Inhaltsverzeichnis

1 SW02 – Übungen Datenmodellierung

1.1 1. Selbststudium

1.1.0.1 Welche Assoziationstypen gibt es? Assoziationstypen (Kardinalitäten genannt) sagen aus, wie oft ein Element der jeweiligen Entitätsmenge in der Beziehung vorkommen kann. Insegesamt gibt es vier: 1. Konditionelle Assoziation Keine bis eins Typ c 2. Einfache Assoziation Genau eins Typ 1 3. Mehrfache-konditionelle Assoziation Keins bis viele Typ mc 4. Mehrfache Assoziation Eins bis viele Typ m #### Wann ist eine Spezialisierung vollständig? Wann ist sie disjunkt? Sofern sie eindeutig identifizierbar ist. Disjunkt ist eine Spezialisierung wenn sich Entitätsmengen gegenseitig ausschliessen. Zum Beispiel kann ein Mitarbeiter nicht gleichzeiteig Führungskraft sein. #### Wozu werden die Normalformen eingesetzt, und aus welchem Grund? Normalformen helfen bei der Vermeidung von Redundanz. Redundanzen führen zu Anomalien. #### Was ist eine Löschanomalie? Erklären Sie dies anhand eines konkreten Beispiels. Eine Lösch-Anomalie (Delete-Anomalie) entsteht, wenn durch das Löschen eines Datensatzes mehr Informationen als erwünscht verloren gehen. Sie entsteht, wenn ein Datensatz mehrere unabhängige Informationen enthält. Durch das Löschen der einen Information wird dann auch die andere gelöscht, obwohl diese noch benötigt wird. Als Beispiel dient Eine Tabelle die Module und Studierende enthält. Wird nun ein Modul gelöscht, werden auch Studierende entfernt. Dies führt zu inkonsistenten Daten. #### Was ist eine funktionale Abhängigkeit? Eine funktionale Abhängigkeit bildet die Grundlage für die Normalisierung von Relationsschemata. Funktional Abhängig bedeutet wenn Attribute zur eindeutigen Identifizierung von anderen Attributen abhängt. Zum Beispiel in einer Kundatendatenbank ist die Strasse und Ort funktional Abhängig vom Namen und Geburtsdatum des Kunden. Denn über den Namen sowie Geburtsdatum ist der Kunde eindeutig identifizierbar. #### Was ist eine volle funktionale Abhängigkeit? Die volle funktionale Abhängigkeit wird mit der 2NF erreicht. Eine vollständig funktionale Abhängigkeit liegt dann vor, wenn dass Nicht-Schlüsselattribut nicht nur von einem Teil der Attribute eines zusammengesetzten Schlüsselkandidaten funktional abhängig ist,

sondern von allen Teilen eines Relationstyps. Die Tabellen werden so aufgeteilt, dass sie eigene Entitäten enthalten. #### Was ist eine transitive Abhängigkeit? Mit der 3NF wird die transitive Abhängigkeit erreicht. Eine transitive Abhängigkeit liegt dann vor, wenn Y von X funktional abhängig und Z von Y, so ist Z von X funktional abhängig. Diese Abhängigkeit ist transitiv. #### Welchen Bezug haben diese Abhängigkeiten zu den Normalformen 1 – 3? Siehe in den vorherigen drei Fragen. #### Was ist der Unterschied zwischen einer Tabelle und einer Relation? Eine Tabelle ist eine Entitätsmenge und eine Relation ist eine Beziehungsmenge, diese sind mittels Schlüssel refernziert. #### Welches sind die zwei wichtigen Schlüsseleigenschaften? Primär-/ Fremdschlüssel #### Warum braucht es für einfach-komplexe und einfach-einfache Beziehungsmengen keine Bezeiehungstabelle? Entitäten sind nachwievor eindeutig identifizierbar.

2 SW02 – Übungen Datenmodellierung

2.1 3. Normalformen

2.1.0.1 Was ist der Schlüssel dieser Tabelle, gemäss Schlüsseleigenschaften? Spiel und User #### In welcher Normalform ist diese Tabelle? Begründen Sie ihre Antwort ganz genau aufgrund ihrer Analyse. Die Tabelle liegt in der ersten Normalform vor. Attribute sind zur eindeutigen Identifikation von andere Attributen abhängig. #### Wie würde diese Tabelle in die dritten Normalform transformiert? Aufteilen von Spiel und Benutzer.

3 SW05 – Übungen SQL Grundlagen

3.1 1. Selbststudium

3.1.0.1 Welche Benutzergruppen gibt es und wie interagieren sie mit der Datenbank? Alle verwenden die gleiche Datenbanksprache. Somit können sie auch Erfahrungen und Wissen untereinander austauschen. ###### Datanbankadmin Verwalttet mit einer Datenbanksprache die Datenbeschreibungen für Tabellen und Merkmale. Unterstützt werden sie von einem Data-Dictionary-System. Mit Hilfe der Datenarchitekten sorgen sie dafür, dass die Beschreibungsdataten einheitlich und konstitent verwaltet und nachgeführt werden. Kontrollieren Datenformate und legen Berechtigungen fest, wie Daten genutzt und verändert werden dürfen. ###### Datanbankspezialist Definieren installieren und überwachen Datenbanken mit Hilfe von Systemtabellen. Mit Hilfe des Systemkatalogs, welcher zur Betriebszeit alle notwendigen Datenbankbeschreibungen und statistischen Angaben umfasst, können die Datenbankspezialisten sich ein aktuelles Bild über sämtliche Datenbanken machen. Dazu nutzen sie vorgegebene Abfragen, welche sich nicht um Benutzerdaten kümmern. Diese sollten aus Datenschutzgründen nur in Ausnahmesituationen zugänglich sein. ###### Abwendungsprogrammier/in Verwenden

die Datenbanksprache um neben Auswertungen auch Änderungen auf den Datenbanken vornehmen zu können. Sie benötigen ein Cursorkonzept um den Datenbankzugriff in ihre softwareentwicklung zu integrireren. ###### Datenanalysten Verwenden Datenbanksprachen für tägliche Auswertungen von fachlichen Informationen. Diese stellen sie Fachabteilungen zur Verfügung. #### Was ist der Unterscheid zwischen mengenorientierten und relationalen Operatoren? ###### mengenorientierte Operatoren Definieren in der relationalen Algebra die Operationen auf einer oder zwei Tabellen mit gleichen Attributen wirken. Diese sind; 1. Vereinigung (Union) 2. Durchschnitt (intersection) 3. Differenz (difference) 4. kartesische Produkt (cartesian prodcut) ###### relationale Operatoren Die Relationen müssen hier nicht vereinigungsverträglich sein. Dies bedeutet, dass man mit diesen Operationen Daten selektiert oder projeziiert. Mit Hilfes des Verbundsoperator (join) können zwei Relationen über ein gemeinsames Merkmal kombiniert werden. Können in Hardware realsiert werden -> entsprechend schnell. #### Was ist der Zusammenhang von mengenorientierten Abfragesprachen und der Relationenalgebra? Alle Operationen lassen sich auf fünf Grundoperatoren der Relationenalgebra zurückführen (Vereinigung, Differenz, kartesisches Produkt, Projektions- und Selektionsoperatoren). Insbesondere ist die Relationenalgebra bei Optimierungen von Nutzen. #### Wie wird die Selektion in SQL umgesetzt? Mit Hilfe der WHERE-Klausel #### Wie wird die Projektion in SQL umgesetzt? Mit der SELECT-Klausel #### Wie wird der Join in SQL umgesetzt? Es gibt mehrere Möglichkeiten: Eine Variante ist die Tabellen Kommagetrennt anzugeben SELECT table1, table2 #### Wie zeigt sich die Eigenschaft von SQL, dass sie deskriptiv ist? Die Ausdrücke beschreiben das gewünschte Resultat und nicht etwa die dazu erforderlichen Rechenschritte. #### Was bedeutet die Aussage, dass SQL relational vollständig ist? Die Aussage bedeutet, dass die fünf grundlegenden Operationen der Relationenalgebra von SQL unterstützt werden. ## 2. Forschungsliteratur #### Was war die Grundidee von SEQUEL? Datenmanipulation oder Abfrage von Daten in einer relationalen Datenbank mittels Keywords ermöglichen welche von Programmieren sowie von unregelmässigen Benutzern verwendet werden kann. #### Welche zwei Gründe sprachen für die Einführung von deklarativen Sprachen? 1. Programmierung war sehr aufwändig und damit teuer. Eine deklarative Sprache kann einfacher und schneller gelernt werden. 2. Die Nutzung von Computern soll nicht nur Spezialisten vorenthalten sein. Der Erfolg ist abhängig von “Normalbenutzern” #### Was ist der grosse Unterscheid zwischen SQUARE und SEQUEL? SEQUEL nutzt ein English-Keyword Format, SQUARE hingegen präzise mathematische Notationen #### Finden Sie einige Unterschiede zwischen dem ursprünglichen SEQUEL und dem heutigen SQL? Integer values in ’’ -> s.255 Union and Intersection mit keyword and/or möglich -> s.257 ## 3. SQL Workbench #### 1.1

```
SELECT * FROM movies;
```

3.1.0.2 1.2

```
SELECT username from user;
```

3.1.0.3 1.3 22

```
SELECT count(*) from category;
```

3.1.0.4 1.4 7292

```
SELECT count(distinct(lastname)) from crew;
```

3.1.0.5 1.5 yes

```
SELECT title from movies
    where title = 'a beautiful mind';
```

3.1.0.6 1.6

```
SELECT * from award
    order by name desc;
```

3.1.0.7 1.7

```
SELECT title from movies
    where budget > 280000000;
```

3.1.0.8 1.8 yes

```
SELECT username from user
    where username like '%norris%';
```

3.1.0.9 1.9

```
SELECT name from keywords
    where name like 'can%';
```

3.1.0.10 1.10

```
SELECT username, location from user
    where location like 'Ba__';
```

3.1.0.11 1.11 age is always 0

```
SELECT username, age from user
    where watched > 800 and age < 11;
```

3.1.0.12 1.12

```
SELECT username, age from user
    where watched > 800 and age between 1 and 11;
```

3.1.0.13 1.13 can be solved with the keyword between as used in ex 1.12. there are 2 users:

```
SELECT count(username) from user
    where watched > 800 and age between 1 and 11;
```

```
SELECT age, username
    FROM user
    WHERE age < 12
    AND watched > 800 AND age <> 0;
```

3.1.0.14 1.14 1691

```
SELECT title, imdbRating from movies
    where imdbRating = 7.0 or imdbRating = 8.0;
```

3.1.0.15 1.15 234

```
SELECT title, imdbRating, metascore from movies
    where metascore > 80 or imdbRating > 8.0 and year > 2012;
```

3.1.0.16 2.1

```
SELECT title, code from movies m, country c, playsInCountry pic
    where pic.m_id = m.id and c.id = pic.c_id;
```

3.1.0.17 2.2 simple

```
SELECT m.title, c.firstname, c.lastname, cf.name
    from movies m, crew c, crewFunction cf, isPartOf ipo
    where ipo.m_id = m.id and ipo.p_id = c.id and c.f_id = cf.id;
```

with joins

```
SELECT m.title, c.firstname, c.lastname, cf.name
    from isPartOf ipo
    left join movies m on ipo.m_id = m.id
    left join crew c on ipo.p_id = c.id
    left join crewFunction cf on c.f_id = cf.id;
```

3.1.0.18 2.3

```
SELECT m.title, c.firstname, c.lastname, cf.name
    from movies m, crew c, crewFunction cf, isPartOf ipo
    where ipo.m_id = m.id and ipo.p_id = c.id and c.f_id = cf.id
    order by m.title asc;
```

3.1.0.19 2.4

```
SELECT m.title from movies m, user u, hasWatched hw
    where hw.m_id = m.id and hw.u_id = u.id and u.username = 'SwissMarco';
```

3.1.0.20 2.5

```
SELECT title,rank FROM movies,hasRank,movieRank
    WHERE movies.id = m_id AND movieRank.id = r_id AND name = "All Time Worldwide"
    ORDER BY rank asc LIMIT 5
```

3.1.0.21 2.6

```
SELECT m.title from movies m, hasKeyword hk, keywords k
    where m.id = hk.m_id and hk.k_id = k.id and k.name = 'teen movie' and m.year > 2000;
```

3.1.0.22 2.7

```
SELECT avg(m.year), c.name from movies m
    left join hasCategory hc on m.id = hc.m_id
    left join category c on c.id = hc.c_id
    group by c.name;
```

3.1.0.23 2.8 19122600

```
SELECT sum(duration) from movies;
```

3.1.0.24 2.9

```
SELECT m.title as Film, c.name as Kategorie
    from movies m, hasCategory hc, category c
    where hc.m_id = m.id and hc.c_id = c.id and c.name = 'horror';
```

3.1.0.25 2.10

```
SELECT name from category
    union
select name from featureCategory;
```

3.1.0.26 2.11

```
SELECT m.title, m.imdbRating, c.name from movies m
    left join hasCategory hc on hc.m_id = m.id
    left join category c on hc.c_id = c.id
    where  m.imdbRating >= 8.0 and (c.name = 'Action' or c.name = 'Comedy');
```

```
SELECT title AS Movie ,name AS Category, imdbRating as IMDB
    FROM movies,hasCategory,category
    WHERE movies.id=m_id AND category.id=c_id AND imdbRating >8 AND name IN  ("action","c
```

3.1.0.27 2.12

```
SELECT distinct(m.title) from movies m
    left join hasAward ha on ha.m_id = m.id
    inner join award a on ha.a_id = a.id
    where a.name = 'Golden Globes';
```

#SW06 - SQL Sprachkonzepte ## Daten manipulieren #### Erstellen Sie eine Tabelle Hilfsassistenten mit gleichen Attributen wie Tabelle Assistenten

```
create table hilfsassistenten as select * from assistenten where 1=0;
```

3.1.0.28 Fügen Sie Hilfsassistenten ein

```
insert into hilfsassistenten values (3006,'Newton','Naturphilosophie',2136),(4001,'Chomsk
```

3.1.0.29 Ändern des Fachgebietes von Newton

```
update hilfsassistenten set fachgebiet = 'idealistische Mataphysik'
    where name = 'Newton';
```

3.1.0.30 Chomsky löschen

```
delete FROM hilfsassistenten where persnr = 4001;
```

3.2 Daten aggregieren

3.2.0.1 Wieviele Profs mit Rang C3 gibt es? 7

```
select count(rang) from professoren;
```

3.2.0.2 Anzahl Semester min

2

```
select min(semester) from studenten;
```

avg

7.625

```
select avg(semester) from studenten;
```

max

18

```
select max(semester) from studenten;
```

3.2.0.3 Semesterwochenstunden gruppiert nach Prof

```
select p.name, sum(v.sws) from professoren p
    left join vorlesungen v on v.gelesenvon = p.persnr
    group by p.name;
```

3.3 Mengenvergleiche (in)

3.3.0.1 Prof finden, der noch keine Prüfung abgenommen hat (mit Inline View)

```
select p.name from professoren p where p.persnr not in (select pr.persnr from pruefen pr
```

3.4 Inner Join

3.4.0.1 Welche Vorlesungen hört Carnap bei welchen Profs

```
select s.name as Student, v.titel as Vorlesung, p.name as Professor
    from hoeren h, studenten s, vorlesungen v, professoren p
    where h.matrnr = s.matrnr and h.vorlnr = v.vorlnr and v.gelesenvon = p.persnr and s.
```

3.5 Left outer join

3.5.0.1 Liste aller Proifs mit Assistent

```
select * from professoren p
    left join assistenten a on p.persnr = a.boss;
```

3.6 Full outer join

3.6.0.1 List aller Studis und Vorlesungen die sie hören. Studi oder Vorlesung trotzdem ausgeben auch wenn nicht besucht

```
select s.matrnr, s.name, v.vorlnr, v.titel from hoeren h
    full outer join studenten s on s.matrnr = h.matrnr
    full outer join vorlesungen v on h.vorlnr = v.vorlnr;
```

3.7 Right outer join

Vorgabe ist dieser Query

```
select v.Titel, s.name
    from vorlesungen v
    left outer join hoeren h on h.VorlNr = v.VorlNr right outer join Studenten s on s.Mat
```

3.7.0.1 Was geschieht wenn right outer mit left outer join ersetzt wird?

Mit dem left outer werden alle Vorlesungen ausgegeben, auch wenn sie nicht von Studenten besucht werden. Was geschieht, wenn in der letzten Zeile full outer join verwendet wird? alle Studenten und Vorlesungen werden ausgegeben, ob die Vorlesung besucht bzw. ob ein Student eine Vorlesung besucht. ## Metadaten #### UseCase von Metadaten und wie kann Data Dictionary durchsucht werden? Ein UseCase ist zum Beispiel das finden einer Spalte wovon man lediglich den Namen weiss. Ein weiterer könnte das auffinden von Spalten mit bestimmen Datenttypen sein.
Das Data Dictionary kann wie eine normale Datenabfrage mittels Query durchsucht werden. Dies funktioniert, weil sämtliche Metadaten auch in Tabellen verwaltet werden. #### Welche Tabellen enthält die Spalte “Fachgebiet” Hilfs-/Assistenten

```
select columns.table_name from information_schema.columns
    where column_name like '%fachgebiet%';
```

3.7.0.2 Welche Spalten haben den Datentyp Integer?

```
select column_name from information_schema.columns
    where data_type = 'integer';
```

3.7.0.3 Welche Tabellen enthält das Information Schema?

alle ausser sich selber

```
Select * from information_schema.tables;
```

3.8 Nullwerte

Gegeben folgendes Query:

```
select 1, count(*) from studenten
    where semester < 13 or semester >= 13
union
    select 2, count(*) from studenten;
```

3.8.0.1 Was fällt beim ausführen auf? Wieso ist Resultat inkonsistent?

Beim 1. Query wird der Null-Wert in den Semestern nicht mitgezählt. Beim Vergleich zählt weder zu < 13 noch zu >=13. Deshalb "fehlt" ein Stundent. #### Wie muss Query definiert werden, damit auch bei Nullwerten konsistentes Resultat ausgegeben wird? Mit den Vergleichskeyword is null oder is not null arbeiten:

```
select 1, count(*) from studenten
    where semester < 13 or semester >= 13 or semester is null
union
    select 2, count(*) from studenten;
```

3.9 Existenz

3.9.0.1 Was ist der Unterschied zwischen den Mengenvergleichen IN und EXISTS? Wann kann man etwas nur mit dem Exists-Operator abfragen?

Mit IN erhält man eine Spalte mit möglichen Resultaten. Diese Resultate werden dann von der left-hand-side jeweils durchsucht und verglichen. EXISTS liefert hingegen Zeilen. Deren Inhalt eigentlich keine Relevanz beizumessen ist. Sofern Zeilen existieren gibt EXISTS true zurück, wenn nicht false.

```
select pr.vorlnr from pruefen pr
    where exists(select s.matrnr, p.persnr, p.rang
        from studenten s, professoren p
        where s.semester >= 10 and p.rang = 'C4');
```

3.10 Fallunterscheidung

```
select matrnr, (case when note >= 5.75 then 'Sehr gut'
    when note >= 4.75 then 'Gut'
    when note >= 3.75 then 'Genügend'
    else 'ungenügend'
    end)
from pruefen;
```

3.10.0.1 Warum muss nur Obergrenze geprüft werden?

Der jeweils erste Case welcher auf true auswertet wird ausgeführt. ## Rekursion Gegeben folgendes Query:

```
with recursive
vorlesungsnachfolger as (
    select vg.titel as von, nf.titel as nach, 1 as länge from voraussetzen vr
        join vorlesungen vg on vg.vorlnr = vr.vorgänger
        join vorlesungen nf on nf.vorlnr = vr.nachfolger
        ),
    pfad(von,nach,länge,folge) as (
    select v0.von, v0.nach, 1, v0.von || ','|| v0.nach
    from vorlesungsnachfolger v0
    union all
    select p.von, vn.nach, p.länge+1,
    p.folge ||','|| vn.nach from vorlesungsnachfolger vn join pfad p on p.nach = vn.von
        )
select * from pfad
where nach like '%thik';
```

3.10.0.2 Was macht das Query genau? Wo befindet sich der Rekursionsschritt? Erklären Sie die Funktionsweise dieser Query.

Das Query gibt die Wege aus, wie Module nacheinander absolviert werden können. Der Rekursionsschritt befindet sich beim `join pfad` im `select` nach dem `union all` ## Windowing #### Query schreiben wo pro Professor Anzahl Wochenstunden zusammenrechnet und den Rang ausgibt. Es sind Windowing-Functions zu verwenden:

```
select p.name, sum(v.sws) as swsh, dense_rank() over(order by sum(v.sws) desc) as Rang fr
inner join vorlesungen v on v.gelesenvon = p.persnr
group by p.name;
```

4 SW07 - SQL Performance

4.1 Selbststudium

4.1.0.1 Welche der Schichten der Datenbankarchitektur sind für die Anfrageoptimierung relevant, und weshalb?

4.1.0.1.1 Schicht1: Mengenorientierte Schnittstelle Es werden Datenstrukturen beschrieben, Operationen auf Mengen bereitgestellt (Relationenalgebra/Kalkül), Zugiffsbedingungen geprüft und Konsistenzanforderungen geprüft. Weiter wird die Syntax geprüft, Namen aufgelöst und Zugriffspfade ausgewählt werden. Bei der Wahl der Zugriffspfade können wesentliche Optimierungen vorgenommen werden. ##### Schicht2: Satzorientierte Schnittstelle Diese Schicht überführt logische Sätze und Zugriffspfade in physische Strukturen. Mit Hilfe der Transaktionenverwaltung wird die Konsistenz gewährleistet. Das Cursorkonzept erlaubt das Navigieren oder Durchlaufen von Datensätzen nach der physischen Speicherungsreiehenfolge, das Positionieren bestimmter Datensätze innerhalb einer Tabelle oder Bereitstellen von Datensätzen in wertabhängiger Sortierreihenfolge. ##### Schicht3: Speicher- und Zugriffsstrukturen Die dritte Schicht implementiert physische Datensätze und Zugriffspfade auf Pages. Speicherstrukturen (Baum-, Hash) sind so ausgelegt, dass die den Zugriff auf externe Speichermedien effizient bewerkstelligen. Physische Cluster oder mehrdimensionale Zugriffpfade können weitere Optimierungen erzielen. ##### Schicht4: Seitenzuordnung Zur Effizienzsteigerung und Recovery-Verfahren unterteilt diese schicht den linearen Adressraum in sogenannte Segmente mit einheitlichen Seitengrenzen. Je nach Dateiverwaltung können in einem Puffer Seiten bereitgestellt werden. Welche auch aktualisiert werden können. Neben der direkten gibt es auch die indirekte Zuordnunge wie Schattenspeicher- oder Cacheverfahren. Dadurch lassen sich mehrere Sieten atomar in den Puffer einbringen. ##### Schicht5: Speicherzuordnung Realisiert die Speicherzuordnungsstrukturen und bietet der Schicht 4 eine blockorientierte Dateiverwaltung. HW-Eigenschaften bleiben verborgen. Unterstützt normalerweise dynamisch wachsende Dateien mit definierbaren Blockgrössen. #### Wie wirkt sich ein Index auf die Leistung des Nested Join (Verschachtelter Verbund) aus? Sind Strukturen sortiert, ist die Laufzeit linear. Das entspricht einem grossen Performancegewinn. #### Was ist ein B-Baum, und wozu dient er im Zusammenhang mit der Anfrageoptimierung? Ein B-Baum ist ein sogenannter blattorientierter Mehrwegbaum. Daten werden in einer Baumstruktur gespeichert. Um aufwändige Seitezugriffe zu umgehen wächst der B-Baum eher in die Breite, anstatt in die Tiefe. Ein B-Baum hat min. n Teilbäume und maximal 2*n Elemente bzw. auch Teilbäume. Mit der Baumstruktur können schnelle Zugriffe auf Daten gewährleistet werden. #### Warum ist eine Query, welche mit Map-Reduce parallelisiert wird, schneller, als wenn sie sequenziell bearbeitet wird? Eine Aufgabe wird in mehrere Teilaufgaben und auf versch. Rechner verteilt. Nach der Verarbeitung werden die Teilresultate gesammelt und zentral ausgewertet. ## Interaktion mit der Datenbank Lokal wird eine Datenbank mit einer grösseren Anzahl Datensätzen erstellt. #### Selektieren Sie eine Studentin über die Matrikelnummer. Wie lange dauert die Anfrage?

```
Use moreUniData2;
select * from moreStudenten where MatrNr = 1012345;
```

Die Anfrage dauert bei 10 Abfragen zwischen `23-33ms` #### Selektieren Sie die gleiche Studentin über den Namen

```
select * from moreStudenten where Name = 'Studentin_12345';
```

Die Anfrage dauert bei 10 Abfragen zwischen `193-203ms` #### Wie viel länger dauert welche Query? Wie erklären Sie sich diesen Unterschied? Die Abfrage über den Namen dauert 6 bis 7 mal länger als über die Matrikel-Nr. Die Matrikel-Nr ist gleichzeitig PrimaryKey und damit automatisch indexiert. Evtl. könnte zudem ein Zahlenvergleich effizienter als ein Stringvergleich sein? ##Query Execution Plan Syntax zu erstellen eines Ausführungsplans

```
EXPLAIN <query>;
EXPLAIN select * from Professoren;
```

4.1.0.2 Zeigen Sie den Explain Plan für die beiden vorherigen Anfragen an. Vergleichen Sie

```
Explain select * from moreStudenten where MatrNr = 1012345;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	moreStudenten	NULL	const	PRIMARY	PRIMARY	4	const	1	100	NULL

```
EXPLAIN select * from moreStudenten where Name = 'Studentin_12345';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	moreStudenten	NULL	ALL	NULL	NULL	NULL	NULL	996887	10	Using where

Bei der Abfrage über den Namen wurden beinahe 1Mio. Rows geprüft (mit Where), bei der Abfrage via Matrikelnummer lediglich eine. Weiter sagt und die Spalte filtered, dass mit der Abfrage zu 100 bzw. 10% eingeschränkt wurde. Was bei erster natürlich nicht besser möglich ist. Weiter gibt der Typ auch Auskunft über die Effizienz, bei const hat die Tabelle einen Treffer - bei all muss ein fullscan gemacht werden. ## Logische Optimierung Führen Sie folgende Query aus:

```
select s.Name, v.titel, p.Name from
(select * from moreStudenten where MatrNr > 55555) s join morehoeren h on (s.MatrNr = h.M
join moreVorlesungen v on (h.VorlNr = v.VorlNr) join moreProfessoren p on (p.PersNr = v.g
```

4.1.0.3 Wie lange dauert die Abfrage? Bei 4 Aufrufen zwischen 12s660ms-12s990ms #### Wie sieht der Explain Plan für diese Anfrage aus? ```sql Explain select s.Name, v.titel, p.Name from (select * from moreStudenten where MatrNr > 55555) s join morehoeren h on (s.MatrNr = h.MatrNr) join moreVorlesungen v on (h.VorlNr = v.VorlNr) join moreProfessoren p on (p.PersNr = v.gelesenVon) where s.Name = 'Studentin_12400'; ``` | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra | | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | | 1 | SIMPLE | h | NULL | ALL | NULL | NULL | NULL | NULL | 10279308 | 100 | Using where | | 1 | SIMPLE | v | NULL | eq_ref | PRIMARY | PRIMARY | 4 | moreunidata2.h.VorlNr | 1 | 100 | Using where | | 1 | SIMPLE | p | NULL | eq_ref | PRIMARY | PRIMARY | 4 | moreunidata2.v.gelesenVon | 1 | 100 | NULL | | 1 | SIMPLE | moreStudenten | NULL | eq_ref | PRIMARY | PRIMARY | 4 | moreunidata2.h.MatrNr | 1 | 10 | Using where | #### Wie sieht der Anfragebaum für diese Anfrage aus? Das Problem ist, dass erst alle Tabellen vereinigt werden und erst am Schluss die Studentin gesucht wird. #### Wie sieht der optimierte Abfragebaum aus? Nach der Studentin soll bereits beim ersten join eingeschränkt werden. #### Wie sieht das SQL der optimierten Query aus? ```sql select s.Name, s.MatrNr, v.titel, p.Name from (select MatrNr, Name from moreStudenten where MatrNr = 1012400) s join morehoeren h on (s.MatrNr = h.MatrNr) join moreVorlesungen v on (h.VorlNr = v.VorlNr) join moreProfessoren p on (p.PersNr = v.gelesenVon); ``` #### Wie lange dauert das logisch optimierte Query? Bei 4 Anfragen zwischen3s151ms-3s261ms#### Wie sieht der Explain Plan für die logisch optimierte Anfrage aus? | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra | | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | | 1 | SIMPLE | moreStudenten | NULL | const | PRIMARY | PRIMARY | 4 | const | 1 | 100 | NULL | | 1 | SIMPLE | h | NULL | ALL | NULL | NULL | NULL | NULL | 10279308 | 10 | Using where | | 1 | SIMPLE | v | NULL | eq_ref | PRIMARY | PRIMARY | 4 | moreunidata2.h.VorlNr | 1 | 100 | Using where | | 1 | SIMPLE | p | NULL | eq_ref | PRIMARY | PRIMARY | 4 | moreunidata2.v.gelesenVon | 1 | 100 | NULL | ## Erstellung von Indexen #### Wie lange dauert die Erstellung des Indexes? ```sql alter table moreHoeren add constraint primary key (MatrNr, VorlNr); ``` Das Ausführen des Befehls dauert10s601ms#### Was bedeutet dies bezüglich Kosten/Nutzen Überlegungen? Die Kosten sind realtiv hoch und dauert länger als obiges optimiertes Query. Der Index lohnt sich sofern die Daten mehr als einmal abgefragt werden. #### Führen Sie das obige Query nochmals aus, wie lange dasuert das Query jetzt? ```sql select s.Name, v.titel, p.Name from (select * from moreStudenten where MatrNr > 555555) s join morehoeren h on (s.MatrNr = h.MatrNr) join moreVorlesungen v on (h.VorlNr = v.VorlNr) join moreProfessoren

p on (p.PersNr = v.gelesenVon) where s.Name = 'Studentin_12400'; ``` Die Abfrage dauert noch zwischen207msbis243ms‘. #### Wie sieht der Explain Plan aus? Woran sehen Sie die Optimierungen gegenüber vorher? | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra | | :— | :— | :— | :— | :— | :— | :— | :— | :— | :— | :— | | 1 | SIMPLE | moreStudenten | NULL | range | PRIMARY | PRIMARY | 4 | NULL | 498443 | 10 | Using where | | 1 | SIMPLE | h | NULL | ref | PRIMARY | PRIMARY | 4 | moreunidata2.moreStudenten.MatrNr | 99 | 100 | Using index | | 1 | SIMPLE | v | NULL | eq_ref | PRIMARY | PRIMARY | 4 | moreunidata2.h.VorlNr | 1 | 100 | Using where | | 1 | SIMPLE | p | NULL | eq_ref | PRIMARY | PRIMARY | 4 | moreunidata2.v.gelesenVon | 1 | 100 | NULL |

Am Typ erkennt man es schnell, vorher war es All, nun nur noch range. Mit der logischen Optimierung wär auch dieses Query noch effizienter. #### Erstellen Sie einen Index auf das Attribute `Name` in der Tabelle `moreStudenten`. Wie sieht die Syntax aus?

```
create index studenName on moreStudenten(Name);
```

4.1.0.4 Führen Sie nun die Query aus Aufgabe 5 erneut aus, wie lange dauert das Query jetzt?

```
select s.Name, v.titel, p.Name from
(select * from moreStudenten where MatrNr > 555555) s join morehoeren h on (s.MatrNr = h
join moreVorlesungen v on (h.VorlNr = v.VorlNr)
join moreProfessoren p on (p.PersNr = v.gelesenVon) where s.Name = 'Studentin_12400';
```

Das Query dauert noch zwischen `24ms-37ms` #### Wie sieht der Explain Plan aus? Woran sehen Sie die Optimierung gegenüber vorher? | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra | | :— | :— | :— | :— | :— | :— | :— | :— | :— | :— | :— | | 1 | SIMPLE | moreStudenten | NULL | range | PRIMARY,studenName | studenName | 36 | NULL | 1 | 100 | Using where; Using index | | 1 | SIMPLE | h | NULL | ref | PRIMARY | PRIMARY | 4 | moreunidata2.moreStudenten.MatrNr | 99 | 100 | Using index | | 1 | SIMPLE | v | NULL | eq_ref | PRIMARY | PRIMARY | 4 | moreunidata2.h.VorlNr | 1 | 100 | Using where | | 1 | SIMPLE | p | NULL | eq_ref | PRIMARY | PRIMARY | 4 | moreunidata2.v.gelesenVon | 1 | 100 | NULL |

Die Filterung ist zu 100 Prozent effizient. #### Wie lange dauert folgende Anfrage? Warum? erklären Sie anhand des Explain plans.

```
select s.Name, v.titel, p.Name from
(select * from moreStudenten where MatrNr > 555555) s join morehoeren h on (s.MatrNr = h
join moreVorlesungen v on (h.VorlNr = v.VorlNr)
join moreProfessoren p on (p.PersNr = v.gelesenVon) where s.Name like 'Studentin_12400';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	h	NULL	range	PRIMARY,VorlNr	PRIMARY	4	NULL	5490486	100	Using where; Using index
1	SIMPLE	v	NULL	eq_ref	PRIMARY,gelesenVon	PRIMARY	4	moreunidata2.h.VorlNr	1	100	Using where
1	SIMPLE	p	NULL	eq_ref	PRIMARY	PRIMARY	4	moreunidata2.v.gelesenVon	1	100	NULL
1	SIMPLE	moreStudenten	NULL	eq_ref	PRIMARY,StudentenName	PRIMARY	4	moreunidata2.h.MatrNr	1	50	Using where

Die Abfrage dauert rund `12s595ms`. Als Vergleichsoperator wird Like verwendet und die Filterausbeute sinkt auf 50%, das heisst dass die Hälfte aller Zeilen gegeben das Argument verglichen werden müssen. #### Wie lange dauert die folgende Anfrage? Warum? Erklären Sie anhand des Explain Plans.

```
select s.Name, v.titel, p.Name from
(select * from moreStudenten where MatrNr > 555555) s join morehoeren h on (s.MatrNr = h
join moreVorlesungen v on (h.VorlNr = v.VorlNr)
join moreProfessoren p on (p.PersNr = v.gelesenVon) where left(s.Name, 17) = 'Studentin_1
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	h	NULL	range	PRIMARY,VorlNr	PRIMARY	4	NULL	5490486	100	Using where; Using index
1	SIMPLE	v	NULL	eq_ref	PRIMARY,gelesenVon	PRIMARY	4	moreunidata2.h.VorlNr	1	100	Using where
1	SIMPLE	p	NULL	eq_ref	PRIMARY	PRIMARY	4	moreunidata2.v.gelesenVon	1	100	NULL
1	SIMPLE	moreStudenten	NULL	eq_ref	PRIMARY,StudentenName	PRIMARY	4	moreunidata2.h.MatrNr	1	10	Using where

Die Abfrage dauert `12s852ms`. Der Index auf dem Namen wird nicht mehr angewendet, weshalb die ganze Range abgesucht wird.

5 SW08 – Datenbankprogrammierung

5.1 1. Lehrmittel

5.1.0.1 Was ist ein Cursor? Definieren Sie das Konzept in Ihren eigenen Worten. Ein Cursor ist ein Zeiger mit welchen man zeilenweise Tupel verarbeiten kann. Basis des Cursors ist eine Select Abfrage welche alle konformen SQL konstrukte unterstützt. Die Verarbeitung finet typisch in einer While-Schlaufe statt. Der Cursor muss explizit geöffnet/geschlossen werden. Ein nächster Datensatz muss mit Fetch aktiviert werden. Die Daten selber werden in Variablen gespeichert. #### Aus welchem Grund (warum) und zu welchem Zweck (wozu) braucht man Cursors? Konventionelle Programme können keine ganze Tabellen auf einen Schlag verarbeiten. Damit können zeilenweise Daten verwendet/manipuliert werden. #### Wozu werden DAtenbanksprachen in andere Sprachen eingebettet? Um eine einheitliche Schnittstelle zu realisieren und iterativ auf Datenelemente zugreifen zu können. #### Was ist der Unterschied zwischen objektorientierten und objektrelationalen Datenbanken? Objektorientierte DBs können über Fremdschlüssel ein Objekt vortäuschen. In der Objektrelationealen DB können zusätzlich Objettypen definieren oder generische Operatoren enthalten. Ausserdem untertützt es die Vererbung von Objekteigenschaften. #### Was ist ein Surrogat? und was hat es mit Objektorientierung zu tun? Ein Surrogat ist ein Fremdschlüssel womit sich jeder Record einer anderen Tabelle eindeutig identifizieren lässt. Damit lassen sich "klassen" bilden und trotzdem die Normalenform einhalten. #### Was ist das NF^2 Modell? Was ist der Zusammenhang von NF^2 mit der Objektorientierung? NF^2 = Non First Normal Form. Diese erlaubt geschachtelte Tabellen. Das bilden von Objekten die wiederum Objekte enthalten? #### Was ist objekt-relationales Mapping, und was sind die Vorteile? Das automatische Mapping zwischen Objektorientierter SW-Entwicklung und relationaler Datenspeicherung. Es bruaucht keine aufwendiges Führen von Mappingalgorithmen.

6 SW10 – Datenbanksicherheit

6.1 Selbststudium

6.1.0.1 Was heisst Datensicherheit? Datensicherheit sind technische und softwaregestützte Massnahmen zum Schutz der Daten vor Verfälschung, Zerstörung oder Verlust. #### Was ist der Unterscheid zwischen Datensicherheit und Datenschutz? Unter Datenschutz versteht man den Schutz der Daten vor unbefugtem Zugriff und Gebrauch. Datensicherheit ist mit obiger Antwort erläutert. #### Welche Rolle Spielen Sichten für die Datensicherheit? Mittels Views können wesentliche Datenschutzmechanismen implementiert werden. Restriktionen werden insofern erreicht, da die Views nur die Tabellen und Tabellenansichten zur Verfgügung stellen, welche die berechtigten Benutzer benötigen. Die View wird auf Basis eines Select-Statements erstellt. Als Nachteil einer View könnte man anbringen, dass diese für Änderungen nicht in jedem Fall verwendet

werden können. Kriterien für eine Änderbare View * Die Sicht bezieht sich auf eine einzige Tabelle (keine Joins erlaubt). * Diese Basistabelle hat einen Primärschlüssel * Der definierende SQL-Ausdruck beinhaltet keine Operationen, welche die Anzahl Zeilen der Resultatmenge beeinflussen (z. B. aggregate, group by, distinct, etc.)

```
CREATE VIEW view-name AS <SELECT-statement>
```

6.1.0.2 Was heisst Grant? Was heisst Grant Option?

Grant ist die Operation um einem Benutzer oder Gruppe Privilegien wie Lese-, Einfüge-, oder Löschoperationen auf bestimmte Tabellen/Views zu vergeben. Mit Revoke können die Privilegien wieder entzogen werden.

```
GRANT <Privileg> ON <Tabelle> TO <Benutzer>
```

```
REVOKE <Privileg> ON <Tabelle> FROM <Benutzer>
```

Die Grant Option ermöglicht es der berechtigten Gruppe oder Benutzer dieses Recht, oder ein eingeschränktes Leserecht, in Eigenkompetenz weiterzugeben und auch später wieder zurückzunehmen. Mit diesem Konzept lassen sich Abhängigkeitsbeziehungen von Rechten festlegen und verwalten. #### Was ist SQL-Injection? Wie schützt man sich davor? Eine SQL Injection ist eine Schwachstelle in einer Applikation (Web) wodurch sich vordefinierte Datenbankabfragen (zB. Formulare) verändern lassen. Wird zum Beispiel eine BenutzerID über ein Post-Parameter übergeben, kann mit der sehr einfachen `or 1=1` eine gültige immer valide Abfrage erzeugen. Der beste Schutz bieten Stored Functions (Statement Sanitation, Prepared Statements) die nur typisierte Parameter erlauben. ## Externe Angriffvektoren | Begriffe | Definition | Gegenmassnahmen | | ————- |———— —-| —-| | Trust Security | Autom. Authentifizierungsmethode von lokalen Benutzer | Beim starten von `initdb` Flag -A verwenden -> `initdb -A` | | Passwords Theft | Passwort Diebstahl | Schutz der Passwörter mittels seriösen Verschlüsselungsalgorithmen und random Salt | | Network Snooping | Abhören des Netzwerkverkehrs | Verbindung mit SSL Verschlüsseln | | Network Spoofing | Vorgauckeln des SQL-Servers (Fake) | Nutzen von Zertifikaten auf Client/Server -> verify CA | | Server / Backup Theft | Diebstahl von / Eindringen zur Hardware | Festplatten Verschlüsseln, mechanische Zerstörung | ## Autorisierung: Views und Grants In der Uni-DB sollen folgende Zugriffsrechte gelten: * C4-Profs dürfen in allen Tabellen lesen und schreiben * C3-Profs dürfen in allen Tabellen lesen und schreiben, ausser in der Tabelle «Professoren»: Diese dürfen sie nur lesen * Assistenten dürfen alle Tabellen lesen. Schreiben dürfen Sie nur in die Tabelle Studenten * Studenten dürfen nichts schreiben und nur folgende Tabellen Lesen: Vorlesungen und Professoren #### Erstellen Sie die Rollen Professor, Assistent, Student und erteilen Sie die oben beschriebenen Berechtigungen mit SQL.

```
create role 'Professor', 'Assistent', 'Student';
grant all on * to Professor; -- don't know how to select "Rang" value
grant select on * to Assistent;
```

```
grant insert, update on Studenten to Assistent;
grant select on Vorlesungen to Student;
grant select on Professoren to Student;
```

6.1.0.3 Erstellen Sie Prof, Assi und Student einen Benutzer

```
SET @s:='';
SELECT @s:= concat(@s, 'CREATE USER ', name, ' IDENTIFIED BY "Passw0rd!";\n') as c
from(
select 'Student' as rolle, name from studenten
union select 'C4', name from professoren where Rang = 'C4' union select 'C3', name from p
)a;
select @s;
```

```
GRANT 'Professor' TO 'Xenokrates';
```

6.1.0.4 Die Studenten sollen in der Tabelle Prüfungen nur Ihre eigenen Noten sehen können. Programmieren Sie eine Lösung

```
SELECT Note from prüfen inner join Studenten S on prüfen.MatrNr = S.MatrNr where s.Name =
```

6.2 SQL-Injection: Programmierung

6.2.0.1 Auf welche weite Arten ist der Server mit SQL angreifbar? Kommen Sie an die Passwörter ran?

? Es werden alle PW ausgegeben. #### Verhinder Sie SQL-Injection mit Hilfe eines ##### Mit einem Client Side prepared statement

```
PreparedStatement ps = connection.prepareStatement("SELECT * from Studenten s join prüfen
                        ".MatrNr = p.MatrNr " +
                        "Where " +
                        "s.Name = ? and s.Passwort = ?");
                ps.setString(1, user);
                ps.setString(2, pw);
                ResultSet resultset = ps.executeQuery();
```

6.2.0.1.1 Mit einem Server Side prepared statement

```
prepare login_statement from 'SELECT * from Studenten s join prüfen p on s.MatrNr = p.Mat
```

7 SW11 – Integritätssicherung

7.1 Selbststudium

Unter dem Begriff Integrität oder Konsistenz (engl. integrity, consistency) versteht man die Widerspruchsfreiheit von Datenbeständen. Eine Datenbank ist integer oder konsistent, wenn die gespeicherten Daten fehlerfrei erfasst sind und den gewünschten Informationsgehalt korrekt wiedergeben. Die Datenintegrität ist dagegen verletzt, wenn Mehrdeutigkeiten oder widersprüchliche Sachverhalte auftreten. ### Welche 3 Arten von strukturellen Integritätsbedingungen gibt es? #### Eindeutigkeitsbedingung Jede Tabelle besitzt einen Identifikationsschlüssel (Primary Key), der jeder Tupel in der Tabelle auf eindeutige Art bestimmt. Das Prüfen auf Eindeutigkeit von PK's selbst wird vom DBS vorgenommen. #### Wertebereichsbedingung Die Merkmale einer Tabelle können nur Datenwerte aus einem vordefinierten Wertebereich annehmen. Das DBS kann nur teilweise die Bedingungen gewährleisten. Als Beispiel kann das System die Bedeutung bzw. Wahrheit einer Adresse nur bedingt prüfen oder die Länge eine VARCHAR(25) sagt auch nichts über deren Inhalt aus. Eine Alternative bieten Aufzählungstypen. Dabei werden alle Datenwerte, die das Merkmal annehmen kann, in einer Liste angegeben (zB. Jahrgang, Berufe, usw.) #### Referenzielle Integritätsbedingung Jeder Wert eines Fremdschlüssels muss effektiv als Schlüsselwert in der referenzierten Tabelle existieren. Dies bedeutet, dass Referenzen nur eingefügt werden können (also der FK), wenn dieser auch existiert. Dann gilt die referenzielle Integritätsbedingung. Diese Bedingung hat natürlich Einfluss auf alle Datenbankoperationen (einfügen, modifzieren, löschen). Wird ein Tupel gelöscht, welches eine Referenz als FK enthält, sind mehrere Varianten von Systemreaktionen möglich: * Restriktive Löschung (restricted deletion); eine Löschoperation wird nicht ausgeführt, sofern andere Tupel aus der gleichen oder auch aus anderen Tabellen Referenzen auf das zu löschende Tupel zeigen. * Fortgesetzte Löschregel (cascaded deletion); ein löschen bewirkt, dass sämtliche abhängigen Tupel auch entfernt werden. * Nullwerte (set null); die Werte wo Fremdschlüssel eingetragen sind werden auf «null» gesetzt. ### Wie lautet die Definition für den Begriff «Referenzielle Integrität»? aus Wikipedia: > „Die referentielle Integrität (auch Beziehungsintegrität) besagt, dass Attributwerte eines Fremdschlüssels auch als Attributwert des Primärschlüssels vorhanden sein müssen.“ > „Über die referentielle Integrität werden in einem DBMS die Beziehungen zwischen Datenobjekten kontrolliert." ### Welche 8 versch. Arten von deklarativen Integritätsbedingungen gibt es? * Primärschlüsseldefinition (Primary Key) * Fremdschlüsseldefinition (Foreign Key -> mit Angaben von References) * Eindeutigkeit (Unique) * Keine Nullwerte (Not Null) * Prüfregel (Check) * Ändern oder Löschen mit Nullsetzen (On Update/On Delete Set Null) * Restriktives Ändern oder Löschen (On Update/On Delete Restrict) * Fortgesetztes Ändern oder Löschen (On Update/On Delete Cascade) ### Können Sie diese 8 deklarativen Integritätsbedingungen den 3 strukturellen Integritätsbedingungen zu ordnen? #### Eindeutigkeitsbedingung * PK * FK * Unique #### Wertebereichsbedingung * Not Null * Check #### Referenzielle Integritätsbedingung * Set Null * Restrict * Cascade ### Wie ist der Begriff der

Transaktion definiert? Eine Transaktion sind Datenbankoperationen welche an Integritätsregeln gebunden sind und die Datenbankzustände konsistenzerhaltend nachführen. Es ist eine Folge von Operationen die atomar, konsistent, isoliert und dauerhaft sein muss. #### Atomarität Eine Transaktion wird entweder komplett durchgeführt, oder sie hinterlässt keine Spuren ihrer Wirkung auf der Datenbank. Zwischenzustände sind für konkurrierende Transaktionen nicht ersichtlich. Damit bildet die Transaktion eine Einheit für die Rücksetzbarkeit nicht abgeschlossener Transaktionen. #### Konsistenz Eine Transaktion bewirkt das Überführen einer Daten aus einem konsistenten Zustand in einen neuen konsistenten Zustand und garantiert die Widerspruchsfreiheit der Daten. Während der Transaktion können Konsistenzbedingungen verletzt sein, bei Transaktionsende müssen diese aber erfüllt werden. #### Isolation Das Prinzip der Isolation verlangt, dass parallel ablaufende Transaktionen dieselben Resultate liefern wie in Einbenutzerumgebungen (Seriell). Damit bleiben die Transaktionen von ungewollten Seiteneffekten geschützt. ##### Isolationsstufen Es gibt vier Isolationsstufen: * Read Uncommitted; keine Konsistenzsicherung * Read Committed; nur festgeschriebene Änderungen werden von anderen Transaktionen gelesen * Repeatable Read; Leseanfragen geben wiederholt dasselbe Resultat * Serializable; setzt die volle serialisierbare ACID-Konsistenz durch #### Dauerhaftigkeit Datenbankzustände müssen so lange gültig und erhalten bleiben, bis sie von Transaktionen verändert werden. Die Dauerhaftigkeit garantiert bei Programmfehlern, Systemabbrüchen oder Fehler auf externen Speichermedien die Wirkung einer korrekt abgeschlossener Transaktion. ### Transaktionen in SQL Um eine Folge als Transaktion zu deklarieren können die Operationen mit `BEGIN TRANSACTION` und durch ein `END_OF_TRANSACTION` gekennzeichnet werden. Mit dem SQL Befehl `COMMIT` werden Änderungen festgeschrieben und allfällige Fehler können mit `ROLLBACK` vollständig widerrufen werden. ### Erklären Sie das Prinzip der Serialisierbarkeit Bei parallel ablaufenden Transaktionen garantiert das Prinzip der Serialisierbarkeit, dass die Resultate auf den Datenbanken identisch sind, gleichgültig ob die Transaktionen streng nacheinander ausgeführt worden sind oder nicht. Eine Menge von Transaktionen ist serialisierbar, wenn die zugehörigen Präzedenzgraphen keine Zyklen aufweisen. ### Was ist der Unterscheid zwischen pesimistischen und optimistischen Verfahren der Serialisierung? Die beiden Verfahren gewährleisten die Serialisierbarkeit. #### Pessimistische Verfahren Pessimistische Verfahren verhindern Konflikte in parallelen Transaktionen im vorherein. Dabei nutzen sie Lock auf dem zu verändernden Objekten. Sind Locks gesetzt, müssen andere Transaktionen warten bis der Lock wieder freigegeben wird (Unlock). Alle Sperren werden im Locking Protocol festgehalten. Das *two-phase locking procotol* grantiert die Serialisierbarkeit parallel ablaufender Transaktionen, indem nur Locks angefordert werden dürfen bis ein erster Unlock vorliegt. Danach dürfen keinen neuen Locks mehr angefodert werden. Im Sinne von Locks gibt es also eine Wachstumsphase (gemessen an Anzahl von Sperren) und eine Schrumpfphase wo die Locks wieder freigegeben werden. #### Optimistische Verfahren Optimistische Verfahren nehmen Konflikte in Kauf, diese werden aber im Nachhinein durch Zurücksetzen der konfliktträchtigen Transaktionen behoben. Grund dieses Verfahren einzusetzen ist, dass man davon aus geht, dass Konflikte selten vorkommen und ohne aufwendiges Sperren/Entsperren kann die Wartezeit verkürzt und die Parallelität gesteigert werden. Dieses Verfahren läuft grundsätzlich in die drei Phasen

Lese-, Validierungs- und Schreibphase ab. * Lesephase; Objekte werden gelesen und in eigenen Arbeitsbereich gespeichert und verarbeitet. Objekte können parallel gelesen werden! * Validierungsphase; Originale werden dahingehend überprüft, ob andere Transaktionen die Objekte in der Zwischenzeit verändert haben. Falls ja wird die Transaktion zurückgestellt. * Schreibephase; neue Daten werden committed Die Mengen *READ_SET* und *WRITE_SET* müssen disjunkt sein, damit die Transaktion serialisierbar bleibt. ### Was heisst Recovery? Welchen Zusammenhang hat es mit dem ACID-Prinzip? Recovery bedeutet das Wiederherstellen eines korrekten Datenbankzustandes nach einem Fehlerfall. Um Transaktionen rückgängig zu machen oder zu wiederholen benötigt ein DBS gewisse Informationen. Diese werden im einer Logdatei geschrieben. Das log file enthält den Zustand des Objektes vor der Änderung (Before Image) sowie auch Marken die den Beginn (BOT = Begin of Transaction) und Ende (EOT = End of Transaction) einer Transaktion signalisieren. Weiter werden auf Anweisung von Programmen oder bei Systemereignissen *Checkpoints* (Sicherungspunkte) gesetzt. Diese enthält eine Liste mit zu diesem Zeitpunkt aktiver Transaktionen. Mit Hilfe der Checkpoints können Rollbacks effizienter durchgeführt werden. Das DBS muss ab dem letzten Checkpoint alle Transaktionen nochmals ausführen. ## Referenzielle Integrität in SQL ### Definieren Sie alle referentiellen Constraints als Primär- und Fremdschlüssel für die Uni-DB

```
select CONSTRAINT_NAME,
       TABLE_NAME,
       COLUMN_NAME,
       ORDINAL_POSITION,
       POSITION_IN_UNIQUE_CONSTRAINT,
       REFERENCED_TABLE_NAME,
       REFERENCED_COLUMN_NAME
from information_schema.KEY_COLUMN_USAGE
where CONSTRAINT_SCHEMA = 'uni4';
```

CONSTRAINT_NAME	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	POSITION_IN_UNIQUE_CONSTRAINT	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
PRIMARY	Professoren	PersNr	1	NULL	NULL	NULL
Raum	Professoren	Raum	1	NULL	NULL	NULL
PRIMARY	Assistenten	PersNr	1	NULL	NULL	NULL
PRIMARY	Vorlesungen	VorlNr	1	NULL	NULL	NULL
PRIMARY	hören	MatrNr	1	NULL	NULL	NULL
PRIMARY	hören	VorlNr	2	NULL	NULL	NULL
PRIMARY	voraussetzen	Vorgänger	1	NULL	NULL	NULL
PRIMARY	voraussetzen	Nachfolger	2	NULL	NULL	NULL
PRIMARY	prüfen	MatrNr	1	NULL	NULL	NULL
PRIMARY	prüfen	VorlNr	2	NULL	NULL	NULL
PRIMARY	studenten	MatrNr	1	NULL	NULL	NULL
assistenten_ibfk_1	Assistenten	Boss	1	1	Professoren	PersNr
vorlesungen_ibfk_1	Vorlesungen	gelesenVon	1	1	Professoren	PersNr

CONSTRAINT_NAME	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	POSITION_IN_UNIQUE_CONSTRAINT	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
hören_ibfk_1	hören	MatrNr	1	1	Studenten	MatrNr
hören_ibfk_2	hören	VorlNr	1	1	Vorlesungen	VorlNr
voraussetzen_ibfk_1	voraussetzen	Vorgänger	1	1	Vorlesungen	VorlNr
voraussetzen_ibfk_2	voraussetzen	Nachfolger	1	1	Vorlesungen	VorlNr
prüfen_ibfk_1	prüfen	MatrNr	1	1	Studenten	MatrNr
prüfen_ibfk_2	prüfen	VorlNr	1	1	Vorlesungen	VorlNr
prüfen_ibfk_3	prüfen	PersNr	1	1	Professoren	PersNr

7.1.1 Sorgen Sie dafür, dass bei Änderungen der PK alle FK aktualisiert werden

7.1.2 table Vorlesungen

```
alter table Vorlesungen drop foreign key vorlesungen_ibfk_1;

alter table Vorlesungen
    add constraint vorlesungen_ibfk_1
        foreign key (gelesenVon) references Professoren (PersNr)
            on update cascade on delete set null;
```

7.1.2.1 table voraussetzen

```
alter table voraussetzen drop foreign key voraussetzen_ibfk_1;

alter table voraussetzen
    add constraint voraussetzen_ibfk_1
        foreign key (Vorgänger) references Vorlesungen (VorlNr)
            on update cascade;

alter table voraussetzen drop foreign key voraussetzen_ibfk_2;

alter table voraussetzen
    add constraint voraussetzen_ibfk_2
        foreign key (Nachfolger) references Vorlesungen (VorlNr)
            on update cascade;
```

7.1.2.2 table studenten

Enthält keine FKs #### table prüfen

```
alter table prüfen drop foreign key prüfen_ibfk_1;

alter table prüfen
```

```
    add constraint prüfen_ibfk_1
        foreign key (MatrNr) references studenten (MatrNr)
            on update cascade;

alter table prüfen drop foreign key prüfen_ibfk_2;

alter table prüfen
    add constraint prüfen_ibfk_2
        foreign key (VorlNr) references Vorlesungen (VorlNr)
            on update cascade;

alter table prüfen drop foreign key prüfen_ibfk_3;

alter table prüfen
    add constraint prüfen_ibfk_3
        foreign key (PersNr) references Professoren (PersNr)
            on update cascade on delete set null;
```

7.1.2.3 Professoren

Enthält keine FKs #### hören

```
alter table hören drop foreign key hören_ibfk_1;

alter table hören
    add constraint hören_ibfk_1
        foreign key (MatrNr) references studenten (MatrNr)
            on update cascade;

alter table hören drop foreign key hören_ibfk_2;

alter table hören
    add constraint hören_ibfk_2
        foreign key (VorlNr) references Vorlesungen (VorlNr)
            on update cascade;
```

7.1.2.4 Assistenten

```
alter table Assistenten drop foreign key assistenten_ibfk_1;

alter table Assistenten
    add constraint assistenten_ibfk_1
        foreign key (Boss) references Professoren (PersNr)
            on update cascade on delete set null;
```

7.2 Statische Integrity Constraints in SQL

7.2.1 Profs können nur die Ränge C2, C3 und C4 haben

```
alter table Professoren
add constraint Rang CHECK (Rang in ('C2', 'C3','C4' ));
```

7.2.2 Profs haben Einzelbüros

```
alter table Professoren
add constraint Raum unique(Raum);
```

7.2.3 Note darf nur zwischen 1.0 und 5.0 sein

```
alter table prüfen
add constraint Note check ( Note >= 1.0 and Note <= 5.0 );
```

7.2.4 Name darf nicht leer sein

```
alter table Professoren
modify Name varchar(30) not null;
alter table studenten
modify Name varchar(30) not null;
alter table Assistenten
modify Name varchar(30) not null;
```

7.3 Trigger

Als Vorbereitung folgendes Statement ausführen: Es soll verhindern, dass der Rang degradiert werden kann und höchstens eine Stufe erhöht wird.

```
DELIMITER $$
CREATE TRIGGER keineDegradierung before update on professoren FOR EACH ROW BEGIN
if (old.Rang is not null) then
if old.Rang = 'C3' and new.Rang = 'C2' then set new.Rang = 'C3'; elseif old.Rang = 'C4' t
elseif new.Rang is null
or new.Rang not in ('C2', 'C3', 'C4') then
set new.Rang = old.Rang; end if;
end if;
END $$
```

7.3.1 Testen des Triggers, prüfen ob Rang degradiert werden kann und Server reponse auswerten

Updatequery

```
update Professoren set Rang = 'C2' where Rang = 'C3' or Rang = 'C4';
```

Console output

```
Query OK, 0 rows affected (0.00 sec)
Rows matched: 7  Changed: 0  Warnings: 0
```

7.3.2 Schreiben Sie einen Trigger der prüft ob Studis Vorbedingungen erfüllen (Modul mit Note <= 3.0)

```
delimiter $$
drop trigger if exists  checkVorbedingung;
create trigger checkVorbedingung
    before insert
    on hören
    for each row
begin
    declare vorgaengerId integer;
    declare grade integer;
    declare success boolean default true;
    declare done boolean default false;
    declare vorgaengerCursor cursor for select Vorgänger from voraussetzen where Nachfolg
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;


    open vorgaengerCursor;

    read_loop:
    LOOP
        FETCH vorgaengerCursor into vorgaengerId;
        if done then
            close vorgaengerCursor;
            leave read_loop;
        end if;
        set grade = (select Note from prüfen where VorlNr = vorgaengerId and MatrNr = new
        if (grade is null or grade > 3.0)
        then
            set success = false;
```

```
        end if;
    end loop;

    if (success)
    then
        set new.MatrNr = new.MatrNr;
        set new.VorlNr = new.VorlNr;
    end if;
end
$$
```

Testquery

```
insert into hören values(25403,5216);
```

Console output

```
`delimiter $$
drop trigger if exists  checkVorbedingung;
create trigger checkVorbedingung
    before insert
    on hören
    for each row
begin
    declare vorgaengerId integer;
    declare grade integer;
    declare success boolean default true;
    declare done boolean default false;
    declare vorgaengerCursor cursor for select Vorgänger from voraussetzen where Nachfolg
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;


    open vorgaengerCursor;

    read_loop:
    LOOP
        FETCH vorgaengerCursor into vorgaengerId;
        if done then
            close vorgaengerCursor;
            leave read_loop;
        end if;
        set grade = (select Note from prüfen where VorlNr = vorgaengerId and MatrNr = new
        if (grade is null or grade > 3.0)
        then
```

```
            set success = false;
        end if;
    end loop;

    if (success)
    then
        set new.MatrNr = new.MatrNr;
        set new.VorlNr = new.VorlNr;
    end if;
end
$$
```

7.4 Tansaction Isolation Levels

Isolation ist die Trennung von Transaktionen auf eine Weise, dass eine laufende Transaktion nicht von einer parallel laufenden Transaktion durch Änderung der benutzten Daten in einen undefinierten Zustand gebracht werden kann. ### Read Uncommited Bei dieser Isolationsebene ignorieren Leseoperationen jegliche Sperren, deshalb können die Anomalien Lost Update, Dirty Read, Non-Repeatable Read und das Phantom-Problem auftreten. Lost Updates können aber immer noch auftreten. ### Read Committed Diese Isolationsebene setzt für die gesamte Transaktion Schreibsperren auf Objekten, die verändert werden sollen, setzt Lesesperren aber nur kurzzeitig beim tatsächlichen Lesen der Daten ein. Daher können Non-Repeatable Read und Phantom Read auftreten, wenn während wiederholten Leseoperationen auf dieselben Daten, zwischen der ersten und der zweiten Leseoperation, eine Schreiboperation einer anderen Transaktion die Daten verändert und committed. ### Repeatable Read Bei dieser Isolationsebene ist sichergestellt, dass wiederholte Leseoperationen mit den gleichen Parametern auch dieselben Ergebnisse haben. Sowohl bei Lese- als auch bei Schreiboperationen werden für die gesamte Dauer der Transaktion Sperren gesetzt. Dies führt dazu, dass bis auf Phantom Reads keine Anomalien auftreten können. ### Serializable Die höchste Isolationsebene garantiert, dass die Wirkung parallel ablaufender Transaktionen exakt dieselbe ist wie sie die entsprechenden Transaktionen zeigen würden, liefen sie nacheinander in Folge ab. Eine Transaktion kann vom Datenbanksystem aus abgebrochen werden muss. Eine Anwendung, die mit einer Datenbank arbeitet, bei der die Isolationsebene Serializable gewählt wurde, muss daher mit Serialisationsfehlern umgehen können und die entsprechende Transaktion gegebenenfalls neu beginnen.

8 Big Data & NoSQL Datenbanken

8.1 Review Questions

8.1.1 Aus welchen Grüden entstand die NoSQL-Bewegung?

Mit dem Siegeszug des WWW entstanden auch weitere BEdürfnisse in Bezug auf Datenhaltung im Petabyte-Bereich. Die bestehenden SQL-Lösungen sind aber weit mehr als reine Datenspeicher und bieten einen hohen Grad an Verarbeitungslogik. Ideal für Konsistenzsicherung, welche aber mit viel Aufwand und Rechenleistung verbunden ist. Dadurch stossen diese bei umfangreichen Datenmengen schneller an ihre Grenzen. Auch hinsichtlich Flexibilität wird man durch die Mächtigkeit des Datenbankverwaltungssystems ein wenig eingeschränkt. Auch aus diesem Grund wurde in der Open Source- und Web-Development-Community schon bald die Entwicklung massiv verteilter DBS vorangetrieben. ### Was ist der Unterscheid zwischen SQL und NoSQL? * Datenbankmodell ist nicht relational * NoSQL ist auf verteilte und horizontale Skalierbarkeit ausgerichtet * NoSQL hat schwache oder keine Schema-Restriktionen * Einfache Datenreplikation * Einfacher Zugriff über eine API * NoSQL hat anderes Konsistenzmodell ### Welche Vorteile hat SQL gegenüber NoSQL? Unter anderem Konsistenz und Sicherheit der Daten. Ideal für Banken und Versicherungen. * Mächtige deklarative Sprachkonstrukte * Schemata, Metadaten * Konsistenzgewährung * Referenzielle Integrität, Trigger * Recovery, Logging * Mehrbenutzerbetrieb, Synchronisierung * User, Rollen, Security * Indexierung ### Welche Vorteile hat NoSQL gegenüebr SQL? Performant und kann immense Datenmengen effizient verarbeiten. Ideal für Web-/SocialMedia Anwendungen. ### Was heisst Schemafreiheit? Es muss keine Tabelle mit Spalten und Datentypen spezifiziert werden. Daten können unter einem beliebigen Schlüssel abgelegt werden. ### In welchen Szenarien setzt man optimal auf welche der beiden Varianten? SQL in Szenarien wo Konsistenz und Sicherheit wichtiger sind, als Performanz und Datenwachstum. ### Welche NoSQL-System-Typen gibt es? Welche spezifischen Vorteile haben die einzelnen Arten jeweils? #### Schlüssel-Wert-Datenbanken Identifizierendes Datenobjekt ist der Schlüssel. Zu jedem Schlüssel gibt es genau ein assoziiertes deskriptives Datenobjekt, welches den Wert des Schlüssels darstellt. Mit der Angabe des Schlüssel kann der zugehörige Wert aus der Datenbank abgefragt werden. Bei grosser Skalierung werden die Key/Values Stores in mehrere Cluster (Shards) aufgeteilt werden. Ein Shard nimmt dabei nur ein Teilraum der Schlüssel bei sich auf. So kann die Datenbank auf viele Rechner verteilt werden. Das Verteilen basiert auf einem Hash und Modulo Operation um auf die Hash Slots verteilen zu können. Die Daten können unter beliebigen Schlüssel abgelegt werden, sind also quasi schemafrei und somit flexibel in der Art der zu speicherenden Daten. #### Spaltenfamilien-Datenbanken Ist eine Erweiterung des Schlüssel-Wert-Konzept mit etwas mehr Struktur. Für die Optimierung des Lesezugriffs werden die Daten nicht zeilenweise, sonder spaltenweise gespeichert. Oft werden nur einzelne Spalten abgefragt und selten die ganze Zeile. Die Spalten werden in Gruppen von Spalten zusammengeführt, welche häufig zusammen gelesen werden. Die Vorteile von

Spaltenfamilien-Datenbanken sind hohe Skalierbarkeit und Verfügbarkeit durch massibe Verteilung (wie K/V-Stores). Ausserdem bieten sie eine Struktur durch ein Schmea mit Zugriffskontrolle und Lokalisation von verteilten Daten auf Ebene Spaltenfamilie. Trotzdem lassen sie innerhalb der Spaltenfamilie genügend Freiraum mit der möglichen Verwendeung von beliebigen Spaltenschlüsseln. ##### BigTable Ist eine von Google vorgestelltes Datenbankmodell für die verteilte Speicherung von strukturierten Daten. Im BigTable-Modell ist eine Tabelle eine dünnbesetzte, verteilte, multidimensionale, sortierte Map mit folgenden Eigenschaften. * Es handelt sich bei der Datenstruktur um eine Abbildung, welche Elemente aus einer Definitionsmenge Elementen einer Zielmenge zuordnet. * Die Abbildung ist sortiert, d. h. es existiert eine Ordnungsrelation für die Schlüssel, welche die Zielelemente adressieren. * Die Adressierung ist mehrdimensional, d. h. die Funktion hat mehr als einen Parameter. * Die Daten werden durch die Abbildung verteilt, d. h. sie können auf vielen verschiede- nen Rechnern an räumlich unterschiedlichen Orten gespeichert sein. * Die Abbildung ist dünnbesiedelt, muss also nicht für jeden möglichen Schlüssel einen Eintrag aufweisen. ###### Rudimentäres Schema Spaltenfamilien sind die einzigen festen Schemaregeln der Tabelle und müssel explizit durch Änderung der Schemas der Tabelle erstellt werden. Innerhalb der Spaltenfamilien können aber beliebige Spaltenschlüssel für die Speicherung von Daten verwendet werden. #### Dokument-Datenbanken Dieses Modell vereinigt die Schemafreiheit von K/V-Speichern mit Möglichkeit zur Strukturiereung der gespeicherten Daten. Effektiv werden nicht Dateien (Office, Video, Musik, usw.) gespeichert sondern strukturierte Daten in Datensätze, welche Dokumente genannt werden. Sie wurden für den Einsatz für Webdienste entwickelt und sind einfach in Webtechnologien wie JavaScript und HTTP integrierbar. Weiter sind sie einfach horizontal skalierbar (sharding). Ein Nachteil aus der Schemafreiheit kann der Verzicht auf referenzielle Integrität und Normalisierung sein. Aber dadurch ist sie extrem flexibel in der Speicherung unterschiedlichster Daten (Variety). Ein Dokument wird als strukturierte Datei (zB. JSON) gespeichert. Diese enthält jeweils Attribut-Wert-Paaren dar. Diese können rekursiv wiederum Listen von Attribut-Wert-Paare enthalten. Die Dokumente haben untereinander keine Beziehung, sondern enthalten eine in sich abgeschlossene Sammlung von Daten. ##### Eigenschaften • Sie ist eine Schlüssel-Wert Datenbank. • Die gespeicherten Datenobjekte als Werte zu den Schlüsseln werden Dokumente genannt; die Schlüssel dienen der Identifikation. • Die Dokumente enthalten Datenstrukturen in der Form von rekursiv verschachtelten Attribut-Wert-Paaren ohne referenzielle Integrität. • Diese Datenstrukturen sind schemafrei, d. h. in jedem Dokument können beliebige Attribute verwendet werden, ohne diese zuerst in einem Schema zu definieren. #### XML-Datenbanken Eine native XML-Datenbank hat folgende Eigenschaften * Die Daten werden in Dokumenten gespeichert, ist also eine Dokument-Datenbank. * Die strukturierten Daten in den Dokumenten sind kompatibel mit dem XML-Standard. * XML-Technologien wie XPath, XQuery und XSL/T können zur Abfrage und Mani- pulation der Daten angewendet werden. Native XML-Datenbanken speichern Daten streng hierarchisch in einer Baumstruktur. Sie sind dann besonders gut geeignet, wenn hierarchische Daten in standardisiertem Format gespeichert werden sollen, beispielsweise bei Web-Services in der serviceorien- tierten Architektur (SoA). Ein großer Vorteil ist der vereinfachte Datenimport in die Datenbank,

der bei einigen Datenbanksystemen durch simples Drag and Drop von XML-Dateien erreicht werden kann. #### Graphdatenbanken Haben als strukturierendes Schema dasjenige des Eigenschaftsgraphen. Daten werden in Form von Knoten und Kanten gespeichert, welche zu einem Knoten- bzw. Kantentyp gehört und Daten in Form von Attribut-Wert-Paare enthalten. Es handelt sich um ein implizites Schema. Datenobjekte können zu einem bisher nicht existierenden Knoten- oder Kantentyp direkt eingefügt werden ohne den Typ vorher zu definieren. Die Beziehungen zwischen Datenobjekten sind explizit als Kanten vorhanden. Die Referenzielle Integrität wird vom DBS sichergestellt. Graphdatenbanken kommen überall dort zum Einsatz, wo Daten in Netzwerken organisiert sind. Der einzelne Datensatz ist weniger wichtig, sondern die Verknüpfung der Datensätze untereinander (zb. Social Media, Infrastrukturnetzen, Internet-Routing, usw.). Vorteil ist die Eigenschaft der indexfreien Nachbarschaft (findet direkten Nachbarn ohne sämtliche Kanten zu berücksichten). Der Aufwand für die Abfrage von Beziehungen zu einem Knoten ist konstant, unabhänig vom Datenvolumen. Für den schnellen Zugriff auf Knoten und Kanten bereitzustellen werden Indexe eingesetzt, wobei dieser als balancierte B-Bäume aufgebaut werden. Heutige Graphdatenbanken unterstützen das Sharding noch nicht und kann höchstens mit domänenspezifischen Wissen verteilt werden. ##### Eigenschaften • Die Daten und/oder das Schema werden als Graphen (siehe Abschn. 2.4) oder graphähnlichen Strukturen abgebildet, welche das Konzept von Graphen generalisieren (z. B. Hypergraphen). • Datenmanipulationen werden als Graph-Transformationen ausgedrückt, oder als Operationen, welche direkt typische Eigenschaften von Graphen ansprechen (z. B. Pfade, Nachbarschaften, Subgraphen, Zusammenhänge, etc.). • Die Datenbank unterstützt die Prüfung von Integrita€tsbedingungen, welche die Datenkonsistenz sicherstellt. Die Definition von Konsistenz bezieht sich direkt auf Graph- Strukturen (z. B. Knoten- und Kantentypen, Attribut-Wertebereiche und referenzielle Integrität der Kanten).

9 Graphdatenbanken

9.1 Selbststudium

9.1.1 Was ist ein Eigenschaftsgraph?

Ein Eigenschaftsgraph (engl. property graph) besteht aus Knoten (Konzepte, Objekte) und gerichteten Kanten (Beziehungen), die Knoten verbinden. Sowohl die Knoten wie die Kanten tragen einen Namen (engl. label) und können Eigenschaften (engl. properties) aufweisen. Die Eigenschaften werden als Attribut-Wert-Paare der Form (attribute: value) mit Attributnamen und zugehörigen Werten charakterisiert. ### Wie könnten Eigenschaftsgraphen mathematisch definiert werden? Mit der Graphentheorie. ### Welche Gemeinsamkeiten und Unterschiede bestehen zwischen relationalen und graphorientierten Datenbanken? Entitäten entsprechen Knoten und die Beziehungen entsprechen

Kanten. ### Wie werden Entitätsmengen im Graphschema umgesetzt? Jede Entitätsmenge wird als eigenständiger Knoten dargestellt. Alle Merkmale der Entitätsmenge wird als Eingenschaft des Knoten geführt. ### Wie werden Beziehungsmengen im Graphschema umgesetzt? Jede Beziehungsmenge kann als ungerichtete Kante in der Graphdatenbank definiert werden. Die Eigenschaften der Beziehungsmenge werden den Kanten zugeordnet (attributierte Kanten). ### Wie werden Attribute im Graphschema umgesetzt? Die Kanten tragen Labels ## Cypher Tutorial Cypher Tutorial (https://sql-nosql.org/de/cypher-tutorial) ## Neo4J Workbench ### Basic Queries

9.1.1.1 Basic Queries - Exercise 1:

Question: Find the movie A Beautiful Mind

```
MATCH (m:Movie) WHERE lower(m.movie_name)="a beautiful mind" RETURN m
```

9.1.1.2 Basic Queries - Exercise 2:

Question: Find the movie a clockwork orange (Hint: use the lower function)

```
MATCH (m:Movie) WHERE lower(m.movie_name)="a clockwork orange" RETURN m
```

9.1.1.3 Basic Queries - Exercise 3:

Question: Find all actors order by actor_name What is the problem? To solve it, continue to the next exercise

```
MATCH (a:Actor) RETURN a ORDER BY (a:actor_name)
```

```
ERROR: Resultset too large (over 1000 rows)
```

9.1.1.4 Basic Queries - Exercise 4:

Question: Find 50 actors order by actor_name. (Hint: Use this time limit)

```
MATCH (a:Actor)  RETURN a ORDER BY (a:actor_name) limit 50
```

9.1.1.5 Basic Queries - Exercise 5:

Question: List all categories by their name (category_name) orderd by their name in reverse order(Hint: Switch to table view)

```
MATCH (n:Category)
RETURN n.category_name
ORDER BY n.category_name DESC
```

9.1.1.6 Basic Queries - Exercise 6: Question: List all Movies who have an IMDB Rating (movie_imdbRating) below 5 and Metascore is bigger than 55 (movie_metascore)

```
MATCH (n:Movie)
WHERE (n.movie_imdbRating < '5')AND( n.movie_metascore > 55)
RETURN n
```

9.1.2 Advanced Queries

9.1.2.1 Advanced Queries - Exercise 1: Question: List all features of the movie Shrek

```
MATCH (n:Movie{movie_name:"Shrek"})-[:linked_to]->(Feature)
RETURN Feature.feature_name
```

9.1.2.2 Advanced Queries - Exercise 2: Question: Show all movies who are assoziated to the keyword "swordsman"

```
MATCH (n:Movie)-[:has]->(Keyword{keyword_name:"swordsman"})
RETURN n.movie_name
```

```
13 Assassins
Kingdom of Heaven
```

9.1.2.3 Advanced Queries - Exercise 3: Question 1: How many users have watched the movie Titanic

```
MATCH (n:Movie{movie_name:"Titanic"})<-[:watched]-(User)
RETURN count(n)
```

```
320
```

Question 2: Find all movies who has been watched by more then 400 users (Hint: Use with)

```
MATCH (n:Movie)-[:watched]-(User)
WITH n, count(User) AS watchers
WHERE watchers > 400
RETURN (n.movie_name)
```

Advanced Queries - Exercise 4: Question 1: List all Movies who are related trough user like with the movie Agora.

```
MATCH (user:User) --> (movie:Movie)
WHERE movie.movie_name = "Agora"
RETURN (user), (movie)
```

Question 2: Show all related to the related.