Sven Oppliger, Arman Selimovic, Dimitrios Lanaras

Exposee

Dies ist unsere Dokumentation über unser Projekt DeeGeeTec, wir entwickeln eine Applikation mit Datenbank für einen Onlineshop und lernen dabei viel neues, dies wird hier festgehalten.

Deegeetec

M226 + M153 Projekt

§

Inhaltsverzeichnis

[INFORMIEREN 2](#_Toc87359867)

[Einleitung 2](#_Toc87359868)

[Detaillierte Anforderungen 2](#_Toc87359869)

[Analyse 2](#_Toc87359870)

[PLANEN 0](#_Toc87359871)

[ENTSCHEIDEN 0](#_Toc87359872)

[REALISIEREN 0](#_Toc87359873)

[Erweiterungsmöglichkeiten 1](#_Toc87359874)

[KONTROLLIEREN 1](#_Toc87359875)

[Testscript 1](#_Toc87359876)

## INFORMIEREN

## Einleitung

Wir machen eine Applikation für DeeGeeTec. Diese Firma ist Digitec sehr ähnlich. Man kann User Accounts erstellen und mit diesem Computerartikel bestellen. Das Lager soll vorerst nicht digital geführt werden, da Sie einfach die Artikel neu bestellen und so früh wie möglich liefern. Man soll beliebig viele Artikel bestellen können. Dies wird in einer Datenbank geregelt.

Auf DeeGeeTec kann man Artikel in eine Merkliste speichern, diese dann im Korb importieren (oder auch Artikel direkt im Korb platzieren). Damit dies möglich ist muss man angemeldet sein. Wenn der Korb nicht leer ist, kann man diese Artikel bestellen, dabei wird noch eine Bestellung erstellt mit genauen Informationen über die Lieferung und die gekauften Artikel.

## Detaillierte Anforderungen

1. Kunden sind einmalig erfassbar
2. Datenbank in der 3. Normalform
3. Passwörter sind gehashed
4. Datenbank wäre erweiterbar
5. Mitarbeiter können mit Chef erfasst werden
6. Bestellungen können storniert werden
7. Jeder Mitarbeiter hat einen Account (Damit spezifisch Mitarbeiterangebote angezeigt werden können)
8. Jeder Kunde hat einen Account (Damit spezifisch Treueangebote angezeigt werden können)
9. Datenbank sollte erweiterbar sein für ein Lager
10. Kunde kann selbstständig im Programm stornieren
11. Wishlist wird gespeichert
12. Warenkorb leert sich nach neuladen des Programms
13. Bestellungen und Accounts sind nicht einfach löschbar
14. Doppelte Usernamen und Emails sind nicht erlaubt

## Analyse

Da jeder Mitarbeiter und Kunde einen Account hat können wir diese als Spezialisierung von einem Table Account machen damit wir keine Redundanzen haben. Anhand von diesem können wir auch spezielle Angebote anzeigen. Den Chef können wir in einer Rekursion über den Mitarbeiteraccount eintragen lassen, da jeder Mitarbeiter einen Account hat. Die Bestellung selbst läuft über den Account damit Kunden sowie Mitarbeiter im Shop bestellen können. Stornierte Bestellungen dürfen nicht gelöscht werden, sondern werden einfach einen boolean namens cancelled haben. Dazu haben wir eine View für die stornierten Bestellungen und die aktiven. Wir haben uns für eine Denormalisierung bei den Adressen entschieden da es das ganze einfacher macht und zudem würde eine 3. NF bei den Adressen nicht gross Redundanzen reduzieren. Die Wishlist wird online gespeichert damit diese nicht verloren geht beim neuladen der Applikation. Anders als der Warenkorb welcher sich beim neuladen löscht. Wir brauchen eine Zwischentabelle zwischen Order und Items damit wir für jede Order, beliebig viele Items hinzufügen können. Dazu brauchen wir eine Zwischentabelle zwischen Orders und Adressen damit wir jeder Bestellung eine Adresse hinzufügen können. Dies ist die Adresse wo das Paket hingeschickt wird. Zwischen Wishlist und Items brauchen wir natürlich auch eine Zwischentabelle damit wir die Items dort speichern können

# PLANEN



Wir hätten in unserem Projekt das Planen besser durchführen sollen, wir hatten oft unser Ziel nicht erreicht und ein bisschen zu spät unsere Aufgaben erledigt. Dies sollte uns in Zukunft nicht mehr passieren. Darum hatten wir am Schluss ziemlich Zeitdruck, welches nicht Vorteilhaft ist. Wir denken es war zwar gut geplant, aber es war unsere Schuld die Aufgaben hinauszuzögern. Dazu hätten wir jedem pro Woche eine Aufgabe zuteilen sollen.

# ENTSCHEIDEN

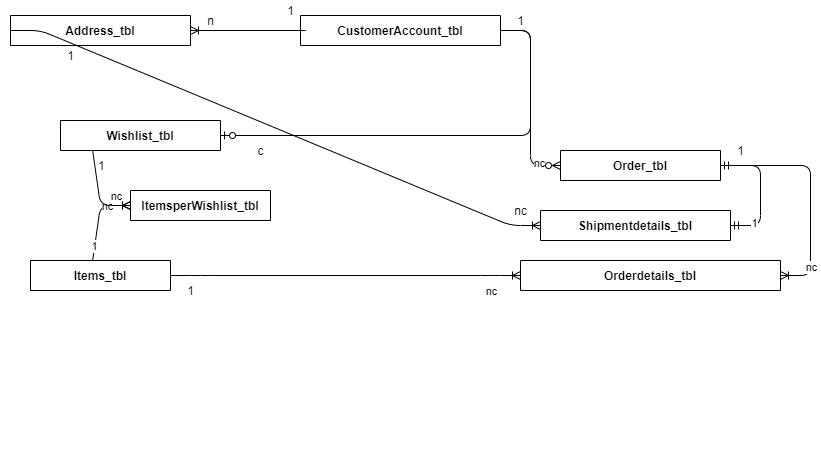
Die einzig grosse Entscheidung für uns war welches Projekt wir umsetzen möchten. In Frage standen Kinoverwaltung, Onlineshop und eine Liegenschaftsverwaltung. Letztendlich haben wir uns für einen Onlineshop entschieden, welcher von Digitec inspiriert ist. Darum heisst unser Projekt DeeGeeTec. Wir haben uns für den Onlineshop entschieden, da dies uns alle am meisten angesprochen hat, es passt gut in unseren Auftrag und wir sind alle oft auf Digitec unterwegs und kennen die Merkmale eines Onlineshops.

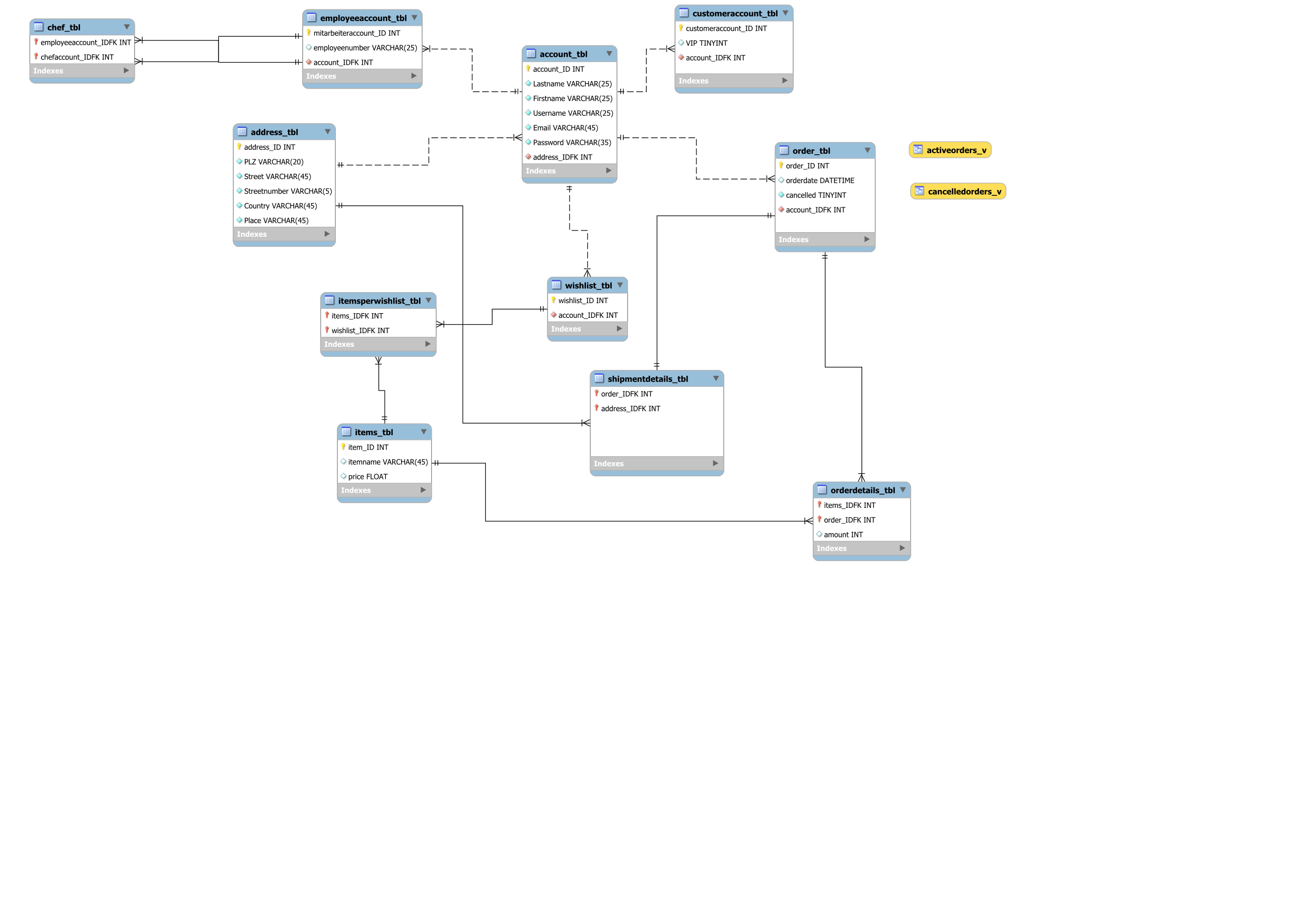
Eine weitere Entscheidung, die wir treffen mussten, ist was wir genau in der DB speichern. Zuerst wollten wir den Warenkorb auch in der DB speichern, schlussendlich haben wir uns entschieden den Warenkorb nur im Programm laufend zu speichern, bis dieses neu gestartet wird. Jedoch die Wishlist wird in der DB gespeichert damit man diese nicht einfach verliert.

# REALISIEREN

Zuerst haben wir ein Use Case Diagramm erstellt, damit wir davon das ERM und die Applikation planen können. Unser sieht wie folgt aus.

Wir haben ein ERM erstellt, dies war unsere erste Planungsphase für die DB. Unser ERD hat sich später davon unterschieden. Wir haben zuerst die DB entwickelt, da unser Programm später darauf aufbaut



Hier hatten wir noch keine Generalisierung/Spezialisierung, sowie die Mitarbeiter Tabelle hatte uns gefehlt. Schlussendlich sah unser ERD wie folgt aus: (grösser in deegeetec.mwb file)

Hier sieht man haben wir die Mitarbeiter- und Kundenaccounts als eine Spezialisierung von Accounts umgesetzt. Dazu gibt es bei Mitarbeiter eine Rekursion auf sich selbst, um einen Chef festlegen zu können.

Wir mussten uns vorallem entscheiden, ob wir die Adresse denormalisieren wollen oder nicht. Es hat für uns keinen Sinn gemacht Länder usw zu erfassen und mit Foreign Keys eine Adresse zu erstellen. Erstens wäre dieses System kompliziert für das Programm selbst und die Redundanzen, welche entstehen sind nicht so gross, damit eine grosse Gefahr besteht. Dazu wird oft denormalisiert bei Adressen wie in unserem Fall. Dies konnten wir neu lernen und dies hilft uns sicher beim nächsten Projekt. Wir dachten immer, denormalisieren wäre schlecht was aber nicht stimmt.

Wir haben uns für zusätzliche Indexierungen beim Account und der Adresse entschieden, um diese schneller zu machen. Beim Account wird zuerst über Usernamen dann E-Mail und als letzten Nachnamen. Da wir dachten Usernamen und Emails sind selten mehrmals vorhanden und der Nachname ist auch auf jeden Fall nicht sehr oft doppelt vorhanden Bei Adresse wird zuerst nach Strasse, Postleitzahl und dann Ortschaft indexiert. Da die Strasse und Postleitzahl sehr wenig doppelt vorkommen, danach der Ortschaft, obwohl es auch hier vielleicht mal doppelte Einträge gibt aber nicht all zu oft.

## Erweiterungsmöglichkeiten

Natürlich haben wir auf Wunsch von DeeGeeTec sinnvolle Erweiterungsmöglichkeiten berücksichtigt, falls diese sich entscheiden sich zu expandieren. Erstens könnte sich DeeGeeTec mit unserem Modell, entscheiden doch ein Lager einzufügen. Dazu müsste man einfach eine Lager Tabelle erstellen und eine Zwischentabelle zwischen Lager und Artikel. Dann könnte man jedem Lager bestimmte Artikel mit der Menge zuweisen. Eine weitere Sinnvolle Erweiterung wäre Abteilungen hinzuzufügen und diese den Mitarbeitern als Foreign Key zuzuweisen. Falls die Firma wächst, wäre es so strukturierter und man könnte bestimmte Auswertungen dazu umsetzen. Theoretisch könnte man, dann eine Umsatz Tabelle erstellen und den verschiedenen Abteilungen zuweisen, damit man sieht wie viel Umsatz jede Abteilung macht. Ein weiterer sinnvoller Ergänzungspunkt wäre Kategorien für die jeweiligen Produkte hinzuzufügen. Dann könnte man auch dort wieder sinnvolle Auswertungen erstellen.

# KONTROLLIEREN

## Testscript

Dokument:  
 

Um unser Projekt zu Testen haben wir ein SQL Testscript -> deegeetec\_tests.sql und für die Applikation haben wir Unit Tests geschrieben.

# REFLEXION

Wir denken, wir haben das Projekt so weit gut umgesetzt. Wir müssen einfach an unserer Planung und Arbeitsaufteilung arbeiten, damit wir pünktlich die Aufgaben fertig haben und nicht so einen Stress wie dieses Mal haben. Dies ist eine wichtige Lektion für uns da wir dieses Problem sicher noch öfters antreffen werden. Wir hätten an der DB vielleicht mehr Testfälle abdecken können, denken jedoch die wichtigsten sind abgedeckt. Die Idee mit dem Onlineshop passt auf jeden Fall gut in den Auftrag und sollte auch gut umsetzbar sein mit unserer DB.

# Arbeitsrapporte

## Woche 1

Sven: Am ERM gearbeitet, Projekt festlegen

Arman: Am ERM gearbeitet, Projekt festlegen

Dimitrios: Am ERM gearbeitet, Projekt festlegen

## Woche 2

Sven: An ERM gearbeitet, Use Case Diagramm erstellt

Arman: Doku angefangen, Applikation angefangen

Dimitrios: Applikation angefangen, Use Case Diagramm erstellt

## Woche 3

Sven: **ERM fertiggestellt**, Applikation gearbeitet, ERD angefangen

Arman: Doku weitergeführt, Zeitplan erstellt

Dimitrios: Sequenzdiagramm angefangen, **Applikationsgrundlage fertiggestellt**

## Woche 4

Sven: ERD weitergearbeitet, **Sequenzdiagramm fertiggestellt**

Arman: Doku ergänzt, Sequenzdiagramm geholfen

Dimitrios: Sequenzdiagramm geholfen, ERD weitergearbeitet

## Woche 5

Sven: ERD fast fertiggestellt, Applikation Javadoc ergänzt

Arman: Javadoc ergänzt, Doku ergänzt, Dimitrios bei Applikation geholfen

Dimitrios: Applikation weitergearbeitet

## Woche 6

Sven: **ERD fertiggestellt** und Arman/Dimitrios erklärt, Javadoc ergänzt, Doku weitergeschrieben

Arman: Unit tests geschrieben, Javadoc ergänzt

Dimitrios: Unit tests geschrieben, Gedanken für API gemacht

## Woche 7

Sven: Doku weitergeschrieben, SQL testing script angefangen

Arman: SQL testing script Sven geholfen, Doku ergänzt

Dimitrios: Unit tests geschrieben, SQL-Abfragen im Code ergänzt

## Woche 8

Sven: **Doku fertiggestellt**, **SQL testing script fertiggestellt**, Javadoc ergänzt

Arman: **Doku fertiggestellt**, **SQL testing script fertiggestellt**

Dimitrios: Applikation kontrolliert und getestet, **Doku fertiggestellt**