gibt mehrfach vorhandene Zeilen auch mehrfach aus



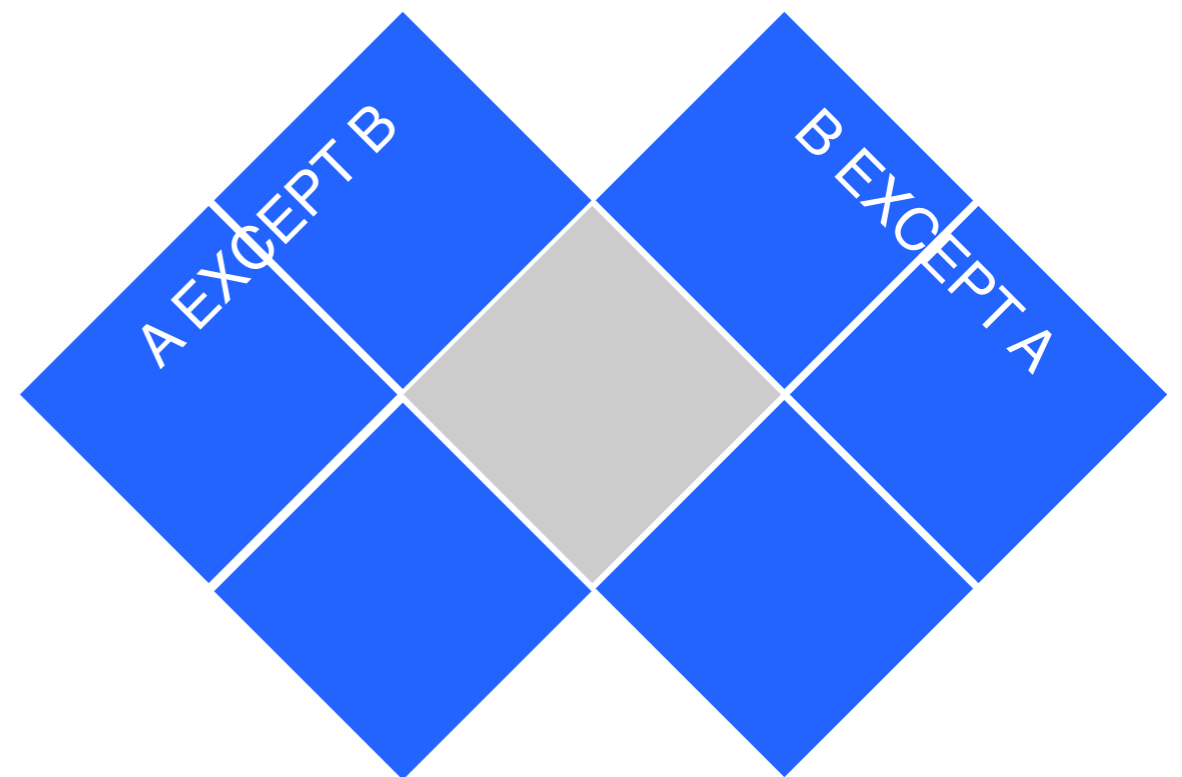
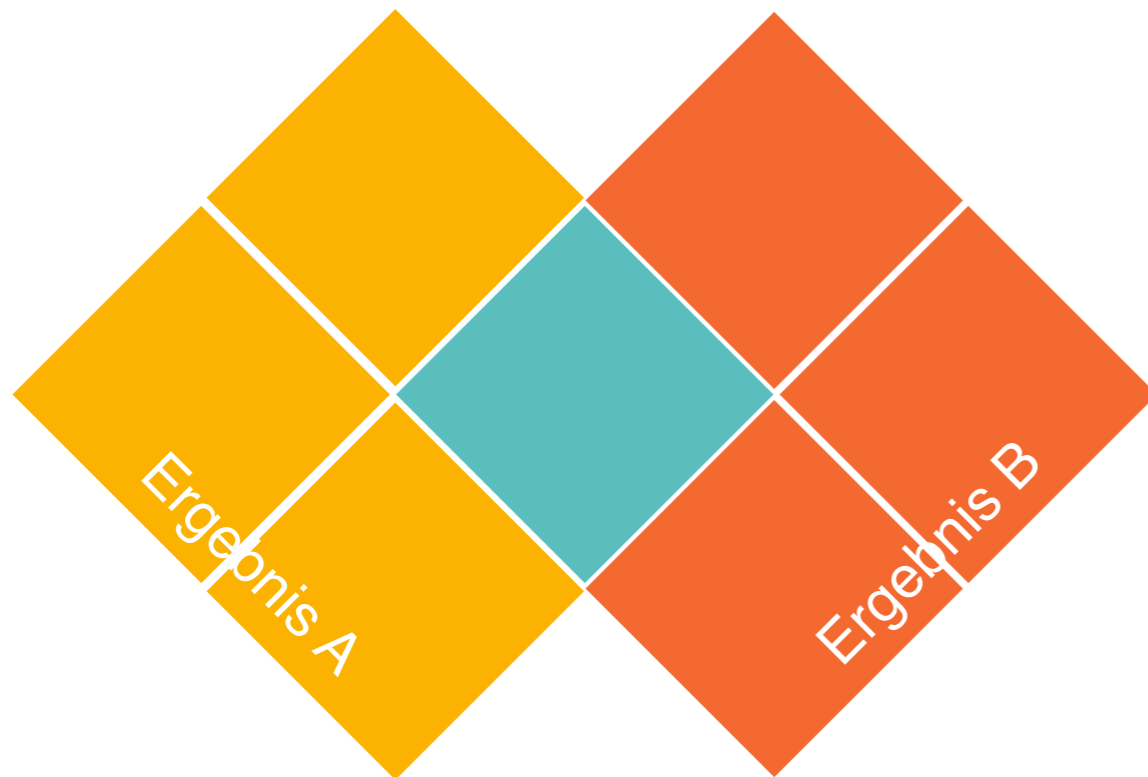
Schnittmenge – INTERSECT

- **INTERSECT** entspricht der Schnittmenge



Differenzmenge – EXCEPT

- **EXCEPT** ergibt die Differenzmenge zweier Ergebnismengen
- Die Sätze der ersten Tabelle, die nicht in der zweiten Tabelle vorkommen, werden ausgegeben.
- **EXCEPT** ist nicht kommutativ.



Mengenoperationen

- Erstelle eine Liste mit allen bekannten Ortsnamen aus den Tabellen `filiale` und `region`, aufsteigend alphabetisch nach Ortsnamen sortiert. Erstelle die Liste einmal mit Mehrfachnennung von Ortsnamen, einmal ohne.
- Ermittle unter Verwendung von Mengenoperationen, in welchen Regionshauptstädten sich eine Filiale befindet.
- Welche Orte, in denen Filialen betrieben werden, sind nicht Regionshauptstadt?
- In welchen Regionshauptstädten ist keine Filiale?

(Hinweis: Du darfst vereinfacht davon ausgehen, dass es keinen Ort gibt, der genauso heißt wie die Hauptstadt einer Region, selbst aber nicht Regionshauptstadt ist)

Daten aus mehreren Tabellen

- Oft werden Daten aus mehreren Tabellen gleichzeitig, d.h. in einer Liste benötigt

PERSNR	NAME	GEBURTSTAG	GEBURTSORT
1622	Poquelin	15.01.1622	FR00001
1694	Arouet	21.11.1694	FR00001
1564	Shakespeare	26.04.1564	EN01196
...

Person

ORT	NAME	ALT_MIN	ALT_MAX	KOORDINATEN
FR00001	Paris	28	131	N48D51ME02D21
EN01196	Stratford-upon-Avon	72	null	N52D11MW01D42
...

Ort



PERSNR	NAME	Geboren in
1622	Poquelin	Paris
1694	Arouet	Paris
1564	Shakespeare	Stratford-upon-Avon
...

Tabellenverknüpfung (Join)

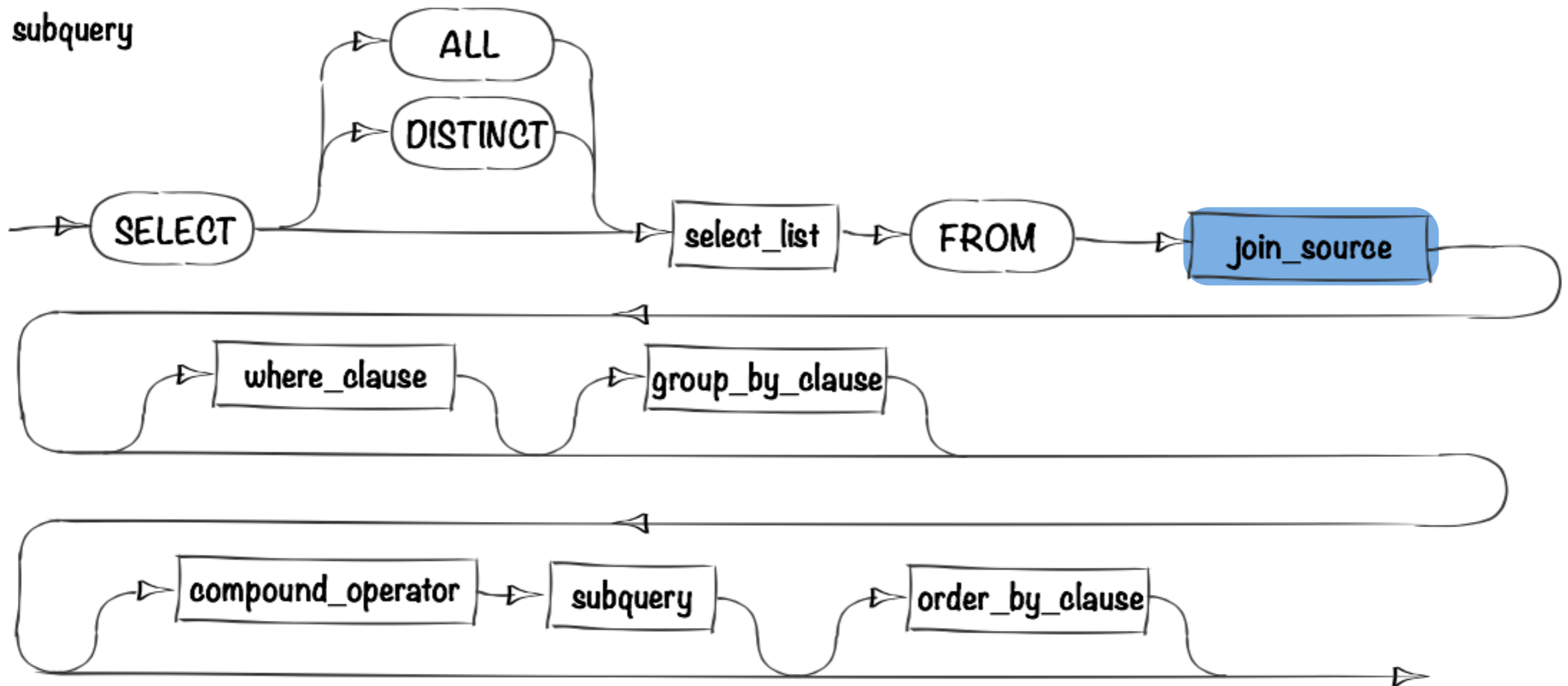
- Oft sind (z.B. durch die Normalisierung) Daten, die in einer Ergebnistabelle ausgegeben werden sollen, über mehrere Tabellen verteilt gespeichert.
- Um Daten aus mehreren Tabellen gleichzeitig abzufragen und die Spalten in einer gemeinsamen Ergebnistabelle anzuzeigen, verwendet man einen Join.
- Zeilen in einer Tabelle werden mit Zeilen aus einer anderen Tabelle über gemeinsame Werte in entsprechenden Spalten – in der Regel Primär- und Fremdschlüsselspalten – verknüpft

Tabellenverknüpfung – Namenskollision

- Wenn mehrere Tabellen in einer Anweisungen verwendet werden, kann es zu Namenskollisionen kommen.
- Der Tabellename kann vor den Spaltennamen gesetzt werden. Bei gleich benannten Spalten in unterschiedlichen Tabellen muss der Tabellename vor den Spaltennamen gesetzt werden.
- Schreibweise: Tabellename und Spaltenname werden mit einem Punkt getrennt
- Beispiel:
 - `select ort.name, person.name ...`

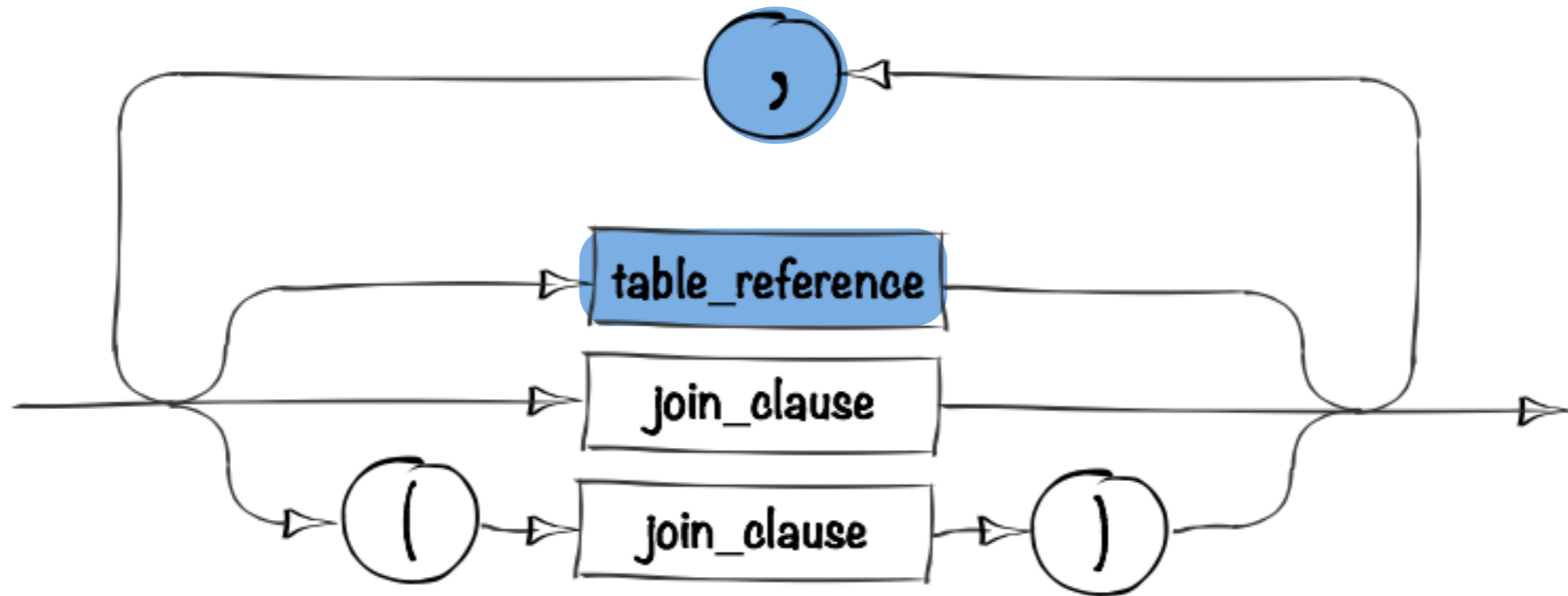
Tabellenverknüpfung (Join)

- Erweiterung der bisher bekannten Syntax



Tabellenverknüpfung (Join)

join_source



Tabellenverknüpfung – Join-Bedingung

- Die Join-Bedingung steht (bei dieser Syntaxvariante) in der **WHERE**-Klausel. Sie gibt an, welche Zeilen der Tabellen nach welchen Kriterien verknüpft werden.
- Für jede zusätzliche Tabelle in der **FROM**-Klausel ist i.d.R. mindestens eine Join-Bedingung erforderlich.
 - Für n Tabellen werden $n-1$ Join-Bedingungen benötigt
- Meist dient eine „gemeinsame“ Spalte als Join-Kriterium (Tabellenbeziehung).

Tabellenverknüpfung – Beispiel

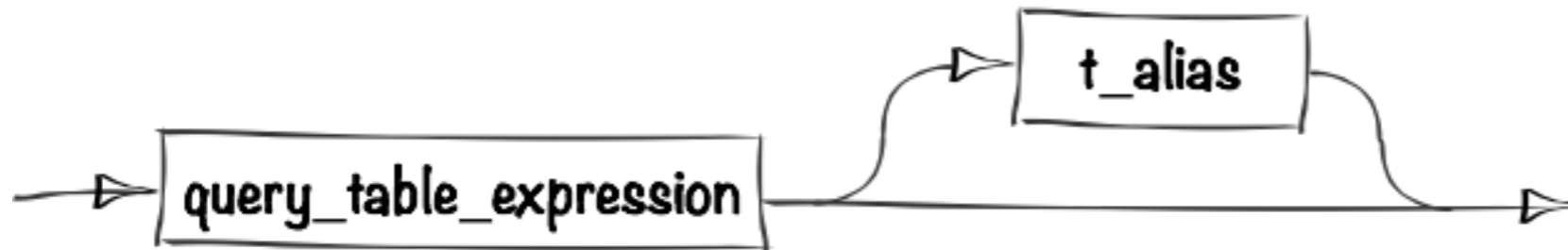
- Beispiele:
 - ```
select person.name, ort.name
from person, ort
where person.geburtsort = ort.ort; -- Join-Bedingung
```
  - ```
select *
from   werk, person, ort
where  werk.veroeffentlichung < '1700-01-01'
       and werk.sprache = 'FR'
       and werk.autor = person.persnr      -- Join-Bedingung
       and person.geburtsort = ort.ort;    -- Join-Bedingung
```

Aliasnamen in Joins

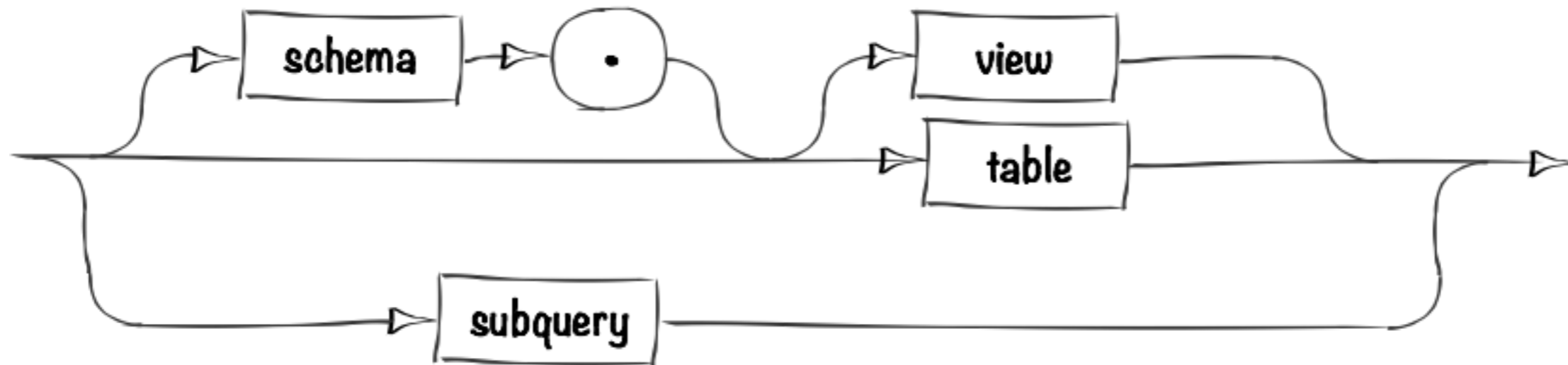
- In der **FROM**-Klausel können für die Tabellennamen Aliase vergeben werden. Diese Aliase können in den übrigen Klauseln statt der Tabellennamen verwendet werden.
- Wenn Aliasnamen vergeben werden, „ersetzen“ sie die Tabellennamen, d.h. die Aliasnamen müssen dann statt des jeweiligen Tabellennamens verwendet werden.
- Beispiel:
 - ```
select p.*, o.*
 from person p, ort o
 where p.geburtsort = o.ort;
```

# Aliasnamen – Syntax

table\_reference



query\_table\_expression



# Equi-Join

- Wenn ausnahmslos das Gleichheitszeichen als Vergleichsoperator im Join-Prädikat benutzt wird, spricht man von einem Equi-Join.
- Die Anwendung eines Equi-Joins ergibt sich in der Regel aus dem Datenmodell.
- Bei einem Equi-Join zwischen zwei Tabellen wird erzwungen, dass die Werte in der verknüpften Spalte in beiden Tabellen identisch sind.
- Equi-Joins sind oft – aber nicht zwingend – Primär- und Fremdschlüsselpaarungen.

# Kartesisches Produkt

- Das kartesische Produkt verknüpft jede Zeile der einen Tabelle mit allen Zeilen der anderen Tabelle
- Das Ergebnis ist oft nicht aussagekräftig.
- Oftmals ist das kartesische Produkt nur das unbeabsichtigte Ergebnis eines Joins, bei dem die Schlüsselverknüpfung der beteiligten Tabellen vergessen oder nicht vollständig definiert wurde.

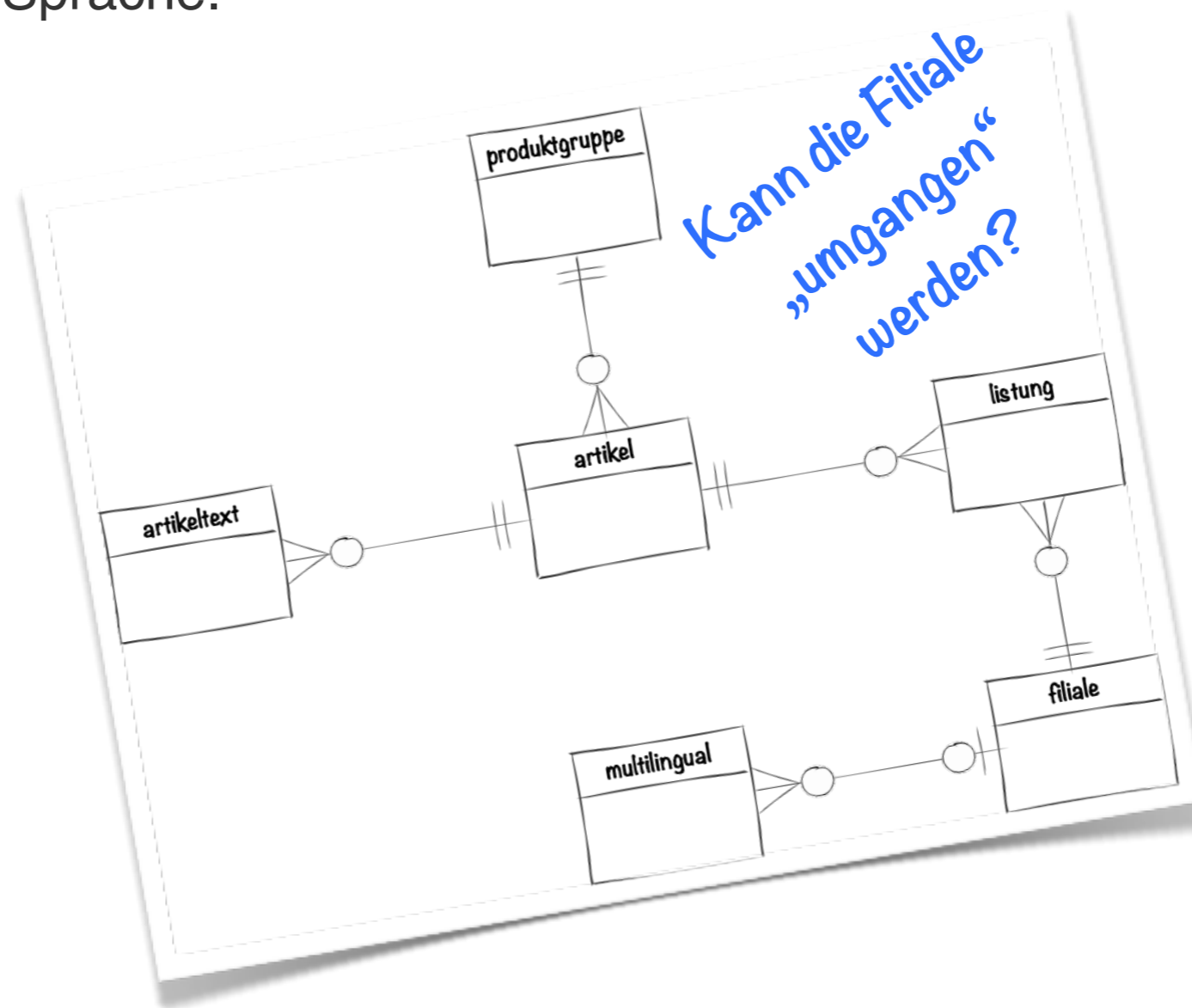


# Equi-Join

- Erzeuge eine Liste mit Angabe der Filialnummer und dem Namen der Region für die Filialen *1000*, *1660* und *1886*.
- Erzeuge eine Liste mit Angabe der Filialnummer, dem Ort und dem Namen der Region für die Filialen, die in einer Regionshauptstadt liegen.
- Wie teuer ist der teuerste Artikel aus der Produktgruppe *BNBG*, der in der Filiale *1390* verkauft wird? Gib lediglich den Verkaufspreis aus, weitere Daten sind nicht erforderlich.
- Bilde das kartesische Produkt der Tabellen **workshop** und **workshoptutor** und sieh dir den Effekt (sich wiederholende Zeile aus einer Tabelle pro Zeile der anderen Tabelle) an.
- (Fortsetzung nächste Seite...)

# Equi-Join

- Zusatzaufgabe: Gib alle relevanten Basis-Artikeltexte zur Etikettierung der *Devilsticks* in Filiale 1886 aus. Verwende hierzu die Tabellen **artikel**, **artikeltext**, **listung**, **multilingual** (für die Auswahl der verwendeten Sprachen in der Filiale) und **produktgruppe**. Sortiere die Liste nach Artikelnummer und Sprache.



# Non-Equi-Join

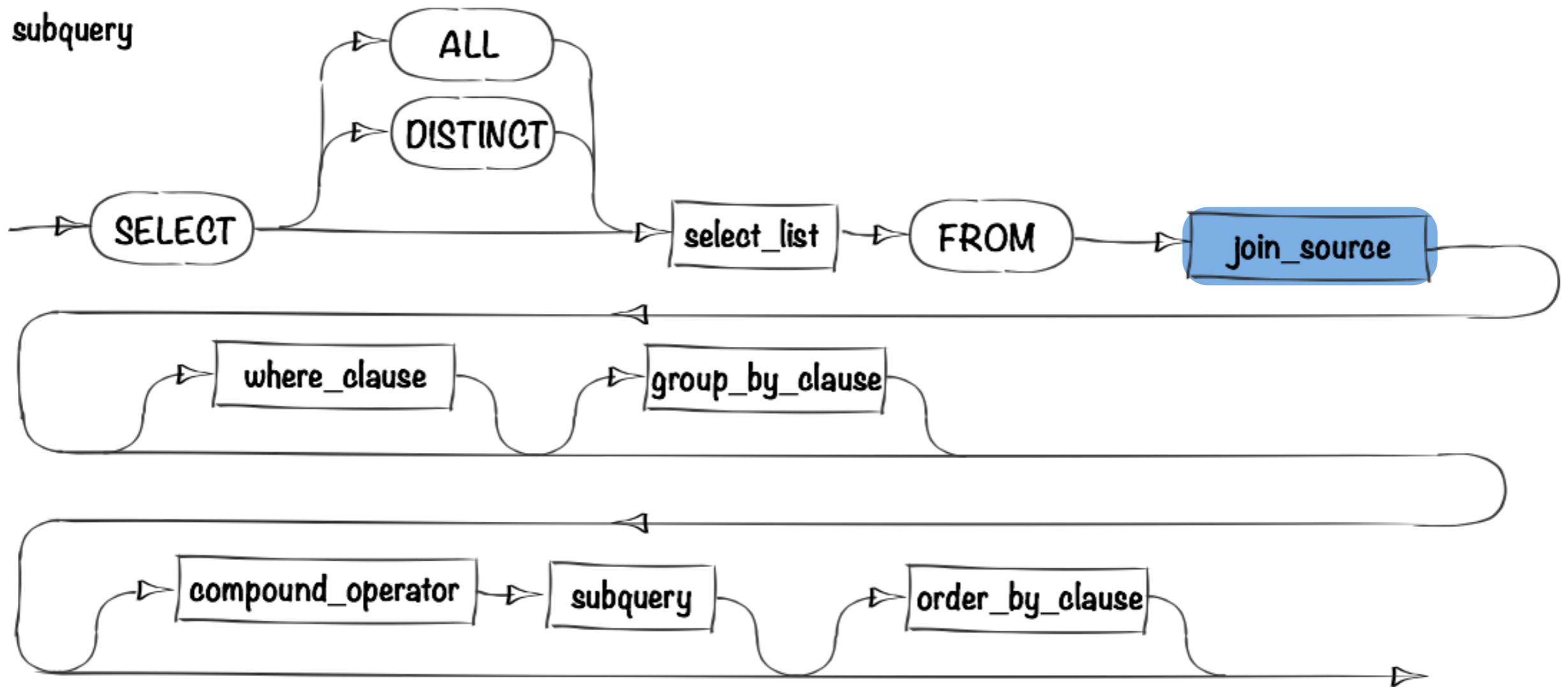
- Ein Non-Equi Join ist ein Join, bei dem der Operator der Join-Bedingung kein Gleichheitszeichen ist.
- Beispiele:
  - ```
select p.name, o.name
from   person p, ort o
where  p.geburtsort <> o.ort;
```
 - ```
select w.titel, e.bezeichnung
from werk w, epoche e
where w.veroeffentlichung
 between e.beginn and e.ende;
```

# Zweite Variante für die Join-Syntax

- In der **FROM**-Klausel wird der Join explizit formuliert.
- In der direkt folgenden **ON**-Klausel wird die Join-Bedingung formuliert.
- In der **WHERE**-Klausel sind keine Join-Bedingungen mehr erforderlich.

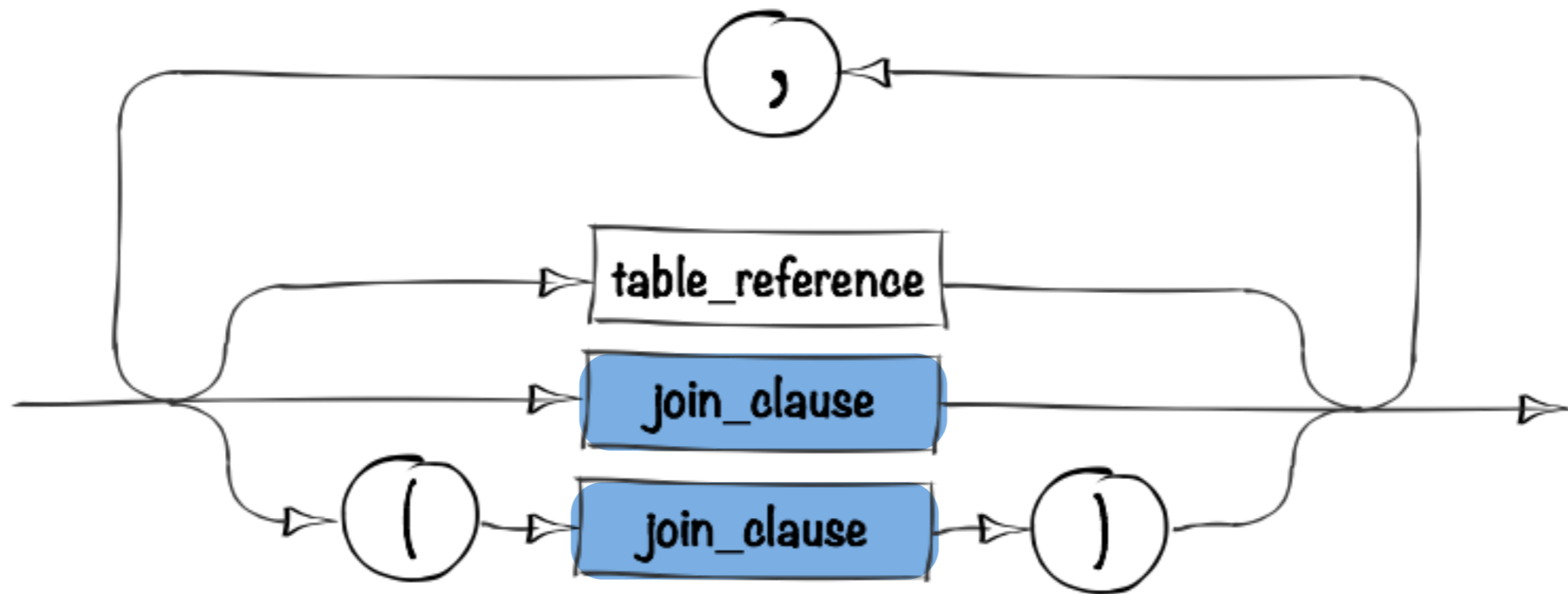
| Implizit                                                                               | Explizit                                                                                      |
|----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| <pre>select *<br/>from person p,<br/>      ort o<br/>where p.geburtsort = o.ort;</pre> | <pre>select *<br/>from person p<br/>      join ort o<br/>      on p.geburtsort = o.ort;</pre> |

# Explizite Join Syntax



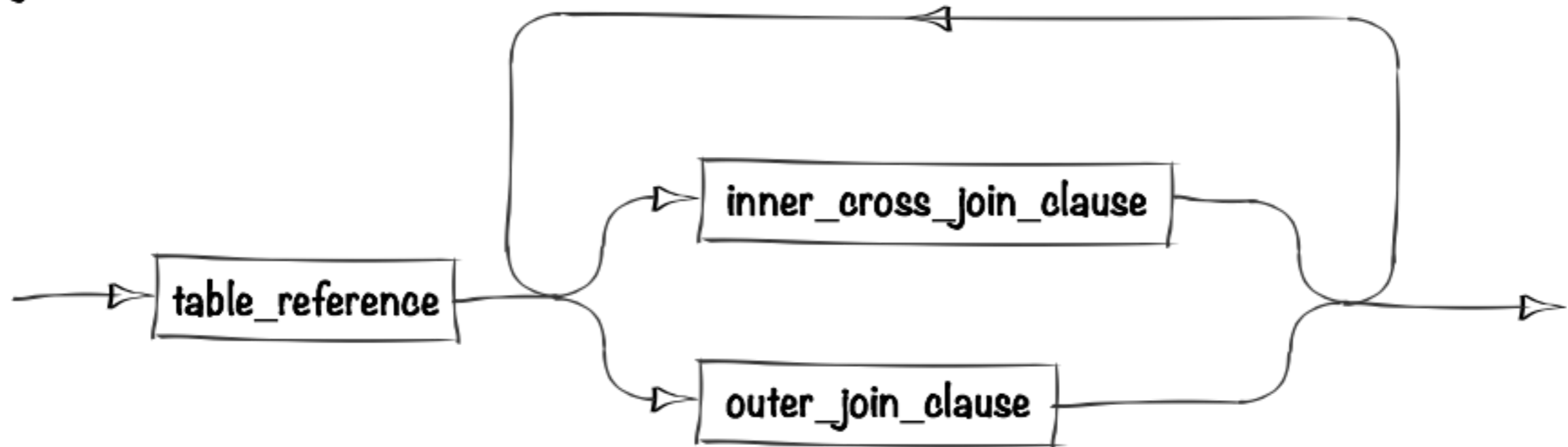
# Join Klausel

join\_source



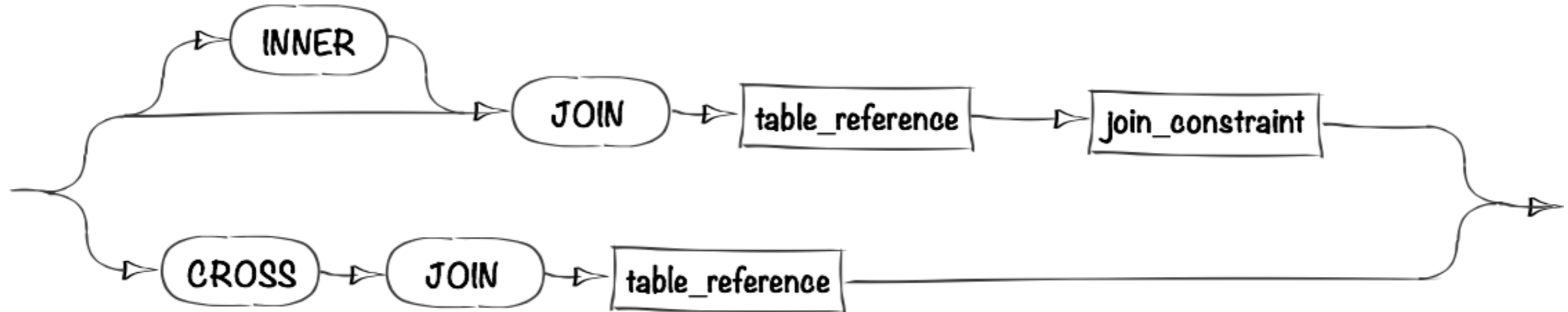
# Aufbau der Join Klausel

join\_clause

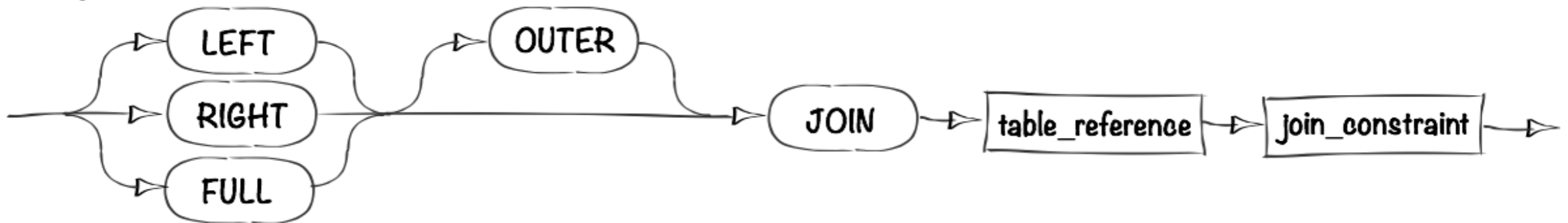


# Inner und Outer Join Klauseln

inner\_cross\_join\_clause

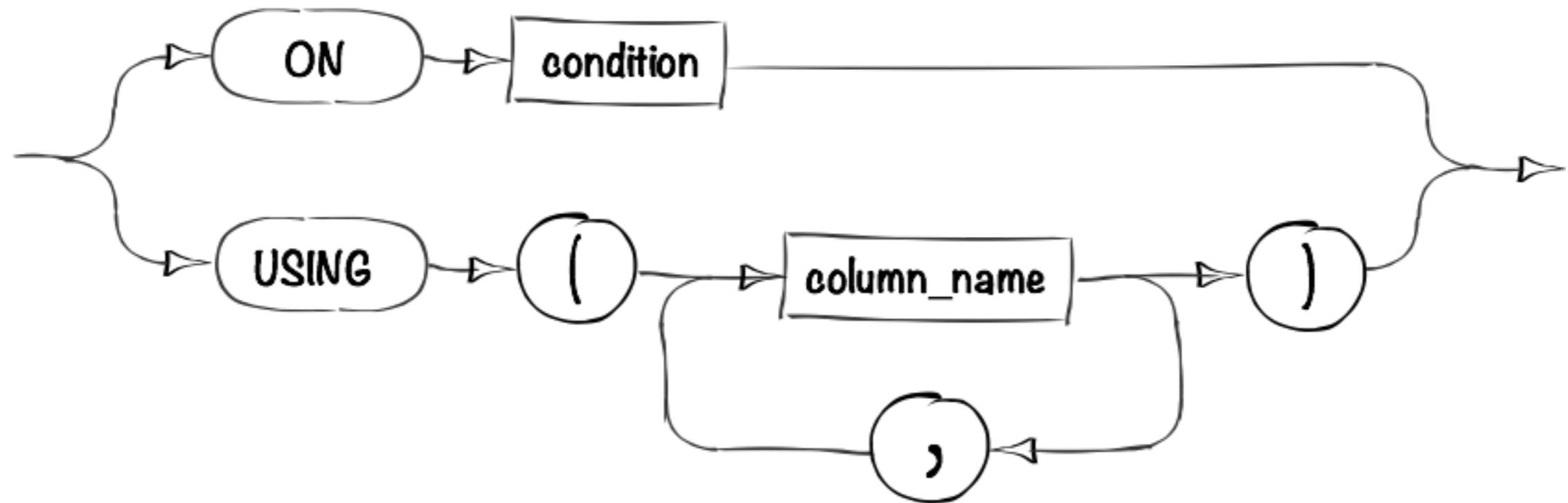


outer\_join\_clause



# Inner und Outer Join Klauseln

join\_constraint



# Inner Join vs. Outer Join

- Der innere Join (inner join) liefert nur Zeilen, die gemäß der Join-Bedingung eine Entsprechung in der Partner-Tabelle haben.
- Ein äußerer Join (outer join) liefert auch Zeilen, die kein Pendant in der anderen Tabelle haben.
- Alle Joins, die keine outer joins sind, sind inner joins (d.h. outer joins müssen als solche markiert werden).

# Inner Join

| PK A | FK B | PK B | Ergebnis A | Ergebnis B |
|------|------|------|------------|------------|
| A1   | B1   | B1   | A1         | B1         |
| A2   | B2   | B2   | A2         | B2         |
| A3   |      |      |            |            |
| A4   |      |      |            |            |

- Beispiele

- ```
select p.name, o.name
from   person p join ort o on p.wohnort = o.ort;
```

Personen ohne festen Wohnsitz sind in dieser Liste nicht enthalten.
- Inner Joins können auch explizit als inner join angegeben werden.

```
select p.name, o.name
from   person p inner join ort o on p.wohnort = o.ort;
```

Outer Join

PK A	FK B	PK B	Ergebnis A	Ergebnis B
A1	B1	B1	A1	B1
A2	B2	B2	A2	B2
A3			A3	null
A4			A4	null

- Beispiel
 - ```
select p.name, o.name
from person p left outer join ort o on p.wohnort = o.ort;
```

Alle Personen, also auch Personen ohne festen Wohnsitz sind in dieser Liste enthalten, der Ort wird als **NULL** angegeben.
  - Die Angabe **LEFT** oder **RIGHT** im Join gibt an, welche der links bzw. rechts des Schlüsselworts **JOIN** stehenden Tabellen komplett ausgegeben werden soll.

```
select p.name, o.name
from ort o right outer join person p on p.wohnort = o.ort;
```

# Outer Join

| PK A | FK B | PK B | Ergebnis A | Ergebnis B |
|------|------|------|------------|------------|
| A1   | B1   | B1   | A1         | B1         |
| A2   | B2   | B2   | A2         | B2         |
| A3   |      |      | A3         | null       |
| A4   |      |      | A4         | null       |

- Beispiel

- Da bereits durch die Schlüsselwörter **LEFT** und **RIGHT** klar ist, dass es sich um einen Outer Join handelt, kann das Schlüsselwort **OUTER** weggelassen werden.

```
select p.name, o.name
from person p left join ort o on p.wohnort = o.ort;
```

- Für Genießer: Wenn beide am Join beteiligten Tabellen mit allen Zeilen aufgeführt werden sollen, wird der **FULL OUTER** Join verwendet.

```
select p.name, o.name
from person p full join ort o on p.wohnort = o.ort;
```

# Join-Klausel mit USING

- Wenn der Join über eine Spalte erfolgt, die in beiden Tabellen gleich heißt, kann die Join-Bedingung mit **USING** formuliert werden.
- Beispiel:
  - Personen, die so heißen wie ein Ort  
`select * from person join ort using (name) ;`

# Cross-Join – Kartesisches Produkt

- Um explizit ein kartesisches Produkt zu erzeugen, wird es mit **CROSS JOIN** formuliert.
- Wegen der Natur des kartesischen Produkts, ist hierbei keine **ON** oder **USING** Klausel erforderlich.
- Beispiel:
  - Wie viele Zeilen hat das kartesische Produkt der Personen und Werke  
`select count(*) from person cross join werk;`

# Natural Join

- Ein Join über gleich benannte Spalten zweier Tabellen kann auch direkt mit einem **NATURAL JOIN** erfolgen.
- Als Join-Bedingungen werden alle Spalten herangezogen, die in beiden Tabellen jeweils gleich heißen...
- ...auch wenn dies eher zufällig der Fall ist!
- Beispiel:
  - Personen, die so heißen wie ein Ort  
`select * from person natural join ort;`

# Self-Join

- Beim Self-join wird eine Tabelle mit sich selbst verknüpft.
- In der **FROM**-Klausel wird die Tabelle dazu mehrfach aufgeführt und jeweils unterschiedliche Alias-Namen vergeben.
- Beispiel
  - ```
select 'Die Mutter von '
      || kind.vorname || ' ' || kind.name
      || ' ist '
      || mutter.vorname || ' ' || mutter.name
from   person kind, person mutter
where  kind.mutter_persnr = mutter.persnr;
```

Verschiedene Joins

- Erstelle eine Liste der geplanten Workshops mit Angabe des Kürzels des Tutors.
- Erstelle eine Liste aller Workshops, die neben dem Workshoptitel, sofern bereits ein Tutor benannt ist, das Tutorenkürzel enthält.
- Erweitere die eben erstellte Anweisung so, dass statt des Tutorenkürzels der Vor- und Nachname des Tutors ausgegeben wird.
- Formuliere die Abfrage so, dass alle Workshops und alle Tutoren ausgegeben werden.
- Wie viele Zeilen enthält das kartesische Produkt der Tabellen **workshop** und **workshoptutor**?
- Gib zu allen Artikeln aus der Produktgruppe *BNBG* eine Liste aus, in der die Artikelnummer mit Bezeichnung des Artikels zusammen mit der Artikelnummer und Bezeichnung eines eventuellen Hauptartikels aufgeführt ist.
- (Fortsetzung nächste Seite...)

Verschiedene Joins



- Zusatzaufgabe: Erstelle eine Preisliste mit den Angaben Artikelnummer, Filialnummer, relevanter Verkaufspreis und die Art des Verkaufspreises (Sonderpreis oder Normalpreis) für die Artikel *114800* und *114900* in den Filialen *1390*, *1690* und *1886* zum Termin *27. Mai des Vorjahres* (darf als Literal angegeben werden).
- Der relevante Verkaufspreis zu einem Stichdatum ergibt sich wie folgt: Wenn für den Artikel und die Filiale ein Eintrag in der Tabelle **sonderpreis** angelegt ist, mit Stichdatum zwischen **gilt_ab** und **gilt_bis** (Intervallgrenzen inklusive), dann gilt der Sonderpreis, sonst der Normalpreis aus der Tabelle **verkaufspreis**.
- Du kannst über die Tabelle **listung** einsteigen. Artikel- oder filialspezifische Daten werden nicht benötigt.
- Das **gilt_ab** Datum in den Tabellen **listung** und **verkaufspreis** sowie **gilt_bis** in der Tabelle **listung** darfst du für diese Aufgabe ignorieren.
- Achte darauf, nur wirklich relevante Sonderpreise (Gültigkeitszeitraum beachten) zu verwenden.
- Du kannst mit **coalesce** und **case when** arbeiten, um Normal- und Sonderpreise zu differenzieren.

Übungsaufgabe

Unterabfragen – Subselects

- Eine Unterabfrage (subselect) ist eine **SELECT**-Anweisung, die in eine Klausel einer anderen SQL-Anweisung eingebettet ist.
- Die Verwendung einer verschachtelten Unterabfrage entspricht der Ausführung von zwei Abfragen:
 - Die Unterabfrage (innere Abfrage) wird einmal vor der Hauptabfrage (äußere Abfrage) ausgeführt.
 - Die Hauptabfrage verwendet danach die Ergebnismenge der Unterabfrage.
- Die innere und äußere Abfrage können Daten aus der gleichen Tabelle oder aus verschiedenen Tabellen abrufen.

Subselects mit single-row Operator

- Bei Subselects mit single-row-Operatoren muss die Ergebnismenge des Subselects exakt eine Zeile sein.
- Beispiele:
 - Personen, die nach einer bestimmten Epoche geboren sind

```
select vorname, name from person
where geburtsdatum > (select ende from epoche
                        where bezeichnung = 'Romantik');
```
 - Personen, die in einer bestimmten Epoche geboren sind

```
select vorname, name from person
where geburtsdatum between
      (select beginn from epoche
       where bezeichnung = 'Barock')
and (select ende from epoche
     where bezeichnung = 'Barock');
```

Subselects mit multiple-row Operator

- Bei Subselects mit multiple-row-Operatoren werden Ergebnismengen der Subselects betrachtet. Diese können aus keiner, einer oder mehreren Zeilen bestehen.
- Multiple-row Operatoren: **IN, ANY, ALL, SOME**
- Beispiel
 - Alle Personen, die in einem Ort oberhalb 1000m geboren sind

```
select name from person
where geburtsort in (
    select ort from ort
    where alt_min > 1000);
```

Korrelierte Subselects

- Korrelierte Subselects werden für die zeilenweise Verarbeitung verwendet.
- Der Subselect liefert pro Zeile der äußeren Abfrage ein individuelles Ergebnis.
- Beispiel
 - Werke, die der Autor vor seinem 20. Lebensjahr veröffentlichte

```
select *  
from   werk aussen  
where  veroeffentlichung < (  
        select date_add(geburtsdatum, interval 20 year)  
        from   person innen  
        where  innen.persnr = aussen.autor);
```

EXISTS

- Der **EXISTS**-Operator prüft, ob Zeilen in der Ergebnismenge der Unterabfrage enthalten sind.
- Der Suchvorgang in der inneren Abfrage beschränkt sich auf die Feststellung der Existenz, die tatsächlichen Daten werden nicht zurückgegeben. Die Rückgabe des **EXISTS**-Operators ist ein boolescher Wert.
- Der **EXISTS**-Operator kann mit **NOT** logisch verneint werden (... **WHERE NOT EXISTS** ...).
- Bei **EXISTS** wird oft die innere mit der äußeren Suchanfrage verbunden (korreliert).

EXISTS

- Beispiel
 - Orte, in denen bisher niemand an einem 1. Januar geboren wurde

```
select name, koordinaten
from   ort o
where not exists (
    select 1
    from   person p
    where p.geburtsort = o.ort
          and extract(day from p.geburtsdatum) = 1
          and extract(month from p.geburtsdatum) = 1);
```

Subselects in der Spaltenliste

- Eine Unterabfrage kann u.a. auch in der Spaltenliste verwendet werden.
- Sinnvoll bei einer Verwendung in der Spaltenliste sind nur Ergebnismengen mit einer Zeile.
- Prinzip
 - `select spalten, (select spalten from ...)
from ...
where ...`

Subselects

- Erstelle für die folgenden Aufgaben jeweils mindestens eine Lösung mit einem Subselect. Alternativlösungen mit Mengenoperationen oder Joins können zum Vergleich zusätzlich geschrieben werden.
- Welche Filialen sind in einer Regionshauptstadt?
- In welchen Regionen sind keine Filialen?
- Welche Tutoren sind für einen Workshop eingetragen?