Datenbankpraktikum SoSe 2008

Simulation einer automatischen Gepäckaufbewahrung

Tobias Riedel 725241

19. September 2008

Universität Potsdam

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Aufgabenstellung

Aufgabe des Datenbankpraktikums war es, eine automatische Gepäckaufbewahrung zu simulieren, ähnlich denen, die man an modernen Flughäfen oder Bahnhöfen finden kann. Bei diesen gibt der Nutzer sein Gepäck an einem Terminal auf und erhält dann irgend eine Art der Quittung für das Gepäckstück. Das kann zum Beispiel in Form eines Zettels mit Strichkode sein. Hat der Besitzer also seinen Koffer oder Ähnliches abgegeben, sorgt das System nicht nur für die Erzeugung der Quittung, sondern auch für den Transport des Objektes zur Aufbewahrungsstelle, wo es eingelagert wird, bis der Nutzer es wieder über das Terminal abholt. Alternativ kann es auch sein, dass Gepäck, das sich länger als 24 Stunden im Lagerungssystem befindet, entsorgt wird.

In Anlehnung an diese realen Vorlagen ergaben sich folgende Vorgaben[1] für das Praktikum, die mit Abbildung 1[1] unterstützt wurden:

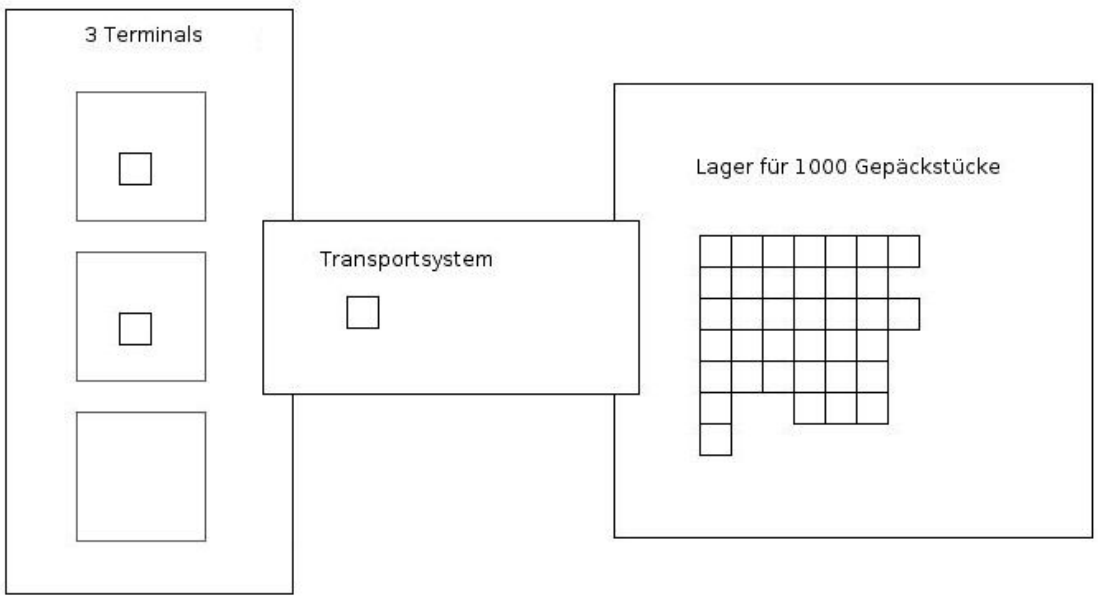

Abbildung 1: Skizze zur Aufgabenstellung

[1] Quelle: [DBPW]

1. *Es sollen mindesten 1000 Gepäckstücke verwaltet werden können.*
2. *Überschreitet die Einlagerungsdauer eines Gepäcks 24 Stunden, soll das Gepäckstück automatisch aus dem Lager entfernt werden.*
3. *Es sollen mindestens 3 Terminals betrieben werden, an denen gleichzeitig Gepäck aufgegeben bzw. abgeholt werden kann.*
4. *Es soll ein reibungsloser Transport des Gepäcks zwischen Terminal und dem Einlagerungsort gewährleistet werden.*

In dieser Ausarbeitung wird mein Projekt als das **Baggage Management System** - kurz **BMS** - bezeichnet.

2 Verwendete Software

2.1 Entwicklungssoftware

Da es, wie in Kapitel 1 beschrieben, keine Vorgaben für die zu verwendende Software gab, waren die Projektrestriktionen und persönliche Präferenzen ausschlaggebend. Aufgrund langjähriger Erfahrung mit **MySQL**[1] und der Tatsache, dass es kostenlos war, entschied ich mich für dieses Datenbanksystem. Die wichtigste, noch nicht genannte, Einschränkung lautete, dass die komplette Gepäckaufbewahrungssimulation in der Datenbank abgewickelt werden muss. Also kam man um *Stored Procedures / Functions*[2] nicht herum. Um außerdem einen kontinuierlichen Ablauf des Transports zu gewährleisten, sah ich nur die Möglichkeit, mit Events zu arbeiten. Diese werden erst seit Version 5.1[3] von MySQL unterstützt. Letztendlich wurde mein Projekt unter **Version 5.1 und der Alphaversion 6.0 von MySQL** entwickelt.

Für das Web Front-End wurde Server-seitig **PHP 5**[4] und Client-seitig JavaScript eingesetzt. Die **jQuery**[5] Library erleichterte die AJAX[6]-Programmierung in JavaScript.

[1] http://www.mysql.com/
[2] In der Datenbank gespeicherte Prozeduren und Funktionen
[3] Release Candidate, Stand: 11.09.2008
[4] http://www.php.net/
[5] http://www.jquery.com/
[6] Asynchronous JavaScript and XML

Mit Hilfe folgender Programme entstand das BMS:

- Server-Software:
  - Apache 2.2.8 Win32
  - MySQL 6.0 Alpha und 5.1 RC
  - PHP 5.2.5
  - phpMyAdmin 2.11.4[7]
- Client-Software:
  - jQuery 1.2.6
  - jQuery Live Query Plugin 1.0.2[8]

2.2 Testsoftware

Getestet wurde das Projekt mit dieser Software:

- Betriebssysteme:
  - Windows XP Home 32 Bit Service Pack 2
  - Windows Vista Home Premium 32 Bit Service Pack 1
- Browser:
  - Firefox 3.0.1[9]
  - Opera 9.52[10]
  - Internet Explorer 7.0.5730.13[11]

[7] http://www.phpmyadmin.net/
[8] http://brandonaaron.net/docs/livequery/
[9] http://www.mozilla-europe.org/de/firefox/
[10] http://de.opera.com/
[11] http://www.microsoft.com/germany/windows/downloads/ie/getitnow.mspx

3 Verwendete Hardware

Drei Testrechner standen zur Verfügung (Tabelle 1). Das Projekt wurde auf diesen PCs unterschiedlicher Leistungsstärke getestet. Die Wegfindung hat sichtbare Unterschiede bei den verschiedenen Hardware-Konfigurationen aufgedeckt. So lief sie auf dem langsamsten Testrechner Nummer 1 bei der Betrachtung im Web Front-End wesentlich holpriger ab als auf den anderen Testsystemen. Sonst gab es keine weiteren Unterschiede, mit Ausnahme eines kleinen Problems, dass in Kapitel 7 „Bekannte Fehler“ erläutert wird. Trotzdem wird ein Rechner, vergleichbar mit Testsystem 2, zur Betrachtung des BMS über das Web Interface empfohlen.

Komponente	**Testrechner 1**	**Testrechner 2**	**Testrechner 3**
Prozessor	Pentium 4	AMD 64 X2 Dual Core	Intel Core 2 Duo T9300
Taktung	1 x 2,6 GHz	2 x 2,0 GHz	2 x 2,5 GHz
RAM	0,5 GB	2,0 GB	4,0 GB
Systemtyp	32 Bit	64 Bit	64 Bit
Betriebssystem	Windows XP 32 Bit	Windows XP 32 Bit	Windows Vista 32 Bit

Tabelle 1: Testrechnerkonfigurationen

4 Installation

Die Installation wird detailliert in der **readme.txt**[12] beschrieben. Hier nur eine kurze Zusammenfassung:

4.1 Datenbank einrichten

1. In den „sql“-Ordner im Installationsverzeichnis wechseln:
   cd sql

2. MySQL Server 5.1 (oder höher) starten:
   mysqld −−event-scheduler=1

3. Per Konsole auf selbigen verbinden:
   mysql -uYourUsername -pYourPassword

4. Neue Datenbank mit beliebigem Namen erstellen:
   CREATE DATABASE YourDatabase;

[12] Befindet sich im Installationsordner

5. Auf die neue Datenbank verbinden:
   USE YourDatabase;

6. Installationsskript ausführen:
   SOURCE init.sql;

Wie das System in Verbindung mit der Konsole gesteuert werden kann, ist in Kapitel 5.1 nachzulesen.

4.2 Web Front-End-Konfiguration

Es folgt die Konfiguration des Web Front-Ends:

1. Datei **config.inc.php** im Installationsordner mit beliebigem Texteditor öffnen

2. Werte für **CFG_DB_HOST** (Name des MySQL-Servers, z.B: „localhost“), **CFG_DB_NAME** (Name der zu verwendenden Datenbank), **CFG_DB_USER** (Benutzername für den MySQL-Server) und **CFG_DB_PWD** (dazugöriges Passwort) anpassen

3. Falls nicht bereits geschehen: **JavaScript im Browser aktivieren**

4. Per Webbrowser ins Installationsverzeichnis wechseln und die Datei **index.php** aufrufen

Die Steuerung des BMS über das Web Front-End wird in Abschnitt 5.2 genauer beschrieben.

5 Steuerung

5.1 Konsole

Wichtig bei der Arbeit mit der Konsole ist es, dass der Event Scheduler aktiviert wird. In meinen Testfällen hat er sich mit jedem Server Shutdown wieder auf 0 zurückgesetzt. Ohne den Scheduler wird das Gepäck nicht weiterbewegt. Hat man den MySQL-Server - wie in Kapitel 4.1 beschrieben - mit dem Parameter `--event-scheduler=1` gestartet, braucht man sich nicht mehr darum kümmern. Ohne den Parameter, muss man ihn mit dem Aufruf `CALL setEventScheduler(1);` einschalten. Nun kann man alle Anfragen, die in Tabelle 3 aufgelistet sind, zur Steuerung des BMS' verwenden.

5.2 Web Front-End

Im Gegensatz zum alleinigen Einsatz der Konsole, braucht man sich als Nutzer des Web Front-Ends nicht um den Event Scheduler sorgen. Beim ersten Betreten der Webseite, wird er automatisch aktiviert. Das zeigt sich auch in dem Query Log. Was welche Knöpfe und Kontrollfelder des Front-Ends bewirken, wird direkt neben ihnen beschrieben und dürfte selbst erklärend sein.

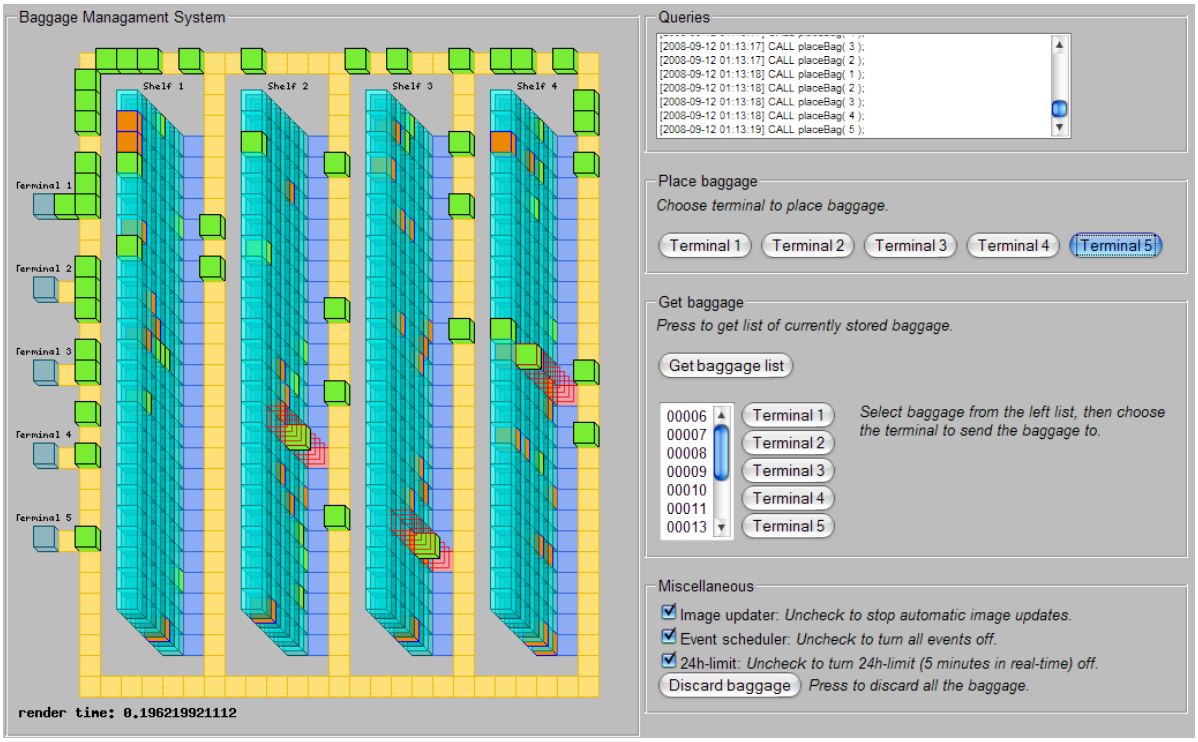

Abbildung 2: Screenshot des Web Front-Ends

5.3 Debugging

Für erleichtertes Debugging ist es empfehlenswert, die Konstante `SHOW_BAG_LABELS` in der Datei **config.inc.php** im Installationsverzeichnis auf den Wert 1 zu setzen. Durch diese Änderung wird auf jedem Gepäckstück im Web Interface die eigene ID angegeben. Die Deaktivierung des Event Schedulers bewirkt eine Pausierung des gesamten Geschehens. Und nicht zu vergessen, bei längeren Tests: die Deaktivierung des 24h-Events.

6 Aufbau

6.1 Infrastruktur

In Grafik 3[13] wird das ER-Diagramm zur Datenbankstruktur dargestellt. Neben den dort visualisierten Relationen existieren noch zwei Views **bag_list** und **bag_image**. Während erstere die Identifikationsnummern der Gepäckstücke zurückliefert, die bereits in einem Regal eingelagert wurden, wird die zweite View nur von dem Web Front-End verwendet, um das Bild des Systems zu erzeugen. Die weitreichenden Abhängigkeiten durch Fremdschlüssel in den Datentabellen werden in Abbildung 4[14] aufgezeigt.

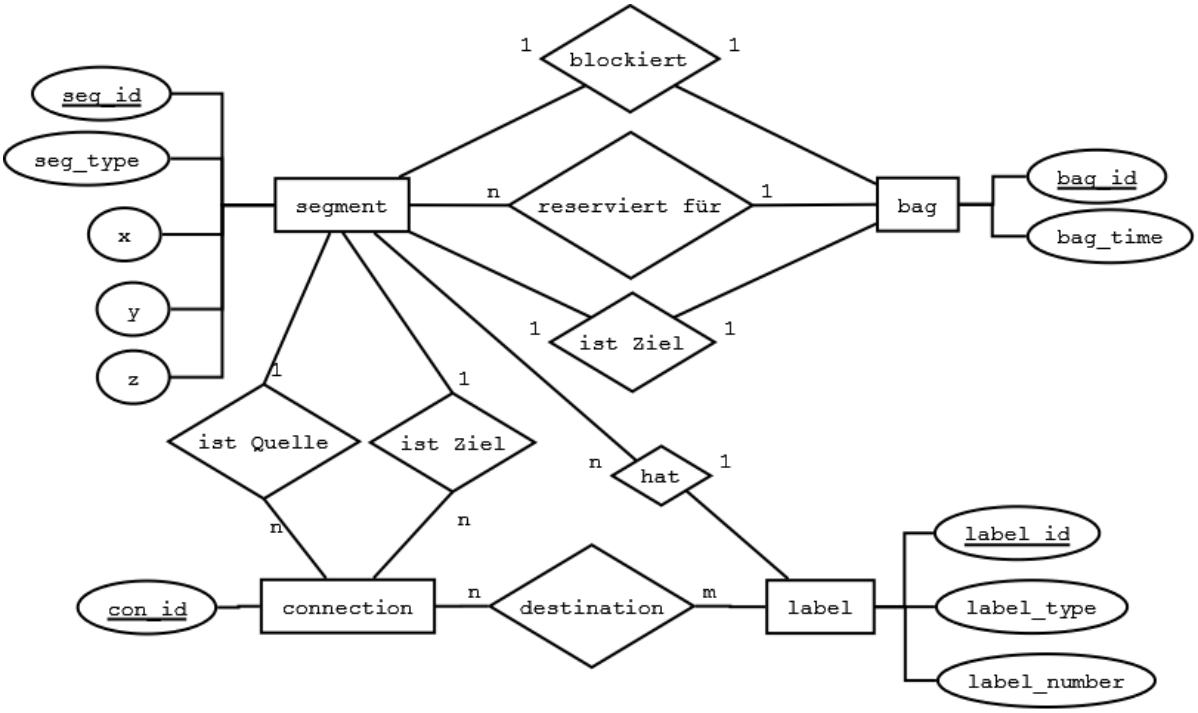

Abbildung 3: Entity Relationship Diagram

[13] Quelle: [Eigene Darstellung], Bild in Originalgröße: `doc/img/er.png`

[14] Erzeugt mit DbVisualizer Free 6.0.12, http://www.dbvis.com/products/dbvis/, Bild in Originalgröße: `doc/img/foreign_keys.png`

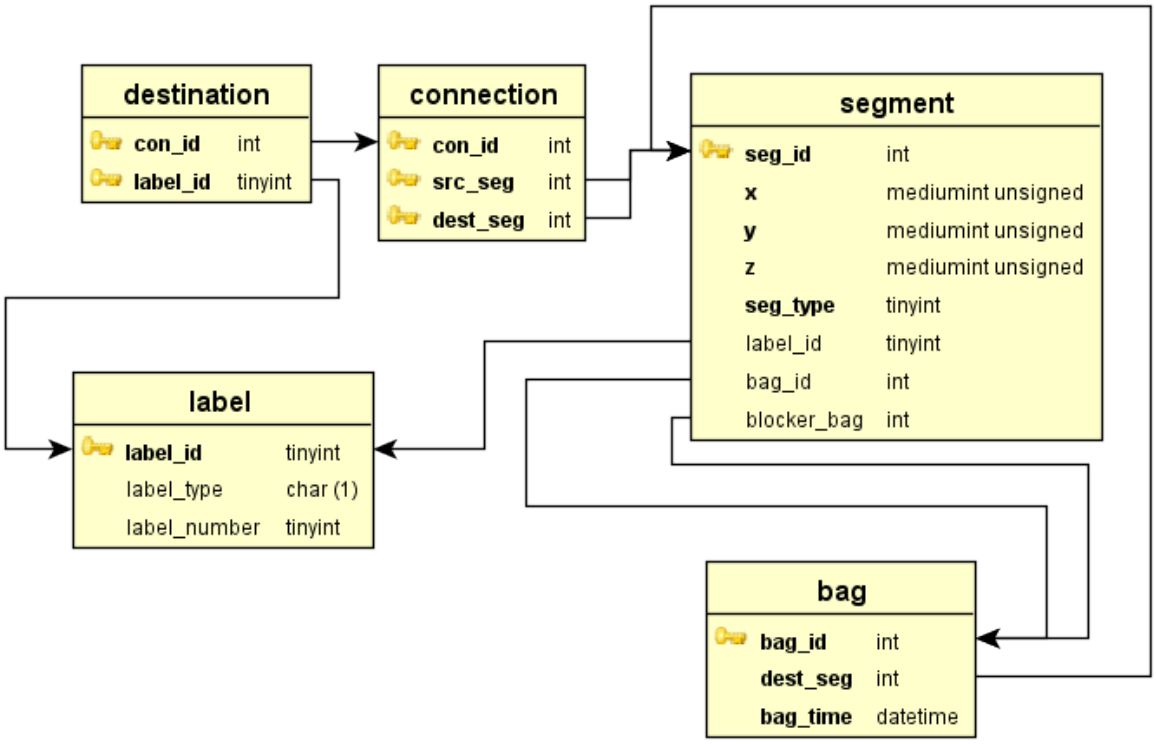

Abbildung 4: Fremdschlüsselverknüpfungen der Datentabellen

6.1.1 segment

Zentralstes Element des ganzen Systems sind die *Segmente*, die in der Datentabelle **segment** abgespeichert sind. Diese Bausteine stellen nicht nur die gesamte sichtbare Infrastruktur bestehend aus Terminals, Fließbändern, Lifts und Regalfächern[15] dar; sondern enthalten auch Informationen darüber, wo sich welches Gepäckstück gerade befindet (Attribut `bag_id`). Zusätzlich wird in den Segmenten auch abgespeichert, welches von ihnen für welches Gepäck reserviert oder blockiert wird (Attribut `blocker_bag`).

Typ	Typname	Funktion
0	Fließband	Transportiert Gepäck von Terminals zu Regallifts und zurück
1	Lift	Transportiert Gepäck von Fließbändern zu Regalfächern und zurück
2	Terminal	Aufgabe und Entgegennahme von Gepäck
3	Regalfach	Lagerungsort für genau ein Gepäckstück

Tabelle 2: Datenbank: Segmenttypen

[15] Siehe Typenbeschreibung in Tabelle 2

6.1.2 bag

Alle Koffer, Taschen, etc. werden nach der Abgabe am Terminal in der Datentabelle **bag** gespeichert.

6.1.3 connection

Mit einigen Ausnahme sind alle benachbarten Segmente miteinander verbunden, um den Transport zu gewährleisten. Diese gerichteten Verknüpfungen werden in der Datentabelle **connection** gesichert.

6.1.4 destination

Wie eben in Abschnitt 6.1.3 beschrieben, liegen Vorgänger-Nachfolger-Beziehungen zwischen den einzelnen Segmenten vor. An einigen Stellen gibt es aber mehr als nur einen möglichen Nachfolger. Um dennoch an das richtige Ziel zu gelangen, werden Meta-Informationen zu solchen Weichen in der Datentabelle **destination** hinterlegt. Diese erlauben es, an eine *connection* beliebig viele *labels* zu hängen, die auf die über diese Verbindung erreichbaren Ziele verweisen.

6.1.5 label

In der **label**-Datentabelle stehen die Markierungen, über die Segmente als Teil eines Regals oder eines Terminals identifiziert werden können. Wie oben erwähnt, wird in Verbindung der destination-Tabelle und den labels die Wegfindung ermöglicht.

6.2 Prozeduren und Funktionen

Aufruf	**Funktion**
Steuerung (Datei: proc_bags.sql)	
CALL placeBag(x);	Neues Gepäckstück an Terminal **x** aufgeben
CALL retrieveBag(x, y);	Gelagertes Gepäckstück **x** zu Terminal **y** schicken
CALL discardBags();	Alle Gepäckstücke verwerfen
SELECT * FROM 'bag_list';	Alle eingelagerten Gepäckstücke auflisten
Datenbankeinstellung (Datei: proc_events.sql)	
SELECT getEventScheduler();	Gibt den aktuellen Status des Event Schedulers zurück
CALL setEventScheduler(x);	Schaltet den Event Scheduler ein (**x=1**) oder aus (**x=0**)
SELECT getEvent24hLimit();	Gibt den aktuellen Status des 24h-Events zurück
CALL setEvent24hLimit(x);	Schaltet das 24h-Event ein (**x=1**) oder aus (**x=0**)
Transport (Datei: proc_pathfinding.sql)	
CALL getSucc(x);	Bewegt Gepäckstück **x** einen Schritt weiter
CALL checkSucc(x, y, z);	Testet, ob Segment **x** für Tasche **y** frei (**z**) ist
CALL moveBags();	Bewegt alle Gepäckstücke einen Schritt weiter
Debugging (Datei: proc_debug.sql)	
CALL fill(x);	Fülle die 1.000 Regalfächer um **x** Taschen auf

Tabelle 3: Datenbank: Prozeduren und Funktionen

6.3 Ablauf

Die folgenden Prozedur- und Funktionsnamen beziehen sich auf Tabelle 3. Die Parameterwerte sind exemplarisch.

6.3.1 Gepäckaufgabe

Nachdem ein neues Gepäckstück von dem Nutzer über den Aufruf `CALL placeBag(1);` an Terminal 1 aufgegeben werden soll, wird in selbiger Prozedur zunächst überprüft, ob das Terminal frei ist. Dafür wird `checkSucc` zurate gerufen. Gibt diese Prozedur ein positives Signal zurück, muss noch ein freies Regalfach gefunden werden, wo die Tasche eingelagert werden kann. Die Auswahl erfolgt per Zufall unter den freien Fächern. Wurde ein solches gefunden, kann die neue Tasche in der Datentabelle `bag` erzeugt werden. Gleichzeitig wird über das Attribut `bag.dest_seg` das Ziel abgespeichert.

Nachdem dies geschehen ist, wird das Gepäck auf dem Terminal platziert. Dazu wird dessen Attribut `segment.bag_id` auf die ID der neu erzeugten Tasche gesetzt. Direkt danach wird die zuvor ermittelte Regalbox für die Tasche reserviert. Also erhält das

Attribut `segment.blocker_bag` des Fachsegments ebenfalls die ID der Tasche. Damit ist das Fach für andere Taschen nicht mehr als frei markiert.

6.3.2 Transport zum Regal

Wie in Abschnitt 6.1 beschrieben, sind die einzelnen Segmente, über die das Gepäck zu seinem Ziel gelangen soll, miteinander über die `connection`-Datentabelle verbunden. Jede Sekunde wird das Event `ev_moveBags` aufgerufen, dass wiederrum die Prozedur `moveBags` ausführt. Diese veranlasst für jedes Gepäckstück, dass noch nicht an seinem Ziel angekommen ist, sich nach Möglichkeit einen Schritt weiter in Richtung Bestimmungsort zu bewegen. Also wird die Prozedur `getSucc` aufgerufen, welche sowohl die Wegfindung als auch das Weitersetzen der Taschen übernimmt.

Zuerst werden vom aktuellen Standort der Tasche ausgehend die Möglichen Verbindungen zu Nachbarelementen gezählt. Sogleich werden von einer dieser Verbindungen Informationen gesammelt. Nun gibt es zwei Möglichkeiten. Als erstes kann es sein, dass es nur einen Nachfolger gibt. Ist dies der Fall, wird dieser überprüft, ob er nicht gerade von einem anderen Gepäckstück belegt, oder für eine andere Tasche reserviert ist. Bei freiem Status, wird die `bag_id` des alten Segments auf `NULL` gesetzt, während der Nachfolger wieder die ID der Tasche erhält.

Tritt jedoch alternativ der Fall ein, dass es mehr als einen Nachfolger gab, kommt die `destination`-Datentabelle ins Spiel. Ihre Einträge verweisen an Weggabelungen auf die verschiedenen Regale. Somit kann der richtige Weg ermittelt werden.

Am richtigen Regal angekommen, muss noch das richtige Fach gefunden werden. Dazu wird bei mehreren Nachfolgern ohne Zielbestimmung folgendes simple Schema angewendet:

1. Schiebe Tasche nach Süden, bis Tasche und Fach auf gleichem y-Wert sind.
2. Bewege Tasche einen Schritt nach Westen auf den Lift.
3. Lass die Tasche mit dem Lift nach oben fahren, bis Tasche und Fach den gleichen z-Wert haben.
4. Transportiere Tasche ein Stück nach Westen in das Fach hinein.

Wird der Lift betreten, reserviert das System alle Segmente dieser x-y-Koordinate für diese Tasche (`segment.blocker_bag`). Der Fahrstuhl wird beim Verlassen wieder freigegeben.

6.3.3 Rücktransport zum Terminal

Mit dem Aufruf `SELECT * FROM 'bag_list';` durch den Nutzers kann sich selbiger die eingelagerten Gepäckstücke anzeigen lassen. Führt er nun die Anfrage `CALL retrieveBag(1, 3);` aus, möchte er damit bezwecken, dass der Koffer mit der `bag_id` 1 zu dem Terminal mit der `label_number` 3 gesendet wird, um es da abzuholen.

Nun wird nachgeforscht, ob das Terminal frei ist. Wenn dem so ist, bekommt das Gepäck das Terminal als neues Ziel zugeordnet (`bag.dest_seg`). Damit greift wieder das Event, dass den Koffer nun als Element identifiziert, das nicht an seinem Bestimmungsort angelangt ist. Es tritt wieder ein fest definiertes Schema für das Gepäck in Aktion:

1. Regal nach Osten verlassen und damit den Fahrstuhl betreten. Gleichzeitig das Fach wieder für andere Taschen freigeben.
2. Den Lift ganz nach unten fahren.
3. Den Fahrstuhl nach Osten verlassen und wieder auf das Fließband gelangen.
4. Dem Förderband solange nach Süden folgen, bis es wieder nur einen Nachfolger gibt.

Angekommen an den Terminals gibt es wieder viele Weichen, welche aber alle mit Destinations versehen sind, und somit den richtigen Weg weisen. Ist das gewünscht Terminal schließlich erreicht, wird die Tasche einfach aus der `bag`-Tabelle gelöscht. Durch die Fremdschlüsselabhängigkeiten[16] zu der Tasche, werden automatisch von der Datenbank alle bestehenden Reservierungen auf `NULL` zurückgesetzt.

6.3.4 Verfall des Gepäcks

Alle fünf Minuten wird das Event `ev_24hLimit` aufgerufen, welches alle Gepäckstücke, die länger als fünf Minuten im Umlauf sind, löscht. Die Fremdschlüsselverknüpfungen besorgen wieder den Rest. Dadurch kann es passieren, dass Koffer maximal 9:59 Minuten im System vorhanden sind. Mit `CALL setEvent24hLimit(0);` lässt sich dieses Event deaktivieren.

In meinen Testfällen war es allerdings so, dass wenn der MySQL-Server geschlossen wurde, während das Event deaktiviert war, es beim nächsten Start des Servers nicht mehr in der Datenbank vorhanden war. Es musste neu installiert werden.

[16] Siehe Abbildung 4

7 Bekannte Fehler

Problem: In einigen Fällen passierte es, dass sich das Bild des Gepäcksystems im Web Front-End nicht mehr von alleine aktualisierte.
Lösung: Durch Aktualisierung der Webseite (Taste [F5]) funktioniert das Auto-Update wieder. Ursache für dieses Problem könnten im Caching liegen. Der Fehler trat nur auf Testrechner 3[17] auf, auf dem Windows Vista lief.

8 Anmerkungen

Es gäbe noch die Möglichkeit, das BMS zu erweitern. So könnte man statt der Ad Hoc-Wegberechnung eine einmalige einführen. Momentan wird bei jedem Aufruf der Wegfindungsprozedur nur der nächste Schritt berechnet und gegebenfalls ausgeführt. Je mehr Gepäckstücke auf einmal im Umlauf sind, desto langsamer könnte das System werden. Als Alternative könnte man den kompletten Weg von Start- bis Zielsegment in dem Augenblick berechnen lassen, in dem die Tasche am Terminal abgegeben wird.

Diesen kompletten Weg könnte man in der Datenbank speichern und dann bei jedem Schritt nur noch abfragen. Der logische Nachteil, der sich aus diesem Ablauf ergeben würde, wäre ein erhöhter Speicherbedarf.

Als anderer Punkt der Performanceoptimierung steht die Visualisierung im Web Interface an. In einer früheren Version habe ich *Scalable Vector Graphics* (SVG) verwendet, um das Bild zu erzeugen. Aufgrund des unglaublich langsamen Bildaufbaus war ich allerdings gezwungen, auf die gd-Library zurückzugreifen, um klassische PNGs zu erzeugen. PNG kann die gleichen Bildinhalte um ein Vielfaches schneller darstellen als SVG. Da ein Heranzoomen an die Grafik nicht notwendig ist, kann deshalb getrost auf SVG verzichtet werden.

Literatur

[DBPW] Datenbankpraktikum Webseite:
Bild von http://www.cs.uni-potsdam.de/wv/lehre/08SS/08-DB-Praktikum/inhalt.html,
Stand: 11.09.2008

[17] Siehe Tabelle 1