MAD76 Academy: E. SQL Databases

Frank Tränkle*
Hochschule Heilbronn, Germany

November 21, 2025

*frank.traenkle@hs-heilbronn.de

Contents

1   Agenda

- What is SQL? (see Section 2)

- Database for MAD76 race management (see Section 3)

- SQL in Python (see Section 4)

Teaching Objectives

- Understand relational databases

- Understand relations (tables) and relationships

- Learn how to organize data in tables

- Learn how to query and manage data in SQL

- Learn how to use SQLite3

- Learn how to call SQLite from Python

2   What is SQL

- SQL (Structured Query Language) is a programming language for relational databases [1]

- SQL is an ANSI standard

- SQL is supported by many database systems

- SQL was invented by IBM in the 1970s

- SQL is used for querying, updating, and managing data in databases

- SQL can be used from command line, programming languages, and GUIs / web frontends

- SQL has an interpreter

2.1   Relational Database Systems

- A relational database is a collection of data organized in tables (relations)

SimpleCountryTable		
Name	**Capital**	**Continent**
Germany	Berlin	Europe
USA	Washington, D.C.	North America
Japan	Tokyo	Asia
France	Paris	Europe

- Each table consists of rows and columns

- Mathematical concept: *Relational Algebra*

- Design concept: *Entity-Relationship Model (E-R Model)*

  - entities (e.g, country)

  - attributes (e.g., name, capital)

  - relationships (e.g., belongs to continent)

- Tables can be related to each other through *primary* and *foreign keys*

CountryTable

ID	Name	Capital	ContinentTable.ID
1	Germany	Berlin	1
2	USA	Washington, D.C.	2
3	Japan	Tokyo	4
4	France	Paris	1
5	Kenya	Nairobi	3
primary key			foreign key

ContinentTable

ID	Name
1	Europe
2	North America
3	Africa
4	Asia
5	Australia
primary key	

- Relational databases use SQL for querying and managing data

- Commercial database systems: IBM DB2, Oracle Database, Microsoft SQL Server

- Open-source database systems:

  – MySQL (from Oracle)

  – MariaDB (open-source fork of MySQL)

  – PostgreSQL (advanced)

  – SQLite (serverless, alternative to file storage, only for single user, for embedded systems)

- *ACID* properties for reliable *transactions*

  – *Atomicity*: transactions are all-or-nothing

  – *Consistency*: transactions bring the database from valid states to another

  – *Isolation*: transactions do not interfere with each other

  – *Durability*: transactions are permanent, once committed (even in system crashes)

- SQLite has poor performance for multi-user web / cloud applications, since writing operations lock the entire database
  ↝ extremely slow

2.2   Other Database Systems

- NoSQL databases (Not-only SQL)
  - Non-relational
  - No fixed schema
  - Designed for scalability and flexibility
  - Not-only SQL: can support SQL-like queries
  - Use by large-scale web applications
  - Examples:
    * Document stores (e.g., MongoDB, CouchDB)
    * Key-value stores (e.g., Redis, DynamoDB)
    * Column-family stores (e.g., Apache Cassandra, HBase)
    * Graph databases (e.g., Neo4j, Amazon Neptune)
- Object-oriented databases: not so popular as in the 1990s

3   MAD76 Database

- MAD76 database is for race management

- and stores

  - race events

  - drivers

  - cars

  - race results: lap times

- MAD76 database is implemented in SQLite

- and stored in a single file `~labor/src/mad76/mad_ws/install/mbmadmgmt/share/mbmadmgmt/data/mad.db`

- SQLite has

  - the command line interface (CLI) `sqlite3` for SQL commands

    ```
    sqlite3 ~labor/src/mad76/mad_ws/install/mbmadmgmt/share/mbmadmgmt/data/mad.db
    ```

    * `sqlite3` further has special commands, e.g., `.help` or `.tables`

      ```
      .help
      .tables
      ```

    * `sqlite3` can be exited by hitting `Ctrl-D` or typing `.exit`

  - the GUI `sqlitebrowser` for browsing and editing the database

    ```
    sqlitebrowser ~labor/src/mad76/mad_ws/install/mbmadmgmt/share/mbmadmgmt/data/mad.db
    ```

  - application-programming interfaces (APIs) for many programming languages. We use

    * Python package `sqlite3` (see Section 4)

    * Python package peewee: an object-relational mapper (ORM), which maps tables to Python classes (not covered here)

3.1   MAD76 Tables (Relations)

- List the tables in `mad.db` with command `.tables`

```
sqlite3 ~labor/src/mad76/mad_ws/install/mbmadmgmt/share/mbmadmgmt/data/mad.db
.tables
```

- – Table `race` stores a list of race events

- – Table `driver` stores a list of drivers

- – Table `car` stores a list of cars

- Switch on headers to see the tables' column names in next commands

```
.headers on
```

Table `race`

race		
id	name	timestamp
1	MAD76	2025-09-28 09:11:55.926030
...	...	...
INTEGER primary key	VARCHAR(64)	DATETIME

- Display all rows (race events) of table `race`

```
select * from race;
```

- Display table information including column names and datatypes

```
PRAGMA table_info(race);
```

  - INTEGER: integer number

  - VARCHAR(n): string of maximum length n

  - DATETIME: date and time

- The *primary key* is column `id` which must be unique for each row

- *Primary keys* are used as *foreign keys* in other tables for relationships

- You may create a new race event by SQL command `insert`

```
insert into race (name, timestamp) values ('school visit', '2025-09-29 04:00:00');
```

  - Attibute `id` is automatically generated as the next integer number 2, because it is defined as `INTEGER PRIMARY KEY`

- Re-display all rows

```
select * from race;
```

Table `driver`

driver		
id	name	robot
1	Zeus	0
2	Apollo	0
3	Athena	0
4	Hera	0
...	...	...
INTEGER	VARCHAR(64)	INTEGER
primary key		0 for human drivers, 1 for robot drivers

- Display all rows of table `driver`

```
select * from driver;
```

Table `car`

car					
id	minlaptime	maxavgspeed	carid	race_id	driver_id
1	3.99969887733459	0.391122311353683	0	1	1
2	3.7996678352356	0.411522477865219	0	1	2
3	3.3248438835144	0.470151901245117	0	1	3
4	3.57491326332092	0.437629461288452	0	1	4
...	...	...	...	...	...
INTEGER	REAL	REAL	INTEGER	INTEGER	INTEGER
primary key	minimum lap time of individual car in race	maximum speed of individual car inrace	real car id 0 = orange/red car	foreign key to table race	foreign key to table driver

- Display all columns and rows of table `car`

  select * **from** car;

- Display columns `id` and `minlaptime`, only

  select id, minlaptime **from** car;

- Compute all-time best lap time of all cars in all races

  select min(minlaptime) **from** car;

- Compute average lap speed of all cars in all races

```
select avg(maxavgspeed) from car;
```

3.2   Joining Tables

- Tables can be joined by *inner join*, *left join*, *right join*, and *full outer join*

- Joins are based on relationships through primary and foreign keys

- Here: *inner join* of tables `car` and `driver` by matching `car.driver_id = driver.id`

```sql
select car.id, car.minlaptime, driver.name from car inner join driver on car.driver_id = driver.id;
```

- Sort by minimum lap time

```sql
select car.id, car.minlaptime, driver.name from car inner join driver on car.driver_id = driver.id order by car.
    minlaptime;
```

- Alternative short form

```sql
select car.id, car.minlaptime, driver.name from car, driver where car.driver_id = driver.id order by car.
    minlaptime;
```

- This is how the ranking table is computed on MAD76 webpage `http://localhost:8082`

3.3   So why do we split data into multiple tables?

- To reduce data redundancy and improve data integrity

- To organize data in a logical way

- To make it easier to manage and query data

- To allow for more flexible and efficient data access (less memory and CPU usage)

- Bad design would be to store all data in a single table

race_driver_car					
id	race	driver	carid	minlaptime	maxavgspeed
1	Sunday	Athena	0	0.3	0.5
2	Sunday	Zeus	0	0.35	0.4
3	Sunday	Apollo	0	0.4	0.45
4	Monday	Athena	0	0.5	0.6
5	Monday	Zeus	0	0.55	0.65

   – Redundant (duplicate) data: Driver and race names are repeated

   – No data integrity: If a driver's name or race name changes, it must be updated in multiple rows

- Avoid redundancy by *normalization* (*normal forms* 1 to 5) [1]

---

4 SQL in Python

- Package `sqlite3` allows calling SQL commands from Python

```python
import sqlite3 as sql

# Connect to the database
db = sql.connect("/home/labor/src/mad76/mad_ws/install/mbmadmgmt/share/mbmadmgmt/data/mad.db")

# Execute SQL commands
rows = db.execute("select car.id, car.minlaptime, driver.name from car inner join driver on car.driver_id =
    driver.id order by car.minlaptime").fetchall()
for row in rows:
    print(row)

# Close the connection
db.close()
```

- `fetchall()` returns a list of tuples containing the query result

- Each tuple contains one row of the result

- Each element of the tuple contains one column of the result

References

[1]   Edwin Schicker. *Datenbanken und SQL*. 6th. Springer, 2025.