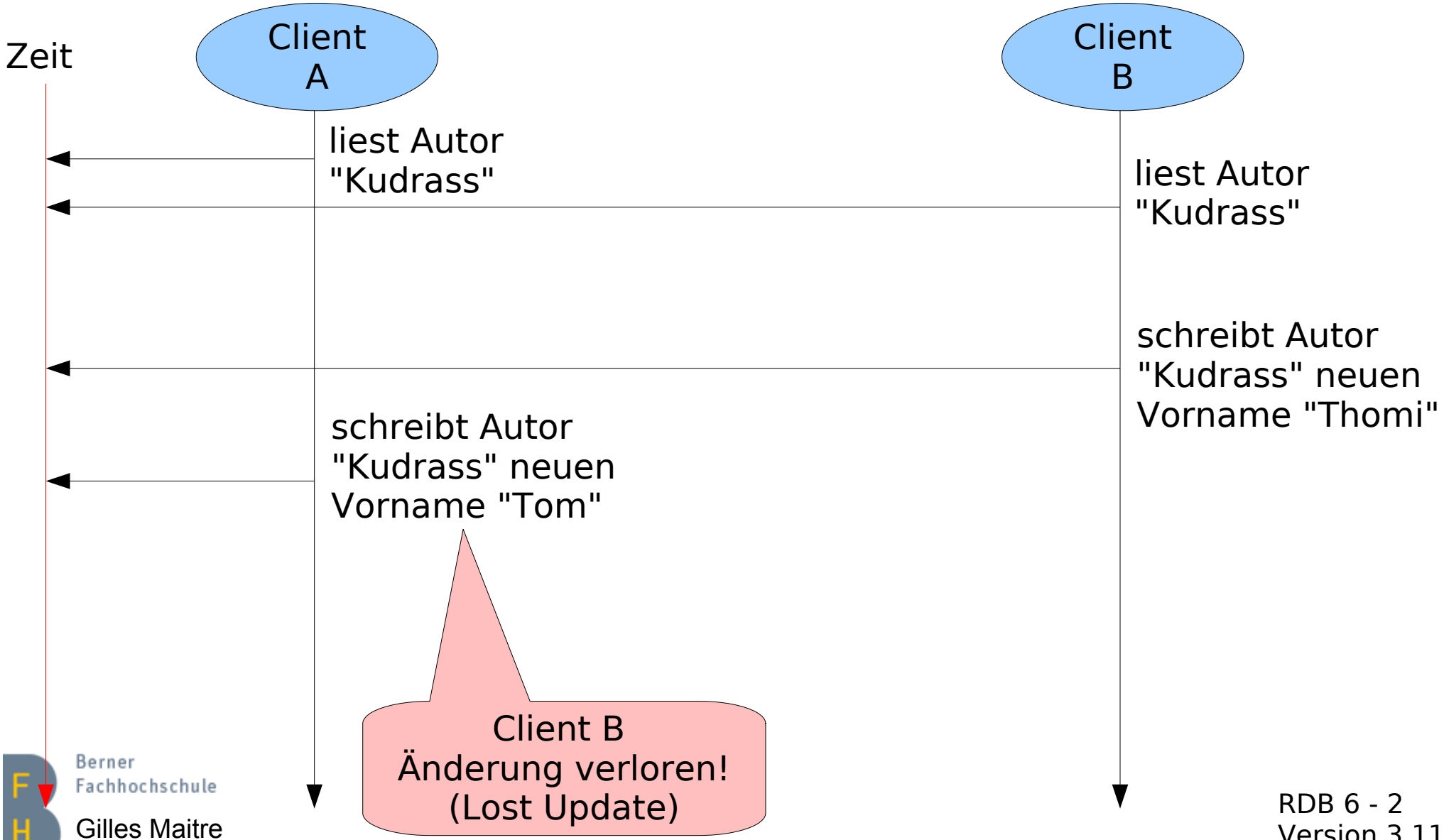
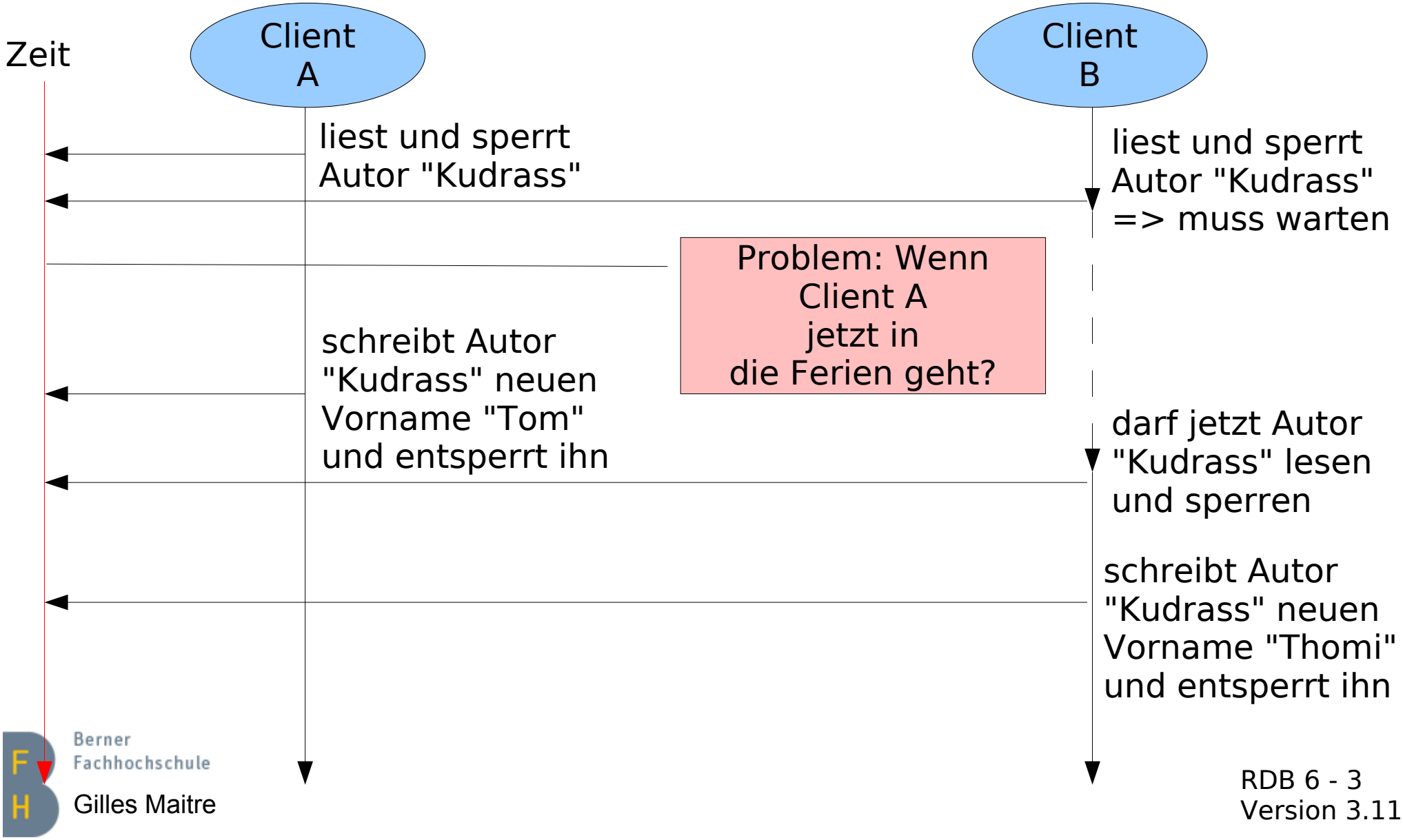
Tag 6
Inhaltsverzeichnis

- Zwei einfache Sperrverfahren in Java
  - Pessimitic, optimistic Locking
- Performance
  - System, DB, Applikation
- DB Zugriffsberechtigungen
- SQL Injection, Gefahr und Gegenmassnahmen
- Metadaten
- Views
- Stored Programs
  - Stored Procedures (SP)
  - Triggers
- Die Wikipedia Architecture
- Übungen

Zwei einfache Sperrverfahren
Problem wenn keine Synchronisation...



Zwei einfache Sperrverfahren
Pessimistic Locking



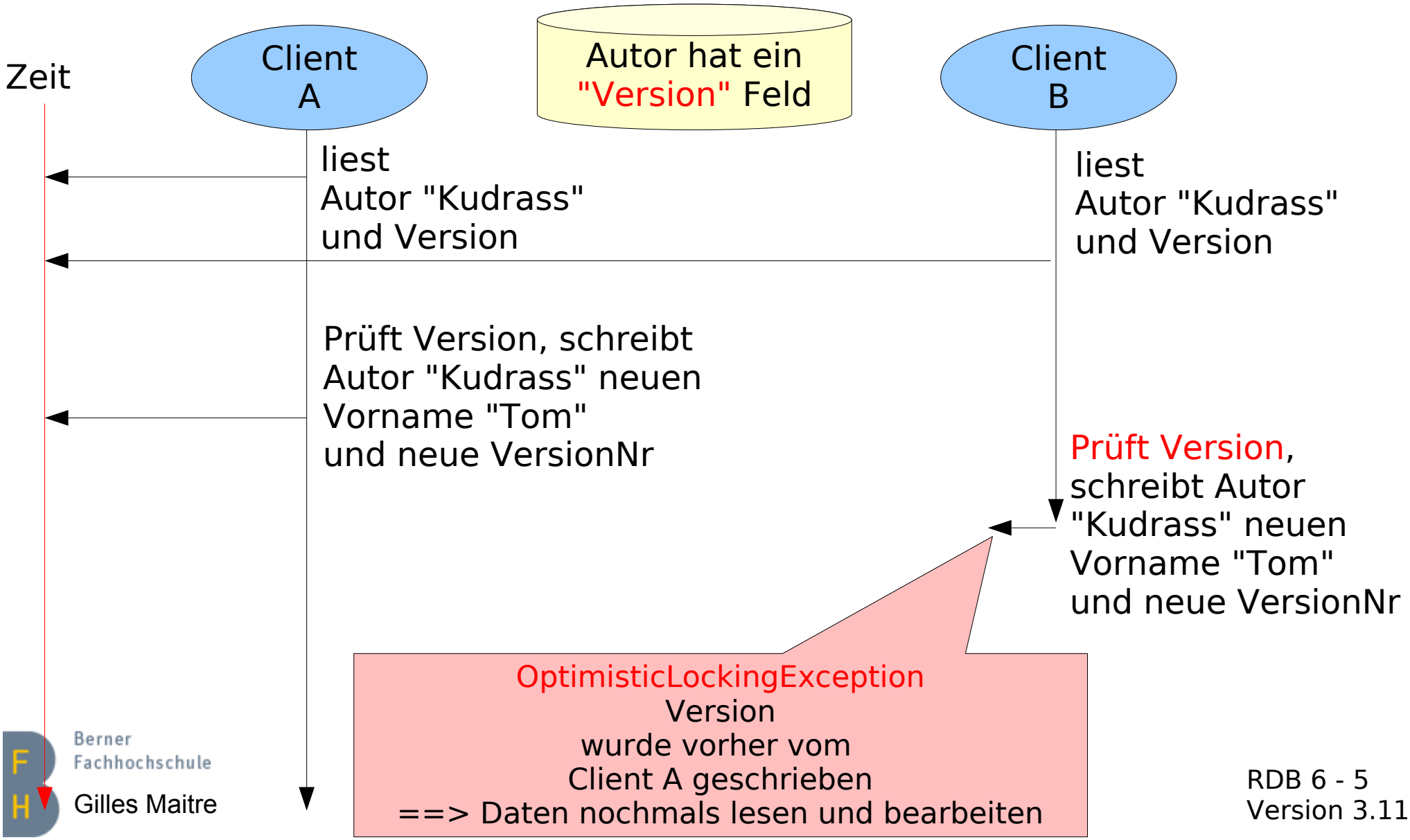
Pessimistic Locking
Einfache Implementation mit JDBC

```
// Annahme:
// In der Variable "autor"
// (vom Typ Autor, mit den Feldern PersNr, Name und Vorname)
// habe ich vorher den Autor "Kudrass" gelesen
connection.setAutoCommit(false);
String selectQuery =
    "SELECT * FROM Autor WHERE PersNr=? FOR UPDATE";
PreparedStatement selectStm = connection.prepareStatement(selectQuery);

// Select for update ausführen
selectStm.setString(1, autor.getPersNr());
ResultSet rs = selectStm.executeQuery();
// Hier muss ich warten, falls jemand diese Zeile schon gelockt hat...
// Dann
// Update vorbereiten
String updateQuery =
    "UPDATE Autor SET Vorname=? WHERE PersNr=?";
PreparedStatement updateStm = connection.prepareStatement(updateQuery);

// Update ausführen
updateStm.setString(1, autor.getVorname());
updateStm.setString(2, autor.getPersNr());
int nbrOfModifiedRecords = updateStm.executeUpdate();
connection.commit();
connection.setAutoCommit(true);
```

Zwei einfache Sperrverfahren
Optimistic Locking



Optimistic Locking
Einfache Implementation mit JDBC

```
// Annahme:
// In der Variable "autor"
// (vom Typ Autor, mit den Feldern PersNr, Version, Name und Vorname)
// habe ich vorher den Autor "Kudrass" gelesen

// Update vorbereiten
String updateQuery =
    "UPDATE Autor SET Vorname=?, Version=? WHERE PersNr=? AND Version=?";
PreparedStatement updateStm = connection.prepareStatement(updateQuery);

// Update ausführen
updateStm.setString(1, autor.getVorname());
updateStm.setInt(2, autor.getVersion() + 1); // Version wird inkrementiert
updateStm.setInt(3, autor.getPersNr());
updateStm.setInt(4, autor.getVersion()); // Meine Ursprungs-Version

// Klappt nur wenn kein Update von "Kudrass" in der Zwischenzeit
int nbrOfModifiedRecords = updateStm.executeUpdate();
if (nbrOfModifiedRecords == 0) {
    throw new OptimisticLockingException();
}
```

Pessimistic und Optimistic Locking
Anwendungsfälle

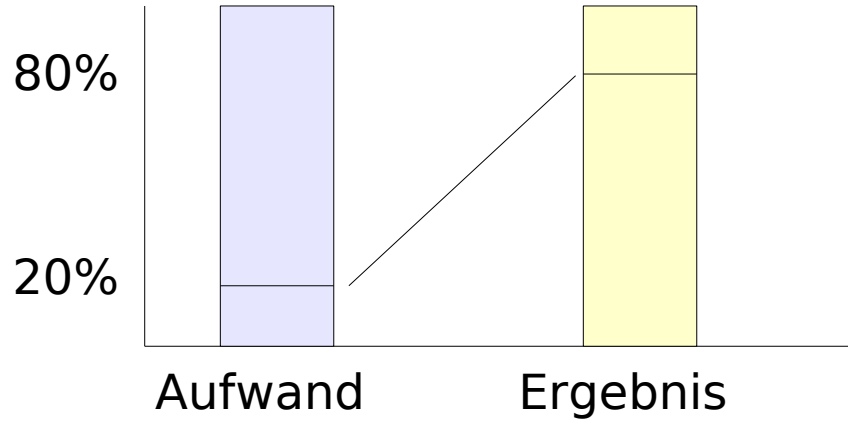
- Wenn... wenig Zeilen eine kurze absehbare Zeit gesperrt werden müssen und kleines Codefragment
  → Pessimistic Locking genügt

- Wenn... diverse Zeilen und/oder Dauer der Sperrung nicht absehbar (z.B. GUI Client/Server Applikation)
  → Optimistic Locking besser geeignet

- Zur Info: Ab Oracle 10g, Optimistic Locking vereinfacht dank "ora_rowscn" Pseudo-Column

  http://robertgfreeman.blogspot.com/2005/06/orarowscn-new-10g-pseudo-column.html

Performance
Das Pareto-Prinzip (80-zu-20-Regel)



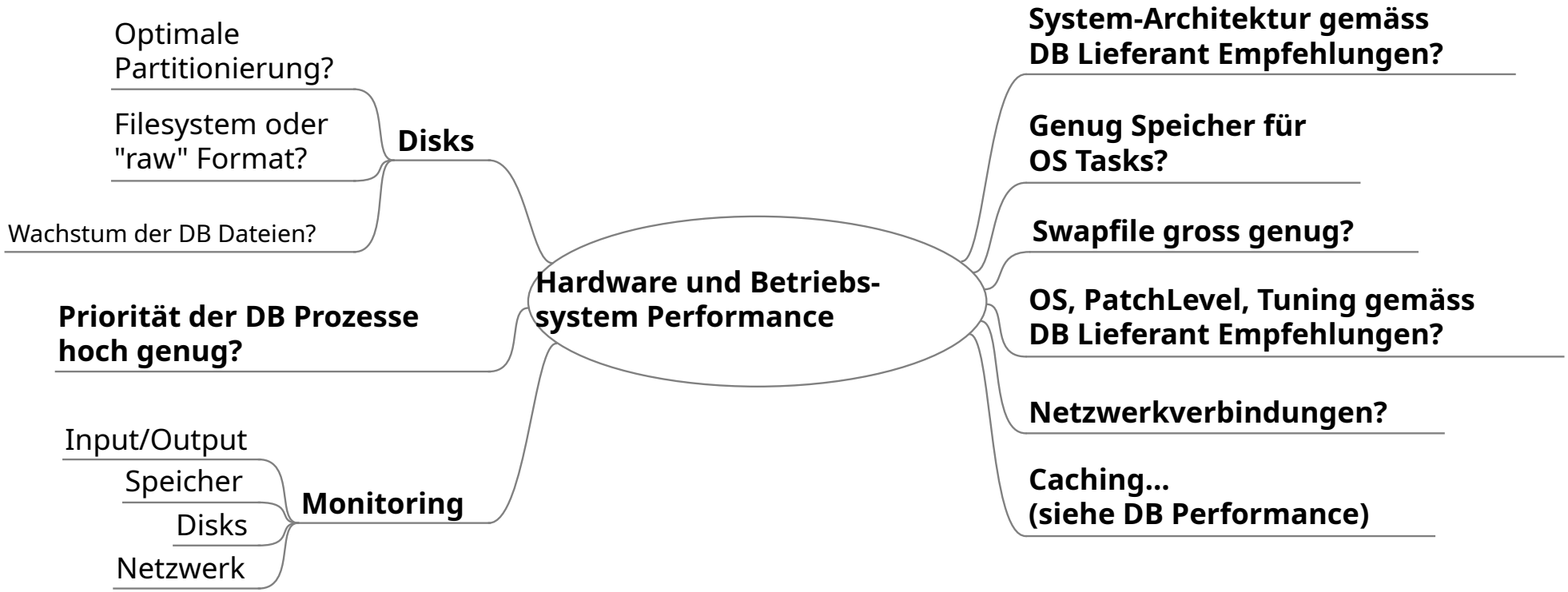
- Im Durchschnitt erzeugen 20% der Kunden einer Firma 80% des Umsatzes

  => Firma muss sich *vor allem* um diese 20% kümmern

- Im Computer-Bereich...

  => Nur optimieren, wo es sich lohnt!

System Performance
Hardware und Betriebssystem



DB Performance
Indexierung

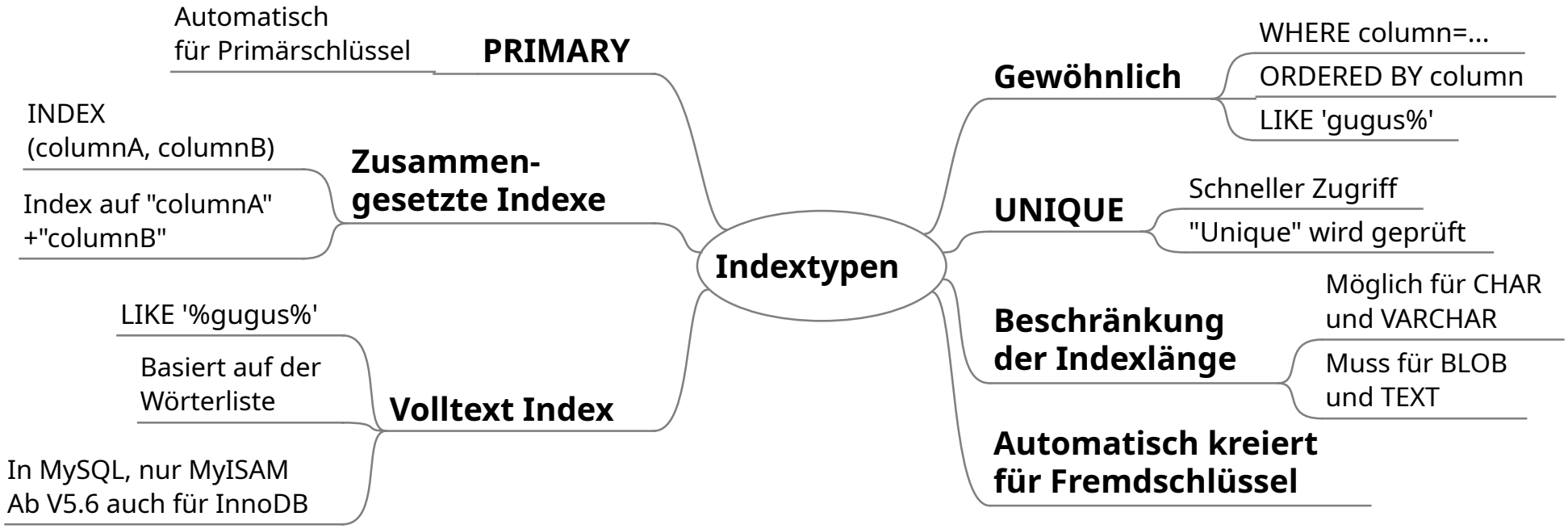
- Extra Daten, um Zugriff auf DB Inhalt zu optimieren
  - Werte in Spalten
  - Joins
  - Sortierung / Aggregation

- Pro: Datenzugriff *schneller*
  Kontra: Datenänderungen *aufwendiger*

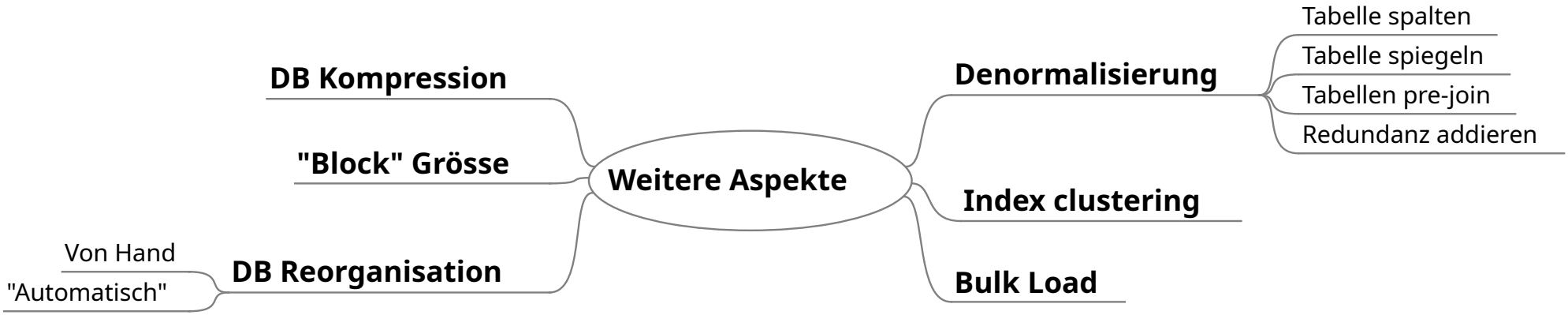
  => Kompromiss *suchen*

- Implementation
  - B-Bäume
  - Hashtabellen (*nicht implementiert in MySQL V5*)

DB Performance (1)
Indextypen



DB Performance (2)
Weitere DB-Aspekte



DB Performance
Indexeinschränkungen

Indexe werden **nicht** verwendet, wenn

- Index auf Spalte mit vielen ähnlichen Werten, z.B. boolean
- Funktionsaufruf im Ausdruck
  z.B.: **`where date(column)...`**
- Beim Join: Ungleicher Typ für Primär- und Fremdschlüssel
- ORDER BY + andere Kriterien
- ...

*==> Das sollte man **überprüfen**, mit dem EXPLAIN-Befehl (wird später erklärt)*

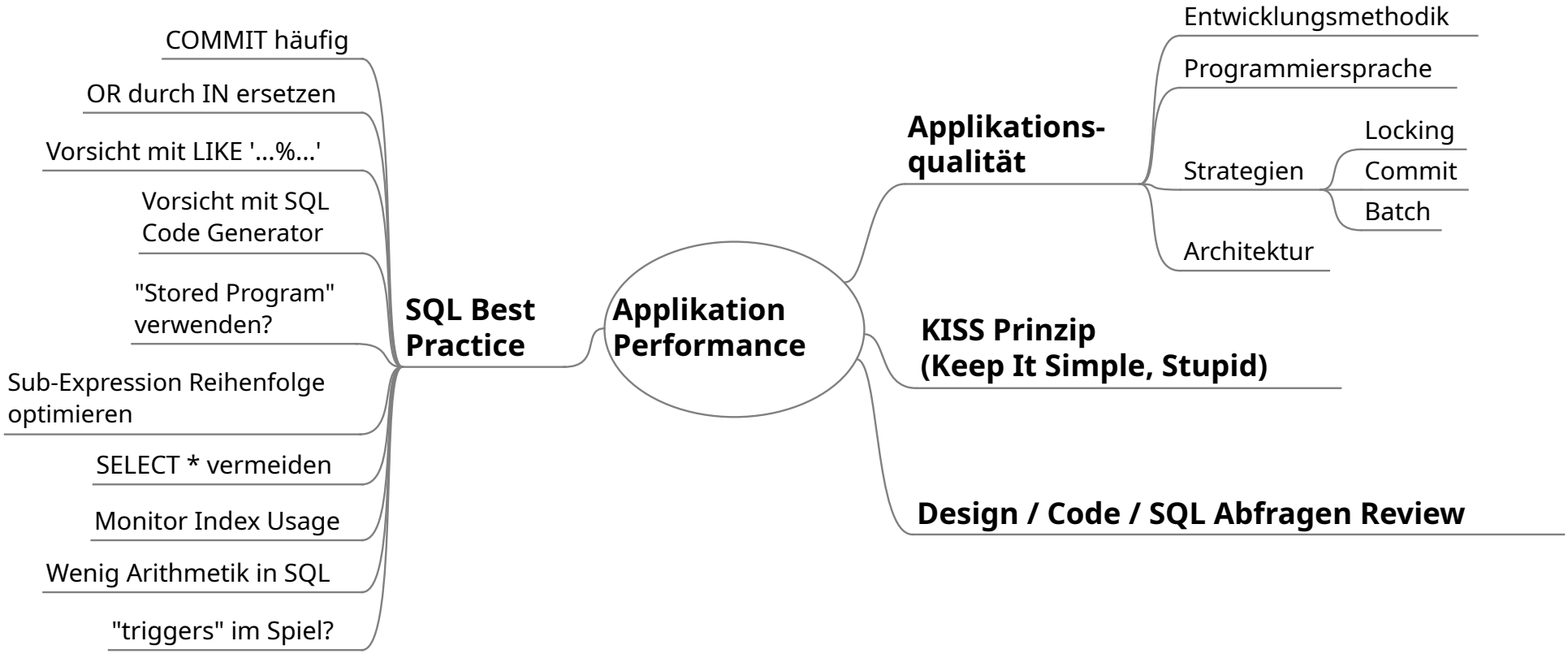
DB Performance
SQL Indexbefehle

```
-- Erstellen
mysql> create [unique] index index_name
       on <table> (<column>,...)

-- Anzeigen
mysql> show index from <table>;

-- Löschen
mysql> drop index <index> on <table>;
```

Applikation Performance
Ein paar Aspekte



Applikation Performance
MySQL Monitoring

- Befehl "SHOW ENGINE INNODB STATUS\G" verwenden
  (*siehe MySQL Dokumentation*)
  - => Detailinfos über InnoDB Engine
- Befehl EXPLAIN SELECT... verwenden
  - => Detailinfos über SELECT Statement(Indexverwendung, ...)
- SHOW FULL PROCESSLIST
- Analyse der Logdatei
- Warten auf Tag 7 ;-)

DB Zugriffsberechtigungen
SQL GRANT (vereinfacht)

grant <priviledgeList>
on <database>.<table>
to <user>@<host> [identified by 'password']

```
mysql> grant Select, Insert on MeineCDs.* to gmaitre@'%';

mysql> grant All on *.* to ''@localhost; -- never do that
```

DB Zugriffsberechtigungen
SQL REVOKE (vereinfacht)

revoke <priviledgeList>
on <database>.<table>
from <user>@<host> [identified by 'password']

```
mysql> revoke Select, Insert on MeineCDs.* from gmaitre@'%';

mysql> revoke All on *.* from admin@'%';  -- never do that
```

MySQL Zugriffsberechtigungen
SQL SHOW GRANTS

```
mysql> show grants for gmaitre@'%'\G

*************************** 1. row ***************************
Grants for gmaitre@%: GRANT USAGE ON *.* TO 'gmaitre'@'%' IDENTIFIED BY PASSWORD
'*A170434C682FB67394D0F469BB2236F6B62F3A85'
*************************** 2. row ***************************
Grants for gmaitre@%: GRANT ALL PRIVILEGES ON `MeineCDs`.* TO 'gmaitre'@'%'
WITH GRANT OPTION
*************************** 3. row ***************************
Grants for gmaitre@%: GRANT ALL PRIVILEGES ON `AutorBuch`.* TO 'gmaitre'@'%'
WITH GRANT OPTION
```

SQL Injection
Gefahr und Gegenmassnahmen

- **Was ist das?**
  Manipulation von SQL-Anweisungen, durch Parameter, die *bösartigen SQL-Code enthalten*

- **Prinzip und Beispiele**
  - https://de.wikipedia.org/wiki/SQL-Injection
  - https://de.wikipedia.org/wiki/SQL-Injection#Vorgang
  - https://xkcd.com/327/

- **Gegenmassnahmen**
  - Programmatische Überprüfung der Eingaben
  - Systematisches Code-Review
    (die das Problem angehen! => Checklisten verwenden)
  - Verwendung von SQL Prepared-Statements

Metadaten
Anwendung in DB

- Siehe RDB Tag 5, Slide 21 ;-)
- Metadaten ::= Daten, die die Daten beschreiben
- Beispiel von einem Buch:
  - Daten ::= Buchtext
  - Metadaten ::= Titel, Autor, Verlag, ISBN, Text
- Anwendung für relationale Datenbanken:
  - Daten ::= Tabellen-Schemas und Daten
  - Metadaten ::= Das DataDictionary =>
    Die Daten von MySQL selber
    - Tabellen, ihre Schemas, Zugriffsrechte, ...

MySQL
Metadaten und Grenzen

- Alle Metadaten in DB **information_schema** gespeichert (nur Lesezugriff)
- `mysql> use information_schema` funktioniert
- SHOW & DESCRIBE brauchen diese Information...
- Tabellenstruktur nach ANSI/ISO SQL:2003
- Wichtige Tabellen:
  - TABLES, COLUMNS, VIEWS, TRIGGERS, ROUTINES, ...
- Siehe `"Understanding the MySQL Information Schema Database"`

Views
Was ist das?

- Wirkt wie eine logische Tabelle oder Tabellengruppe
- Basiert (*im Prinzip*) auf einer SELECT Anweisung (oder einer anderen View)
- Spalten und/oder Zeilen können eingeschränkt werden

Views
Was bringt das?

- Eingeschränkter Zugriff auf Spalten und Linien
- Komplexität des Datenzugriffs kann versteckt werden
- Abfragen können wiederverwendet werden
- Tabellen- und/oder Spaltenname können umbenannt werden

Views
Einschränkungen

- Viele Einschränkungen
- Vor allem in INSERT, UPDATE oder DELETE Befehle (und im Bezug zu Joins)
- Im SELECT kein:
  - GROUP BY
  - DISTINCT
  - LIMIT
  - UNION
  - HAVING
  - ...

Quellen:
- Restrictions on Views
- Updatable and Insertable Views

Views
Anwendungen

```
mysql> create view V_AutornameBuchTitel as
       select a.Name, b.Titel
       from Autor a, Autor_Buch ab, Buch b
       where a.PersNr = ab.PersonNr and b.ISBN = ab.ISBN;

mysql> select Name, Titel from V_AutornameBuchTitel;

mysql> show create view V_AutornameBuchTitel;

mysql> rename view v1 to v2;

mysql> drop view v2;        -- nur das View wird gelöscht!
```

Views
Views vs. Materialized views

- MySQL Version 8.* unterstützt das (noch) nicht.
- Sehr guter Überblick hier: Views and Materialized Views in SQL
- Moral der Geschichte / Anwendung:
  - Views: Ideal wenn wenig Datenzugriffe und aktuelle Daten nötig.
  - M-Views: Ideal wenn viele Datenzugriffe und Performance kritisch. Kann aber ziemlich teuer werden, je nach Aktualisierungsgrad und Datenmenge.
- Fragestellung: Wer zahlt für den Zugriff? Der Consumer oder der Producer der Daten?

Stored Programs
Prinzipien (1)

- Parametrierbar und auf dem DB Server *gespeichert*
  - Erlaubt Vorarbeiten...
  - Vermeidung von SQL Code Redundanz...
  - Bessere Sicherheit...
  - Weniger Netzwerkverkehr...
- Ausgeführt auf dem DB Server
- SP Programmiersprache
  - Kontrollanweisungen ("if", "loop", etc...)
  - Variablen
  - "CURSOR"-Elemente (wirkt wie JDBC ResultSet)
- Nachteil: Diverse DBMS Implementationen

Stored Programs
Prinzipien (2)

- Typen
  - Stored Procedures (SP)
  - Triggers

- MySQL: Untermenge von ANSI SQL:2003 SQL/PSM
  - Ähnlich zu Oracle's PL/SQL und SQL Server's Transact-SQL
  - Starke Einschränkungen in der Anwendung

Stored Programs
Triggers, was ist das?

- Ereignis-orientiertes Programmieren in DB
- Automatischer Aufruf eines "Stored Program's" bei einer Veränderung von Daten
- Anwendungsbeispiele
  - DB Integrität garantieren
  - Journal
- Gefahren
  - Nicht voraussehbare *Kettenreaktion* <=> Einfluss auf Performance
  - DB Zustandswechsel schwer zu *debuggen*

Die Wikipedia Architecture
Datenbankperspektive

- Aktuell:
  https://meta.wikimedia.org/wiki/Wikimedia_servers
- Weniger:
  https://meta.wikimedia.org/wiki/File:Wikimedia-servers-2010-12-28.svg

Übungen

1) In der "gmcd" Datenbank, Tabelle "cd", setzten Sie ein Index auf das Feld "datum" und zeigen es an.

2) In der "gmcd" Datenbank, Tabelle "cd", schauen Sie ob Indexes definiert sind und analysieren Sie das Verhalten folgender Abfragen (das Output von explain select ist hier dokumentiert):

```
explain select * from cd where beschreibung = "Dresden"
explain select * from cd where datum = "1968-01-13"
explain select * from cd where datum != "1968-01-13"
explain select * from cd where datum like "19%"
```

3) Mit einem SQL-Befehl basierend auf die Metadaten-DB "information_schema" suchen Sie alle Datenbanken, die auf Ihrem Computer definiert sind und listen Sie die Tabellen der Datenbank "gmcd".