Datenbanken – Lab

SQL – EINFÜHRUNG UND ERSTE ÜBUNGEN

DBMS

Datenbank-System

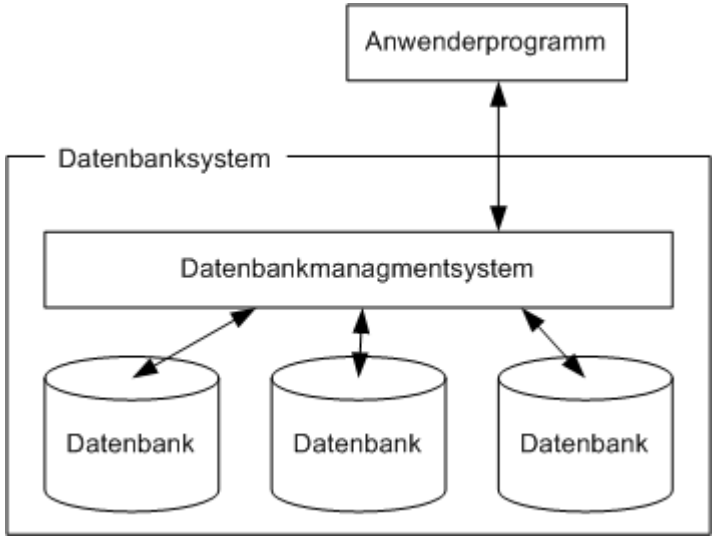
Datenbestand = unstrukturierte Datenbasis

DBMS

+ Regale
+ Ablagesysteme
+ Sortierung

Datenbank-System

Softwareschicht zwischen Daten und Anwendungen



Database Management System (DBMS)

- Software zur effizienten Speicherung, Organisation, Verwaltung und Abfrage von Daten in einer Datenbank
- Schnittstelle zwischen dem Benutzer und der Datenbank
- Gewährleistung der Sicherheit, Konsistenz und Persistenz der Daten
- Hauptaufgaben:
  - Speicherung und Organisation der Daten (Realisierung der Struktur)
  - Bearbeitung von Anfragen (CRUD-Prinzip)
  - Gewährleistung der Datenintegrität
  - Verwaltung der Zugriffsrechte

Vorteile eines DBMS

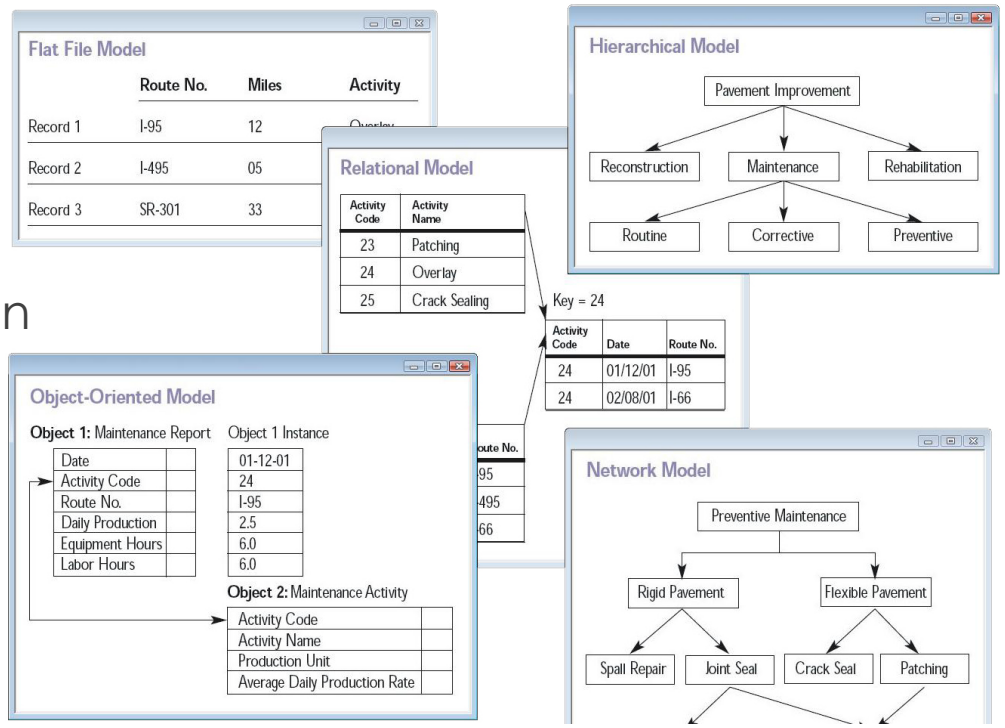
- Zentralisierte Datenverwaltung
- Reduktion von Redundanz (keine doppelte Datenspeicherung)
- Einheitliche Schnittstellen für den Zugriff (z. B. SQL)
- Skalierbarkeit für große Datenmengen

Datenbank-Modell

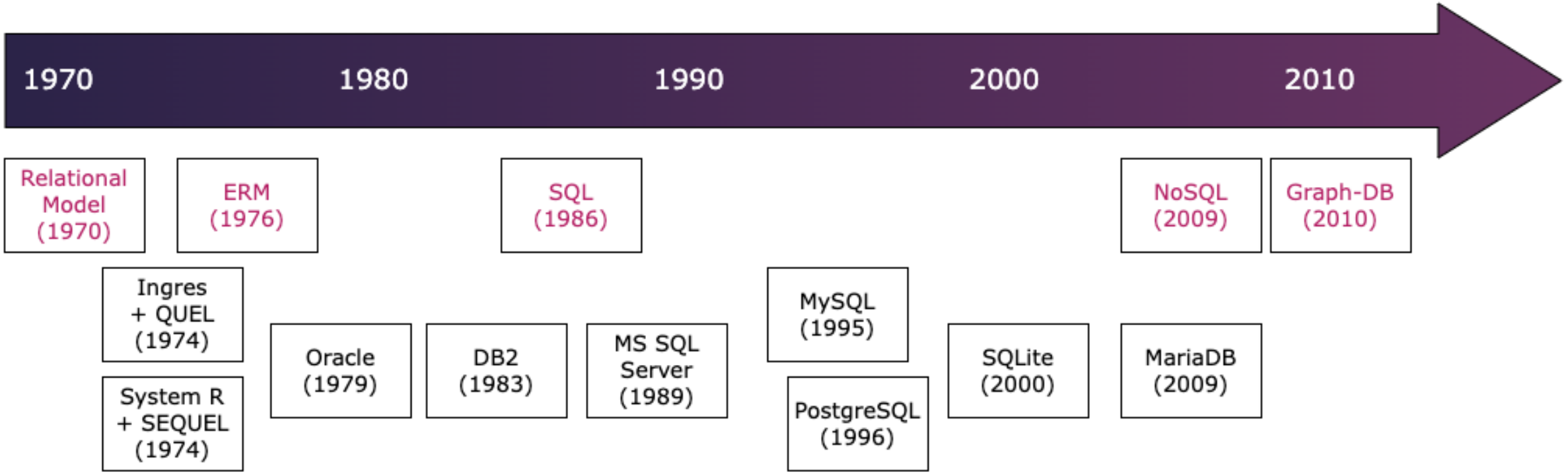
- Eigenschaften:
  - Datenstruktur: Definition der Struktur wie die Daten gespeichert werden (Tabellen, Dokumente, Graphen, …)
  - Operatoren: Festlegung der möglichen Datenbankoperationen, d.h. der möglichen Aktionen auf den Daten (z.B. Abfragen, Einfügen, Löschen, Ändern)
  - Integritätsbedingungen: Regeln zur Einschränkung der zulässigen Inhalte für korrekte und konsistente Daten (z.B. Primärschlüssel, referentielle Integrität)
- Bekanntestes & meistverbreitetes Datenbankmodell: relational
- DBMS legt DB-Modell fest & implementiert dessen Eigenschaften und Funktionen

Beispiele für DB-Modelle

- Relational: Tabellen und Beziehungen
  DBMS: PostgreSQL, MySQL, MariaDB, DB2, …
- Dokumentenorientiert: z.B. JSON-Dokumente
  DBMS: CouchDB, MongoDB, …
- Objektorientiert: Objekte mit Methoden & Attributen
  DBMS: db4o, ObjectDB, Objectivity/DB, …
- Hierarchisch: Baumstruktur
  DBMS: IBM IMS, MS Windows Registry, Adabas, …
- Graphenbasiert: Graph mit Knoten und Kanten
  DBMS: Neo4j, JanusGraph, ArangoDB, OrientDB, …

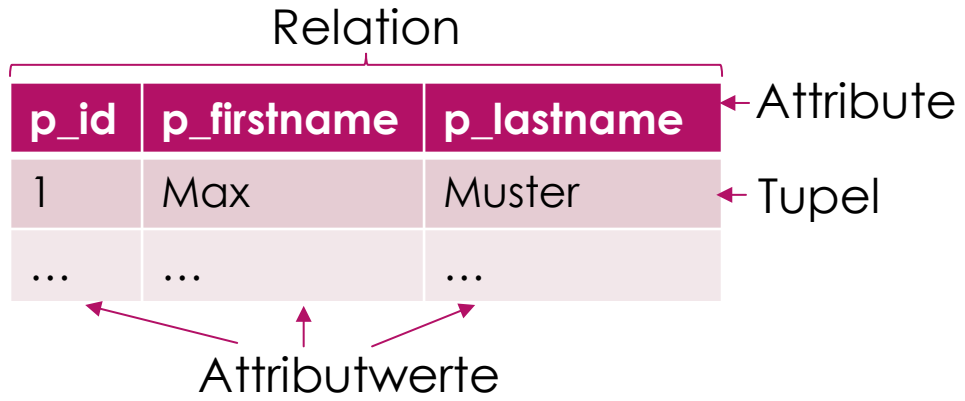


Entwicklung



Relationale DBMS (RDBMS)

- Speicherung von Daten in Tabellen („Relationen“)
  - Spalten: Attribute eines Entities
  - Zeilen („Tupel“): Datensatz enthält Daten zu bestimmter Instanz eines Entities
- Verknüpfung von Tabellen (Beziehungen)
  - Primärschlüssel (Attribut oder Kombination von Attributen): eindeutiger Identifikator für jeden Datensatz in einer Tabelle (z.B. ID)
  - Fremdschlüssel: Attribut, das auf den Primärschlüssel in anderer Tabelle verweist → stellt Beziehung zwischen Tabellen her
- Oft Unterstützung von Transaktionen und ACID-Konformität
- Standard-Abfragesprache: SQL



Transaktionen & ACID-Konformität

- Transaktion
  - Ausführung von mehreren Datenbankoperationen als logische Einheit
  - **Entweder** erfolgreiche Ausführung aller Operationen (COMMIT)
  - **Oder** Ausführung keiner Operation (ROLLBACK)
  - Beispiel: Überweisung von 100 Euro auf anderes Bankkonto
  - → Ziel: Datenbank arbeitet auch in kritischen Szenarien korrekt
- ACID: Eigenschaften garantieren zuverlässige Verarbeitung von Transaktionen
  - Atomicity: Alles oder nichts
  - Consistency: garantiert gültiger Zustand
  - Isolation: Unabhängigkeit
  - Durability: dauerhafte Speicherung

Beispiele für den Einsatz von RDBMS

- E-Commerce
  Verwaltung von Produkten, Bestellungen, Kunden und Lagerbeständen in einer strukturierten Form
- Personalverwaltung
  Speichern und Verwalten von Mitarbeiterdaten, Gehältern, Abteilungen und Arbeitszeiten
- Bankwesen
  Kontoverwaltung, Transaktionen und Kundeninformationen
- Studentenverwaltung
  Verwaltung von Studenten, Kursen, Dozenten und Noten
- Gesundheitswesen
  Patientenakten, Behandlungen und Terminverwaltung

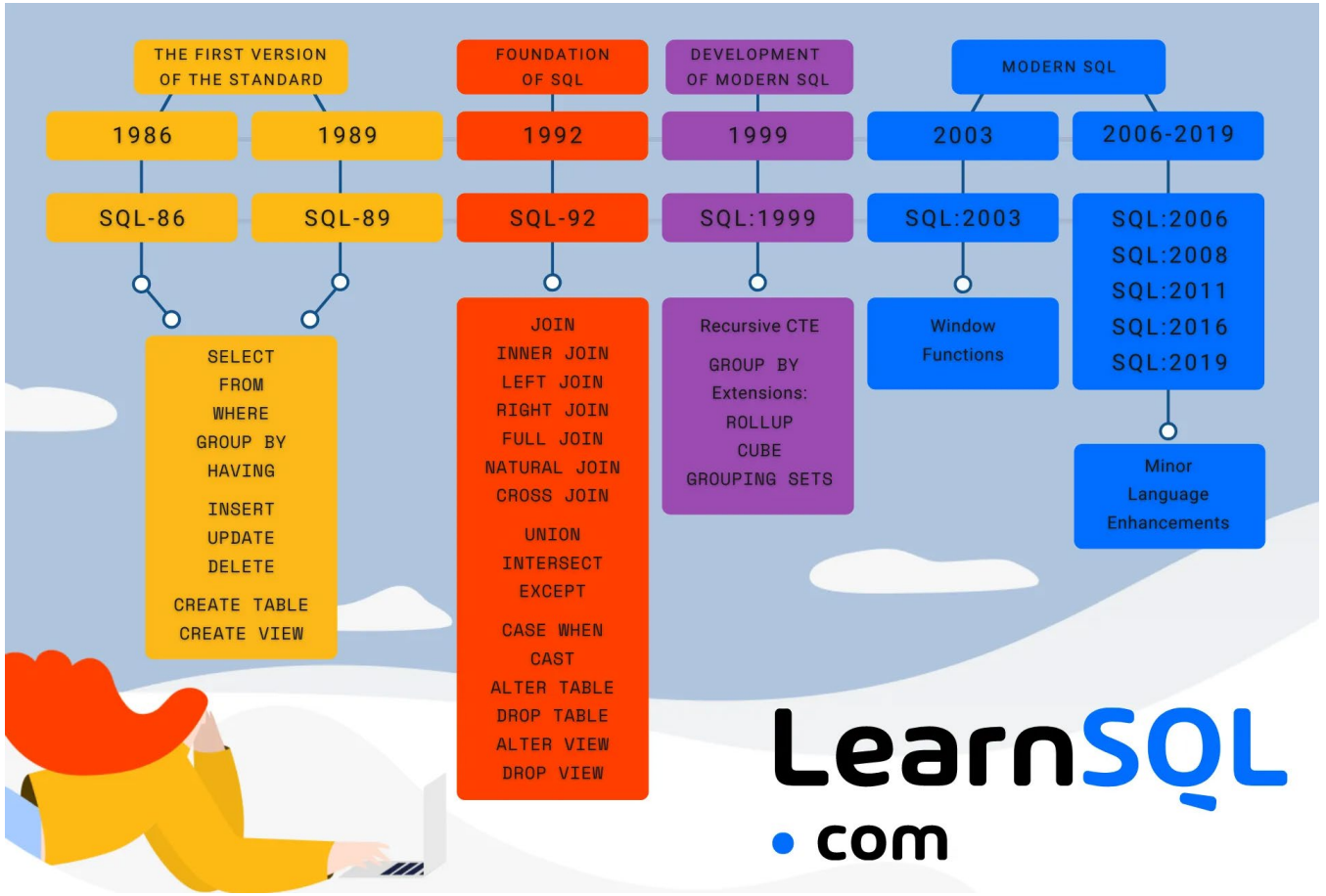
SQL

Structured Query Language

- Deklarative Datenbanksprache für relationale Datenbanken
  - Zur Definition von Datenstrukturen in relationalen Datenbanken
  - Zum Abfragen und Bearbeiten von darauf basierenden Datenbeständen
- Sprache basiert auf relationaler Algebra, sehr mächtig
- Syntax einfach aufgebaut, semantisch ähnlich englische Umgangssprache
  Bsp: SELECT id, name FROM persons WHERE id=1;
- Standardisiert
  - Plattformübergreifende Nutzung mit verschiedenen DBMS möglich
  - Aber dennoch Unterschiede (Syntax, Funktionen, Erweiterungen)

SQL-Standards

- **Wichtigster Standard: SQL-92** von fast allen RDBMS unterstützt
- Problem: Verschiedene DBMS implementieren Standards unterschiedlich



Übersicht: (Open Source) RDBMS

MySQL

- Einsatz: Web-Anwendungen, CMS, …
- Sehr weit verbreitet
- Vorteile: benutzerfreundlich, einfach, große Community
- Nachteile: begrenzte Unterstützung von erweiterten Funktionen, Einschränkungen bzgl. Skalierbarkeit und komplexen Transaktionen, (Performance)

MariaDB

- Einsatz: MySQL-Anwendungen die bessere Performance & mehr Funktionen benötigen
- Wächst stetig
- Vorteile: kompatibel mit MySQL, erweiterte Funktionen, bessere Performance, aktive Entwicklung, schneller Release-Zyklus
- Nachteile: kleinere Community als MySQL

PostgreSQL

- Einsatz: komplexe Anwendungen
- Beliebt in Forschung / Finanzwesen
- Vorteile: hochgradig Standard-konform, zahlreiche erweiterte Funktionen, hohe Zuverlässigkeit & Datenintegrität, hohe Skalierbarkeit
- Nachteile: komplexere Konfiguration & Installation, höherer Ressourcenverbrauch

SQL-Standardkonformität

- MySQL: grundlegende Standardkonformität
  - Unterstützt Teile der Standards
  - Weniger Unterstützung für rekursive Abfragen, CTE, Window Functions
- MariaDB: Starke Standardkonformität
  - Zusätzliche Funktionen und Erweiterungen im Vergleich zu MySQL (z.B. JSON-Support, Storage-Engines, …)
  - Balance aus MySQL-Kompatibilität & breiter Standardkonformität mit erweiterten Funktionen
- PostgreSQL: Umfangreiche Standardkonformität
  - Implementiert breite Palette von SQL-Funktionen gemäß Standard
  - Auch zahlreiche erweiterte Funktionen (CTE, rekursive Abfragen, Window Functions, …)

MariaDB

Allgemeines

- Fork von MySQL mit zusätzlichen Funktionen und Verbesserungen
- MySQL: 2008 Übernahme durch Sun Microsystems (seit 2010 Oracle)
  - Kritik seit Übernahme: große Unterschiede zwischen kommerzieller und freier Version
  - MySQL entwickelt sich zu geschlossenem Projekt
  - → Wird mittlerweile oft von MariaDB abgelöst
- Bevorzugtes Einsatzgebiet: Webservices (z.B. CMS, Flickr, YouTube, Facebook, Twitter)
  - Für kleine und große Anwendungen geeignet (bessere Performance)
  - Schnell, zuverlässig, flexibel, benutzerfreundlich, einfach zu nutzen
- Struktur: Server zur Speicherung der Daten, Standard-Port 3306

Datentypen

- Für Attribute (Spalten einer Tabelle) müssen Datentypen definiert werden
- Beispiele für Standard-Datentypen:
  - Ganze Zahlen (32 Bit): INTEGER
  - Festkommazahlen: DECIMAL(n,m) → n Gesamt-Anzahl, m Nachkommastellen
  - Gleitkommazahlen: FLOAT, DOUBLE
  - Zeichenketten: VARCHAR(n), CHAR(n) → n<255; varchar: n=max, char: n=fix
  - Text: TEXT (max. Länge 65535 Zeichen)
  - Datums- und Zeitangaben: DATE (YYYY-MM-DD), Time (HH:MM:SS), Datetime
- Vgl. SQL Data Types (W3C)

Funktionen für Datentypen

- Zahlen:
  - Normale Rechenoperatoren (+, -, *, /)
  - Betrag (abs), Modulo (mod), Winkelfunktionen (cos, sin, tan, …)
  - Rundung (floor, ceil, round)
  - Exponentialfunktionen (exp, log, pow, …)
- Zeichenketten:
  - Zeichenkettenlänge & Co (concat, trim, length, …)
  - Substrings & Co (locate, position, substring, …)
  - Großschreibung, Änderungen (lower, upper, reverse, …)

Speicher-Engines

- Unterschiedliche Typen für Tabellen → Typ legt Speicher-Engine für Anfragen fest
- Speicher-Engines unterstützen verschiedene Funktionen
  - Je nach Einsatzgebiet unterschiedliche Performance
  - Jede Engine hat Vor- und Nachteile
  - ➔ Zum Anwendungsfall passende Engine wählen
- MariaDB stellt viele verschiedene Storage Engines zur Verfügung

  z.B. MyISAM, InnoDB, Aria, TokuDB, ColumnStore, Memory, Spider, S3, Connect, …
- Standard: InnoDB

InnoDB

- Default Speicher-Engine für MySQL & MariaDB mit vollem Support von Transaktionen
- Gut geeignet für Anwendungen mit hohen Ansprüchen an Transaktionssicherheit, Datenintegrität und Leistung
  - Unterstützt ACID-Transaktionen, Row-Level-Locking und Fremdschlüssel
  - Hohe Performance bei hohen Schreiblasten
  - Multiversion Concurrency Control für gleichzeitige Datenbankzugriffe
- Höherer Speicherbedarf (maximale Tabellengröße 64TB)

MyISAM

- Ältere Speicher-Engine, war bis MySQL 5.5 Default Storage Engine
- Schnell und speichereffizient
  → gut geeignet für Anwendungen überwiegend lesenden Zugriffen, ABER:
  - Kein Row-Level-Locking für parallele Aktionen
  - Keine Transaktionsunterstützung
  - Keine Fremdschlüsselunterstützung
- Optimiert auf Kompression und Geschwindigkeit
  - Schneller Lesezugriff auf Tabellen und Indizes
  - Geringer Speicherbedarf: Unterstützt große Tabellen (256TB) → Kompression möglich

Aria

- Ersatz für MyISAM in MariaDB
- Optimiert für Geschwindigkeit und verbesserte Zuverlässigkeit
  - Schnelle Lesezugriffe → hohe Performance
  - Unterstützt Crash-Recovery (sowohl transaktional als auch nicht-transaktional)
  - Kompatibel mit MyISAM, aber flexibler
- Nachteile
  - Keine vollständige Transaktionssicherheit
  - Keine Fremdschlüsselunterstützung

Weitere Optimierungen in MariaDB

- Performance
  - Verbesserter Query Optimizer gegenüber MySQL (v.a. bei komplexen Abfragen)
  - Thread Pooling zur Verwaltung der aktiven Threads → Optimierung bei großen Workloads (gleichzeitigen Verbindungen) und hoher Parallelität
- Vereinfachte Rechteverwaltung: Rollen mit spezifischen Berechtigungen für Benutzer
- JSON-Unterstützung: native Unterstützung für JSON-Datentypen und Abfragen

Installation im Container

- Installation von Docker Desktop
  - Docker Desktop installieren: https://docs.docker.com/desktop/
  - Ggf. WSL-2 installieren: https://learn.microsoft.com/en-us/windows/wsl/install
- Umgebung für Übungen: phpMyAdmin + MariaDB
  - docker-compose.yml aus Moodle herunterladen
  - Terminal im Ordner der Datei öffnen und `docker-compose up -d` ausführen
- Funktionstest:
  - PhpMyAdmin im Browser öffnen + einloggen
  - Version und Datum im Terminal des DB-Containers ausgeben lassen:
    - Im Container-Terminal zu SQL wechseln: `mariadb -u benutzer -p`
    - SELECT VERSION(), CURRENT_DATE();

Alternativen zur Docker

- Lokale Installation:
  - MySQL Installer: https://dev.mysql.com/downloads/installer/
  - Wichtig: Root-Passwort setzen + merken
  - Ggf. Standard-Port ändern und Samples mit installieren
  - Funktionstest: Version & Datum ausgeben lassen
    - Shell öffnen und zu SQL wechseln: \sql
    - Mit Server verbinden: \c benutzer@localhost:port (z.B. root@localhost:3306) → PW
    - SELECT VERSION(), CURRENT_DATE();
- Notlösung:
  Online-Playgrounds wie https://sqlfiddle.com oder https://www.db-fiddle.com/ (MySQL)

Konventionen / Best Practices

- Allgemein
  - Englische Begriffe, Kleinschreibung, keine Leer- / Sonderzeichen, Abkürzungen vermeiden
  - Sprechende, beschreibende Namen
  - Konsistente Regeln (z.B. snake_case oder CamelCase)
- Tabellen
  - Namen meist im Plural (users, orders, …), keine unnötigen Präfixe / reservierte Wörter
  - Constraints nutzen (z.B. Keys, NOT_NULL, …)
- Spaltennamen
  - Eindeutigkeit im Kontext (z.B. first_name, last_name, product_name statt jeweils name)
  - Primärschlüssel meist id ⇔ Fremdschlüssel Tabellenname + _id (z.B. user_id, order_id)

SQL-Teilsprachen

SQL-Teilsprachen

- Data Definition Language (DDL):
  Definition des DB-Schemas (z.B. CREATE, ALTER, DROP)
- Data Manipulation Language (DML)
  Bearbeiten / Einfügen / Löschen von Daten (z.B. UPDATE, INSERT, DELETE)
- Data Query Language (DQL):
  Abrufen von Daten (SELECT mit vielen Funktionen)
- Data Control Language (DCL):
  Verwaltung von Rechten (z.B. GRANT, REVOKE)
- Transaction Control Language (TCL):
  Kontrolle von Transaktionen (z.B. COMMIT, BEGIN, ROLLBACK)

Wichtigste DDL-Befehle

- Anlegen einer Datenbank: CREATE DATABASE <name>;
- Anlegen einer Tabelle: CREATE TABLE <name>
  (<column> <datatype> <constraint>);
- Ändern einer Tabelle:
  ALTER TABLE <name> + ADD <column> <datatype> <constraints>;
  DROP <column>;
  MODIFY <column> <datatype> <constraints>;
- Löschen einer Datenbank / Tabelle: DROP DATABASE / TABLE <name>;
- Constraints für Tabellen / Spalten: z.B. NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, DEFAULT '<value>', CREATE INDEX

Erste Datenbank aufsetzen [1]

- Eigene Datenbank anlegen und nutzen
  - Neue DB anlegen: CREATE DATABASE mydb;
  - DB auswählen: USE mydb;
  - DB-Tabellen anzeigen: SHOW TABLES; (empty set: keine Tabellen bisher erstellt)
- Tabelle mit Constraints anlegen

  CREATE TABLE students
  (id INTEGER AUTO_INCREMENT, name VARCHAR(100) NOT NULL,
  first_name VARCHAR(100) NOT NULL, birth DATE, PRIMARY KEY (id));
- Tabellenstruktur anzeigen

  DESCRIBE students;

Erste Datenbank aufsetzen [2]

- Tabelle ändern
  - Geburtsdatum löschen:
    ALTER TABLE students DROP birth;
  - Spalte Nachnamen umbenennen:
    ALTER TABLE students CHANGE COLUMN name last_name VARCHAR(100) NOT NULL;
  - Spalte Vorname ändern:
    ALTER TABLE students MODIFY first_name VARCHAR(100);
  - Spalte Alumni hinzufügen:
    ALTER TABLE students ADD is_alumni BOOLEAN;
- Tabelle nochmal überprüfen: DESCRIBE students;

Wichtigste DML-Befehle

- Einfügen von Daten:
  - INSERT INTO <table> (column1, column2) VALUES (value1, value2);
  - Mehrere VALUES auf einmal: Mehrere Datenpaare in Klammern dahinter
- Löschen von Daten:

  DELETE FROM <table> WHERE <condition> ;
- Bearbeiten von Daten:

  UPDATE <table> SET <column1>=<value1>, <column2>=<value2>,… WHERE <condition>;

Datensätze einfügen und ausgeben

- Daten in Tabelle einfügen:
  - INSERT INTO students (last_name, first_name, is_alumni) VALUES ('Maier', 'Hugo', false);
  - INSERT INTO students (last_name, is_alumni) VALUES ('Schmidt', 1);
  - INSERT INTO students (last_name, first_name) VALUES ('Maja', 'Huber');
  - INSERT INTO students (last_name) VALUES ('Eder');
  - INSERT INTO students (last_name, is_alumni) VALUES ('Liebel', true);
  - INSERT INTO students (first_name) VALUES ('Bine'); → nicht möglich
- Informationen ausgeben:
  - Alle Daten ausgeben: SELECT * FROM students;
  - Bestimmte Daten (Spalte) der Tabelle ausgeben: SELECT last_name FROM students;

Datensätze verändern

- Datensatz löschen:
  - DELETE FROM students WHERE id = 5;
  - Liebel fehlt
- Datensatz ändern:
  - UPDATE students SET is_alumni=true WHERE last_name='Eder';
  - UPDATE students SET first_name='Martin' WHERE id = 2;
- Daten erneut ausgeben und prüfen:

  SELECT * FROM students;

Exkurs:

Normalisierung

Normalisierung

- Prozess zur Organisation von Daten in einer Datenbank, um Redundanzen zu minimieren und die Datenintegrität zu maximieren
- Relevanz:
  - Vermeidung von Dateninkonsistenzen (z.B. widersprüchliche Einträge)
  - Verbesserung der Effizienz bei Updates und Abfragen
  - Klare Strukturierung der Daten

Überblick über die Normalformen

- 1. Normalform:
  - Alle Attributwerte sind atomar (nicht weiter aufteilbar)
  - D.h.: Jede Spalte enthält nur einen Wert pro Zelle
- 2. Normalform:
  - Erfüllt die erste NF UND
  - Alle Nicht-Schlüssel-Attribute müssen vollständig vom Primärschlüssel abhängig sein
- 3. Normalform:
  - Erfüllt die erste und zweite NF UND
  - Es gibt keine transitiven Abhängigkeiten, d.h. Nicht-Schlüsselattribute dürfen nicht von anderen Nicht-Schlüsselattributen abhängen

Wie weit normalisieren?

- Hängt vom Anwendungsfall ab, häufig bis zur dritten Normalform
- Balance zwischen Normalisierung und Performance:
  - Normalisierte Daten oft effizienter bei Updates
  - Aber ggf. bei Abfragen wegen vieler Joins langsamer
- ➔ Kein blindes Anwenden jeder Normalform, sondern abwägen

Zweite Tabelle anlegen

- Studenten sind in Kursen organisiert → Tabelle classes anlegen
  - id ist PRIMARY KEY und AUTO_INCREMENT
  - name ist VARCHAR(50) mit NOT NULL
- Einfügen von zwei Kursen

  INSERT INTO classes (name) VALUES ('AI2022'), ('AI2023');
- Studenten zu Kursen zuweisen
  - Tabelle students ändern, sodass id der Class als Fremdschlüssel hinzugefügt wird
  - ALTER TABLE students ADD class_id INT;
  - ALTER TABLE students ADD CONSTRAINT fk_students_classes FOREIGN KEY (class_id) REFERENCES classes(id);

Datensätze updaten

- UPDATE students SET class_id=3 WHERE id=1;
  - → Fehler, da Kurs mit ID=3 noch nicht existiert
- UPDATE students SET class_id=1 WHERE id=1;
  - → Funktioniert
- Problem:

  Wie verhalten sich die Einträge in Students bei Löschen des Kurses mit der ID=1?
  - → Hängt von der Definition der Constraints ab

Referentielle Integrität

- Sicherstellen der Konsistenz von Beziehungen zwischen Tabellen
  - Jeder Fremdschlüsselwert einer Tabelle benötigt Primärschlüsselwert in anderer Tabelle
  - Prinzip verhindert Einfügen von ungültigen Daten → dient der Konsistenz
- Komponenten:
  - Primary Key:
    Attribut / Kombination von Attributen zur eindeutigen Identifikation eines Datensatzes
  - Foreign Key:
    Attribut / Kombination von Attributen, das auf Primärschlüssel in anderer Tabelle verweist

Regeln der referentiellen Integrität

- Insert Rule:

  FK-Wert kann nur eingefügt werden, wenn er einem vorhandenen PK-Wert in der referenzierten Tabelle entspricht (oder null ist, sofern erlaubt)

- Delete Rule:

  PK-Wert kann nicht gelöscht werden, wenn es noch referenzierende FK-Werte in anderen Tabellen gibt – außer: geeignete Aktion ist definiert (ON DELETE)

- Update Rule:

  PK-Wert kann nicht aktualisiert werden, wenn es noch referenzierende FK-Werte in anderen Tabellen gibt – außer: geeignete Aktion ist definiert (ON UPDATE)

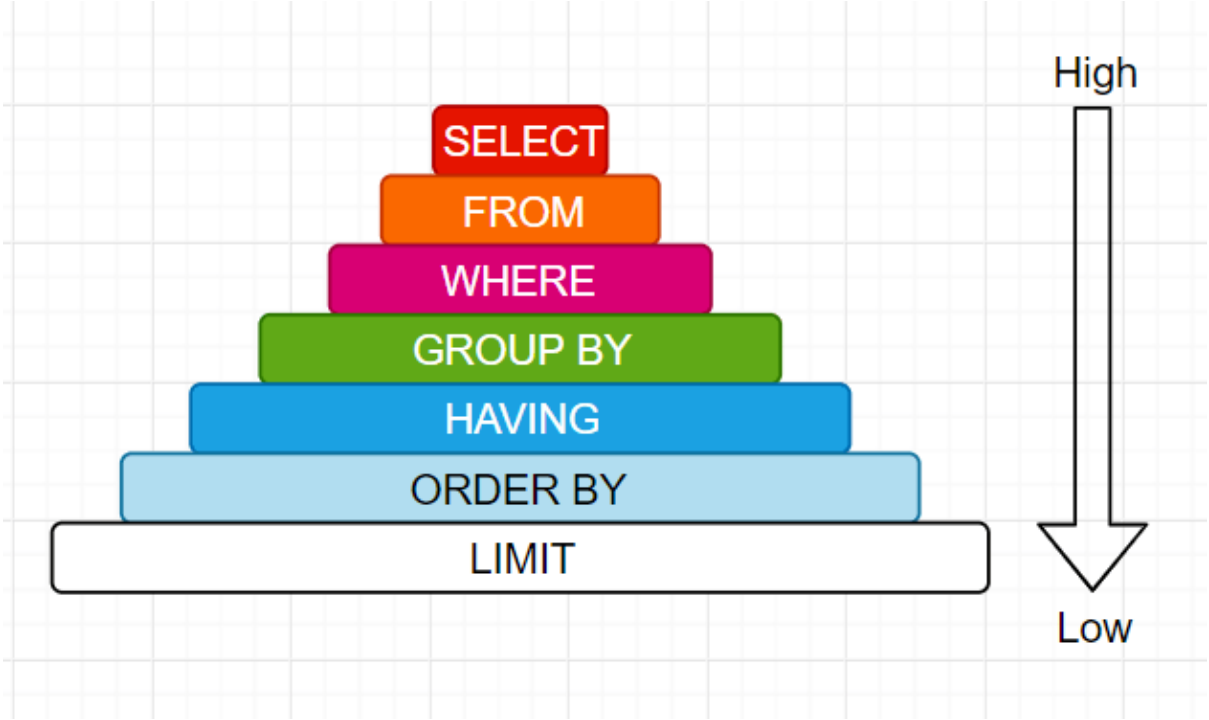
Aktionen der referentiellen Integrität

Aktion	'ON DELETE'-Bedeutung	'ON UPDATE'-Bedeutung
CASCADE	Löscht referenzierende Zeilen	Aktualisiert referenzierende Zeilen
SET NULL	Setzt FK-Wert auf NULL	Setzt FK-Wert auf NULL
SET DEFAULT	Setzt FK-Wert auf Standard-Wert	Setzt FK-Wert auf Standard-Wert
RESTRICT	Verhindert Löschung	Verhindert Aktualisierung
NO ACTION	Verhindert Löschung (nach Transaktion)	Verhindert Aktualisierung (nach Transaktion)

Weitere Tabellen anlegen

- Annahme:
  - Es gibt Lehrveranstaltungen, die von jedem Studenten (unabhängig von seinem Kurs) belegt werden können.
  - CREATE TABLE courses (id INT, name VARCHAR(100) NOT NULL, PRIMARY KEY id);
- Mapping von Studenten zu Lehrveranstaltungen
  - Ein Student kann mehrere LV haben
  - Eine LV kann von mehreren Studenten besucht werden
  - Mapping-Tabelle Student in Lehrveranstaltung notwendig (n:m-Beziehung)
  - CREATE TABLE student_in_course (course_id INT, student_id INT,
    FOREIGN KEY (course_id) REFERENCES courses(id),
    FOREIGN KEY (student_id) REFERENCES student(id));

Zentrale DQL-Elemente



SELECT-Befehl

- Startet Anfrage zum Abrufen von Datensätzen
- SFW-Block: SELECT *Spaltenname* FROM *Tabellenname* WHERE *Bedingung*
  - Projektion (Auswahl spezieller Spalten): SELECT id, last_name FROM students;
  - Selektion (Auswahl von Zeilen mit Bedingung): SELECT * FROM students WHERE id>3;
- Umbenennung von Spalten durch Alias: SELECT name AS Nachname FROM person;
- Redundanzen eliminieren: SELECT DISTINCT Spaltenname
- Ergebnisse filtern: WHERE mit Vergleichsoperatoren
  - Eine Abfrage kann nur eine Bedingung haben → WHERE-Klausel
  - Mehrere Bedingungen durch logische Verknüpfungen möglich

Conditions via WHERE-Klausel

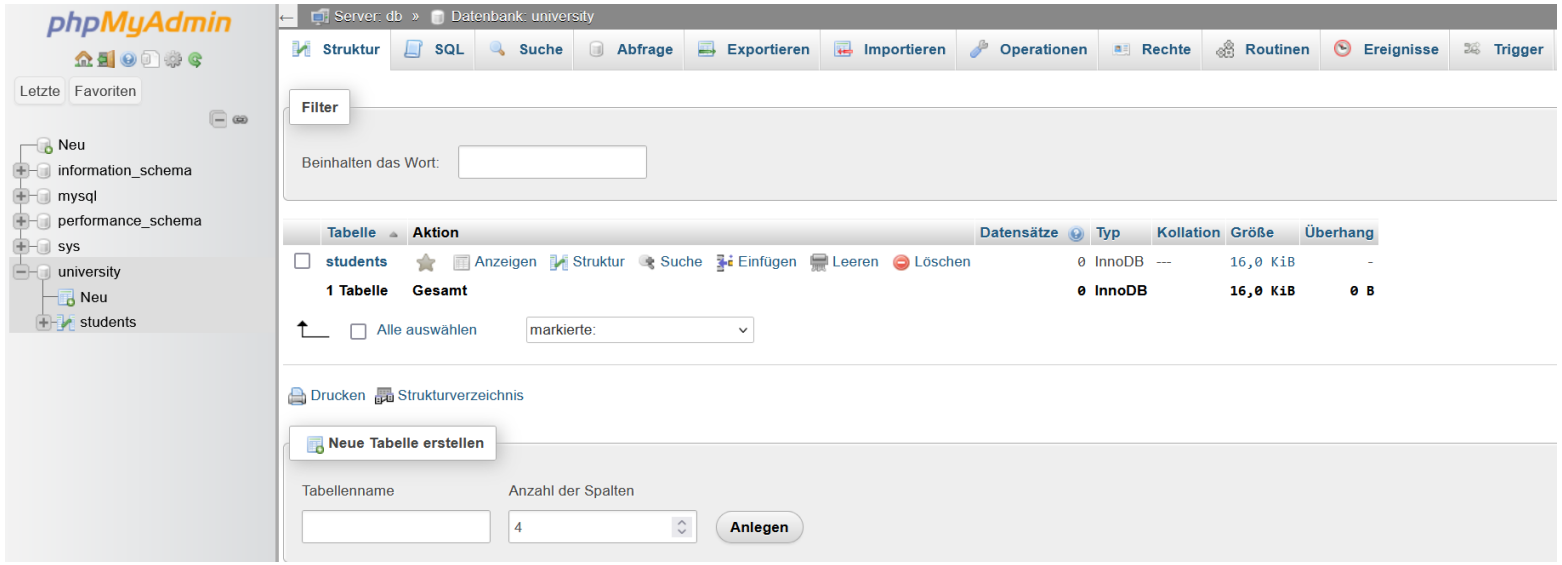
Operatoren:

- Verknüpfung mehrerer Bedingungen durch AND, OR, NOT
- gleich =, verschieden <>, kleiner <, größer >, kleiner-gleich <=, größer-gleich >=
- Leere Zelle IS NULL, Zelle mit Wert IS NOT NULL
- Zwischen den Werten: (NOT) BETWEEN → Bsp: WHERE age BETWEEN 0 AND 18
- Ergebnis kommt in Ausdruck vor: (NOT) IN → Bsp: WHERE name IN (Hugo, Maja)
- Textmuster vergleichen: (NOT) LIKE → Bsp: WHERE name LIKE 'M_ _er' oder '%z%'
- Spezielle Zeichen: _ bel. einzelner Buchstabe, % mehrere bel. Buchstaben

Aufgabe

Legen Sie die besprochenen Tabellen an und fügen Sie einige Datensätze (VVZ von diesem Semester) ein.

→ Tipp: Nutzen Sie der Einfachheit halber phpmyadmin



Weiterführende Infos und Hilfe

- SQL Tutorial von w3schools
- SQL Tutorial und SQL Exercices von w3resource
- SQL-Nachschlagewerk: Umfassende Informationen inkl. Beispielen