

SQL

Grundlagen

Einfache SQL-Abfragen

ROLLBACK
COMMIT
RDBMS
DELETE
VALUES
DATABASE
DEFAULT
CREATE
LEFT OUTER JOIN
INSERT
OUTER
SAVEPOINT
TABLE
JOIN
INNER
VIEW
SELECT
EXTRACT
TRIGGER
DML
TRANSACTION
NUMERIC
ISOLATION LEVEL
DATA DEFINITION LANGUAGE
DEADLOCK
EXPLAIN PLAN
DROP
RIGHT
JOIN

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)
- Für die Übungsdatenbank: Mac/PC mit aktueller Java-Version

Übersicht

- Was deckt SQL ab?
- Kurzeinführung in Syntaxbeschreibungen
- Beispiele für SQL-Abfragen
- Datentypen, der Leerwert **NULL**, bedingte Auswahl
- Sortierung
- Rechnen im **SELECT** und einfache Funktionen
- Datumsangaben (der Datentyp **DATE**)
- Gruppierung und Aggregatfunktionen

SQL – Structured Query Language

- SQL ermöglicht es, Daten in einer relationalen Datenbank definieren, manipulieren und kontrollieren zu können (C.J. Date)
- Der Standard schreibt vor, dass man mit den Befehlen wie **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **CREATE** und **DROP** fast alle Aufgaben, die mit der Datenbank zusammenhängen, durchführen kann
- Beispiele für RDBMS: Oracle, DB2, MySQL, MS SQL Server, PostgreSQL, u.v.a.

Übersicht Befehlsgruppen

- Datendefinition (DDL)
- Datenmanipulation (DML)
- Datenkontrolle

Übersicht Befehlsgruppen

- Datendefinition (DDL)
 - Tabellen anlegen (**CREATE**)
 - Tabellen ändern (**ALTER**)
 - Tabellen löschen (**DROP**)
- Datenmanipulation (DML)
- Datenkontrolle

Übersicht Befehlsgruppen

- Datendefinition (DDL)
- Datenmanipulation (DML)
 - Daten abfragen (**SELECT**)
 - Daten einfügen (**INSERT**)
 - Daten aktualisieren (**UPDATE**)
 - Daten löschen (**DELETE**)
- Datenkontrolle

Übersicht Befehlsgruppen

- Datendefinition (DDL)
- Datenmanipulation (DML)
- Datenkontrolle
 - Datensicherheit I: Sicherbarkeit und gleichzeitiger Zugriff (**COMMIT** / **ROLLBACK**)
 - Datensicherheit II: Zugriffskontrolle (**GRANT** / **REVOKE** und Views)

SELECT – Anweisung

- Mit der **SELECT** Anweisung werden Daten aus der Datenbank angezeigt.
- Dabei wird angegeben, aus welcher Tabelle die Daten stammen, welche Spalten und welche Zeilen ausgegeben werden sollen.
- Die Ergebnisliste kann sortiert ausgegeben werden.

SELECT – Beispiele

- Auswahl aller Spalten und Zeilen einer Tabelle
`select * from person;`
- Einzelne Spalten
`select titel from werk;`
- Mehrere Spalten
`select vorname, name, geburtsdatum
from person;`

SELECT – Beispiele (2)

- Auswahl aller Spalten und Zeilen einer Tabelle

```
select * from person;
```

- Einzelne Spalten

```
select titel from werk;
```

- Mehrere Spalten

```
select vorname, name, geburtsdatum  
from person;
```

select
Daten anzeigen

SELECT – Beispiele (3)

- Auswahl aller Spalten und Zeilen einer Tabelle

```
select * from person;
```

- Einzelne Spalten

```
select titel from werk;
```

- Mehrere Spalten

```
select vorname, name, geburtsdatum  
from person;
```

* oder Spaltenname(n)
welche Spalten

SELECT – Beispiele (4)

- Auswahl aller Spalten und Zeilen einer Tabelle

```
select * from person;
```

- Einzelne Spalten

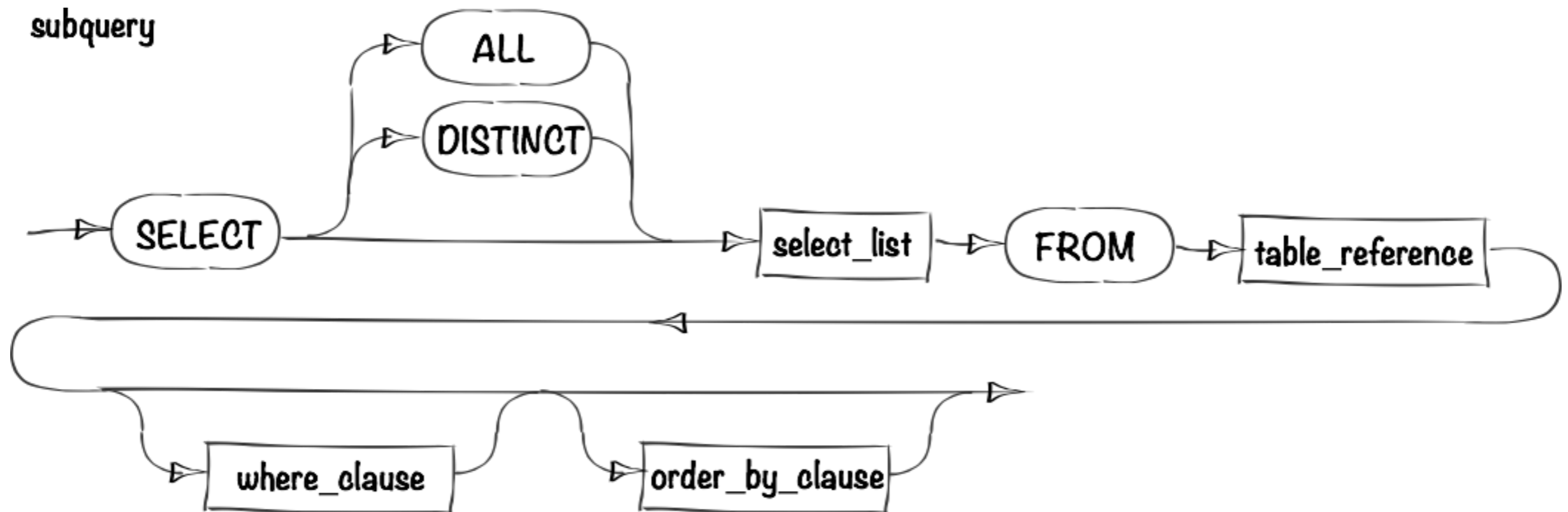
```
select titel from werk;
```

- Mehrere Spalten

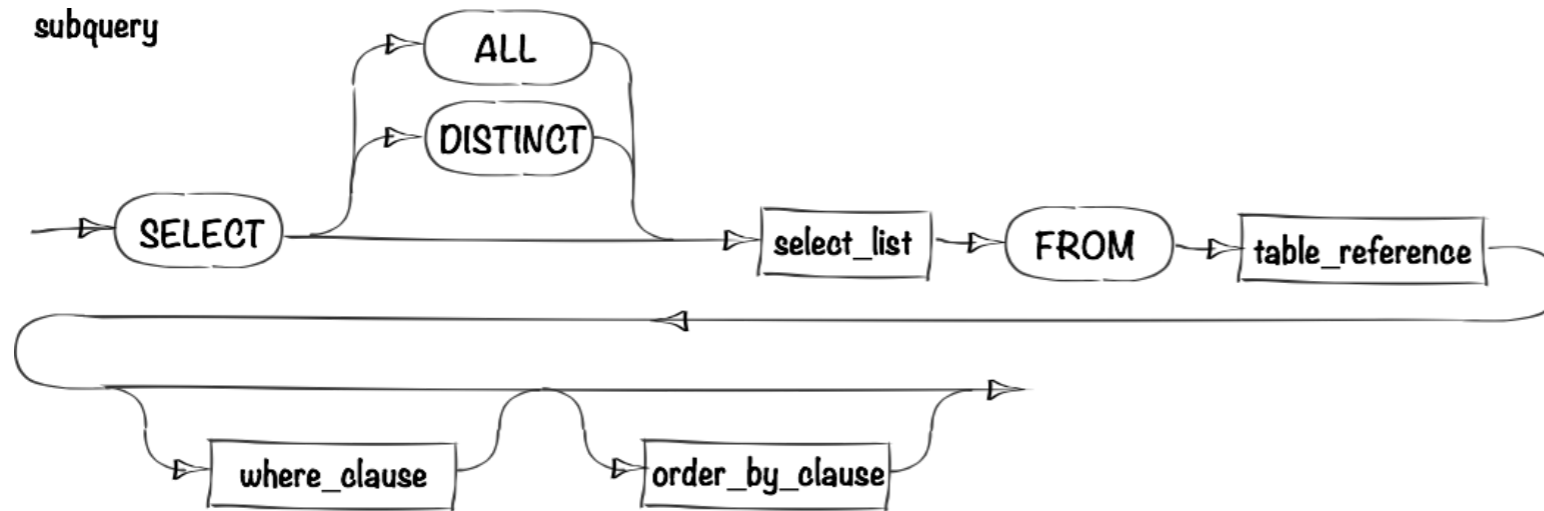
```
select vorname, name, geburtsdatum  
from person;
```

from tabellenname
woher stammen die Daten

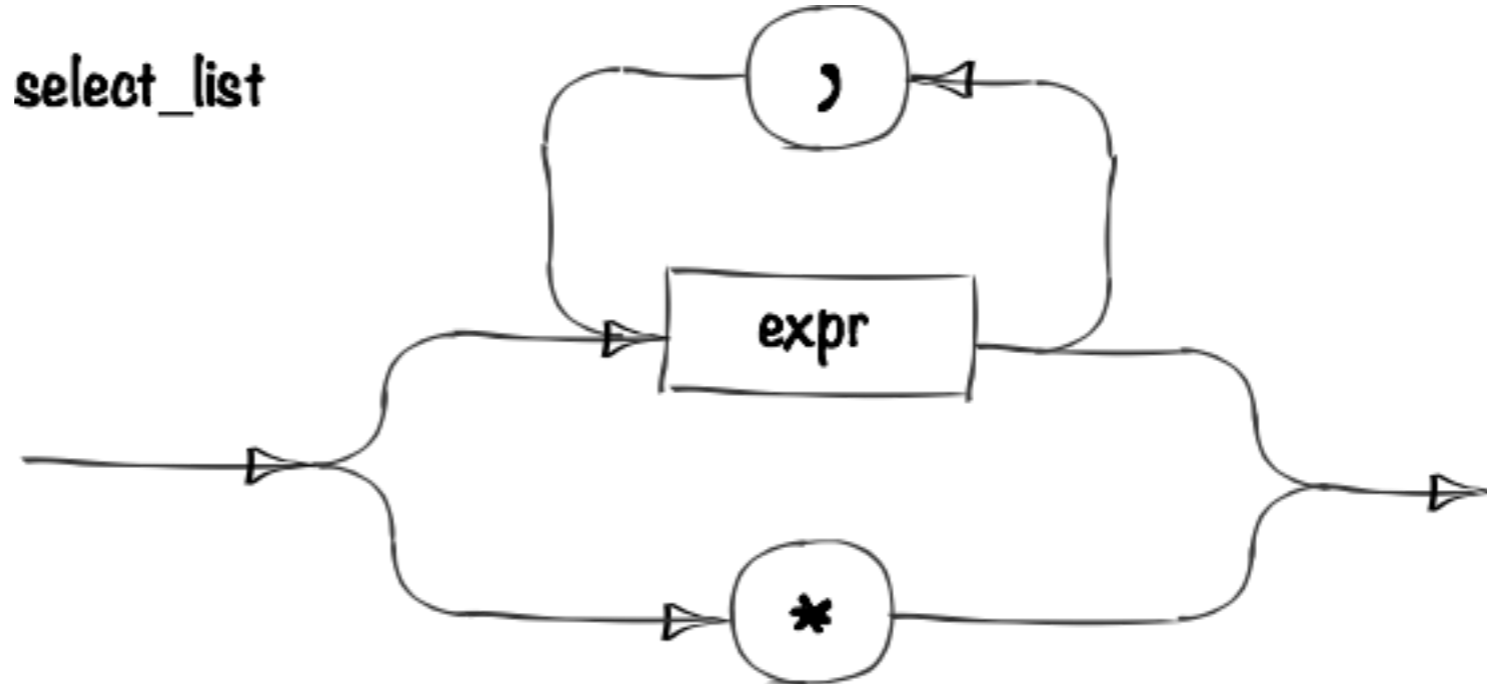
SELECT – Verallgemeinerte Syntax



SELECT – Verallgemeinerte Syntax (2)

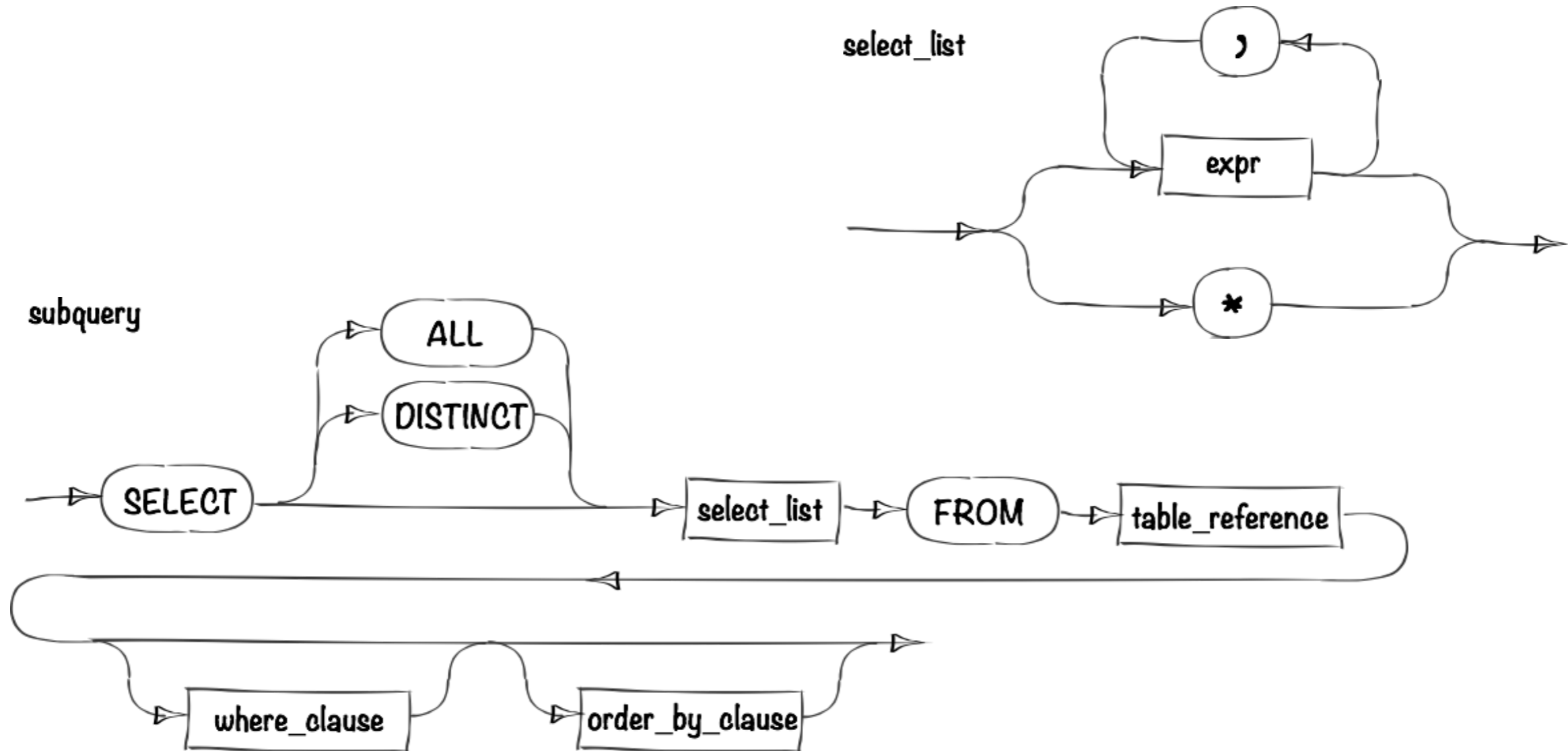


select_list



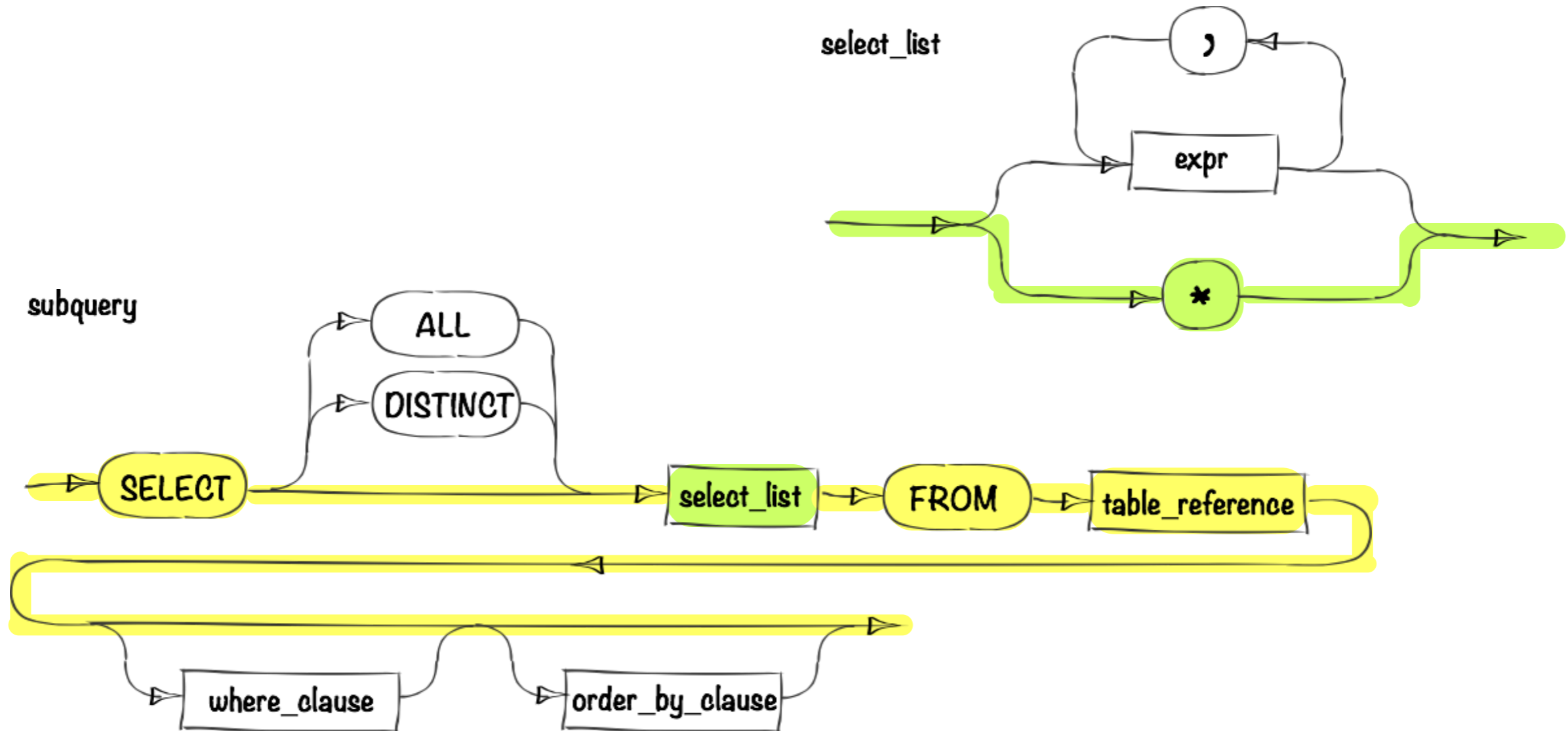
SELECT – Verallgemeinerte Syntax (3)

- *table_reference* wird später betrachtet – für den Anfang ist dies einfach ein Tabellename



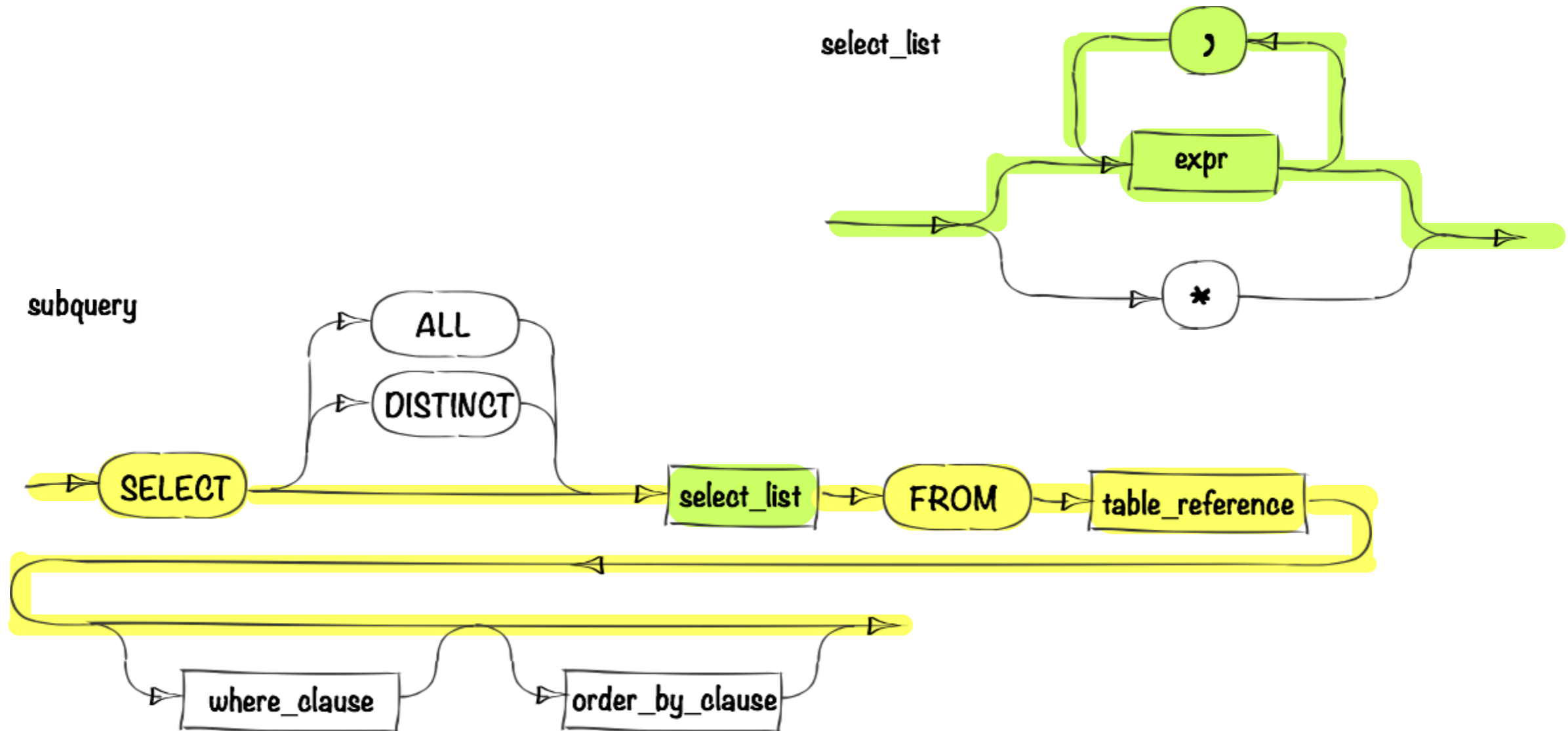
SELECT – Syntaxbeispiel

- `select * from person;`



SELECT – Syntaxbeispiel (2)

- `select vorname, name from person;`



SELECT – Beispiele erweitert

- Einmalige Anzeige von Werten (Ausprägungskatalog)
`select distinct name from person;`

SELECT – Zusatzbemerkungen

- Schlüsselwörter können beliebig in Groß- oder Kleinschreibung (oder gemischt) angegeben werden
- SQL-Anweisungen werden mit Semikolon abgeschlossen
 - In den meisten SQL Clients können einzelne Anweisungen auch ohne abschließendes Semikolon ausgeführt werden

Einfache SELECT-Anweisungen

- Selektiere in drei einzelnen Abfragen jeweils alle Zeilen und alle Spalten der Tabellen **region**, **filiale** und **produktgruppe**.
- Selektiere alle Zeilen der Tabelle **filiale**, hierbei jedoch nur die Filialnummer und den Ort.
- Erzeuge eine Liste der Hauptstädte aller Verkaufsregionen (in der Tabelle **region** abgebildet).
- Ermittle alle *unterschiedlichen* Textarten aus der Tabelle **artikeltext**.
- Erstelle eine Auflistung aller unterschiedlichen Regionen, in denen es eine Filiale gibt.

Eine weitere Metasyntax

- Vereinfachte Backus-Naur Notation:
 - `[]` Optionale Elemente
 - `{ }` Von den aufgeführten Elementen ist nur eines erforderlich
 - `|` Trennt Alternativen
 - `...` Das vorhergehende Element kann wiederholt werden
 - `Trennzeichen` Trennzeichen (außer `[]` `|` `{ }`) sind wie angegeben zu verwenden
 - **fett** Schlüsselwörter sind fett gedruckt. SQL akzeptiert bei Schlüsselwörtern Groß- und Kleinschreibung.
 - `normal` Platzhalter, die ersetzt werden

Erläuterung zu Syntaxdefinitionen

- Beispiel: Syntaxdefinition einer Telefonnachricht
 - Die “korrekte” Schreibweise von Vor- und Nachnamen wird vorausgesetzt
 - Namen dürfen entweder als “Vorname Nachname” oder als “Nachname, Vorname” geschrieben werden. Personen können mehrere Vornamen haben.

informell ::= vorname [vorname]... nachname

formell ::= nachname , vorname [vorname]...

name ::= { formell | informell }

nachricht ::= name hat angerufen.

Syntaxdefinition subquery – die Zweite

subquery ::=

```
SELECT [ { DISTINCT | UNIQUE } | ALL ] select_list  
FROM table_reference  
[ where_clause ]  
[ order_by_clause ]
```

Formatierung von SQL-Anweisungen

- SQL-Anweisungen können weitgehend frei formatiert werden
 - d.h. Leerzeichen, Zeilenumbrüche, Tabulatoren können beliebig zwischen den einzelnen Syntaxelementen eingefügt werden
- SQL-Anweisungen können Kommentare enthalten
 - Kommentare sind für den Entwickler oder Benutzer der SQL-Anweisung gedacht, sie werden nicht ausgeführt
 - Kommentare werden wie folgt markiert:
 - -- leitet Kommentare bis zum Ende der Zeile ein
 - Mehrzeilige Kommentare oder Kommentare innerhalb einer Zeile werden in /* und */ eingeschlossen

Formatierung von SQL-Anweisungen

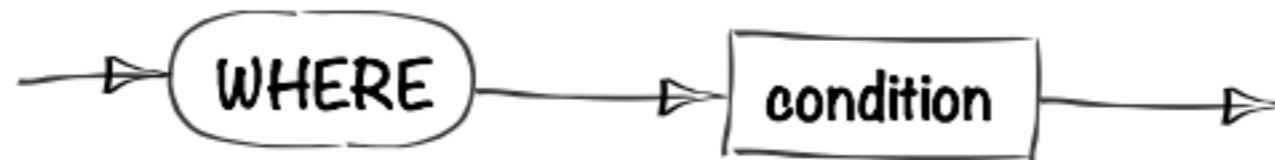
- Beispiele für Formatierungen und Kommentare

```
SELECT * -- Kommentar bis Zeilenende
FROM    PERSON
WHERE   GEBURTSORT = /* ... */ 'FR00001'
/*
    Mehrzeiliger Kommentar
*/
ORDER
    BY VORNAME DESC;
```

Bedingte Auswahl

- Eine SQL-Anweisung muss nicht immer alle Zeilen einer Tabelle als Ergebnismenge liefern
- Mit der **WHERE**-Klausel wird eingeschränkt, welche Zeilen in der Ergebnismenge enthalten sind

where_clause



Bedingte Auswahl

- Eine **WHERE**-Klausel beschreibt eine Bedingung
- Die und nur die Datensätze, bei denen die Bedingung erfüllt ist, sind in der Ergebnismenge enthalten
- Beispiele für Bedingungen (nicht-technisch formuliert)
 - Farbe ist rot
 - Alter über 18
 - Gehalt über 5.000,- Euro

Vergleichsoperatoren

- Ausdrücke in Bedingungen werden mit Vergleichsoperatoren verglichen

Operator	Bedeutung	Beispiel
=	gleich, entspricht	geburtsjahr = 1935
≠	ungleich	geburtsjahr ≠ 1994
!=	ungleich (andere Schreibweise)	geburtsjahr != 1994
<	kleiner als	anzahl < 11
<=	kleiner oder gleich	anzahl <= 10
>	größer als	gehalt > 4999.99
>=	größer oder gleich	gehalt >= 5000

Datentypen

- Übersicht häufig verwendeter Datentypen

Anforderung	Datentyp	Beschreibung
Text, alphanumerische Daten	CHAR (<i>size</i>)	Zeichendaten fester Länge (mit Leerzeichen aufgefüllt)
	VARCHAR (<i>size</i>)	Zeichendaten variabler Länge
Zahlen, numerische Daten	NUMERIC NUMERIC (<i>l</i> , <i>n</i>)	Numerische Daten (Angabe der Länge und Genauigkeit, d.h. Nachkommastellen optional)
	INTEGER	Ganzzahlen
	DECIMAL	Gleitkommazahlen
Datums- und Zeitangaben	DATE	Datums- und Zeitangaben
	TIME , TIMESTAMP	Zeitangaben, Zeitstempel
Binärdaten (z.B. Bilder)	BINARY , OTHER , OBJECT	

Datentypen

- Bei Vergleichen müssen die Werte auf beiden Seiten des Vergleichsoperators einen verträglichen Datentyp haben
 - d.h. Vergleich von Text mit Text, Zahl mit Zahl, Datum mit Datum
- Der Vergleich von Zeichenketten folgt dem Schema
 - $a < ab < b$, d.h bei gleichem Anfang ist der kürzere Text “kleiner”
 - $0..9 < A..Z < a..z$
 - Sonderzeichen wie z.B. Umlaute kommen nach den “normalen” Buchstaben
- Der Vergleich von Zahlen entspricht den mathematischen Regeln
 - $-n < 0 < n$

Datentypen – Literale

- Festwerte (Literale) müssen in einem entsprechenden Format eingegeben werden
 - Zahlenwerte als Zahl, z.B.
`(...) WHERE verkaufspreis > 50.00`
 - Zeichenketten werden in Hochkomma (Apostroph) eingeschlossen, z.B.
`(...) WHERE titel = 'Gestern war heute noch morgen.'`
 - In einer Zeichenkette enthaltene Hochkommata / Apostrophe müssen „verdoppelt“ werden, z.B.
`(...) WHERE titel = 'L' 'Équipe'`
 - Datumswerte – siehe später

Felder ohne Inhalt

- Ein leerer “Inhalt” in einem Feld wird durch den besonderen Wert **NULL** gekennzeichnet
- Vergleiche mit **NULL** können nur mittels **IS NULL** (Feld hat keinen Inhalt) und **IS NOT NULL** (Feld enthält einen regulären Wert) erfolgen, nicht mit den normalen Vergleichsoperatoren
- **NULL**-Felder verbrauchen keinen Speicherplatz

Felder ohne Inhalt

- Beispiel für eine Abfrage auf **NULL**
 - `select * from person where wohnort is not null;`
- Eine aufbereitete Ausgabe leerer Inhalte kann z.B. mittels **CASE - WHEN** oder **COALESCE** erfolgen
 - `select name,
 case when wohnort is null
 then 'Wohnort unbekannt oder ohne festen Wohnsitz'
 else wohnort end
from person;`
 - `select name,
 coalesce(wohnort,
 'Wohnort unbekannt oder ohne festen Wohnsitz')
from person;`

Datentypen und NULL

- Wie lautet die Nummer des Artikels mit der Bezeichnung *Diabolo Birdie*?
- Welche Filialen sind in der Region *ACAL*?
- Welche Artikel (Ausgabe Artikelnummer, Preis und Währung) kosten einen Betrag von mehr als *700* (in der angegebenen Währung)?
- Welche Artikel (Ausgabe Artikelnummer, Preis und Währung) kosten einen Betrag von *0,48* oder weniger (in der angegebenen Währung)?
- Wie lauten die Artikelnummern und Bezeichnungen der Artikel aus der Produktgruppe *GLOWB*?
- Bei welchen Produktgruppen ist keine Bezeichnung hinterlegt?

Bedingte Auswahl – Bereiche

- Die Treffermenge kann neben der fachlichen Selektion auch rein technisch mit **LIMIT** beschränkt werden
 - Beispiel: `select limit 0 3 * from person;`
- Ein Intervall kann mit **BETWEEN** angegeben werden
 - Beispiel: `select * from ort where alt_min between 50 and 75;`
 - Die Intervallgrenzen sind jeweils inklusive (d.h. geschlossenes Intervall)
- Auswahl aus einer Liste von Werten mit **IN**
 - Beispiel:
`select * from ort
where name in ('Paris', 'London', 'Besançon');`

Bedingte Auswahl – Wildcards

- Wildcards können mit dem Vergleichsoperator **LIKE** verwendet werden
- Beispiel 1: Personen, deren Name mit M beginnt und auf er endet
`select * from person where name like 'M%er';`
- Beispiel 2: Alle Orte mit up im Namen
`select * from ort where name like '%up%';`
- Beispiel 3: Suche nach einem einzelnen Zeichen als Wildcard
`select * from epoche where bezeichnung like 'Biederme_er';`
- Beispiel 4: Suche nach Wildcardzeichen im Text
`select * from wahlgesetz
where beschreibung like '%5!% Hürde%' escape '!';`^[1]

[1] Die Tabelle WAHLGESETZ wird nicht mit den Beispieldaten angelegt.

Wildcards – Volltextsuche

- Wildcards mit **LIKE** können problemlos und einfach für „normale“ Abfragen verwendet werden.
- Für beliebige Volltextsuchen in großen Datenbeständen, evtl. über mehrere Spalten, bieten sich spezialisierte Erweiterungen an. (z.T. vom Datenbankhersteller oder eigenständige Produkte zur Volltextindizierung)

Bedingte Auswahl – Negation

- **BETWEEN, IN** und **LIKE** können durch ein vorangestelltes **NOT** verneint werden
- Beispiel:

```
select * from werk  
where autor not in (1564, 1888, 1876);
```

Bedingte Auswahl

- Wie lauten die Produktgruppen, deren Bezeichnungen mit den Buchstaben *A* bis *D* beginnen?
- Gib die ersten 10 Zeilen der Tabelle **listung** aus.
- Welche Filialen haben eine Filialnummer zwischen *1700* und *2000*, je einschließlich?
- Welche Artikel sind in den Produktgruppen *GLOW*, *GLOWB* und *NITE*?
- Bei welchen Artikeln beginnt die Bezeichnung mit *Beard*?
- Welche Filialen sind nicht in den Regionen *ILE*, *PVAL* und *AVRA* gelegen?
- Welche Artikeltexte enthalten ein Prozentzeichen?
- Welche Filiale ist in der Straße *Chemin de l'Enfer*?

Verknüpfte Bedingungen

- Ein Beispiel aus der Mengenlehre



Verknüpfte Bedingungen

- Die logischen Operatoren **AND** und **OR** kombinieren das Ergebnis zweier Bedingungen zu einem einzigen Ergebnis
- Der logische Operator **NOT** negiert eine Bedingung
- Werden mehrere Bedingungen verknüpft, gelten die Regeln der Booleschen Algebra (siehe Folgeseite)
- Bedingungen können mit runden Klammern () beliebig tief verschachtelt werden

Boolesche Algebra

- Kommutativgesetz
 - $a \text{ AND } b = b \text{ AND } a$
 - $a \text{ OR } b = b \text{ OR } a$
- Assoziativgesetz
 - $(a \text{ AND } b) \text{ AND } c = a \text{ AND } (b \text{ AND } c)$
 - $(a \text{ OR } b) \text{ OR } c = a \text{ OR } (b \text{ OR } c)$
- Distributivgesetz
 - $a \text{ AND } (b \text{ OR } c) = (a \text{ AND } b) \text{ OR } (a \text{ AND } c)$
 - $a \text{ OR } (b \text{ AND } c) = (a \text{ OR } b) \text{ AND } (a \text{ OR } c)$

Operatoren

- Auswertung der Operatoren

Konjunktion

<i>AND</i>	<i>falsch</i>	<i>wahr</i>
<i>falsch</i>	<i>falsch</i>	<i>falsch</i>
<i>wahr</i>	<i>falsch</i>	<i>wahr</i>

Disjunktion

<i>OR</i>	<i>falsch</i>	<i>wahr</i>
<i>falsch</i>	<i>falsch</i>	<i>wahr</i>
<i>wahr</i>	<i>wahr</i>	<i>wahr</i>

Negation

	<i>NOT</i>
<i>falsch</i>	<i>wahr</i>
<i>wahr</i>	<i>falsch</i>

Verknüpfte Bedingungen

- Beispiele
 - `alter < 30 and gehalt > 5000`
 - `vorname = 'Elvis' or geburtsjahr = 1935`
 - `not typ = 'KFZ'`

Präzedenz der Operatoren – Klammern

- **AND** bindet stärker als **OR**
 - $A \text{ AND } (B \text{ OR } C) = A \text{ AND } B \text{ OR } A \text{ AND } C$
 - $(A \text{ AND } B) \text{ OR } C = A \text{ AND } B \text{ OR } C$
- Die Regeln bei der Klammersauflösung sind analog der Auflösungsregeln bei arithmetischen Ausdrücken, wobei **AND** vergleichbar ist mit der Multiplikation, **OR** mit der Addition; es gilt „Punkt vor Strich“.

Historisches

- Die Boole'sche Algebra ist benannt nach dem irischen Mathematiker und Philosophen George Boole (1815 – 1864).

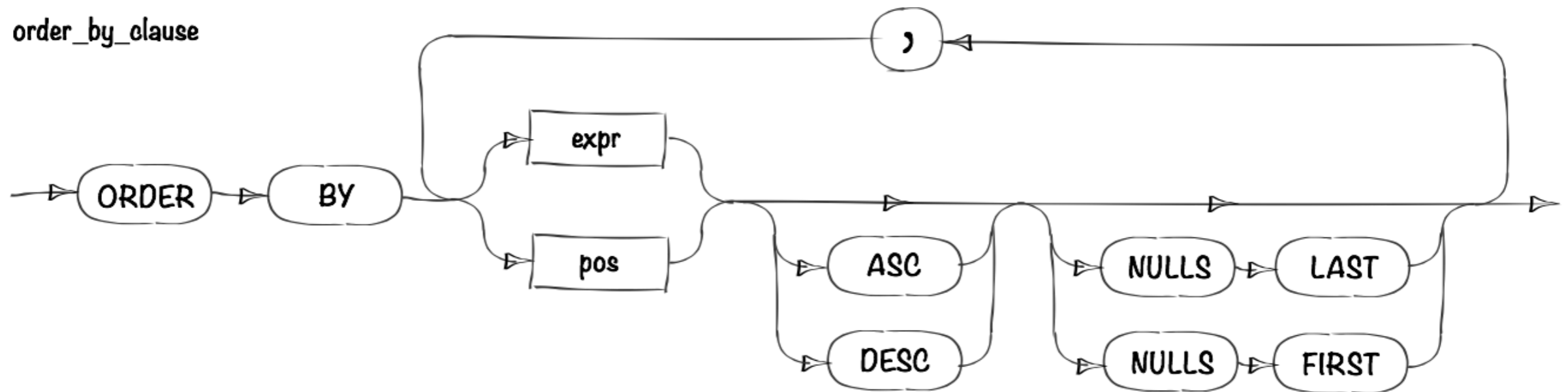


Bildquelle: School of Computer Science
The University of Birmingham

Verknüpfte Bedingungen

- Was kostete der Artikel mit der Artikelnummer *120300* in der Filiale *1390* während der Sonderaktion (Tabelle **sonderpreis**)?
- Wird der Artikel mit der Artikelnummer *212300* in den Filialen *1500* und *1886* (d.h. in mindestens einer der beiden Filialen) verkauft?
- Welche Filialen sind aus der Region *ACAL* und haben die Hausnummer *20*, oder aber sind in der Region *ILE* gelegen und in einer Straße ansässig, deren Namen nicht *Rue...* ist? Achte auf die Wildcards bei der Betrachtung der Hausnummern und Straßennamen.
(Zusatzaufgabe: Welche Aspekte müssten bei einer Zerlegung der Spalte mit Straße und Hausnummer in einzelne Spalten – Stichwort Normalisierung – beachtet werden?)

Sortierte Ausgabe



- Aufsteigende Sortierung mit **ASC** (ascending)
- Absteigende Sortierung mit **DESC** (descending)
- **ASC** ist die Voreinstellung und muss daher nicht extra angegeben werden

Sortierte Ausgabe

- Als Sortierkriterium kann ein Spaltenname oder die Position der Spalte in der Abfrage angegeben werden
 - Beispiel für Spaltenname: `order by artikelnummer`
 - Beispiel für Position: `order by 1`
- Mehrere Sortierkriterien können mit Komma getrennt angegeben werden
 - Beispiel: `order by nachname, vorname`
- Das Sortierkriterium muss nicht zwingend als Spalte ausgegeben werden
 - Beispiel: `select name from person order by geburtsdatum desc;`

Wiederholung Vergleichsoperatoren

- Der Vergleich von Zeichenketten folgt dem Schema
 - $a < ab < b$, d.h. bei gleichem Anfang ist der kürzere Text “kleiner”
 - $0..9 < A..Z < a..z$
 - Sonderzeichen wie z.B. Umlaute kommen nach den “normalen” Buchstaben
- Der Vergleich von Zahlen entspricht den mathematischen Regeln
 - $-n < 0 < n$

Bereits aus dem Thema Datentypen bekannt

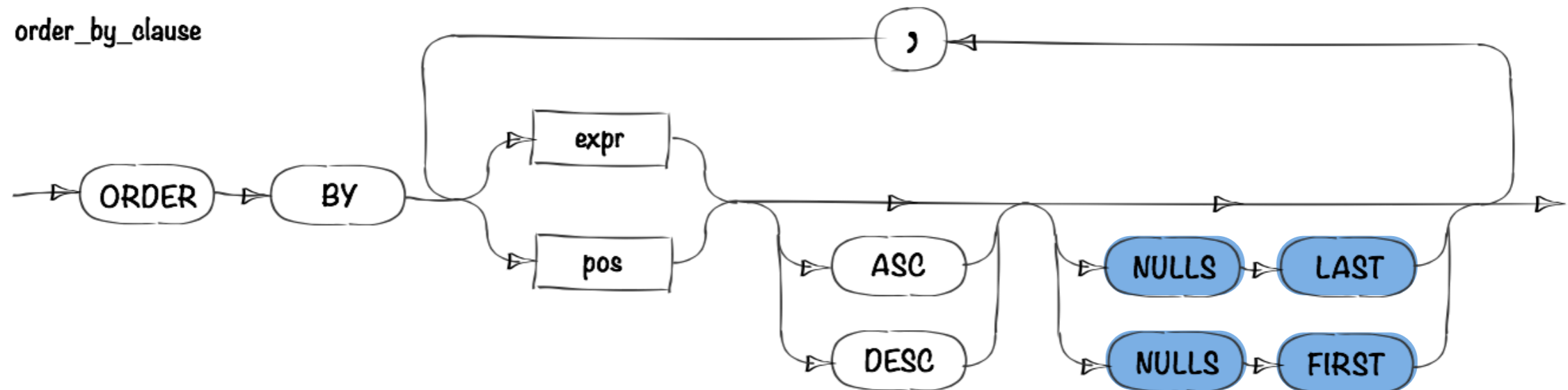
Landes- und sprachspezifisch vergleichen

- Um bei der Sortierung landes- und sprachspezifische Eigenheiten wie z.B. Umlaute oder Accents zu berücksichtigen, kann mit **COLLATE** gearbeitet werden.
- Beispiele:

```
select * from filiale order by ort collate 'French 2';  
select * from filiale order by ort collate 'German'
```
- Wenn auch Vergleiche in z.B. **WHERE**-Klauseln diese Eigenheiten berücksichtigen sollen, kann eine **COLLATION** allgemein eingestellt werden.
- Das Problem besteht für mehrsprachige Inhalte trotzdem weiterhin.

Sortierung – Felder ohne Inhalt

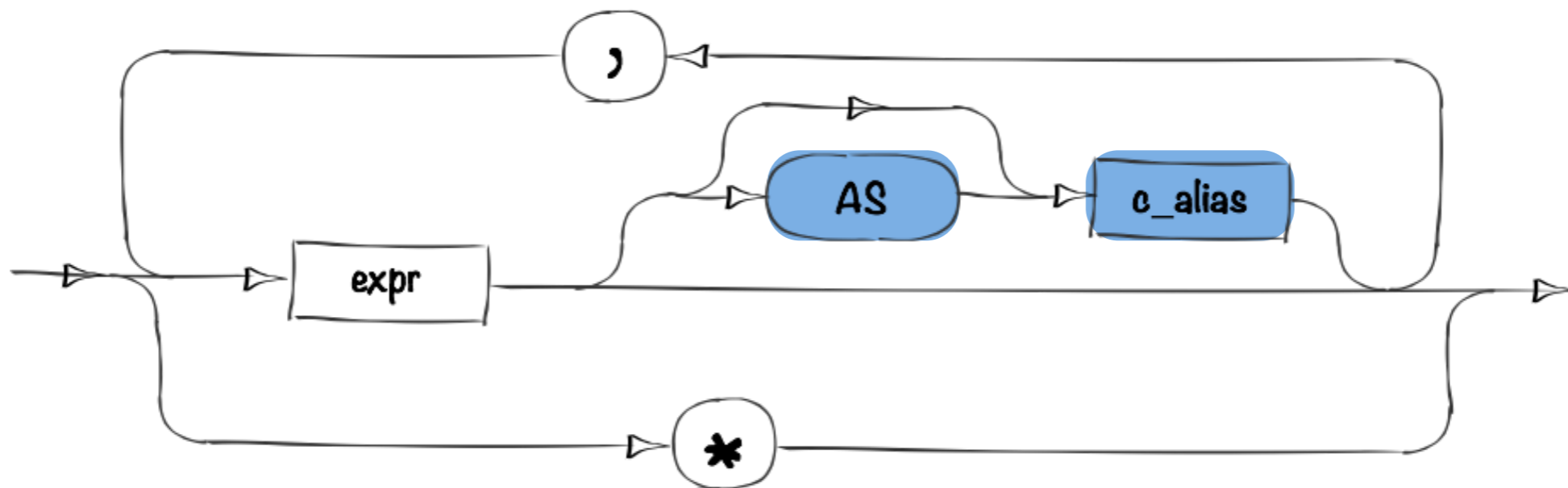
- Wegen der nicht eindeutigen und intuitiven Definition, ob nichts größer oder kleiner als ein regulärer Wert ist, ist ein Vergleich laut Definition immer falsch. Für die Sortierung kann daher die Position von **NULL**-Werten explizit angegeben werden



Aliasnamen und rechnen im SELECT

- Zu jeder Spalte kann ein Alias für die Ausgabe angegeben werden
- Beispiele:
`select name "Nachname" from person;`
`select name Nachname from person;`
`select name as Nachname from person;`
- In der Variante mit Anführungszeichen wird zwischen Groß- und Kleinschreibung unterschieden.

`select_list`



Aliasnamen und rechnen im SELECT

- Mit den Spaltenwerten kann gerechnet werden
 - Operatoren + - * / für Addition, Subtraktion, Multiplikation und Division
 - Beispiel: `select alt_max - alt_min from ort;`
- Zeichenketten können aneinander gefügt (konkateniert) werden
 - Operator ||
 - `select name || ', ' || vorname from person;`
- Bei anderen Datenbanksystemen u.U. andere Operatoren für die Konkatenation (z.B. + bei MS SQL Server)

Sortierung, Aliase und Berechnungen

- Gib alle Filialen aus, deren Filialnummer mit einer 2 beginnt. Die Ausgabe soll nach der Region absteigend und bei gleicher Region nach dem Ortsnamen aufsteigend erfolgen.
- Erzeuge für alle Filialen, deren Filialnummer mit 11 beginnt, nachfolgende Liste:

Filiale	Anschrift
Filiale 1100	35000, Rennes, 14, Rue des Chênes
Filiale 1110	51100, Reims, 20, Rue de la Chevalerie
Filiale 1120	76600, Le Havre, 16, Rue Claude Monet
Filiale 1130	42000, Saint-Étienne, 11, Rue Colbert
Filiale 1140	83000, Toulon, 14, Rue des Écoles
Filiale 1150	38000, Grenoble, 19, Rue Edouard Brigeon
Filiale 1160	49000, Angers, 17, Chemin de l'Enfer
Filiale 1170	21000, Dijon, 18, Chemin des Essarts
Filiale 1180	29200, Brest, 8, Rue Foch
Filiale 1190	30000, Nîmes, 25, Rue de la Fontaine de May

Einfache Funktionen (single-row)

- Einfache Funktionen können innerhalb des **SELECT**-Befehls die Position „normaler“ Ausdrücke einnehmen.

- Prinzip:

```
select function_name(spalte)
from   tabelle
where  function_name(spalte) = function_name(wert) ;
```

- Beispiel: Vorname aller Personen mit Nachname Presley (Groß- / Kleinschreibung soll beim Vergleich nicht signifikant sein) in Kleinbuchstaben ausgeben

```
select lower(vorname)
from   person
where  upper(name) = 'PRESLEY' ;
```

- Funktionen können beliebig verschachtelt werden. Die Auswertung erfolgt von innen nach außen.

Einfache Funktionen (single-row)

- Auswahl an Funktionen

Funktionsname	Beschreibung
<code>round(spalte [, n])</code>	Rundet auf n Nachkommastellen
<code>trunc(spalte[, n])</code>	Schneidet bei n Nachkommastellen ab
<code>mod(spalte, n)</code>	Modulo n, d.h. Rest nach Division durch n
<code>abs(spalte)</code>	Betrag eines Werts
<code>position(sub in str)</code>	Sucht sub in str und gibt die Position zurück – 0 wenn die gesuchte Zeichenkette nicht enthalten ist.
<code>substring(str from start [for len])</code> alternativ: <code>substr(str, start [len])</code>	Liefert den Teil der Zeichenkette str ab Position start bis zum Ende oder mit len Zeichen Länge
<code>concat(expr[, expr])</code>	Konkatenation von Zeichenketten (Alternative zum Operator)
<code>length(str)</code>	Länge einer Zeichenkette (Achtung: Das Ergebnis ist in diesem Fall numerisch)
<code>rtrim(str)</code> und <code>ltrim(str)</code>	Schneidet Leerzeichen ab
<code>upper(str)</code> und <code>lower(str)</code>	Wandelt str in Großbuchstaben respektive Kleinbuchstaben um

Datumsformat – DATE (1)

- Datum und Uhrzeit werden in einem internen Format **DATE**, **TIME** oder **TIMESTAMP** gespeichert.
- Bei diesen Datentypen gibt es i.d.R. leichte Unterschiede zwischen den verschiedenen Datenbanksystemen.
- Das Ausgabeformat bestimmt, welcher Teil der Datums- bzw. Zeitangabe angezeigt wird.
- Die Umwandlung eines Datums in Textdarstellung erfolgt mit der Funktion `to_char()` umgekehrt mit der Funktion `to_date()`.

Datumsformat – DATE (2)

- Formatelemente für die Ausgabe
 - **YYYY** Vierstelliges Jahr
 - **MM** Zweistelliger Monat
 - **MON, MONTH**
Monatsname abgekürzt oder ausgeschrieben
 - **DD** Zweistelliger Tag
 - **HH24** Stunde in 24-Stunden Darstellung (ohne AM / PM)
 - **MI** Zweistellige Minuten
 - **SS** Zweistellige Sekunden (...weitere Formatelemente sind möglich)

Datumsformat – DATE (3)

- Beispiele

- `select to_char(geburtsdatum, 'DD.MM.YYYY') from person;`

Datumsanzeige ohne Uhrzeit

- `select to_char(geburtsdatum, 'DD.MM.YYYY HH24:MI:SS')
from person;`

Datumsanzeige mit Uhrzeit (je nach DB evtl. nicht in **DATE** enthalten)

- `select to_char(geburtsdatum, 'MM/YYYY') from person;`

Monat und Jahr

- `select to_char(geburtsdatum, '"Geburtstag: " DD.MM.YYYY')
from person;`

Geburtstag mit vorangestelltem Text

Verwendung von DATE

- Liegen auf beiden Seiten eines Operators bereits Felder im **DATE**-Format (oder **TIME**) vor, sind Vergleiche unproblematisch
 - Beispiel: `datum1 <= datum2`
- Werden Datumsangaben in der **WHERE**-Klausel verwendet, sind oftmals Formatwandlungen oder Rundungsoperationen erforderlich

Verwendung von DATE (2)

- Beispiele
 - `select * from person
where geburtsdatum = to_date('08.01.1935', 'DD.MM.YYYY');`
 - `select * from person
where geburtsdatum between
to_date('01.01.1999 20:00:00', 'DD.MM.YYYY HH24:MI:SS')
and
to_date('01.01.1999 22:00:00', 'DD.MM.YYYY HH24:MI:SS');`
- Das Standardformat kann ohne `to_date()` verwendet werden (je nach Datenbank und Locale verschieden)
 - `select * from person
where geburtsdatum = '1935-01-08';`

Funktionen und Datumsformat DATE

- Gib für alle Artikel, deren Bezeichnung mit *Feuer* beginnt, eine Liste mit folgenden Spalten aus:
 - Bezeichnung in Kleinbuchstaben, Bezeichnung in Großbuchstaben, Länge der Bezeichnung
- Erzeuge eine Liste mit allen Filialen, deren Ortsname länger als 16 Zeichen ist.
- Erzeuge eine Liste der Sonderpreise, die ab dem 3. Oktober letzten Jahres (als Literal vorgegeben) galten.
- (Fortsetzung nächste Seite...)

Funktionen und Datumsformat DATE



- Zusatzaufgabe: Erzeuge die Liste der vorherigen Teilaufgabe, diesmal mit dynamischer Datumsermittlung. Das aktuelle Systemdatum kann direkt mit **CURRENT_DATE** angesprochen werden. Mit der Funktion **EXTRACT** können Teile aus einem Datumswert extrahiert werden.

- Syntax:

```
EXTRACT ( { YEAR | MONTH | DAY | HOUR | MINUTE | SECOND  
          | TIMEZONE_HOUR | TIMEZONE_MINUTE | DAY_OF_WEEK | WEEK_OF_YEAR }  
        FROM expr )
```

Gruppenfunktionen

- Gruppenfunktionen wirken sich auf Gruppen von Zeilen aus (im Gegensatz zu den bisher eingeführten single-row Funktionen, die sich immer auf einzelne Attributwerte in jeder Zeile beziehen).
- Gruppenfunktionen liefern ein Ergebnis (in einer Ergebniszeile) pro Gruppe.
- Die einfachste Gruppe ist die gesamte Tabelle.
- Gruppenfunktionen können in **SELECT** Spaltenlisten, in **ORDER BY**- und in **HAVING**-Klauseln (werden noch eingeführt) verwendet werden.

Gruppenfunktionen – Beispiel

TITEL	VEROEFFENTLICHUNG
The Taming of the Shrew	01.01.1594
Romeo and Juliet	01.01.1597
Le Malade imaginaire	10.02.1673
Hound Dog	05.06.1956
White Christmas	25.12.1941
Martin Eden	01.01.1909

Ältestes Werk
=
kleinster Wert in
VEROEFFENTLICHUNG

```
select min(veroeffentlichung)
from werk;
```



MIN(VEROEFFENTLICHUNG)

01.01.1594

Gruppenfunktionen

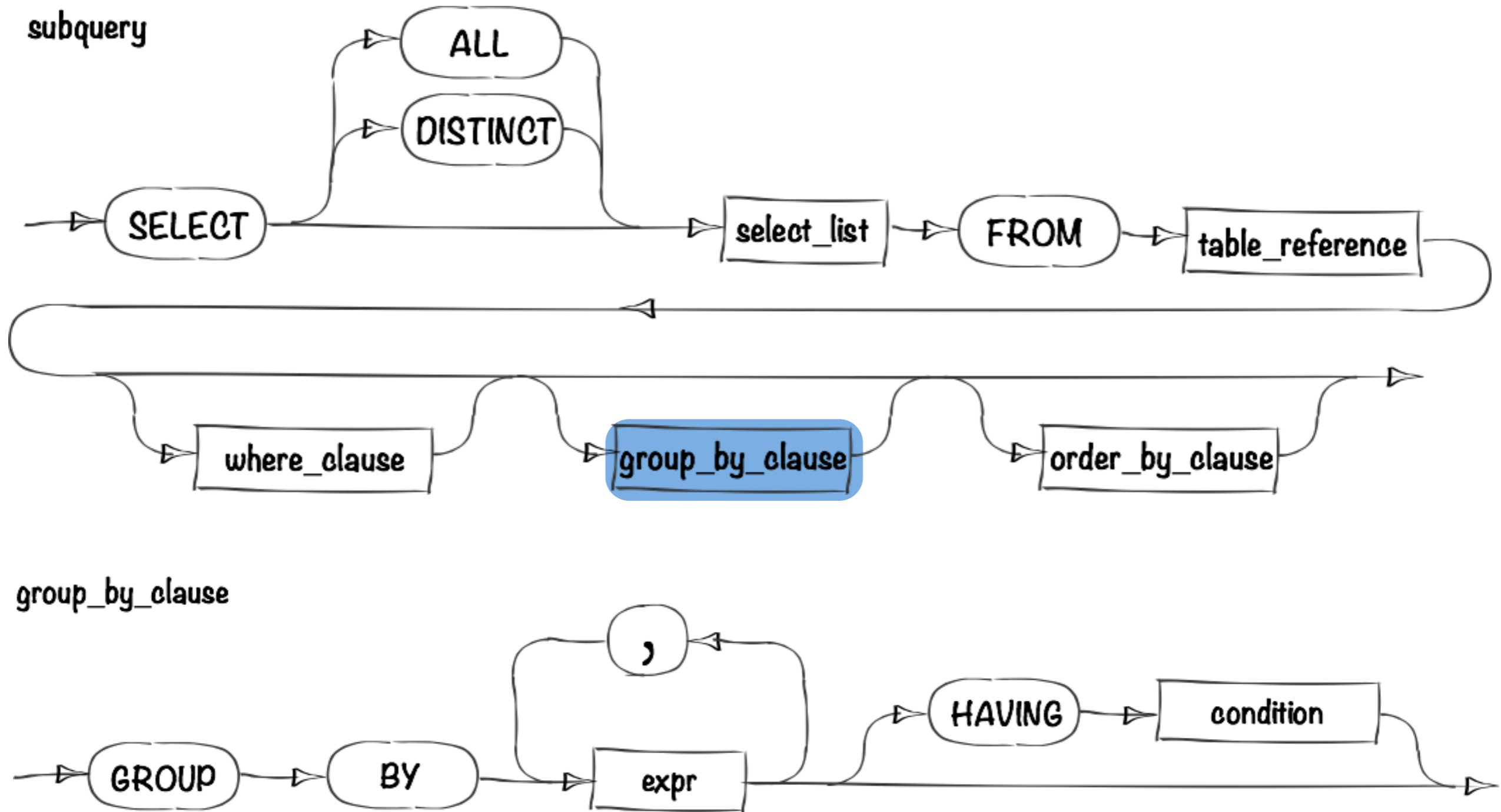
- Auswahl an Gruppenfunktionen

Funktionsname	Beschreibung
count	Anzahl der Datensätze, die im angegebenen Feld keinen NULL-Wert enthalten. Bei Angabe von * werden alle Datensätze gezählt. <code>select count(untertitel) from werk;</code> <code>select count(distinct sprache) from werk;</code> <code>select count(*) from werk;</code>
sum	Summe der Werte im angegebenen Feld (für numerische Felder). <code>select sum(alt_min) from ort;</code>
avg	Durchschnittswert der Inhalte im angegebenen Feld (für numerische Werte) <code>select avg(alt_min) from ort;</code>
min	Ermittelt den kleinsten Wert im angegebenen Feld <code>select min(geburtsdatum) from person;</code>
max	Ermittelt den größten Wert im angegebenen Feld <code>select max(veroeffentlichung) from werk;</code>

Daten gruppieren

- Mit der **GROUP BY**-Klausel können Datensätze zu Gruppen zusammengefasst werden.
- Auf die einzelnen Gruppen können Gruppenfunktionen angewendet werden. Pro Gruppe wird eine Ergebniszeile erzeugt.
- Durch eine **HAVING**-Klausel kann auf Basis des Ergebnisses pro Gruppe gefiltert werden (Gruppen können ein- bzw. ausgeschlossen werden).
- Nur Gruppierungskriterien und Gruppenfunktionen können in der Ergebnisliste ausgegeben werden.

SELECT – Syntax mit GROUP BY



GROUP BY – Beispiele

- Ausgabe aller Geburtsorte bekannter Personen, jeweils mit Anzahl der Personen, die dort geboren sind
 - ```
select geburtsort, count(*)
 from person
 group by geburtsort;
```
- Nur Ortschaften ausgeben , in denen mindestens zwei bekannte Personen geboren wurden
  - ```
select    geburtsort, count(*)  
  from    person  
  group by geburtsort  
  having   count(*) >= 2;
```

Gruppenfunktionen und GROUP BY

- Zähle mit einer Gruppenfunktion die Anzahl der Zeilen in der Tabelle **listung**.
- Ermittle den kleinsten, den größten und den durchschnittlichen Preis aus der Tabelle **verkaufspreis** per Aggregatfunktionen in *einer* **SELECT**-Anweisung.
- Zähle die *unterschiedlichen* Regionen, in denen es Filialen gibt.
- Liste die Produktgruppen aus der Tabelle **artikel** gruppiert auf.
- Erweitere die **SELECT**-Anweisung aus der vorherigen Teilaufgabe um eine weitere Spalte, in der die jeweilige Anzahl der Artikel in der Produktgruppe aufgeführt ist. Die Ausgabe soll aufsteigend nach dieser Anzahl sortiert sein.
- Erweitere die **SELECT**-Anweisung aus der vorherigen Teilaufgabe so, dass nur noch Produktgruppen mit mindestens 60 Artikeln ausgegeben werden.