JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Protokoll INSY Stored Routines

Stored Routines MySQL

INSY

5bHITT 2016/17

Philipp Kogler

Begonnen am 4. Dezember 2016

Betreuer: Prof. Martinidez

Beendet am 4. Dezember 2016

Inhaltsverzeichnis

1 Einführung

Diese Übung soll einen Einblick in das Themengebiet der Stored Routines in MySQL bieten. Hierbei sollen alle bereits durchgeführten Übungen, welche mithilfe von PostgreSQL umgesetzt wurden, für MySQL umgeschrieben und angepasst werden.

1.1 Ziele

Das Ziel dieser Übung ist die Umsetzung und Anpassung aller Aufgaben im Bereich Stored Routines an das DBMS MySQL

Zusaätzlich soll ein Protokoll erstellt werden, welches diesen Vorgang protokolliert und entsprechende Merkmale hervorhebt

1.2 Voraussetzungen

- Grundlagen von MySQL
- Grundlagen von psql
- Stored Routines
- Grundlagen sql

1.3 Aufgabenstellung

Unter Verwendung von MySQL sollen alle durchgeführten Übungen als stored Procedures bzw. Functions umgeschrieben werden, sodass diese die geforderte FUnktionalität sowie Anforderungen der jeweiligen Aufgabenstellung erfüllen.

2 Allgemein - Stored Routines MySQL

MySQL 5.1 kennt auch gespeicherte Routinen (Prozeduren und Funktionen). Eine gespeicherte Prozedur ist eine Menge von SQL-Anweisungen, die auf dem Server gespeichert werden kann. So mussen Clients nicht immer wieder die jeweiligen Einzelanweisungen ausfuhren, sondern konnen stattdessen die gespeicherte Prozedur aufrufen.

Stored Routines werden folgendermasen eingesetzt:

- Wenn mehrere Clientanwendungen in verschiedenen Sprachen geschrieben sind oder auf verschiedenen Plattformen laufen, aber dieselben Datenbankoperationen ausfuhren mussen.
- Wenn Sicherheit sehr wichtig ist. Banken verwenden zum Beispiel fur alle haufigen Operationen gespeicherte Prozeduren und Funktionen. Das gewahrleistet eine konsistente und sichere Umgebung sowie eine korrekte Protokollierung jeder einzelnen Operation. In einer solchen Umgebung haben Anwendungen und Benutzer keinen Direktzugriff auf die Datenbanktabellen, sondern konnen nur bestimmte gespeicherte Routinen ausfuhren.

Gespeicherte Routinen bieten eine bessere Leistung, da weniger Informationen zwischen Server und Client ubermittelt werden mussen. Der Nachteil ist der, dass die Belastung des Datenbankservers steigt, weil mehr Arbeit auf der Serverseite und weniger Arbeit auf der Seite des Clients (der Anwendungen) erledigt werden muss. Dies mussen Sie berucksichtigen, wenn viele Clientcomputer (wie beispielsweise Webserver) von nur einem oder sehr wenigen Datenbankservern bedient werden.

2.1 Stored Procedures

Stored Proceduren in MySQL unterscheiden sich von Stored Functions. Prozeduren können nur mittels *call procname* und nicht innerhalb einer *select* Anweisung aufgerufen werden.

Prinzipiell können Parameter bei Prozeduren als **IN** oder **OUT** Parameter definiert werden.

Prozeduren können nur Werte mittels Output Parameter an den Aufrufer zurückliefern.

2.1.1 Syntax

```
-- Syntax/Synopsis CREATE STORED PROCEDURE
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
    | LANGUAGE SQL
    | [NOT] DETERMINISTIC
    | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
    | SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement
```

2.2 Stored Functions

Stored Functions in MySQL unterscheiden sich von Stored Prozeduren. Functions können mittels *select* Anweisung aufgerufen werden.

Functions können grundsätzlich nur mithilfe eines *RETURNS* werte an den Aufrufer zurückliefern. Grundsätzlich ist es nicht möglich ein **set of** mit mysql functions zurückzuliefern.

Functionen können keine **FLUSH** Statements verwenden, wobei Prozeduren dies sehr wohl unterstützen.

Rekursive Funktionen werden von MySQL nicht unterstützt. Prozeduren sind standardgemäß nicht rekursiv können aber mittels Änderung von bestimmten System Variables angepasst werden.

2.2.1 Syntax

```
-- Syntax/Synopsis CREATE STORED PROCEDURE
CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
    | LANGUAGE SQL
    | [NOT] DETERMINISTIC
    | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
    | SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement
```


3 Vorbereitung

Um alle Anforderungen der Aufgabenstellung zu erfüllen müssen zuerst gewisse Vorbereitungen getroffen werden. Bevor die eigentlichen Prozeduren erstellt werden können muss zuerst die bestehende Datenbank von psql auf mysql migriert und umgeschrieben werden.

3.1 Anpassen des Create - Scripts

Prinzipiell versteht PostgreSQL sowie auch MySQL den standard SQL dialekt, jedoch gibt es sehr wohl Unterschiede, welche ein problemloses migrieren des Create - Scripts nicht erlauben.

Hierbei mussten zuerst alle psql spezifischen Befehle wie */c,* durch MySQL Befehle ersetzt werden. Ansonsten konnte das Create - Script 1:1 übernommen werden, um alle Daten sowie Tabellendefinitionen in MySQL zu importieren.

Anschließend wurde nur noch das Create Script ausgeführt und die Datenbank wurde entsprechend erstellt.

4 Ergebnisse

Alle Ergebnisse werden untersucht und in bezug auf Postgresql verglichen. Wobei letztendlich die execution Time sowie die Schwierigkeit der Umsetzung bewertet und analysiert wird.

4.1 AU01 - *preisTo99() & loescheRechnung()*

4.1.1 Umsetzung - *preisTo99()*

```
1 --  Alle Preise auf .99 aufrunden
2 DROP PROCEDURE IF EXISTS preisTo99;
3 delimiter //
4     CREATE PROCEDURE preisTo99()
5     BEGIN
6         UPDATE speise SET preis=ceil(
              preis)-0.01;
7     END //
8 delimiter ;
```

Listing 1: preisTo99() - MySQL

```
--  Alle Preise auf .99 aufrunden
--      ceil(preis) - 0.01
--      floor(preis) + 0.99
--      trunc
CREATE FUNCTION preisTo99() returns void
    as '
UPDATE speise set preis = ceil(preis) -
    0.01;
' language sql;
```

Listing 2: preisTo99() - PSQL

4.1.2 Umsetzung - *loescheRechnung()*

```
1 DROP PROCEDURE IF EXISTS loescheRchng;
2 delimiter //
3     CREATE PROCEDURE loescheRechnung()
4     BEGIN
5         DELETE from rechnung where rnr
              not in (
6         select rnr from bestellung
7         );
8     END //
9 delimiter ;
```

Listing 3: loescheRchng() - MySQL

```
--  Alle Rechnungen loeschen
--      ceil(preis) - 0.01
--      floor(preis) + 0.99
CREATE FUNCTION loescheRechnung()
    returns void as '
    DELETE from rechnung where rnr not
        in (
        select rnr from bestellung
    );
' language sql;
```

Listing 4: loescheRchng() - PSQL

4.1.3 Bemerkungen

Umsetzung konnte problemlos durchgeführt werden. Einzige Unterschiede lassen sich aus der Syntax der jeweiligen Functions ersehen. Außerdum kann eine Prozedur nur mittels call aufgerufen werden.

4.1.4 Screenshots

```
mysql> call preisTo99();
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM speise;
+-----+----------------------+-------+
| snr | bezeichnung          | preis |
+-----+----------------------+-------+
|   1 | Heisse Liebe         |  4.99 |
|   2 | Schoko Palatschinken |  4.99 |
|   3 | Pute gebacken        |  4.99 |
|   4 | Pute natur           |  4.99 |
|   5 | Puten-Cordon         |  4.99 |
|   6 | Menue fuer 2         | 16.99 |
|   7 | Menue fuer 3         | 20.99 |
|   8 | Menue fuer 4         | 24.99 |
+-----+----------------------+-------+
8 rows in set (0.00 sec)
```

```
mysql> call loescheRechnung();
Query OK, 0 rows affected (0.00 sec)

mysql> select * from rechnung limit 8;
+-----+------------+-------+----------+------+
| rnr | datum      | tisch | status   | knr  |
+-----+------------+-------+----------+------+
|   1 | 2013-03-07 |     1 | bezahlt  |    1 |
|   2 | 2013-03-07 |     2 | offen    |    2 |
|   3 | 2013-03-07 |     1 | gedruckt |    3 |
|   4 | 2013-03-07 |     1 | gedruckt |    1 |
|   5 | 2013-03-07 |     1 | bezahlt  |    1 |
|   6 | 2013-03-07 |     2 | offen    |    2 |
| 100 | 2016-12-04 |     1 | bezahlt  |    1 |
| 101 | 2016-12-04 |     1 | bezahlt  |    1 |
+-----+------------+-------+----------+------+
8 rows in set (0.00 sec)
```

4.2 AU02 - *preiserhoehung()*

4.2.1 Umsetzung - *preiserhoehung()*

```
1  DROP PROCEDURE IF EXISTS increasePreis;
2  delimiter //
3      CREATE PROCEDURE increasePreis(IN pr
           DECIMAL, IN abs INTEGER)
4      BEGIN
5          DECLARE average DECIMAL DEFAULT
               (SELECT avg(preis) from
               speise);
6          UPDATE speise SET preis=abs
7          where preis <= average;
8          UPDATE speise SET preis=preis *
               (100 + pr) / 100
9          where preis > average;
10     END //
11 delimiter ;
```

Listing 5: preiserhoehung() - MySQL

```
--  Erhoehen von ausgewaehlten Speicsen
    um einen spezifischen Wert
-- Erhoehen der anderen Preise um einen
    Prozentwert
CREATE FUNCTION increaseAverage(DECIMAL
    (6,2), INTEGER) returns void as
'
    UPDATE speise
    set preis = preis + $2
    where preis <= $1;

    Update speise
    set preis = preis * (100 + $2) / 100
    where preis > $1;
' language sql;
```

Listing 6: preiserhoehung() - PSQL

4.2.2 Bemerkungen

Ebenfalls mussen parameter bei einer prozedur mittels **IN** und **OUT** definiert werden, im Unterschied zu psql woebi es nur "*IN*" Parameter gibt.

4.2.3 Screenshots

```
mysql> select * from speise;
+-----+----------------------+-------+
| snr | bezeichnung          | preis |
+-----+----------------------+-------+
|   1 | Heisse Liebe         |  2.00 |
|   2 | Schoko Palatschinken |  2.00 |
|   3 | Pute gebacken        |  2.00 |
|   4 | Pute natur           |  2.00 |
|   5 | Puten-Cordon         |  2.00 |
|   6 | Menue fuer 2         | 33.12 |
|   7 | Menue fuer 3         | 40.90 |
|   8 | Menue fuer 4         | 48.71 |
+-----+----------------------+-------+
8 rows in set (0.00 sec)
```

```
mysql> call increasePreis(10,2);
Query OK, 3 rows affected, 2 warnings (0

mysql> select * from speise where snr >=
+-----+--------------+-------+
| snr | bezeichnung  | preis |
+-----+--------------+-------+
|   4 | Pute natur   |  2.00 |
|   5 | Puten-Cordon |  2.00 |
|   6 | Menue fuer 2 | 36.43 |
|   7 | Menue fuer 3 | 44.99 |
|   8 | Menue fuer 4 | 53.58 |
+-----+--------------+-------+
5 rows in set (0.01 sec)
```

4.3 AU03 - Tagesumsatz je Kellner

4.3.1 Benoetigte Inserts

```
--          Inserts weil keine CURRENT-DATE Datensaetze
INSERT INTO rechnung VALUES (100, CURRENT_DATE, 1, 'bezahlt', 1);
INSERT INTO rechnung VALUES (110, CURRENT_DATE, 2, 'bezahlt', 2);
INSERT INTO rechnung VALUES (120, CURRENT_DATE, 1, 'bezahlt', 3);
INSERT INTO rechnung VALUES (130, CURRENT_DATE, 1, 'bezahlt', 1);
INSERT INTO rechnung VALUES (140, CURRENT_DATE, 1, 'bezahlt', 1);
...

INSERT INTO bestellung VALUES (1, 100, 7);
INSERT INTO bestellung VALUES (1, 110, 8);
INSERT INTO bestellung VALUES (9, 120, 1);
INSERT INTO bestellung VALUES (9, 130, 1);
INSERT INTO bestellung VALUES (9, 140, 2);
...
```

Listing 7: inserts Au03 - MySQL

4.3.2 Umsetzung - *kellnerUmsatz()*

```
DROP FUNCTION IF EXISTS tagesumsatz;
create function tagesumsatz(kellner INT)
returns decimal(6,2) deterministic
    return
(
    select sum(preis*anzahl)
    from bestellung natural join speise
    natural join rechnung
    where knr = kellner
        and datum = CURRENT_DATE
        and status = 'bezahlt'
);
```

Listing 8: kellnerUmsatz() - MySQL

```
--  KellnerUmsatz
--  Umsatz jedes Kellners anzeigen
--  Nur des heutigen Tages
create function kellnerUmsatz(INTEGER)
    returns DECIMAL(6,2) as $$
    select sum(preis*anzahl)
    from bestellung natural join speise
    natural join rechnung
    where knr = $1
        and datum = CURRENT_DATE
        and status = 'bezahlt';
$$ language sql;
```

Listing 9: kellnerUmsatz() - PSQL

4.3.3 Bemerkungen

Hierfür konnte keine Procedur verwendet werden da der Aufruf in einer select ANweisung erfolgt. Vom Aufbau her sind beide Funktionen sehr ähnlich.

4.3.4 Screenshots

```
mysql>     select sum(preis*anzahl)
    ->     from bestellung natural join speise
    ->     natural join rechnung
    ->     where knr = 1
    ->         and datum = CURRENT_DATE
    ->         and status = 'bezahlt';
+-------------------+
| sum(preis*anzahl) |
+-------------------+
|            197.56 |
+-------------------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> select tagesUmsatz(1);
+-----------------+
| tagesUmsatz(1) |
+-----------------+
|         197.56 |
+-----------------+
1 row in set (0.00 sec)

mysql> select tagesUmsatz(2);
+-----------------+
| tagesUmsatz(2) |
+-----------------+
|         170.15 |
+-----------------+
1 row in set (0.00 sec)
```


4.4 AU04 - Mehrwehrtssteuer

4.4.1 Umsetzung - *mwst()*

```
1  DROP FUNCTION IF EXISTS bruttopreis;
2  CREATE FUNCTION bruttoPreis(preis
       DECIMAL(6,2))
3      RETURNS DECIMAL(6,2) DETERMINISTIC
4      RETURN (SELECT preis * 1.2);
5
6  DROP FUNCTION IF EXISTS mwst;
7  CREATE FUNCTION mwst(preis DECIMAL(6,2))
8      RETURNS DECIMAL(6,2) DETERMINISTIC
9      RETURN ( SELECT round(bruttoPreis(
           preis) - preis,2));
10
11 --          Aufruf
12 SELECT bezeichnung, preis as "Netto",
       bruttoPreis(speise.preis) as "Brutto
       ", MwSt(speise.preis) as "MwSt"
13 FROM   speise;
```

Listing 10: mwst() - MySQL

```
CREATE OR REPLACE FUNCTION bruttoPreis(
    speise) RETURNS DECIMAL(6,2) AS $$
    SELECT round($1.preis * 1.2,2);
$$ LANGUAGE SQL;

--    MwST
CREATE OR REPLACE FUNCTION MwSt(speise)
    RETURNS DECIMAL(6,2) as $$
    SELECT round(bruttoPreis($1) - $1.
        preis,2)
    FROM   speise;
$$ LANGUAGE SQL;

SELECT bezeichnung, preis as "Netto",
    bruttoPreis(speise) as "Brutto",
    MwSt(speise) as "MwSt"
FROM   speise;
```

Listing 11: mwst() - PSQL

4.4.2 Bemerkungen

Ähnlich wie bei psql wurden meherere Funktionen erstellt um ans Ziel zu kommen. Letztendlich sieht der Aufruf sehr ähnlich aus und die Schwierigkeit der Funktionen ist sehr ähnlich.

4.4.3 Screenshots

```
mysql> SELECT bezeichnung, preis as "Netto", bruttoPreis(speise.preis) as "Brutto", MwSt
(speise.preis) as "MwSt"
    -> FROM   speise;
+----------------------+-------+--------+-------+
| bezeichnung          | Netto | Brutto | MwSt  |
+----------------------+-------+--------+-------+
| Heisse Liebe         |  2.00 |   2.40 |  0.40 |
| Schoko Palatschinken |  2.00 |   2.40 |  0.40 |
| Pute gebacken        |  2.00 |   2.40 |  0.40 |
| Pute natur           |  2.00 |   2.40 |  0.40 |
| Puten-Cordon         |  2.00 |   2.40 |  0.40 |
| Menue fuer 2         | 36.43 |  43.72 |  7.29 |
| Menue fuer 3         | 44.99 |  53.99 |  9.00 |
| Menue fuer 4         | 53.58 |  64.30 | 10.72 |
+----------------------+-------+--------+-------+
8 rows in set, 6 warnings (0.00 sec)
```

4.5 AU05 - Noch nie bestellt

4.5.1 Umsetzung - *mwst()*

```mysql
-- Noch nie Bestellte Speisen
-- sollen angezeigt werden.
DROP PROCEDURE IF EXISTS niebestellt;
DELIMITER $$
    CREATE DEFINER=CURRENT_USER
        PROCEDURE niebestellt()
    BEGIN
        SELECT bezeichnung AS "
            Bezeichnung", preis AS "
            Nettopreis"
        FROM speise
        WHERE speise.snr NOT IN (
            SELECT snr
            FROM bestellung
        );
    END $$
delimiter ;
```

Listing 12: nochNieBestellt() - MySQL

```psql
CREATE TABLE temp (
    "Bezeichnung" VARCHAR(255),
    "Nettopreis" DECIMAL(6,2)
);

CREATE OR REPLACE FUNCTION nieBestellt()
    RETURNS SETOF temp AS $$
        SELECT bezeichnung AS "
            Bezeichnung", preis AS "
            Nettopreis"
        FROM speise
        WHERE speise.snr NOT IN (
            SELECT snr
            FROM bestellung
        );
    $$
LANGUAGE SQL;
```

Listing 13: nochNieBestellt() - PSQL

4.5.2 Bemerkungen

Hier wurde eine Prozedur gewählt da man ein set zurückgeben kann bzw. simulieren kann. In psql wird hierbei eine temporäre Tabelle benötigt um das gleiche Ergebnis zu erreichen.

4.5.3 Screenshots

```
mysql> SELECT bezeichnung AS "Bezeichnung", preis AS "Nett
    ->         FROM speise
    ->         WHERE speise.snr NOT IN (
    ->             SELECT snr
    ->             FROM bestellung
    ->         );
+---------------+------------+
| Bezeichnung   | Nettopreis |
+---------------+------------+
| Pute gebacken |       2.00 |
| Menue fuer 2  |      36.43 |
+---------------+------------+
2 rows in set (0.00 sec)
```

```
mysql> call nieBestellt();
+---------------+------------+
| Bezeichnung   | Nettopreis |
+---------------+------------+
| Pute gebacken |       2.00 |
| Menue fuer 2  |      36.43 |
+---------------+------------+
2 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql>
```

4.6 AU06 - Kellnerdaten

4.6.1 Umsetzung - *kellnerdaten()*

```mysql
drop function if exists anz_rechnungen;
create function anz_rechnungen(kellner
    INT)
    returns INT deterministic return
    (
        select count(*)
        from kellner natural join
            rechnung
        where kellner.knr = kellner
    );

drop function if exists
    spaeteste_rechnung;
create function spaeteste_rechnung(
    kellner INT)
    returns TEXT deterministic return
    (
        select status
        from rechnung
        where rechnung.knr = kellner
        order by rechnung.datum DESC
        limit 1
    );

-- Aufruf
select name, anz_rechnungen(knr) as "
    Anzahl der Rechungen",
    spaeteste_rechnung(knr) as "
    Spaeteste Rechnung"
from kellner;
```

Listing 14: kellnerdaten() - MySQL

```psql
-- Anzahl der Rechnungen
-- der Kellner
CREATE OR REPLACE FUNCTION anzRechnungen
    (INTEGER)
    RETURNS bigint AS $$
        select count(*)
        from kellner natural join
            rechnung
        where kellner.knr = $1;
    $$ Language sql;

-- Spaeteste Rechnung
-- des Kellners welcher mittels
-- paramter ausgewaehlt wurde
CREATE OR REPLACE FUNCTION
    spaetesteRechnung(INTEGER)
    RETURNS character AS $$
        select status
        from rechnung
        where rechnung.knr = $1
        order by rechnung.datum DESC
        limit 1;
    $$ Language sql;

-- Aufruf der obigen Funktionen
select name, anzRechnungen(knr) as "
    Anzahl der Rechungen",
    spaetesteRechnung(knr) as "Spaeteste
     Rechnung"
from kellner;
```

Listing 15: kellnerdaten() - PSQL

4.6.2 Bemerkungen

Funktionen sind wiederum sehr ähnlich aufgebaut. Es musste lediglich die Syntax der jeweiligen Funktion angepasst werden, wobei die parametisierung geändert werden musste, jedoch die grundsätzliche select anweisung unverändert gelassen worden konnte.

4.6.3 Screenshots

```
mysql> SELECT bezeichnung, preis as "Netto", bruttoPreis(speise.preis) as "Brutto", MwSt
(speise.preis) as "MwSt"
    -> FROM   speise;
+----------------------+-------+--------+-------+
| bezeichnung          | Netto | Brutto | MwSt  |
+----------------------+-------+--------+-------+
| Heisse Liebe         |  2.00 |   2.40 |  0.40 |
| Schoko Palatschinken |  2.00 |   2.40 |  0.40 |
| Pute gebacken        |  2.00 |   2.40 |  0.40 |
| Pute natur           |  2.00 |   2.40 |  0.40 |
| Puten-Cordon         |  2.00 |   2.40 |  0.40 |
| Menue fuer 2         | 36.43 |  43.72 |  7.29 |
| Menue fuer 3         | 44.99 |  53.99 |  9.00 |
| Menue fuer 4         | 53.58 |  64.30 | 10.72 |
+----------------------+-------+--------+-------+
8 rows in set, 6 warnings (0.00 sec)
```

4.7 AU07 - Tagesumsatz aller Kellner

4.7.1 Umsetzung - *tagesumsatz_pro_kellner()*

```
1  drop procedure if exists
       tagesumsatz_pro_kellner;
2      delimiter $$
3          create procedure
               tagesumsatz_pro_kellner()
4          begin
5              select name as "kname", sum(
                   preis*anzahl)
6              from bestellung
7                  natural join speise
8                  natural join rechnung
9                  natural join kellner
10             where rechnung.status = '
                   bezahlt'
11             group by name;
12         end $$
13     delimiter ;
14
15 --           Aufruf
16 call tagesumsatz_pro_kellner();
```

Listing 16: tagesumsatzK() - MySQL

```
create table temp (
kname VARCHAR(255),
kumsatz DECIMAL(6,2)
);

--          Funktion
--          @patam NONE
--          @return SETOF temp
create or replace function tagesumsatz()
returns setof temp as $$
select name as "kname", sum(preis*anzahl
    )
from bestellung natural join speise
natural join rechnung
natural join kellner
where rechnung.status = 'bezahlt'
group by name;
$$ language sql;

select tagesumsatz();
```

Listing 17: tagesumsatzK() - PSQL

4.7.2 Bemerkungen

Es ist nicht möglich diese Aufgabe mithilfe von Functions zu lösen. Deshalb wurde auf Prozeduren zurückgegriffen. Hierbei wurde letztendlich eine select anweisung aufgerufen und zurückgegeben, da eine Funktion in mysql nicht die Möglichkeit bietet ein set zurückzugeben.

In Psql wurde aufgrund dessen eine zusätzliche temporäre Tabelle entworfen um für den Zeitraum des Aufrufs die Datenaufteilung zu representieren damit sinnvolle Werte zurückgeliefert werden können.

4.7.3 Screenshots

```
mysql> select name as "kname", sum(preis*anzahl)
    ->     from bestellung natural join speise
    ->                     natural join rechnung
    ->                     natural join kellner
    ->     where rechnung.status = 'bezahlt'
    ->     group by name;
+----------+-------------------+
| kname    | sum(preis*anzahl) |
+----------+-------------------+
| Kellner1 |            237.56 |
| Kellner2 |            170.15 |
| Kellner3 |             36.00 |
+----------+-------------------+
3 rows in set (0.00 sec)
```

```
mysql> call tagesumsatz_pro_kellner();
+----------+-------------------+
| kname    | sum(preis*anzahl) |
+----------+-------------------+
| Kellner1 |            237.56 |
| Kellner2 |            170.15 |
| Kellner3 |             36.00 |
+----------+-------------------+
3 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql>
```


Listings

