Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht Ferienclub

Jan Weinschenker

Business Intelligence: Extraktion, Transformation, Laden

Betreuender Prüfer: Prof. Dr. Kai v. Luck
Zweitgutachter: Prof. Dr. Olaf Zukunft

Fakultät Technik und Informatik
Studiendepartment Informatik

Jan Weinschenker

Projektbericht Ferienclub

Business Intelligence

Stichworte

Business Intelligence Data Warehousing Datenbanken ETL Analyse Extraktion Transformation Laden

Kurzzusammenfassung

Zur Analyse der im Projekt anfallenden Daten wird ein Data Warehouse implementiert. In diesem Projektbericht wird speziell der Aspekt der Extraktion der Daten aus den Produktivsystemen und die Transformation der Daten in eine analyseoptimierte Form behandelt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im Wintersemester 2005/2006 wurde im Studiendepartment Informatik der HAW Hamburg im Rahmen des Masterprogramms *Verteilte Systeme* das Projekt *Ferienclub* durchgeführt.

Dabei war es das Ziel, die IT-Landschaft für einen Ferienclub zu entwerfen, welche sich Technologien bedient, die derzeit den aktuell verfügbaren Stand der Technik widerspiegeln bzw. Gegenstand der aktuellen Forschung darstellen. Wichtige Stichworte sind in diesem Zusammenhang Ortsbezogene Dienste, Semantic Web und Serviceorientierte Architekturen. Viele Teilgruppen des Projekts haben sich damit auseinandergesetzt.

Die Idee hinter dem Teilprojekt *Business Intelligence* war, dass in den anderen Teilprojekten höchstwahrscheinlich große, aber voneinander getrennte Datenbestände entstehen, auf die eine einheitliche Sicht geschaffen werden soll. Mit Hilfe eines Data Warehouses war es unser Ziel, diese Datenbestände durch geeignete Verfahren zu analysieren, um daraus neue Erkenntnisse zu gewinnen. Weiterhin war beabsichtigt, ursprünglich getrennte Datenbestände miteinander zu integrieren, um daraus neues Wissen zu erlangen. Den Schwerpunkt dieser Ausarbeitung bildet der Prozess der Extraktion, Transformation und des Ladens (ETL, siehe Bauer u. a. (2004)) der Produktivdaten in das Data Warehouse. Die Aspekte, welche sich mit der Analyse der Daten beschäftigen, werden im Paper von Sven Elvers vorgestellt und erläutert (siehe Elvers (Wintersemester 2005/06)).

Im nachfolgenden Text werden in einem Grundlagenteil zunächst der Prozess des ETL und eine Referenzarchitektur vorgestellt, wie sie in der aktuellen Literatur beschrieben werden (siehe Bauer u. a. (2004); Humm und Wietek (2005)). Daran schließt sich eine kurze Erläuterung des Einsatzes von Werkzeugen im Teilprojekt an, bevor die Zeitplanung der Teilgruppe Business Intelligence (BI) beschrieben wird. Danach wird auf das konkrete Vorgehen im Projekt eingegangen. Die Ausarbeitung endet klassisch mit einer Zusammenfassung und einem Ausblick. In dieser Ausarbeitung verwendete Abkürzungen werden im Glossar am Ende des Textes erläutert. Grundkenntnisse aus dem Bereich der Datenbanken, insbesondere Fachbegriffe und Fremdwörter werden vorausgesetzt. Für weitere Erläuterungen dazu siehe Heuer und Saake (2000) oder Saake u. a. (2005).

2 Grundlagen

An dieser Stelle erfolgt keine Einführung in die Grundprinzipien des Business Intelligence oder des Data Warehousing. Dazu sei auf die einschlägige Literatur verwiesen, wie Humm

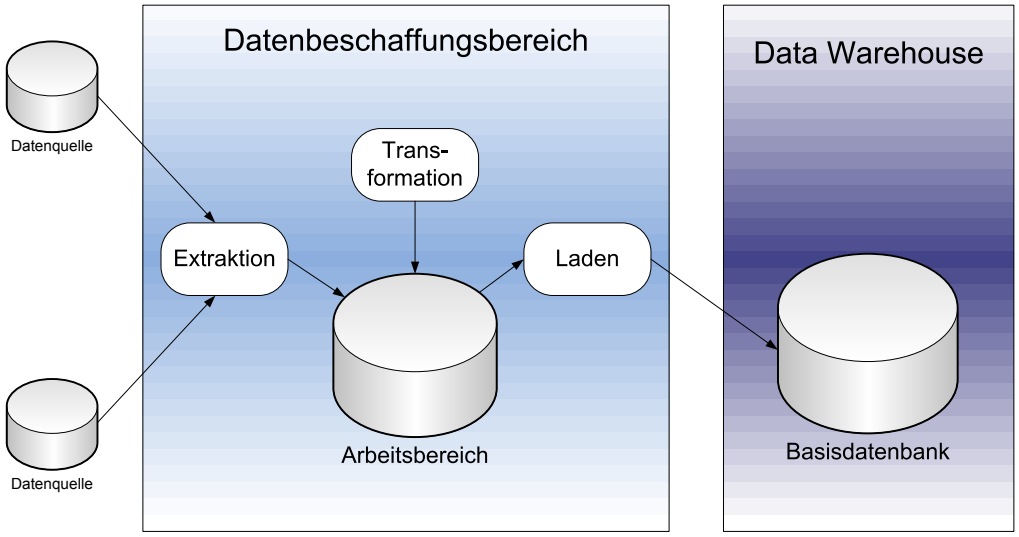

Abbildung 1: Architektur des Datenbeschaffungsbereichs nach Bauer u. a. (2004)

und Wietek (2005), Bauer u. a. (2004) oder Inmon (2005). Wenn im folgenden Text von einem Data Warehouse die Rede ist, orientieren sich die Ausführungen an der durch (Bauer u. a., 2004, S. 31) vorgestellten Referenzarchitektur.

Den Schwerpunkt der Projekttätigkeit des Autors bildete der Prozess des ETL, also der Import von Daten aus den Produktivsystemen (welche in der Literatur auch Operativsysteme genannt werden) in das Data Warehouse und die Transformation der Daten in eine analyseoptimierte Form. Im Kontext der genannten Referenzarchitektur war das Ziel also die Implementierung eines geeigneten Datenbeschaffungsbereichs.

2.1 Referenzarchitektur Datenbeschaffungsbereich

Abbildung 1 zeigt die auf unser Projekt angepasste Architektur des Datenbeschaffungsbereichs mit seinen einzelnen konzeptionellen Komponenten sowie den angrenzenden Komponenten. Es folgen kurze Beschreibungen der einzelnen Bestandteile der Architektur.

2.2 Extraktionskomponente

Sie hat Lesezugriff auf alle Datenquellen, die für die Analyse relevante Daten enthalten. Automatisiert oder manuell werden die Datenbestände aus den Produktivsystemen in den Arbeitsbereich kopiert. Qualitätsanforderungen sind normalerweise hohe Performanz und eine geringe Beeinträchtigung der Produktivsysteme durch den Kopiervorgang. Typischerweise wird die Extraktionskomponente zu Tageszeiten aktiv, in denen mit den Produktivsystemen gar nicht oder nur in geringem Maße gearbeitet wird.

Relationale Datenbanksysteme, die in den allermeisten Fällen die Datenquellen sein dürften, bieten als Schnittstellen für Anwendungen standardmäßig JDBC oder ODBC an. Beim Zugriff über JDBC/ODBC kommen jedoch Mechanismen zum Tragen, welche unter anderem

die Konsistenz der Daten sicherstellen sollen oder Sicherheitsaspekte berücksichtigen. Diese ansonsten sinnvollen oder sogar unverzichtbaren Mechanismen beeinträchtigen jedoch die Performanz der Extraktion. Beim Data Warehousing wird Letztere normalerweise als wichtigeres Designziel gesehen, wohingegen Daten- und Transaktionssicherheit nachrangig sind und an dieser Stelle durch andere Mechanismen garantiert werden können (Bauer u. a., 2004). Technisch wird dieses Ziel in der Praxis oft dadurch realisiert, dass die Extraktion nicht über die standardmäßigen Schnittstellen der jeweiligen Datenquellen abgewickelt wird. Stattdessen kommen so genannte *Bulkload*-Werkzeuge zum Einsatz, die große Datenmengen in kurzer Zeit unter Umgehung der Standardschnittstellen kopieren können.

2.3 Transformationskomponente

Nachdem die Extraktionskomponente Daten aus den Produktivsystemen in den Arbeitsbereich kopiert hat, müssen die Daten im nächsten Schritt in eine analyseoptimierte Form transformiert werden. Der Grund ist, dass die Datenmodelle in den Produktivsystemen auf den täglichen Arbeitsbetrieb hin optimiert wurden, also in Normalformen unterschiedlicher Stärke gebracht wurden, um möglichst wenig Redundanz zu haben (siehe Saake u. a. (2005)). Zur Analyse ist in vielen Fällen wieder eine Integration der Daten notwendig. Eine Normalisierung arbeitet dem entgegen.

Bei einem hohen Grad an Normalisierung werden entsprechend aufwendige und performanzlastige Verbundoperationen über den Daten notwendig. Letzteres ist ebenfalls notwendig, wenn die Produktivdaten in schlechter Art und Weise modelliert wurden und eventuell gar keiner Normalform mehr entsprechen. Dies ist in der Praxis oft der Fall, wenn Datenmodelle über Jahre hinweg gewachsen sind und bereits durch mehrere Administratoren gepflegt und gewartet wurden.

Um Verbundoperationen während der Analyse zu vermeiden, sollen sie nur einmalig in dieser Phase erfolgen. Der Sinn der Transformation ist also, die Datenbasis für die Analyse aufzubereiten. Wichtig dabei ist jedoch, dass die Daten noch nicht für eine spezielle Analyse optimiert werden, sondern möglichst nur auf eine geringstmögliche Granularität gebracht werden, damit sie Grundlage jeglicher Analyse werden können (Bauer u. a., 2004, S. 51ff).

Eine weitere Aufgabe ist, Fehler und Doubletten in der Datenbasis zu erkennen und möglichst auch zu beseitigen. Dies ist nicht immer vollautomatisch möglich, obwohl ausgefeilte Algorithmen und Mechanismen existieren (Bauer u. a., 2004, S. 83ff). Zur Auflösung und Beseitigung von Konflikten in den Daten ist dann menschlicher Eingriff notwendig. Entsprechend unklare Daten können unter Umständen erst durch einen Anwender in einen Zusammenhang miteinander gebracht werden.

2.4 Arbeitsbereich

Der Arbeitsbereich ist eine Datenbank, auf der die Operationen zur Datentransformation durchgeführt werden. Daten aus den Produktivsystemen werden hierher kopiert, wie in Abschnitt 2.3 beschrieben, transformiert und dann in die Basisdatenbank geladen. Anschließend werden die Daten im Arbeitsbereich wieder gelöscht. Sie werden hier also nur temporär für die Dauer der Transformation vorgehalten.

Der Arbeitsbereich ist konzeptionell als separate Datenbank vorgesehen. Die teilweise sehr rechenintensiven Transformationsoperationen sollen weder den Betrieb der Produktivsysteme stören, noch die Analysen auf dem eigentlichen Data Warehouse beeinträchtigen.

2.5 Ladekomponente

Die Ladekomponente sorgt dafür, dass die fertig transformierten Daten aus dem Arbeitsbereich in die Basisdatenbank geladen werden. Sie benötigt dazu schreibenden Zugriff auf die Basisdatenbank. Gleichzeitig muss sichergestellt werden, dass Daten, die dort bereits gespeichert sind, nicht mehr geändert werden. Je nach dem Umfang der zu bewegenden Datenmenge und den Anforderungen an die Performanz muss diese Komponente unter Umständen mit einem Bulkloader realisiert werden.

2.6 Basisdatenbank

In der Basisdatenbank werden die fertig transformierten Daten abgespeichert. Wie schon in Abschnitt 2.3 erläutert, liegen die Daten hier in der kleinstmöglichen Granularität vor und sind noch nicht auf eine spezielle Analyse hin optimiert. Dennoch können bereits einfache Analysen direkt auf der Basisdatenbank durchgeführt werden.

Mit der wichtigste Aspekt der Basisdatenbank ist, dass darin enthaltene Daten nicht geändert werden dürfen. Es kommen laufend neue Daten hinzu, Änderungsoperationen sind jedoch nicht erlaubt. Der Grund ist, dass nur korrekte Daten in diesem Bestand enthalten sind. Dafür soll bereits die Transformationskomponente sorgen, welche die Daten erzeugt, die hier gespeichert werden. Da alle nachfolgenden Analysen des Data Warehouse auf der Basisdatenbank basieren, muss sie der *Single Point of Truth* sein (Bauer u. a., 2004). Änderungen an diesem Bestand würden alle bereits erstellten Analysen wieder ungültig machen. Die Anwender, welche die Analysen erstellen, müssen sich jedoch darauf verlassen können, dass die Basisdatenbank nur wahre Informationen liefert.

3 Zeitplanung

Im nun folgenden Abschnitt werden die drei von der Projektleitung geforderten Meilensteine vorgestellt und beschrieben.

3.1 Meilenstein I

Das Ziel des ersten Meilensteins war das Aufsetzen einer funktionsfähigen Arbeitsumgebung. Die von uns verwendeten Entwicklungswerkzeuge, sowie die für den Betrieb des Data Warehouse notwendigen Komponenten sollten lauffähig sein. Ausführliche Erläuterungen zu den verwendeten Werkzeugen befinden sich in Abschnitt 4. Dieser Meilenstein wurde genau eingehalten.

3.2 Meilenstein II

Das Ziel dieses Meilensteins kann man mit der in der Software-Entwicklung sehr verbreiteten Redewendung *Proof of Concept* umschreiben. Anhand von beispielhaften Datenmodellen, die mit Testdaten gefüllt waren und alle miteinander integriert werden sollten, wurden die von uns gewählten Werkzeuge einem Test unterzogen. Anhand dieses Tests sollte sich zeigen, ob wir geeignete Werkzeuge nutzen und damit im Projekt weiter arbeiten können. Für diesen Meilenstein haben wir eine Woche länger benötigt, als ursprünglich vorgesehen.

Nach dem Erreichen dieses Meilensteins war klar, dass mit dem gewählten ETL-Werkzeug Octopus (OCTOPUS, 2006) und dem Datenbankmanagementsystem BizGres (BIZGRES, 2006) in zufriedenstellendem Maße gearbeitet werden kann.

3.3 Meilenstein III

Der dritte Meilenstein sah vor, die Daten eines im Ferienclub laufenden Produktivsystems mit Hilfe des Data Warehouse in ein Starschema zu transformieren und darauf anschließend Analysen durchzuführen. Die Ergebnisse der Analysen sollten in grafischer Form präsentiert werden. Dieser Meilenstein wurde genau eingehalten.

4 Einsatz von Werkzeugen

Dieser Abschnitt stellt die im Rahmen dieser Ausarbeitung relevanten Werkzeuge vor.

4.1 BizGres DBMS

BizGrez (BIZGRES, 2006) ist eine spezielle Distribution des DBMS PostgreSQL (POSTGRESQL, 2006), welche Letzteres um diverse Code-Modifizierungen und Zusatzwerkzeuge erweitert. Es wurde BizGres in der Version 0.7 verwendet[1]. Diese basiert auf PostgreSQL 8.0, enthält jedoch auch einige Funktionen, die als Neuerungen in PostgreSQL 8.1 einfließen werden.

Das verwendete DBMS bot uns viele Funktionen, die sonst nur in kommerziellen DBMS enthalten sind (siehe Saake u. a. (2005)):

- Ein umfassendes Transaktionskonzept
- Geschachtelte Abfragen (Subselects)
- Views
- Fremdschlüssel
- Trigger und Stored Procedures

[1] Siehe dazu `http://www.bizgres.org/articles/?id=3`

4.2 Enhydra Octopus

Octopus ist ein auf Java basierendes ETL-Werkzeug. Mittels XML-Konfigurationsdateien lassen sich ETL-Prozesse definieren, die dann durch Octopus abgearbeitet werden. Ein Octopus-Prozess, im Folgenden *Job* genannt, enthält diese Bestandteile:

1. Definition der Quelldatenbank (JDBC-Verbindung)
2. Definition der Zieldatenbank (JDBC-Verbindung)
3. SQL-Anweisungen zum Bereinigen des Arbeitsbereichs
4. SQL-Anweisungen zum Entfernen der Integritätsbedingungen aus dem Arbeitsbereich
5. **Eigentliche Beschreibungen der Extraktion bzw. der Transformation**
6. SQL-Anweisungen zum Wiederherstellen der Integritätsbedingungen im Arbeitsbereich

Mit Ausnahme von Punkt 5 sind diese Bestandteile sehr einfach zu verfassen, da dies größtenteils automatisiert erledigt werden kann. Aus diesem Grund wird nur dieser eine Punkt, welcher die Importdefinitionen umfasst, näher erläutert. Ein Octopus-Job kann beliebig viele Importdefinitionen umfassen.

4.2.1 Importdefinitionen

Für jede Importdefinition wird zunächst eine Datenquelle angegeben. Dies kann entweder eine Tabelle sein, ein View oder eine SQL-Abfrage. Die Datenquelle muss zu der zuvor definierten Quelldatenbank gehören.

4.2.2 Value-Columns

Für jede Importdefinition lassen sich *Value-Columns* definieren. Eine *Value-Column* definiert eine Zuordnung einer Spalte aus der Datenquelle dieser Importdefinition auf eine beliebige Spalte einer beliebigen Tabelle innerhalb der Zieldatenbank. Diese Zuordnung wird später bei einfachen 1-zu-1-Kopieroperationen berücksichtigt.

4.2.3 Zieltabellen

Zu jeder Importdefinition gehört eine Liste von Zieltabellen. In den Zieltabellen werden die Tabellen in der Zieldatenbank definiert, die bei einer 1-zu-1-Kopieroperation berücksichtigt werden sollen. Wie die Kopieroperation im Einzelnen durchgeführt wird, ist in den *Value-Columns* beschrieben. Es werden nur die Tabellen gefüllt, die hier in den Zieltabellen auftauchen und für die eine Zuordnung in den *Value-Columns* existiert.

4.2.4 Transformationen

Importdefinitionen können beliebig viele Transformationen beinhalten. Jede Transformation hat Quellspalten, die aus der Datenquelle der Importdefinition stammen. Zielspalten können beliebige Spalten aus beliebigen Tabellen der Zieldatenbank sein. Octopus bietet einige vorgefertigte Transformationsoperationen an. Es ist jedoch auch möglich, eine Java-Klasse anzugeben, die das Interface `org.webdocwf.util.loader.Transformer` implementiert. Mit einer solchen Transformerklasse lässt sich beinahe jede Transformation durchführen, da sämtliche Möglichkeiten der Programmiersprache Java genutzt werden können.

4.2.5 Counter-Columns

Das Prinzip der Counter-Columns sieht vor, dass innerhalb einer Tabelle der Zieldatenbank alle Zähler verwaltet werden. Eine solche Counter-Tabelle besteht aus zwei Spalten: Countername und Counterwert. Jedes Tupel dieser Tabelle ist einer Tabelle aus der Basisdatenbank zugeordnet. Beim Einfügen eines neuen Tupels in eine der Tabellen der Basisdatenbank wird der Zähler für die betreffende Tabelle durch Octopus entsprechend inkrementiert. Auf diese Weise kann der Countertabelle entnommen werden, wieviele Tupel die einzelnen Tabellen der Basisdatenbank enthalten, ohne dort entsprechend aufwendige Abfragen mit dem `count()`-Operator durchführen zu müssen.

4.3 JDBC-Treiber und Bulkloader

Im Teilprojekt wurden JDBC-Treiber eingesetzt, um unter anderem Octopus (siehe Abschnitt 4.2), der Eclipse Entwicklungsumgebung und dem Aqua Data Studio den Zugriff auf die Datenbanken des Ferienclub-Projekts zu ermöglichen. Einsatz fanden der PostgreSQL-JDBC-Treiber (POSTGRESQL, 2006) und jTDS (JTDS, 2006). Letzterer ermöglicht den JDBC-Zugriff auf den Microsoft SQL Server, welcher von der Teilgruppe *Location Based Services* eingesetzt wurde (siehe Thomé (Wintersemester 2005/06)).

Wird bei der Extraktion mit sehr großen Datenmengen gearbeitet, so bringen Bulkload-Werzeuge einen Geschwindigkeitsvorteil (siehe Abschnitt 2.2 und Bauer u. a. (2004)). Bei den Datenmengen, mit denen im Projekt zu rechnen war (einige Tausend Testdatensätze), hätte ein Bulkloader jedoch keinen nennenswerten Geschwindigkeitsvorteil gebracht. Die Ladezeiten der Werkzeuge, die den Datenbankzugriff per JDBC realisieren, waren akzeptabel. In Anbetracht des Zeitaufwands für die Einarbeitung in ein Bulkload-Werkzeug, wurde darauf verzichtet, ein solches einzusetzen. Stattdessen wurden sämtliche ETL-Operationen mittels JDBC-Verbindungen und Octopus realisiert.

4.4 Eclipse Entwicklungsumgebung

Für die Implementierung von Java-Klassen und der Job-Beschreibungen für Octopus wurde die Eclipse Entwicklungsumgebung eingesetzt (ECLIPSE, 2006). Es wurden weiterhin Eclipse-Plugins verwendet für das Syntax-Highlighting von XML und für den Zugriff auf die verwendeten Datenbanken. Siehe dazu XMLBUDDY (2006) und QUANTUM (2006). Zur Automa-

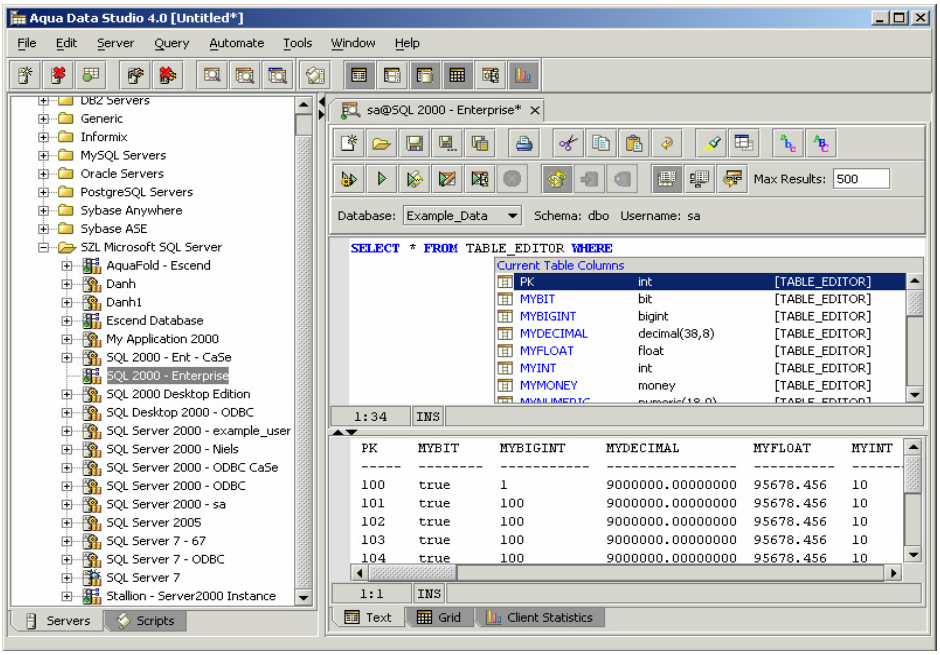

Abbildung 2: Screenshot der Benutzeroberfläche des Aqua Data Studios

tisierung der Kompilierungsvorgänge wurde das in Eclipse integrierte Apache Ant verwendet. Siehe dazu ANT (2006).

4.5 Aqua Data Studio

Das Aqua Data Studio ist ein in Java implementiertes Datenbank-Administrationswerkzeug. Mittels JDBC kann es auf so gut wie alle DBMS, proprietär oder opensource, zugreifen (AQUADS, 2006). Im Teilprojekt BI wurde es hauptsächlich zum Entwickeln und Implementieren der Datenmodelle für das Data Warehouse und zum Generieren und Debuggen von SQL-Abfragen eingesetzt. Abbildung 2 auf Seite 11 zeigt einen Screenshot der Benutzeroberfläche der Anwendung.

4.6 Fedora Linux

Auf den vom Teilprojekt BI verwendeten Rechnern wurde als Betriebssystem die freie Linux-Distribution Fedora Linux installiert. Siehe FEDORA (2006).

5 Vorgehen im Projekt

In den folgenden Abschnitten wird das konkrete Vorgehen der Teilgruppe BI im Projekt erläutert. Dabei sollen insbesondere die Probleme herausgearbeitet werden, die im Verlauf des Projekts auftraten und wie diese gelöst wurden. Verweise auf erstellten Quellcode werden als

Pfade angegeben, die innerhalb der Archivdatei `dwh.zip` angelegt wurden. Diese Datei kann unter `http://www.weinschenker.name/dwh/dwh.zip` heruntergeladen werden.

5.1 Einsatz von Octopus

Nachdem der erste Meilenstein erfolgreich erreicht wurde (siehe Abschnitt 3.1), konnten die ersten Tests des ETL-Tools Octopus beginnen. Zu diesem Zweck wurden drei unterschiedliche Datenmodelle entworfen, in denen Adressdaten gespeichert waren. Im nächsten Schritt sollten miteinander integriert werden.

Das Vorgehen mit Octopus beginnt in diesem Fall damit, dass es per JDBC-Verbindung Zugriff auf die Tabellen enthält und zunächst die Tabellenstruktur in den Quelldaten extrahiert und sogenannte *Skeleton-Jobs* erstellt. Diese generierten Jobs sind bereits dazu in der Lage, 1-zu-1-Kopieroperationen durchzuführen. Sie erstellen dazu die notwendigen SQL-Skripte und Importdefinitonen, welche von der Syntax her fast komplett sowohl an die Quell- als auch an die Zieldatenbank angepasst sind.

Probleme bereitete die Tatsache, dass das Schema-Konzept für Tabellennamen aus PostgreSQL dabei nicht berücksichtigt wird. Sollen Tabellen beispielsweise im Schema `workspace` erstellt werden, so müssen sie in allen SQL-Anweisungen folgendermaßen angesprochen werden: `workspace.tabelle1` oder `workspace.tabelle2`. Von Octopus erstellte Skripte enthalten jedoch keine Schemanamen, weswegen sie in allen erzeugten Dateien von Hand nachgetragen werden müssen.

5.2 Probleme mit JDBC-Treibern

Beim Kopieren von Daten des SQL-Datentyps Timestamp zeigt sich, dass unterschiedliche JDBC-Implementierungen damit Probleme haben. Kopiervorgänge aus dem Microsoft SQL Server in die PostgreSQL-Datenbank, bei denen zwei unterschiedliche JDBC-Treiber beteiligt sind, sind unproblematisch. Bei Kopiervorgängen innerhalb von PostgreSQL gehen jedoch die Uhrzeit-Informationen des Timestamps verloren. Ein Grund dafür war nicht feststellbar. Ebensowenig konnte dieser Fehler auf einfache Weise umgangen werden. Er wurde vom Projektteam als nicht kritisch eingestuft und zunächst ignoriert.

5.3 Implementierung der ETL-Jobs

Nachdem im Teilprojekt *Location Based Services* ein erstes durch das Teilprojekt BI verwendbares Datenmodell entworfen wurde, welches auch bis zum Ende des Gesamtprojekts bestehen bleiben würde, sollte dieses die Grundlage für das weitere Vorgehen werden. Abbildung 3 auf Seite 13 zeigt das Entitiy-Relationship-Modell des *Pocket-Task-Timers* (im Folgenden mit PTT abgekürzt). Für weitere Informationen zu diesem Teilprojekt siehe das entsprechende Paper von Mark Thomé (Thomé, Wintersemester 2005/06). Erläuterungen zu Entity-Relationship-Modellen im Allgemeinen finden sich in Heuer und Saake (2000) und Saake u. a. (2005).

Da zu diesem Zeitpunkt noch kein weiteres Datenmodell einer anderen Teilgruppe verfügbar gewesen ist, wurde zunächst ein Starschema für die PTT-Daten entworfen, ohne dabei eine Integration mit anderen Datenbeständen aus dem Ferienclub zu implementieren. Dies sollte erfolgen, sobald entsprechende Daten zur Verfügung stünden. Für Erläuterungen zum Thema

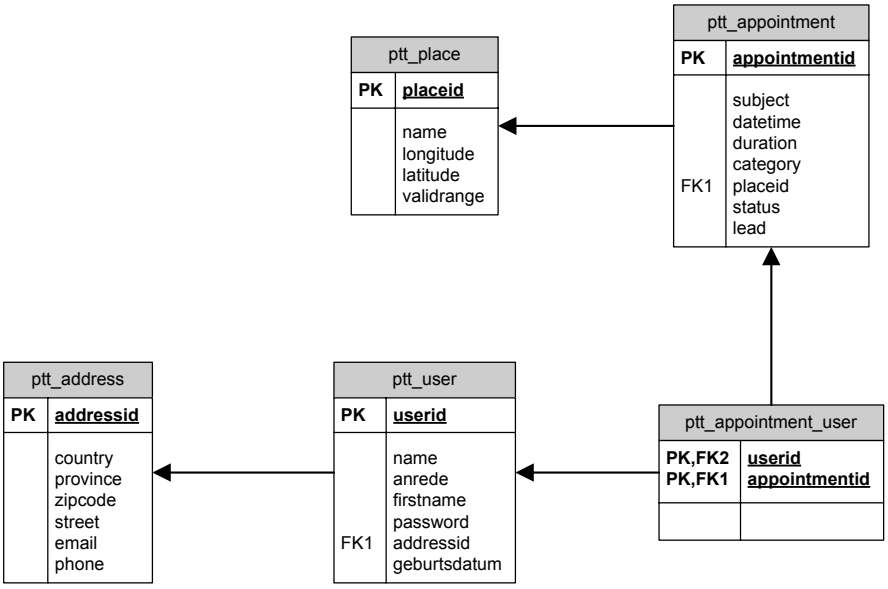

Abbildung 3: Entitiy-Relationship-Modell des Daten des *Pocket-Task-Timers*

Starschema siehe Bauer u. a. (2004). Abbildung 4 auf Seite 14 zeigt das entsprechende Starschema.

Für die Transformation wurden Importdefinitionen (siehe Abschnitt 4.2.1) verfasst, deren Funktionsweise im nachfolgenden Text erklärt wird. Der Quellcode dieser Definition befindet sich in der Datei `DWH/etljobs/transform/xml/ImportDefinition.oli`.

5.3.1 Import der Dimensionsdaten

Dimensionsdaten werden nur dann importiert, wenn sie neu sind und noch nicht in den Dimensionstabellen enthalten sind. Die entsprechende Prüfung, ob die zu importierenden Daten neu sind, kann über eine SQL-Anfrage realisiert werden, die für die Importdefinition als Quelltabelle angegeben wird. Beispielhaft sei hier der Import neuer Kategorien genannt. Alle anderen Dimensionen werden auf ähnliche Weise importiert und transformiert.

Listing 1: SQL-Anfrage für neue Kategorien

```
SELECT DISTINCT category
        FROM arbeitsbereich_esb.ptt_appointment
        WHERE category NOT IN
                (SELECT DISTINCT category
                        FROM basedb_ptt.ptt_dimension_categories)
```

Listing 1 zeigt die SQL-Anfrage separat und Listing 2 auf Seite 17 (Anhang A) die komplette Importdefinition für die Kategorien. Die Anfrage verwendet sowohl den `Distinct`-Operator als auch das `NOT IN` Konstrukt, um zu verhindern, dass Doubletten importiert werden.

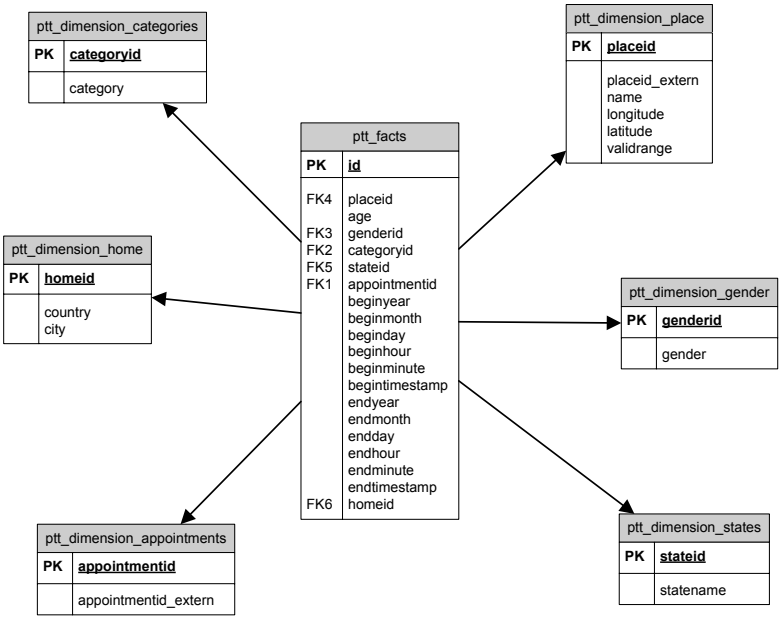

Abbildung 4: Starschema des *Pocket-Task-Timers* als Grundlage für Analysen

5.3.2 Import der Faktentabelle

Beim Importieren der Dimensionsdaten wurde zwar schon kein einfacher 1-zu-1-Import mehr durchgeführt, dennoch war dieser Vorgang relativ leicht zu implementieren. Die Faktentabelle hat jedoch mehrere Quelltabellen, deren Daten nun in einer Einzigen zusammengefasst werden müssen. Dabei muss der semantische Zusammenhang der einzelnen Fakten erhalten bleiben. Gleichzeitig müssen die Verweise auf die einzelnen Dimensionseinträge erhalten bleiben, wobei einige Dimensionseinträge bereits vor dem Fakt ins Data Warehouse importiert worden sein können.

Beispielsweise soll der Datensatz einer Verabredung von Person *A* mit Person *B* an Ort *C* aus dem PTT-Schema importiert werden. Die beiden Personen und der Ort existieren bereits im Data Warehouse. Dies ist immer der Fall, da Personen und Orte im vorangegangenen Schritt als Dimensionen bereits importiert wurden. Person *A* ist dem Data Warehouse als *X* bekannt, Person *B* als *Y* und Ort *C* als *Z*. Der neue Datensatz muss mit den Data-Warehouse-Schlüsseln *X*, *Y* und *Z* angelegt werden. Dazu ermittelt das ETL-Tool die entsprechenden Abbildungen der PTT-Schlüssel auf die Data-Warehouse-Schlüssel und fügt dann Letztere als neuen Fakten-Datensatz in die Faktentabelle ein.

Realisiert wird dieser Schritt, indem über dem PTT-Schema ein View implementiert wird, der die Abbildung auf das Starschema durchführt. Dieser View bildet also die PTT-Daten bereits so ab, dass ihre Struktur mit der der Faktentabelle des Data Warehouse identisch ist. Mit Unterabfragen werden die korrekten Fremdschlüssel zu den Dimensionstabellen ermittelt. Das ETL-Tool nutzt diesen View indirekt als Quelle zum Füllen der Faktentabelle und kann diesen Vorgang als vermeintlichen 1-zu-1-Kopiervorgang erledigen, jedoch auf Basis des View. Für die SQL-Definition des View, siehe Anhang B.

Um Doubletten in der Faktentabelle zu vermeiden, wurde die Faktentabelle um eine Spalte `identcode` erweitert. Diese Spalte hat den Datentyp `varchar(32)`. Über jeden neuen Datensatz in der Faktentabelle wird ein MD5-Hash berechnet und in `identcode` gespeichert. Für jeden neu einzufügenden Datensatz wird ebenfalls ein MD5-Hash berechnet. Neue Datensätze werden nur dann eingefügt, wenn ihr MD5-Hash noch nicht in der Spalte `identcode` enthalten ist. Dies wird durch den View `ptt_import_view_without_doublettes` realisiert, welcher in der Datei `DWH/etljobs/extern_ptt/sql/CreateTables.sql` definiert wird. Dieser View setzt auf dem View aus dem vorherigen Absatz auf, er liefert ebenfalls die mit der Faktentabelle kompatible Datenstruktur und dient dem ETL-Tool als direkte Kopiervorlage.

6 Zusammenfassung und Ausblick

Das Ziel des Teilprojekts war, Business Intelligence im Ferienclub zu ermöglichen. Die gesteckten Ziele wurden verwirklicht. Wir hätten gerne zusätzliche Ziele realisiert. Dies konnte jedoch aus zeitlichen Gründen nicht erreicht werden, die Gründe dafür wurden bereits in Diskussionsrunden mit Professoren und Studenten erörtert und sollen hier nicht nochmals ausgeführt werden.

Im Rahmen dieser Ausarbeitung konnte der komplette Arbeitsschritt eines ETL-Prozesses umgesetzt werden. Vom Import über Transformation bis zum Laden der Daten in einen Bereich der Datenbank, der von den Analysewerkzeugen genutzt wird. Dabei diente das Datenbankschema der Teilgruppe *Location Based Services* als Grundlage (Thomé, Wintersemester 2005/06). Die Einbindung weiterer Schemata wurde nicht vorgenommen, da diese erst spät im Projekt verfügbar waren und die Ressource Zeit in der Teilgruppe BI ebenfalls knapp wurde, insbesondere zum Ende des Projekts.

Die Auswahl der verwendeten Werkzeuge stellte sich nicht als falsch heraus. Insbesondere das ETL-Tool Octopus (OCTOPUS, 2006) war eine gute Wahl. Auch wenn es den Funktionsumfang vergleichbarer kommerzieller Werkzeuge noch nicht erreicht, war es doch umfassend dokumentiert. Auch waren keine Bugs oder ähnliche Fehler aufgetreten. Zusatzwerkzeuge erledigen einen Großteil des Verfassens der XML-Quellcodes automatisch, was eine große Arbeitserleichterung bedeutete. Das verwendete Datenbankmanagementsystem PostgreSQL (POSTGRESQL, 2006) ließ beim Projektteam ebenfalls keine Wünsche offen.

Was der Autor gerne noch weiter entwickelt hätte, wären eine ausgefeiltere Erkennung von falschen Daten oder eine weiter gehende Erkennung von Doubletten im Datenbestand. Anbieten würde sich für diese Anforderung eine regelbasierte Anwendung. Entsprechende Werkzeuge scheinen, jedenfalls unter einer freien Lizenz, noch nicht in großem Umfang zu existieren. Ein auf der Arbeit des Teilprojekts aufbauendes Nachfolgeprojekt könnte untersuchen, ob eine entsprechende Anwendung mit vertretbarem Aufwand selbst erstellt werden könnte.

Weiterhin bietet es sich an, die Struktur der Daten in der Basisdatenbank für spezielle Analysen weiter zu optimieren. Für spezielle Analysen könnte die derzeitige Struktur zu so genannten Data-Marts weiter entwickelt werden (siehe Bauer u. a. (2004)). Dies würde insbesondere dann Sinn machen, sobald mehr als eine externe Datenquelle angezapft und importiert wird. Das in seiner jetzigen Form vorliegende Data Warehouse bietet dazu eine gute Grundlage.

Literatur

[ANT 2006] *Apache Ant*. `http://ant.apache.org/`, Februar 2006. – Die Homepage des Apache Ant Projekts

[AQUADS 2006] *Aqua Data Studio*. `http://www.aquafold.com/`, Februar 2006. – Die Homepage des Aqua Data Studio

[BIZGRES 2006] *BizGres*. `http://bizgres.org/`, Februar 2006. – Ein relationales Datenbankmanagementsystem

[ECLIPSE 2006] *The Eclipse Project*. `http://www.eclipse.org/`, Februar 2006. – Die Homepage des Eclipse Projekts

[OCTOPUS 2006] *Enhydra Octopus*. `http://octopus.objectweb.org/`, Februar 2006. – Ein Java-basiertes ETL-Tool

[FEDORA 2006] *Fedora Linux Project*. `http://fedora.redhat.com/`, Februar 2006. – Die Homepage des Fedora Linux Projekts

[JTDS 2006] *The jTDS Project*. `http://jtds.sourceforge.net/`, Februar 2006. – Eine JDBC-Implementierung, die den Zugriff auf den Microsoft SQL Server ermöglicht.

[POSTGRESQL 2006] *PostgreSQL*. `http://www.postgresql.org/`, Februar 2006. – Ein relationales Datenbankmanagementsystem

[QUANTUM 2006] *QuantumDB Eclipse Plugin*. `http://quantum.sourceforge.net/`, Februar 2006. – Die Homepage des QuantumDB Eclipse Plugins

[XMLBUDDY 2006] *XMLBuddy*. `http://xmlbuddy.com/`, Februar 2006. – Das XMLBuddy Eclipse Plugin

[Bauer u. a. 2004] BAUER, Andreas ; GÜNZEL, Holger u. a.: *Data Warehouse Systeme*. 2. Auflage. Heidelberg : dpunkt.verlag, Juni 2004

[Elvers Wintersemester 2005/06] ELVERS, Sven: Projekt Ferienclub — Business Intelligence: Analyse / HAW Hamburg — Masterprogramm Verteilte Systeme. Wintersemester 2005/06. – Forschungsbericht

[Heuer und Saake 2000] HEUER, Andreas ; SAAKE, Gunter: *Datenbanken: Konzepte und Sprachen*. 2. Auflage. Mitp-Verlag, 2000

[Humm und Wietek 2005] HUMM, Bernhard ; WIETEK, Frank: Architektur von Data Warehouses und Business Intelligence Systemen. In: *Informatik Spektrum* 28 (2005), Februar, Nr. 1, S. 3–14

[Inmon 2005] INMON, William H.: *Building the Data Warehouse*. 4. Auflage. New York : John Wiley and Sons, Oktober 2005

[Saake u. a. 2005] SAAKE, Gunter ; HEUER, Andreas ; SATTLER, Kai-Uwe: *Datenbanken: Implementierungstechniken*. 2. Auflage. Mitp-Verlag, 2005

[Thomé Wintersemester 2005/06] THOMÉ, Mark: Pocket Task Timer - A personal approach on Location-Based Services / HAW Hamburg — Masterprogramm Verteilte Systeme. Wintersemester 2005/06. – Forschungsbericht

Glossar

BI: In dieser Ausarbeitung die Abkürzung für Business Intelligence

DBMS: Abkürzung für *Datenbankmanagementsystem*. Siehe Heuer und Saake (2000); Saake u. a. (2005)

ETL: Extraktion, Transformation, Laden. (Bauer u. a., 2004)

HAW: Hochschule für angewandte Wissenschaften

JDBC: Java Database Connectivity. Eine Schnittstelle zum Zugriff auf Datenbanken. (Heuer und Saake, 2000; Saake u. a., 2005)

ODBC: Open Database Connectivity. Eine Schnittstelle zum Zugriff auf Datenbanken. (Heuer und Saake, 2000; Saake u. a., 2005)

PTT: Pocket Task Timer. Ein weiteres Teilprojekt im Projekt *Ferienclub*.

Referenzarchitektur: Im Kontext dieser Ausarbeitung ist mit Referenzarchitektur die Architektur für Data Warehouses nach (Bauer u. a., 2004, S. 31) gemeint.

SQL: Structured Query Language. Eine Abfragesprache für relationale Datenbanken. Siehe Heuer und Saake (2000); Saake u. a. (2005).

A Quellcode: Importdefinition für Kategorien

Listing 2: Importdefinition für Kategorien

```
<importDefinition name="PTT_DIMENSION_CATEGORIES"
    selectStatement="SELECT␣DISTINCT␣category
␣␣␣␣␣␣␣␣FROM␣arbeitsbereich_esb.ptt_appointment
␣␣␣␣␣␣␣␣WHERE␣category␣NOT␣IN
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(SELECT␣DISTINCT␣category
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣FROM␣basedb_ptt.ptt_dimension_categories)">
    <valueColumns>
        <valueColumn sourceColumnName="category"
            targetColumnName="category" targetTableID="0"
            targetTableName="basedb_ptt.ptt_dimension_categories"
```

```
            valueMode="SetIfCreated"/>
    </valueColumns>

    <counterColumns
        counterTableName="basedb_ptt.countertable"
        counterNameColumn="countername"
        counterValueColumn="countervalue">

        <counterColumn counterName="categories"
            counterStartValueReset="false" valueMode="SetIfCreated"
            targetTableName="basedb_ptt.ptt_dimension_categories"
            targetColumnName="categoryid" targetTableID="0"/>

    </counterColumns>

    <tables>
        <table insert="true" oidLogic="false"
            tableID="0" tableMode="Query"
            tableName="basedb_ptt.ptt_dimension_categories"/>
    </tables>
</importDefinition>
```

B Quellcode: VIEW und SQL-Anfrage für Import der Faktentabelle

Listing 3: SQL-Anfrage für Import der Faktentabelle

```
CREATE OR REPLACE VIEW basedb_ptt.ptt_import_view
AS
SELECT
  (SELECT placeid FROM basedb_ptt.ptt_dimension_place
    WHERE arbeitsbereich_esb.ptt_place.placeid =
    basedb_ptt.ptt_dimension_place.placeid_extern) AS placeid,
  EXTRACT('year' FROM AGE(arbeitsbereich_esb.ptt_user.geburtsdatum))
    AS age,
  (SELECT genderid FROM basedb_ptt.ptt_dimension_gender
    WHERE arbeitsbereich_esb.ptt_user.anrede =
    basedb_ptt.ptt_dimension_gender.gender) AS genderid,
  (SELECT categoryid FROM basedb_ptt.ptt_dimension_categories
    WHERE arbeitsbereich_esb.ptt_appointment.category =
    basedb_ptt.ptt_dimension_categories.category) AS categoryid,
  (SELECT stateid FROM basedb_ptt.ptt_dimension_states
    WHERE arbeitsbereich_esb.ptt_appointment.status =
    basedb_ptt.ptt_dimension_states.statename) AS stateid,
  (SELECT appointmentid FROM basedb_ptt.ptt_dimension_appointments
    WHERE arbeitsbereich_esb.ptt_appointment.appointmentid =
    basedb_ptt.ptt_dimension_appointments.appointmentid_extern)
```

```
    AS appointmentid,
  EXTRACT(YEAR FROM arbeitsbereich_esb.ptt_appointment.datetime)
    AS beginyear,
  EXTRACT(MONTH FROM arbeitsbereich_esb.ptt_appointment.datetime)
    AS beginmonth,
  EXTRACT(DAY FROM arbeitsbereich_esb.ptt_appointment.datetime)
    AS beginday,
  EXTRACT(HOUR FROM arbeitsbereich_esb.ptt_appointment.datetime)
    AS beginhour,
  EXTRACT(MINUTE FROM arbeitsbereich_esb.ptt_appointment.datetime)
    AS beginminute,
  arbeitsbereich_esb.ptt_appointment.datetime
    AS begintimestamp,
  EXTRACT(YEAR FROM arbeitsbereich_esb.ptt_appointment.datetime
    + CAST(arbeitsbereich_esb.ptt_appointment.duration
      || ' hours' AS INTERVAL))
    AS endyear ,
  EXTRACT(MONTH FROM arbeitsbereich_esb.ptt_appointment.datetime
    + CAST(arbeitsbereich_esb.ptt_appointment.duration
      || ' hours' AS INTERVAL))
    AS endmonth ,
  EXTRACT(DAY FROM arbeitsbereich_esb.ptt_appointment.datetime
    + CAST(arbeitsbereich_esb.ptt_appointment.duration
      || ' hours' AS INTERVAL))
    AS endday ,
  EXTRACT(HOUR FROM arbeitsbereich_esb.ptt_appointment.datetime
    + CAST(arbeitsbereich_esb.ptt_appointment.duration
      || ' hours' AS INTERVAL))
    AS endhour ,
  EXTRACT(MINUTE FROM arbeitsbereich_esb.ptt_appointment.datetime
    + CAST(arbeitsbereich_esb.ptt_appointment.duration
      || ' hours' AS INTERVAL))
    AS endminute ,
  arbeitsbereich_esb.ptt_appointment.datetime
    + CAST(arbeitsbereich_esb.ptt_appointment.duration
      || ' hours' AS INTERVAL)
    AS endtimestamp,
  (SELECT homeid FROM basedb_ptt.ptt_dimension_home
    WHERE arbeitsbereich_esb.ptt_address.country
      || arbeitsbereich_esb.ptt_address.province =
    basedb_ptt.ptt_dimension_home.country
      || basedb_ptt.ptt_dimension_home.city)
    AS homeid FROM arbeitsbereich_esb.ptt_address

RIGHT JOIN (
  (
    (
```

```
        arbeitsbereich_esb.ptt_appointment
       RIGHT JOIN arbeitsbereich_esb.ptt_appointment_user
       ON arbeitsbereich_esb.ptt_appointment.appointmentid =
        arbeitsbereich_esb.ptt_appointment_user.appointmentid
     )
       LEFT JOIN arbeitsbereich_esb.ptt_place
       ON arbeitsbereich_esb.ptt_appointment.placeid =
        arbeitsbereich_esb.ptt_place.placeid
  )
    LEFT JOIN arbeitsbereich_esb.ptt_user
    ON arbeitsbereich_esb.ptt_appointment_user.userid =
     arbeitsbereich_esb.ptt_user.userid
)
ON arbeitsbereich_esb.ptt_address.addressid =
arbeitsbereich_esb.ptt_user.addressid;
```

C Inhalt der Archiv-Datei dwh.zip

Die Datei `dwh.zip` ist ein gepacktes Eclipse-Projekt, welches direkt in den Workspace einer Eclipse 3.1 Anwendung entpackt werden kann. Die folgende Liste zeigt alle relevanten Verzeichnisse oder Dateien des Projekts. Hier nicht aufgeführter Inhalt der Archivdatei ist für das Projekt nicht relevant. Diese Datei kann unter `http://www.weinschenker.name/dwh/dwh.zip` runtergeladen werden.

- `src/etl/` – Verzeichnis mit Java-Quellcodes
  - `Connect.java` – Diese Klasse schreibt Testdaten in die Datenbank
  - `DatumTransformer.java` – Gleicht einen Kopierfehler von Microsoft SQL-Server nach PostgreSQL aus. Siehe Importdefinition in Datei `DWH/etljobs/extern_ptt/xml/ImportDefinition.oli`
  - `Person.java` – Klasse Person. Wird beim Erstellen von Testdaten benutzt.
  - `TestdataGenerator.java` – Generiert Testdaten, die durch die Connect-Klasse in die Datenbank geschrieben werden.
  - `TestJob.java` – Mittlerweile kein *Test* mehr. Diese Klasse startet die ETL-Jobs.
- `src/etl/transformation/` – Verzeichnis mit Java-Quellcodes
  - `TestTransformer.java` – Diese Klasse wurde nur zu Testzwecken angelegt und erfüllt keinen tieferen Sinn.
- `etlbuild/` – Zielverzeichnis von Apache-Ant. Enthält eine auf den Linux-Rechnern des Teilprojekts BI lauffähige JAR-Datei mit den ETL-Jobs.
- `etljobs/extern_ptt/` – Verzeichnis mit der Definition des ETL-Jobs für den Import der externen Daten

  - `SQL/` Verzeichnis mit den durch diesen Job verwendeten SQL-Skripten
  - `XML/` Verzeichnis mit der durch diesen Job verwendeten Importdefinition und der Logging-Konfiguration
  - `LoaderJob_MS.olj` Zentrale Datei mit der Definition des ETL-Jobs
  - `LoaderJob_PS_PS.olj` Irrelevant, wird nicht verwendet
  - `LoaderJob.olj` Irrelevant, wird nicht verwendet

- `etljobs/training/` – Verzeichnis mit der Definition eines ETL-Jobs, der für Testzwecke verwendet wurde. Für das Funktionieren der Anwendung ist dieser Job ohne Belang.

- `etljobs/transform/` – Verzeichnis mit der Definition des ETL-Jobs für die Transformation der externen Daten in eine analyseoptimierte Form

  - `SQL/` Verzeichnis mit den durch diesen Job verwendeten SQL-Skripten
  - `XML/` Verzeichnis mit der durch diesen Job verwendeten Importdefinition und der Logging-Konfiguration
  - `LoaderJob_MS.olj` Zentrale Datei mit der Definition des ETL-Jobs

- `lib/` – Verzeichnis mit allen benötigten Java-Bibliotheken

- `log/` – Verzeichnis mit den Logdateien der Octopus-Jobs

- `ObjectLoader/` – Verzeichnis den Logdateien eines Octopus-Hilfsprogramms