Chapter 7

Databases & SQL

What is the movie with the largest cast? How many directors have directed more than ten movies? The internet movie database (www.imdb.com) contains the answer to such questions, but writing a new program that evaluates the data in a specific way for every such question is laborious. Relational databases can store large amounts of structured data and answer possibly complex questions about it.

7.1 Relational Databases

Definition 7.1 (Table, Row, Column, Database). A table consists of rows, so that each row (data record) contains the same fields, i.e., kinds of entries. When the rows of a table are written line by line, the fields form the columns of the table. Each column is referred to by a descriptive name, and is associated with the type of the respective field, e.g., integer, floating point, string, or a date. A database is a collection of tables.

Remarks:
- In the database context, tables are also called relations, because the entries in each row are related to each other, namely by belonging to the same row.

<table>
<thead>
<tr>
<th>movies</th>
<th>director</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 Angry Men</td>
<td>Sidney Lumet</td>
<td>1957</td>
</tr>
<tr>
<td>Raiders of the Lost Ark</td>
<td>Steven Spielberg</td>
<td>1981</td>
</tr>
<tr>
<td>War of the Worlds</td>
<td>Steven Spielberg</td>
<td>2005</td>
</tr>
<tr>
<td>Manos: The Hands of Fate</td>
<td>Harold P. Warren</td>
<td>1966</td>
</tr>
</tbody>
</table>

Figure 7.2: A database containing a single table called “movies” storing the title, director, and year of release for each movie.

Remarks:
- Databases as we study them are accessed using the so-called structured query language (SQL). Thus they are referred to as SQL or relational databases.
- There are also databases for storing other data, e.g., key-value pairs (Chapter 8), graphs (Chapter 2), or whole documents. Such databases are sometimes called NoSQL databases.
- Some people pronounce SQL letter-by-letter, while others prefer to say “sequel”, which stems from a predecessor with that name.
- MySQL and PostgreSQL are two popular open source SQL databases.
- MongoDB, CouchDB and Redis are popular open source NoSQL databases.
- Like http servers, SQL databases typically run as a daemon process on some server. Client applications connect to the server and authenticate themselves via username and password.
- Multiple users accessing the same database may result in concurrency issues. Some form of concurrency control is necessary!
- Other databases are tailored to single-user processing. They relieve developers from the burden of implementing efficient data structures for relational data. SQLite is one such example, and is used, e.g., in Firefox, Chrome, Android, Adobe Lightroom, and Windows 10.
- How can we store the data from Figure 7.2 using a SQL database?

7.2 SQL Basics

Definition 7.3 (SQL Data Types). SQL defines the following types of columns.
- CHARACTER(m) and CHARACTER VARYING(m) for fixed and variable length strings of (maximum) length m,
- BIT(m) and BIT VARYING(m) for fixed and variable length bit strings of (maximum) length m,
- NUMERIC, DECIMAL, INTEGER, and SMALLINT for fixed point and integer numbers,
- FLOAT, REAL, and DOUBLE PRECISION for floating point numbers,
- DATE, TIME, and TIMESTAMP for points in time, or
- INTERVAL for ranges of time.

Remarks:
- The range of each type includes the special value NULL. Note that NULL is different from the string ‘NULL’, the empty string, and from the number 0 (zero). NULL indicates that the row has no value for the corresponding field.
- For the CHARACTER VARYING type, some database systems support strings of arbitrary length.
Many databases implement more types, e.g., geographic coordinates, IP addresses, geometric objects, or large integers.

All SQL statements end with a semicolon. The SQL language is case insensitive, but by convention keywords are often typed in upper case.

The SQL specification is over 600 pages long. To add insult to injury there are lots of vendor specific “SQL dialects”, i.e., modifications and extensions.

However, the basic set of commands for creating, manipulating, and querying tables are largely the same across database implementations.

### Listing 7.4
Creating the database moviedb containing a table movies.

```sql
1:   CREATE DATABASE moviedb;
2:   USE moviedb;
3:   CREATE TABLE movies (title CHARACTER VARYING(200) NOT NULL,
                          director CHARACTER VARYING(200) DEFAULT 'Steven Spielberg',
                          year INTEGER);
```

CREATE DATABASE database-name;
Additional parameters allow to set database-specific options, e.g., user-based permissions, or default character sets for text strings. How a database is opened depends on the implementation. Listing 7.4 shows how to do it in MySQL.

CREATE TABLE table-name (field-name type, field-name type, . . .);
To enforce that all rows have a value for a particular field, one can add NOT NULL to the type when creating the table. Fields have a default value, which is NULL if not specified by adding DEFAULT value to the type description.

Remarks:

- There are also GUI and web-based client applications (that execute locally or on an http-server, respectively) and offer access to the database in a more intuitive manner than the classic command line tools. Examples for web-based interfaces are phpPgAdmin and phpMyAdmin for PostgreSQL and MySQL, respectively.

- Such tools are especially helpful for creating the databases and tables. They also feature importing data from various formats, e.g., CSV files, instead of using SQL statements to populate the tables.

### Listing 7.5
Populating the movies table with data.

```sql
4:   INSERT INTO movies (title, director, year) VALUES ('12 Angry Men', 'Sidney Lumet', 1957),
      ('Raiders of the Lost Ark', DEFAULT, 1981),
      ('War of the Worlds', DEFAULT, 2005), ...
      ('Manos: The Hand of Fate', 'Harold P. Warren', 1966);
```

Listing 7.6
Querying the movies table.

```sql
5:   SELECT * FROM movies;
6:   SELECT * FROM movies WHERE director = 'Steven Spielberg';
7:   SELECT title FROM movies WHERE year BETWEEN 1990 AND 1999;
8:   SELECT * FROM movies WHERE title IS NULL OR director IS NULL;
9:   SELECT title, director FROM movies WHERE title LIKE '%the%';
```

SELECT field-name, . . . FROM table-name WHERE condition;
Lists all specified fields of all rows in the table that fulfill the condition. The special field * lists all fields. The WHERE condition may be omitted to list the whole table. A condition can include comparisons (<, >, =, <>), between fields constants. The special value NULL can be tested with IS NULL. Conditions can be joined using parenthesis and logic operators like AND, OR, and NOT. Strings can be matched with patterns using field-name LIKE pattern. In the pattern, an underscore (_) matches a single character, whereas % matches arbitrarily many.

### Listing 7.7
Aggregation with SQL.

```sql
10:  SELECT MIN(year) FROM movies;
11:  SELECT AVG(year) FROM movies WHERE director='Sidney Lumet';
12:  SELECT COUNT(*) FROM movies;
13:  SELECT COUNT(DISTINCT director) FROM movies;
```

SELECT aggregate, . . .
Functions for aggregation include AVG to compute the average of a certain field, MIN and MAX for the minimum and maximum value, SUM for the sum of a field, and COUNT to count the number of occurrences. In an aggregation, the keyword DISTINCT indicates that only distinct values should be considered.
7.3 Modeling

The way our example table from Figure 7.2 is designed results in lots of duplicate data—the director’s name is stored anew for each row, and two directors with the same name cannot be distinguished. The situation worsens when we want to store the cast of each movie. Clearly the way we modeled our data can be improved. Entity-Relationship (ER) diagrams are a tool to find good representations for data.

Remarks:
• Query 14 in Listing 7.8 reports how many movies of each director are in the database, and query 16 breaks the same down by year.

Listing 7.9 Updating and removing rows.
17: UPDATE movies SET title = 'Star Wars Episode IV: A New Hope' WHERE title = 'Star Wars';
18: DELETE FROM movies WHERE title = '';

UPDATE table_name SET field-name = value, . . . WHERE condition;
 Updates the specified fields in all rows fulfilling the condition.

DELETE FROM table-name WHERE condition;
 Removes all rows fulfilling the condition from the table.

7.3 Modeling

The way our example table from Figure 7.2 is designed results in lots of duplicate data—the director’s name is stored anew for each row, and two directors with the same name cannot be distinguished. The situation worsens when we want to store the cast of each movie. Clearly the way we modeled our data can be improved. Entity-Relationship (ER) diagrams are a tool to find good representations for data.

7.3 Modeling

The way our example table from Figure 7.2 is designed results in lots of duplicate data—the director’s name is stored anew for each row, and two directors with the same name cannot be distinguished. The situation worsens when we want to store the cast of each movie. Clearly the way we modeled our data can be improved. Entity-Relationship (ER) diagrams are a tool to find good representations for data.
### 7.4. JOINS

How can we access the data, which is now scattered across multiple tables?

#### Remarks:
- The same scheme can be used for 1-to-1 and 1-to-n relations. However, one may also include the relation in the table storing the entity on the 1-side.
- Tables dedicated to capturing relations are often called join tables.

#### 7.4 Joins

How can we access the data, which is now scattered across multiple tables?

**Remarks:**
- The same scheme can be used for 1-to-1 and 1-to-n relations. However, one may also include the relation in the table storing the entity on the 1-side.
- Tables dedicated to capturing relations are often called join tables.

**Listing 7.14** Example query that returns the table depicted in Figure 7.15.

```sql
SELECT movie.title, director.name AS director, movie.year FROM movie
INNER JOIN director ON movie.director id = director.id;
```

**Remarks:**
- Similarly, a 1-to-1 relation can be turned into an attribute of one of the entities.
- Tables dedicated to capturing relations are often called join tables.

**Listing 7.16** Finding the 10 movies with the largest cast.

```sql
SELECT movie.title, COUNT(*) AS cast_size FROM cast INNER JOIN movie ON cast.movie_id = movie.id
GROUP BY movie.id ORDER BY cast_size DESC LIMIT 10;
```

**Remarks:**
- The query from Listing 7.16 uses a LIMIT clause to return only the ten first entries of the sorted results.
- An INNER JOIN where the condition is TRUE returns the Cartesian product of both tables. This special case can also be obtained with the CROSS JOIN clause.
- An inner join will only return those rows of one table that have a matching row (that satisfies the condition) in the other table. For example, in Listing 7.14, a director with id 5 would not appear in the result if there are no movies which have director id=5.
- If you want unmatched rows to appear in the result, you need to use an OUTER JOIN.

**Listing 7.17** Example query that returns the table depicted in Figure 7.18.

```sql
SELECT movie.title, director.name AS director, movie.year FROM movie
RIGHT OUTER JOIN director ON movie.director_id = director.id;
```

**Remarks:**
- In a query, one can create aliases for field and table names using the AS keyword, see Listing 7.14.
- The result of a JOIN clause can be ordered, fields can be aggregated and grouped, and conditions can be added using WHERE clauses.
- For example, we can combine joins and aggregations to answer our initial question of which movie has the largest cast.
### 7.5 Keys & Constraints

What is stopping us from inserting a row in the cast table that contains an actor_id or a movie_id that does not exist? Or from creating a director with a duplicate id?

#### Definition 7.20 (Key).
In a table, a column (or set of columns) is a **unique key** if the corresponding values uniquely identify the rows within the table. The **primary key** of a table is a designated unique key. A **foreign key** is a column (or set of columns) that references the primary key of another table.

**Remarks:**
- SQL databases can automatically enforce these constraints. For example, a row containing a foreign key can only be inserted if it references an existing primary key. Vice versa, a row may only be removed if its primary key is not referenced by any foreign key.

#### Listing 7.21 Adding constraints to the database.

```sql
1. ALTER TABLE cast ADD CONSTRAINT UNIQUE (actor_id, character, movie_id);
2. ALTER TABLE directors ADD PRIMARY KEY id;
3. ALTER TABLE movies ADD FOREIGN KEY (director_id) REFERENCES directors;
```

- **ALTER TABLE table ADD CONSTRAINT UNIQUE (field-name, . . .);** Any two rows must differ in at least one of the specified fields.
- **ALTER TABLE table ADD PRIMARY KEY (field-name, . . .);** Sets the specified fields as the primary key for the table. Any two rows must differ in at least one of the specified fields. The entries in these fields must not be NULL.
- **ALTER TABLE left-table ADD FOREIGN KEY (field-name, . . .) REFERENCES right-table;** Ensures that the values in the specified fields in the left table are the primary key of a row in the right table.

**Remarks:**
- Constraints for new tables can also be set using CREATE TABLE.
- Other ALTER TABLE queries add different constraints (e.g., checking that an integer field contains only certain values), remove constraints, and change the name, type or default value of fields.
- To ensure that checking constraints and searching for data is fast, databases rely on index data structures.

#### 7.6 Indexing

**Definition 7.22 (Index).** In the database context, an **index** is a data structure that speeds up searching for rows with specific values.
Remarks:

- Without an index data structure, rows with a specific value can only be found by scanning through the whole table.
- In Chapter 8 you learned how hash tables can retrieve the row associated with a key quickly. Many databases implement hash tables as one possible index data structure.

Listing 7.23 Adding a hash table index to our database.

```sql
CREATE INDEX directorid ON director USING HASH (id);
```

Remarks:

- The director associated with a movie is now found quickly when performing a join.
- Some database implementations automatically create index data structures to speed up queries that involve frequently used fields.
- Index data structures have a name—“directorid” in Listing 7.23. This is for referencing it later, e.g., if one decides to delete the index.
- Hash tables scatter the data across the storage (volatile or persistent), and it is likely that every access incurs overhead. Many database queries require scanning through ranges of the data sequentially. For example, when searching the movies from 2000–2005. Thus, accessing supposedly closeby rows requires accessing items at many different places.
- B+ trees are a data structure designed to minimize the amount of I/O operations for both searching and scanning.

Listing 7.24 Adding a B+ tree index to our database.

```sql
CREATE INDEX movieyear ON movies USING BTREE (year);
```

Definition 7.25 (B+ Tree).
A B+ Tree of order \( b \) is a rooted search tree mapping keys to rows. B+ trees are balanced, i.e., all leaf nodes are at the same depth.

Every non-leaf node has between \( \lceil b/2 \rceil \) and \( b \) children. A non-leaf node \( v \) with \( k \) children contains exactly \( k - 1 \) keys, in sorted order. The \( i \)th key of a non-leaf node \( v \) is identical to the smallest key in the subtree rooted at \( v \)'s \( (i + 1) \)st child.

Leaf nodes contain all keys inserted into the tree, together with a pointer which points to the row associated with that key. Every leaf has pointers to at least \( \lceil (b - 1)/2 \rceil \) and at most \( b - 1 \) table rows. Additionally, every leaf \( w \) has a pointer which points to its next sibling \( w' \).

Algorithm 7.27 B+SplitUp(\( v, k \))

1. Given a B+ tree, a key \( k \), and a node \( v \)
2. Create a new node \( v' \)
3. Distribute \( k \) and the keys in \( v \) among \( v \) and \( v' \) s.t. \( v' \) gets the larger keys and both nodes are half filled
4. Let \( k' \) be the smallest key in \( v' \)
5. Let \( p \) be \( v \)'s parent
6. if \( p \) is full then
7. SplitUp(\( p, k' \))
8. end if
9. Insert \( k' \) with child \( v' \) at node \( p \)
Remarks:

- If the root node is split into two nodes \(v, v'\), then a new root \(r\) containing key \(k\) and \(v\) as children is created, and the recursion stops.
- Inserting a key \(k\) is now performed by first making room using \(B+\) SplitUp if necessary, and then inserting \(k\) at the leaf.

Algorithm 7.28 B+Insert(\(k, r\))

1: Given a B+ tree, a key \(k\), and a root \(r\)
2: Perform a search for \(k\) to find the leaf \(v\) at which \(k\) must be inserted
3: if \(v\) contains \((b-1)\) keys then
4: \(B+\) SplitUp(\(v, k\))
5: Replace child of key \(k\) with row \(r\) in node \(v\)
6: else
7: Insert key \(k\) with row \(r\) into node \(v\)
8: end if

Remarks:

- Vice versa, when deleting a key, nodes with too few keys need to be filled up or removed from the tree.

Algorithm 7.29 B+MergeUp(\(v\))

1: Given a node \(v\) containing less than \((b-1)/2\) keys
2: Let \(l\) and \(r\) be the left and right sibling of \(v\)
3: if \(l\) contains more than \((b-1)/2\) keys then
4: Move largest key \(x\) from \(v\)'s left sibling to \(v\)
5: Update key in parent corresponding to \(x\) to \(x\)
6: else if \(r\) has more than \((b-1)/2\) keys then
7: Let \(x\) and \(y\) be the smallest and second-smallest key in \(r\), respectively
8: Move smallest key \(x\) from \(r\) to \(v\)
9: Update key in parent corresponding to \(x\) to \(y\)
10: else
11: Merge all keys of \(v\) and one of \(v\)'s siblings (use the node that is further to the left to store the keys)
12: Remove the now empty node and its corresponding key from \(v\)'s parent
13: if \(v\)'s parent contains less than \((b-1)/2\) keys then
14: \(B+\) MergeUp(\(p\))
15: end if
16: end if

Remarks:

- If \(v\) does not have a left or right sibling, the corresponding if statement is ignored.

Algorithm 7.30 B+Delete(\(k\))

1: Given a B+ tree and a key \(k\)
2: Perform a search for \(k\) to find the leaf \(v\) containing \(k\)
3: Remove \(k\) from \(v\)
4: if \(v\) contains less than \((b-1)/2\) keys then
5: \(B+\) MergeUp(\(v\))
6: end if

Remarks:

- The properties of B+ trees ensure that every node has at least one sibling. Thus, the merge operation (Lines 11–15) always has two nodes to work with.
- If no keys can be "borrowed" from a sibling, the merge may propagate until the last two children of the root node are merged into one node. In that case the root node is replaced by the merged node, decreasing the height of the tree by 1.

7.7 Transactions

Definition 7.31 (Transaction). A database transaction is a sequence of statements that is executed atomically, i.e. the transaction appears to be instantaneous to an observer.

Remarks:

- Why would we need transactions? Consider a bank managing customer’s accounts using a database system. Alice wants to calculate the liquid assets, and Bob wants to make a money transfer.

Listing 7.32 Concurrency issues in databases.

Alice’s statement:
1: SELECT SUM(balance) FROM accounts;

Bob’s statements:
2: UPDATE accounts SET balance=balance+100 WHERE customer='Bob';
3: UPDATE accounts SET balance=balance−100 WHERE customer='Jim';
7.8 Programming with Databases

How do you write an application that relies on a SQL database to store data? Should you construct the necessary SQL statements by manipulating strings, send them to the SQL server, and then parse the result?

Remarks:

- Writing such a SQL client is one possibility, but this is error-prone. The compiler used for the application will not be able to detect errors made in the SQL statements. Moreover, the declarative SQL most likely does not mix well with the programming language chosen for the application.
- One way to mitigate these issues in object oriented programming languages is object/relational mapping.

7.8. PROGRAMMING WITH DATABASES

Remarks:

- Assuming that the database uses multiple threads or processes to process queries, Alice’s query may be CHF 100 short.
- To execute the queries atomically, both Alice and Bob can use transactions.

BEGIN TRANSACTION; statement; . . . ; END TRANSACTION;

Executes the statements atomically.

Remarks:

- One way to implement transactions is to keep track of all fields read from and written to (the read- and write-set, respectively). Then, before a transaction ends, the database system checks whether another transaction wrote to any value in the read-set. If the read-set is unchanged, the write-set can be applied atomically, e.g., by using a global lock.
- SQL offers different so-called isolation levels. The isolation level defines when writes of one transaction become visible to others. The above technique implements the repeatable reads level, ensuring that read values were committed before and are not written by another transaction.
- Consider some transaction A that selects all years between 1999 and 2004. What happens if another transaction B concurrently inserts an entry for the year 2000? In the repeatable reads isolation level, A may not see B’s data if B’s insert is scheduled after A read all other entries for the year 2000, and A would still be allowed to finish. Repeatable reads do not ensure atomicity . . .
- The highest isolation level is called serializable. This level ensures that the transactions behave “as if they were executed in some sequential order”, possibly at the cost of low concurrency.

Definition 7.33 (Object/Relational Mapping). Object/Relational Mapping (ORM) is a design pattern used in object oriented programming to store objects in and retrieve them from relational (SQL) databases.

Remarks:

- In the simplest case, an ORM simply maps a class to a table. An object then corresponds to a row, and the object’s attributes correspond to the row’s fields.
- The ORM takes care of storing and retrieving object in the database and performs type conversions where necessary. It provides object oriented abstractions for database queries involving WHERE and other clauses. ORMs also remove boilerplate code, i.e., setting up the SQL connection, error handling, data conversion, etc.
- This way no—or only very little—SQL code “leaks” into the object-oriented program.
- Popular ORMs include SQLAlchemy for Python, ActiveRecord for Ruby, Hibernate for Java, and the Entity Framework for .NET.

Listing 7.34 Using Hibernate for Java to change the personal information of an existing director.

```java
1. Director director = session.load(Director.class, new Long(3464377));
2. // director: id = 3464377, name = "Larry Wachowski", gender = "m"
3. director.setName('Lana Wachowski');
4. director.setGender('f');
5. commit();
```

Remarks:

- The ORM needs to know how it should translate between objects and rows. For that, many ORM implementations allow to specify the database layout using object oriented methods.
- Many ORM mappers also support creating the database using the object oriented specification. This ensures that the database and what the ORM expects are kept in sync.
- What if you need to add or remove a column without deleting and re-creating the database? There are so-called migration tools that facilitate this process.
- Some concepts from object oriented programming are difficult to model with database concepts, and vice versa. The problems arising from combining these two paradigms are called the Object-relational impedance mismatch.
Chapter Notes

In 1970, Edgar F. Codd proposed the relational database model [5] while working at IBM research. Later in the 70s, another group at IBM developed SQL’s predecessor SEQUEL (Structured English QUEry Language) [3]. After being renamed SQL due to trademark issues, it was standardized by the ISO in 1987 and later revised [7]. Other companies started developing relational databases, and nowadays there are many SQL databases implementing different feature sets to choose from. Around the same time, ER diagrams were conceived as a modeling tool [2, 4]. The Unified Modeling Language (UML), first standardized by the ISO in 1995 [8] and revised in 2012, also includes diagrams that model databases.

B Trees were invented in 1970 [1] for use in file systems. Many variants were studied, among them B* Trees [9], in which at most 1/3 of the memory is unused instead of 1/2 for B Trees. People soon realized that (also for file systems) scanning subsequent rows is an important operation. B+ Trees require at most one I/O operation to find the next element, cf. [9, 6].

Techniques from database systems can also be found in other areas of computer science. Transactions as a parallel programming model have been adopted for other programming languages under the term transactional memory. Ideas developed to ensure that database transactions appear atomic w.r.t. writing data to disk were adopted by general purpose file systems under the name journaling.

This chapter was written in collaboration with Jochen Seidel.

Bibliography


