Mit buw zur Nr. 1

[buw]

OBJEKTORIENTIERTE ENTWICKLUNG IN DER DATENBANK

Anja Hildebrandt
buw Unternehmensgruppe

Berlin, 26. April 2016

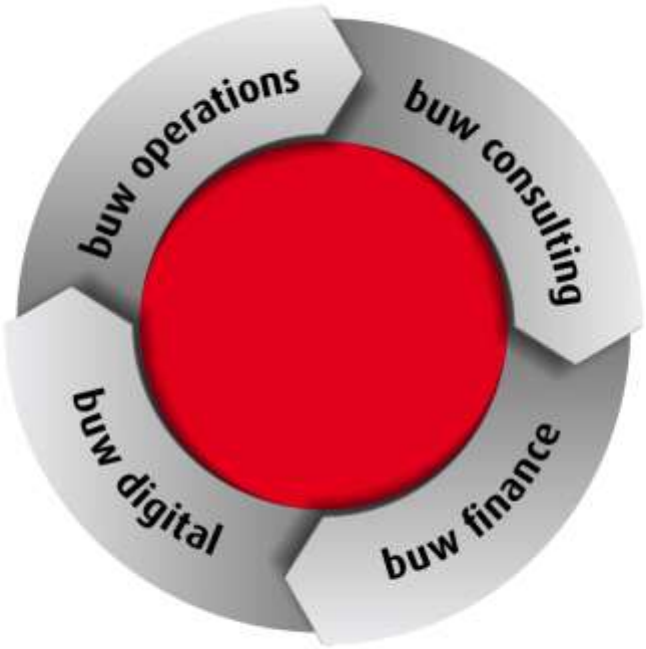
ÜBER MICH

- 1998 – 2005 Informatik-Studium Otto-von-Guericke Universität Magdeburg
- Seit 1999 datenbanknahe Entwicklung mit Pl/Sql
- Seit 2004 Konzeption und Realisierung diverser Webanwendungen (htmlDB, Apex)
- 2010-2014 Freelance Apex Consultant
- Seit 01/2015 Entwickler bei buw Unternehmensgruppe in Osnabrück
- Blog: http://apex2rule-the-world.blogspot.com
- ….

ÜBER BUW

buw operations
Führender Qualitätsdienstleister
im Kundenmanagement-Outsourcing

buw digital
Experten für Digitales Kunden-
management: Social Media,
Social Software & Social Commerce:
Strategie, Befähigung, Umsetzung



buw consulting
Deutschlands größte
Kundenmanagement-Beratung:
Vertrieb, Service, CC

buw finance
Partner für das Outsourcing
von Finanzprozessen:
BPO Finance & Payment,
Forderungsmanagement, Inkasso

ÜBER BUW

- **Gründung:**
  3. März 1993
- **Gründer und geschäftsführende Gesellschafter:**
  Jens Bormann, Karsten Wulf
- **Weitere Geschäftsführer:**
  Daniel Benzenhöfer, Dr. Claudio Felten
- **Mitarbeiter:**
  über 6000 fest angestellte Mitarbeiter in Voll- und Teilzeit
  (Stand Februar 2015)
- **Umsatz:**
  2014 (vorläufig): 137,5 Mio. Euro
  2013: 122,5 Mio. Euro
- **Standorte:**
  Osnabrück, Münster, Halle, Schwerin, Leipzig, Wismar,
  Gera, Frankfurt a.M., Duisburg, Wuppertal,
  Pécs (Ungarn), Timisoara (Rumänien), Cluj-Napoca (Rumänien)

Agenda

AGENDA

- Abgrenzung (Um was geht’s? Um was geht’s nicht?)
- Grundlagen
- Pilotprojekt
- Demo
- Erfahrungen
- Weiterführende Informationen

Abgrenzung

Worum geht es? Worum nicht?

ABGRENZUNG

Worum geht's?

- Erster Einblick in den Umgang mit Oracle Object Types (ADTs)
- Vorstellung einer Lösung für ein universelles Importtool
- Vor- und Nachteile

Worum nicht?

- Keine vollständige Einführung
- Keine detaillierte Erklärung des Klassendiagramms, Design Patterns usw.
- Projekt ist noch in Arbeit, Bugs und unschöner bzw. unfertiger Code zu erwarten

Grundlagen

EINFACHE IMPLEMENTIERUNG

```
create or replace type rimportsatz IS OBJECT
(
  -- Attributes
  ds_id   number,
  ds_text varchar2(100),

  -- Member functions and procedures
  STATIC FUNCTION get_typename return varchar2,

  MEMBER FUNCTION row_exists(i_id NUMBER) RETURN BOOLEAN,
  MEMBER PROCEDURE row_insert,
  MEMBER PROCEDURE row_update,
  MEMBER PROCEDURE row_save,
  MEMBER PROCEDURE row_select(i_id in number),
  MEMBER PROCEDURE row_default,

  CONSTRUCTOR FUNCTION rimportsatz RETURN SELF AS RESULT,
  CONSTRUCTOR FUNCTION rimportsatz(i id in number) RETURN SELF AS RESULT

)
```

```
MEMBER PROCEDURE row_save IS
begin
  IF row_exists(i_id => self.ds_id) THEN
    row_update;
  ELSE
    row_insert;
  END IF;

end row_save;
```

- Instanz referenzieren mit SELF

PL/SQL-SYNTAX (OBJEKTTYPEN)

Attribute
Datenelemente von Objekten können alle eingebauten Oracle Datentypen und alle Benutzerdefinierten Objekttypen sein, auf denen der Besitzer des Objektes Ausführungsrechte besitzt. Nicht erlaubt dagegen sind Benutzerdefinierte Datentypen wie sie in PL/SQL Packages definiert werden können, also z.B. TYPE KundenNummer is NUMBER(10,0).

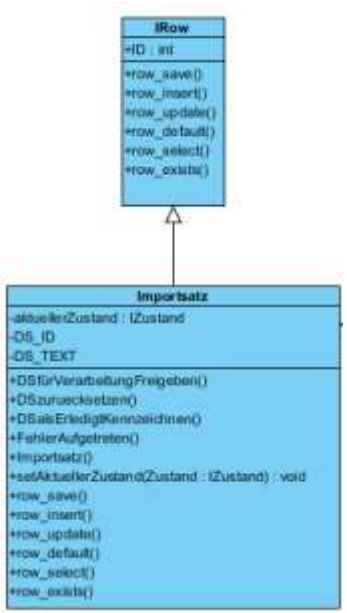
CONSTRUCTOR Funktionen
Konstruktoren sind Funktionen, die ein Objekt initialisieren. Sie werden im Augenblick der Objekterstellung durchlaufen. Eine Klasse kann beliebig viele Konstruktoren haben (Überladung).

MEMBER Methoden
Member Methoden können die Daten einer Instanz verändern oder mit ihnen eine Berechnung durchführen. Dabei bezeichnet das Schlüsselwort SELF die Instanz, zu der die Methode gehört.

STATIC Methoden
Statische Methoden gehören zur Klasse, nicht zu einer einzelnen Instanz. Sie stellen Funktionen zur Verfügung, die sich nicht eindeutig einem einzelnen Objekt zuordnen lassen, oder für deren Ausführung keine Objektinstanz benötigt wird.
Der Aufruf statischer Funktionen erfolgt über den vorangestellten Klassennamen, nicht über den Namen einer Objektvariablen, im folgenden Beispiel also rimportsatz.get_typename().

UNDER / [NOT] FINAL / [NOT] INSTANTIABLE / OVERRIDING

```
CREATE OR REPLACE TYPE IRow force
is object
(
    -- attributes
    ID number,

    NOT FINAL MEMBER FUNCTION row_exists (i_id NUMBER) RETURN BOOLEAN,
    NOT FINAL MEMBER PROCEDURE row_insert,
    NOT FINAL MEMBER PROCEDURE row_update,
    NOT FINAL MEMBER PROCEDURE row_save,
    NOT FINAL MEMBER PROCEDURE row_select(i_id NUMBER),
    NOT FINAL MEMBER PROCEDURE row_default,
    NOT FINAL static function get_attributes return tool_map_list
) NOT FINAL NOT INSTANTIABLE
```




```
create or replace type rimportsatz UNDER IROW
(
-- Attributes
  ds_id   number,
  ds_text varchar2(100),

-- Member functions and procedures
  STATIC FUNCTION get_typename return varchar2,

  OVERRIDING MEMBER FUNCTION row_exists(i_id NUMBER) RETURN BOOLEAN,
  OVERRIDING MEMBER PROCEDURE row_insert,
  OVERRIDING MEMBER PROCEDURE row_update,
  OVERRIDING MEMBER PROCEDURE row_save,
  OVERRIDING MEMBER PROCEDURE row_select(i_id in number),
  OVERRIDING MEMBER PROCEDURE row_default,

  CONSTRUCTOR FUNCTION rimportsatz RETURN SELF AS RESULT,
  CONSTRUCTOR FUNCTION rimportsatz(i_id in number) RETURN SELF AS RESULT

) FINAL
```

VERERBUNG - SYNTAX

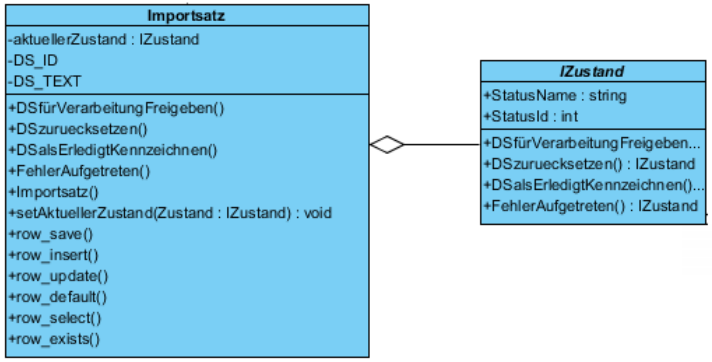
UNDER
Wird eine neue Klasse von einer bereits bestehenden NOT FINAL Klasse abgeleitet, wird das Schlüsselwort UNDER benutzt, um die Vererbung darzustellen.

FINAL / NOT FINAL
Das Schlüsselwort FINAL gibt an, dass eine Methode oder auch eine ganze Klasse endgültig ist. Damit ist es nicht möglich, diese Funktion in einer abgeleiteten Klasse zu übersteuern. Von endgültigen Klassen kann man keine weiteren Klassen ableiten.

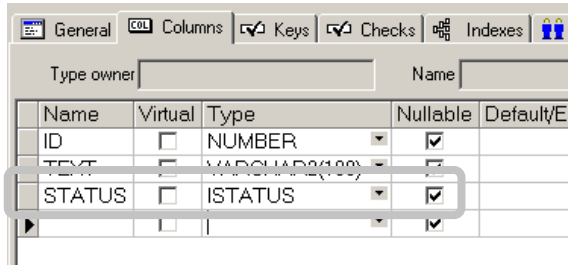
OVERRIDING
Soll das Verhalten einer NOT FINAL Methode, die aus einer Vaterklasse geerbt wurde, in einer abgeleiteten Klasse übersteuert werden, dann muss diese mit dem Schlüsselwort OVERRIDING deklariert werden.

INSTANTIABLE / NOT INSTANTIABLE
(Virtuelle Methoden und virtuelle Klassen)
Eine Funktion heißt rein virtuell, wenn sie nur deklariert, aber nicht implementiert wurde. Eine solche Funktion wird als NOT INSTANTIABLE deklariert. Eine Klasse, die solche Funktionen oder Methoden enthält, muss selbst ebenfalls als NOT INSTANTIABLE, also nicht instanzierbar, deklariert werden. Sie kann also nur als Vaterklasse für andere, spezialisierte Klassen dienen.
Statische Methoden, also Methoden, die nicht zu einer Instanz gehören, müssen nicht als NOT INSTANTIABLE deklariert werden.

KOMPOSITION




```
create or replace type rimportsatz UNDER IROW
(
-- Attributes
  ds_id            number,
  ds_text          varchar2(100),
  aktuellerStatus IStatus,

-- Member functions and procedures
  STATIC FUNCTION get_typename return varchar2,

  OVERRIDING MEMBER FUNCTION row_exists(i_id NUMBER) RETURN BOOLEAN,
  OVERRIDING MEMBER PROCEDURE row_insert,
  OVERRIDING MEMBER PROCEDURE row_update,
  OVERRIDING MEMBER PROCEDURE row_save,
  OVERRIDING MEMBER PROCEDURE row_select(i_id in number),
  OVERRIDING MEMBER PROCEDURE row_default,

  MEMBER PROCEDURE DSfuerVerarbeitungFreigeben,
  MEMBER PROCEDURE DSzuruecksetzen,
  MEMBER PROCEDURE DSalsErledigtKennzeichnen,
  MEMBER PROCEDURE FehlerAufgetreten,
  MEMBER PROCEDURE setAktuellerStatus(iAktuellerStatus in IStatus),

  CONSTRUCTOR FUNCTION rimportsatz RETURN SELF AS RESULT,
  CONSTRUCTOR FUNCTION rimportsatz(i_id in number) RETURN SELF AS RESULT

)
FINAL
```



UMWANDLUNG VON SUPERTYPE IN SUBTYPE - TREAT

Test script | DBMS Output | Statistics | Profiler | Trace

```
1  declare
2    p person:=new person();
3    m mitarbeiter;
4  begin
5    select treat(p as mitarbeiter) into m from dual;
6    m.gehalt:=1000;
7  end;
```

Mit der Funktion **TREAT** können Sie verschiedene Objekt-Typen ineinander umwandeln. Voraussetzung dafür ist, dass sie zur gleichen Objekthierarchie gehören. Mit anderen Worten, *expr* muss ein Sub- oder Supertyp von *type* sein. Wenn Sie versuchen, ein Objekt in einen Objekttyp umzuwandeln, der in der Klassenhierarchie weiter untern steht (also spezieller ist) als der am meisten spezialisierte Typ der auf das Objekt anwendbar ist, dann gibt TREAT NULL zurück.

AUFRUF VON METHODEN DES SUPERTYPE – (SELF AS ….)

```
create or replace type PERSON force is object
(
  name          varchar2(100),
  vorname       varchar2(100),
  geburtsdatum date,

  constructor function PERSON return self as result,
  member procedure set_default
)
NOT FINAL

/

create or replace type body person is

  constructor function person return self as result is
  begin
    set_default();
    return;
  end;

  member procedure set_default is
  begin
    self.name          := 'Mustermann';
    self.vorname       := 'Max';
    self.geburtsdatum := sysdate;
    return;
  end;

end;

/
```

```
create or replace type MITARBEITER UNDER PERSON
(

  gehalt number,

  constructor function MITARBEITER return self as result
)

/

create or replace type body MITARBEITER is

  constructor function MITARBEITER return self as result is
  begin
    (self as PERSON).set_default();
    self.gehalt := 1000;
    return;
  end;

end;

/
```

OBJEKTTABELLEN & SQL SYNTAX

SQL | Output | Statistics

```
select * from int_dat_example
```

	ID	TEXT	STATUS.OBJECT_TYPE_NAME	STATUS.STATUSNAME	STATUS.STATUSID
1	1	erstellt:	APPFIA.CO_ZUIMPORTIEREN	defaults gesetzt	30
2	2	für Verarbeitung freigegeben:	APPFIA.CO_DEFAULTSGESETZT	defaults gesetzt	30
3	3	Fehler aufgetreten:	APPFIA.CO_FEHLER	Fehler aufgetreten	99
4	4	zurückgesetzt:	APPFIA.CO_DEFAULTSGESETZT	defaults gesetzt	30
5	5	erledigt:	APPFIA.CO_PROCESSED	erledigt	40

SQL | Output | Statistics

```
select id,text,status from int_dat_example
```

	ID	TEXT	STATUS.OBJECT_TYPE_NAME	STATUS.STATUSNAME	STATUS.STATUSID
1	1	erstellt:	APPFIA.CO_ZUIMPORTIEREN	defaults gesetzt	30
2	2	für Verarbeitung freigegeben:	APPFIA.CO_DEFAULTSGESETZT	defaults gesetzt	30
3	3	Fehler aufgetreten:	APPFIA.CO_FEHLER	Fehler aufgetreten	99
4	4	zurückgesetzt:	APPFIA.CO_DEFAULTSGESETZT	defaults gesetzt	30
5	5	erledigt:	APPFIA.CO_PROCESSED	erledigt	40

SQL | Output | Statistics

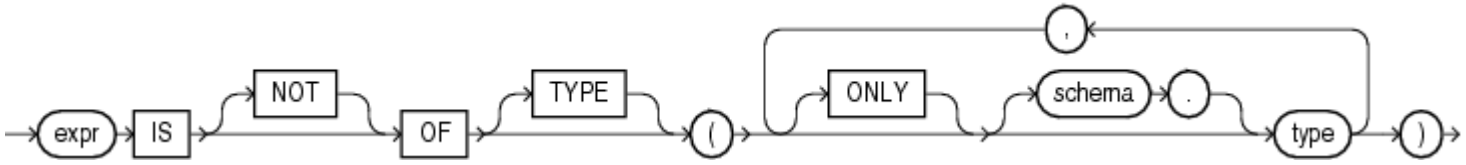
```
select id,text,status.statusname from int_dat_example
```

SQL | Output | Statistics

```
select t.id, t.text, t.status.statusname from int_dat_example t
```

	ID	TEXT	STATUS.STATUSNAME
1	1	erstellt:	defaults gesetzt
2	2	für Verarbeitung freigegeben:	defaults gesetzt
3	3	Fehler aufgetreten:	Fehler aufgetreten
4	4	zurückgesetzt:	defaults gesetzt
5	5	erledigt:	erledigt

SQL SYNTAX - IS [NOT] OF ...



SQL | Output | Statistics

```
select t.id,t.text,t.status from int_dat_example t
where t.status is of (APPFIA.CO_DEFAULTSGESETZT)
```

	ID	TEXT	STATUS.OBJECT_TYPE_NAME	STATUS.STATUSNAME	STATUS.STATUSID
1	12	Ich bin ein Beispieltext!	APPFIA.CO_DEFAULTSGESETZT	defaults gesetzt	30

SQL | Output | Statistics

```
select t.id,t.text,t.status from int_dat_example t
where t.status is of (APPFIA.CO_DEFAULTSGESETZT,APPFIA.CO_ZUIMPORTIEREN)
```

	ID	TEXT	STATUS.OBJECT_TYPE_NAME	STATUS.STATUSNAME	STATUS.STATUSID
1	12	Ich bin ein Beispieltext!	APPFIA.CO_DEFAULTSGESETZT	defaults gesetzt	30
2	13	Ein anderer Text	APPFIA.CO_ZUIMPORTIEREN	zu importieren	10

SQL | Output | Statistics

```
select t.id,t.text,t.status from int_dat_example t
where t.status is not of (APPFIA.CO_DEFAULTSGESETZT)
```

	ID	TEXT	STATUS.OBJECT_TYPE_NAME	STATUS.STATUSNAME	STATUS.STATUSID
1	13	Ein anderer Text	APPFIA.CO_ZUIMPORTIEREN	zu importieren	10

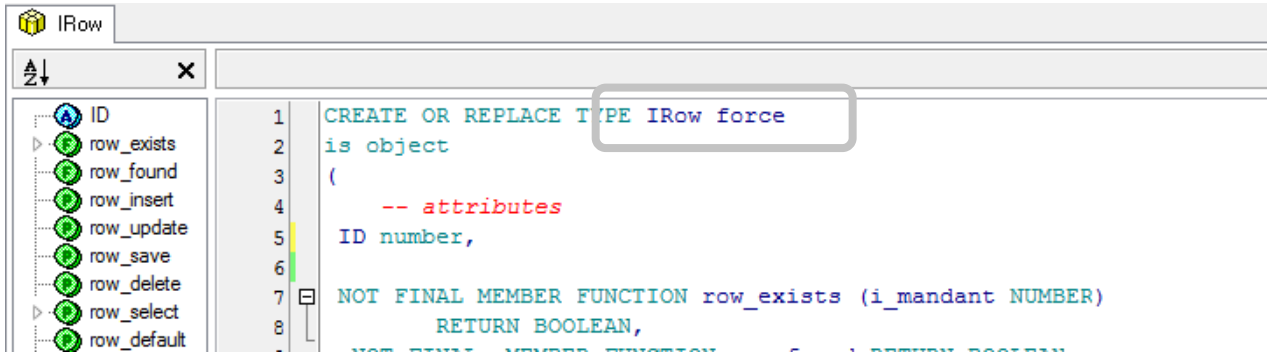
ANPASSUNG VON BESTEHENDEN TYPEN – TYPE EVOLUTION

Anpassung von Typen ohne abhängige Objekte

- Typen, von denen keine weiteren Typen abgeleitet sind und die nicht in einer Tabelle als Datentyp für eine Spalte genutzt werden, können beliebig geändert werden. Die Anpassung funktioniert wie bei der Änderung eines einfachen Packages.

Anpassung von Typen mit abhängigen Objekten

- Typen mit abhängigen Objekten können nicht einfach geändert werden
- Sind die abhängigen Objekte selbst nur Typen kann das Schlüsselwort **FORCE** genutzt werden



- Wird der Typ zum Beispiel als Typ für eine Tabellenspalte genutzt, darf der Typ nur per **Type Evolution** angepasst werden, da sonst die bereits in der Tabelle abgelegten Objektinstanzen nicht mehr lesbar wären. In diesem Fall darf auf keinen Fall FORCE eingesetzt werden.

TYPE EVOLUTION

- Wird der zu ändernde Typ als Typ für eine Tabellenspalte genutzt, darf er nur per Type Evolution angepasst werden, da sonst die bereits in der Tabelle abgelegten Objektinstanzen nicht mehr lesbar wären. In diesem Fall darf **auf keinen Fall** FORCE eingesetzt werden.

```
SQL> alter type co_fehler add attribute (fehlercode number, fehlermessage varchar2(4000)) cascade;

Type altered
```

```
SQL> desc CO_FEHLER;
Element                        Type
------------------------------ -------------
OBJECT_TYPE_NAME               VARCHAR2(100)
STATUSNAME                     VARCHAR2(100)
STATUSID                       NUMBER
DBMS_OUTPUT                    PROCEDURE
TO_STRING                      FUNCTION
COMPARE                        FUNCTION
COMPARE2                       FUNCTION
GETTYPE                        FUNCTION
GETOBJASANYDATA                FUNCTION
DSFUERVERARBEITUNGFREIGEBEN    FUNCTION
DSZURUECKSETZEN                FUNCTION
DSALSERLEDIGTKENNZEICHNEN      FUNCTION
FEHLERAUFGETRETEN              FUNCTION
CO_FEHLER                      FUNCTION
```

```
SQL> desc CO_FEHLER;
Element                        Type
------------------------------ --------------
OBJECT_TYPE_NAME               VARCHAR2(100)
STATUSNAME                     VARCHAR2(100)
STATUSID                       NUMBER
FEHLERCODE                     NUMBER
FEHLERMESSAGE                  VARCHAR2(4000)
DBMS_OUTPUT                    PROCEDURE
TO_STRING                      FUNCTION
COMPARE                        FUNCTION
COMPARE2                       FUNCTION
GETTYPE                        FUNCTION
GETOBJASANYDATA                FUNCTION
DSFUERVERARBEITUNGFREIGEBEN    FUNCTION
DSZURUECKSETZEN                FUNCTION
DSALSERLEDIGTKENNZEICHNEN      FUNCTION
FEHLERAUFGETRETEN              FUNCTION
CO_FEHLER                      FUNCTION
```

- Die Verwendung von TREAT bei der Abfrage ist notwendig, da die zusätzlichen Attribute im Subtyp deklariert sind.

SQL | Output | Statistics

```
select t.id,t.text,treat(t.status as APPFIA.CO_FEHLER) as status from int_dat_example t
where t.status is of (APPFIA.CO_FEHLER)
```

	ID	TEXT	STATUS.OBJECT_TYPE_NAME	STATUS.STATUSNAME	STATUS.STATUSID	STATUS.FEHLERCODE	STATUS.FEHLERMESSAGE
1	14	Ein neuer Datensatz	APPFIA.CO_FEHLER	Fehler aufgetreten	99		

TYPE EVOLUTION

- Die durch ALTER TYPE vorgenommenen Änderungen wirken sich nur auf die Spezifikation des Typs aus. Sind Änderungen am Body nötig, werden diese normal durchgeführt und der Body einzeln kompiliert.
- Die Änderungen an der Spezifikation werden in Form von ALTER TYPE – Befehlen angezeigt.
- Die Option CASCADE des ALTER TYPE Befehls propagiert die Änderungen am Typ an alle abhängigen Typen und Tabellen.

```
create or replace type co_Fehler force under IStatus
(
  -- Member functions and procedures
  overriding member function DSfuerVerarbeitungFreigeben return IStatus,
  overriding member function DSzuruecksetzen return IStatus,
  overriding member function DSalsErledigtKennzeichnen return IStatus,
  overriding member function FehlerAufgetreten return IStatus,

  CONSTRUCTOR FUNCTION co_Fehler RETURN SELF AS RESULT
)
```

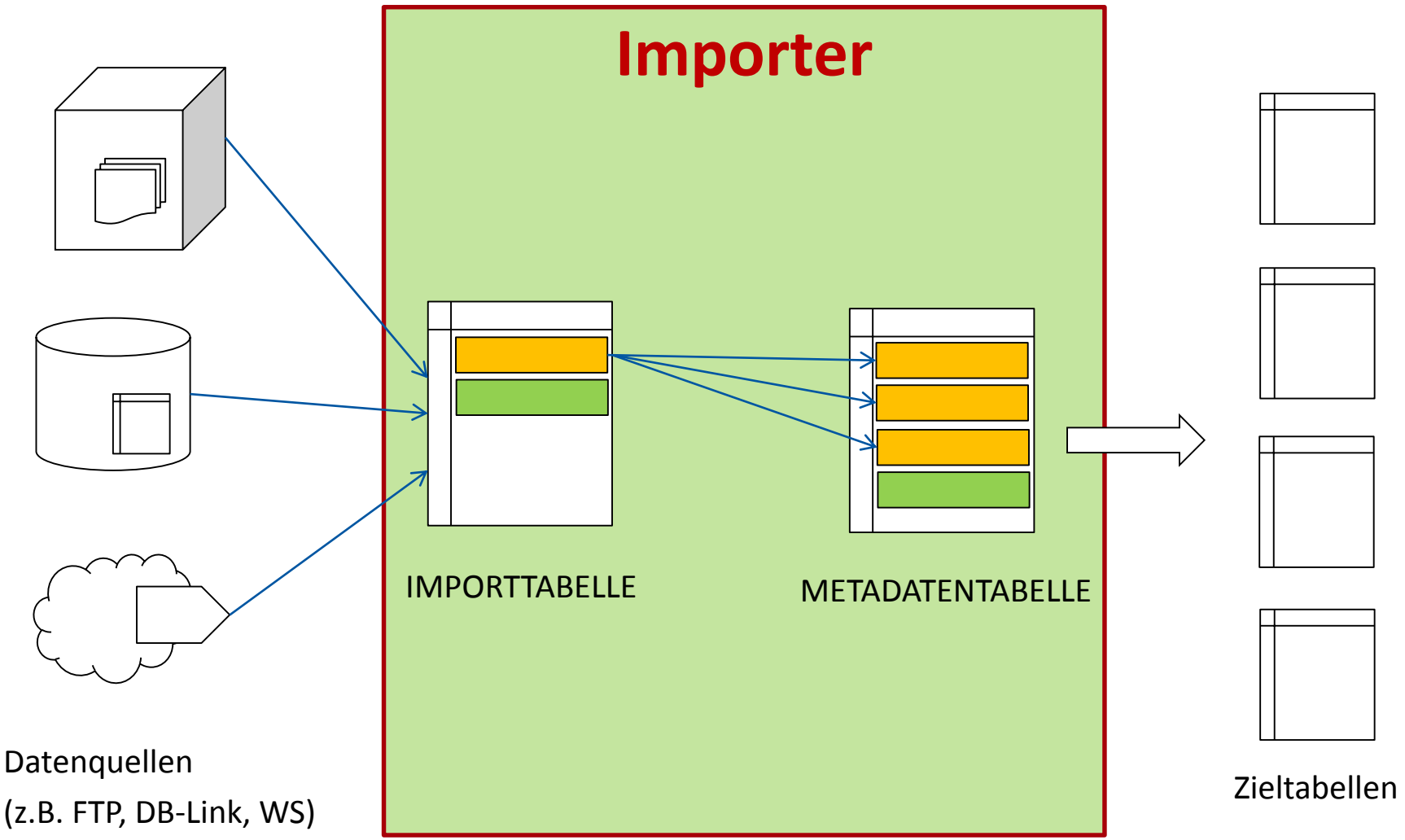
```
alter type co_fehler add attribute (fehlercode number, fehlermessage varchar2(4000)) cascade
```

TYPE EVOLUTION

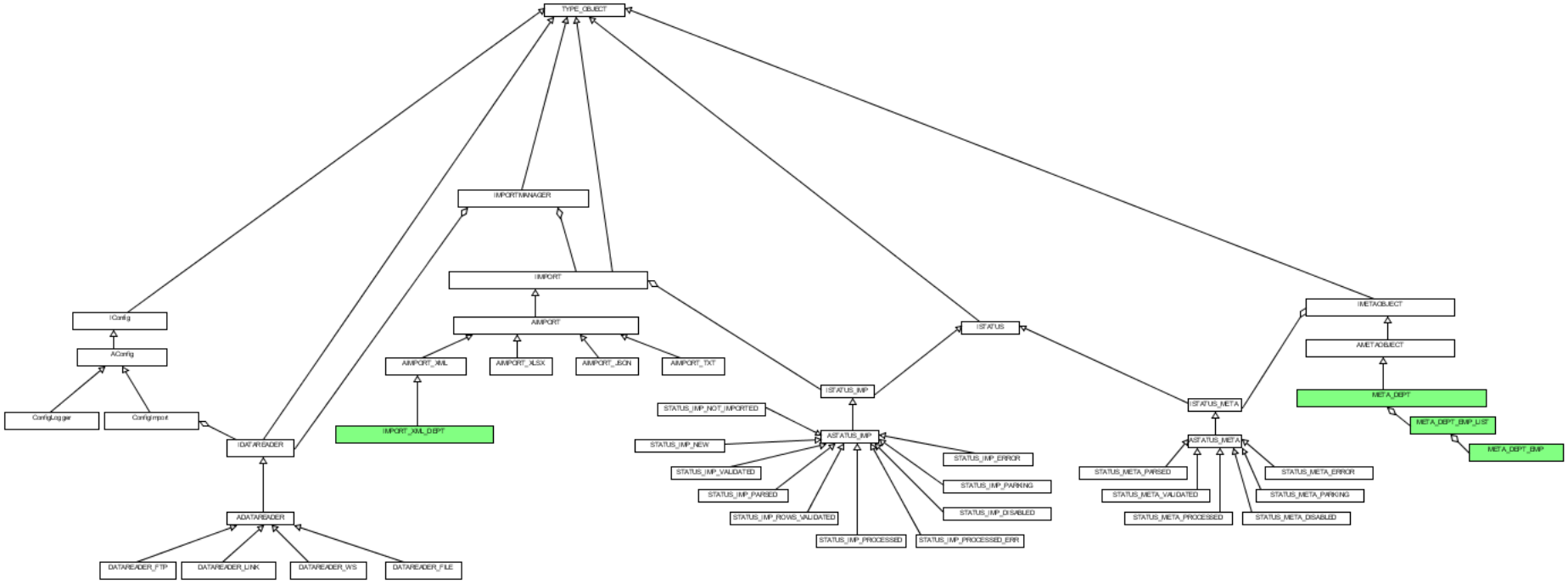
- Folgende Änderungen an einem Typ sind mit Hilfe von Type Evolution möglich:
  - Hinzufügen und Löschen von Attributen und Methoden
  - Modifizieren / Vergrößern von alphanumerischen und numerischen Attributen
  - Ändern der FINAL und INSTANTIABLE Eigenschaft des Typs
  - Anpassen von Limit und Größe eines VARRAYs
  - Anpassen von length, precision und scale von Collection Elementen

Pilotprojekt

PILOTPROJEKT



PILOTPROJEKT



Demo

Erfahrungen

ERFAHRUNGEN

PRO

- Zwingt den Entwickler sich vorab genaue Gedanken über seine Architektur zu machen, sofern er Objekttabellen einsetzen möchte.
- Abbildung reiner Businesslogik oder Zugriffsschicht kommt ohne Type Evolution aus
- Diverse Design Pattern lassen sich so umsetzen und deren Vorteile nutzen
- Kleine überschaubare Codeblöcke
- (Trennung von Daten und Businesslogik möglich)
- Lesbarkeit des Codes verbessert

CONTRA

- Zwingt den Entwickler sich vorab genaue Gedanken über seine Architektur zu machen, sofern er Objekttabellen einsetzen möchte.
- Bei Anpassung abhängiger Typen muss Type Evolution genutzt werden.
- Keine privaten Attribute möglich
- sehr viele einzelne Objekte (Klassendiagramm hilft den Überblick zu behalten)

Weiterführende Informationen

WEITERFÜHRENDE INFORMATIONEN

Design Pattern
http://www.philipphauer.de/study/se/design-pattern.php
https://de.wikipedia.org/wiki/Entwurfsmuster
http://www.amazon.de/Patterns-Elements-Reusable-Object-Oriented-Software/dp/0201633612
http://www.oodesign.com/

Oracle Object Types
https://docs.oracle.com/database/121/ADOBJ/adobjint.htm#ADOBJ001
http://plsql-object-types.blogspot.de/2008/10/object-oriented-rowtype.html

Sonstiges
http://apex2rule-the-world.blogspot.de

Type Evolution
http://docs.oracle.com/cd/B28359_01/server.111/b28286/conditions014.htm
http://docs.oracle.com/cd/B28359_01/appdev.111/b28371/adobjadv.htm#i1006726
http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_4002.htm#i2057828
http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/alter_type.htm#i2112566

HERZLICHEN DANK!

buw Unternehmensgruppe | Anja Hildebrandt | Entwicklung
Rheiner Landstraße 195 | D-49078 Osnabrück
tel +49 541 9462-22827 | mobil +49 173 2896 264 | fax +49 541 9462-987
anja.hildebrandt@buw.de

anjeli2001 anjeli2001 Anja_Hildebrandt Anja Hildebrandt

www.buw.de