

Projekt Fahrradverleih

M153 Datenbankmodelle entwickeln

Winkler Olivier, INF6J

16.05.2020, Version 1.0



Inhaltsverzeichnis

Projekt Fahrradverleih	1
Inhaltsverzeichnis	2
Abbildungsverzeichnis	2
1. Management-Zusammenfassung	3
2. Projektbeschreibung	4
2.1. Anforderungen	4
2.2. Fragen (aus Interview mit Kunde)	4
2.3. Lösungsansätze	5
2.4. Abgrenzungen	5
3. Datenbankmodelle	6
3.1. Konzeptionelles Datenbankmodell	6
3.2. Logisches Datenbankmodell	7
3.3. Physisches Datenbankmodell	8
3.3.1. Physisches Datenbankmodell als Diagramm	9
3.3.2. Berechtigungen in der Datenbank	9
4. Test	10
4.1. Testdaten	10
4.2. Loadtests	11
4.3. Performancetests	12
5. Erweiterungsmöglichkeiten	13
6. Anhang / Referenzen	14

Abbildungsverzeichnis

Abbildung 1	6
Abbildung 2	7
Abbildung 3	8
Abbildung 4	9
Abbildung 5	9
Abbildung 6	9
Abbildung 7	10
Abbildung 8	10
Abbildung 9	10
Abbildung 10	11
Abbildung 11	11
Abbildung 12	11
Abbildung 13	11
Abbildung 14	12
Abbildung 15	12
Abbildung 16	12
Abbildung 17	12

1. Management-Zusammenfassung

Ich habe die Aufgabe bekommen bei einer Fahrradvermietung «*Veloeggä AG*» den Vorgang zur Vermietung von Fahrrädern zu verbessern & zu vereinfachen umso eine bessere Übersicht für die Buchhaltung zu schaffen. Zurzeit wird die Buchhaltung mit einem Excel-Sheet gehandhabt, welches aber bei einigen Situationen an seine Grenzen stösst und so den Betrieb hindert. Die Lösung soll als Form eines Datenbanksystems realisiert werden. Als erstes wurde ein Gespräch mit dem Geschäftsführer von «*Veloeggä AG*» getätigt, um die Anforderungen an das neue Produkt zu klären. Bei diesem Gespräch haben sich wichtige Fragen herauskristallisiert, welche einen wichtigen Bestandteil in der Datenbankstrukturierung bieten. Bei der Umsetzung soll es lediglich um eine Datenbank gehen und nicht über ein zusätzliches Userinterface. Dies könnte jedoch ein zukünftiges Feature sein, welches zu einem späteren Zeitpunkt implementiert wird. Mit einer groben Vorstellung von der ganzen Sache habe ich erste grafische Konzepte entwickelt, welche später im Dokument nachgeschlagen werden können, erstellt. Mit diesen habe ich dann die Datenbank realisiert und diverse Systemtests mitsamt Performancetests durchgeführt. Zum Abschluss habe ich noch die Erweiterbarkeit für dieses Projekt erwähnt.

2. Projektbeschreibung

2.1. Anforderungen

- Alle Fahrräder sollen erfasst sein
- Alle Mieter sollen erfasst werden können
- Die Struktur soll möglichst einfach gehalten sein.
- Die Lösung soll erweiterbar sein (z.B. bei zweitem Standort)
- Die Vermietungen sollen eingetragen werden können

2.2. Fragen (aus Interview mit Kunde)

1. Wie sieht die aktuelle Situation aus?

Im Moment arbeiten wir mit einem Excel-Sheet. Da dies aber manchmal an seine Grenzen geht, suchen wir nach einer besseren Lösung.

2. Was ist genau mit "Alle Fahrräder sollen erfasst sein" gemeint?

Hier sind alle Modelle gemeint.

3. Was braucht es für das Fahrrad noch?

Da wir als Fahrradverleih auch für Qualität sprechen wollen, muss der Zustand jedes einzelnen Velos erfasst sein. Für uns ist es unmöglich ein qualitativ schlechtes Fahrrad zu vermieten.

4. Was betrifft alle Modelle?

Alle Typen (z.B.: Mountainbike) Alle Marken (z.B.: Scott)

5. Was soll bei dem Mieter erfasst werden?

Vorname Name Handynummer Adresse

6. Welche Daten braucht eine Vermietung?

Damit wir Zeit für die Buchhaltung sparen können müssen folgende Daten ersichtlich sein:
Wer hat das Velo gemietet Welche Fahrräder sind vermietet worden Preis der Vermietung Mietdatum

7. Nach welchen Daten wird am meisten gesucht?

Da Unsere Mitarbeiter nicht alle Daten benötigen, sind dies, nach denen am Meisten gesucht wird: Kunden Nummer Fahrrad Nummer Verleih Preis Verleihdatum

8. Haben die einzelnen Benutzer verschiedene Rollen?

Ja, wir unterscheiden folgende Benutzer (-Rollen); Chef - Das Sekretariat und Ich dürfen alles verwalten. Sekretärin Mitarbeiter - Die Mitarbeiter nehmen jedoch nur Aufträge entgegen und bereiten das Material vor.

2.3. Lösungsansätze

Zusammen mit dem Kunden habe ich mich entschieden das Projekt mit PostgreSQL umzusetzen. Da die Erweiterbarkeit bei diesem Projekt eine grosse Rolle spielt, kann dies mit PostgreSQL in Zukunft einfach & schnell umgesetzt werden ohne grosse Hürden meistern zu müssen. Die Anforderungen an das System sind nicht unbedingt sehr hoch um spezifisch auf eine Technologie zu setzen. Falls jedoch die Umsetzung mit PostgreSQL nicht möglich wäre oder es mit dieser Technologie zu Problemen führen könnte, würde eine andere zum Einsatz kommen (z.B. MySQL).

2.4. Abgrenzungen

Die geforderte Lösung des Kunden basierte zum Zeitpunkt der Auftragserteilung auf einer reinen Datenbankapplikation ohne zusätzlichem Userinterface etc. Zusätzlich bietet die hier vorgestellte Lösung nur die Sicherung von Daten in einer Datenbank umso einen effizienteren Überblick über die Fahrradvermietung zu bekommen. In Zukunft könnte die Datenbank in die Webseite integriert werden umso eventuell direkt Käufe und Vermietungen tätigen zu können.

3. Datenbankmodelle

3.1. Konzeptionelles Datenbankmodell

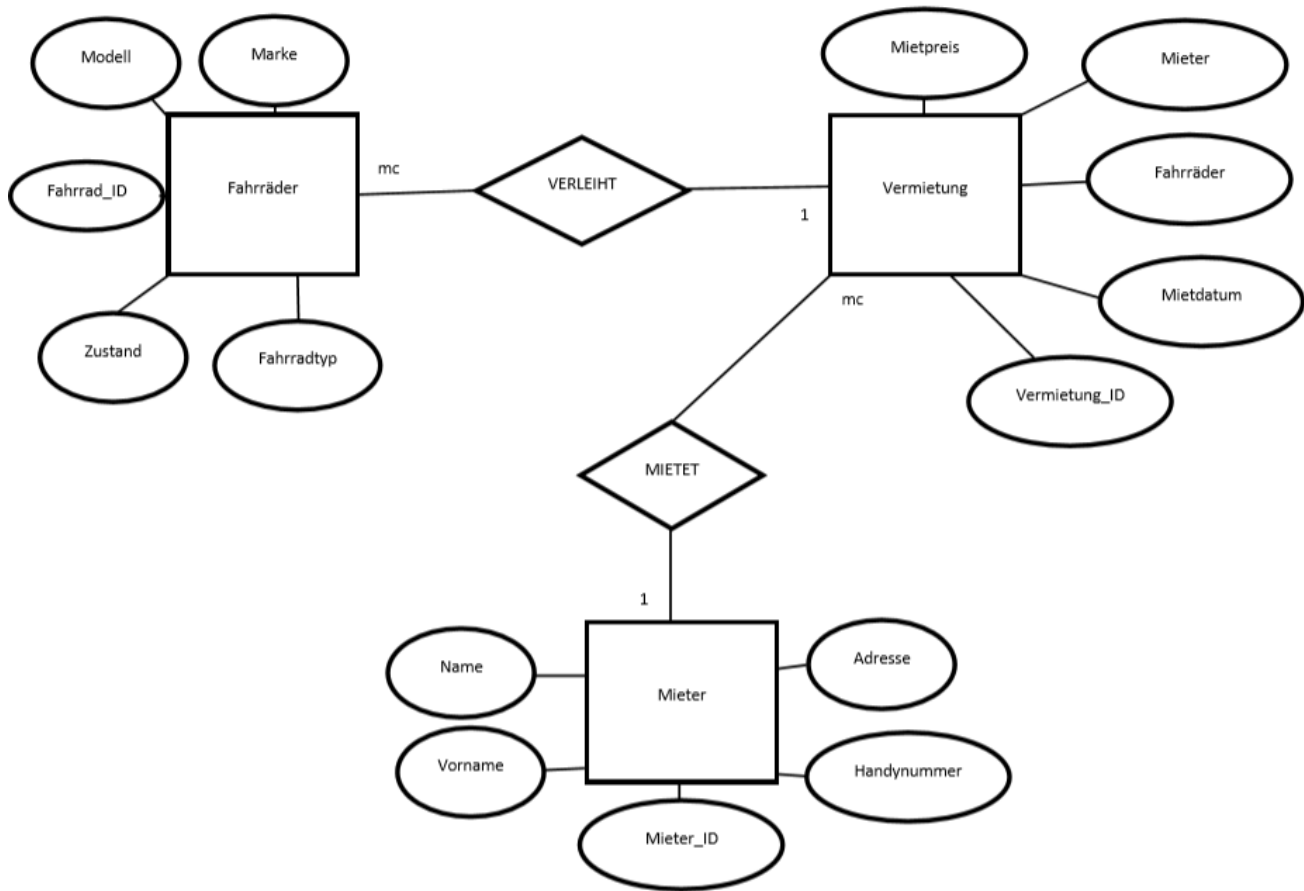


Abbildung 1

Dies ist ein konzeptionelles Datenbankmodell. Ich habe dies als allererstes erstellt, um einen groben Überblick über die Struktur der späteren Datenbank zu bekommen. Zu sehen sind die drei Tabellen mit den dazugehörigen Attributen. Zwischen den Tabellen sind die Beziehungen mitsamt Notation abgebildet. Mein Konzept ist auf einer Haupttabelle aufgebaut. Die Vermietung ist die Haupttabelle und bekommt Daten für die Vermietung aus den Untertabellen «*Mieter*» und «*Fahrräder*».

Die Tabelle «*Mieter*» besteht aus einer Identifikationsnummer für die Zusammenführung zur Tabelle «*Vermietung*». Ein Mieter hat einen Vor- und Nachnamen. Dazu kommt seine Wohnadresse und Handynummer. Grundsätzlich dient diese Tabelle dazu, die Person zu identifizieren und so einer Vermietung zuweisen zu können.

Die Tabelle «*Fahrräder*» enthält alle wichtigen Daten, die über ein einzelnes Fahrrad benötigt werden. Einerseits ist wieder ein Identifikator für die Identifikation in der Vermietung vorhanden. Ein Fahrrad gehört einer bestimmten Fahrradmarke an und repräsentiert ein spezifisches Fahrradtyp (z.B. Mountainbike oder Citybike). Zusätzlich kann ein Fahrrad ein bestimmtes Modell sein (z.B. MTB-Fully). Zudem ist bei jedem Fahrrad der Zustand hinterlegt, um so eine hohe Qualität dem Kunden zu bringen.

In der Vermietung kommen diese Daten zusammen inklusive dem Mietpreis & Mietdatum.

3.2. Logisches Datenbankmodell

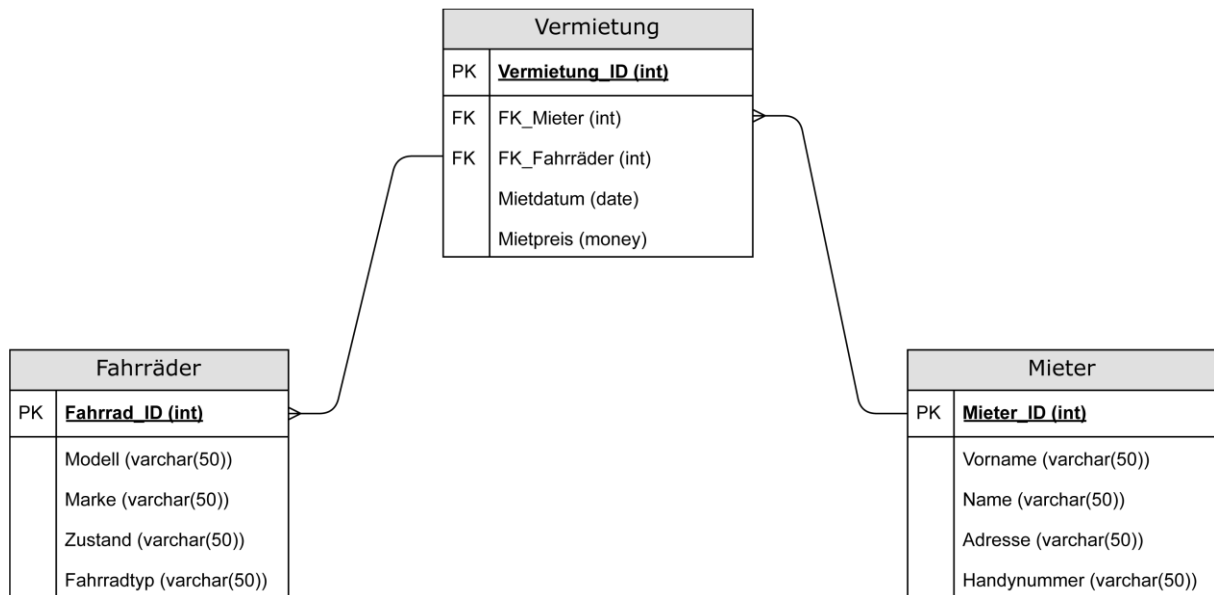


Abbildung 2

Im logischen Datenmodell wird das konzeptionelle weiterentwickelt. Es werden bei den einzelnen Attributen die Datentypen festgelegt. Hier sind auch gut die Beziehungen und die dazugehörigen Schlüssel zu sehen. Für die Identifikatoren wird der Datentyp **int** in diesem System verwendet. Für reinen Text wird **varchar** mit 50 Zeichen verwendet. Die 50 Zeichen reichen für die einzelnen Attribute aus, da es meistens nur einzelne Wörter sind. Für ein Datum wurde der Typ **date** verwendet, um das Mietdatum zu hinterlegen. Beim Preis wurde der Datentyp **money** verwendet, um einen Preis darzustellen.

3.3. Physisches Datenbankmodell

```

create table if not exists fahrräder
(
    fahrrad_id serial not null
        constraint fahrräder_pk
            primary key,
    modell varchar(50),
    marke varchar(50),
    zustand varchar(50),
    fahrradtyp varchar(50)
);

create unique index if not exists fahrräder_fahrrad_id_uindex on fahrräder (fahrrad_id);

create table if not exists mieter
(
    mieter_id serial not null
        constraint mieter_pk
            primary key,
    vorname varchar(50),
    nachname varchar(50),
    adresse varchar(50),
    handynummer varchar(50)
);

create unique index if not exists mieter_mieter_id_uindex on mieter (mieter_id);

create table if not exists vermietung
(
    vermietung_id serial not null
        constraint vermietung_pk
            primary key
        constraint fk_fahrräder
            references fahrräder
                on update cascade on delete cascade
        constraint fk_mieter
            references mieter
                on update cascade on delete cascade,
    mietdatum date,
    mietpreis money
);

create unique index if not exists vermietung_vermietung_id_uindex on vermietung (vermietung_id);

```

Abbildung 3

Mit dem physischen Datenmodell wird die Datenbank mit den dazugehörigen Tabellen erstellt. Ein solches Script kann von Hand oder auch via Datagrip oder anderen Anwendungen generiert werden. Zuerst wird jede Tabelle erstellt und überprüft ob nicht schon eine Tabelle mit dem gleichen Namen existiert. In dieser Sequenz werden alle Attribute einer Tabelle initialisiert mitsamt ihrem Datentyp. Gut zu sehen sind die Fremdschlüssen in der Tabelle «**Vermietung**». Mit dem Kennwort «**references**» wird ein Attribut mit einem Attribut aus einer anderen Tabelle verknüpft. Zusätzlich wird das Verhalten angegeben was mit der Beziehung passieren soll, wenn sich der Wert in einer der Tabellen ändert. Hier wurde «**cascade**» verwendet welches den Eintrag in beiden Tabellen anpasst beziehungsweise löscht. Bei einem «**PrimaryKey**» wird festgelegt, dass dieser **unique**, **not null** und **automatisch hochzählend** ist.

3.3.1. Physisches Datenbankmodell als Diagramm

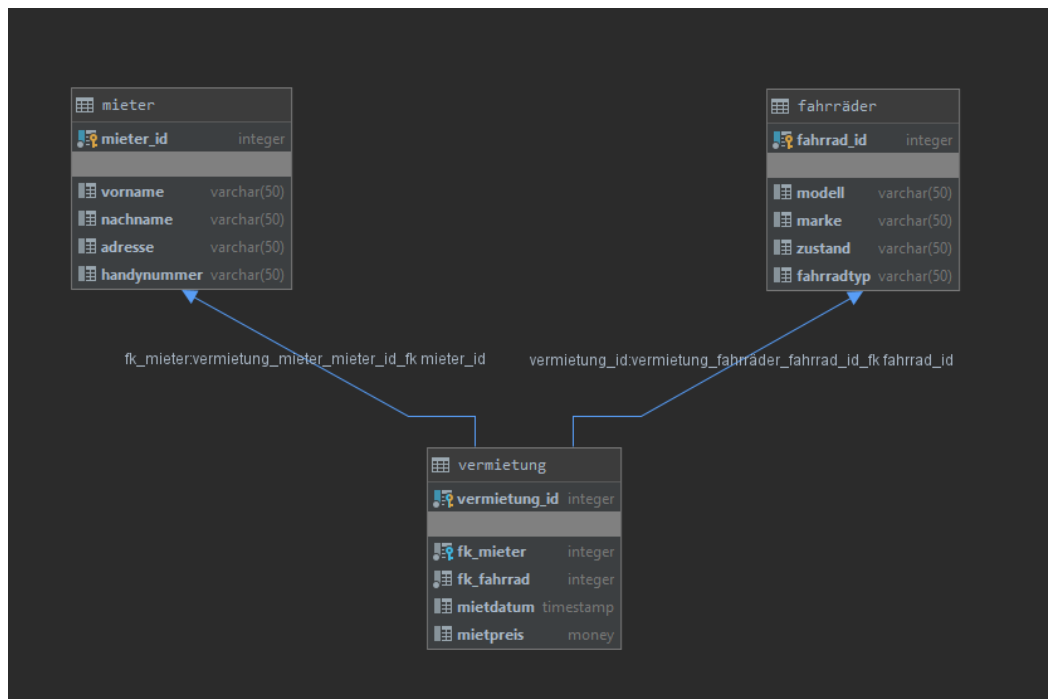


Abbildung 4

3.3.2 Berechtigungen in der Datenbank

Der Kunde hat sich gewünscht, dass die Datenbank mit Benutzern mit diversen Berechtigungen bestattet ist. Die Anforderungen an diese Funktion waren folgende:

- **Chef / Sekretärin** - Das Sekretariat und Ich dürfen alles verwalten.
- **Mitarbeiter** - Die Mitarbeiter nehmen jedoch nur Aufträge entgegen und bereiten das Material vor.

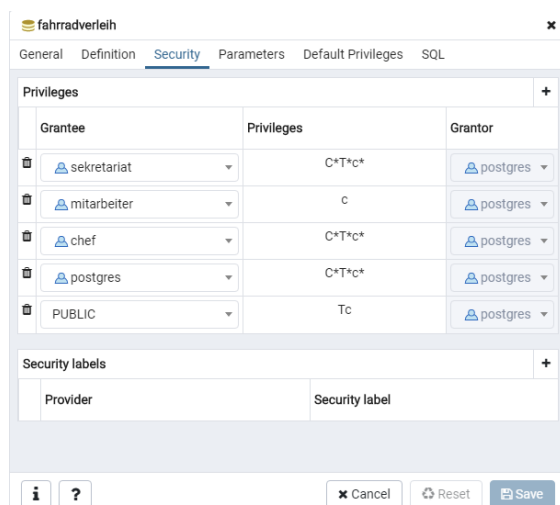


Abbildung 6

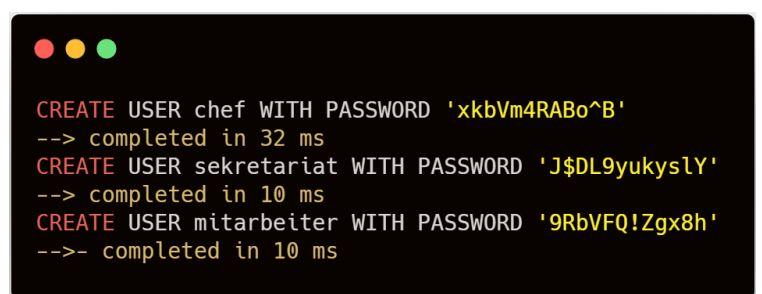
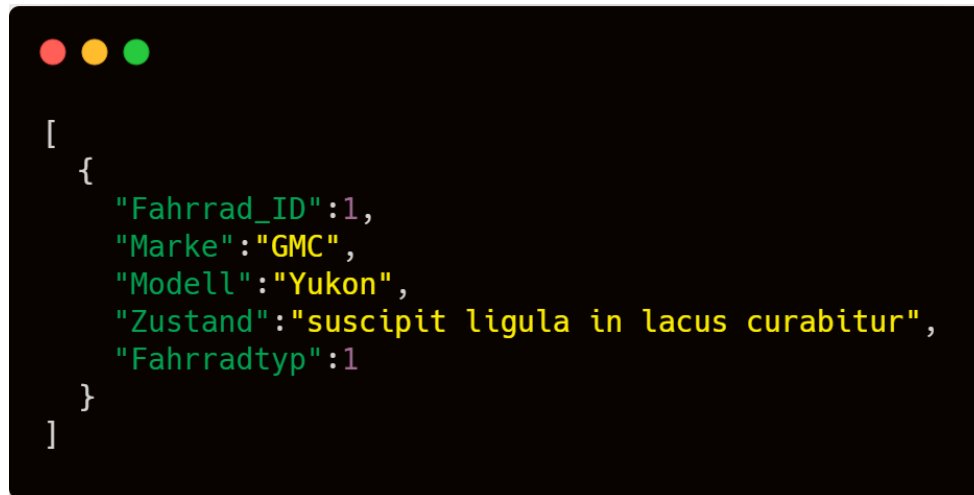


Abbildung 5

4. Test

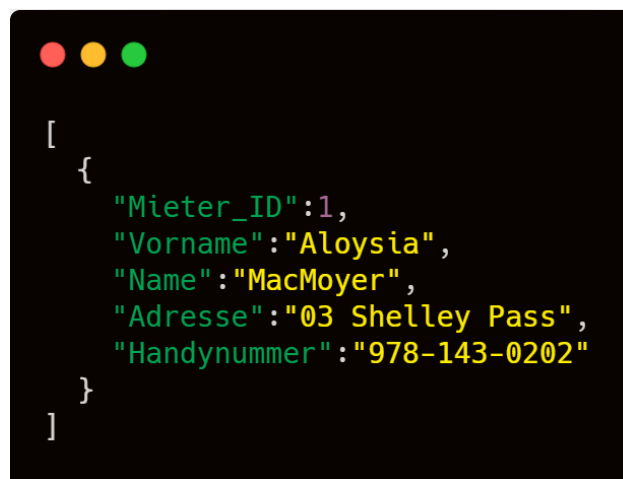
4.1. Testdaten

Je erster Eintrag von einer Tabelle.



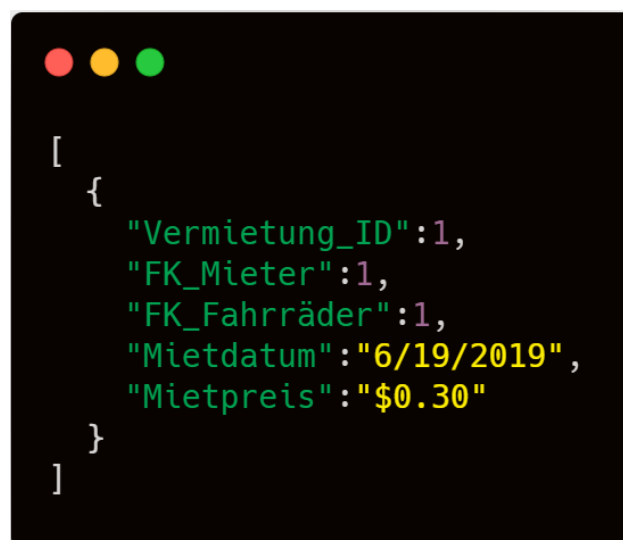
```
[
  {
    "Fahrrad_ID":1,
    "Marke":"GMC",
    "Modell":"Yukon",
    "Zustand":"suscipit ligula in lacus curabitur",
    "Fahrradtyp":1
  }
]
```

Abbildung 7



```
[
  {
    "Mieter_ID":1,
    "Vorname":"Aloysia",
    "Name":"MacMoyer",
    "Adresse":"03 Shelley Pass",
    "Handynummer":"978-143-0202"
  }
]
```

Abbildung 8



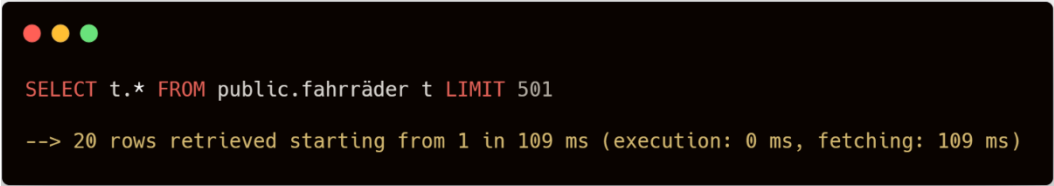
```
[
  {
    "Vermietung_ID":1,
    "FK_Mieter":1,
    "FK_Fahrräder":1,
    "Mietdatum":"6/19/2019",
    "Mietpreis":"$0.30"
  }
]
```

Abbildung 9

4.2. Loadtests

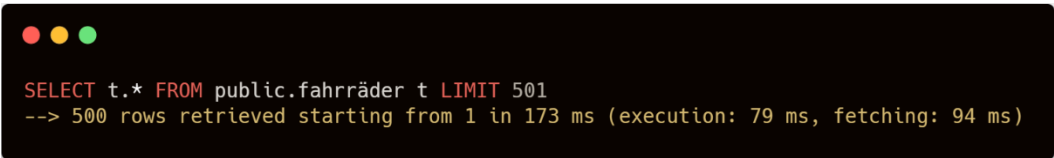
In diesem Abschnitt habe ich die Ladezeiten der verschiedenen Tabellen in der Datenbank verglichen. Der Loadtest bestand daraus, wie lange die Ladezeit für die Darstellung aller beinhaltenden Daten dauert. Bei dem ersten Loadtest mit der Tabelle «Fahrräder» wurden mit 20 Reihen die Daten zwischen 50ms und 115ms ausgelesen. Bei den anderen zwei Tabellen habe ich ähnliches festgestellt. Die Ladezeiten variieren zum Teil zwischen etwa 50ms und 120ms. Danach habe ich die einzelnen Tabellen mit 1000 Reihen Daten gefüllt und erzielte ungefähr die gleichen Ladezeiten.

Man kann also daraus schliessen, dass die Datenbank mit dieser Anzahl Daten sehr effizient läuft. Ich denke es braucht noch viel mehr Datensätze in den Tabellen, um einen wirklichen Unterschied feststellen zu können.



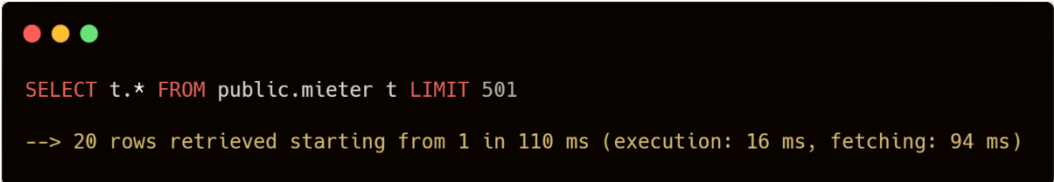
```
SELECT t.* FROM public.fahrräder t LIMIT 501
--> 20 rows retrieved starting from 1 in 109 ms (execution: 0 ms, fetching: 109 ms)
```

Abbildung 10



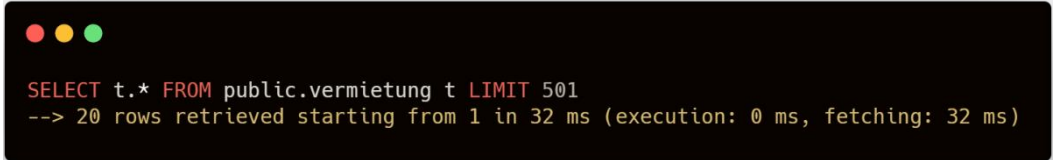
```
SELECT t.* FROM public.fahrräder t LIMIT 501
--> 500 rows retrieved starting from 1 in 173 ms (execution: 79 ms, fetching: 94 ms)
```

Abbildung 11



```
SELECT t.* FROM public.mieter t LIMIT 501
--> 20 rows retrieved starting from 1 in 110 ms (execution: 16 ms, fetching: 94 ms)
```

Abbildung 12



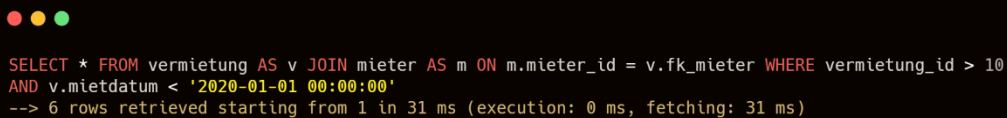
```
SELECT t.* FROM public.vermietung t LIMIT 501
--> 20 rows retrieved starting from 1 in 32 ms (execution: 0 ms, fetching: 32 ms)
```

Abbildung 13

4.3. Performancetests

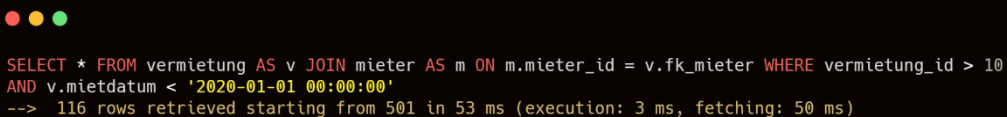
Bei den Performancetests habe ich die Leistung der Datenbank bei verschiedenen Operationen getestet. Dies habe ich zuerst wieder mit 20 Reihen und dann später mit 1000 Reihen Daten durchgeführt. Auch hier bin ich zu einem ähnlichen Resultat gekommen. Ob ich jetzt nur 20 Reihen Daten oder 1000 Reihen ausgabe, erzielt schlussendlich etwa die gleiche Dauer.

Diese Tests zeigen jedoch wie schnell und performant eine solche Datenbank die Daten im tausender Bereich verarbeitet. Dies ist für die Erweiterung und Vergrößerung des Systems ein Vorteil.



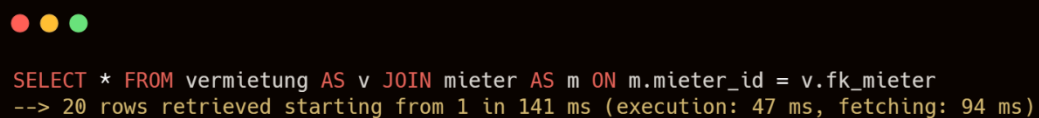
```
SELECT * FROM vermietung AS v JOIN mieter AS m ON m.mieter_id = v.fk_mieter WHERE vermietung_id > 10
AND v.mietdatum < '2020-01-01 00:00:00'
--> 6 rows retrieved starting from 1 in 31 ms (execution: 0 ms, fetching: 31 ms)
```

Abbildung 14



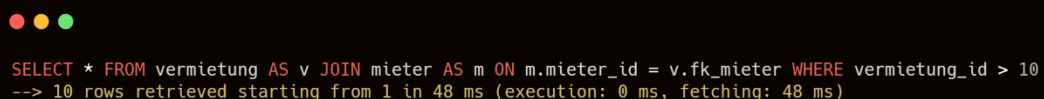
```
SELECT * FROM vermietung AS v JOIN mieter AS m ON m.mieter_id = v.fk_mieter WHERE vermietung_id > 10
AND v.mietdatum < '2020-01-01 00:00:00'
--> 116 rows retrieved starting from 501 in 53 ms (execution: 3 ms, fetching: 50 ms)
```

Abbildung 15



```
SELECT * FROM vermietung AS v JOIN mieter AS m ON m.mieter_id = v.fk_mieter
--> 20 rows retrieved starting from 1 in 141 ms (execution: 47 ms, fetching: 94 ms)
```

Abbildung 16



```
SELECT * FROM vermietung AS v JOIN mieter AS m ON m.mieter_id = v.fk_mieter WHERE vermietung_id > 10
--> 10 rows retrieved starting from 1 in 48 ms (execution: 0 ms, fetching: 48 ms)
```

Abbildung 17

5. Erweiterungsmöglichkeiten

Schon bei dem ersten Gespräch wurde die Erweiterbarkeit erwähnt. Somit ist das Projekt auch mit für die Erweiterbarkeit ausgelegt. Mit PostgreSQL ist dies sehr einfach und kann mit wenig Aufwand und kleinen Umständen durchgeführt werden. Die Datenbank wäre auch bereit für die Integration in die Webseite. Die Datenbank könnte so erweitert werden, dass ganze Zahlungen abgewickelt werden könnten und so könnte man den Shop mit den Vermietungen an einem Ort verwalten.

6. Anhang / Referenzen

- Modellerstellung <https://app.diagrams.net/>
- Datagrip <https://www.jetbrains.com/de-de/datagrip/>
- Codedarstellung <https://carbon.now.sh/>
- Testdaten <https://mockaroo.com/>
- PostgreSQL <https://www.postgresql.org/docs/>
- Stackoverflow <https://stackoverflow.com/>