Anika Haushälter

Verteilte Systeme

Portfolio

Inhaltsverzeichnis

[Aufgabe 1 2](#_Toc36050857)

[a) 2](#_Toc36050858)

[b) 3](#_Toc36050859)

[c) 4](#_Toc36050860)

[d) 4](#_Toc36050861)

# Aufgabe 1

## a)

Fehler maskieren:

Als Fallbeispiel dient eine Applikation, in der Daten in einer Datenbank gespeichert werden. Mehr kann diese Applikation nicht. Damit die Datenbank die Informationen richtig abspeichern kann, soll ein bestimmtes Schema eingehalten werden. Dieses Schema ist in der Applikation hinterleg.

Nun kann es passieren, dass ein User das Schema nicht einhält. Hierfür reicht es aus, wenn nicht alle Daten vorhanden sind, um das Schema einzuhalten. Der User möchte jedoch die bereits vorhandenen bereits abspeichern, um diese später ergänzen zu können.

Mit dem Fehler kann umgegangen werden, indem die fehlenden Daten mit einer leeren Menge befüllt werden und so an die Datenbank übergeben werden. Dadurch erkennt die Datenbank nicht, dass Daten fehlen. Beim Aufrufen der fehlenden Daten erkennt der User dennoch, dass Daten fehlen.

Fehler beheben:

Als Fallbeispiel dient erneut die Applikation, die Daten in eine Datenbank speichern soll. Diese Datenbank benötigt natürlich eine Festplatte, auf der die Daten gespeichert sind. Nun kann es passieren, dass diese Festplatte irgendwann voll ist.

Dies kann passieren, wenn der Anwender zu viele Daten bereits abgespeichert hat. Es kann auch passieren, wenn nicht nur die Datenbank auf diese Festplatte zugreift, sondern auch andere Applikationen diese Festplatte verwenden.

Das Problem kann nun behoben werden, indem die Datenbank eigenständig erkennt, dass nicht genügend Speicher frei ist. Sie gibt diese Meldung weiter und die Daten können auf andere Festplatten umgeschichtet werden. Sollten keine anderen vorhanden sein, können auch Daten eigenständig gelöscht werden. Dazu sollten diese gelöscht werden, die schon sehr lange nicht mehr verwendet wurden.

## b)

Betrachtet wird das DMS/ECM-System ELO.

Ortstransparenz:

Das System ELO besitzt eine Ortstransparenz. Das System wird auf einem Server installiert. Die Anwendung ruft der User mit seinem PC oder Laptop auf. Um die Anwendung auszuführen, muss der Anwender jedoch nicht wissen, auf welchem Server das System installiert ist. Er muss lediglich eine URL wissen, die bei der Installation vergeben wird. Nach einer einmaligen Eingabe ist diese aber auch auf diesem PC gespeichert.

Mobilitätstransparenz:

Das System kann jederzeit auf einen anderen Server verschoben werden. Wie bereits erwähnt, muss der Nutzer lediglich die URL kennen, unter der das System läuft. Wenn die Ressource verändert wird, muss lediglich beachtet werden, dass die URL nicht verändert wird. Der Nutzer selbst merkt von diesem Umzug demnach nichts.

Persistenztransparenz:

Das System ELO speichert Daten nicht automatisch. Da es ein Dokumentenmanagementsystem ist, auf dem mehrere Anwender gleichzeitig arbeiten können, muss ein Anwender explizit sagen, dass er ein Dokument bearbeitet. Er muss das Dokument auschecken, wodurch es anderen nicht mehr verfügbar ist. Der Anwender muss das Dokument nach seiner Bearbeitung wieder einchecken, wodurch es anderen wieder verfügbar ist. Dabei werden die Änderungen auch gespeichert. Vorher werden die Änderungen nicht gespeichert.

Replikationstransparenz:

ELO läuft lediglich auf dem Server, auf dem es installiert wird. Eine Replikation findet üblicherweise nicht statt. Diese kann zwar ergänzt werden, jedoch ist es kein Standard. Die Ausfallsicherheit ist daher hier nicht gewährleistet. Auch Performanceschwierigkeiten können auftreten. Auch bei dem AddOn der Replikation findet keine Performanceverteilung statt. Hierbei kann lediglich gewährleistet werden, dass bei einem Ausfall die Daten nicht verloren gehen.

## c)

Das CAP-Theorem besagt, dass jedes System nach drei Zielen strebt. Das erste Ziel ist die Konsistenz, das zweite die Verfügbarkeit und das dritte die Partitionstoleranz. Das Theorem besagt jedoch auch, dass die drei Ziele nicht alle gleichzeitig erreicht werden können. Es können maximal zwei dieser Ziele erreicht werden. Ein Beispiel hierfür ist ein Banksystem. Hier ist es notwendig, dass eine hohe Konsistenz und Partitionstoleranz erreicht wird. Die Verfügbarkeit ist jedoch zu vernachlässigen. Das kommt daher, dass bei einer Einzahlung, Abhebung oder Überweisung keine Daten verloren gehen dürfen und diese auch möglichst schnell auf anderen Systemen verfügbar ist. Sonst könnte man zum Beispiel zweimal das gleiche Geld abheben, wenn man schnell genug ist. Auch sollte das System weiterarbeiten können, wenn gerade kein Geld an einem Automaten abgehoben werden kann. Allerdings ist es in Ordnung, beziehungsweise nicht vermeidbar, dass dieser Automat eine Zeit lang nicht mehr funktioniert.

## d)

Logische Uhr:

Diese Uhr dient dazu, verteilte Systeme zu synchronisieren. In verteilten Systemen können nicht die lokalen Uhren verwendet werden, da diese aufwendig zu synchronisieren sind. Daher werden logische Uhren verwendet, um die Reihenfolge festzulegen. Sie definieren, welche Aktionen wann stattgefunden haben. Der absolute Zeitpunkt ist dabei nicht relevant. Lediglich die relative Reihenfolge wird bewertet. Die logische Uhr gibt jeder Aktion eine Nummer, die aufsteigend gezählt wird. Daher weiß man, dass eine Aktion mit einer kleineren Nummer vor der größeren Nummer stattfand.

Lamport Uhr:

Die Lamport Uhr ist eine logische Uhr. Sie wird ergänzt dadurch, dass sie Kausalitäten beachtet. Sie betrachtet also nicht nur, wann welche Aktion abgeschlossen wurde, sondern auch, ob die Aktionen zusammenhängen. Wenn aus einer Aktion zwei Ergebnisse hervorkommen, ist das Ergebnis, was vor dem anderen Ergebnis kam auch in der Reihenfolge vorne. Wenn eine Aktion eine Nachricht sendet und eine andere diese empfängt, bedeutet das, dass der Sender vor dem Empfänger kommt. Die Umsetzung findet statt, indem jeder Prozess eine lokale Uhr bekommt.