



Einführung

SQL Masterclass



Wie ist dieser Kurs aufgebaut?

- **Dieser Kurs deckt alles ab:**
 - Du lernst MySQL **und** PostgreSQL
- **Diverse Themengebiete:**
 - SELECT, WHERE
 - Tabellen verwalten (ALTER TABLE,...), Fremdschlüssel
 - Komplexere Abfragen: JOINS, GROUP BY
 - Abfragen Beschleunigen: Indexe



Wie ist dieser Kurs aufgebaut?

- **Weitere Themen:**

- Rechnen mit Datumsangaben
- Expertenwissen
- Transaktionen
- Datenbank modellieren
- Benutzer & Berechtigungen verwalten
- Stored Procedures & Functions
- Volltextsuche
- Trigger



MySQL oder PostgreSQL?

- **Du solltest dich (erstmal) für ein Datenbanksystem entscheiden**
- Anfangs gibt es auch noch kaum Unterschiede
- Im ersten Drittel des Kurses zeige ich quasi alles doppelt
- Später teilt sich der Kurs aber auf - ich weise dich dann nochmal drauf hin!





MySQL

- Ein einfacheres Datenbanksystem
- Weniger Features als PostgreSQL
- Dafür oft bei Webhostern enthalten
- **Ideal für:**
 - Du bist Webentwickler
 - Du betreust Webseiten
 - Einfachere Anwendungen

Powered by

MySQL®





MariaDB vs. MySQL

- MariaDB:
 - MySQL wurde von Oracle gekauft
 - Der Haupt-Entwickler hat daraufhin einen Fork gestartet: MariaDB
 - Bisher sind die Datenbanken relativ kompatibel
 - Bei neueren Features unterscheiden sie sich aber





PostgreSQL

- Ein komplexes Datenbanksystem
- Deutlich mehr Features als MySQL
- **Ideal für:**
 - Komplexere Anwendungen
 - Du kannst selbst entscheiden, welches Datenbanksystem du verwenden möchtest





MySQL oder PostgreSQL?

- **Du solltest dich (erstmal) für ein Datenbanksystem entscheiden**
- **Entscheidungshilfe:**
 - Du möchtest dich (auch) mit Webentwicklung beschäftigen
 - => MySQL mit phpmyadmin
 - Sonst:
 - PostgreSQL mit pgadmin4

Pgadmin vs. phpmyadmin

- Die Datenbank kümmert sich nur um das Verwalten der Daten
- Wir möchten die Datenbank aber über ein "grafisches Userinterface" einsehen können





Erste Schritte mit SQL: SELECT-Abfrage

SQL Masterclass



Daten abfragen: SELECT

- Mit einem SELECT kannst du Daten aus einer Tabelle abfragen:

```
SELECT * FROM tabelle
```

```
SELECT spalte1, spalte2,... FROM tabelle
```

```
=> SELECT firstname, age FROM customers
```

```
=> SELECT * FROM customers
```



Erste Schritte mit SQL: SELECT ... WHERE

SQL Masterclass



Daten abfragen: SELECT ... WHERE

- Bisher konnten entscheiden, welche Spalte wir ausgeben möchten
- Können wir nicht auch entscheiden, welche Daten wir ausgeben wollen?

```
SELECT * FROM tabelle WHERE bedingung
```



Daten abfragen: SELECT ... WHERE

- Welche Bedingungen werden unterstützt?
 - Mathematische Vergleiche: <, >, >=, <=
 - Gleichheit / Ungleichheit: = bzw. <>

```
SELECT * FROM tabelle WHERE age > 21
```

```
SELECT * FROM tabelle WHERE city = 'Mainz'
```



Daten abfragen: SELECT ... WHERE

- Bedingungen können auch komplex verschachtelt werden!
- **Beispiel:**
 - *age* > 21 AND *age* < 25
 - *city* = 'Mainz' OR *city* = 'Stuttgart'
 - *city* <> 'Mainz'
 - NOT (*city* = 'Mainz')



Erste Schritte mit SQL: SELECT COUNT(*), SELECT DISTINCT

SQL Masterclass



SELECT COUNT(*)...

- Manchmal interessieren dich die eigentlichen Daten nicht...
- ... und du möchtest nur die Anzahl der Einträge herausfinden
- Das geht mit einem SELECT COUNT(*)!
- Natürlich kannst du dies auch mit einem WHERE kombinieren!
- => SELECT COUNT(*) FROM customers WHERE ...




SELECT DISTINCT

- Gibt nur unterschiedliche Einträge aus
- `SELECT DISTINCT firstname FROM customers`
 - Gibt jeden Vornamen maximal einmal aus
- `SELECT DISTINCT firstname, lastname FROM customers`
 - Gibt jede Kombination von Vor- und Nachnamen nur
1x aus



SELECT COUNT(DISTINCT firstname))

- Natürlich können wir beides auch noch kombinieren:
- => SELECT COUNT(DISTINCT *firstname*)
- Gibt die Anzahl der unterschiedlichen Werte der Spalte *firstname* aus



Erste Schritte mit SQL: Werte durchsuchen mit LIKE

SQL Masterclass



SELECT ... WHERE spalte LIKE ...

- Oft möchtest du einen Text genauer durchsuchen
- Beispiel: Finde alle Kunden, deren E-Mail mit @gmail.com endet
- Mit einem LIKE kannst du auch solche Abfragen formulieren
- Hierbei gilt:
 - % => keiner, einer oder mehrere beliebige Buchstaben
 - _ => exakt ein Buchstabe



SELECT ... WHERE spalte LIKE ...

- **Hierbei gilt:**

- % => keiner, einer oder mehrere beliebige Buchstaben
- _ => exakt ein Buchstabe

- **Beispiel:**


- SELECT ... WHERE firstname LIKE 'a%'
- SELECT ... WHERE firstname LIKE '%a%'
- SELECT ... WHERE email LIKE '%@gmail.com'



SELECT ... WHERE spalte LIKE ...

- **Hinweise:**

- Hierbei durchsucht die Datenbank alle Einträge der Tabelle
- Dies kann gerade bei vielen Daten recht lange dauern
- Teils kann ein Index helfen, teils nicht
- Eine echte "Volltextsuche" ist oft schneller
- Darauf gehen wir aber später noch ein



Erste Schritte mit SQL: Werte mit IN und BETWEEN filtern

SQL Masterclass



SELECT ... WHERE spalte IN ...

- **Manchmal möchtest du z.B. 3 Einträge nach Namen finden:**
 - `WHERE firstname = 'Anne' OR firstname = 'Hartmut' OR
firstname = 'Jutta'`
- Das ist aber recht aufwendig zu schreiben
- **Sehr viel einfacher:**
 - `WHERE firstname IN ('Anne', 'Hartmut', 'Jutta')`



SELECT ... WHERE spalte BETWEEN

- Manchmal möchtest du z.B. nach dem Alter filtern:
 - **Beispiel:** Welche Kunden sind im erwerbsfähigen Alter?
 - **WHERE age >= 18 AND age <= 67**
- Das geht mit BETWEEN einfacher!
 - => **WHERE age BETWEEN 18 AND 67**



Erste Schritte mit SQL: ORDER BY und LIMIT

SQL Masterclass




ORDER BY...

- ORDER BY kommt ganz am Schluss der Query!
- Mit Hilfe von ORDER BY kannst du die Ergebnisse sortieren:
 - ... ORDER BY *firstname*
 - ... ORDER BY *firstname* ASC
 - ... ORDER BY *firstname* DESC
- Du hast auch die Möglichkeit, nach mehreren Spalten zu sortieren
 - ... ORDER BY *age* DESC, *firstname* ASC



LIMIT

- LIMIT kommt ganz am Schluss der Query - noch nach eine ORDER BY
- Hiermit kannst du z.B. die Ergebnisse auf 20 Einträge begrenzen:
 - ... LIMIT 20
- Du kannst aber auch angeben, ab wo die Einträge ausgegeben werden sollen, z.B. 20 Zeilen ab Zeile 40:
 - MySQL: LIMIT *offset*, *count* => ... LIMIT 40, 20
 - PostgreSQL: OFFSET *offset* LIMIT *count*
 - => ...OFFSET 40 LIMIT 20



Erste Schritte mit SQL: SELECT AS und Eingebaute Funktionen

SQL Masterclass



SELECT *spalte1* AS ...

- Mit dem Wort AS kannst du eine Spalte umbenennen:
 - SELECT *firstname* AS *fname* FROM customers
 - Die Spalte *firstname* wird mit *fname* beschriftet



Eingebaute Funktionen

- Es gibt Funktionen, die Daten “zusammenfassen”:
 - COUNT(*): Berechnet die Anzahl von Einträgen
 - MIN(spalte): Berechnet das Minimum in der Spalte
 - MAX(spalte): Berechnet das Maximum in der Spalte
 - AVG(spalte): Berechnet den Durchschnitt
 - SUM(spalte): Summiert alle Werte in der Spalte auf

Eingebaute Funktionen

- Es gibt aber auch Funktionen, die einzelne Daten verarbeiten:
 - UPPER(spalte): Großbuchstaben
 - LOWER(spalte): Kleinbuchstaben
 - LENGTH(spalte): Wie lang ist der Text in der Spalte?
 - SUBSTR(spalte, pos, len):
 - Ermittle einen Teil vom Text,
 - CONCAT(spalte1, spalte2,...)
 - Hängt Textbausteine aneinander



Erste Schritte mit SQL: Daten einfügen

SQL Masterclass



Daten einfügen

- Natürlich kannst du nicht nur Daten abfragen, du kannst auch neue Daten in eine Tabelle hineinschreiben
- Ein einfacher Insert-Befehl:
 - `INSERT INTO tabelle (spalte1, spalte2, ...)`
`VALUES (wert1, wert2, ...)`
- Mehrere Insert-Befehle auf einmal:
 - `INSERT INTO tabelle (spalte1, spalte2, ...)`
`VALUES (wert1, wert2, ...),`
`(wert1, wert2, ...);`



Daten einfügen

- Die Daten werden dann sofort in die Datenbank geschrieben
 - Später können wir das Verhalten auch noch konfigurieren: Transaktionen, ...
- Wir geben jetzt erstmal immer alle Spalten an (bis auf die ID)
 - Später ist dies nicht zwingend immer erforderlich
 - Warum, wieso, weshalb - darauf gehen wir dann auch noch später ein!



Erste Schritte mit SQL: Daten updaten

SQL Masterclass



Daten updaten

- Du kannst auch einen Eintrag aktualisieren:
 - `UPDATE tabelle`
 `SET spalte1='Wert1'`
 `WHERE ...`
 - `UPDATE tabelle`
 `SET spalte1='Wert1', spalte2='Wert2'`
 `WHERE ...`
- Das WHERE ist nicht zwingend erforderlich
- Wenn es weggelassen wird, werden aber alle Einträge aktualisiert / überschrieben!



Daten updaten

- Hierbei kannst du statt festen Werten auch Ausdrücke verwenden - diese werden dann ausgewertet:
 - `UPDATE customers`
 `SET age=age + 1`
 `WHERE ...`
- Hierbei wird zuerst der Ausdruck `age + 1` ausgewertet und dann zurück in die Spalte `age` geschrieben



Erste Schritte mit SQL: Daten löschen (MySQL)

SQL Masterclass



Daten löschen

- Einträge können über ein DELETE gelöscht werden;
 - `DELETE FROM tabelle1`
`WHERE ...`
- Das WHERE ist nicht zwingend erforderlich
- Aber: Dann werden alle Einträge gelöscht!
- **Wichtig:**
 - Die Daten werden sofort gelöscht!
 - Auch hier bietet es sich also an, explizit nach ID zu filtern



Erste Schritte mit SQL: Tabellen anlegen

SQL Masterclass



Tabelle erstellen

- In Excel entspricht dies einem Tabellenblatt
- Die Schreibweise ist dann wie folgt:

```
CREATE TABLE name (  
    spalte1 datatype,  
    spalte2 datatype  
)
```




Was hat es mit dem Datentyp auf sich?

- Gibt an, was für Daten in dieser Spalte gespeichert werden:
 - VARCHAR: Text mit maximaler Länge
 - TEXT: Text mit keiner festen Länge
 - Exakte Zahlen
 - Gerundete Zahlen
 - Datumswerte



Wie speichern wir Text?

- **VARCHAR:** Hier müssen wir eine Maximallänge angeben
- **MYSQL, TEXT:**
 - Limitiert auf ca. 65k Zeichen
 - MEDIUMTEXT: Limitiert auf ca. 16MB
 - LONGTEXT: Limitiert auf ca. 4GB
- **PostgreSQL, TEXT:**
 - Unbegrenzte Länge
 - Aber generell limitiert auf maximal 1GB an Text
(allgemeines Limit)



Erste Schritte mit SQL: Wie verwalten wir die Spalten einer Tabelle?

SQL Masterclass




Wie kannst du eine Tabelle verwalten?

- Manchmal möchtest du noch nachträglich eine Spalte entfernen / hinzufügen
- Dazu kannst du die Tabelle über ALTER TABLE verwalten
- Das passiert dann doch recht oft - oft zeigt sich erst später, dass doch noch eine zusätzliche Spalte benötigt wird!
- **Hinweis:**
 - U.U. kann dies recht lange dauern
 - Währenddessen kann es sein, dass Abfragen für diese Tabelle nicht beantwortet werden können



Tabelle verwalten

- **Tabelle entfernen:**
 - `DROP TABLE table`
- **Spalte entfernen:**
 - `ALTER TABLE table DROP COLUMN column`
- **Spalte hinzufügen:**
 - `ALTER TABLE table ADD COLUMN column datatype [BEFORE / AFTER] column2`
- **Spalte verändern:**
 - `ALTER TABLE table MODIFY COLUMN column datatype`



Erste Schritte mit SQL: Wie werden Zahlen gespeichert?

SQL Masterclass



Wie werden Zahlen gespeichert?

- **Wir müssen unterscheiden:**
 - **Ganze Zahlen**
 - **Kommazahlen:**
 - **Exakte Zahlen**
 - **Gerundete / Gleitkommazahlen**




Wie speichern wir exakte Zahlen?

- Wir können verschieden viele Bytes pro Zahl verwenden
- 1 Byte = 8 Bit, d.h. 8 Bit pro Zahl: 0 0 0 0 0 0 0 1
 - Bei 1 Byte = 8 Bit können wir also $2^8 = 256$ verschiedene Zahlen speichern
- 2 Byte = 16 Bit = 2^{16}
 - => 65.536 verschiedene Zahlen
- 4 Byte = 32 Bit = 2^{32}
 - => 4.294.967.296 verschiedene Zahlen
- 8 Byte = 64 Bit = 2^{64}
 - => $1,8 \cdot 10^{19}$ verschiedene Zahlen!



Wie speichern wir exakte Zahlen?

- 2 Byte, 2^{16} = 65.536 Zahlen:
 - **MySQL & PostgreSQL:**
 - SMALLINT, -32768 bis +32767
 - **Nur MySQL:**
 - UNSIGNED SMALLINT: 0 bis 65535
- 4 Byte, 2^{32} Zahlen:
 - PostgreSQL: INTEGER
 - MySQL: INT
- 8 Byte, 2^{64} Zahlen:
 - BIGINT



Erste Schritte mit SQL: Wie werden Kommazahlen gespeichert?

SQL Masterclass



Wie speichern wir Kommazahlen?

- **Exakte Kommazahlen:**

- DECIMAL: Hier können wir selbst bestimmen, wie viele Ziffern vor dem Komma oder nach dem Komma unterstützt werden sollen
- Beispiel: 123,45€
- Anzahl Ziffern (precision): 5
- Anzahl Ziffern hinter dem Komma (scale): 2
- => DECIMAL(5, 2)



Wie speichern wir gerundete Zahlen?

- **Gerundete Zahlen:**

- Je mehr Bytes, desto genauer können wir einen Wert speichern
- Der Wert wird aber gerundet - Achtung, nicht immer so, wie wir es erwarten würden!
- Achtung beim Vergleichen von Zahlen: 2,123 ist nicht immer das gleiche wie 2,123
- Vergleich: Prüfe ob: $\text{ABS}(a - b) < 0.00000001$
- REAL: 4 Bytes
- DOUBLE: 8 Bytes



Wann was?

- **Ganze Zahlen**
- **Kommazahlen:**
 - **Exakte Kommazahl (DECIMAL,...):** Business-Logik
 - **Gerundete Kommazahlen (DOUBLE / FLOAT):**
 - **Wissenschaftliche Berechnungen / ggf. Messwerte,...**



Erste Schritte mit SQL: NULL und Standardwert

SQL Masterclass



Was ist NULL? Wo ist der Unterschied zu einer 0?

- **NULL:**

- Steht für: “Keine Angabe” / “Nicht definiert”
- Beispiel: Wir speichern die Anzahl der getätigten Einkäufe pro Kunde in einer Spalte:
 - 0 => Kein Einkauf
 - 2 => Bereits 2 Einkäufe bei uns getätigt
 - NULL => Anzahl unbekannt
- Nicht immer möchten wir NULL erlauben
- Hier würde es eventuell wenig Sinn machen
- Aber: “Name nicht ausgefüllt”



Spalte mit NULL erstellen

Spalte mit NULL erstellen:

```
CREATE TABLE t (  
    num_orders INT NULL  
)
```

Spalte ohne NULL erstellen:

```
CREATE TABLE t (  
    num_orders INT NOT NULL  
)
```



Nach NULL suchen?

- Nach Null kannst du nicht suchen!
- NULL hat hierbei eine besondere Bedeutung, jeder Vergleich mit NULL ergibt NULL
- Also auch der Vergleich, ob eine Spalte NULL enthält
 - **Geht nicht: ... WHERE SPALTE = NULL**
- Lösung:
 - **... WHERE SPALTE IS NULL**
 - **... WHERE SPALTE IS NOT NULL**



Erste Schritte mit SQL: NULL

SQL Masterclass



Erste Schritte mit SQL: Was sind Standardwerte?

SQL Masterclass



Warum benötigst du Standardwerte?

- Standardwerte ermöglichen es, dass du bei einem INSERT diese Spalte nicht zwingend angeben musst
- Standardwert + NULL:
 - Wenn NULL erlaubt ist, kann aber auch NULL als Standardwert verwendet werden
 - Daher durften wir beim vorherigen Thema ein paar der Spalten auch mal weglassen
- Warum sind Standardwerte denn so wichtig?



Beispiel: Standardwerte

- **Beispiel:**
 - Wir möchten zu jedem Nutzer speichern, wie viele Bücher er schon gekauft hat
 - Wir könnten es (ohne NULL) als Spalte *num_books* modellieren:
 - 0 => 0 Bücher
 - 1 => 1 Buch gekauft,...
 - Unsere existierende Anwendung fügt die Nutzer aber ohne die Spalte *num_books* in die Datenbank hinzu
 - Das INSERT wird also fehlschlagen - NULL ist hier nicht erlaubt!




Beispiel: Standardwerte

- **Beispiel:**
 - Hier können wir dieser Spalte also einen Standardwert von 0 geben
 - Unsere existierende Anwendung kann also diese Spalte weglassen
 - Und die Datenbank trägt dann die 0 in die Spalte (automatisch) ein
 - Wir sparen uns so das Umprogrammieren der existierenden Anwendung!




Standardwerte definieren

- **DEFAULT:**
 - Hiermit kannst du einen Standardwert bestimmen
 - Dann muss später beim Einfügen diese Spalte nicht angegeben werden!
 - Das funktioniert natürlich auch mit anderen Datentypen, statt einer 0 könnte hier auch ein Wert (in einfachen Anführungszeichen) stehen
 - **CREATE TABLE t (**
 num_books INT NOT NULL DEFAULT 0
)



Erste Schritte mit SQL: Warum benötigen wir die Spalte "ID" (Excel-Beispiel)?

SQL Masterclass



Erste Schritte mit SQL: Warum benötigen wir die Spalte "ID"?

SQL Masterclass



Was ist eine ID?

- Die Spalte, in der die ID gespeichert ist, wird auch “Primärschlüssel” genannt
- Dadurch können wir uns eindeutig auf einen Eintrag beziehen
- Wir sagen damit quasi der Datenbank:
 - Diese Spalte ist eindeutig (pro Tabelle)
- Wie erstellst du eine solche Spalte?
 - Das ist unterschiedlich je nach Datenbank!



Primärschlüssel + Auto Increment

- **MySQL:**

- Primärschlüssel:
 - Ist Eindeutig pro Tabelle
 - Darüber können Einträge angesteuert werden
- Auto Increment:
 - Wenn wir einen neuen Eintrag einfügen, wird automatisch eine neue ID vergeben

- **PostgreSQL:**

- Datentyp SERIAL / BIGSERIAL




Primärschlüssel zur Tabelle hinzufügen

- **MySQL:**

- CREATE TABLE t (
 id INT NOT NULL AUTO INCREMENT,
 ...,
 PRIMARY KEY (id)
)

- **PostgreSQL:**

- CREATE TABLE t (
 id SERIAL PRIMARY KEY,
 ...
)



Komplexere Abfragen in SQL: SUBSELECT!

SQL Masterclass



Komplexere Abfragen

- Die Datenbank unterstützt aber noch viel komplexere Abfragen:
 - Subselect
 - JOIN
 - GROUP BY
 - Common Table Expression (WITH ...)
- Oft können wir über verschiedene Wege zum gleichen Ergebnis kommen, manchmal aber nicht!
- Unterschiedliche Wege sind unterschiedlich performant!
 - Mehr dazu später noch!



SUBSELECT!

- Ein Subselect ist quasi eine Query in der Query:
- ```
SELECT
 (SELECT ...) [AS something]
FROM tabelle
```
- Hierbei wird die innere Query i.d.R. einmal pro Datensatz der äußeren Tabelle ausgeführt
- Dadurch können wir mehrere Tabellen miteinander verknüpfen und richtig komplexe Abfragen ausführen



# SUBSELECT!

- **Beispiele:**

- Generiere eine Kundenliste, inklusive:
  - Anzahl der Bestellungen pro Kunde
  - Wann war die letzte Bestellung pro Kunde?
- Filtern nach Subselect:
  - Wie viele Kunden sind Stammkunden (haben bisher schon mehr als 5 Bestellungen bei uns getätigt)?
  - Wie viele Kunden haben noch keine Bestellung bei uns getätigt?



# Komplexere Abfragen in SQL: JOIN!

SQL Masterclass



# Tabellen verknüpfen: JOINS!

- **Es gibt verschiedene Typen:**

- CROSS JOIN
- LEFT JOIN
- RIGHT JOIN
- INNER JOIN
- FULL JOIN
- ...



# CROSS JOIN

- Bei einem CROSS JOIN wird das kartesische Produkt von 2 Tabellen gebildet
- **Was bedeutet das genau?**

| id | name   |
|----|--------|
| 1  | Max    |
| 2  | Moritz |


| c_id | course   |
|------|----------|
| 1    | Englisch |
| 2    | Spanisch |

| id | name   | c_id | course   |
|----|--------|------|----------|
| 1  | Max    | 1    | Englisch |
| 1  | Max    | 2    | Spanisch |
| 2  | Moritz | 1    | Englisch |
| 2  | Moritz | 2    | Spanisch |



# CROSS JOIN

- Hier wird also jeder Eintrag von Tabelle A mit jedem Eintrag von Tabelle B kombiniert
- **Die Ergebnismenge wird dadurch sehr groß:**
  - $[\text{Anzahl Einträge Tabelle A}] * [\text{Anzahl Einträge Tabelle B}]$
- **Natürlich können wir die Ergebnisse filtern:**
  - Mit einem WHERE
- Später schauen wir uns noch die anderen JOIN-Typen an - dort können wir die Ergebnismenge direkt einschränken.



# Komplexere Abfragen in SQL: INNER JOIN!

SQL Masterclass






# CROSS JOIN - Beispiel Bestellung mit Kundendaten

- **Was wir bisher gemacht haben:**
  1. Wir bauen das kartesische Produkt von der Tabelle Kunden ("customers") und Bestellungen ("orders") auf
  2. Anschließend filtern wir dieses, sodass nur Einträge übrig bleiben, wo eine Bedingung erfüllt ist:
    - `WHERE orders.customer_id = customers.id`
- In Schritt (1) werden sehr viele Datensätze generiert, die in Schritt (2) direkt wieder entfernt werden!
- Können wir das nicht effizienter gestalten?



# CROSS JOIN vs. INNER JOIN

- **Cross Join:**
  - `SELECT * FROM A CROSS JOIN B WHERE BEDINGUNG`
  - **Hier wird zuerst das kartesische Produkt gebildet, anschließend wird gefiltert nach BEDINGUNG**
- **INNER JOIN:**
  - `SELECT * FROM A INNER JOIN B ON BEDINGUNG`
  - **Hier wird nicht das komplette kartesische Produkt aufgebaut**
  - **=> Tendenziell performanter**



# Komplexere Abfragen in SQL: INNER JOIN vs. LEFT JOIN vs. RIGHT JOIN

SQL Masterclass



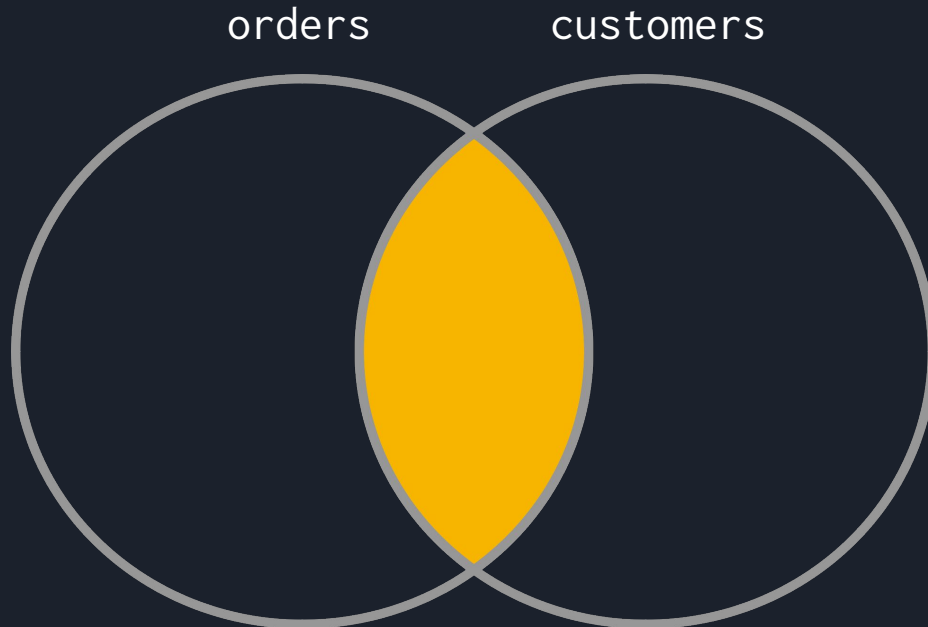
# INNER JOIN vs. RIGHT JOIN vs. LEFT JOIN

- **Es gibt noch andere Join-Typen:**
  - RIGHT JOIN / RIGHT OUTER JOIN
  - LEFT JOIN / LEFT OUTER JOIN
  - FULL JOIN / FULL OUTER JOIN
- **Diese können wir uns grafisch veranschaulichen...**
  - ... als Venn-Diagramm (Mengenlehre)
  - **Aber schauen wir uns zuerst mal den INNER JOIN nochmal grafisch an!**

# INNER JOIN

```
SELECT * FROM orders INNER JOIN customers ON orders.customer_id = customers.id
```

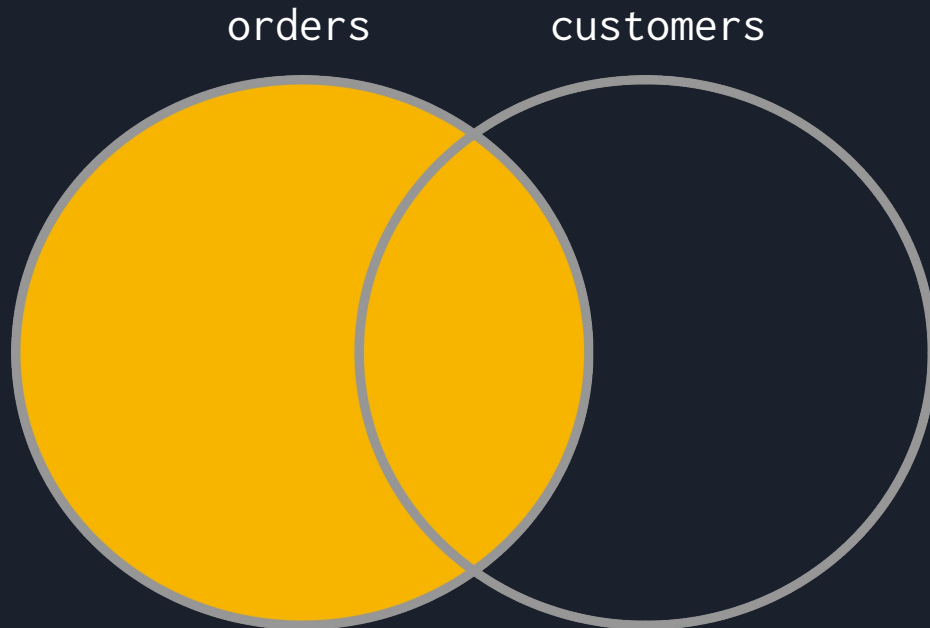
=> Nur Einträge, bei denen es sowohl eine Bestellung als auch einen Kunden gibt



# LEFT JOIN

```
SELECT * FROM orders LEFT JOIN customers ON orders.customer_id = customers.id
```

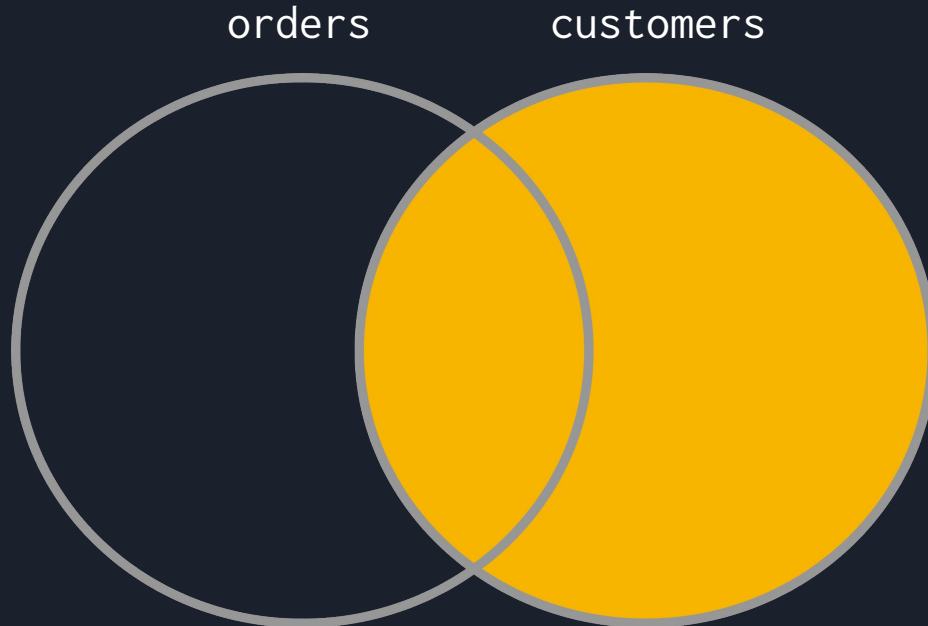
=> Alle Bestellungen, egal ob es dazu einen Kunden gibt



# RIGHT JOIN

```
SELECT * FROM orders RIGHT JOIN customers ON orders.customer_id = customers.id
```

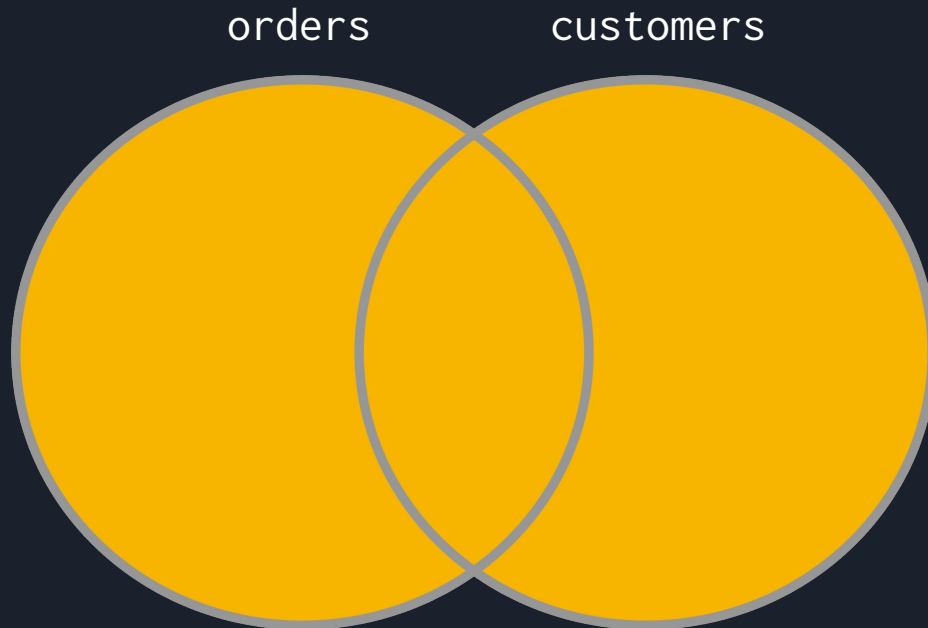
=> Alle Kunden, egal ob es dazu eine Bestellung gibt



# FULL JOIN

```
SELECT * FROM orders FULL JOIN customers ON orders.customer_id = customers.id
```

=> Alle Bestellungen und alle Kunden







# Komplexere Abfragen in SQL: Daten gruppieren

SQL Masterclass



# Group By vs. Subselect

- **Mit einem Group By können wir oft das gleiche Ergebnis erreichen wie mit einem Subselect**
- **Warum benötigen wir es dann überhaupt?**
  - Manche Sachen können wir mit einem Subselect nicht ausdrücken (und andersherum)
  - Tendenziell gilt aber (meine Erfahrung):
    - Group By ist performanter als ein Subselect
    - (Ausnahmen bestätigen die Regel - Testen!)



# Daten gruppieren: Group By

- **SQL kann die Daten für uns auch gruppieren:**
  - `SELECT ... FROM ... GROUP BY firstname`
  - Hier werden die Daten nach der Spalte "firstname" gruppiert
  - **Was dürfen wir hier als `SELECT` auswählen?**
    - Die Spalten, nach denen wir gruppieren
    - Funktionen, die die Daten zusammenfassen
      - `COUNT(*)`, `MAX(spalte)`, `AVG(spalte)`, ...
  - `SELECT COUNT(*), title FROM customers GROUP BY title`



# Komplexere Abfragen in SQL: Daten gruppieren (Teil 2)

SQL Masterclass



# Daten gruppieren: Group By

- **Wo wird das GROUP BY platziert?**

- SELECT ... FROM tabelle
- (LEFT JOIN / INNER JOIN / RIGHT JOIN / ...)
- WHERE ...
- GROUP BY ...
- HAVING ...



## Daten gruppieren: Group By

- **Where:**
  - **Filtert die Daten, bevor sie gruppiert werden**
- **Having:**
  - **Filtert die Daten, nachdem sie gruppiert wurden**



## Daten gruppieren: Group By

- **Wir dürfen auch nach mehreren Spalten gruppieren:**
  - GROUP BY spalte1, spalte2
- **Wir dürfen aber auch dem Ergebnis einer Funktion gruppieren:**
  - MySQL: GROUP BY YEAR(spalte)
  - PostgreSQL: GROUP BY DATE\_PART('year', spalte)
- **Zudem:** Wir dürfen GROUP BY auch mit einem (oder mehreren) JOINS kombinieren



# Rechnen mit Zeit: Datumswerte in SQL

SQL Masterclass





# Rechnen mit Datumswerten

- **Wie speichern wir z.B. einen Arzttermin in der Datenbank?**
  - **Möglichkeit 1:**
    - Unix-Timestamp
  - **Möglichkeit 2:**
    - Spezieller Datentyp von unserer Datenbank



# Was ist ein Unix-Timestamp?

- **Unix-Timestamp:**

- Eine Zahl, die ein Datum repräsentiert
- Idee: Wir starten irgendwann (hier: 1.1.1970), und zählen einfach die Sekunden\*
- Damit können wir ein Datum als einfache Zahl speichern
- Beispiel: 1571657687 = 21.10.2019, 11:34:47 UTC
- Wir können Unix Timestamps leicht vergleichen:
  - Zeitdifferenz in Sekunden: Einfach voneinander subtrahieren



# Nachteile von Unix-Timestamps

- **Unix-Timestamp:**

- Bei 32 Bit (INTEGER-Datentyp) können wir nur Datumswerte bis zum 19.1.2038 um ca. 3 Uhr darstellen
- Das reicht aktuell noch i.d.R. aus
- Aber was tun wir, wenn die Anwendung 25+ Jahre funktionieren soll?



# Datumswerte in der Datenbank abspeichern

- **Alternative: Datum als Datumswert in Datenbank speichern**
- **Hier gibt es verschiedene Möglichkeiten:**
  - Uhrzeit
  - Uhrzeit mit Zeitzone
  - Datum
  - Datum mit Uhrzeit
  - Datum mit Uhrzeit & Zeitzone
- Diese werden wir uns jetzt pro Datenbanksystem separat anschauen!



# MySQL: Datumswerte

SQL Masterclass



# MySQL: Datumsangaben speichern

- **MySQL unterstützt verschiedene Typen:**
  - YEAR: Speichert nur das Jahr
  - TIME: Speichert eine Uhrzeit (oder ein Zeitdauer)
  - DATE: Speichert ein Datum (ohne Uhrzeit)
  - TIMESTAMP / DATETIME: Speichern ein Datum mit Uhrzeit



# MySQL: TIMESTAMP vs. DATETIME

- **TIMESTAMP:**

- Wird beim Speichern aus der lokalen Zeitzone nach UTC umgewandelt
- Wird beim Auslesen wieder zurück in die lokale Zeitzone der Anwendung umgewandelt

- **DATETIME:**

- Wird weder beim Speichern noch beim Auslesen umgewandelt
- Hier muss die Anwendung also wissen, welche "Zeitzone" hier verwendet wird!



# MySQL: Wie mit Datumswerten rechnen?

SQL Masterclass





# MySQL: Mit Datumswerten rechnen

- NOW(): Gibt die aktuelle Uhrzeit aus
- UTC\_TIMESTAMP(): Gibt die aktuelle UTC-Uhrzeit aus
- YEAR(timestamp), MONTH(timestamp), DAY(timestamp):
  - Gibt das Jahr bzw. Monat bzw. Tag von einem Datumswert aus
- DATE\_ADD(timestamp, interval) / DATE\_SUB:
  - Addiert einen Zeitraum auf ein Datum
- DATE\_FORMAT(timestamp, format):
  - Gibt ein Datum formatiert aus
- DATEDIFF



# MySQL: DATE\_FORMAT

- DATE\_FORMAT(timestamp, format):
  - Im Parameter format kann das Format angegeben werden:
  - <https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>



# PostgreSQL: Datumswerte

SQL Masterclass



# PostgreSQL: Datumsangaben speichern

- **PostgreSQL unterstützt verschiedene Typen:**
  - **TIMESTAMP / TIMESTAMP WITH TIME ZONE**
    - Datumsangabe (inkl. Uhrzeit)
  - **DATE:**
    - Datumsangabe (ohne Uhrzeit)
  - **TIME / TIME WITH TIME ZONE:**
    - Uhrzeitangabe
    - Wichtig: Unterscheidung nach Sommer-/Winterzeit nicht möglich!



# TIMESTAMP vs. TIMESTAMP WITH TIME ZONE

- **TIMESTAMP:**
  - Wird 1:1 so abgespeichert wie angegeben
  - Es findet keine Umrechnung statt
- **TIMESTAMP WITH TIME ZONE:**
  - Wird beim Abspeichern in UTC umgewandelt
  - Wird beim Auslesen von UTC zurück in die lokale Zeitzone umgewandelt



# PostgreSQL: Mit Datumswerten rechnen

SQL Masterclass



# PostgreSQL: Mit Datumswerten rechnen

- CURRENT\_TIMESTAMP, LOCALTIMESTAMP:
  - Gibt die aktuelle Uhrzeit aus
- DATE\_PART(part, timestamp): Gibt einen Teil vom Datum aus
  - z.B. das Jahr, den Monat,...+
- Mit Datumswerten rechnen:
  - `SELECT timestamp '2020-01-01 00:00:00' - timestamp '2019-08-20 00:00:00'`
  - `SELECT timestamp '2019-01-01 00:00:00' + interval '2 days'`



# Indexe: Wie funktionieren sie?

SQL Masterclass





## Indexe: Warum brauchen wir sie?

- Bisher werden bei jedem Filtern die gesamten Daten durchsucht
- Das ist ausgesprochen langsam
- Mit einem Index können wir dies massiv beschleunigen
- $O(n) \Rightarrow O(\log n)$
- Aber wie funktioniert ein Index genau?



# Fremdschlüssel

SQL Masterclass



# Fremdschlüssel: Motivation

- Mit einem Fremdschlüssel können wir in SQL direkt eine Beziehung "validieren"...
- **Beispiel:**
  - Tabelle customers, Spalte ID: Jeder Kunde hat eine eindeutige ID
  - Tabelle orders, Spalte customer\_id: Jede Bestellung kann einem Kunden zugeordnet sein
- **Was wir vermeiden möchten:**
  - Bestellung mit customer\_id = 500, aber es gibt keinen Kunden mit der id = 500!



# Fremdschlüssel

- Genau diese Prüfung kann die Datenbank für uns übernehmen!
- ```
CREATE TABLE orders (  
    ...  
    customer_id bigint,  
    FOREIGN KEY (customer_id) REFERENCES customers(id)  
)
```
- Oder wir fügen diese Prüfung nachträglich hinzu:
 - ```
ALTER TABLE orders ADD FOREIGN KEY (customer_id)
 REFERENCES customers(id)
```



# Fremdschlüssel - Teil 2

SQL Masterclass



## Fremdschlüssel: ON UPDATE

- Wie gehen wir damit um, sollte sich eine Kunden-ID nachträglich ändern?
  - ON UPDATE RESTRICT:
    - ALTER TABLE orders ADD FOREIGN KEY (customer\_id) REFERENCES customers(id) ON UPDATE RESTRICT
    - Hiermit wird verboten, dass in der Tabelle customers die id nachträglich geändert wird (wenn es eine Bestellung von dem Kunden gibt)



## Fremdschlüssel: ON UPDATE

- Wie gehen wir damit um, sollte sich eine Kunden-ID nachträglich ändern?
  - ON UPDATE SET NULL:
    - ALTER TABLE orders ADD FOREIGN KEY (customer\_id) REFERENCES customers(id) ON UPDATE SET NULL
    - Wenn in der Tabelle customers zu einem Eintrag die id nachträglich geändert wird, wird in der Tabelle orders zu den entsprechenden Einträgen die customer\_id auf NULL gesetzt



## Fremdschlüssel: ON UPDATE

- Wie gehen wir damit um, sollte sich eine Kunden-ID nachträglich ändern?
  - ON UPDATE CASCADE:
    - ALTER TABLE orders ADD FOREIGN KEY (customer\_id) REFERENCES customers(id) ON UPDATE CASCADE
    - Wenn in der Tabelle customers zu einem Eintrag die id nachträglich geändert wird, wird die customer\_id in der orders-Tabelle auch entsprechend angepasst





## Fremdschlüssel: ON DELETE

- Wie gehen wir damit um, wenn ein Kunde gelöscht wird?
  - ON DELETE RESTRICT:
    - Wenn es Bestellungen gibt, darf der Kunde nicht gelöscht werden
  - ON DELETE SET NULL:
    - Die customer\_id wird auf NULL gesetzt
  - ON DELETE CASCADE:
    - Die entsprechenden Bestellungen werden auch gelöscht



# Fremdschlüssel: Die gesamte Query

- **Generell gilt:**

- Der Datentyp muss 1:1 übereinstimmen
- ALTER TABLE orders

ADD FOREIGN KEY (customer\_id) REFERENCES

customers(id)

ON UPDATE CASCADE

ON DELETE SET NULL



# Wichtige Unterschiede

- Tabelle: orders, Spalte customer\_id
- **MySQL:**
  - Wenn wir den Fremdschlüssel hinzufügen, wird automatisch auch ein Index erstellt (sofern er nicht existiert)
  - Dadurch werden die ON UPDATE / ON DELETE-Klauseln effizient ausgeführt
- **PostgreSQL:**
  - Ein Index wird für uns nicht automatisch erstellt
  - Er ist aber i.d.R. empfehlenswert (für die Performance) - wir sollten ihn also manuell anlegen!



# Komplexe Abfragen speichern - Views!

SQL Masterclass



# Views

- Views sind eine Art “virtuelle Tabelle”
- Wir können eine Query “abspeichern”, und als eine “virtuelle Tabelle” verwenden
- Wenn wir dann diese Tabelle abfragen, wird unsere ursprünglich abgespeicherte Query ausgeführt



# Views: Anwendung

- Komplexe Queries können so vereinfacht werden
  - => Achtung, nur für uns!
  - => Nicht für die Datenbank - die muss immer noch die komplexe Query ausführen
- Migration zu einer neuen Struktur:
  - Beispiel: Eine Anwendung erwartet, dass die Spalte für den Nachnamen "surname" heißt
  - In der neuen Anwendung heißt die Spalte aber "lastname"
    - Wir können einen View erstellen, der die Spalte für die alte Anwendung "umbenennt"



# Views: Limitierungen

- UPDATE und INSERT wird nur eingeschränkt unterstützt
- Die Abfragen können nur mit Indexes beschleunigt werden, die auf den ursprünglichen Tabellen angelegt wurden
- Mit Views wird eine für die Datenbank komplexe Query "versteckt"
- Sonderfall:
  - PostgreSQL - Materialized View



# Transaktionen

SQL Masterclass





# Transaktionen

- Wir können in SQL mehrere Anfragen direkt hintereinander ausführen:
  - UPDATE ...; UPDATE ...;
- Jetzt könnte es aber sein, dass die Datenbank zwischen 2 Queries abstürzt
- Das möchten wir vermeiden:
  - Wir möchte, dass entweder alle Updates durchlaufen
  - Oder gar keine Änderung übernommen wird



# Transaktionen

- Genau dafür gibt es Transaktionen:
  - `START TRANSACTION`
  - `UPDATE ...;`
  - `COMMIT`
- Wir können auch nachträglich eine Transaktion abbrechen
  - `START TRANSACTION`
  - `UPDATE ...;`
  - `ROLLBACK`



# Locking

SQL Masterclass



# SELECT ... FOR UPDATE

- Mit einem FOR UPDATE kann eine Transaktion den Lesezugriff auf einen Datensatz blockieren!
- Dieser Lesezugriff ist geblockt, bis die Transaktion abgeschlossen wurde (COMMIT / ROLLBACK)
- Dadurch können wir bestimmtes Problem verhindern!



```
graph TD; A[START TRANSACTION] --> B[SELECT balance FROM accounts WHERE name = 'max']; B --> C[UPDATE accounts SET balance = balance - 50 WHERE name = 'max']; C --> D[COMMIT];
```

START TRANSACTION

SELECT balance FROM accounts  
WHERE name = 'max'

UPDATE accounts SET balance =  
balance - 50 WHERE name = 'max'

COMMIT

```
graph TD; A[START TRANSACTION] --> B[SELECT balance FROM accounts WHERE name = 'max']; B --> C[UPDATE accounts SET balance = balance - 50 WHERE name = 'max']; C --> D[COMMIT];
```

START TRANSACTION

SELECT balance FROM accounts  
WHERE name = 'max'

UPDATE accounts SET balance =  
balance - 50 WHERE name = 'max'

COMMIT



```
graph TD; A[START TRANSACTION] --> B[SELECT balance FROM accounts WHERE name = 'max']; B --> C[UPDATE accounts SET balance = balance - 50 WHERE name = 'max']; C --> D[COMMIT];
```

START TRANSACTION

SELECT balance FROM accounts  
WHERE name = 'max'

UPDATE accounts SET balance =  
balance - 50 WHERE name = 'max'

COMMIT

```
graph TD; A[START TRANSACTION] --> B[SELECT balance FROM accounts WHERE name = 'max']; B --> C[UPDATE accounts SET balance = balance - 50 WHERE name = 'max']; C --> D[COMMIT];
```

START TRANSACTION

SELECT balance FROM accounts  
WHERE name = 'max'

UPDATE accounts SET balance =  
balance - 50 WHERE name = 'max'

COMMIT



```
graph TD; A[START TRANSACTION] --> B[SELECT balance FROM accounts WHERE name = 'max' FOR UPDATE;]; B --> C[UPDATE accounts SET balance = balance - 50 WHERE name = 'max']; C --> D[COMMIT];
```

START TRANSACTION

SELECT balance FROM accounts  
WHERE name = 'max' **FOR UPDATE;**

UPDATE accounts SET balance =  
balance - 50 WHERE name = 'max'

COMMIT

```
graph TD; A[START TRANSACTION] --> B[SELECT balance FROM accounts WHERE name = 'max' FOR UPDATE; => Neuer Kontostand wird zurückgegeben!];
```

START TRANSACTION

SELECT balance FROM accounts  
WHERE name = 'max' **FOR UPDATE;**

=> Neuer Kontostand wird  
zurückgegeben!



# SELECT ... FOR UPDATE

- SELECT ... FOR UPDATE:
  - Blockiert andere UPDATES auf die Daten
  - Blockiert Lesezugriffe, sofern die andere Query auch ein SELECT ... FOR UPDATE enthält





## Locking (2)

SQL Masterclass



# Locking

- Es gibt jetzt hier noch unzählige Nuancen:
  - Schreib-Lock:
    - FOR UPDATE
  - Lese-Lock:
    - FOR SHARE / LOCK IN SHARED MODE
  - Tabellen-Locking
  - Locks ignorieren
- Weitere Themen:
  - Deadlocks



# Locking

- Wenn dich das weiter interessiert:
  - Suche nach:
    - “Deadlock”
    - <https://dev.mysql.com/doc/refman/8.0/en/innodb-locking-reads.html>



# Locking

- Es gibt jetzt hier noch unzählige Nuancen:
  -



# Rechteverwaltung

SQL Masterclass



# Rechteverwaltung

- Bisher durfte unser Nutzer immer alles
  - Administrator / Root
- Wenn wir jetzt aber eine Anwendung schreiben:
  - Benötigt dieser Benutzer nicht so viele Rechte
  - Z.B. müssen Tabellen (oft) nicht mehr nachträglich abgeändert werden
  - Dem Benutzer können wir also diese Rechte entziehen



# Rechteverwaltung

- Zudem können wir Informationen vor unserer Anwendung “verstecken”:
  - Beispielsweise könnten wir Kreditkartendaten für die Anwendung ausblenden wollen
  - Diese sind dann nur über eine “Stored Procedure” zugänglich, und hierfür gibt es zusätzliche Plausibilitätschecks / Sicherheitschecks



# Rechteverwaltung


- Wobei der Sicherheitsaspekt nicht immer so wichtig ist:
  - Wenn ein Angreifer es schafft, unseren (Web-)Server zu hacken, kann er über kurz oder lang oft eh auf alle Daten zugreifen
  - Dennoch ist es eine zusätzliche Hürde, die uns mehr Zeit einräumen kann!





# Rechteverwaltung

- MySQL und PostgreSQL unterscheiden sich hier bei ihrer Rechteverwaltung
- Dieser Kurs “teilt” sich also wieder auf!



# Stored Procedures / Stored Functions

SQL Masterclass



# Stored Procedures

- Erlauben es dir, eigene Programme zu schreiben
- Diese werden direkt in der Datenbank ausgeführt
- Dadurch können wir auch komplexe Anwendungslogik direkt in der Datenbank ausführen
- Die Rechte können ein Stück weit unabhängig festgelegt werden



# Stored Procedures

- Signifikante Unterschiede zwischen PostgreSQL und MySQL
- Wir schauen uns das hier also separat an!

A decorative graphic in the top-left corner consisting of two overlapping parallelograms. The front one is blue and the back one is light green. Both are tilted at an angle.

# Anwendungslogik in Stored Procedure?

SQL Masterclass



## Stored Procedure...

- Kein Overhead (z.B. PHP -> MySQL -> PHP)
- Kann direkt auf Daten zugreifen
- Ein Befehl kann mehrere Aktionen ansteuern (z.B. Log-Eintrag schreiben, Daten zurückgeben,...)
- Das kann sehr komfortabel sein



# Stored Procedure...

- **Aber:**

- Das Testen von Stored Procedures ist oft aufwendiger
- Für die Anwendung existiert oft schon ein Testing-Framework
- Man ist gefesselt an ein Datenbanksystem
- Stored Procedures unterscheiden sich sehr zwischen MySQL und PostgreSQL
- Oft ist das Aufwendige eher das Zusammensammeln der Daten (z.B. komplexe GROUP BYs)
- Das kann auch eine Procedure nicht beschleunigen



Volltextsuche

SQL Masterclass





# Volltextsuche

- **Problem:**

- Das Durchsuchen von vielen Dokumenten dauert oft sehr lange
- Es gibt viele "nutzlose" Wörter, die nicht viel mit dem Inhalt zu tun haben: der, die, das, ist, ...
- **Zudem: Plural-Formen müssen in Singular-Formen umgewandelt werden (Stemming):**
  - Bücher -> Buch
  - Datenbanken -> Datenbank



# Volltextsuche

- Volltextsuche ist also kein “triviales” Problem
- Wir können eine rudimentäre Volltextsuche auch direkt in MySQL oder PostgreSQL ausführen
- **Wichtig:**
  - Wenn wir mehr Kontrolle über den Prozess benötigen...
  - ... Hadoop, Spark, Elasticsearch,...
  -



# Volltextsuche

- **Zudem:**

- MySQL und PostgreSQL unterscheiden sich in der Schreibweise und den Features sehr
- Der Kurs teilt sich hier also auf

- **Generell gilt:**

- PostgreSQL bietet hier ein paar mehr Optionen
- Dafür ist das Ansteuern etwas komplizierter

A blue parallelogram and a light green parallelogram are positioned in the upper-left corner of the slide. The blue shape is partially behind the green one. Both shapes are oriented diagonally, matching the overall geometric theme of the slide.

Trigger

SQL Masterclass



# Trigger

- **Problem:**

- Manchmal möchten wir, dass Dinge vor / nach einem Update / Insert / Delete automatisch passieren
- Beispiel:
  - Eine Überweisung wird in die entsprechende Tabelle geschrieben
  - Der Kontostand vom Kunden soll automatisch aktualisiert werden



# Trigger

- Mit Triggern können wir.
  - Code ausführen
  - Zeitpunkt:
    - Vor / Nach
    - UPDATE / DELETE / INSERT



# Trigger

- MySQL und PostgreSQL bieten beide die Möglichkeit für Trigger
- Allerdings unterscheidet sich die Schreibweise sehr
- => Wir gehen daher auf beide Datenbanksysteme separat ein!



# Constraints

SQL Masterclass





# Constraints

- Mit Constraints können Daten automatisch validiert werden
- Beispiel:
  - Der Titel muss mindestens 3 Zeichen lang sein
- **Aber:**
  - Wo gehört diese Validierung hin?
  - Datenbank?
  - Anwendungslogik?



# Constraints

- Werden von PostgreSQL nativ unterstützt
- MySQL unterstützt keine Constraints (nur Foreign Key Constraints)
  - Aber wir können über Trigger ein ähnliches Verhalten erzeugen!



Schlussworte

SQL Masterclass



Schlussworte:  
Vielen Dank, dass ich dir SQL  
beibringen durfte!

SQL Masterclass



## Du hast jetzt einiges gelernt...

- Von einfachen Queries...
- ... bis hin zu komplexen Abfragen,
- Indexes, Fremdschlüssel,
- Volltextsuche,
- Stored Functions / Procedures, Trigger,...



## Wie geht es jetzt weiter?

- Du beherrscht jetzt SQL als Abfragesprache
- **Aber:**
  - I.d.R. entwickelst du ja eine Anwendung, die SQL nutzt
  - Du könntest dir dort noch die Tools anschauen, wie du möglichst komfortabel die Datenbank ansteuern kannst
  - **Beispiel:** Object Relational Mapper