Dokumentation

ETL-Prozess: Bugzilla-Palo

Forschungsprojekt MIS206

Michael Vitz

Christian ter Stein

viadee®
IT-Unternehmensberatung

Hochschule Niederrhein
University of Applied Sciences
Wirtschaftswissenschaften
Faculty of Business Administration and Economics

Inhaltsverzeichnis

1 Einleitung ... 1
1.1 Dimensionen ... 1
1.1.1 Entwickler ... 1
1.1.2 Produkt ... 1
1.1.3 Zeit ... 1
1.1.4 Offene Bugs ... 1
1.2 Würfel ... 2
1.2.1 Aufwand-Ist ... 2
1.2.2 Aufwand-Plan ... 2
1.2.3 Bugs-Offen ... 2
1.3 Verwendete Software ... 2
1.4 Annahmen ... 3
2 Installation & Durchführung ... 4
2.1 Projekt importieren ... 4
2.2 Bugzilla Connection ändern ... 4
2.3 Palo Connection ändern ... 4
2.4 Durchführung des ETL-Prozesses ... 4
3 ETL-Prozess ... 5
3.1 Quell- und Zielsysteme ... 5
3.1.1 Bugzilla ... 5
3.1.2 Palo ... 5
3.2 Extraktion von logischen Basiseinheiten ... 6
3.2.1 Products ... 6
3.2.2 Developers ... 8
3.2.3 Priorities ... 9
3.2.4 BugStatus ... 9
3.2.5 Bugs ... 10
3.2.6 OpenBugs ... 15
3.2.7 Activities ... 15
3.2.8 EffortActivities ... 18
3.2.9 Times ... 20
3.3 Erstellung der Dimensionen ... 20
3.3.1 Entwickler ... 20
3.3.2 Produkt ... 21
3.3.3 Zeit ... 22
3.3.4 Offene Bugs ... 23

3.4 Erstellung der Würfel 24
3.4.1 Aufwand-Ist 24
3.4.2 Aufwand-Plan 25
3.4.3 Bugs-Offen 26
3.5 Vorhandene Jobs 27

1 Einleitung

Ziel des hier vorgestellten Prozesses ist es, eine Auswertungsmöglichkeit der im Bugtracking-Tool Bugzilla enthaltenen Daten mit Hilfe des Palo OLAP-Server zu ermöglichen. Dazu erfolgt die Erstellung eines Connectors (ETL-Prozess) zur Befüllung des Palo OLAP-Servers. Es werden jedoch keine Änderungen am Bugzilla Datenmodell sowie eine Anreicherung der Daten durch externe Quellen vorgenommen.

1.1 Dimensionen

Am Ende des ETL-Prozesses befinden sich in Palo insgesamt vier Dimensionen, die von den Würfeln verwendet werden können. Im folgenden werden die Dimensionen genannt und kurz erläutert.

1.1.1 Entwickler

Die Dimension Entwickler entspricht 1:1 dem Entwickler Konzept Bugzillas. Für die im späteren Verlauf beschriebenen Würfel sind jedoch nur der Name des Entwicklers von Belang, weshalb auch nur dieser in der Dimension verwendet wird.

1.1.2 Produkt

Produkt als Dimension bildet die in Bugzilla vorhandenen Produkte ab. Diese entsprechen zumeist einem Projekt und enthält alle Bugs, die bei diesem Produkt aufgetreten sind. Weiterhin werden die verschiedenen Versionen, die man einem Produkt in Bugzilla zuweisen kann, hierarchisch unter das Produkt gegliedert.

1.1.3 Zeit

Die Zeit als Dimension bezieht sich in diesem Falle auf Tage als niedrigste Einheit. Darüber befinden sich Jahre und Monate als Aggregationsmöglichkeit. Dabei werden nicht alle Tage in die Dimension geladen, sondern lediglich Tage, an denen es Aufwand für mindestens einen Bug gab.

1.1.4 Offene Bugs

Die Dimension offenen Bugs lädt alle zum Zeitpunkt der Durchführung des ETL-Prozesses Bugs, die noch nicht geschlossen wurden. Dabei werden diese vorher nach ihrer Priorität gegliedert und damit in eine Hierarchie gebracht.

1.2 Würfel

Bei den zu füllenden Würfeln handelt es sich um:

1.2.1 Aufwand-Ist

Hier werden die durch Entwickler getätigten Aufwände hinterlegt. Als Dimensionen kommen hierzu Entwickler, Zeit und Produkt zum Einsatz.

1.2.2 Aufwand-Plan

Analog zum Aufwand-Ist Würfel werden hier alle geplanten Aufwände geladen. Da sich der Planaufwand nicht einem gezielten Tag zuweisen lässt, sind hier als Dimensionen lediglich Entwickler und Produkt vertreten.

1.2.3 Bugs-Offen

Hier werden alle noch offenen Bugs in den Würfel geladen. Als Dimensionen besitzt der Würfel „Offene Bugs“ und Entwickler.

1.3 Verwendete Software

Der vorliegende ETL-Prozess stützt sich auf die Bugzilla Version 3.2.4 und dessen Schema.
Änderungen am Bugzilla SQL Schema , die in späteren Versionen folgend könnten, können unter http://www.ravenbrook.com/project/p4dti/tool/cgi/bugzilla-schema/ eingesehen werden.

Zur Umsetzung des ETL-Prozesses wurde der PALO-ETL-Server[1] genutzt. Dieser ermöglicht die Extraktion, Transformation und das Laden von Massendaten aus heterogenen Quellen. Der Palo-ETL-Server wurde als Bestandteil der Palo Suite 3.1[2] genutzt.

Die OLAP-Funktionalitäten wurden mittels Palo OLAP Server[3] (ebenfalls in der Suite enthalten) realisiert. Der Palo OLAP Server ist ein multidimensionaler in-memory OLAP Server (MOLAP). Alle Daten die der Palo OLAP Server beinhaltet sind in Würfel, Dimensionen, Elemente und Element-Attribute organisiert. Als OLAP-Oberfläche wurde das – auch in der Suite verfügbare - PALO ExcelAddIn[4] genutzt, so dass die OLAP Funktionalitäten in der gewohnten Excel Umgebung genutzt werden können.

[1] http://www.jedox.com/de/produkte/Palo-Suite/palo-etl-server.html

[2] http://www.jedox.com/de/produkte/Palo-Suite.html

[3] http://www.jedox.com/de/produkte/Palo-Suite/palo-olap-server.html

[4] http://www.jedox.com/de/produkte/palo-for-excel/palo-for-excel-add-in-ms-excel.html

1.4 Annahmen

Da der hier vorgestellte ETL-Prozess möglichst allgemeingültig sein soll, mussten einige Annahmen über die Nutzung des Bugzilla Systems getätigt werden. Diese werden im Folgenden erläutert:

1. Es werden nur die Standardmäßig in Bugzilla vorhandenen Felder unterstützt.
2. Der Entwickler, der Aufwand für einen Bug einträgt, hat diesen auch verursacht.
3. Es ist möglich negativen Aufwand einzutragen. Dies ist nötig, um am Ende eines Monats evtl. Korrekturen zu erfassen.
4. Der für einen Bug geplanter Aufwand wird zu 100% dem Bug Verantwortlichen angerechnet. Wird dieser im Nachhinein geändert, so gilt der komplette Planaufwand für den nun gewählten Verantwortlichen.
5. Eine Aggregation auf Tage reicht aus, es ist nicht nötig, die genaue Stunde des Aufwandes zu erfassen.
6. Für die Aggregation der Tage reicht es, die Tage aufzunehmen, an denen auch tatsächlich Aufwand entstanden ist. Dies erspart viele Felder, die nur mit 0 gefüllt sind.

2 Installation & Durchführung

Als Voraussetzung für die Installation und die spätere Ausführung des ETL-Prozesses müssen die in Kapitel 1.3 beschriebenen Komponenten installiert sein und es müssen die zugehörigen Logindaten bekannt sein. Anschließend müssen die folgenden vier Schritte durchgeführt werden um den ETL-Prozess zu installieren und auch anschließend durchzuführen.

2.1 Projekt importieren

In der Palo ETL-Server Web Applikation muss man sich zuerst in den „ETL-Manager“ begeben. Dort gelangt man über den Punkt „Upload project“ zur Importierfunktion. Hier kann die gelieferte XML Datei angegeben werden. Diese wird über einen Klick auf „Upload“ in das System geladen. Anschließend sollte unter dem Punkt „ETL Server“ der Unterpunkt „Bugzilla“ erscheinen.

2.2 Bugzilla Connection ändern

Da die mitgelieferten Standard Logindaten für Bugzilla höchst wahrscheinlich nicht denen Ihres Systems entsprechen, müssen diese geändert werden. Über die Punkte „ETL Server“ → „Bugzilla“ → „Connections“ → „Mysql“ → „Bugzilla“ gelangt man zu den Einstellungen für die Bugzilla Connection. Hier müssen die richtigen Angaben eingegeben werden. Anschließend muss die Änderung durch einen Klick auf den Punkt „Save“ gespeichert werden.

2.3 Palo Connection ändern

Analog zur Bugzilla Connection muss auch die Palo Connection in der Regel geändert werden. Diese befindet sich unter dem Punkt „ETL Server“ → „Bugzilla“ → „Connections“ → „Palo“ → „Palo“. Auch hier muss nachdem die Änderungen eingegeben wurden das Projekt durch einen Klick auf „Save“ gespeichert werden.

2.4 Durchführung des ETL-Prozesses

Als letztes kann nun der ETL-Prozess ausgeführt werden. Hierzu muss man den Punkt „ETL Server“ → „Bugzilla“ → „Jobs“ auswählen. Anschließend ist der Job „LoadAll“ auszuwählen und durch einen Klick auf „Execute“ auszuführen. Nach kurzer Zeit sollte als Result „successful“ erscheinen und die Dimensionen und Würfel des Prozesses sind nun in Palo angelegt und gefüllt.

3 ETL-Prozess

Im Folgenden wird der eigentliche ETL-Prozess beschrieben. Auch wenn dieser mit dem Palo ETL-Server implementiert wurde, sollte es möglich sein, die hier beschriebenen Schritte mit einem anderen ETL-Tool umzusetzen. Hierbei muss darauf geachtet werden, dass sich die Typ-Bezeichnung eines Schrittes durchaus unterscheiden kann, da diese nicht Standardisiert ist.

Zunächst werden die Quell- bzw. Zielsysteme beschrieben, hierauf folgen die Beschreibung der von uns gewählten Basiseinheiten und deren Extraktion aus Bugzilla. Anschließend wird aufgezeigt, wie wir von den Basiseinheiten zu Dimensionen und den gefüllten Würfeln gelangen. Den Abschluss dieses Kapitels bildet die Beschreibung der gewählten Jobs, die im Palo ETL-Server ausgeführt werden können.

3.1 Quell- und Zielsysteme

Dieser ETL-Prozess benötigt mit Bugzilla als Quell- und Palo als Zielsystem zwei Systeme. Diese können im Palo ETL-Server unter dem Punkt Connections angesehen und geändert werden.

3.1.1 Bugzilla

Diese Connection wird für sämtliche Extraktionen der Daten aus Bugzilla benötigt. Standardmäßig sind folgende Werte eingestellt:

Type: MySQL

Parameter:

Host	localhost
Port	
Database	bugzilla_07
User	root
Password	

3.1.2 Palo

Diese Connection dient den Load-Prozessen als Zielquelle. Die folgenden Werte sind dabei standardmäßig eingestellt:

Type: Palo

Parameter:

Host	localhost
Port	7921
Database	bugzilla_02

User	admin
Password	admin

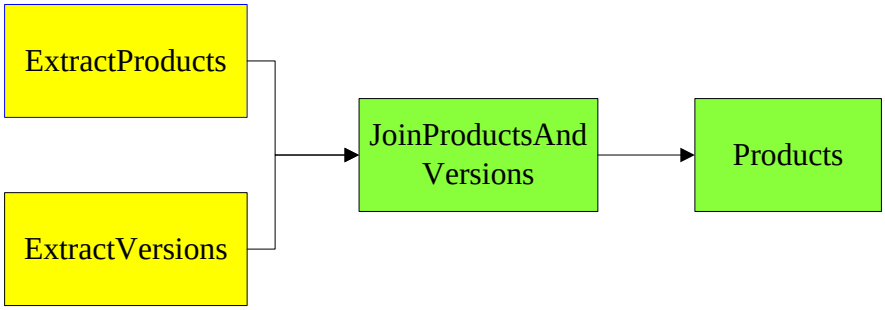
3.2 Extraktion von logischen Basiseinheiten

Nach den ersten Implementierungsversuchen für den ETL-Prozess hat sich herausgestellt, dass man für die Erstellung der Dimensionen bzw. das füllen der Würfel gewisse Schritte mehrmals durchführen muss. So müssen z.B. bei fast allen Extraktionen anschließend IDs über Joins aufgelöst werden, um z.B. die Daten der Bugs um den Entwickler zu erweitern. Somit wurde sich dafür entschieden, bevor die Transformationen in Richtung der eigentlichen Dimension bzw. des Würfels vorgenommen werden, so genannte Basiseinheiten zu erstellen.

Basiseinheiten bestehen nur aus Informationen, die aus Bugzilla stammen. Zumeist sind für die Erstellung der Basiseinheiten Joins nötig, um z.B. Bugs mit dem zuständigen Entwickler zu verknüpfen. Hierneben wird auch die eine oder andere Umformung von Werten vorgenommen.

Die fertigen Basiseinheiten bilden anschließend die Basis für das erstellen der Dimensionen bzw. die Füllung der Würfel.

3.2.1 Products



Die erste identifizierte Basiseinheit ist Products. Dieses besteht, wie in der grafischen Darstellung des Prozesses zu sehen ist, aus den Bugzilla Elementen Produkt und Version. Benötigt wird diese Kombination um später bis auf einzelne Version filtern zu können.

3.2.1.1 ExtractProducts

Hier werden alle in Bugzilla definierten Produkte extrahiert. Dabei sind nur die ID und der Name für die Weiterverarbeitung von Interesse.

Type: Extracts (Relational)

Input: 3.1.1 Bugzilla

SQL:

```
SELECT
    id,
    name,
FROM
    products
```

Output:

Feld Name	Feld Wert
product_id	Id
product_name	name

3.2.1.2 ExtractVersions

Da in der hierarchischen Darstellung der Produkte auch die verschiedenen Produktversionen berücksichtigt werden sollen, müssen auch diese aus Bugzilla geladen werden. Hier sind die ID des Produktes, sowie der Name der Version von Interesse.

Type: Extracts (Relational)

Input: 3.1.1 Bugzilla

SQL:

```
SELECT
    product_id,
    value,
FROM
    versions
```

Output:

Feld Name	Feld Wert
product_id	product_id
version	value

3.2.1.3 JoinProductsAndVersions

Die zuvor geladenen Produkte und Versionen werden nun in diesem Schritt vereinigt.

Type: Transforms (TableJoin)

Input: 3.2.1.1 ExtractProducts | 3.2.1.2 ExtractVersions

Join-Type: inner

Join Keys:

- product_id | product_id

Output:

Feld Name	Feld Wert	
	3.2.1.1	**3.2.1.2**
product_id	product_id	

product_name	product_name	
version		version

3.2.1.4 Products

Da gleiche Versionen in verschiedenen Produkten vorkommen können (z.B. 1.0.0), könnte eine so benannte Version von Palo nicht mehr eindeutig referenziert werden. Aus diesem Grunde wird aus product_name und version der neue Wert product_version gebildet.

Type: Transforms (FieldTransform)

Input: 3.2.1.3 JoinProductsAndVersions

Functions:

Name:	product_version
Type:	Concatenation
Inputs:	• product_name • version
Function:	#{product_name}: #{version}

Output:

Feld Name	Feld Wert
product_id	product_id
product_name	product_name
product_version	product_version
version	version

3.2.2 Developers

Als zweite Basiseinheit dienen die Developers. Diese repräsentieren die in Bugzilla vorhandenen Entwickler, welche Bugs bearbeiten und den Plan- und Istaufwand der Bugs pflegen.

3.2.2.1 Developers

Dieser Schritt extrahiert alle in Bugzilla vorhandenen Entwickler. Hierbei sind lediglich die ID und der Name des Entwicklers von Interesse, so dass auch nur diese extrahiert werden.

Type: Extracts (Relational)

Input: 3.1.1 Bugzilla

SQL:

```
SELECT
    userid,
    realname,
FROM
    profiles
```

Output:

Feld Name	Feld Wert
developer_id	userid
developer_name	realname

3.2.3 Priorities

Neben Products und Developers werden auch die Priorities benötigt. Diese Prioritäten spiegeln die Wichtigkeit eines Bugs wieder und werden für die Anzeige der noch offenen Bugs benötigt.

3.2.3.1 Priorities

Extrahiert die vorhandenen Prioritäten.

Type: Extracts (Relational)

Input: 3.1.1 Bugzilla

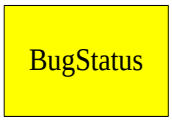
SQL:

```
SELECT
    value
FROM
    priority
```

Output:

Feld Name	Feld Wert
priority	value

3.2.4 BugStatus



Die vierte Basiseinheit ist die des BugStatus. Der BugStatus wird benötigt, da hier die Information hinterlegt ist, ob der aktuelle Bug noch offen, oder bereits geschlossen ist.

3.2.4.1 BugStatus

In diesem Schritt wird der Bug Status aus der Bugzilla Datenbank extrahiert.

Type: Extracts (Relational)

Input: 3.1.1 Bugzilla

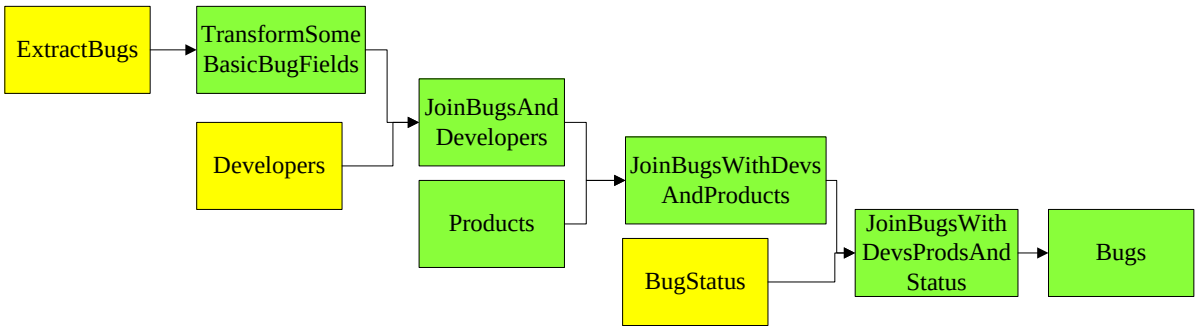
SQL:

```
SELECT
    is_open,
    value
FROM
    bug_status
```

Output:

Feld Name	Feld Wert
bug_is_open	is_open
bug_status	value

3.2.5 Bugs



Die Basiseinheit Bugs ist der zentrale Punkt des ETL-Prozesses. Die Informationen werden praktisch für alle Würfel benötigt. Aus diesem Grunde muss der reine Bug, den man aus Bugzilla bekommt um die Informationen über den zuständigen Entwickler, des zugehörigen Produktes, und den Status des Bugs ergänzt werden.

3.2.5.1 ExtractBugs

Extrahiert die Bug-Daten aus Bugzilla. Hierbei sind zahlreiche Felder von Bedeutung, da Bugs im Zentrum des ETL-Prozesses und der nachfolgenden Auswertungen stehen.

Type: Extracts (Relational)

Input: 3.1.1 Bugzilla

SQL:

```
SELECT
    assigned_to,
    bug_id,
    bug_status,
    estimated_time,
    priority,
    product_id,
    remaining_time,
```

```
    short_desc,
    version
FROM
    bugs
```

Output:

Feld Name	Feld Wert
developer_id	assigned
bug_id	bug_id
bug_status	bug_status
bug_estimated_time	estimated_time
priority	priority
product_id	product_id
bug_remaining_time	remaining_time
bug_description	short_desc
version	version

3.2.5.2 TransformSomeBasicBugFields

Nach dem extrahieren der Bugs, werden hier insgesamt zwei verschiedene Transformationen vorgenommen.

Zu einem werden aufgrund eines festgestellten Bugs des Palo-ETL-Servers in diesem Schritt die zuvor extrahierten Zeiten (bug_estimated_time und bug_remaining_time) in eine einheitliche dezimale Repräsentation verwandelt.

Zum anderen wird aus der Bug ID und der Kurzbeschreibung ein einmaliger Name für den Bug generiert, der für einen Menschen so besser zuzuordnen ist, als die reine Bug ID.

Type: Transforms (FieldTransform)

Input: 3.2.5.1 BugStatus

Functions:

Name:	bug_remaing_time_decimal
Type:	NumberFormat
Inputs:	• bug_remaining_time
Function:	**Pattern:** 0.00

Name:	bug_estimated_time_decimal
Type:	NumberFormat
Inputs:	• bug_estimated_time
Function:	**Pattern:** 0.00

Name:	bug_id_5_digits
Type:	NumberFormat
Inputs:	• bug_id
Function:	**Pattern:** 00000

Name:	bug_name_
Type:	Concatenation
Inputs:	• bug_id_5_digits • bug_name
Function:	**Template:** #{bug_id_5_digits}: #{bug_description}

Output:

Feld Name	Feld Wert
bug_estimated_time_decimal	bug_estimated_time
bug_id	bug_id
bug_name	bug_name
bug_remaining_time_decimal	bug_remaining_time
bug_status	bug_status
developer_id	developer_id
priority	priority
product_id	product_id
version	version

3.2.5.3 JoinBugsAndDevelopers

Nun werden die Bugs um den Namen des zuständigen Entwicklers ergänzt. Darum wird ein Join mit den Entwicklerdaten vorgenommen.

Type: Transforms (TableJoin)

Input: 3.2.5.2 TransformSomeBasicBugFields | 3.2.2.1 Developers

Join-Type: inner

Join-Keys:
- developer_id | developer_id

Output:

Feld Name	Feld Wert	
	3.2.5.2	**3.2.2.1**
bug_estimated_time	bug_estimated_time	
bug_id	bug_id	
bug_name	bug_name	

bug_remaining_time	bug_remaining_time	
bug_status	bug_status	
developer_id	developer_id	
developer_name		developer_name
Priority	priority	
product_id	product_id	
version	version	

3.2.5.4 JoinBugsWithDevsAndProducts

Nachdem der Bug nun um Entwickler-Daten angereichert wurde, folgen in diesem Schritt die Produkt-Daten.

Type: Transforms (TableJoin)

Input: 3.2.5.3 JoinBugsAndDevelopers | 3.2.1.4 Products

Join-Type: inner

Join-Keys:
- product_id | product_id
- version | version

Output:

Feld Name	**Feld Wert**	
	3.2.5.3	**3.2.1.4**
bug_estimated_time	bug_estimated_time	
bug_id	bug_id	
bug_name	bug_name	
bug_remaining_time	bug_remaining_time	
bug_status	bug_status	
developer_id	developer_id	
developer_name	developer_name	
Priority	priority	
product_id	product_id	
product_name		product_name
product_version		product_version
version	version	

3.2.5.5 JoinBugsWithDevsProdsAndStatus

Zusätzlich braucht man noch nun noch die Status-Daten für die vorhandenen Bugs.

Type: Transforms (TableJoin)

Input: 3.2.5.4 JoinBugsWithDevsAndProducts | 3.2.4.1 BugStatus

Join-Type: inner

Join-Keys:
- bug_status | bug_status

Output:

Feld Name	Feld Wert	
	3.2.5.4	**3.2.4.1**
bug_estimated_time	bug_estimated_time	
bug_id	bug_id	
bug_is_open		bug_is_open
bug_name	bug_name	
bug_remaining_time	bug_remaining_time	
bug_status		bug_status
developer_id	developer_id	
developer_name	developer_name	
Priority	priority	
product_id	product_id	
product_name	product_name	
product_version	product_version	
version	version	

3.2.5.6 Bugs

Nachdem die Bugs nun um alle wichtigen Daten ergänzt wurden, werden nicht mehr benötigte Felder herausgenommen, um die weitere Arbeit zu erleichtern.

Type: Transforms (FieldTransform)

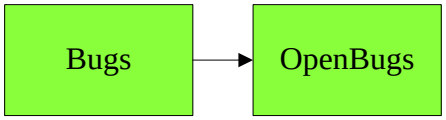
Input: 3.2.5.5 JoinBugsWithDevsProdsAndStatus

Output:

Feld Name	Feld Wert
developer_name	bug_developer_name
bug_estimated_time	bug_estimated_time
bug_id	bug_id
bug_is_open	bug_is_open
bug_name	bug_name
bug_remaining_time	bug_remaining_time
product_name	product_name

product_version	product_version
priority	priority
version	version

3.2.6 OpenBugs



Als nächste Basiseinheit dienen die OpenBugs. Diese werden einfach aus der Menge aller Bugs herausgefiltert.

3.2.6.1 OpenBugs

In diesem Schritt werden alle offenen Bugs gesammelt.

Type: Transforms (TableView)

Input: 3.2.5.6 Bugs

Sort by:
- bug_name

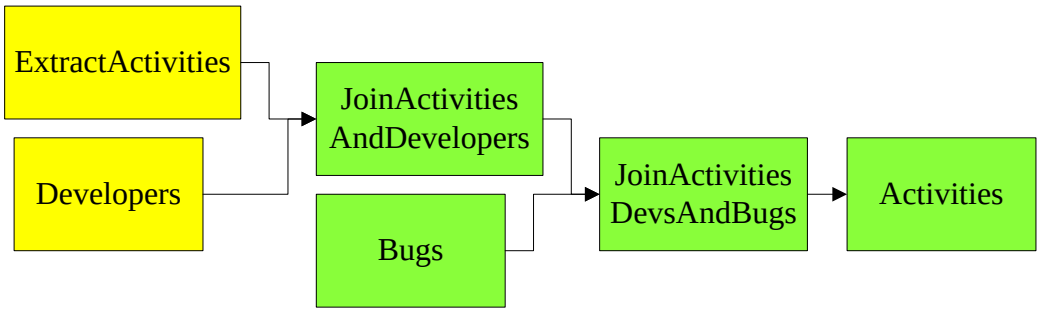
Filter Type: and

Filter:

Feld Name	Filter Type	Operator	Value
bug_is_open	accept	equal	1

Output: Es sind nur noch Bugs vorhanden, deren bug_is_open Wert = 1 ist.

3.2.7 Activities



Als nächstes wird die Basiseinheit Activities erstellt. Diese zeichnet alle Änderungen durch Entwickler an Bugs auf. Besonders von Interesse sind hier die Änderungen an den Feldern zum Plan- und Istaufwand.

3.2.7.1 ExtractActivities

Hier werden alle in Bugzilla durchgeführten Aktivitäten extrahiert. Diese Aktivitäten werden erstellt, sobald sich etwas an einem Bug ändert. Von Interesse sind hier die ID des zugehörigen Bugs, wer die Aktivität durchgeführt hat, welches Feld verändert wurde, der neue Wert des Feldes, sowie der Zeitpunkt der Änderung.

Type: Extracts (Relational)

Input: 3.1.1 Bugzilla

SQL:

```
SELECT
    added,
    bug_id,
    bug_when,
    fieldid,
    who
FROM
    bugs_activity
```

Output:

Feld Name	Feld Wert
activity_value	added
bug_id	bug_id
activity_date	bug_when
activity_field	fieldid
developer_id	who

3.2.7.2 JoinActivitiesAndDevelopers

In diesem Schritt werden die zuvor extrahieren Aktivitäten mit den Informationen des zugehörigen Entwicklers angereichert.

Type: Transforms (TableJoin)

Input: 3.2.7.1 OpenBugs | 3.2.2.1 Developers

Join-Type: inner

Join-Keys:

- developer_id | developer_id

Output:

Feld Name	Feld Wert	
	3.2.7.1	**3.2.2.1**
activity_date	activity_date	
activity_field	activity_field	

activity_value	activity_value	
bug_id	bug_id	
developer_id	developer_id	
developer_name		developer_name

3.2.7.3 JoinActivitiesDevsAndBugs

Nachdem die Aktivitäten durch die Entwickler Informationen ergänzt wurden, folgen nun die Bug Informationen.

Type: Transforms (TableJoin)

Input: 3.2.7.2 JoinActivitiesAndDevelopers | 3.2.5.6 Bugs

Join-Type: inner

Join-Keys:
- bug_id | bug_id

Output:

Feld Name	**Feld Wert**	
	3.2.7.2	**3.2.5.6**
activity_date	activity_date	
activity_field	activity_field	
activity_value	activity_value	
bug_developer_name		bug_developer_name
bug_estimated_time		bug_estimated_time
bug_id	bug_id	
bug_is_open		bug_is_open
bug_name		bug_name
bug_remaining_time		bug_remaining_time
developer_id	developer_id	
developer_name	developer_name	
priority		priority
product_name		product_name
product_version		product_version
version		version

3.2.7.4 Activities

Hier finden einige Transformationen statt. Zum einen wird aus dem kompletten Timestamp *activity_date* (yyyy-MM-dd HH:mm:ss.S) der Tag extrahiert, da eine detailliertere Aggregation auf z.B. Stunden nicht benötigt wird.

Zudem werden einige nicht mehr benötigte Felder entfernt.

Type: Transforms (FieldTransform)

Input: 3.2.7.3

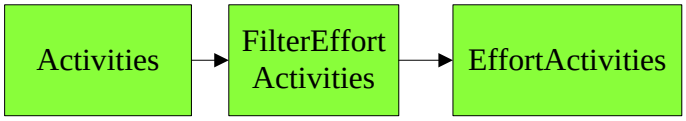
Functions:

Name:	activity_day
Type:	DateFormat
Inputs:	• activity_date
Function:	**Targetformat:** yyyy-MM-dd

Output:

Feld Name	Feld Wert
activity_day	activity_day
developer_name	activity_developer_name
activity_field	activity_field
activity_value	activity_value
bug_developer_name	bug_developer_name
bug_estimated_time	bug_estimated_time
bug_id	bug_id
bug_is_open	bug_is_open
bug_name	bug_name
bug_remaining_time	bug_remaining_time
priority	priority
product_name	product_name
product_version	product_version
version	version

3.2.8 EffortActivities



Die achte Basiseinheit bilden die EffortActivities. Dies sind Activites, bei denen eine Änderung am Istaufwand durchgeführt wurde.

3.2.8.1 FilterEffortActivities

Durch das herausfiltern aller Aktivitäten, die eine Änderung am Aufwandsfeld vornehmen, erhält man die EffortActivities.

Type: Transforms (TableView)

Input: 3.2.7.4 Activities

Sort by:
- bug_name

Filter Type: and

Filter:

Feld Name	Filter Type	Operator	Value
activity_field	accept	equal	47

Output: Nur noch die Aktivitäten deren activity_field Wert = 47 ist sind vorhanden.

3.2.8.2 EffortActivities

Nachdem Filtern, kann nun die *activity_value* in einen dezimal Wert geändert werden.

Type: Transforms (FieldTransform)

Input: 3.2.8.1 FilterEffortActivities

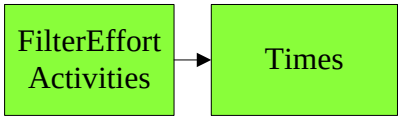
Functions:

Name:	activity_effort
Type:	NumberFormat
Inputs:	• activity_value
Function:	**Pattern:** 0.00

Output:

Feld Name	Feld Wert
activity_day	activity_day
developer_name	activity_developer_name
activity_effort	activity_effort
bug_developer_name	bug_developer_name
bug_estimated_time	bug_estimated_time
bug_id	bug_id
bug_is_open	bug_is_open
bug_name	bug_name
bug_remaining_time	bug_remaining_time
priority	priority
product_name	product_name
product_version	product_version
version	version

3.2.9 Times



Die letzte Basiseinheit für diesen ETL-Prozess bildet die Times Basiseinheit. Diese stellt die Tage dar, an denen tatsächlich Istaufwand für einen Bug vollzogen wurde.

3.2.9.1 Times

Da nur die Tage benötigt werden, die auch eine Änderung am Aufwand enthalten, können einfach die Tage der EffortActivities extrahiert werden. Zusätzlich wird hierbei der erste Schritt in Richtung hierarchische Zeitdarstellung getan. Hierzu wird das bisherige Datum (yyyy-MM-dd) in die drei Teile **year** (yyyy), **month** (yyyy-MM) und **day** (yyyy-MM-dd) unterteilt.

Type: Transforms (FieldTransform)

Input: 3.2.8.2 EffortActivities

Functions:

Name:	year
Type:	DateFormat
Inputs:	• activity_day
Function:	**Targetformat:** yyyy

Name:	month
Type:	DateFormat
Inputs:	• activity_day
Function:	**Targetformat:** yyyy-MM

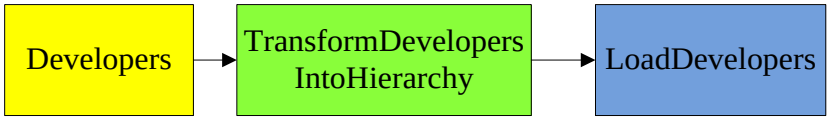
Output:

Feld Name	**Feld Wert**
activity_day	day
month	month
year	year

3.3 Erstellung der Dimensionen

Im Folgenden werden für jede der vier Dimensionen die Schritte erläutert, die aus den Basiseinheiten fertige hierarchische Dimensionen erstellen und diese anschließend in Palo importieren.

3.3.1 Entwickler



Die Dimension des Entwicklers basiert auf der Basiseinheit Developers. Diese wird lediglich in eine flache Hierarchie gebracht und anschließend in Palo geladen.

3.3.1.1 TransformDevelopersIntoHierarchy

Die extrahierten Entwickler werden nun in eine zweistufige Hierarchische Darstellung gebracht. Auf der ersten Ebene befindet sich das Element **Alle** darunter die einzelnen *developer_name*.

Type: Transforms (TreeFH)

Input: 3.2.2.1 Developers

Output:

level1	**level2**
Alle	*developer_name*

3.3.1.2 LoadDevelopers

Hier werden die in eine Hierarchie gebrachten Entwickler endgültig als Dimension in das Palo System geladen.

Type: Loads (Dimension)

Input: 3.3.1.1 TransformDevelopersIntoHierarchy

Output: Dimension **Entwickler** in 3.1.2 Palo

3.3.2 Produkt

Als zweite Dimension wird das Produkt in Palo geladen. Hierzu wird die Basiseinheit Products in eine Hierarchie gebracht und danach geladen.

3.3.2.1 TransformProductsIntoHierarchy

Die fertigen Produkte werden nun in eine dreistufige Hierarchie gebracht. Die erste Ebene wird durch das Element **Alle** eingenommen. Hierunter kommt der *product_name* und in die unterste Ebene die *product_version*.

Type: Transforms (TreeFH)

Input: 3.2.1 Products

Output:

level1	level2	level3
Alle	*product_name*	*product_version*

3.3.2.2 LoadProducts

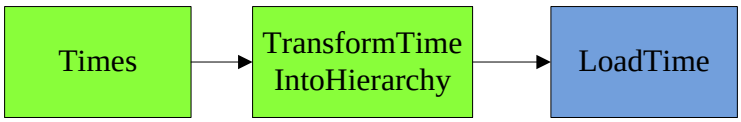
Die in eine Hierarchie gebrachten Produkte werden nun in das Palo System geladen.

Type: Loads (Dimension)

Input: 3.3.2.1 TransformProductsIntoHierarchy

Output: Dimension **Produkt** in 3.1.2 Palo

3.3.3 Zeit



Als nächste Dimension wird die Zeit weiter verarbeitet. Diese besteht aus einer Hierarchie der Basiseinheit Times.

3.3.3.1 TransformTimeIntoHierarchy

Nachdem nun alle Vorbereitungen für die Zeithierarchie fertig sind, wird nun die endgültige Hierarchie hergestellt. Diese besteht aus den vier Stufen: Element **Alle**, *year*, *month* und *day*.

Type: Transforms (TreeFH)

Input: 3.2.9.1 Times

Output:

level1	level2	level3	level4
Alle	*year*	*month*	*day*

3.3.3.2 LoadTime

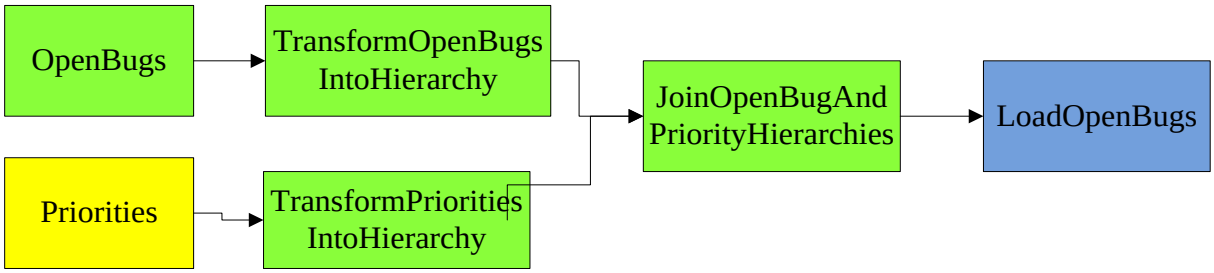
Nun wird die Dimension Zeit endgültig in das Palo System geladen.

Type: Loads (Dimension)

Input: 3.3.3.1 TransformTimeIntoHierarchy

Output: Dimension **Zeit** in 3.1.2 Palo

3.3.4 Offene Bugs



Als letzte Dimension werden die offenen Bugs behandelt. Diese setzen sich aus den Basiseinheiten OpenBugs und Priorities zusammen. Nachdem diese beiden Basiseinheiten in eine gemeinsame Hierarchie gebracht wurden, werden sie dann in Palo geladen.

3.3.4.1 TransformOpenBugsIntoHierarchy

Aufgrund der oben beschriebenen Hierarchie, müssen die OpenBugs nun in die passende Hierarchie gebracht werden.

Type: Transforms (TreeFH)

Input: 3.2.6.1 OpenBugs

Output:

level1	level2	level3
Alle	*priority*	*bug_name*

3.3.4.2 TransformPrioritiesIntoHierarchy

Neben den OpenBugs, müssen auch die im System vorhandenen Priorities in eine Hierarchie gebracht werden.

Type: Transforms (TreeFH)

Input: 3.2.3.1 Priorities

Output:

level1	level2
Alle	*priority*

3.3.4.3 JoinOpenBugAndPriorityHierarchies

Nun müssen die beiden erstellten Hierarchien gejoined werden, um eine große Hierarchie abzubilden.

Type: Transforms (TreeJoin)

Input: 3.3.4.1 TransformOpenBugsIntoHierarchy | 3.3.4.2 TransformPrioritiesIntoHierarchy

Joins:

Input	Root element
TransformPrioritiesIntoHierarchy	**Alle**
TransformOpenBugsIntoHierarchy	**Alle**

Output:

level1	level2	level3
Alle	*priority*	*bug_name*

3.3.4.4 LoadOpenBugs

Nachdem die Hierarchie nun fertig erstellt ist, kann auch diese Dimension in Palo importiert werden.

Type: Loads (Dimension)

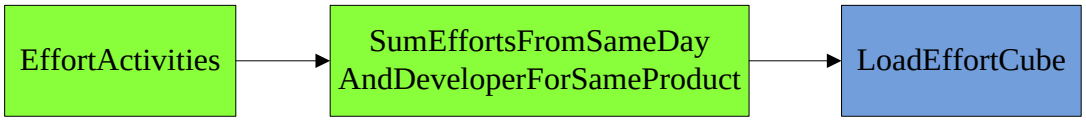
Input: 3.3.4.3 JoinOpenBugAndPriorityHierarchies

Output: Dimension **Offene Bugs** in 3.1.2 Palo

3.4 Erstellung der Würfel

Im Folgenden werden alle Prozesse beschrieben, um aus den Basiseinheiten und den zuvor geladenen Dimensionen die Würfel zu erstellen bzw. mit Daten zu füllen.

3.4.1 Aufwand-Ist



Als erster Würfel wird der Würfel, welcher den Istaufwand repräsentiert erstellt. Basis ist hier EffortActivities. Die hier vorliegenden Aufwände werden noch aggregiert, da hier nur der Tagesaufwand für ein Produkt benötigt wird. Anschließend wird der Würfel in Palo geladen.

3.4.1.1 SumEffortsFromSameDayAndDeveloperForSameProduct

Bevor der Aufwand importiert wird, wird hier eine Konsolidierung vorgenommen. Dabei werden Aufwände, die ein Entwickler am selben Tag an Bugs von identischen Produkten und Versionen vorgenommen hat addiert.

Type: Transforms (TableTransform)

Input: 3.2.8.2 EffortActivities

Mode: Only Aggregation/Projection

Measures:

Feld Name	Feld Wert	Aggregation	Data type
Aufwand	*activity_effort*	sum	numeric

Output:

Feld Name	Feld Wert
Produkt	product_version
Zeit	activity_day
Entwickler	activity_developer_name

3.4.1.2 LoadEffortCube

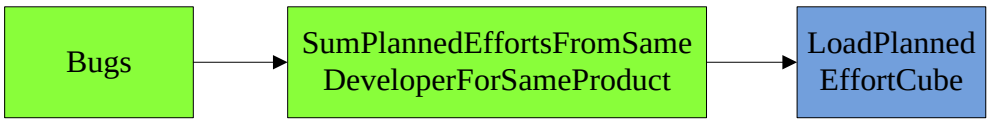
Der noch einmal konsolidierte Aufwand wird nun in den Palo Würfel geladen.

Type: Loads (Cube)

Input: 3.4.1.1 SumEffortsFromSameDayAndDeveloperForSameProduct

Output: Würfel **Aufwand-Ist** in 3.1.2 Palo

3.4.2 Aufwand-Plan



Nachdem der Istaufwand geladen wurde, wird nun der Planaufwand geladen. Auch für diese findet eine Aggregation der Daten statt. Da der Planaufwand nicht für einzelne Tage erfasst werden kann, wird hier nur nach dem Produkt aggregiert.

3.4.2.1 SumPlannedEffortsFromSameDeveloperForSameProduct

Bevor der geplante Aufwand importiert wird, wird hier eine Konsolidierung vorgenommen. Dabei werden alle geplanten Aufwände, die ein Entwickler für Bugs von identischen Produkten und Versionen vorgenommen hat addiert.

Type: Transforms (TableTransform)

Input: 3.2.5.6 Bugs

Mode: Only Aggregation/Projection

Measures:

Feld Name	Feld Wert	Aggregation	Data type
Planaufwand	*bug_estimated_time*	sum	numeric

Output:

Feld Name	Feld Wert
Produkt	product_version
Entwickler	bug_developer_name

3.4.2.2 LoadPlannedEffortCube

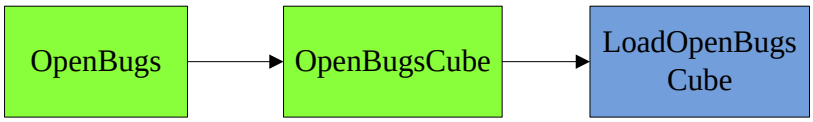
Der konsolidierte Planaufwand wird nun in den Palo Würfel geladen.

Type: Loads (Cube)

Input: 3.4.2.1 SumPlannedEffortsFromSameDeveloperForSameProduct

Output: Würfel **Aufwand-Plan** in 3.1.2 Palo

3.4.3 Bugs-Offen



Als letzter Würfel wird der Würfel für die offenen Bugs geladen. Dieser basiert komplett auf der Basiseinheit OpenBugs. Die Feldnamen der Basiseinheit müssen lediglich umbenannt werden, um denen der Dimensionen zu entsprechen. Nach dieser Umbenennung wird der Würfel in Palo geladen.

3.4.3.1 OpenBugsCube

Dieser Schritt dient lediglich dazu, die Namen der Felder in die Namen der Dimensionen umzuwandeln.

Type: Transforms (FieldTransform)

Input: 3.2.6.1 OpenBugs

Output:

Feld Name	Feld Wert
Offene Bugs	bug_name
Entwickler	bug_developer_name
Restaufwand	bug_remaining_time

3.4.3.2 LoadOpenBugsCube

Alle noch offenen Bugs werden nun in Palo importiert.

Type: Loads (Cube)

Input: 3.4.3.1 OpenBugsCube

Output: Würfel **Bugs-Offen** in 3.1.2 Palo

3.5 Vorhandene Jobs

Die ausführbaren Jobs die der ETL-Prozess enthält umfassen an dieser Stelle nur den Job LoadAll.

LoadAll

Dieser Prozess lädt alle Dimensionen und Würfel in Palo. Dabei werden die Elemente in der folgenden Reihenfolge geladen:

1. 3.3.1 Entwickler
2. 3.3.2 Produkt
3. 3.3.3 Zeit
4. 3.3.4 Offene Bugs
5. 3.4.1 Aufwand-Ist
6. 3.4.2 Aufwand-Plan
7. 3.4.3 Bugs-Offen