Informatikklausur

Datenbanken

Normalformen:

- Datenbanksystem für elektronische Datenverwaltung
- DBMS = Datenbankmanagementsystem (zB SQL)
- Datensätze werden in Datenbank *oder Datenbasis* zusammengefasst
- Zeile in Datenbank = Entität (Objekt der realen Welt) = Datensatz
- Spalte in Datenbank = Datenfeld
- Tabelle = Relation
- Zusammenfassung gleichartiger Entitäten = Entitätstyp = Tabelle
- Felddatentyp (Text, Zahl, Autowert, etc.)
- Doppelungen = Redundanzen
- Datenunregelmäßigkeiten = Datenanomalien
- Fremdschlüssel = Sekundärschlüssel = Attribut in einer Tabelle, das in einer Entität, zu der eine Beziehung besteht, als Primärschlüssel definiert wurde
- Primärschlüssel für eindeutige Identifizierung eines Objekts, dient zur Identifikation einer Entität
- Anzahl der an Beziehung beteiligten Entitäten = Beziehungsmenge = Kardinalität
- Normalisierung = Eliminierung von Anomalien / Vermeidung von Redundanzen (Vorteil: geringerer Arbeitsaufwand, Fehlervermeidung, Eingabeerleichterung, Speicherplatzeinsparung (keine Redundanzen))
- Eigenschaft, die eine Entität beschreibt = Attribut
- Keine Mehrfachmerkmale = atomar (1. Normalform)
- funktionale Abhängigkeit (2. Normalform) = jedes Nichtschlüsselattribut ist nur durch den gesamten Primärschlüssel eindeutig bestimmbar
- transitive Abhängigkeit (3. Normalform) = ein Nichtschlüsselattribut darf nicht (durch ein anderes Nichtschlüsselattribut/transitiv) vom Primärschlüssel abhängig sein (das Nichtschlüsselattribut darf sich nicht aus einem anderen Nichtschlüsselattribut ergeben)
- Relationsschema: Entität(Primärschlüssel, Attribut1, Attribut2, Fremdschlüssel(pfeilHoch)). Falls n-m-Beziehung, Relatition für Primärschlüssel beider Entitäten Relationsentität(Fremdschlüssel1(pfeilHoch), Fremdschlüssel2(pfeilHoch), Attribut)

Java-Implementation:

- Für Funktionalität (wie QueryResult) muss das sqlite-jdbc Paket eingebunden werden
- INFO: Abiturklassen liegen der Klausur bei, Beispielcode für ein SQL Statement und dessen Ergebnis:

```
private DatabaseConnector connector;
connector = new DatabaseConnector("Irgendein Quatsch", 8888, "path/to/database.db",
"username", "password");
connector.executeStatement("SELECT column FROM myTable");
```

QueryResult queryResult = connector.getCurrentQueryResult();
String[][] table = queryResult.getData(); //Tabelleninhalt [Zeile][Spalte]
String[] columnNames = queryResult.getColumnNames(); //Spaltennamen

SQL-Statements:

Generelles Auswählen:

SELECT * FROM Customers
Zuerst Befehl, dann die Spalten mit (*) dann das Stichwort FROM und die Tabelle

Etwas in eine Tabelle einfügen:

INSERT INTO Employees (EmployeeID, LastName, FirstName, BirthDate, Photo, Notes)
VALUES ("11", "Ergin", "Junus", "1991-10-03", "EmpID1.pic", "Hallo Welt, ich lerne SQL")
Es wird mit dem Stichwort INSERT ein neuer Datensatz angelegt, danach kommt in den Klammern die Spalten und in die zweite Klammer die Werte für die Spalte

Es wird ein Average (Durchschnitt) mit einem Join kombiniert:

SELECT AVG(Quantity*Price) AS Average FROM Orders LEFT JOIN OrderDetails ON OrderDetails.OrderID =
Orders.OrderID LEFT JOIN Products ON OrderDetails.ProductID = Products.PrductID WHERE OrderDate LIKE
"1996%"
Es wird mit dem Befehl AVG ein Durchschnitt von den Werten in der Klammer berechnet. Mit dem AS wird die
Durchschnittsspalte benannt, die dann nach dem Beispiel "Average" heißt.

Es wird ein Count (Zähler) gemacht:

SELECT Count(Quantity*Price) AS Orders FROM Orders LEFT JOIN OrderDetails ON OrderDetails.OrderID =
Orders.OrderID LEFT JOIN Products ON OrderDetails.ProductID = Products.ProductID WHERE OrderDate LIKE
"1996%"
Es wird mit dem Befehl Count die Zeilen der gesamten Spalte in den Klammern gezählt. Mit dem AS wird die Zählspalte benannt.
Hier in dem Beispiel lautet die Spalte mit den Zählungen also "Orders".

Es wird eine neue Tabelle erstellt:

CREATE TABLE MyTable AS
SELECT * FROM Orders LEFT JOIN Customers ON Customers.CustomerID = Orders.CustomerID LEFT JOIN OrderDetails ON Orders.OrderID =
OrderDetails.OrderID WHERE CustomerName LIKE "M%" AND Orders.ShipperID = 1
Zunächst gibt man den Befehl "CREATE TABLE" ein, um eine Tabelle zu erstellen. Dahinter kommt der Name der Tabelle. Das Stichwort AS dient dazu, die neue Tabelle zu definieren, also was in die neue Tabelle alles reinsoll.
Mit SELECT und den ganzen JOINS wählt man eine Tabelle aus, die dann durch den "CREATE TABLE"-Befehl als neue Tabelle gespeichert wird.

Tabelle updaten, verändern:
UPDATE Employees SET LastName = "Miller" WHERE EmployeeID = 2
Mit dem Befehl "UPDATE" gibt man an, dass man einen Datensatz verändern möchte. Danch gibt man die Tabelle an, dann das Stichwort "SET" und dann die Spalte, wonach dann die Bedingung kommt. Mit WHERE legt man dann nämlich fest
, wo der Wert verändert werden soll.
Tabelle löschen:
DROP TABLE MyTable
Zunächst den Befehl "DROP TABLE" eingeben. Danach den Namen der Tabelle, die man gerne löschen möchte.
Zweimal JOIN (Beispiel):
SELECT * FROM Orders LEFT JOIN Customers ON Customers.CustomerID = Orders.CustomerID LEFT JOIN OrderDetails ON Orders.OrderID =
OrderDetails.OrderID WHERE CustomerName LIKE "M%" AND Orders.ShipperID = 1
LEFT JOIN:
SELECT * FROM Orders LEFT JOIN Customers ON Customers.CustomerID = Orders.CustomerID WHERE CustomerName LIKE "M%"
"ON" ist die Bedinungen, wo die gejointe Tabelle überall ihre Werte mit einspielen soll. Das Stichwort WHERE gehöt zu dem SELECT.
Mit Buchstabenanfang suchen:
SELECT * FROM Customers WHERE CustomerName LIKE "M%"
Das Prozentzeichen nach dem M gibt an, dass hier alle möglichen Zeichen folgen können. Es soll nur mit M anfangen.
Oder-Abfrage:
SELECT * FROM Customers WHERE City IN ("Madrid", "Berlin")
Das IN kennzeichnet die Oder-Abfrage. Wennn die Stadt "Madrid" oder "Berlin" ist, soll die Zeile angezeigt werden.
Ordnen der Einträge:
SELECT * FROM Employees ORDER BY LastName DESC
Das ORDER BY sorgt für das ordnen nach einer Spalte. DESC sagt hier aus, dass es abwärts geordet werden soll. Somit wäre der letzte Buchstabe im Alphabet hier als erstes. ASC würde eine aufwärts geordnete Spalte anzeigen.
Spaltenrechnung (Beispiel):
SELECT OrderDate, ProductName, Quantity, Price, Quantity * Price AS Total FROM Orders LEFT JOIN Customers ON Customers.CustomerID = Orders.CustomerID LEFT JOIN OrderDetails ON Orders.OrderID =
OrderDetails.OrderID WHERE CustomerName LIKE "M%" AND Orders.ShipperID = 1
Summenberechnung:
SELECT SUM(Quantity * Price) AS Total FROM Orders LEFT JOIN Customers ON Customers.CustomerID = Orders.CustomerID LEFT JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID WHERE CustomerName LIKE "M%" AND Orders.ShipperID = 1

Mit dem Befehl SUM wird eine Spalte ausgewählt, die in den Klammern angegebn wird, wovon die Summe berechnet wird.

Geklammerte Abfrage (Beispiel):

SELECT * FROM Customers WHERE (CustomerName LIKE "A%" AND Country = "Mexico") OR (CustomerName LIKE "L%" AND Country = "USA")

Nur unterschiedliche Werte auflisten:

SELECT DISTINCT column FROM tabelle

Einer Tabelle eine Spalte hinzufügen, ändern oder löschen:

ALTER TABLE tabelle ADD column datentyp
ALTER TABLE tabelle DROP COLUMN column
ALTER TABLE tabelle MODIFY COLUMN column datentyp

Ausgabe Gruppieren:

Gruppiert die Ausgabe entsprechend der angegebenen Spalte. Oft in Verwendung mit SUM(), MIN(), MAX(), AVG() und COUNT() genutzt.
SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;
Gibt die Anzahl an Customers pro Land aus.

Binärbaum

Definition:

- Dynamische, nichtlineare Datenstruktur, bei der jeder Knoten 0/1/2 Nachfolger hat (Binärbaum wegen zwei Nachfolgern)
- Baum ist eine Menge von Knoten. Ohne Vorgänger ist dieser ein Knoten, ohne Nachfolger ein Blatt und wenn beides existiert handelt es sich um einen inneren Knoten
- Algorithmen zur Verarbeitung der Binärbäume sind oft rekursiv, daher gilt folgende, rekursive Definition:
- Ein Binärbaum ist entweder leer oder er besteht aus einer Wurzel mit einem linken und einem rechten Binärbaum.
- Ist ein Baum leer gilt das als Abbruchbedingung für die rekursive Definition
- Die Wurzel (und folgende, innere Knoten) haben also einen linken und rechten Teilbaum (wobei diese auch leer sein können)
- Gegenüber einer zB Liste muss die gesamte Datenstruktur nicht durchiteriert werden da durch Ausschluss bei jeder Entscheidung große Teile unwichtiger Daten aussortiert werden können
- Laufzeit: log(n)

Implementation:

- BinaryTree<String> baum = new BinaryTree<String>(Content pContent, BinaryTree pLeftTree, BinaryTree pRightTree);
- Entweder sind die Funktionsklammern leer, ein Inhalt ist vorhanden oder Inhalt + zwei Bäume

Traversierung:

- Prozess des Übertragens eines Baums in eine Liste

- **preorder** (Knoten-Links-Rechts)
- **inorder** (Links-Knoten-Rechts)
- **postorder** (Links-Rechts-Knoten)
- Funktion: nach der Prüfung ob übergebener Knoten leer ist wird entweder der Knoten einem String hinzugefügt oder die Funktion erneut mit dem linken/rechten Teilbaum aufgerufen (je nach Traversierungsverfahren)

Kenngrößen:

- **Grad eines Knotens**: Anzahl der Nachfolger des Knotens (0/1/2)
- **Inhalt**: Wert/Objekt des Knotens
- **Teilbaum**: Unterbaum eines Baums (rekursiv)
- **Tiefe**: Anzahl der gerichteten Kanten bis zur Wurzel (Striche bzw. Anzahl Knoten - 1)
- **Höhe**: Tiefe des tiefsten Knotens im Baum