ER Modell: Rechteck = Entity; Beziehung = Diamant; Kardinalität = {0, 1, N, M, 0..1} über der Verbindungslinie; Entities = Ovale (Primary-Keys sind Unterstrichen; Foreign keys sieht man nicht, weil implizit über die Verbindung)

INSERT INTO a VALUES (1, 'A'), (2, 'B'), (3, 'C'), (40, 'D'), (50, 'E');

CREATE TABLE a (id INTEGER PRIMARY KEY, comment CHARACTER VARYING NOT NULL);

SELECT: Comparison: =, <>, !=, <, >, <=, >= ; Arithmetic: +, -, *, /, %, ^; Logical: AND, OR, NOT; **Pattern Matching**: LIKE, ILIKE, SIMILAR TO; **Regular Expressions**: ~ (case-sensitive), ~* (case-insensitive), !~ (not match), !~* (not match); **Range**: <@ (contains), @> (is contained by), && (overlaps);

Date literals: 'YYYY-MM-DD', 'YYYY-MM-DD HH:MI:SS'; **Typed literals**: DATE 'YYYY-MM-DD', TIMESTAMP 'YYYY-MM-DD HH:MI:SS'; **to_date('**DD.MM.YYYY', 'DD.MM.YYYY'**)**; INTERVAL '1 day', '2 hours'; **NOW(), CURRENT_DATE, CURRENT_TIMESTAMP, EXTRACT(field FROM source), AGE(source, reference)**

COUNT(expr), SUM(expr), AVG(expr), MIN(expr), MAX(expr), STDDEV(expr), VARIANCE(expr), STRING_AGG(expr, delimiter), ARRAY_AGG(expr), JSON_AGG(expr), BOOL_AND(expr), BOOL_OR(expr), EVERY(expr)

SELECT */* OPTIONAL: UNIQUE / DISTINCT / DISTINCT ON*/* **DISTINCT** /*Einmalige Werte*/ **DISTINCT ON** (spalte1, spalte2) t.spalte1, /* einfache Spalte */ t.spalte2 AS alias2, -- aliasierte Spalte FROM schema.tabelle1 AS t -- Verschiedene JOIN-Typen INNER JOIN schema.tabelle2 AS t2 ON t.fk = t2.id -- INNER JOIN LEFT OUTER JOIN schema.tabelle3 AS t3 USING (gemeinsame_spalte) -- USING RIGHT JOIN schema.tabelle4 AS t4 –RIGHT JOIN NATURAL LEFT JOIN schema.tabelle5 AS t5 CROSS JOIN schema.tabelle6 -- CROSS JOIN FULL JOIN LATERAL (SELECT ...) AS sub ON true -- 2. Filter WHERE t.spalte1 = 'Wert' – Gleichheit AND t.spalte2 LIKE '%muster%' -- Pattern-Matching AND t.spalte3 IN (1,2,3) -- IN-Liste AND t.spalte4 BETWEEN 10 AND 20 – Bereich AND (t.spalte5 IS NULL OR t.spalte6 IS NOT NULL) AND t.datum >= DATE '2025-01-01' AND t.beschreibung ~* 'regex' -- Regex-Operator (case-insensitive) -- 3. Gruppierung für Aggregate GROUP BY t.spalte1, t.spalte2 -- 4. Filter auf Gruppenebene HAVING COUNT(*) > 1 -- 6. Kombinieren mehrerer Queries UNION [ALL] SELECT ... INTERSECT [ALL] SELECT ... EXCEPT [ALL] SELECT ... -- 7. Sortierung ORDER BY t.spalte1 DESC, -- auf- oder absteigend alias2 ASC -- 8. Paging LIMIT 10 OFFSET 20 /* FETCH FIRST 10 ROWS ONLY */;

Normalformen: *Vorherige Normalformen müssen immer erfüllt sein*. **1NF (Normalform - Zeilen und Spalten erweitern**) Alle Attribute atomar sind, d.h. jedes Feld enthält nur einen Wert. Mehrere Attribute in einer Spalte werden zu neuen spalten. Mehrere Datensätze vom gleichen Attribut in der Zelle werden zu einer Neuen Zeile. **2NF**: Nicht-Schlüsselattribute (Spalten), die eigentlich einen anderen Primärschlüssel haben (sofern der in der Tabelle auch vorhanden), sollte dieser mitsamt diesem in eine eigene Tabelle verlegt werden. **3NF**: Gleiches Vorgehen wie bei der 2NF, einziger Unterschied ist, dass der Primärschlüssel ein bereits bestehender Foreign-Key ist.

Integrität (Constraints):

Nur **erlaubte** werte (hauptsächlich eine Zahlen **Range**) `CREATE TABLE ... price numeric **CHECK** (price > 0)` oder `... **NOT NULL**`.

Keine **Doppelten** Primary-Keys/Zeilen (i.e. **Unique**) `CREATE TABLE ... id serial UNIQUE`.

Keine Null-Pointer auf Primary-Keys (**Referentielle Integrität**). **CASCADE** = Alles mit löschen, das darauf referenziert**; RESTRICT=**Nicht Löschen, wenn darauf referenziert. `CREATE TABLE ... FOREIGN KEY (bestellung_id) REFERENCES <<table_name>> (<<id_column>>) **ON UPDATE CASCADE ON DELETE** <<**CASCADE** or **RESTRICT**>>`

Transaktion: BEGIN; =Anfang, **COMMIT**; =Abschluss, **ROLLBACK**; =Abbruch

ACID: 1.**Atomarität**: Eine Transaktion wird entweder komplett durchgeführt, oder sie hinterlässt keine Spuren ihrer Wirkung auf die Datenbank. 2.**Konsistenz**: Während der Transaktion darf Konsistenz verletzt werden, wenn sie am Ende wieder hergestellt ist. 3.**Isolation**: Wenn ein User das macht, muss das gleiche Herauskommen, wie wenn 5 gleichzeitig, z.B. das Letzte Produkt Kaufen. 4.**Dauerhaftigkeit**: Datenbankzustände müssen so lange gültig sein und erhalten bleiben, bis sie von Transaktionen verändert werden.

Block 2

CAP-Theorem

- • **Konsistenz (Consistency):** Alle Knoten zeigen zur gleichen Zeit die gleichen Daten an.
- **Verfügbarkeit (Availability):** Jeder Anfrage wird garantiert eine Antwort geliefert – auch wenn sie nicht den aktuellsten Stand widerspiegelt.
- **Partitionstoleranz (Partition Tolerance):** Das System funktioniert weiter, auch wenn Teile des Netzwerks ausfallen oder nicht miteinander kommunizieren können.
- **CA (Konsistenz + Verfügbarkeit):**
  Funktioniert nur ohne Netzwerkausfall.
  Beispiel: Einzelner MySQL-Server.
- **CP (Konsistenz + Partitionstoleranz):**
  Bleibt bei Ausfall konsistent, aber nicht immer erreichbar.
  Beispiel: HBase, MongoDB (strikte Konsistenz).
- **AP (Verfügbarkeit + Partitionstoleranz):**
  Immer erreichbar, aber Daten können kurz inkonsistent sein.
  Beispiel: Cassandra, DynamoDB.

PYTHON

```python
# --- Pandas ----------
pd.read_csv("path.csv") -> DataFrame          # CSV einlesen
pd.DataFrame(data) -> DataFrame               # DataFrame aus dict/array
df.head(), df.tail()                          # Erste/letzte Zeilen
df.info(), df.describe()                      # Überblick & Statistik
df.loc[row, col], df.iloc[row_idx, col_idx]   # Label- & Positionszugriff
df.groupby(keys).agg(func)                    # Gruppierung & Aggregation
pd.merge(left, right, on="col")               # Tabellen verbinden
df.pivot_table(values, index, columns, aggfunc="mean")  # Schnelles Pivottable
df.isna().sum(), df.fillna(value)             # Fehlende Werte prüfen/behandeln
df.sort_values(by="col"), df.sort_index()     # Sortieren
# --- NumPy ----------
np.array(data)                                # Array erstellen
np.arange(stop), np.linspace(start, stop, num) # Sequenzen
np.zeros(shape), np.ones(shape)               # Nullen/Einsen-Arrays
np.reshape(a, newshape)                       # Form ändern
np.mean(a), np.median(a), np.std(a)           # Statistiken
# --- Matplotlib ----------
import matplotlib.pyplot as plt
plt.figure(figsize=(6,4))                     # Figure erstellen
plt.plot(x, y, label="Linie")                 # Liniendiagramm
plt.scatter(x, y, c="r", label="Punkte")      # Streudiagramm
plt.bar(x, height, label="Balken")            # Balkendiagramm
plt.hist(data, bins=30, alpha=0.7)            # Histogramm
plt.axvline(x, color="k", ls="--")            # Vertikale Linie
plt.xlabel("x-Achse"); plt.ylabel("y-Achse")  # Achsenbeschriftung
plt.title("Titel"); plt.legend(); plt.tight_layout(); plt.show()
```

Block3

Reguläre Ausdrücke (Regex)

Sonderzeichen (`*+?{}[]\|()`) mit `\` escapen.

- **Platzhalter:** `.` (Zeichen), `\d` (Ziffer), `\w` (Buchstabe/Ziffer), `\s` (Whitespace). Großbuchstaben (`\D`, `\W`, `\S`) negieren die Auswahl.
- **Auswahl:** `[...]` (Zeichenmenge), `[^...]` (negierte Menge), `[a-z]` (Bereich).
- **Wiederholung:** `*` (0+), `+` (1+), `?` (0-1), `{n,m}` (Anzahl). `*?` für non-greedy.
- **Logik & Gruppen:** `()` (Gruppe), `(?P<n>...)` (benannte Gruppe), `|` (ODER).
- **Position:** `^` (Anfang), `$` (Ende), `\b` (Wortgrenze).
  Beispiel:

```
^(?P<timestamp>\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2})\s\[(?P<level>INFO|WARN|ERROR)\]\s\
(user:\s'(?P<user>[\w\.-]+@[\w\.-]+)'\)\s"(?P<message>.*?)"$
```

XPath & JSONPath

- **XPath (XML):** Navigation mit `/` (Kind), `//` (Nachfahre), `.` (aktuell), `..` (Eltern). Filtern mit `[...]`, z.B. `[@id='x']` (Attribut) oder `[1]` (Position). Werte mit `@attribut` oder `text()`.
- **JSONPath (JSON):** Start mit `$`. Navigation mit `.` oder `['key']`. `..` (rekursiv), `[*]` (alle Array-Elemente). Filtern mit `[?(@...)]`, wobei `@` das aktuelle Element ist.
  Beispiel:

XML

```
//course[@id='cds104']/lecturer[@type='external' and units > 5]/name/text()
```

```
$.movies[?(@.year > 1999 && @.title =~ /^Matrix.*/)].actors[*].name
```

HTTP-APIs & Web Crawling

- **HTTP-APIs:** Datenaustausch über URLs via REST (Methoden: GET, POST, etc.), oft mit API-Key zur Authentifizierung.
- **Web Scraping/Crawling:** Extrahiert Daten aus HTML. Auswahl mit CSS-Selektoren (z.B. `tag`, `.klasse`, `#id`). Crawler folgen Links (`<a>`). Beachte `robots.txt` und nutze `sitemap.xml`.

CSS

```
main#content article.published:not(.featured) h2 + div.meta a[href*="author"]
```