Übungsblatt 2

# Gespeicherte Programme

## flighttime

DELIMITER **//**

**CREATE** **FUNCTION** `flighttime` **(**p\_departure DATETIME**,** p\_arrival DATETIME**)** **RETURNS** **VARCHAR(**255**)**

**BEGIN**

**DECLARE** v\_output **VARCHAR(**255**)** **DEFAULT** ''**;**

**DECLARE** v\_diff **TIME;**

**DECLARE** v\_hours**,** v\_minutes **INT;**

**SET** v\_diff **=** TIMEDIFF**(**p\_arrival**,** p\_departure**);**

**SET** v\_hours **=** **HOUR(**v\_diff**);**

**SET** v\_minutes **=** **MINUTE(**v\_diff**);**

**IF** v\_hours **>** 0 **THEN** **SET** v\_output **=** **CONCAT(**v\_hours**,** ' hour '**);**

ELSEIF v\_hours **>** 1 **THEN** **SET** v\_output **=** **CONCAT(**v\_output**,** 'hours '**);**

**END** **IF;**

**SET** v\_output **=** **CONCAT(**v\_output**,** v\_minutes**,** ' minute'**);**

**IF** v\_minutes **<>** 1 **THEN** **SET** v\_output **=** **CONCAT(**v\_output**,** 's'**);**

**END** **IF;**

**RETURN** v\_output**;**

**END** **//**

DELIMITER **;**

## book\_asap

Leider wurden mit bei meiner Prozedur immer 2 Datensätze in die Tabelle buchung eingefügt. Ich habe leider keine Ahnung warum das so ist.

Des Weiteren wusste ich nicht, welchen Preis ich setzen soll, da dieser nicht als Parameter übergeben wird. Ich habe ihn deshalb einfach auf 0.0 gesetzt.

**CREATE** **TABLE** `buchung` **LIKE** `FlughafenDB`**.**`buchung`**;**

DELIMITER **//**

**CREATE** **PROCEDURE** `book\_asap`**(IN** p\_start **CHAR(**3**),** **IN** p\_destination **CHAR(**3**),** **IN** p\_time DATETIME**,** **IN** p\_pass **CHAR(**9**))**

**BEGIN**

**DECLARE** v\_flightnr **CHAR(**8**);**

**DECLARE** v\_flight\_id **INT(**11**);**

**DECLARE** no\_flight\_found CONDITION **FOR** **SQLSTATE** '45000'**;**

**DECLARE** v\_passenger\_id **CHAR(**9**);**

**DECLARE** invalid\_pass CONDITION **FOR** **SQLSTATE** '45001'**;**

**SELECT** `passagier\_id` **INTO** v\_passenger\_id

**FROM** `FlughafenDB`**.**`passagier`

**WHERE** `passnummer` **=** p\_pass

**LIMIT** 1**;**

**IF** v\_passenger\_id **IS** **NULL** **THEN**

SIGNAL invalid\_pass **SET** MESSAGE\_TEXT **=** 'Invalid pass number'**;**

**END** **IF;**

**SELECT** `f`**.**`flug\_id`**,** `f`**.**`flugnr` **INTO** v\_flight\_id**,** v\_flightnr

**FROM** `FlughafenDB`**.**`flughafen` **AS** `fhvon`

**INNER** **JOIN** `FlughafenDB`**.**`flug` **AS** `f`

**ON** `f`**.**`von` **=** `fhvon`**.**`flughafen\_id`

**INNER** **JOIN** `FlughafenDB`**.**`flughafen` **AS** `fhnach`

**ON** `f`**.**`nach` **=** `fhnach`**.**`flughafen\_id`

**INNER** **JOIN** `FlughafenDB`**.**`flugzeug` **AS** `fz`

**ON** `f`**.**`flugzeug\_id` **=** `fz`**.**`flugzeug\_id`

**WHERE** `fhvon`**.**`iata` **=** p\_start

**AND** `fhnach`**.**`iata` **=** p\_destination

**AND** `f`**.**`abflug` **>** p\_time

**AND** `fz`**.**`kapazitaet` **>** **(SELECT** **COUNT(\*)** **FROM** `FlughafenDB`**.**`flug` **AS** `fi` **INNER** **JOIN** `FlughafenDB`**.**`buchung` **AS** `bi` **ON** `fi`**.**`flug\_id` **=** `bi`**.**`flug\_id` **WHERE** `fi`**.**`flug\_id` **=** `f`**.**`flug\_id`**)**

**ORDER** **BY** `f`**.**`ankunft` **ASC**

**LIMIT** 1**;**

**IF** v\_flightnr **IS** **NULL** **THEN**

SIGNAL no\_flight\_found **SET** MESSAGE\_TEXT **=** 'Sorry, no flight available'**;**

**END** **IF;**

**INSERT** **INTO** `buchung` **(**`flug\_id`**,** `passagier\_id`**,** `preis`**)** **VALUES** **(**v\_flight\_id**,** v\_passenger\_id**,** 0.0**);**

**SELECT** **CONCAT(**'A seat from '**,** p\_start**,** ' to '**,** p\_destination**,** ' was booked on flight '**,** v\_flightnr**)** **AS** `booked`**;**

**END** **//**

DELIMITER **;**

## erreichbare\_flughaefen

DELIMITER **//**

**CREATE** **PROCEDURE** `erreichbare\_flughaefen`**(IN** p\_id **SMALLINT(**6**),** **IN** p\_hops TINYINT**,** **IN** p\_recursion TINYINT**)**

**BEGIN**

**DECLARE** v\_airport\_id **SMALLINT(**6**);**

**DECLARE** v\_from **SMALLINT(**6**);**

**DECLARE** v\_done TINYINT **DEFAULT** 0**;**

**DECLARE** c\_airport **CURSOR** **FOR**

**SELECT** **DISTINCT** `von`**,** `nach`

**FROM** `FlughafenDB`**.**`flugplan` **AS** `fp`

**WHERE** `fp`**.**`von` **=** p\_id **AND** `fp`**.**`von` **IS** **NOT** **NULL** **AND** `fp`**.**`nach` **IS** **NOT** **NULL;**

**DECLARE** **CONTINUE** HANDLER **FOR** **NOT** **FOUND** **SET** v\_done **=** 1**;**

**IF** p\_recursion **=** 0 **THEN**

**SET** max\_sp\_recursion\_depth **=** p\_hops **+** 1**;**

**DROP** **TABLE** **IF** **EXISTS** `hops`**;**

**CREATE** **TABLE** `hops` **(**

`flughafen\_id` **SMALLINT(**6**)** **NOT** **NULL,**

`hops` TINYINT **NOT** **NULL**

**)**ENGINE**=**MEMORY**;**

**END** **IF;**

**IF** p\_recursion **<=** p\_hops **THEN**

**OPEN** c\_airport**;**

l\_fetch\_data**:** **LOOP**

**FETCH** c\_airport **INTO** v\_from**,** v\_airport\_id**;**

**IF** v\_done **THEN**

**CLOSE** c\_airport**;**

LEAVE l\_fetch\_data**;**

**END** **IF;**

**INSERT** **INTO** `hops` **VALUES** **(**v\_airport\_id**,** p\_recursion**);**

**CALL** erreichbare\_flughaefen**(**v\_airport\_id**,** p\_hops**,** **(**p\_recursion **+** 1**));**

**END** **LOOP** l\_fetch\_data**;**

**END** **IF;**

**END** **//**

DELIMITER **;**

# Triggers uns Views

## View

Beim Insert wird der Fehler „CHECK OPTION failed 'mdeutschl\_FlughafenDB.v\_buchung‘“ geworfen. Die WITH CHECK Klausel wendet alle Constraints der WHERE Klausel und validiert diese gegen alle Statements, die auf die View abgesetzt werden und failt somit wenn versucht wird, einen ungültigen Wert einzufügen.

**CREATE** **OR** **REPLACE** **VIEW** `v\_buchung` **AS**

**SELECT** **\*** **FROM** `buchung` **WHERE** `buchung\_id` **BETWEEN** 10 **AND** 1000

**WITH** **CHECK** **OPTION;**

## Trigger

Die Spalte preis ist mit dem NOT NULL constraint versehen, weshalb sowieso keine NULL-Werte für den Preis eingefügt werden können.

DELIMITER **//**

**CREATE** **TRIGGER** `t\_price\_invalid` **BEFORE** **INSERT** **ON** `buchung`

**FOR** **EACH** **ROW**

**BEGIN**

**IF** **NEW.**`preis` **IS** **NULL** **OR** **NEW.**`preis` **<** 0 **THEN**

**SET** **NEW.**`preis` **=** 10000.0**;**

**END** **IF;**

**END** **//**

DELIMITER **;**

# Transaktionen

## 

Ausführungsreihenfolge: T2, T3, T4, T1

## 

Ausführungsreihenfolge: T2, T3, T4, T1

# 2-Phasen-Sperrprotokoll

|  |  |  |  |
| --- | --- | --- | --- |
| Ta | Tb | Tc |  |
|  |  | r(x) | x++ (1) |
| w(y) |  |  | SET WRITE LOCK y |
| r(x) |  |  | x++ (2) |
|  | w(x) |  | WAIT (Tb wird unterbrochen) |
|  |  | r(x) | x++ (3) |
|  |  | r(z) | z++ (1) |
|  |  | w(z) | SET WRITE LOCK z |
|  | w(y) |  |  |
|  |  | w(x) | WAIT |
| r(y) |  |  | y++ (1) |
| eot |  |  | Reevaluierung von Tb |
|  |  | w(x) | SET WRITE LOCK x |
|  | w(x) |  |  |
|  |  | eot |  |

Tb wartet auch nach dem eot von Ta weiter, da auch Tc eine Lesesperre auf x erstellt hat, entsprechend wird Tb immer wieder unterbrochen. Das führt dazu, dass Ta problemlos fertig wird und danach Tc problemlos fertig wird. Tb kann erst nach Ta und Tc ausgeführt werden.