Praktikum 3

Lernziel

- Einblick in die serverseitige Programmierung.
- Kennenlernen der Nützlichkeit von Triggern in einem typischen Anwendungsgebiet.

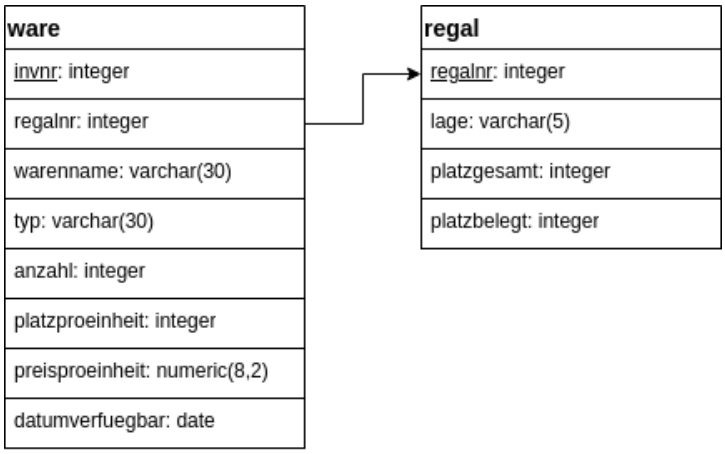
Vorbereitung

Informieren Sie sich über folgende Punkte:

1) Was ist ein Trigger und auf welche Weise wird er in PostgreSQL angelegt ([1], insbesondere die Abschnitte 39 und 43, sowie [2])?
2) Wie programmiert man in PL/pgSQL Stored Procedures, die von Triggern angestoßen werden (Quellen s.o.)?
3) Was sind Views und wie legt man sie an [3]?

Aufgabe 1

In einem Warenlager wird der Warenbestand mithilfe des folgenden Datenbankschemas verwaltet.



Erstellen Sie ein Skript mit den erforderlichen CREATE TABLE-Befehlen und ergänzen Sie das Datenbankschema um eine weitere Tabelle `warenhistorie`, in welcher Änderungen am Warenbestand dokumentiert werden. Für jeden Änderungsvorgang (UPDATE oder DELETE) auf der Tabelle `ware` sollen mit Hilfe eines Triggers in der neuen Tabelle `warenhistorie` jeweils die `invnr` der Ware, die zugehörige `regalnr`, der `warenname`, der alte `preis`, das alte `verfuegbar`-Datum, die durchgeführte `SQL-Operation` (`UPDATE oder DELETE`), der `User-Name` des Datenbank-Benutzers, der die Ware geändert hat, und der `Änderungszeitpunkt` festgehalten werden.

Wenn Sie den Warenbestand mithilfe von INSERT, UPDATE oder DELETE-Operationen auf `ware` ändern, soll das Attribut `platzbelegt` des betroffenen `regal`-Datensatzes entsprechend angepasst werden. Außerdem sollen folgende Geschäftsregeln mithilfe des Triggers überwacht werden:

Regel 1: Neue Ware kann nur dann in das angegebene Regal eingefügt werden, wenn ausreichend Platz vorhanden ist. Ansonsten soll das Hinzufügen unter Angabe einer Fehlermeldung unterbunden werden.

Regel 2: Wenn beim Hinzufügen neuer Ware kein Regal angegeben wird (d.h. das Feld `regalnr` ist leer!), wird das nächste Regal mit ausreichend freier Kapazität gesucht und die Ware in dieses eingefügt. Die Nummer des gewählten Regals soll auf der Console ausgegeben werden (Hinweis: RAISE NOTICE). Falls kein passendes gefunden wird, soll eine Fehlermeldung ausgegeben werden.

Regel 3: Es dürfen aus Gründen der Übersichtlichkeit nicht mehr als vier **unterschiedliche** Waren in das gleiche Regal eingelagert werden. Operationen, die diese Regel verletzen, sollen unter Angabe einer Fehlermeldung unterbunden werden.

Regel 4: Die `Anzahl` einer Bestandsware darf nur so weit erhöht werden, dass durch die Erhöhung der verfügbare Platz im Regal nicht überschritten wird. Die `Anzahl` darf auf 0 gesetzt werden, aber nicht negativ werden. Außerdem darf die Ware nicht in ein anderes Regal umsortiert werden. Ansonsten soll die Änderung unter Angabe einer Fehlermeldung ebenfalls unterbunden werden.

Regel 5: Das Verfügbarkeitsdatum einer Bestandsware kann nicht auf ein früheres Datum geändert werden. In dem Fall bleibt das Verfügbarkeitsdatum auf dem ursprünglichen Datum unter Ausgabe einer entsprechenden Meldung auf der Console. Wenn sich auch sonst kein anderes Feld in dem Datensatz geändert hat, soll kein neuer Eintrag in der Warenhistorie erzeugt werden.
Das Ändern auf ein späteres Datum ist hingegen erlaubt.

Regel 6: Wenn eine Ware aus dem Bestand genommen (= aus der Datenbank entfernt) wird, soll der zugehörige Platz im Regal freigegeben werden. Zusätzlich soll ein Eintrag in der Warenhistorie erstellt werden.

Test

In den Skripten `block1.sql` und `block2.sql` finden Sie INSERT, UPDATE und DELETE Operationen, mit denen Sie Ihren Trigger testen können. Die erwarteten Tabelleninhalte und Ausgaben sind in den Skripten jeweils angegeben.

`block1.sql` erstellt die initiale Datenbasis und kann als Ganzes ausgeführt werden.

`block2.sql` enthält insgesamt 14 Testfälle, mit denen Sie die Regeln und das Füllen der warenhistorie auf verschiedene Weise testen können. Die Befehle aus `block2.sql` sollen einzeln nacheinander zum Testen ausgeführt werden.

Die erwarteten Tabelleninhalte und Ausgaben sind in den Skripten jeweils angegeben. Vergleichen Sie, ob Ihr Trigger die gewünschten Resultate liefert.

Hinweise

- Als Primary Key Ihrer Tabelle `warenhistorie` sollten Sie ein künstliches Schlüsselattribut vom Typ `SERIAL` nehmen. Dann wird für jeden neuen Eintrag in dieser Tabelle automatisch ein eindeutiger Schlüsselwert generiert.
- Das aktuelle Datum und den Namen des aktuellen Datenbank-Benutzers erhalten Sie über die eingebauten Funktionen CURRENT_DATE und CURRENT_USER (benötigen Sie, wenn Sie einen Eintrag in die Tabelle `warenhistorie` einfügen);
- Innerhalb einer Trigger-Funktion können Sie u.a. auf folgende eingebaute Variablen zurückgreifen:

  - NEW: enthält Datensatz mit Werten nach Durchführung der Änderung bzw. Einfügeoperation;
  - OLD: enthält Datensatz mit den alten Werten vor der Durchführung der Änderung;
  - TG_OP: Art der Operation, die den Trigger ausgelöst hat (`INSERT, UPDATE, DELETE`)
- Sie können die Geschäftslogik innerhalb eines einzigen Triggers realisieren oder auf mehrere Trigger aufteilen. Wenn Sie mehrere Trigger für das gleiche Event (z.B. ein INSERT auf der Tabelle ware) definieren, werden die Trigger in der alphabetischen Reihenfolge der Trigger-Namen aufgerufen (also z.B. trigger_A vor trigger_B). Hier der genaue Wortlaut aus der Dokumentation:
  If more than one trigger is defined for the same event on the same relation, the triggers will be fired in alphabetical order by trigger name. In the case of BEFORE and INSTEAD OF triggers, the possibly-modified row returned by each trigger becomes the input to the next trigger. If any BEFORE or INSTEAD OF trigger returns NULL, the operation is abandoned for that row and subsequent triggers are not fired (for that row).

Aufgabe 2

Legen Sie SQL-Statements vor, mit dem im Rahmen Ihres Lösungsansatzes folgende Anfragen beantwortet werden können:

1) Geben Sie pro `ware` die Inventarnummer, den Namen der Ware, das Regal und die Anzahl der Änderungen (UPDATE) für diese Ware aus.
2) Definieren Sie einen View `warenkomplett` mit den Attributen (`invnr, regalnr, warenname, preisproeinheit, datumverfuegbar`), der die aktuellen und früheren Preise aller noch vorhandenen Waren mit den jeweiligen `datumverfuegbar`-Daten enthält. (Hinweis: UNION)
3) Schreiben Sie eine Stored Procedure `getprice(int, date)`, die unter Verwendung des zuvor definierten Views den Preis einer gegebenen Ware (1. Parameter: Inventarnummer) zu einem bestimmten Zeitpunkt (2. Parameter) ermittelt. Testen Sie die Funktion mit folgendem Aufruf:
   ```
   SELECT getprice(2, to_date('01.06.2024', 'DD.MM.YYYY'));
   ```
   Erwartetes Ergebnis: 1.50

Literatur

[1] PostgreSQL 15 Documentation, Part V. Server Programming. Online verfügbar unter: https://www.postgresql.org/docs/15/server-programming.html

[2] Kapitel 5 der DBS-Vorlesung.

[3] Kapitel 6 der DBS-Vorlesung.