



Foto: Thomas Josek

# Datenmodellierung

## Woche 5: Relationale Datenbanken

# Logisches Datenbankschema

Nachdem man die Anforderungen für die Datenbank geklärt und ein konzeptuelles Modell der Datenbank erstellt hat (wie in der Übung in der letzten Sitzung), kann die Datenbank jetzt logisch modelliert werden:

- Für jeden Entitätstyp wird eine Tabelle angelegt
  - Name der Tabelle: Entitätstyp
  - Spalten der Tabelle: Attribute
  - Evtl. Aggregation/Generalisierung (z.B. 'Gemälde' und 'Skulpturen' zusammenfassen zu Entitätstyp 'Werke')
- **Schlüssel** zur eindeutigen Identifizierung jedes Datensatzes
- **Normalisierung** zur Vermeidung von Redundanz
  - Normalformen
- Integritätsbedingungen
- Datentypen

# Normalisierung

Normalisierung: Überführung komplexer Tabellen in einfache Tabellen (durch Aufteilung der Attribute einer Tabelle auf mehrere Tabellen) zur Vermeidung von Anomalien, Redundanz und Inkonsistenz

Normalformen:

- 1NF = Alle Attribute der Tabelle enthalten atomare Werte und die Tabelle ist frei von Wiederholungsgruppen
  - Nicht-atomare in Spalten aufteilen und gleichartige Daten in Zeilen aufteilen
- 2NF = 1NF + Jedes Nicht-Schlüsselattribut ist von jedem Schlüsselkandidaten vollständig funktional abhängig
  - Zu jedem zusammengesetzten Primärschlüsselattribut eine eigene Tabelle erstellen mit den Attributen, die von dieser Kombination abhängig sind
- 3NF = 2NF + Jedes Nicht-Schlüsselattribut ist von keinem Schlüsselkandidaten transitiv abhängig
  - Attribute bestimmen, die von Nicht-Schlüsselattributen abhängig sind; für diese Nicht-Schlüsselattribute eigene Tabellen mit den abhängigen Attributen erstellen

# Implementierung einer relationalen Datenbank

- Anforderungserhebung
  - Welche Daten sollen später wie verarbeitet werden?
- Konzeptuelles Modell
  - Semantisches Datenmodell, z.B. E-R-Diagramm
- Logischer Entwurf
  - Datenbankentwurf, Normalisierung
- Implementierung
  - Umsetzung unter Verwendung eines DBMS
  - Datenbankschema

# SQL

= Structured Query Language; Datenbanksprache

- Internationaler Standard, sorgt für Unabhängigkeit von DBMS  
Aber: SQL-Dialekte, d.h. Abweichungen der DBMS vom SQL-Standard
  - Je nach DBMS im Detail unterschiedliche Umsetzung des Standards, kein DBMS unterstützt den Standard vollständig; bei Portierung zu beachten
- SQL arbeitet mengenorientiert, auf Basis der relationalen Algebra

# SQL

SQL-Befehle können in verschiedene Kategorien unterteilt werden:

- Data Definition Language (DDL) → Definieren und Ändern des Datenbankschemas
- Data Manipulation Language (DML) → Einfügen, Ändern, Löschen von Daten
- Data Query Language (DQL) (oder auch als Teil von DML) → Abfragen von Daten
- Data Control Language (DCL) → Rechteverwaltung
- Transaction Control Language (TCL) → Transaktionskontrolle

# SQL: Integritätsbedingungen, Datentypen

## Integritätsbedingungen (Constraints):

- PRIMARY KEY
- NOT NULL
- CHECK (Bedingung)
- UNIQUE
- REFERENCES table\_name  
(column\_name) → Fremdschlüssel

```
CREATE TABLE eissorten (  
    id INTEGER PRIMARY KEY,  
    name TEXT UNIQUE  
);
```

## Datentypen:

Je nach DBMS werden unterschiedliche Datentypen unterstützt, u. a.:

- INTEGER
- TEXT

# SQL: Syntax

Bestandteile eines SQL-Befehls:

- Schlüsselwörter (z.B. CREATE oder SELECT), anhand derer das DBMS die Informationen innerhalb eines Befehls erkennt
- Name des Objekts (Datenbank, Tabelle, Spalte etc.)
- Evtl. Angabe weiterer Einzelheiten durch weitere Schlüsselwörter und Namen
- Semikolon als Abschluss
- *Häufig werden Schlüsselwörter zur besseren Lesbarkeit in Großbuchstaben geschrieben und Zeilenumbrüche verwendet, aber beides wird vom DBMS nicht unterschieden*
  - syntactic sugar
- Namen von Tabellen, Spalten etc.: mit Buchstaben oder Unterstrich beginnen, keine Schlüsselwörter verwenden

# SQL: Data Definition Language – Beispiele

```
CREATE DATABASE database_name;
```

```
CREATE TABLE table_name (column1,  
column2, column3, ...);
```

```
DROP TABLE table_name;
```

```
ALTER TABLE table_name ...;
```

```
CREATE TABLE eissorten (  
    id INTEGER PRIMARY KEY,  
    name TEXT UNIQUE  
);
```

```
DROP TABLE eissorten;
```

```
ALTER TABLE eissorten  
RENAME TO flavors;
```

# SQL: Data Manipulation Language – Beispiele

**INSERT INTO** *table\_name* (*column1*,  
*column2*, *column3*, ...)  
VALUES (*value1*, *value2*, *value3*, ...);

```
INSERT INTO eissorten (id, sorte)
VALUES (12, 'Stracciatella');
```

**UPDATE** *table\_name*  
SET *column1* = *value1*, *column2* = *value2*, ...  
WHERE *condition*;

```
UPDATE preise
SET preis = '1,40'
WHERE sortenId = 2;
```

**DELETE FROM** *table\_name*  
WHERE *condition*;

```
DELETE FROM eissorten
WHERE id > 10;
```

# SQL: Data Query Language – Beispiele

**SELECT** *column1, column2, ...*  
**FROM** *table\_name*  
**WHERE** *condition;*

- Vorsicht: SELECT \* in Scripts

```
SELECT *  
FROM eissorten;
```

```
SELECT *  
FROM eissorten  
WHERE id <= 10;
```

```
SELECT name  
FROM eissorten  
WHERE name LIKE 'S%';
```

# SELECT ... WHERE ... – Wildcards, Bedingungen

*	alle Spalten
<b>DISTINCT</b>	identische Zeilen in der Ergebnismenge nur einmal ausgeben
=	gleich
<>	ungleich
<	kleiner als
<=	kleiner als oder gleich
>	größer als
>=	größer als oder gleich
<b>LIKE</b>	
%	0, 1 oder mehrere Zeichen
_	1 Zeichen

# SQL: Joins

- Bei Abfragen über mehrere Tabellen hinweg müssen diese mittels Joins verbunden werden
- Join: Verbinden der Spalten zweier Tabellen zu einer Tabelle
- Verwendet in SELECT, INSERT, UPDATE, DELETE usw.
- Verschiedene Arten von Joins: Inner Join, Outer Join, Cross Join, Self Join

eissorten

sortenld	name
1	Schokolade
2	Vanille
3	Erdbeer

preise

sortenld	preis
1	1,50
2	1,20

# Cross Join

- Kartesisches Produkt (auch Kreuzprodukt)
- Jede Zeile von Tabelle 1 wird mit jeder Zeile von Tabelle 2 verbunden (d.h., jede mögliche Kombination)

```
SELECT *  
FROM tabelle1  
CROSS JOIN tabelle2;
```

```
SELECT *  
FROM eissorten  
CROSS JOIN preise;
```

sortenld	name	sortenld	preis
1	Schokolade	1	1,50
1	Schokolade	2	1,20
2	Vanille	1	1,50
2	Vanille	2	1,20
3	Erdbeer	1	1,50
3	Erdbeer	2	1,20

# Inner Join

- INNER JOIN
- NATURAL JOIN → keine Dopplung von Spalten
- JOIN ... ON ...
- Verbindet Datensätze aus zwei Tabellen, wenn sie eine gemeinsame Spalte dieselben Werte enthält

```
SELECT *  
FROM tabelle1  
INNER JOIN tabelle2;
```

```
SELECT *  
FROM tabelle1  
JOIN tabelle2  
ON tabelle1.x = tabelle2.y;
```

```
SELECT *  
FROM eissorten  
INNER JOIN preise;
```

```
SELECT *  
FROM eissorten  
JOIN preise  
ON eissorten.sortenId =  
preise.sortenId;
```

sortenId	preis	sortenId	name
1	1,50	1	Schokolade
2	1,20	2	Vanille

# Outer Join

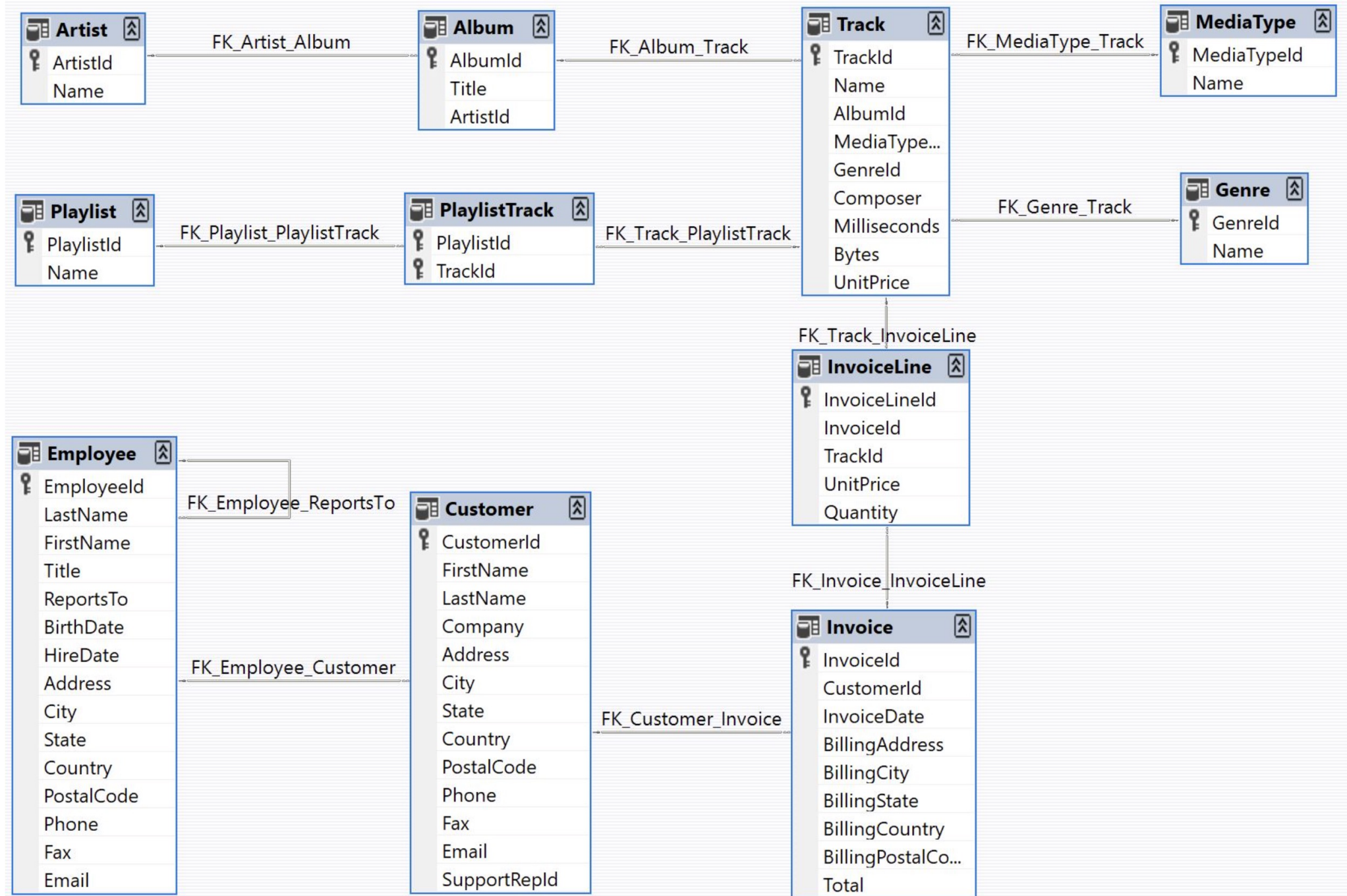
- Es werden auch Datensätze geliefert, für die die Vergleichsbedingung nicht erfüllt ist
- **LEFT OUTER JOIN**
  - Linke Inklusionsverknüpfung: Alle Datensätze aus der ersten (linken) Tabelle werden übernommen, auch wenn keine Werte in der zweiten Tabelle
- **RIGHT OUTER JOIN**
  - Rechte Inklusionsverknüpfung: Alle Datensätze aus der zweiten (rechten) Tabelle werden übernommen, auch wenn keine Werte in der ersten Tabelle

```
SELECT *  
FROM tabelle1  
LEFT OUTER JOIN tabelle2;
```

```
SELECT *  
FROM eissorten  
LEFT OUTER JOIN preise  
ON eissorten.sortenId = preise.sortenId;
```

sortenId	name	sortenId	preis
1	Schokolade	1	1,50
2	Vanille	2	1,20
3	Erdbeer	NULL	NULL

# Übung: Abfragen mit SELECT und JOINS

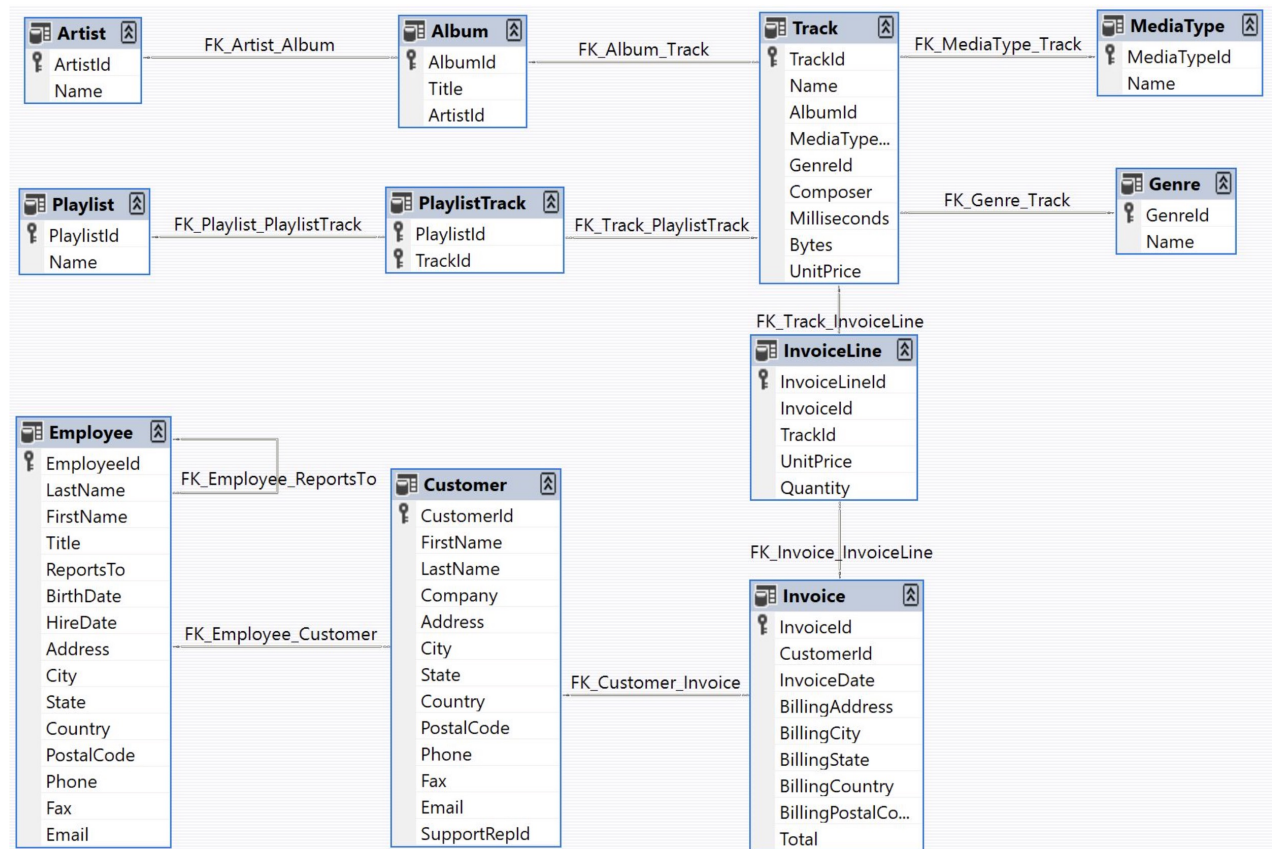


# Übung: Abfragen mit SELECT und JOINS

Beispieldatenbank: <https://github.com/lerochoa/chinook-database>

<https://github.com/lerochoa/chinook-database/releases>

- Download von Chinook\_Sqlite.sqlite
- Öffnen mit DB Browser for SQLite



# Tabellen, Indizes, Trigger, Ansichten

- Tabellen
  - Spalten
  - Primärschlüssel
  - Fremdschlüssel
  - Einschränkungen (Constraints)
- Indizes (Indexes)
  - Von der Datenstruktur getrennte Indexstruktur zur Beschleunigung von Suchvorgängen/Abfragen
- Trigger (Triggers)
  - CREATE TRIGGER ...
  - Werden ausgeführt, wenn eine bestimmte Aktion auf eine bestimmte Tabelle ausgeführt wird, z.B. um bei Änderungen den alten und neuen Wert zu loggen
- Ansichten (Views)
  - CREATE VIEW ...
  - So können SELECT-Abfragen als logische Tabelle gespeichert werden und diese Ansichten später wieder aufgerufen werden, statt erneut die Abfrage zu schreiben

# Relationale Datenbankmanagementsysteme

Erinnerung:

- Datenbanksystem (DBS) =  
Datenbank (DB) + Datenbankmanagementsystem (DBMS)

Unter anderem:

- Oracle Database
- MySQL
- Microsoft SQL Server
- MariaDB (Fork aus MySQL)
- PostgreSQL (objektrelationales DBMS)
- Microsoft Access
- SQLite (eingebettetes Datenbanksystem)



Small. Fast. Reliable.  
Choose any three.

Home About Documentation Download License Support Purchase

Search

SQLite is a C-language library that implements a [small](#), [fast](#), [self-contained](#), [high-reliability](#), [full-featured](#), SQL database engine. SQLite is the [most used](#) database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day. [More Information...](#)

SQLite [source code](#) is in the [public-domain](#) and is free to everyone to use for any purpose.

## Latest Release

[Version 3.53.1](#) (2026-05-05). [Download](#) [Prior Releases](#)

## Sponsors

SQLite is made possible in part by sponsors and [SQLite Consortium](#) members, including:



[Free web GUI for SQLite](#)

## Common Links

- [Features](#)
- [When to use SQLite](#)
- [Getting Started](#)
- [Try it live!](#)
- [SQL Syntax](#)
  - [Pragmas](#)
  - [SQL functions](#)
  - [Date & time functions](#)
  - [Aggregate functions](#)
  - [Window functions](#)
  - [Math functions](#)
  - [JSON functions](#)
- [C/C++ Interface Spec](#)
  - [Introduction](#)
  - [List of C-language APIs](#)
- [The TCL Interface Spec](#)
- [Quirks and Gotchas](#)
- [Frequently Asked Questions](#)
- [Commit History](#)
- [Prior Releases](#)
- [Bugs](#)
- [News](#)

<https://www.sqlite.org/>



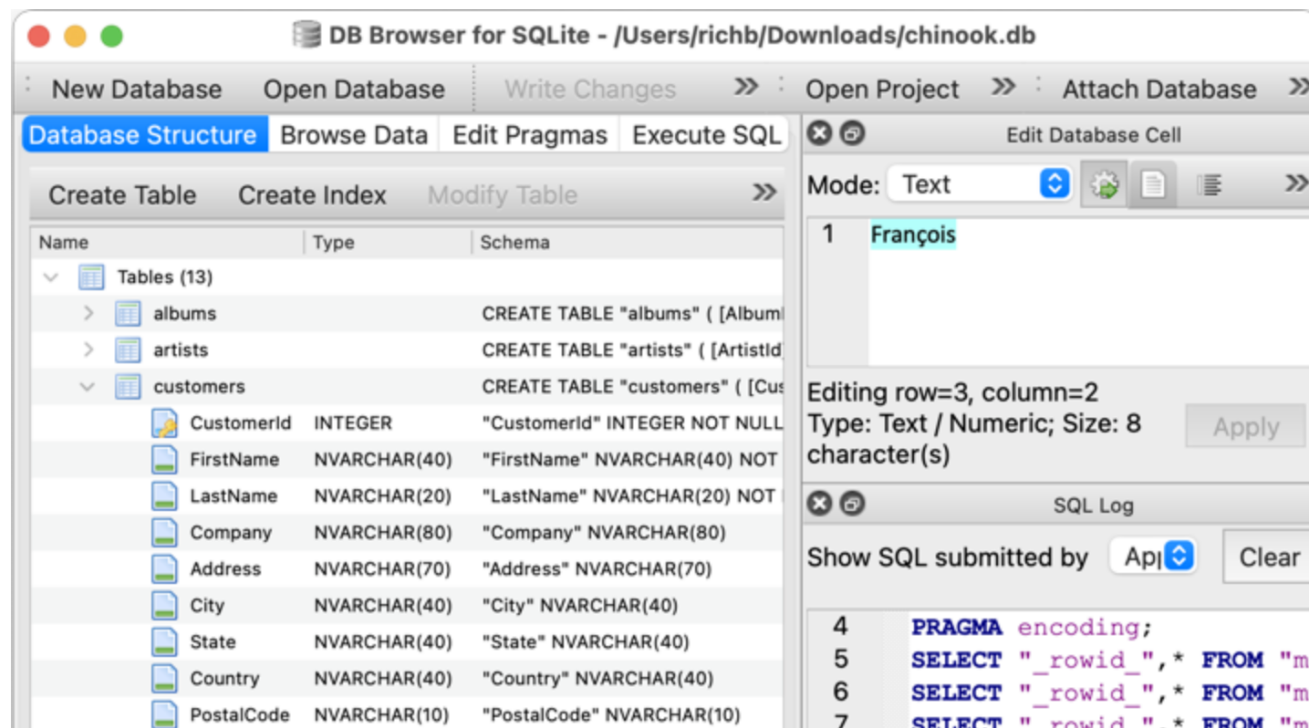
# SQLite

- Verbreitetstes, meist verwendetes Datenbanksystem der Welt
- Ist in Smartphones, Browsern und vielen Anwendungen eingebunden
- Eine SQLite-Datenbank besteht aus einer einzigen Datei
  - Enthält alle Tabellen, Views usw.
  - Lässt sich direkt in Anwendungen einbinden
- Unterschied zu Client/Server-DBMS
  - Geeignet: z.B. in Smartphones und IoT-Geräten; für Webseiten mit geringem/mittlerem Traffic; als Format zum Datenaustausch
  - Ungeeignet: wenn Daten und Anwendung durch ein Netzwerk getrennt sind; bei vielen gleichzeitigen Schreibvorgänge; bei sehr großen Datenmengen
- vgl. <https://www.sqlite.org/whentouse.html>



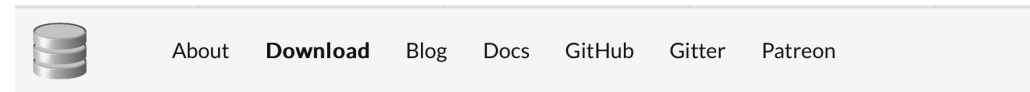
## DB Browser for SQLite

*DB Browser for SQLite* (DB4S) is a high quality, visual, [open source](#) tool designed for people who want to create, search, and edit [SQLite](#) or [SQLCipher](#) database files. DB4S gives a familiar spreadsheet-like interface on the database in addition to providing a full SQL query facility. It works with [Windows](#), [macOS](#), and most versions of [Linux](#) and [Unix](#). Documentation for the program is on the [wiki](#).



# DB Browser for SQLite

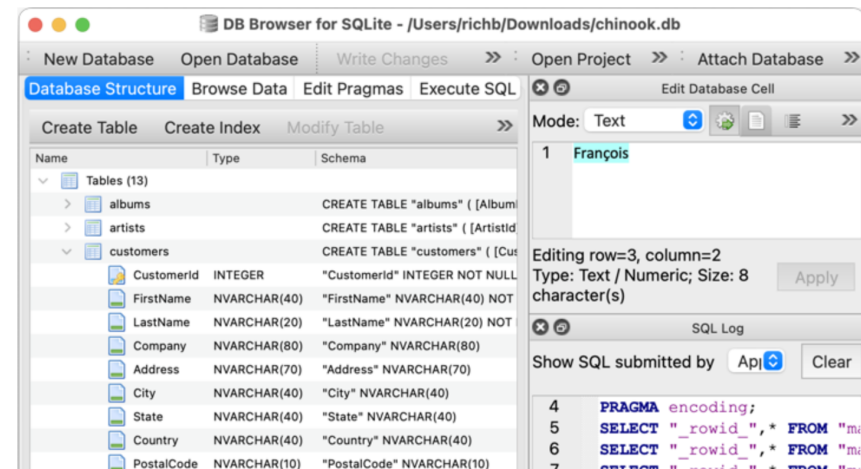
- Download:  
<https://sqlitebrowser.org/dl/>
- Eine grafische Oberfläche für die Erstellung und Betrachtung von SQLite-Datenbankdateien
- Alternativen:
  - Weitere Programme mit grafischer Oberfläche, z.B. [SQLiteStudio](#)
  - Oder SQLite [per Kommandozeile](#)



docs [Wiki](#) chat [on gitter](#) downloads [10M](#) Qt [qmake](#) [coverity](#) [passing](#) donate [Patreon](#)

## DB Browser for SQLite

*DB Browser for SQLite* (DB4S) is a high quality, visual, [open source](#) tool designed for people who want to create, search, and edit [SQLite](#) or [SQLCipher](#) database files. DB4S gives a familiar spreadsheet-like interface on the database in addition to providing a full SQL query facility. It works with [Windows](#), [macOS](#), and most versions of [Linux](#) and [Unix](#). Documentation for the program is on the [wiki](#).



# Studienleistung 1

- Implementieren Sie mithilfe von „DB Browser for SQLite“ eine SQLite-Datenbank.
- Die genaue Aufgabenstellung finden Sie in ILIAS:  
<https://www.ilias.uni-koeln.de/ilias/goto.php/exc/6887393>
- Abgabefrist: Montag, 18.05.2026, 12:00
- Einreichung über ILIAS:  
<https://www.ilias.uni-koeln.de/ilias/goto.php/exc/6887393/147392>
- Bitte reichen Sie eine individuelle Lösung ein, nicht als Gruppenarbeit.
- Bei Fragen oder Problemen: E-Mail [oeide@uni-koeln.de](mailto:oeide@uni-koeln.de)

# Zusammenfassung relationales Datenbankmodell

## Wozu Datenbanken?

- Datenbanken dienen der effizienten elektronischen Datenverarbeitung:
  - Speichern von großen Mengen von Daten: sicher, dauerhaft, widerspruchsfrei, effizient, konsistent
  - Zugriff auf Daten: Eingabe, Update, Abfragen, Auswertung, Zugriffsrechte

## Vor- und Nachteile des relationalen Datenbankmodells

- Am weitesten verbreitetes Datenbankmodell
- + Flexibel
- + Schnell
- + Reduzierung von Redundanz
- + Leicht verständlich
- Performance bei komplexen Abfragen
- Ungeeignet für unstrukturierte Daten
- Single Point of Failure

# Andere Datenbankmodelle (unvollständige Liste)

- Relationale Datenbank
- NoSQL-Datenbanken (not only SQL)
  - Bietet sich an, wenn unstrukturierte Daten vorliegen
  - Nicht nur relationale Daten, sondern auch Textdokumente oder Bilder
  - In-Memory-Datenbanken
- Dokumentenorientierte Datenbank
  - Dokumente als Grundeinheit
  - DBMS: z.B. Elasticsearch, eXist

# Andere Datenbankmodelle (unvollständige Liste)

- Graphdatenbank
  - Graph: Entitäten als Knoten, Beziehungen als Kanten; jeweils mit Eigenschaften
  - Traversieren des Graphen (z.B. Nachbarschaft, kürzester Weg), schneller Zugriff auf vernetzte Daten
  - Skalierbarkeit: Performance unabhängig von Gesamtzahl der Knoten/Kanten
- XML-Datenbank
  - Bietet sich ggf. an, wenn in einem Projekt XML-Daten genutzt werden
  - Mit einer XML-Datenbank kann die Dokumentenstruktur erhalten bleiben, aber trotzdem ein Mehrbenutzerzugriff ermöglicht werden
  - Abfrage z.B. mit XPath möglich