

052400-1 VU Information Management and Systems Engineering (2025S)

Milestone 1

*(Please note: submission deadline: Tue 11.04.2025 13:00)*

Group 1

Student 1: Ateih, Hamza, 11930767

Student 2: Winter,Tobias, 12028429

(*Please note: sort by last name*)

[*DATE*], Vienna

**Contents**

[1](#_5mtac8uzvgfa) Milestone 1 2

[1.1](#_5mtac8uzvgfa) Team - Conceptual Modeling 2

[1.1.1](#_vq4jnjyk9y6b) Describe the Application Domain 2

[1.1.2](#_bxfedhiftw1k) Logical Design – ER Diagram in Chen Notation 2

[1.1.3](#_k80nyvhw2cbu) Relational Modeling – SQL CREATE Statements 2

[1.2](#_qevu2pqcyxrg) Individual - Student 1 3

[1.2.1](#_xqqk1t3t250s) Use Case Definition and Design 3

[1.2.2](#_tvi0y9xsdkpv) Analytics Report 3

[1.2.3](#_g7c9q78315mh) NoSQL Design 3

[1.3](#_skdl4qfixcog) Individual - Student 2 4

[1.3.1](#_kk3okoihq3ll) Use Case Definition and Design 4

[1.3.2](#_2ljpycs8nous) Analytics Report 4

[1.3.3](#_js5g0u8hdvig) NoSQL Design 4

# 

| **Requirements** | **Realization** |
| --- | --- |
| **Entities** |  |
| App |  |
| Benutzer |  |
| PrimeKunde | IS-A Benutzer |
| Gast | IS-A Benutzer |
| Restaurant |  |
| Gericht |  |
| Lieferant |  |
| Warenkorb | Strong entity |
| Orderitem | Weak entity |
| Relationships |  |
| laded ein | unary auf Benutzer |
| benutzt (n:1) | Benutzer - App |
| bewertung (n:m) | Benutzer - Restaurant |
| hat (n:1) | Restaurant - App |
| stellt ein(n:1) | Lieferant - App |
| kocht(n:1) | Gericht - Restaurant |
| bestellt (1:1) | Benutzer - Restaurant |
| bestellt (1:1) | Benutzer - Warenkorb |
| bestellt (1:1) | Warenkorb - Restaurant |
| hat (n:1) | OrderItem - Warenkorb  (identifying relationship) |
| Mitglied (n:1) | Primekunde - App |
| in (n:1) | OrderItem - Gericht |

# 

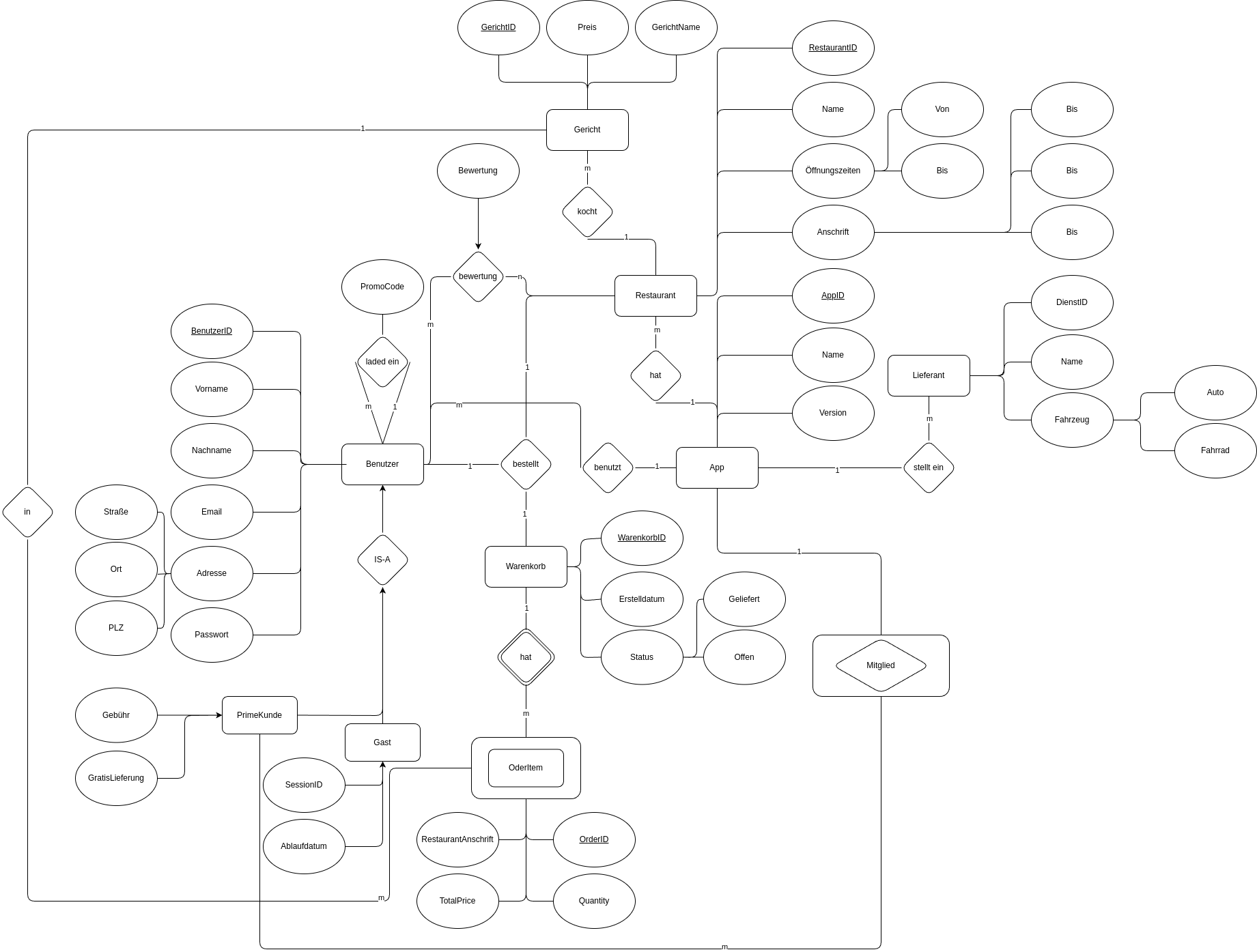
# Milestone 1

## Team - Conceptual Modeling

### Describe the Application Domain

Das Modell beschreibt einen Online-Lieferdienst, der sowohl für gewöhnliche als auch für Premiumkunden geeignet ist. Es gibt eine Liste der Gerichte, die in verschiedenen Restaurants angeboten werden, und es ordnet Lieferaufträge an relevante Lieferanten. Die spezifischen Beziehungen werden wie folgt beschrieben: Der Lieferservice verwaltet mehrere Benutzer, von denen einige auch Premium-Kunden sein können. Zahlreiche Benutzer können in der App verschiedene Gerichte von verschiedenen Restaurants bestellen. Die App beauftragt die Lieferanten, diese Bestellungen zuzustellen. Ein Benutzer kann mehrere andere Benutzer mit PromoCode werben, aber nur ein anderer Benutzer kann ihn werben. Warenkörbe sind eine zentrale Funktion des Lieferservices. Jeder Warenkorb wird durch eine eindeutige WarenkorbID identifiziert und ist einem Benutzer über die BenutzerID zugeordnet. Ein Warenkorb kann mehrere OrderItems enthalten, die die ausgewählten Gerichte repräsentieren. Die ResturantAnschrift im Warenkorb bezieht sich auf das Restaurant, von dem die Bestellung aufgegeben wird. Zusätzlich beinhaltet der Warenkorb Informationen zu LiefernKosten, was die Liefergebühren für die gesamte Bestellung darstellt. OrderItems repräsentieren die individuellen Artikel in einem Warenkorb. Jedes OrderItem hat eine eigene OrderItemID und verweist über WarenkorbID auf den zugehörigen Warenkorb. Es verweist auch auf das spezifische Gericht durch die GerichtID. Jedes OrderItem enthält Informationen wie Price (Preis des Gerichts), Quantity (Menge des Gerichts) und TotalPriceG (Gesamtpreis für das Gericht basierend auf der Menge).

### Logical Design – ER Diagram in Chen Notation



### Relational Modeling – SQL CREATE Statements

APP (AppID, Version Name)

Benutzer (BenutzerID, Vorname, Nachname, E-Mail, Adresse, Passwort, PromoCode, GratisLieferung)

Lieferant (DienstID, Name, FahrzeugTyp)

Restaurant (RestaurantID, Name, OffenVon, OffenBis, Anschrift)

Bestellung (BestellID, WarenkorbID, BenutzerID, RestraurantID)

Warenkorb (WarenkorbID)

OrderItem (OrderItemID, RestaurantAnschrift, TotalPreis, Stückzahl, GerichtID)

Gericht (GerichtID, Preis, GerichtName)

Mitglied (PrimeID)

Bewertung (BenutzerID, RestaurantID, Bewertung)

## Individual - Student 1

Student 1: Ateih, Hamza, 11930767

### Use Case Definition and Design

version 2 (IS-A)

#### Textual Description

##### *Title: Prime a*ktivieren und bestellen (*IS-A relationship)*

**Trigger**: Ein bereits bestehender Benutzer aktiviert die Prime Mitgliedschaft.

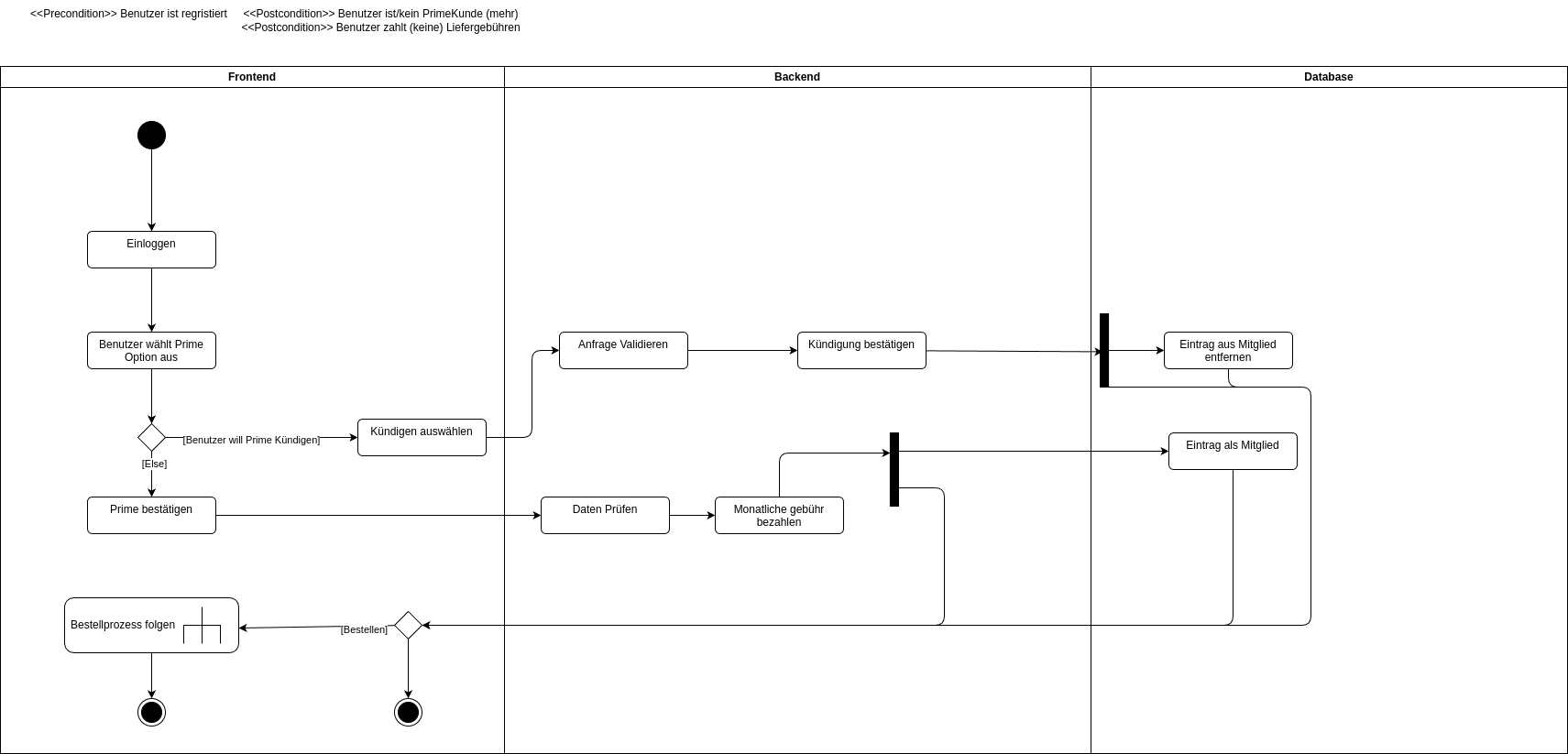
**Preconditions**:   
Benutzer ist registriert   
Benutzer ist kein PrimeKunde

**Main Flow**:   
1) Benutzer wählt die Prime Option aus  
2) Benutzer erstellt ein Abonnement  
3) Benutzer wird als Mitglied Markiert  
4) Benutzer folgt UseCase "Gericht zur Bestellung hinzufügen"

**Postconditions**:   
Benutzer ist Prime Mitglied und in Mitglied Table zu finden  
Benutzer hat keine Liefergebühr bezahlt

**Entities**: Benutzer, Mitglied, Warenkorb, OrderItem, Gericht, Warenkorb

#### Graphical Representation



### Analytics Report

#### Concept

Dieser Report zeigt eine Bestellung von einem Primekunden an.

Das dient als sanity check um zu prüfen ob der Kunde nach Abschluss  
des Abonnements nun Mitglied ist und keine Liefergebühren bezahlt werden.

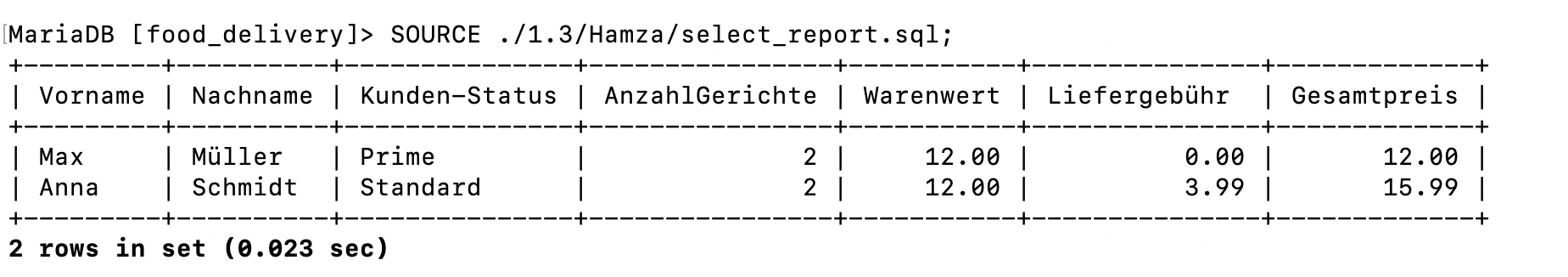
Gefiltert wird nach einem bestimmten User.

Entitäten: Mitglied, Benutzer, OrderItem, Gericht, Warenkorb

#### Proof of Concept

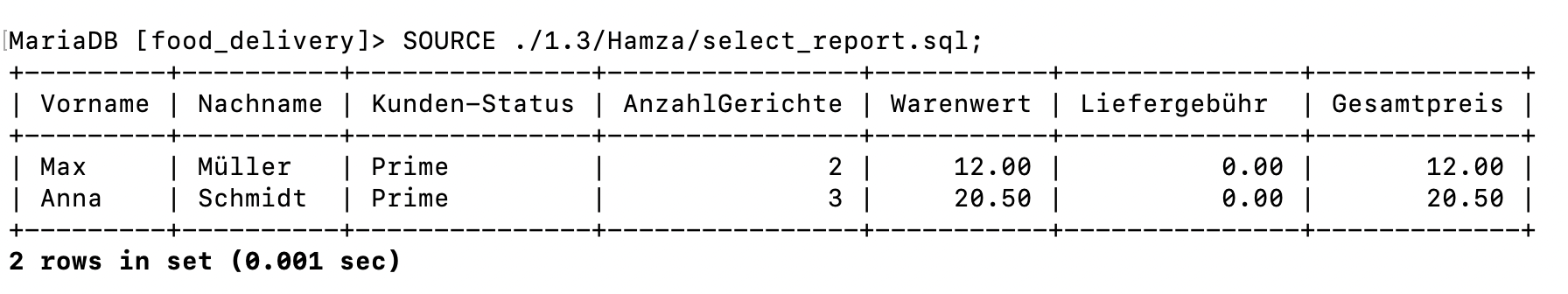
Was die Screenshots zeigen:

Die beiden Screenshots demonstrieren den Effekt des Use Cases "Prime aktivieren und Bestellen"



**Screenshot 1 (Vorher):**

* Zeigt Anna Schmidt als normalen Benutzer (Kunden-Status = Standard).
* Sie hat 2 Gerichte bestellt mit einem Gesamtpreis von 15,99€ davon “3.99€” Liefergebühr.
* Zeigt Max Müller als Prime Benutzer (Kunden-Status = Prime).
* Er hat die gleiche 2 Gerichte bestellt mit einem Gesamtpreis von 12,00€ und “0.00€” Liefergebühr.



**Screenshot 2 (Nachher):**

* Nach Ausführung des Use Cases ist Anna nun Prime-Kunde (Kunden-Status = Prime).
* Sie hat ein weiteres Gericht bestellt (insgesamt 3).
* Der Gesamtpreis ist auf 20,50€ gestiegen.
* Sie zahlt kein Liefergebühren mehr.

### NoSQL Design

Entities: Benutzer, PrimeKunde

#### Design Overview

Erklärung zu den Schemas:

1. Benutzer-Dokument:

- Enthält alle grundlegenden Benutzerinformationen

- Das Prime-Objekt enthält alle wichtigen Informationen zur Prime-Mitgliedschaft

- Durch die Einbettung können wir sofort erkennen, ob der Benutzer Prime-Vorteile hat

- Der Warenkorb mit eingebetteten OrderItems zeigt direkt, dass keine Liefergebühr anfällt

2. PrimeKundenReport-Dokument:

- Optimiert für die Abfrage "Prime aktivieren und bestellen"

- Zeigt nur die relevanten Daten für den Report

- Die Bestellungen werden mit 0.00 Liefergebühr angezeigt

- Vorberechnete Aggregate ermöglichen schnelle Abfragen

Dieses Design unterstützt den Use Case "Prime aktivieren und bestellen" optimal,

da der Prime-Status und die kostenlose Lieferung direkt erkennbar sind.

#### Expected Execution and Possible Changes

Meine Annahmen:

Der "Prime aktivieren und Bestellen" Use Case funktioniert gut mit

eingebettetem Prime-Status im Benutzer-Dokument. Das macht Front- und

Backend-Entwicklung einfacher, weil man direkt sieht, ob Gratislieferung

verfügbar ist oder nicht.

Für den Report wäre es ziemlich ineffizient, alle Benutzer zu durchsuchen.

Deshalb hab ich ein extra PrimeKundenReport-Dokument erstellt - das

enthält nur Prime-Mitglieder und spart uns viel Rechenzeit.

Frage:

Analyze how changes in the number of database operations of your USE CASE (e.g., more

frequent reads or writes) would affect your NoSQL data structure. Would you need to adjust

document embedding, referencing, or indexing strategies? Explain your reasoning.

Ich denke, das Design hält auch mehr Operationen gut aus:

- Prime-Aktivierungen betreffen immer nur ein Dokument

- Bei Bestellungen brauchen wir keine komplexen Verknüpfungen

- Der Report ist schon optimiert durch die Vorfilterung

Eine Anpassung wäre erst nötig, wenn Kunden wirklich viele Bestellungen

anhäufen. Dann könnte man ältere Bestellungen auslagern, aber für jetzt

passt das Design gut.

#### Five Rules of Thumb

Frage:

Discuss your NoSQL design based on these five rules of thumb for document-based NoSQL

databases. Provide brief, individual explanations on how you applied each rule to your design.

1. Favor embedding unless there is a compelling reason not to.

2. The need to access an object on its own is a compelling reason not to embed it

3. High-cardinality arrays are a compelling reason not to embed.

4. Consider the write/read ratio of a collection/document when denormalizing.

5. Structure your data to match the ways that your application queries and updates it.

1. Das Embedding vom Prime-Status war naheliegend - spart mir Abfragen und zeigt sofort,

ob jemand ohne Liefergebühr bestellen kann. Warum kompliziert, wenn's auch einfach geht?

2. Restaurants hab ich bewusst separat gelassen. Die werden von zu vielen Kunden genutzt

und ändern sich kaum. Wäre Unsinn, die in jedem Benutzer zu duplizieren.

3. Klar, wenn jemand wie verrückt bestellt, könnte das Warenkorb-Objekt irgendwann zu groß werden.

Aber ehrlich - für unseren Kurs und den Use Case ist das völlig ausreichend.

4. Mein Design berücksichtigt, dass wir den Prime-Status zig-mal lesen, aber nur selten ändern.

Der Report ist bewusst redundant angelegt - manchmal muss man halt Speicherplatz opfern, um Zeit zu sparen.

5. Hab das Schema praktisch vom Workflow abgeleitet: erst Prime aktivieren, dann bestellen.

Im Benutzer-Dokument ist alles, was dieser Ablauf braucht, und der Report zeigt genau die Infos, die wir für die Auswertung brauchen.

## Individual - Student 2

Student 1: Winter,Tobias, 12028429

### Use Case Definition and Design

version 1 (weak entity)

#### Textual Description

##### *Title*: Gericht zur Bestellung hinzufügen (weak entity)

**Trigger**: Ein Benutzer wählt ein Gericht aus und fügt es seinem Warenkorb hinzu.

**Preconditions**:

1. Benutzer ist regristriert
2. Gericht existiert
3. Restaurant existiert

**Main Flow**:

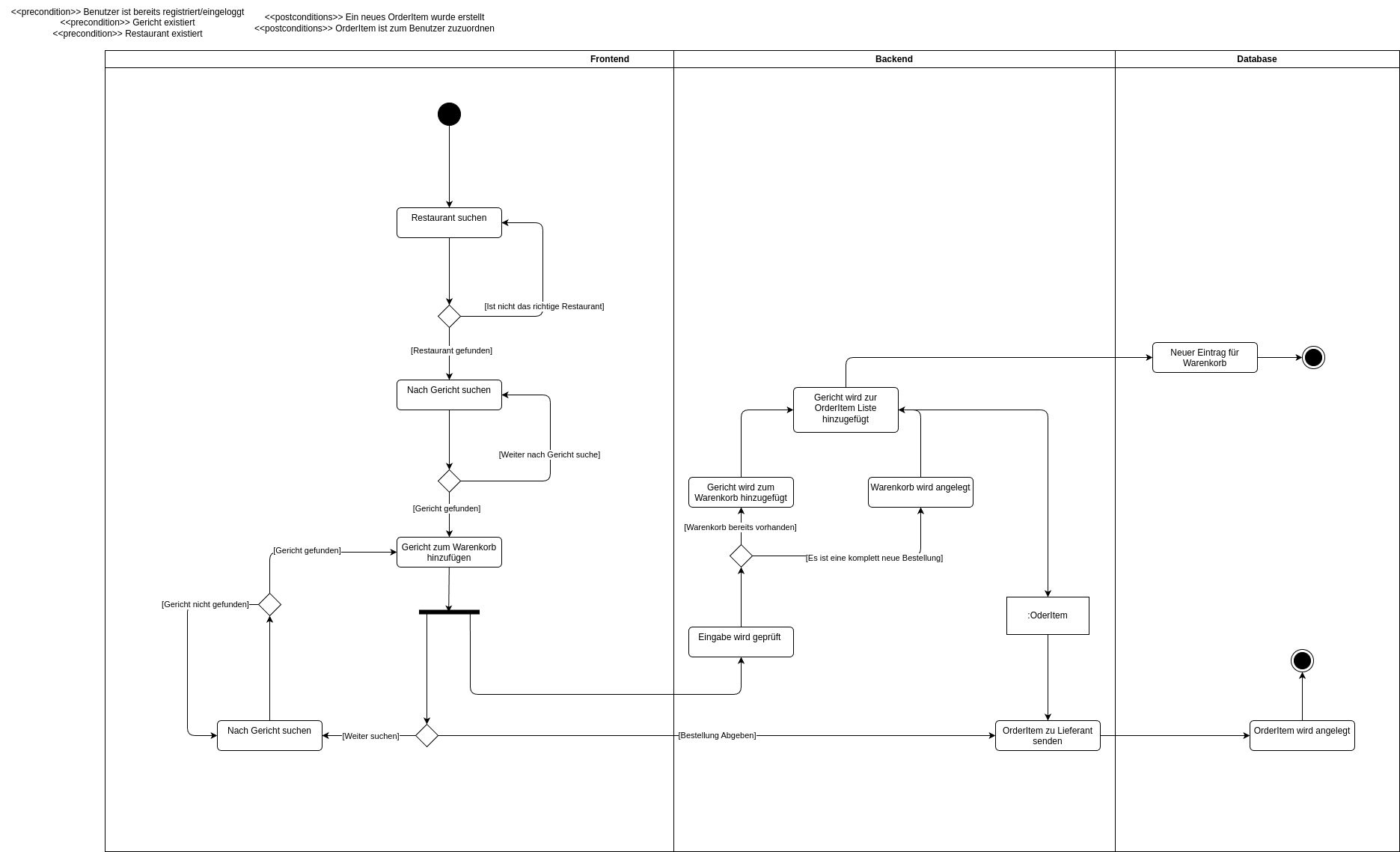
1. Benutzer Browst die App nach Restaurants
2. Benutzer Browst nach Gerichten im ausgewählten Restaurant
3. Benutzer fügt das Gericht zum Warenkorb hinzu
4. Es wird ein OrderItem erstellt
5. Benutzer kann stückzahl anpassen
6. Benutzer kann neue Gerichte aus dem gleichen Restaurant  
   auswählen welche zum gleichen OrderItem hinzgefügt werden
7. Gesamtpreis (Preis essen + eventuelle Liefergebühren) werden berechnet

**Postconditions**:

1. Ein neues OrderItem wurde erstellt
2. OderItem ist dem Benutzer zuzuordnen über den Warenkorb

**Entities**: Benutzer, OrderItem, Gericht, Restauerant, Warenkorb

#### Graphical Representation



### Analytics Report

#### Concept

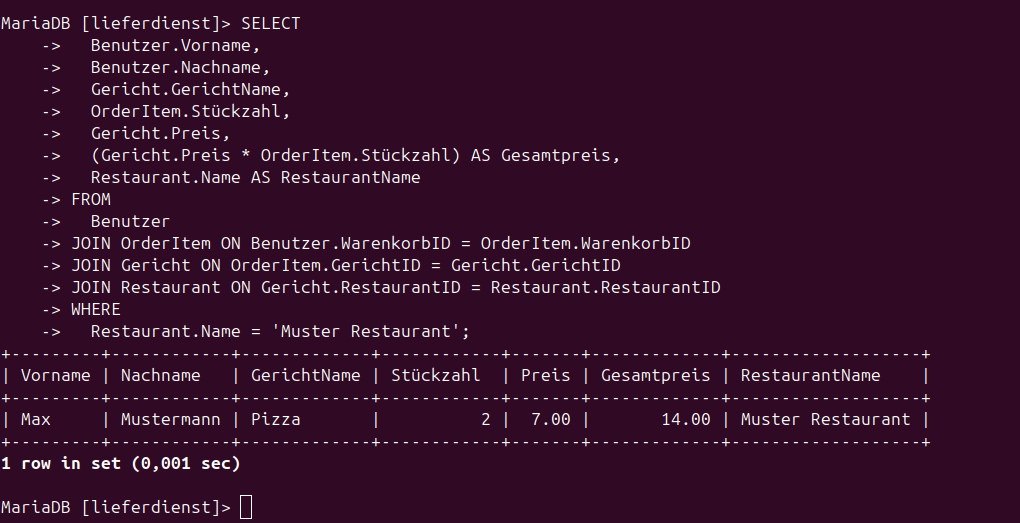
Report zeigt alle Benutzer die aus einem bestimmten Restaurant bestellt haben und  
was bestellt wurde.

Entities: Benutzer, OrderItem, Gericht, Restauerant, Warenkorb

#### Proof of Concept

Description:

Zuerst wird die Datenbank erstellt mit einem CreateScript. Danach wird der usecase eimal durchlaufen um die Datenbank zu befüllen und es wird direkt im anschluss ein report erstellt.



Der usecase wird ein zweitesmal durchlaufen, dieses mal aber mit einem anderen Benutzer.



Um diesen Report zu erstellen filtern wir zuerst nach dem Restaurant und selektieren alle Benutzer die in dem Restaurant bestellt haben, außerdem sehen wir auch was bestellt wurde.

### NoSQL Design

Entities: Benutzer, Restaurant

#### Design Overview

Erklärung zu den Schemas:

Der Benutzer spielt in dem Usecase die Größte rolle.

Das Schema Benutzer:

Benutzer hat eingebettet die App (als meta data) und den Warenkorb.

Im Warenkorb ist eingebettet das Restaurant und die OrderItems. In einem OrderItem is ein Gericht eingebettet.

Das Schema Restaurant:

Hier reicht es nur die App und die Gerichte einzubetten.

Falls man später den Report sehen will aus 1.3.2 Gibt es ein extra report Schema:

Hier sehen wir uns ein Restaurant an. Eingebettet sind die Bestellungen. Darauß lässt sich ein Report erstellen.

#### Expected Execution and Possible Changes

Meine Annahmen:

Das Dokument für Benutzer und Restaurant hilft uns später im erstellen

der Front und Backend da Warenkorb, Restautant und OrderItem mit

Gerichten bereits eingebettet sind. Für das Reporting kann man

das Dokument nur schlecht verwenden aus folgenden Grund:

Für müssen über N Benutzer iterieren und hier im Warenkorb nach

dem richtigen Restaurant filtern. Das Hat 3\*N Vergleichsoperationen.

Deswegen hab ich ein Reporting Dokument das schon nach dem Restaurant

vorgefiltert ist. Hier braucht man dann nur noch über ein Subset der

Totalen Benutzer iterieren.

Frage:

Analyze how changes in the number of database operations of your USE CASE (e.g., more

frequent reads or writes) would affect your NoSQL data structure. Would you need to adjust

document embedding, referencing, or indexing strategies? Explain your reasoning.

Mit dem reporting Dokument braucht man keine neuen Strategien.

Es hat exact die Informationen die man für den Report braucht.

#### Five Rules of Thumb

Frage:

Discuss your NoSQL design based on these five rules of thumb for document-based NoSQL

databases. Provide brief, individual explanations on how you applied each rule to your design.

1. Favor embedding unless there is a compelling reason not to.

2. The need to access an object on its own is a compelling reason not to embed it

3. High-cardinality arrays are a compelling reason not to embed.

4. Consider the write/read ratio of a collection/document when denormalizing.

5. Structure your data to match the ways that your application queries and updates it.

1. Im Benutzer Dokument sieht man ganz gut wie ich diese Regel angewendet

habe. Hier hat man alle nötigen Informationen um eine Bestellung anzulegen.

Man kann in weiteren Zügen andere Dokumente "Querien" um eventuelle checks zu machen.

Wie z.b.: existiert das Restauerant mit diesen Gerichten.

Im Reporting Dokument wurde auch embedding bevorzugt, da man hier nur noch

ein Subset der Benutzer betrachten muss.

2. Ich Eingebettet keine Benutzer in Restaurant obwohl ich Resturant in

Benutzer eingebette. Der Grund dafür - man müsste den Benutzer sonst

an zwei Orten aktuell hatlen. Restaurant Daten ändern sich nicht

so häufig.

3. Diese Regel wird bei mir im moment noch etwas vernachlässigt.

Ich hab versucht mit dem Reporting Dokument nur ein Subset statt

alle einträge von Benutzer einzubetten als "Bestellung".

4. Beim Benutzer wird sich die Bestellung (OrderItem) oft ändern und beim

Reporting wird die "Bestellung" oft aktuallisiert. Hingegen wird die App

als meta Datum eher selten neu geschrieben, deswegen ist diese Entitie auch

eingebettet statt als Eigenes Dokument.

5. Benutzer ist so ausgelegt das man ganz leicht Front und Backend darauf

aufbauen kann. Mit dem Reporting Dokument decke ich den Usecase ab.