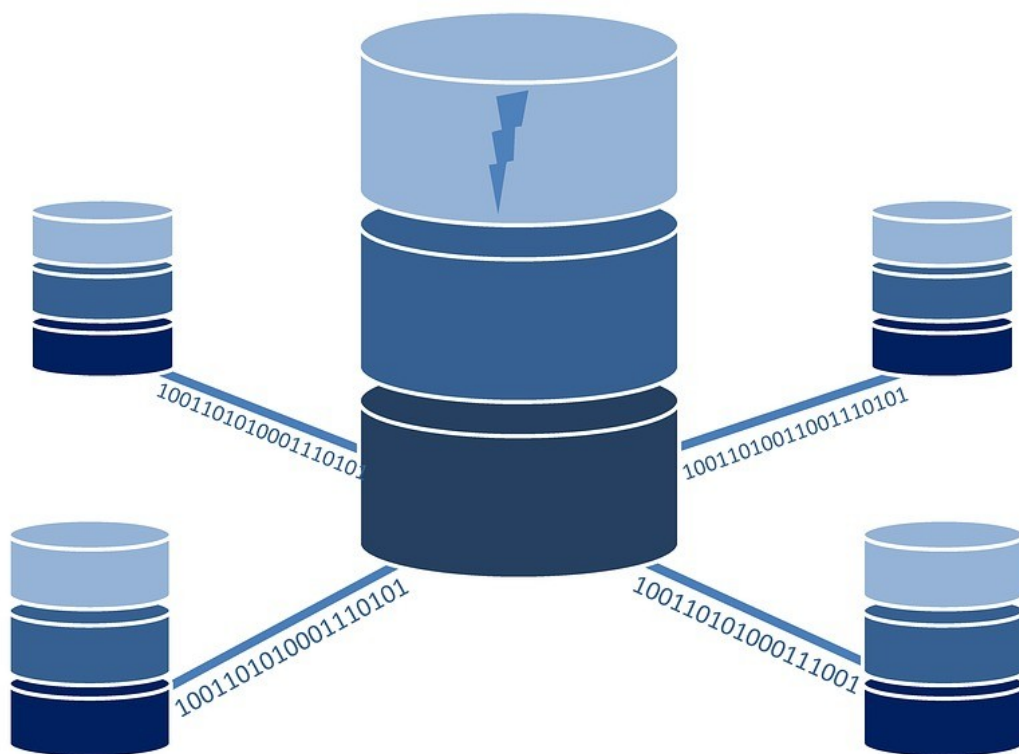


Datenbanken in Web-Applikationen einbinden

Lenny Angst
15.11.2022 - 20.12.2022



Inhalt

1 Design.....	3
1 Layers.....	3
2 N-Tier-Architekturen.....	4
3 MVC-Architektur.....	6
2 Sicherheit.....	8
1 SQL Injection.....	8
2 Manipulation des Clients.....	8
3 Cross-Site-Scripting.....	9
3 Realisierung.....	10
1 Datenmodell.....	10
2 Unterteilung von SQL-Code.....	13
4 Changemanagement.....	15
1 Datenbank.....	15
2 Code.....	15
5 Testing.....	17
1 Während der Entwicklung.....	17
2 Im produktiven Betrieb.....	17
3 Datenbank.....	17

1 Design

Eine sorgfältige Planung ist insbesondere für grössere Projekte von grosser Wichtigkeit. Ein Teil davon ist das Software-Design. Hierbei bestimmt man, wie die Software programmiert und strukturiert werden soll und – falls die Software in einer Weise kommuniziert – wie sie sich mit ihren Partnern austauscht.

Die Multi-Tier-Architektur, auch als N-Tier-Architektur oder Schichtenarchitektur bekannt, ist eine Software-Architektur, bei der eine bestimmte Applikation in verschiedene Komponenten aufgeteilt wird, um so voneinander unabhängige Schichten zu erreichen. Dabei gibt es die 1-Tier, 2-Tier, bis hin zur N-Tier-Architektur, wobei N für beliebig viele Schichten stehen kann. Die 3-Tier-Architektur ist dabei am weitesten verbreitet.

1 Layers

Jede dieser Ebenen steht für einen Layer. Es gibt den *Presentation-Layer*, *UI-Layer*, *Data-Access-Layer*, *Business-Logik-Layer* und den *Data-Storage-Layer*.

1 Presentation-Layer

Der Presentation-Layer ist die Ebene, welche für die Anzeige und Visualisierung der Daten verantwortlich ist. Sie zeigt unter anderem Tabellen oder Grafiken wie Diagramme, Videos und Bilder an.

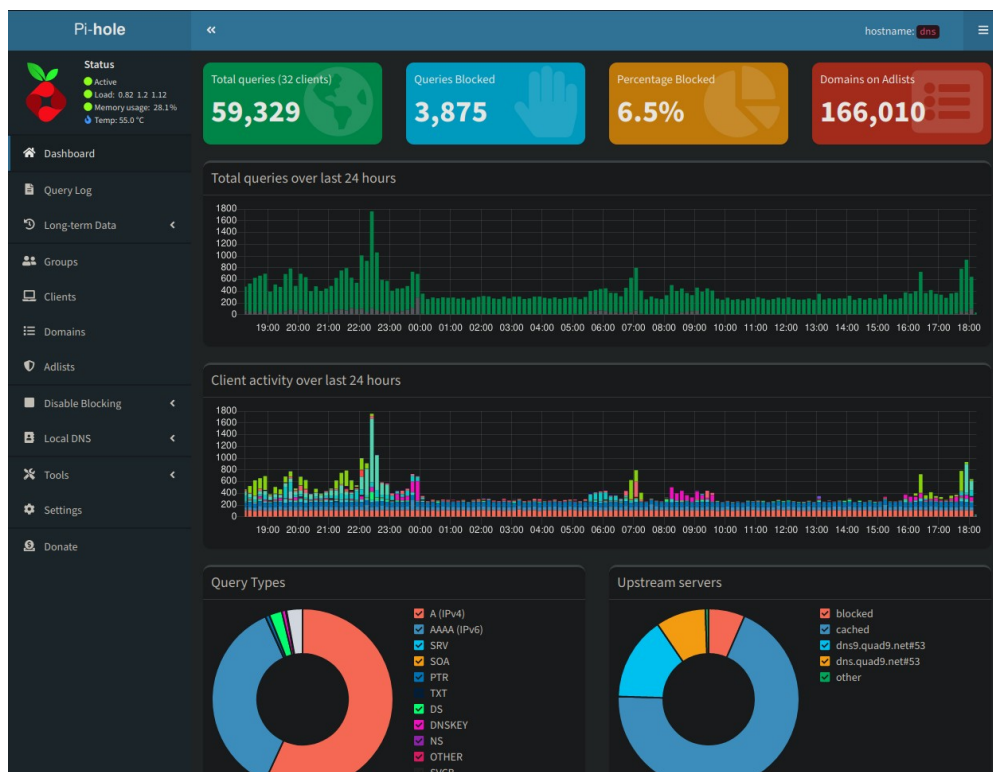


Illustration 1: Beispiel eines Presentation-Layers: das Dashboard meines lokalen DNS-Servers.

2 UI-Layer

Für die Interpretation von Benutzerinteraktionen ist der UI-Layer zuständig. Dort wird festgelegt, was passiert, wenn der Nutzer etwas anklickt oder etwas eingibt.

Ein Nutzer klickt beispielsweise einen «Senden»-Knopf. In diesem Layer wird nun also bestimmt, wie und wann die zu sendenden Daten an den Server geschickt werden. Dieser Layer ist ebenfalls dafür zuständig, die Daten vom Server zu holen.

Der UI-Layer kann auch auf einem Server laufen, beispielsweise wenn mittels PHP, Thymeleaf oder anderen Templating-Technologien fertiges HTML und CSS zur Anzeige produziert wird. Heutzutage ist dies aber immer weniger der Fall und der Client greift direkt auf eine Schnittstelle (bspw. REST oder GraphQL) zu.

3 Business-Logik-Layer

Im Business-Logik-Layer sind wir für die Datenvalidierung, Sicherheit, Manipulation und Verarbeitung zuständig. Wenn etwa eine Bestellung an den Server gesendet wird, werden hier alle weiteren Aktionen, wie die Zahlung, Rechnungsstellung etc., koordiniert.

4 Data Access-Layer

Der Data Access Layer ist eine Schnittstelle zwischen der Datenbank und der Anwendung. Hier werden Funktionen implementiert, um auf die Datenbank zuzugreifen. Hier wird oft auch mit Interfaces bzw. dem Dependency-Injection-Pattern gearbeitet, sodass Abfragen für verschiedene Datenbanksysteme implementiert und einfach ausgetauscht werden können.

5 Data Storage-Layer

Datenbankserver und andere Speicherarten (zum Beispiel auch simple Textdateien) bilden den Data Storage-Layer. Die Daten werden hier persistent und auch möglichst strukturiert abgespeichert, sodass der Zugriff darauf möglichst unkompliziert ist.

2 N-Tier-Architekturen

1 1-Tier-Architektur

Die 1-Tier-Architektur ist heutzutage nicht mehr wirklich anzutreffen. Bei dieser Variante sind alle Layer an einem Ort zusammengefasst. Dies ist beispielsweise bei Textverarbeitungsprogrammen der Fall. Im Allgemeinen sind solche Architekturen jedoch anfällig für Performanzprobleme oder Sicherheitslücken. Daher wird dieses Verfahren kaum noch eingesetzt.

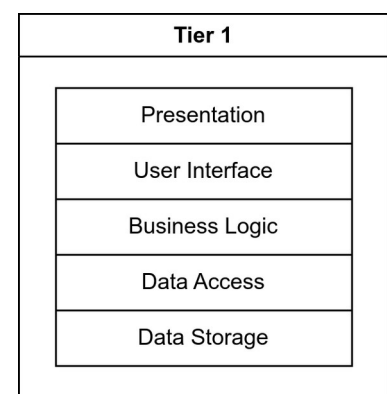


Illustration 2: Aufbau einer 1-Tier-Architektur

2 2-Tier-Architektur

Die 2-Tier-Architektur besteht, wie der Name bereits andeutet, aus zwei Schichten. Oft wird die 2-Tier-Architektur auch als Client-Server-Architektur bezeichnet. Die Oberschicht kann auf die Unterschicht zugreifen, nicht aber umgekehrt. Bei der niedrigeren Schicht handelt es sich um einen Dienstanbieter, der als Server implementiert ist. Ein mögliches Beispiel wäre eine einfache Website mit Formular, welche in PHP implementiert ist. Der PHP-Code greift dabei direkt auf die Datenbank zu.

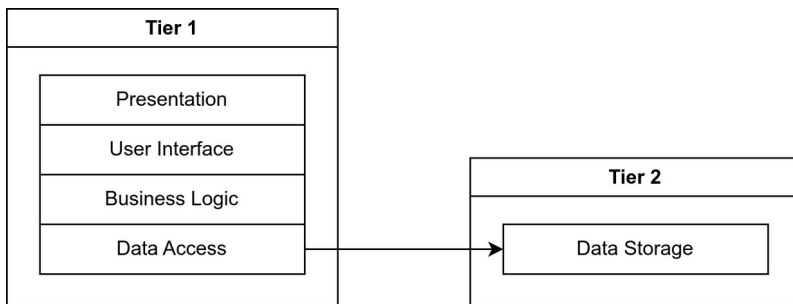


Illustration 3: Aufbau einer 2-Tier-Architektur

3 3-Tier-Architektur

Die Architektur mit den drei Schichten, die 3-Tier-Architektur, hat sich durchgesetzt. Die Präsentationsschicht und die Benutzerschnittstelle befinden sich beim Nutzer. Der Business-Logik-Layer und der Data Access-Layer werden als Teil des Anwendungsservers ausgeliefert. Zuletzt ist der Data-Storage-Layer zu beachten. Letzterer befindet sich auf einem anderen System. Die Performance wird durch diese Aufteilung deutlich verbessert. Durch die Aufteilung können auch verschiedene voneinander unabhängige Clients auf eine etwaige API zugreifen.

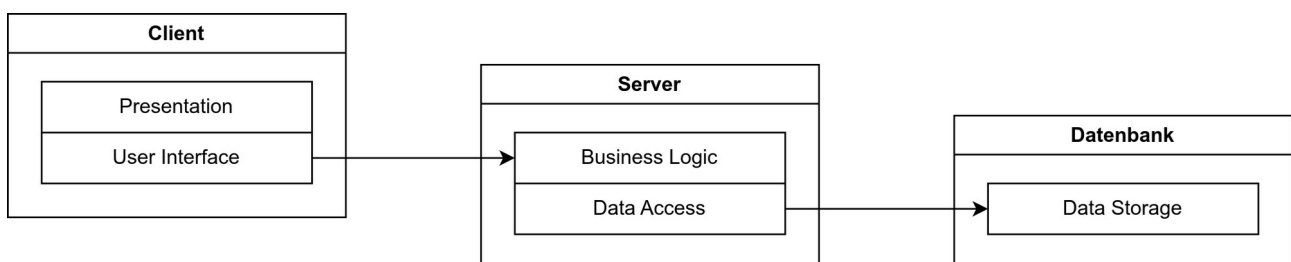


Illustration 4: Aufbau einer 3-Tier-Architektur

4 4-Tier-Architektur

Die 4-Tier-Architektur unterscheidet sich nicht stark von der 3-Tier-Architektur. Hierbei geht es vor allem um die Trennung von einzelnen Komponenten. Populär ist es insbesondere, die Businesslogik vom API-Controlling zu trennen. Als Beispiel: Die Endpunkte einer REST API laufen auf einem System, während die gesamte Businesslogik auf einem anderen System läuft.

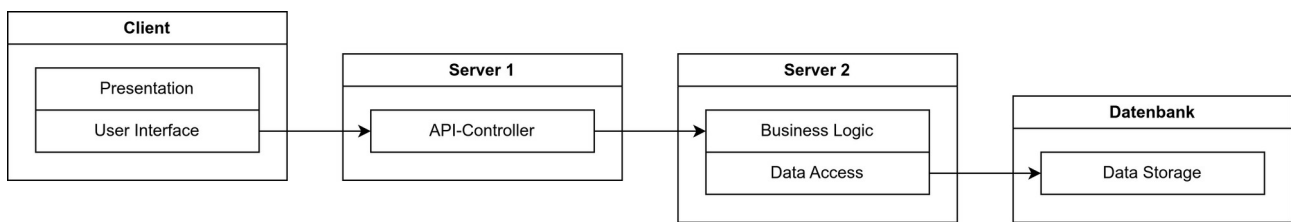


Illustration 5: Beispielhafter Aufbau einer 4-Tier-Architektur

5 5-Tier-Architektur

Die 5-Tier-Architektur ist ähnlich zur vorangehenden. Auch hier wird einfach etwas mehr getrennt. Zusätzlich könnte man den Datenbankzugriff von der Businesslogik abtrennen. Damit haben wir etwa folgende abgetrennte Dienste:

- Client
- REST API Endpunkte
- Businesslogik
- Data-Access
- Datenbank

3 MVC-Architektur

Das MVC-Pattern unterteilt eine Applikation in drei wesentliche Teile.

1 Model

Das Model ist unabhängig von View und Controller. Die Daten des Models werden später auf der View angezeigt. Also alles, was der User sehen kann. In manchen Implementierungen des MVC-Modells übernimmt die Businesslogik die Aufgabe, die Daten zu ändern.

2 View

Die View stellt die Daten schlussendlich gut verständlich und ggf. auch visuell dar. Sie ist somit die Schnittstelle zwischen dem Benutzer und der gesamten Applikation.

3 Controller

Der Controller ist der Vermittler zwischen View und Model. Er liefert etwa die aufbereiteten Daten, welche die View anzeigen kann, überprüft die Berechtigungen, verarbeitet Daten und führt Änderungen auf dem Model durch.

4 Nutzung in der Realität

Die MVC- und Tier-Architekturen sind keine strikten Vorschriften, sondern grobe Vorschläge. Sie bieten einen guten Grundsatz, worauf man sein individuelles Projekt

aufbauen kann. Jedes Projekt hat seine eigenen Bedürfnisse und andere Prioritäten. Die beste Architektur gibt es also nicht wirklich und die Umsetzung kann auch mal von der Norm abweichen.

2 Sicherheit

In der heutigen vernetzten Welt ist Sicherheit in der Informationstechnik wichtiger denn je. Auch eine simple Webapplikation kann von solchen Lücken betroffen sein. Umso wichtiger ist es, best möglichst vorzusorgen und die Wahrscheinlichkeit somit zu verringern.

Im Anschluss werden die am weitesten verbreitete Fehler vorgestellt.

1 SQL Injection

Bei einer SQL-Injection wird versucht, bösartiger Code mittels SQL an die Datenbank zu senden, um sich zum Beispiel unautorisiert in einem anderen Account anzumelden. Solche Attacken sind einfach vorzubeugen, passieren aber dennoch ständig.

Um SQL-Injections in der eigenen Applikation zu verhindern, sollten alle Eingaben des Nutzers vorher bereinigt werden. ORMs auf höherer Abstraktionsstufe machen dies heute schon automatisch, greift man aber manuell direkt auf die Datenbank zu, muss dies selbst getan werden.

Ein Beispiel: Für ein Login wird im Backend folgendes SQL genutzt:

```
SELECT * FROM users WHERE email = '$EMAIL' AND password = '$PASSWORD';
```

Ein Angreifer könnte sich hier ganz einfach als jemand anderes einloggen, indem er diesen speziellen Wert als E-Mail angibt:

```
admin@website.ch'; --
```

Die Abfrage sieht zusammengesetzt dann also so aus:

```
SELECT * FROM users WHERE email = 'admin@website.ch'; --'
```

Ist die Seite schlecht programmiert, so kann der Angreifer sich ohne Passwort einfach in den Admin-Account anmelden.

Im obigen Beispiel hätte erst einmal geprüft werden sollen, ob der Input überhaupt eine E-Mail ist. Als zweiten Schritt könnte man den Input bereinigen, indem man etwa die Apostrophe und Semikolon entfernt. Mit diesen relativ einfachen Massnahmen wäre die Applikation schon um einiges besser geschützt.

2 Manipulation des Clients

Da der Client beim Nutzer läuft, wissen wir – insbesondere bei der 2+-Tier-Architektur – nicht, was uns im Backend erwartet. Der Nutzer könnte den Client beliebig manipulieren oder gar seinen ganz eigenen schreiben. Etwaige clientseitige Validierungen können somit vom Nutzer umgangen werden. Für uns heisst das: Alle Eingaben validieren wir auch serverseitig noch einmal.

Ein Beispiel: Bei einem Registrierungsformular muss der Nutzer eine menschliche Verifikation durchführen, indem er eine simple Rechenaufgabe löst. Diese Aufgabe wird allerdings nur clientseitig überprüft. Will nun jemand unseren Server mit Registrationen überfluten, so muss er lediglich automatisiert Anfragen an den Registrationen-Endpunkt senden, ohne, dass je eine Rechnung gelöst werden muss.

Um dies zu verhindern, muss also auch die Rechenaufgabe im Backend überprüft werden.

3 Cross-Site-Scripting

Cross-Site-Scripting oder kurz XSS ist eine Lücke, von der praktisch jede Website mit Input-Feldern betroffen sein kann. Dabei wird ein böses Script vom Angreifer in einen Input, etwa für einen Blogpost, eingefügt. Dies kann im simpelsten Fall ein einfaches Stück JavaScript in normalen HTML-Script-Tags sein.

Ruft ein ahnungsloser Nutzer nun die Seite mit dem Blogpost auf, wird in seinem Browser JavaScript ausgeführt, das gar nicht vom eigentlichen Seitenbetreiber stammt, sondern vom Angreifer. Dadurch können dann etwa die Cookies gestohlen werden, worin auch sensitive Daten wie Login-Tokens gespeichert sein könnten.

Der Auslöser für diese Sicherheitslücke ist auch hier wieder die fehlende Bereinigung aller Input-Strings, das Stück JavaScript ist schliesslich problemlos durch alle Ebenen gekommen, bis hin zu einem anderen Nutzer.

PHP bietet zur Bereinigung von HTML-Zeichen unter anderem die vorgefertigte Funktion **htmlspecialchars**.

3 Realisierung

1 Datenmodell

1 Konzeptionelles Datenbankdesign

Konzeptionelles Datenbankdesign bezieht sich auf den Prozess der Planung und Gestaltung einer Datenbank, bevor sie tatsächlich implementiert wird. Es umfasst die Identifizierung der Daten, die in der Datenbank gespeichert werden sollen, und die Festlegung der Beziehungen zwischen diesen Daten.

Ein wichtiger Teil des konzeptionellen Datenbankdesigns ist die Überlegung, wie die Daten am besten strukturiert werden, um die Anforderungen der Nutzer zu erfüllen. Dazu gehört die Wahl der richtigen Datenmodellierungstechniken, wie das Entity-Relationship-Modell. Das ER-Modell hilft dabei, die Beziehungen zwischen den verschiedenen Entitäten zu visualisieren und zu beschreiben. Da der Kunde bzw. Auftraggeber eng in diesen Prozess eingebunden ist, sollte dieser diese Diagramme ebenfalls verstehen können.

Ein weiterer wichtiger Aspekt des konzeptionellen Datenbankdesigns ist die Normalisierung der Daten. Durch die Normalisierung werden Redundanzen und Inkonsistenzen in den Daten minimiert, was zu einer effizienteren und zuverlässigeren Datenbank führt. Es gibt verschiedene Normalformen, die beim konzeptionellen Datenbankdesign berücksichtigt werden können, um sicherzustellen, dass die Daten auf eine strukturierte und logische Weise gespeichert werden.

Insgesamt ist das konzeptionelle Datenbankdesign von entscheidender Bedeutung für den Erfolg einer Datenbank. Es hilft dabei, die Daten sinnvoll zu strukturieren und die Anforderungen der Nutzer zu erfüllen, während gleichzeitig Redundanzen und Inkonsistenzen minimiert werden. Eine gut konzipierte Datenbank ist effizient, zuverlässig und leicht zu verstehen und zu pflegen.

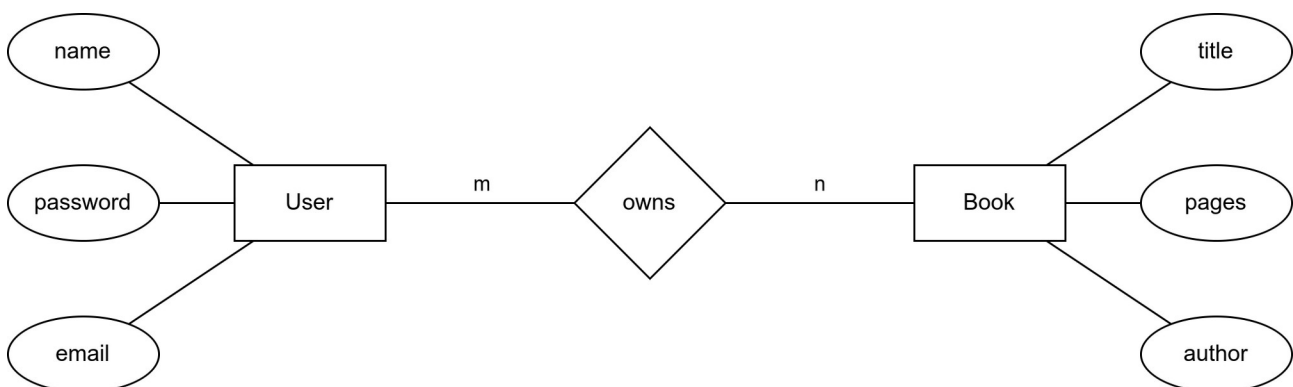


Illustration 6: Beispiel eines ER-Modells

2 Logisches Datenbankdesign

Das logische Datenbankdesign befasst sich mit der tatsächlichen Implementierung der Datenbank. Hier werden also deutlich mehr technische Faktoren berücksichtigt, als dies beim konzeptionellen Design der Fall war.

Beim logischen Datenbankdesign werden die im konzeptionellen Datenbankdesign identifizierten Entitäten und Beziehungen in Tabellen umgesetzt. Dabei werden auch die Attribute der Entitäten festgelegt und ihre Datentypen spezifiziert. Das logische Datenbankdesign legt zudem fest, wie die Tabellen miteinander verbunden werden, um die Beziehungen zwischen den Daten darzustellen.

Ein wichtiger Teil des logischen Datenbankdesigns ist die Festlegung von Indizes und Schlüsseln. Indizes dienen dazu, die Leistung der Datenbank bei der Suche nach bestimmten Daten zu verbessern. Schlüssel sind spezielle Indizes, die zur Identifizierung von Datensätzen in einer Tabelle verwendet werden. Es gibt verschiedene Arten von Schlüsseln, wie den Primärschlüssel, der eindeutig für jeden Datensatz in einer Tabelle ist, und den Fremdschlüssel, der auf einen Primärschlüssel in einer anderen Tabelle verweist.

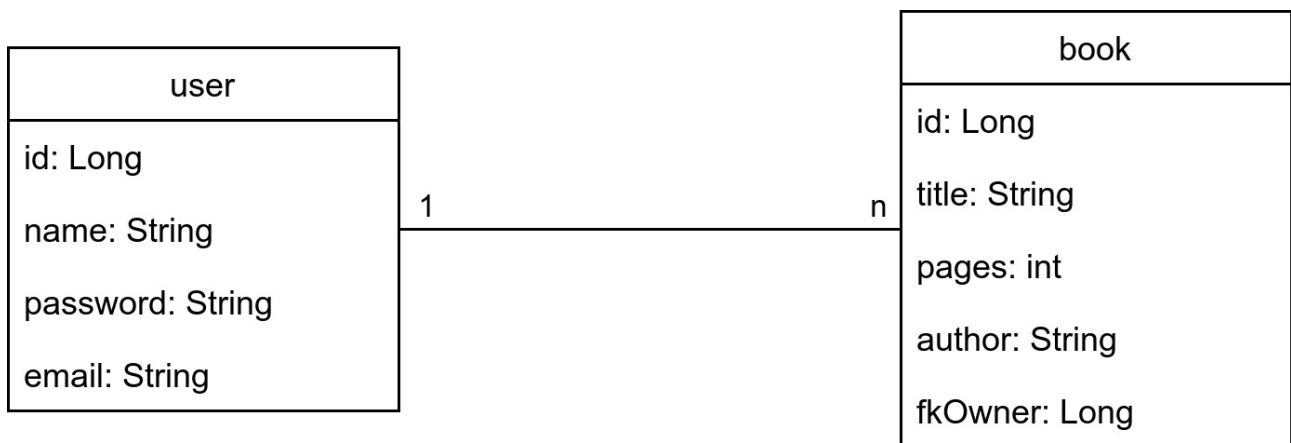


Illustration 7: Beispiel eines simplen logischen Datenbankdesigns

3 Physisches Datenbankdesign

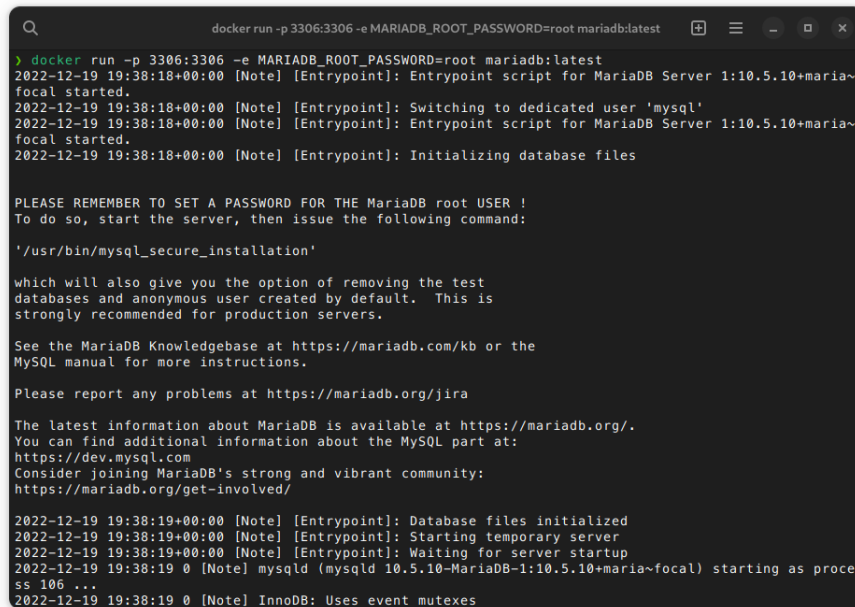
Beim letzten Schritt werden die Erkenntnisse aus dem logischen Design auf ein tatsächliches Datenbanksystem angewandt. Da es verschiedene Datenbanksysteme gibt, muss dieser Schritt – je nachdem – für jedes neue System wiederholt werden.

Insbesondere wird hierbei nochmals Optimierung betrieben. Je nach System sind manche Ansätze schneller oder besser als andere. Festgelegte Datentypen werden in ihre entsprechenden Systemabhängigen Typen gewandelt und Schlüssel gesetzt.

Bei allen drei Schritten ist es von besonderer Wichtigkeit, die Erkenntnisse zu dokumentieren. So kann später jeder Entwickler die Entscheidungen besser nachvollziehen und verstehen.

Beispiel

Zu Demonstrationszwecken wurde via Docker ein MariaDB Server gestartet:



```
docker run -p 3306:3306 -e MARIADB_ROOT_PASSWORD=root mariadb:latest

> docker run -p 3306:3306 -e MARIADB_ROOT_PASSWORD=root mariadb:latest
2022-12-19 19:38:18+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.5.10+mariadb-focal started.
2022-12-19 19:38:18+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2022-12-19 19:38:18+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.5.10+mariadb-focal started.
2022-12-19 19:38:18+00:00 [Note] [Entrypoint]: Initializing database files

PLEASE REMEMBER TO SET A PASSWORD FOR THE MariaDB root USER !
To do so, start the server, then issue the following command:

'/usr/bin/mysql_secure_installation'

which will also give you the option of removing the test
databases and anonymous user created by default. This is
strongly recommended for production servers.

See the MariaDB Knowledgebase at https://mariadb.com/kb or the
MySQL manual for more instructions.

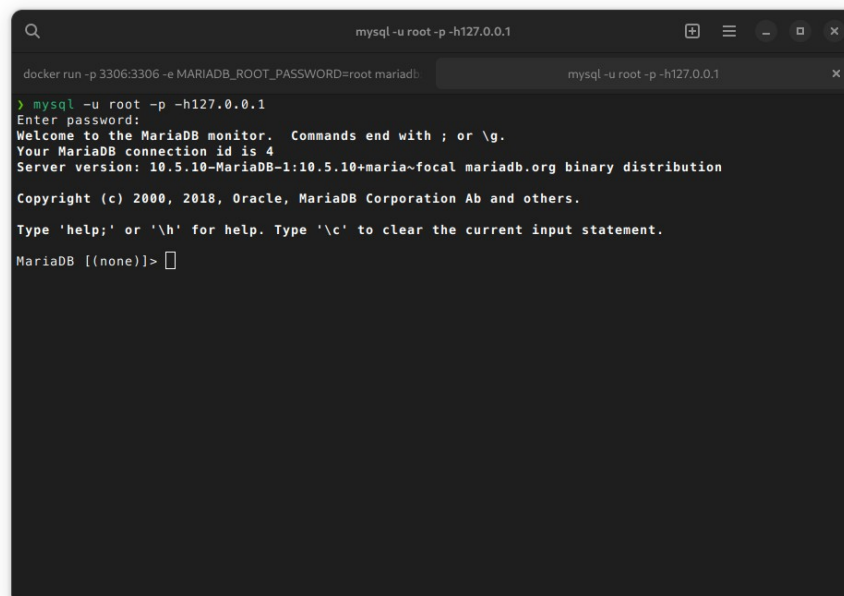
Please report any problems at https://mariadb.org/jira

The latest information about MariaDB is available at https://mariadb.org/.
You can find additional information about the MySQL part at:
https://dev.mysql.com
Consider joining MariaDB's strong and vibrant community:
https://mariadb.org/get-involved/

2022-12-19 19:38:19+00:00 [Note] [Entrypoint]: Database files initialized
2022-12-19 19:38:19+00:00 [Note] [Entrypoint]: Starting temporary server
2022-12-19 19:38:19+00:00 [Note] [Entrypoint]: Waiting for server startup
2022-12-19 19:38:19 0 [Note] mysqld (mysqld 10.5.10-MariaDB-1:10.5.10+mariadb-focal) starting as process 106 ...
2022-12-19 19:38:19 0 [Note] InnoDB: Uses event mutexes
```

Illustration 8: Ein MariaDB Server wird mittels Docker gestartet.

Mit dem MySQL CLI Client können wir uns verbinden:



```
mysql -u root -p -h127.0.0.1

docker run -p 3306:3306 -e MARIADB_ROOT_PASSWORD=root mariadb:latest mysql -u root -p -h127.0.0.1

> mysql -u root -p -h127.0.0.1
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.5.10-MariaDB-1:10.5.10+mariadb-focal mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Illustration 9: Mittels MySQL Client kann man sich auf den Server verbinden.

Mit **CREATE TABLE** können wir unsere Tabellen erstellen.

```
MariaDB [demo]> CREATE TABLE book (id SERIAL, title VARCHAR(255), pages INT, author VARCHAR(255), fk  
Owner BIGINT unsigned REFERENCES user (id));  
Query OK, 0 rows affected (0.010 sec)  
  
MariaDB [demo]> SHOW TABLES;  
+-----+  
| Tables_in_demo |  
+-----+  
| book           |  
| user           |  
+-----+  
2 rows in set (0.000 sec)  
  
MariaDB [demo]> 
```

Illustration 10: Erstellung der book-Tabelle und Anzeige der Übersicht.

Das logische Datenmodell wurde hiermit für MariaDB implementiert.

2 Unterteilung von SQL-Code

SQL-Code kann in verschiedene Teilbereiche unterteilt werden: DDL, DQL, DML, DCL und TCL.

1 Data Definition Language DDL

In der DDL wird die Struktur der Datenbank beschrieben. Dies umfasst etwa Tabellen, Views, Procedures etc. Zur DDL gehören Befehle wie **CREATE**, **DROP**, **ALTER** und **TRUNCATE**.

2 Data Query Language DQL

Zur DQL gehört der **SELECT** Befehl. Er wird genutzt, um Daten aus der Datenbank zu lesen. Da die Syntax des **SELECT** sehr umfangreich ist, hat er gleich seinen eigenen Teilbereich erhalten. Es können Tabellen mit **JOIN** zusammengefügt werden und vieles mehr.

3 Data Manipulation Language DML

In der DML werden Daten manipuliert, sprich, bearbeitet. Wenn neue Daten hinzukommen, geändert oder gelöscht werden, dann wird die DML genutzt. Zur DML gehören Befehle wie **INSERT**, **UPDATE** und **DELETE**.

4 Data Control Language DCL

Die DCL ist insbesondere für die Rechteverwaltung zuständig. Mit den Befehlen **GRANT** und **REVOKE** können die Rechte der Datenbankuser verwaltet werden. Beispielsweise muss nicht jede Applikation auf eine Datenbank schreiben können, dann macht es Sinn, dem User das Schreiben auch gleich zu verwehren.

5 Transaction Control Language TCL

Wie der Name schon vermuten lässt, ist die TCL für Transaktionen zuständig.

Transaktionen können mit **START TRANSACTION** begonnen werden und mit **COMMIT** geschrieben bzw. mit **ROLLBACK** verworfen werden. Transaktionen sind insbesondere bei Mehrbenutzersystemen ein wichtiger Bestandteil. Dank Transaktionen kann das ACID-Prinzip sichergestellt werden.

```
MariaDB [demo]> START TRANSACTION;
Query OK, 0 rows affected (0.000 sec)

MariaDB [demo]> INSERT INTO user (name, password, email) VALUES ('Demo', 'Demo', 'demo@demo.ch');
Query OK, 1 row affected (0.001 sec)

MariaDB [demo]> SELECT * FROM user;
+-----+-----+-----+-----+
| id | name | password | email |
+-----+-----+-----+-----+
| 1 | Demo | Demo | demo@demo.ch |
+-----+-----+-----+-----+
1 row in set (0.000 sec)

MariaDB [demo]> ROLLBACK;
Query OK, 0 rows affected (0.004 sec)

MariaDB [demo]> SELECT * FROM user;
Empty set (0.000 sec)

MariaDB [demo]> 
```

Illustration 11: Transaktionen können Änderungen wieder rückgängig machen.

4 Changemanagement

Wenn viele Personen im Team an einem Projekt arbeiten, kann dies schnell im Chaos enden. Abhilfe dafür schaffen verschiedene Systeme.

1 Datenbank

Das Datenmodell entwickelt sich stetig weiter, auch zwischen einzelnen Releases. Um eine Migration des Produktsystems reibungslos durchführen zu können, müssen so manche Dinge beachtet werden. Es gibt verschiedene Software, die hier aber Abhilfe schaffen kann.

In der Java-Welt gibt es etwa Flyway, welches Migrationen automatisch vornehmen und auch in Kombination mit CI/CD eingerichtet werden kann. Das Datenmodell wird hierbei ebenfalls stets versioniert, ähnlich wie Git. Jedoch sind die Änderungen hier aufbauend, da bestehende Daten in einer Datenbank erhalten bleiben müssen.

Ähnliche Lösungen gibt es auch für andere Technologien.

2 Code

Der Platzhirsch unter den Versionsverwaltungstools ist Git. Mit seiner dezentralen Struktur ist es auch besonders für Open-Source-Projekte gut geeignet. An einem Projekt kann so auch vollständig offline gearbeitet werden.

Es gibt verschiedene Methoden, wie ein Team seinen Git-Workflow definieren kann. Beliebte ist die «Git Flow Strategy». Dabei gibt es fünf verschiedene Arten von Branches:

- Main
- Develop
- Feature
- Release
- Hotfix

1 Main

Auf dem Main-Branch liegt der aktuellste Release. Wenn ein neuer Release bereit ist, wird der aktuelle Stand aus dem entsprechenden Release-Branch in den Main-Branch gemerged.

2 Develop

Dieser Branch enthält den aktuellen Entwicklungsstand. Sobald ein neues Feature oder ein neuer Fix fertig ist, wird er hierhin gemerged, sodass die anderen Entwickler die Änderungen ebenfalls erhalten.

3 Feature

Für jedes Feature oder jeden Fix wird ein Feature-Branch, ausgehend von Develop, erstellt. Der Entwickler kann dort ungestört arbeiten und später, wenn alles fertig ist, den Branch nach Develop mergen.

4 Release

Sobald ein neuer Release ansteht, wird aus Develop ein neuer Release-Branch abgeleitet. Die Software wird dann ausgiebig getestet. Neue Features beeinflussen diesen Branch dann nicht mehr, nur noch Hotfixes sind erlaubt.

5 Hotfix

Muss auf dem Release-Branch ein schwerwiegender Fehler behoben werden, wird ein Hotfix-Branch erstellt. Ähnlich wie beim Feature-Branch wird dieser in den Release-Branch gemerged, wenn der Fix fertig ist.

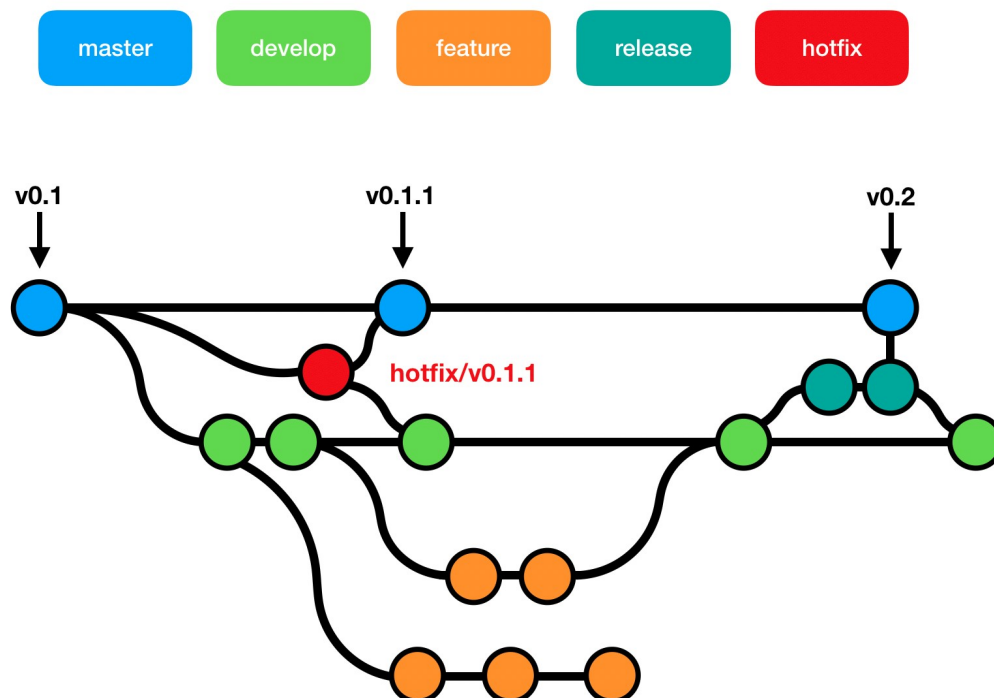


Illustration 12: Visualisierung der «Git Flow Strategy»
<https://www.codewall.co.uk/a-git-flow-explainer-how-to-tutorial/>

5 Testing

Heute hat man als Entwickler nahezu unlimitierte Möglichkeiten, den eigenen Code automatisiert auf Herz und Nieren zu testen. Automatisierte Tests können die Produktivität um einiges steigern.

1 Während der Entwicklung

Jede Sprache und jede Technologie hat seine eigenen Tools zum Testen. In der Java-Welt ist JUnit für Unit-Tests verbreitet, RestAssured kann für E2E-Testing genutzt werden, usw.

1 Unit-Tests

Unit-Tests testen einzelne, kleine Komponenten in einer isolierten Umgebung. So kann etwa jede einzelne Funktion einer Klasse isoliert getestet werden.

2 Integration-Tests

Integration-Tests sind weniger isoliert als Unit-Tests. Hier hängen schon mehr Klassen zusammen und es wird das Zusammenspiel getestet.

3 E2E-Tests

Bei Ende-zu-Ende-Tests wird das gesamte System von aussen getestet. Also so, wie es ein Nutzer sehen würde. Die Applikation agiert dann als Black Box, man achtet also nur auf den Input und Output.

4 User-Acceptance-Tests

UATs werden meist manuell durchgeführt. Dabei testen normale Nutzer die Applikation, bevor sie released wird. Dabei können etwa Unklarheiten bei der Bedienung aufgedeckt werden, an die ein Entwickler nicht gedacht hätte

2 Im produktiven Betrieb

Auch wenn ein Projekt eine ausgezeichnete Testabdeckung hat, können sich Fehler in die Produktivumgebung schleichen. Daher ist es wichtig, auch diese zu überwachen. Fehlermeldungen können gesammelt und ausgewertet werden. Mit Elasticsearch, Kibana und Logstash lässt sich unter anderem ein ganzer Stack zur Log-Auswertung zusammenbauen.

3 Datenbank

Die Datenbank darf ebenfalls keine Fehler enthalten oder langsam sein. Die Datenbank kann man daher Performance-Tests unterziehen. Dabei wird untersucht, wie schnell die

Datenbank unter verschiedenen Bedingungen Abfragen durchführen kann. Damit können etwa schwere Indizes zum Vorschein kommen, welche man dann, falls sie nicht nötig sind, entfernen kann.