4. Übungsblatt (WS 2021)

6.0 VU Datenbanksysteme

Informationen zum Übungsblatt

Allgemeines

Inhalt des vierten Übungsblattes sind die (vertiefenden) Funktionen von SQL. Sie werden das Erstellen eines Datenbankschemas üben, das Definieren und Sicherstellen der Datenintegrität, sowie das Schreiben komplexerer SQL Anfragen (inklusive prozeduraler Programme).

In dieser Übung geben Sie eine einzige ZIP Datei ab (max. 5MB). Diese ZIP Datei enthält alle notwendigen SQL Dateien zum Erstellen und Testen Ihrer Datenbank, siehe Aufgabe 6. Zum Testen ihrer Dateien stellen wir Ihnen einen PostgreSQL Server (Version 11.5) zur Verfügung. Sie können sich dazu per SSH auf `bordo.dbai.tuwien.ac.at` verbinden und ihn mittels `psql` nutzen. Die Zugangsdaten finden Sie auf TUWEL unter `https://tuwel.tuwien.ac.at/mod/assign/view.php?id=1402737`. Beachten Sie die Erklärungen bei den einzelnen Aufgaben.

Wichtig! Stellen Sie sicher, dass die von Ihnen abgegebenen SQL Befehle auf dem Server (`bordo.dbai.tuwien.ac.at`) ausgeführt werden können. D.h. dass es sich um (syntaktisch) gütlige SQL Befehle handelt. Sollte dies nicht der Fall sein (sollte es z.B. Syntaxfehler geben beim Versuch die Files auszuführen), dann bekommen Sie auch **keine Punkte** für die entsprechende Datei.

Das Übungsblatt enthält 6 Aufgaben, auf welche Sie insgesamt 15 Punkte erhalten können.

Deadlines

bis 10.01.	**12:00 Uhr**	Upload der Abgabe über TUWEL
bis 10.01.	**12:00 Uhr**	Anmeldung zu einem Kontrollgespräch über TUWEL

Kontrollgespräch

Dieses Semester werden die Kontrollgespräche über Zoom durchgeführt. Zu Anfang Ihres Gesprächs müssen Sie Ihre Kamera einschalten und Ihren Studierendenausweis dem Prüfer zeigen, um Ihre Identität zu bestätigen. Anschließend findet das eigentliche Kontrollgespräch statt, in welchem nicht nur die Korrektheit Ihrer Lösung, sondern vor allem das Verständnis der Konzepte überprüft wird. Sie müssen daher bei Ihrem Kontrollgespräch in der Lage sein, nicht nur Ihre Beispiele zu erklären, sondern ebenfalls zeigen, dass Sie die in der Vorlesung behandelte Theorie zu diesen Beispielen ausreichend verstanden haben.

Die Bewertung Ihres Übungsblattes basiert zum Überwiegenden Teil auf Ihrer Leistung beim Kontrollgespräch! Daher ist es im Extremfall durchaus möglich, dass eine korrekte Abgabe mit 0 Punkten bewertet wird. Insbesondere werden nicht selbstständig gelöste Abgaben immer mit 0 Punkten bewertet!

Hinweis: Noch einmal der Hinweis, dass Ihre Lösung beim Kontrollgespräch auf dem Server ausführbar sein muss. Testen Sie Ihre Lösung daher bevor Sie sie abgeben auf `bordo.dbai.tuwien.ac.at`. Es spielt keine Rolle, ob Sie beim Kontrollgespräch auftretende Syntaxfehler sofort beheben und erklären können – sollte Ihre Abgabe nicht lauffähig sein wird es **keine Punkte für die entsprechende Datei** geben.

Erscheinen Sie bitte pünktlich zu Ihrem Zoom Kontrollgespräch und halten Sie bitte Ihren Studierendenausweis zum Zoom Kontrollgespräch bereit. Ein Kontrollgespräch ohne Ausweis ist nicht möglich.

Aufgaben: Datenbank mit PostgreSQL

Die folgenden Aufgaben basieren auf der Datenbank die Sie bereits aus Übungsblatt 1 kennen. Wir geben hier nochmals das Relationenschema für Sie an. Sie finden das entsprechende EER-Diagramm in Abbildung 1 auf der letzten Seite der Angabe.

Dividende	Datum, Kurs, Ertrag, WID: *Wertpapier.WID*, DName: *Wertpapier.DName*, BName: *Wertpapier.BName*
Wertpapier	WID, Isin, Name, DName: *Depot.DName*, BName: *Depot.BName*
Fonds	TFC, WID: *Wertpapier.WID*, DName: *Wertpapier.DName*, BName: *Wertpapier.BName*
Aktien	WID: *Wertpapier.WID*, DName: *Wertpapier.DName*, BName: *Wertpapier.BName*
Anleihen	WID: *Wertpapier.WID*, DName: *Wertpapier.DName*, BName: *Wertpapier.BName*
Gewichtung	GID, Gewicht, FWID: *Fonds.WID*, FDName: *Fonds.DName*, FBName: *Fonds.BName*, WWID: *Wertpapier.WID*, WDName: *Wertpapier.DName*, WBName: *Wertpapier.BName*
Depot	DName, Kosten, BName: *Broker.BName*
Broker	BName, KEY, URL, BrokerageGeb
unterstützt	BName: *Broker.BName*, Börse: *Börse.Name*
Börse	Name, Tz, BörsenGeb
Kauforder	OrderId: *Orders.OId*
Verkauforder	OrderId: *Orders.OId*, KEST
Orders	Time, OId, Geb, Preis, Nr, Börse: *Börse.Name*, WID: *Wertpapier.WID*, DName: *Wertpapier.DName*, BName: *Wertpapier.BName*

Aufgabe 1 (Erstellen von Sequenzen & Tabellen) [2 Punkte]

Erstellen Sie eine Datei `create.sql`, in welcher die nötigen CREATE-Befehle gespeichert werden, um das gegebene Relationenschema mittels SQL zu realisieren.

Beachten Sie dabei folgendes:

(a) Ändern Sie das Relationenschema, um auch folgende Sachverhalte korrekt darzustellen:

- Jede Börse hat eine eigene Börsenaktie.
- Jeder Broker verwaltet sein eigenes Brokerdepot.

Diese Änderungen können Sie direkt in den CREATE Statements abbilden.

(b) Realisieren Sie die fortlaufende Nummerierung des Attributs `WID` der Relation `Wertpapier` mit Hilfe einer Sequence. Die Sequence soll bei 0 beginnen und in Zweierschritten erhöht werden.

(c) Realisieren Sie die fortlaufende Nummerierung des Primärschlüssel-Attributs `GID` in der Tabelle `Gewichtung` mit Hilfe einer Sequence. Die Sequence soll bei 5.000 beginnen und in Zwölferschritten erhöht werden.

(d) Das Attribut `BName` in der Tabelle `Broker` kann nur die Werte `'Interactive Brokers'`, `'Fidelity'` und `'Zacks Trade'` annehmen. Erstellen Sie dazu einen ENUM Typ.

(e) Repräsentieren Sie die Gewichte von einer Gewichtung und die TFCs von Fonds als NUMERIC mit zwei Nachkommastellen (dabei werden TFCs in Euro gemessen).

(f) Sollten zwischen zwei Tabellen zyklische FOREIGN KEY Beziehungen existieren, so achten Sie darauf, dass eine Überprüfung dieser FOREIGN KEYs erst zum Zeitpunkt eines COMMITs stattfindet.

(g) Verwenden Sie keine Umlaute für Bezeichnungen von Relationen, Attributen, etc.

(h) Stellen Sie die folgenden Sachverhalte durch geeignete Bedingungen sicher:

- Das Datum einer Dividende muss größer sein als der 1. Jänner 1940.
- Eine international securities identification number eines Wertpapiers (repräsentiert durch das "Isin" Attribut) besteht aus genau 12 Symbolen, die mit zwei Buchstaben beginnt, gefolgt von 10 alphanumerischen Zeichen.
- Das Gewicht einer Gewichtung muss zwischen 0 und 1 liegen (exklusive 0 und inklusive 1).
- Ein Fonds kann ein Wertpapier nicht mehrmals enthalten.

(i) Treffen Sie für alle fehlenden Angaben (z.B.: Typen von Attributen) plausible Annahmen. Vermeiden Sie NULL Werte in den Tabellen, d.h. alle Attribute müssen angegeben werden.

(j) Sie müssen sich nicht um die min/max Notationen aus dem EER-Diagramm in Abbildung 1 sorgen.

Aufgabe 2 (Einfügen von Testdaten) [1 Punkt]

Erstellen Sie eine weitere Datei `insert.sql`, welche die INSERT-Befehle für die Testdaten der in Punkt 2 erstellten Tabellen enthält. Jede Tabelle soll zumindest drei Zeilen enthalten. Sie dürfen die Wahl der Namen, Bezeichnungen etc. so einfach wie möglich gestalten, d.h. Sie müssen nicht "real existierende" Aktien, Anleihen, Fonds, etc. wählen. Stattdessen können Sie auch einfach "Aktie 1", "Aktie 2", "Anleihe 1", "Anleihe 2" etc. verwenden. Sie können zum Anlegen geeigneter Relationen die in Aufgaben 4 angelegten Trigger und Procedures verwenden.

Aufgabe 3 (SQL Abfragen) [4 Punkte]

Erstellen Sie eine Datei `queries.sql`, welche den Code für folgende Views enthält.

(a) Erstellen Sie eine View `NumberOfExpensiveOrders`, welche die Anzahl der teueren Orders pro Börse ausgibt. Die Anzahl der teueren Orders einer Börse ist definiert als die Anzahl aller Orders die innerhalb der Börse getätigt wurden und deren Preis über 1.000.000€ beträgt.

(b) Erstellen Sie eine View `AllHolds`, welche für jedes in einem Fonds X enthaltene Wertpapier Z, ein Tupel $(X_S, X_D, X_B, Z_S, Z_D, Z_B)$ beinhält, sodass X_S, X_D, X_B die Primärschlüsselatribute vom Fonds X und Z_S, Z_D, Z_B die Primärschlüsselatribute vom Wertpapier Z sind. Da ein Wertpapier selbst wiederum ein Fonds sein kann, soll der View `AllHolds` in folgender Art erweitert werden: Wenn laut `AllHolds` ein Fonds A ein Wertpapier B enthält und B ein Wertpapier C enthält, dann soll auch A enthält C im View dargestellt werden. Beachten Sie, dass diese Definition beliebig lange Verkettungen von Wertpapieren miteinbezieht. Sei etwa im obigen Beispiel zusätzlich X ein Fonds der A enthält, dann enthält X auch die Wertpapiere B und C.

(c) Erstellen Sie eine View `WeightingOfHolds`, welche all jene Tupel $(X_S, X_D, X_B, Z_S, Z_D, Z_B, G, S)$ enthält, sodass ein Fonds X (definiert über Primärschlüsselattribute X_S, X_D, X_B) über S "Schritte" Wertpapier Z (definiert über Primärschlüsselattribute Z_S, Z_D, Z_B) enthält und sodass G das kumulierte Produkt der Gewichte ist. Als Beispiel: Angenommen X, Y und Z sind Wertpapiere und X enthält Y mit Gewicht $G_Y = 0,5$ und Y enthält Z mit Gewicht $G_Z = 0,8$, dann hält X auch Z über zwei "Schritte" von Fonds mit Gewicht $G_Y * G_Z = 0,5 * 0,8 = 0,4$. Beachten Sie, dass in diesem Beispiel X auch Y über einen "Schritte" mit Gewicht $0,5$ und Y auch Z über einen "Schritte" mit Gewicht $0,8$ enthält.

Beachten Sie weiters, dass Sie das Produkt der Gewichte im SELECT Ausdruck in den Typen den Sie für das `Gewicht` Attribut verwendet haben umwandeln müssen. Ein kurzes Tutorial zum Thema Typumwandlung in PostgreSQL finden Sie unter `https://www.postgresqltutorial.com/postgresql-cast/`.

Hinweis: *Sie benötigen für die letzten beiden Teilaufgaben jeweils eine rekursive Abfrage. Wir empfehlen, dass Sie entsprechende Einträge in Ihrer* `insert.sql` *Datei anlegen, um diese Abfragen sinnvoll auswerten zu können.*

Aufgabe 4 (Erstellen und Testen von Trigger) [6 Punkte]

Erstellen Sie eine Datei `plpgsql.sql`, welche den Code für die folgenden Trigger und Funktionen enthält.

(a) Erstellen Sie einen Trigger, der beim Anlegen einer `Kauforder` sicherstellt, dass die `KEST` der `Kauforder` gleich $27,5\%$ des `Preises` der zugehörigen `Order` ist. Falls die `KEST` der hinzuzufügenden `Kauforder` gleich 0 ist, so soll die `KEST` (wie im vorigen Satz beschrieben) berechnet werden und die neue `Kauforder`, zusammen mit der berechneten `KEST` der `Kauforder` Tabelle hinzugefügt werden. Falls die `KEST` der hinzuzufügenden `Kauforder` weder gleich 0 noch $27,5\%$ vom `Preis` der zugehörigen `Order` ist, so soll die `Kauforder` nicht angelegt werden.

(b) Erstellen Sie einen Trigger, der folgendes Verhalten bei einer Änderung in der Tabelle `Börse` implementiert:

- Wenn die `BörsenGeb` für ein Tupel verändert wurde, dann soll das Attribut `BrokerageGeb` der Broker, welche diese Börse unterstützen, um die Differenz des neuen Werts vom bisherigen Wert der `BörsenGeb` angepasst werden. Weiters, soll in diesem Fall per `RAISE NOTICE` die neue `BrokerageGeb` als Teil einer Nachricht ausgegeben werden.
- Wenn die `BörsenGeb` einer Börse per `UPDATE` auf den selben Wert gesetzt wurde der bereits gespeichert war, dann soll dazu mittels `RAISE` eine Warnung ausgegeben werden.

  Hinweis: Mehr Informationen zur Ausgabe von Meldungen mittels `RAISE` finden Sie in der Online Dokumentation[1].

(c) Erstellen Sie einen Trigger, der bei einer Änderung der Total Fund Cost (`TFC`) eines Fonds folgendes Verhalten implementiert:

- Wenn die `TFC` eines Fonds F_c um einen bestimmten Betrag verändert wird, dann sollen auch die `TFC`s von allen Fonds F_p die F_c enthalten, um den selben Betrag verändert werden.

Bedenken Sie, dass Sie sich nicht um die rekursiven Abhängigkeiten zwischen den Fonds kümmern müssen, sondern dass Trigger wieder Trigger ausführen.

(d) Erstellen Sie eine Procedure `CreateSecurities` die automatisch neue Wertpapiere anlegt. Die Procedure hat drei Parameter: Eine Zahl welche die Anzahl der zu erstellenden `Wertpapiere` angibt, einen `DepotNamen` und die `Kosten` des Depots in welchem die Wertpapiere gelagert werden sollen.

Beachten Sie folgendes:

- Es muss mindestens ein Wertpapier angelegt werden. Falls der Parameter für die Anzahl an Wertpapieren kleiner als 1 ist, dann geben Sie eine entsprechende Fehlermeldung aus.
- Das durch die Parameter festgelegte Depot soll angelegt werden. Der `DName` und die `Kosten` des neuen Depots sollen den Eingabeparametern entnommen werden. Sie können den Broker, der das Depot verwaltet, nach belieben wählen. Beachten Sie aber, dass der Broker bereits in der `Broker` Tabelle exisitieren muss.

[1] `https://www.postgresql.org/docs/11/static/plpgsql-errors-and-messages.html`

- Weiters, soll die angegebene Anzahl an Wertpapieren wie folgt erzeugt werden:
  - Es soll in Ihrer Prozedur ein Zähler vorhanden sein, der angibt wie viele Wertpapiere bisher erstellt wurden. Je nach der Teilbarkeit des gegenwärtigen Stands des Zählers durch 3 soll ein Fonds, eine Aktie oder eine Anleihe erstellt werden.

    Als Beispiel soll bei Zählerstand 0 ein Fonds, bei 1 eine Aktie, bei 2 eine Anleihe, bei 3 wieder ein Fonds und so weiter erstellt werden. Wir empfehlen, dass Sie dafür den Modulo Operator verwenden. Vergessen Sie nicht die entsprechenden Einträge in der `Fonds`, `Aktien` und `Depot` Tabelle zu generieren. Sie können die TFC von Fonds beliebig wählen.
  - Wählen Sie für die `Namen` der Wertpapiere jeweils "Fonds aus $DepotName", "Aktie aus $DepotName" und "Anleihe aus $DepotName", wobei $DepotName für den Namen des erstellten Depots steht, welcher als Parameter zur Verfügung gestellt wurde.
  - Sie können die `WID` und `Isin` der zu erstellenden Wertpapiere frei wählen. Stellen Sie aber sicher, dass in jedem Schritt eine verfügbare `WID` verwendet wird (damit keine Primärschlüsselbedingung verletzt wird).
  - Zum Abschluss sollen Einträge erzeugt werden, sodass die erstellten Wertpapiere in dem initial angelegten Depot liegen.

Nota bene: Für das angelegte Depots gilt, dass Ihre Funktion mit einer Fehlermeldung abbrechen soll, falls bereits Einträge in der Datenbank mit demselben Depotnamen existieren. Im Fall eines Abbruchs soll die Prozedur alle bisherigen Änderungen rückgängig machen.

Aufgabe 5 (Löschen der angelegten Objekte) [1 Punkt]

Erstellen Sie eine Datei `drop.sql`, welche die nötigen DROP-Befehle enthält, um alle erzeugten Datenbankobjekte wieder zu löschen. Das Schlüsselwort CASCADE darf dabei NICHT verwendet werden.

Aufgabe 6 (Testen der Datenbank und erstellen des Abgabearchivs) [1 Punkt]

(a) Erstellen Sie eine Datei `test.sql`. Dazu überlegen Sie sich eine sinnvolle Testabdeckung für die in Aufgabe 1 gegebenen Bedingungen, für die in Aufgabe 3 erstellten Views und für die PL/pgSQL-Programmteile in Aufgabe 4. Es sollen möglichst alle Fälle, positive als auch negative, abgedeckt werden, z.B. Hinzufügen eines Eintrags in die Tabelle `Dividende` mit einem Datum vor und nach dem 1. Jänner 1940.

(b) Stellen Sie eine Listing-Datei mit dem Namen `listing.txt` bereit, die Sie bei der Ausführung der SQL-Dateien erzeugt haben. Diese Erstellen Sie am Besten auf unserem Übungsserver `bordo.dbai.tuwien.ac.at`. Dort starten Sie `psql`. Mittels "`\o listing.txt`" lässt sich die Ausgabe in die Datei `listing.txt` umleiten. Dann führen Sie die Dateien (sofern vorhanden) in dieser Reihenfolge mittels "`\i xxx.sql`" aus:

(1) `create.sql`

(2) `plpgsql.sql`

(3) `insert.sql`

(4) `queries.sql`

(5) `test.sql`

(6) `drop.sql`

Erstellen Sie aus diesen und der `listing.txt` Datei ein ZIP-Archiv `blatt4.zip` und laden dieses in TUWEL hoch.

EER-Diagramm

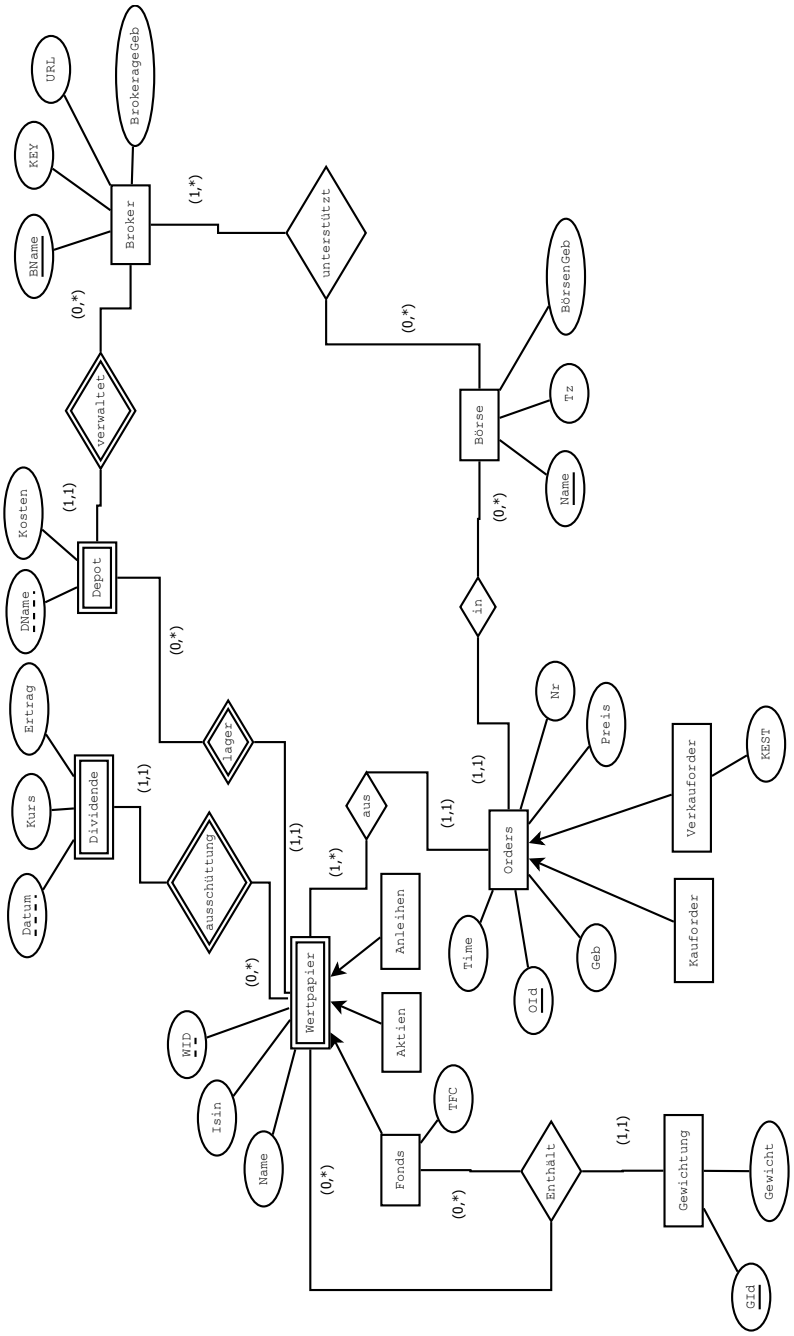

Abbildung 1: EER-Diagramm zu diesem Übungszettel