4. Übungsblatt (WS 2018)

3.0 VU Datenmodellierung 2 / 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 9.6) zur Verfügung. Sie können sich dazu per SSH auf `bordo.dbai.tuwien.ac.at` verbinden und ihn mittels `psql` nutzen. Die Zugangsdaten erhalten Sie von uns in einem E-Mail. 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) so hat dies einen empfindlichen Punkteabzug zur Folge.

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

Deadlines

bis 7.1.	**12:00 Uhr**	Upload der Abgabe über TUWEL
bis 7.1.	**12:00 Uhr**	Anmeldung zu einem Kontrollgespräch

Kontrollgespräch

Im Rahmen des Kontrollgespräches wird nicht nur die Korrektheit, sondern vor allem das Verständnis der Konzepte überprüft. Durch die Übung sollen sowohl Ihre praktische Problemlösungskompetenz als auch das theoretische Wissen über Datenbanksysteme gefördert werden. 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. Dies soll Ihnen die Vorbereitung für die Prüfung erleichtern und so können Sie Ihr Wissen während der Kontrollgespräche selbst testen und gegebenenfalls vertiefen.

Die Bewertung Ihres Übungsblattes basiert zum Überwiegenden Teil auf Ihrer Leistung beim Kontrollgespräch! Es daher ist im Extremfall durchaus möglich, dass eine korrekte Abgabe mit 0 Punkten bewertet wird. Insbesonders 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 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 hat dies einen Punkteabzug zur Folge.

Erscheinen Sie in Ihrem eigenen Interesse bitte pünktlich zum Kontrollgespräch, da andernfalls nicht garantiert werden kann, dass Ihre gesamte Lösung in der verbleibenden Zeit beurteilt werden kann.

Bringen Sie bitte Ihren Studentenausweis zum Kontrollgespräch mit. Ein Kontrollgespräch ohne Ausweis ist nicht möglich.

Tutorensprechstunden (freiwillig)

Rund eine Woche vor der Abgabedeadline bieten die TutorInnen Sprechstunden an. Falls Sie Probleme mit oder Fragen zum Stoff des Übungsblattes haben, es Verständnisprobleme mit den Beispielen oder technische Fragen gibt, kommen Sie bitte einfach vorbei. Die TutorInnen beantworten Ihnen gerne Ihre Fragen zum Stoff, oder helfen Ihnen bei Problemen weiter.

Ziel der Sprechstunden ist es, Ihnen beim **Verständnis des Stoffs** zu helfen, nicht, das Übungsblatt für Sie zu rechnen, oder die eigenen Lösungen vorab korrigiert zu bekommen.

Die Teilnahme ist vollkommen freiwillig — Termine und Orte der Tutorensprechstunden finden Sie in TUWEL.

Weitere Fragen – TUWEL Forum

Sie können darüber hinaus das TUWEL Forum verwenden, sollten Sie inhaltliche oder organisatorische Fragen haben.

Aufgaben: Datenbank mit PostgreSQL

Die folgenden Aufgaben basieren auf der Datenbank die Sie bereits aus Aufgabenblatt 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.

```
Lied        (titel, sid, dauer)
Account     (email, pw, name)
Playlist    (pid, name, info, email: Account.email, datum )
Person      (vname, nname)
KuenstlerIn (vname: Person.vname, nname: Person.nname, foto)
SaengerIn   (vname: KuenstlerIn.vname, nname: KuenstlerIn.nname)
ProduzentIn (vname: Person.vname, nname: Person.nname, url)
Album       (name, jahr,
             ProdVName: ProduzentIn.vname, ProdNName: ProduzentIn.nname,
             SaengVName: SaengerIn.vname, SaengNName: SaengerIn.nname)
CD          (name: Album.name, nr, type, label)
Track       (name: CD.name, nr: CD.nr, type: CD.type,)
             LiedTitel: Lied.titel, sid: Lied.sid, tid, titel)
hoert       (sid: Lied.sid, titel: Lied.titel, email: Account.email, anzahl)
hinzugefuegt(sid: Lied.sid, titel: Lied.titel, email: Account.email,
             pid: Playlist.pid, name: Playlist.name, datum)
teil_von    (ElternPid: Playlist.pid, ElternName: Playlist.name,
             KindPid: Playlist.pid, KindName: Playlist.name)
singt       (vname: SaengerIn.vname, nname: SaengerIn.nname,
             titel: Lied.titel, sid: Lied.sid)
hat_Rechte  (vname: ProduzentIn.vname, nname: ProduzentIn.nname,
             name: Album.name)
alias_von   (AliasVName: Person.vname, AliasNName: Person.nname,
             vname: Person.vname, nname: Person.nname)
```

Aufgabe 1 (Erstellen von Sequenzen & Tabellen) *[6 Punkte]*

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

Beachten Sie dabei folgendes:

(a) Nehmen Sie Änderungen am Relationenschema vor, damit auch folgende Sachverhalte korrekt dargestellt werden:

- Jeder Account hat eine Lieblingsplaylist.
- Die Relation `hoert` hat ein neues Attribut `zuletzt` die angibt wann das Lied vom jeweiligen Account zuletzt gehört wurde.

Diese Änderungen können Sie direkt in den CREATE Statements abbilden.

(b) Realisieren Sie die fortlaufende Nummerierung des Attributs `sid` der Relation `Lied` mit Hilfe einer Sequence. Die Sequence soll bei 1 beginnen und in Einerschritten erhöht werden.

(c) Realisieren Sie die fortlaufende Nummerierung des Primärschlüssel-Attributs `pid` in der Tabelle `Playlist` mit Hilfe einer Sequence. Die Sequence soll bei 10000 beginnen und in Hunderterschritten erhöht werden.

(d) Das Attribut `type` in der Tabelle `CD` kann nur die Werte `'Single'`, `'EP'` und `'Album'` annehmen. Erstellen Sie dazu einen ENUM Typ.

(e) Repräsentieren Sie die Dauer von Liedern in Sekunden als NUMERIC mit einer Nachkommastellen.

(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:

- Ein Lied muss zumindest 1 Sekunde lang sein.
- Ein Accountnamen muss zumindest 3 Zeichen lang sein.
- Das Jahr eines Albums muss zwischen (inklusive) 1800 und 3030 liegen.
- Ein Lied kann von einem Account nicht mehrmals in die selbe Playlist hinzugefügt werden.

(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.

Aufgabe 2 (Einfügen von Testdaten) *[1 Punkte]*

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" Alben, SängerInnen, ProduzentInnen, etc. wählen. Stattdessen können Sie auch einfach "Album 1", "Album 2", "ProduzentIn 1", "ProduzentIn 2" etc. verwenden. Sie können zum Anlegen geeigneter Relationen die in Aufgaben 4 angelegten Trigger und Procedures verwenden.

Aufgabe 3 (SQL Abfragen) *[2 Punkte]*

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

(a) Erstellen Sie eine View `CDDauer` welche die Gesamtdauer einer CD ermittelt. Die Gesamtdauer einer CD ist die Summe der Dauer aller Tracks die auf der CD vorkommen.

(b) Erstellen sie eine View `AllAlias` welche die Alias Relation so erweitert, dass wenn A alias von B ist und B alias von C, dann ist auch A alias von C. Beachten Sie, dass diese Definition beliebig lange Verkettungen von Aliasen miteinbezieht. (Sei etwa im obigen Beispiel zusätzlich X ein alias von A, dann ist X auch alias von B und C.) *(Hinweis: Sie benötigen hierfür eine rekursive Abfrage. Achten Sie darauf, dass Ihre Datenbank keine zyklischen Aliase enthält.)*

Aufgabe 4 (Erstellen und Testen von Trigger) *[4 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 `hinzugefuegt` Beziehung sicherstellt, dass das `datum` in hinzugefuegt nicht vor dem `datum` liegt, an dem die Playlist erstellt wurde. Falls das `datum` des neuen Tupels von hinzugefuegt diese Bedingung verletzt, soll stattdessen das Datum an dem die Playlist erstellt wurde gesetzt werden. (Beispiel, es gibt die Playlist **P** seit dem 23.10.2018. Es wird ein Eintrag erzeugt der sagt, dass ein Lied am 1.4.2017 zu **P** hinzugefuegt wurde. Das Datum soll beim Einfügen mit dem Erstellungsdatum von **P**, dem 23.10.2018, ersetzt werden.)

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

- Wenn `anzahl` für ein Tupel erhöht wurde, dann soll `zuletzt` auf `NOW()` gesetzt werden. Weiters soll in diesem Fall per `RAISE NOTICE` die Zeit die gesetzt wurde als Teil einer Nachricht ausgegeben werden.
- Wenn `anzahl` 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: `https://www.postgresql.org/docs/9.6/static/plpgsql-errors-and-messages.html`.

(c) Erstellen Sie einen Trigger, der beim hinzufügen eines Lieds L in eine Playlist P folgendes Verhalten implementiert:

- Wenn P Kind anderer Playlists ist, dann soll L auch in alle Eltern-Playlists hinzugefügt werden.
- Wenn L bereits in eine Eltern Playlist hinzugefügt wurde (unabhängig von welchem Account), dann soll das Lied **nicht** nochmal hinzugefügt werden.
- Die Felder `datum` und `email` für die neue Tupel werden vom ursprünglichen Eintrag übernommen.

Bedenken Sie, dass Sie sich nicht um die rekursiven Abhängigkeiten zwischen den Playlists kümmern müssen, sondern dass Trigger sehr wohl wieder Trigger ausführen, usw.

(d) Erstellen Sie eine Procedure `CreateCD` die automatisch eine CD zusammenstellt. Die Procedure hat folgende Parameter: Eine Anzahl `songs` an Tracks auf der CD, Vor- und Nachname einer SaengerIn sowie Vor- und Nachnahme einer ProduzentIn.

Beachten Sie folgendes:

- Eine CD muss mindestens 2 Lieder haben. Hat sie weniger als 5 Lieder ist der `type` *EP*, ansonsten ist der `type` *Album*.
- Der Name der CD lautet "Best of $SAENGERIN"wobei hier der entsprechende Name der SängerIn aus den Parametern zu wählen ist. Wenn bereits ein Album mit diesem Namen existiert brechen Sie die Funktion mit einer entsprechenden Fehlermeldung ab.
- Die CD Nummer können Sie frei wählen. Stellen Sie jedoch sicher, dass immer eine verfügbare Nummer gewählt wird.
- Für die Tracks der CD werden dem `songs` Parameter entsprechend viele Titel der gegebenen SängerIn zufällig gewählt. Falls nicht genug Lieder existieren soll die Procedure mit einer Fehlermeldung abgebrochen werden. Stellen Sie sicher, dass in diesem Fall keine Änderungen an der Datenbank vorgenommen werden.
- Dem Schema entsprechend, benötigt jede CD ein zugehöriges Album (unabhängig vom `type` Attribut). ProduzentIn und SaengerIn des Albums können Sie aus den Parametern übernehmen. Für das Attribut `jahr` wählen Sie das aktuelle Jahr (Zur Zeit des Funktionsaufrufs).

Aufgabe 5 (Löschen der angelegten Objekte) *[1 Punkte]*

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 Punkte]*

(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 erstellen Views und für die PL/pgSQL-Programmteile in Aufgabe 4. Es sollen möglichst alle Fälle, auch positive, abgedeckt werden, z.B. Hinzufügen eines Lieds in eine Playlist mit erlaubtem und zu frühem datum.

(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) `queries.sql`

(3) `plpgsql.sql`

(4) `insert.sql`

(5) `test.sql`

(6) `drop.sql`

Fügen Sie alle Dateien zu einer ZIP-Datei `blatt4.zip` hinzu und laden Sie diese in TUWEL hoch.

EER-Diagramm

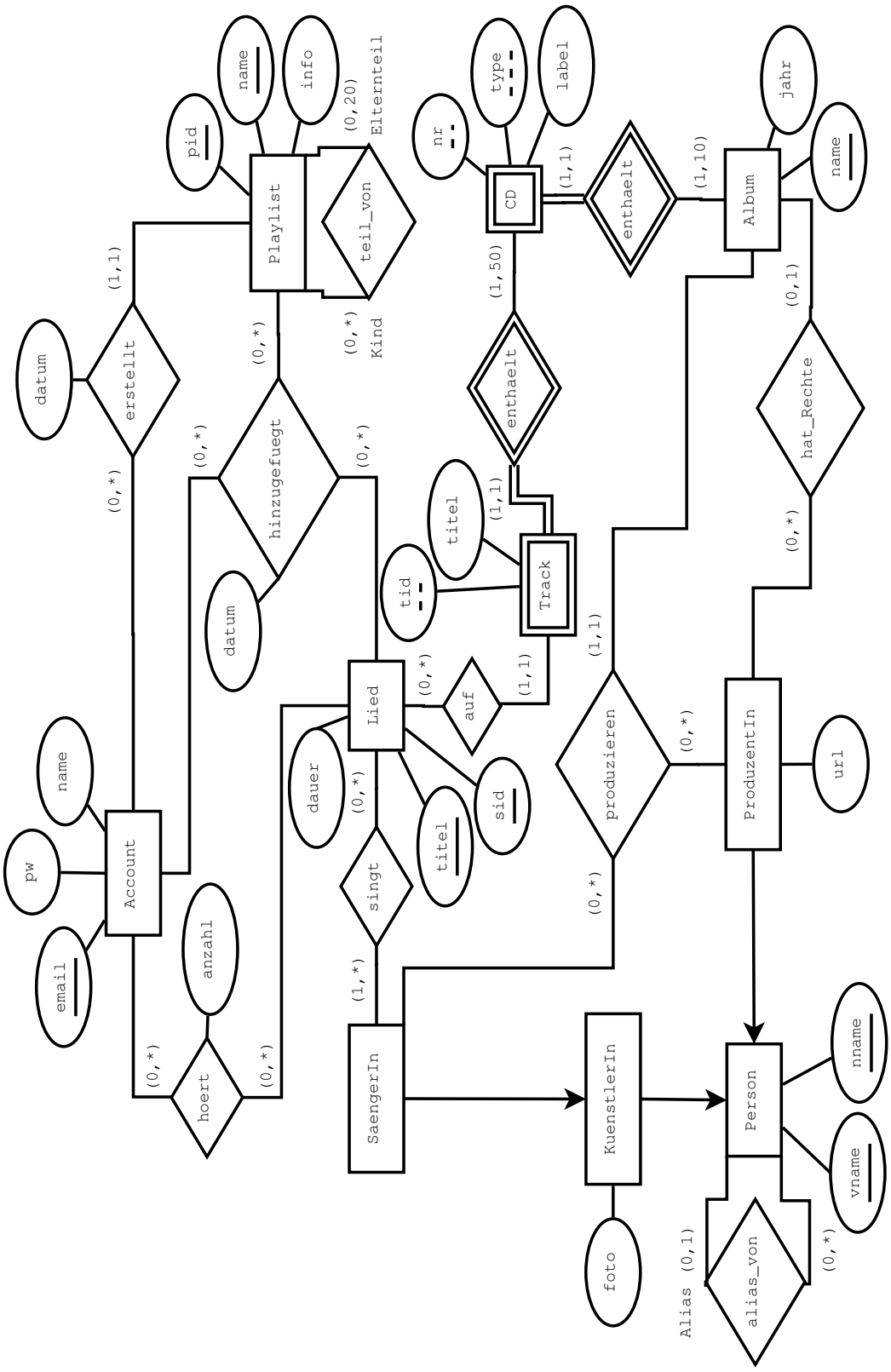

Abbildung 1: EER-Diagramm zu diesem Übungszettel