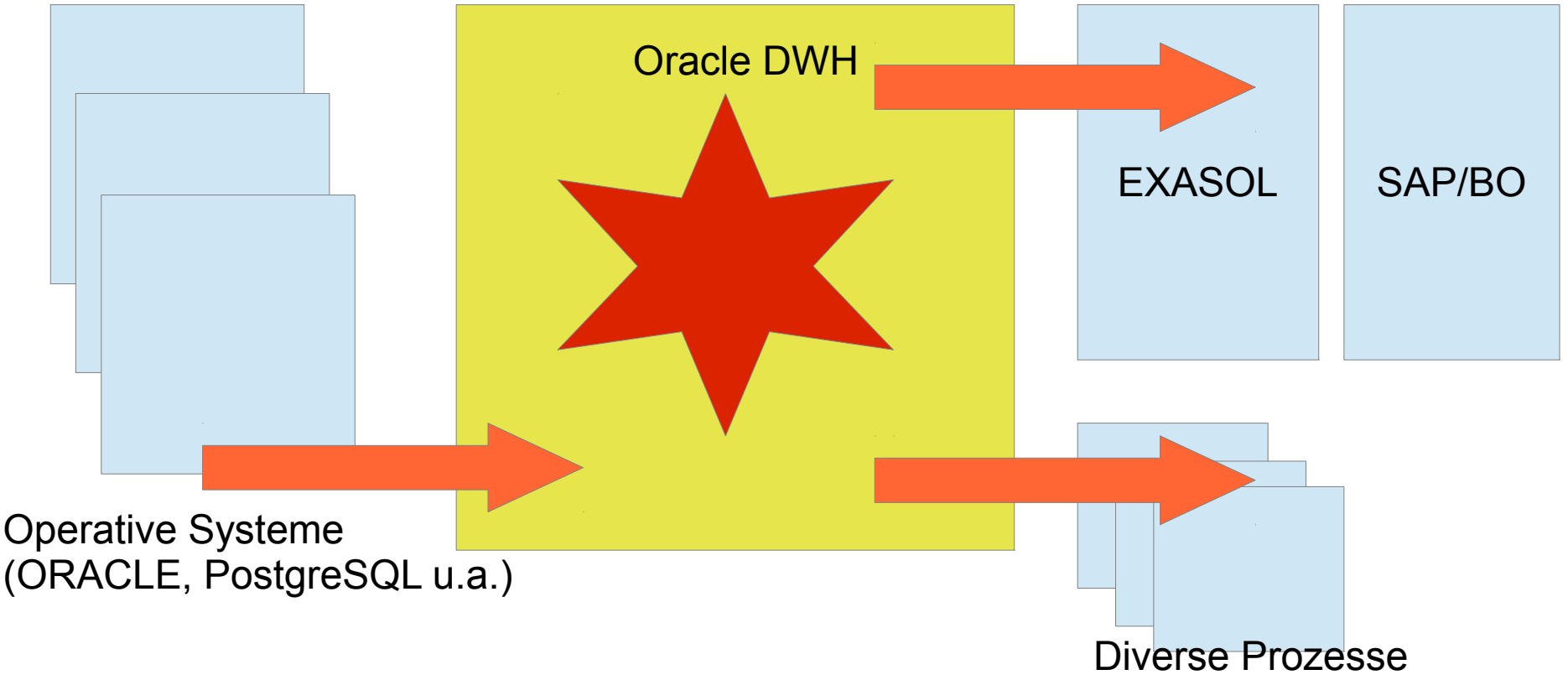
Der Einsatz von Oracle Job-Chains im ETL-Prozeß

Gesamtsituation beim Kunden

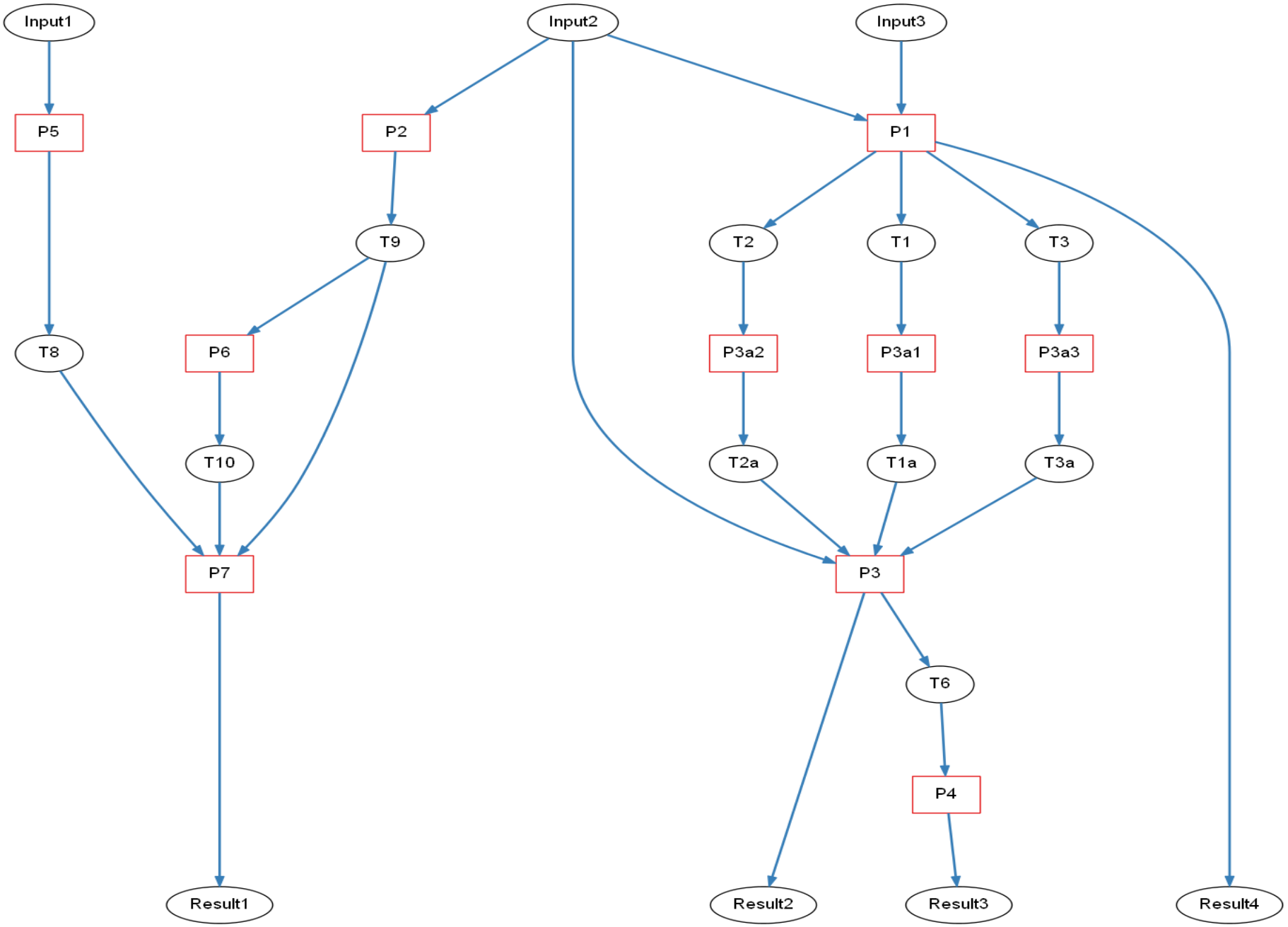


Ausgangslage

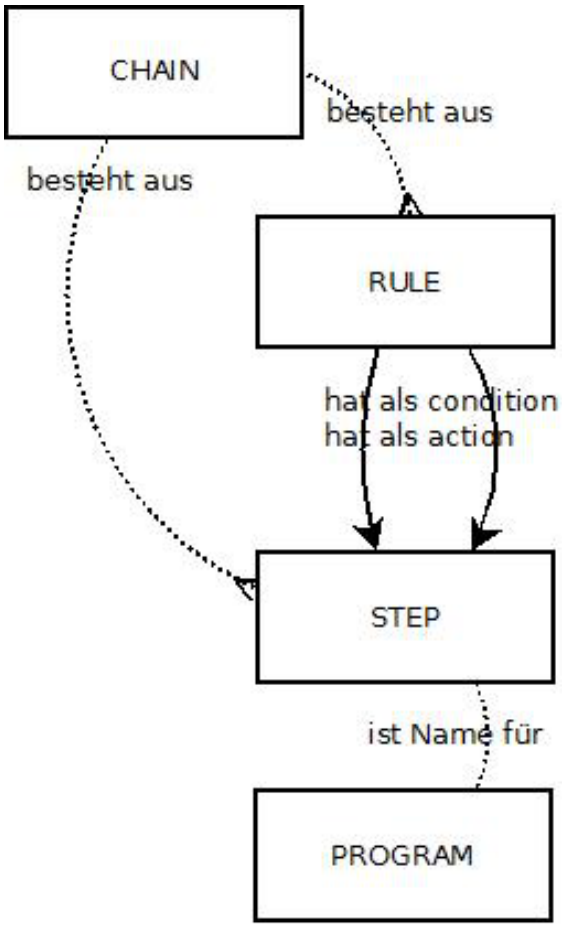
- Zeitfenster 01:00 bis 08:00
- Datenmenge wird stark steigen
- Zeitfenster wird schon mit der jetzigen Datenmenge nur eingehalten, wenn alles glatt geht
- Ca. 30 Verarbeitungsschritte wurden sequentiell abgearbeitet

Durch Job-Chains haben wir erreicht:

- Verringerung der Durchlaufzeit im ersten Anlauf um 2 h
- Fehler sind leichter lokalisierbar
- Nach der Fehlerkorrektur kann leicht wieder gestartet werden
- Von Fehlern nicht betroffene Schritte laufen weiter


Input1
Input2
Input3
P5
P2
P1
T9
T2
T1
T3
T8
P6
P3a2
P3a1
P3a3
T10
T2a
T1a
T3a
P7
P3
T6
P4
Result1
Result2
Result3
Result4

Job-Chains mit DBMS_SCHEDULER



RULE: CONDITION/ACTION

Vorbedingungen:

- TRUE ist immer wahr
- XXX SUCCEEDED
- XXX COMPLETED
- XXX FAILED

Verknüpfungen mit AND und OR

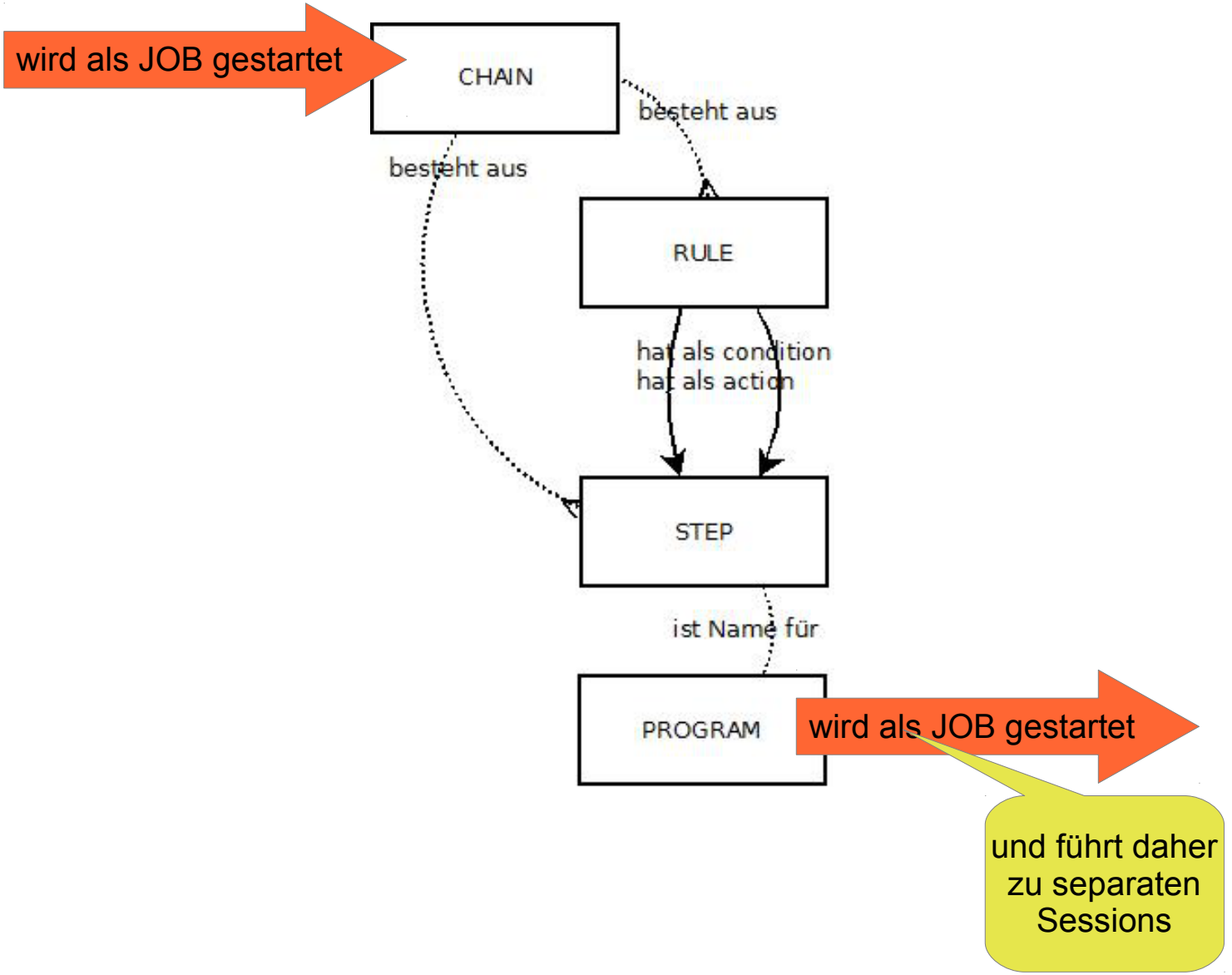
RULE: CONDITION/ACTION

Aktionen:

- END beendet die CHAIN
- START XXX

Es können auch mehrere Schritte gestartet werden

Zur Laufzeit



Die wichtigsten Subprozeduren von DBMS_SCHEDULER (1/4)

```
DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name              IN VARCHAR2,
   rule_set_name           IN VARCHAR2 DEFAULT NULL,
   evaluation_interval     IN INTERVAL DAY TO SECOND DEFAULT NULL,
   comments                IN VARCHAR2 DEFAULT NULL);
```

Die wichtigsten Subprozeduren von DBMS_SCHEDULER (2/4)

```
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name              IN VARCHAR2,
   program_type              IN VARCHAR2,
   program_action            IN VARCHAR2,
   number_of_arguments       IN PLS_INTEGER DEFAULT 0,
   enabled                   IN BOOLEAN DEFAULT FALSE,
   comments                  IN VARCHAR2 DEFAULT NULL);
```

Die wichtigsten Subprozeduren von DBMS_SCHEDULER (3/4)

```
DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
   chain_name              IN VARCHAR2,
   step_name               IN VARCHAR2,
   program_name            IN VARCHAR2);
```

Die wichtigsten Subprozeduren von DBMS_SCHEDULER (4/4)

```
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   chain_name              IN VARCHAR2,
   condition               IN VARCHAR2,
   action                  IN VARCHAR2,
   rule_name               IN VARCHAR2 DEFAULT NULL,
   comments                IN VARCHAR2 DEFAULT NULL);
```

Systemviews der statischen Struktur

- DBA_SCHEDULER_CHAIN_RULES
- DBA_SCHEDULER_CHAIN_STEPS
- DBA_SCHEDULER_JOBS
- DBA_SCHEDULER_PROGRAMS

Systemviews zur Laufzeitkontrolle

- DBA_SCHEDULER_RUNNING_CHAINS
- DBA_SCHEDULER_RUNNING_JOBS
- DBA_SCHEDULER_JOB_LOG
- DBA_SCHEDULER_JOB_RUN_DETAILS

(in dieser Tabelle ist die Log-Id der gesamten Chain in der Spalte ADDITIONAL_INFO zu finden.)

Laufzeitkontrolle

```
select
JOB_NAME, CHAIN_NAME, STEP_NAME, STATE, ERROR_CODE,START_DATE,END_DATE,
DURATION
from DBA_SCHEDULER_RUNNING_CHAINS
order by OWNER, JOB_NAME, START_DATE
```

JOB_NAME	CHAIN_NAME	STEP_NAME	STATE	ERROR_CODE	START_DATE	END_DATE	DURATION
RUN_DB2	C_DB2	S_0000_START	SUCCEEDED	0	31.03.12 14:25:04,539470 +02:00	31.03.12 14:25:10,601074 +02:00	+000000000
RUN_DB2	C_DB2	S_0102_SERVICE	SUCCEEDED	0	31.03.12 14:25:10,722009 +02:00	31.03.12 16:08:15,205578 +02:00	+000000000
RUN_DB2	C_DB2	S_0103_RETOURE	SUCCEEDED	0	31.03.12 14:25:10,743131 +02:00	31.03.12 16:22:12,860838 +02:00	+000000000
RUN_DB2	C_DB2	S_0104_LOGISTIK	FAILED	4063	31.03.12 14:25:10,827835 +02:00	31.03.12 14:25:33,762727 +02:00	+000000000
RUN_DB2	C_DB2	S_0401_PAYMENT_START	SUCCEEDED	0	31.03.12 16:32:21,919985 +02:00	31.03.12 16:32:21,920017 +02:00	+000000000
RUN_DB2	C_DB2	S_0411_PAYM_FILL_BI	SUCCEEDED	0	31.03.12 16:32:23,430245 +02:00	31.03.12 17:01:34,108099 +02:00	+000000000
RUN_DB2	C_DB2	S_0412_PAYM_FILL_ORDER	SUCCEEDED	0	31.03.12 16:32:23,529822 +02:00	31.03.12 16:56:04,032304 +02:00	+000000000
RUN_DB2	C_DB2	S_0413_PAYM_FILL_NOTI	SUCCEEDED	0	31.03.12 16:32:23,530043 +02:00	31.03.12 17:12:16,675682 +02:00	+000000000
RUN_DB2	C_DB2	S_0414_PAYM_FILL_REMD	SUCCEEDED	0	31.03.12 16:32:23,597860 +02:00	31.03.12 16:35:48,366374 +02:00	+000000000
RUN_DB2	C_DB2	S_0420_PAYMENT_PHASE2	SUCCEEDED	0	31.03.12 17:12:16,690766 +02:00	31.03.12 17:44:19,272098 +02:00	+000000000
RUN_DB2	C_DB2	S_0302_MERGE_SERVICE	NOT_STARTED				
RUN_DB2	C_DB2	S_0202_AGG_LOGI_UNIV	NOT_STARTED				
RUN_DB2	C_DB2	S_0201_AGG_ORDER	NOT_STARTED				

Weiterlauf nach Korrektur

```
begin
dbms_scheduler.alter_running_chain  (
job_name=>'RUN_DB2',
step_name=>'S_0104_LOGISTIK',
attribute=>'STATE',
value=>'SUCCEEDED'
);
end;
```

Anforderungen an Programme (1/2)

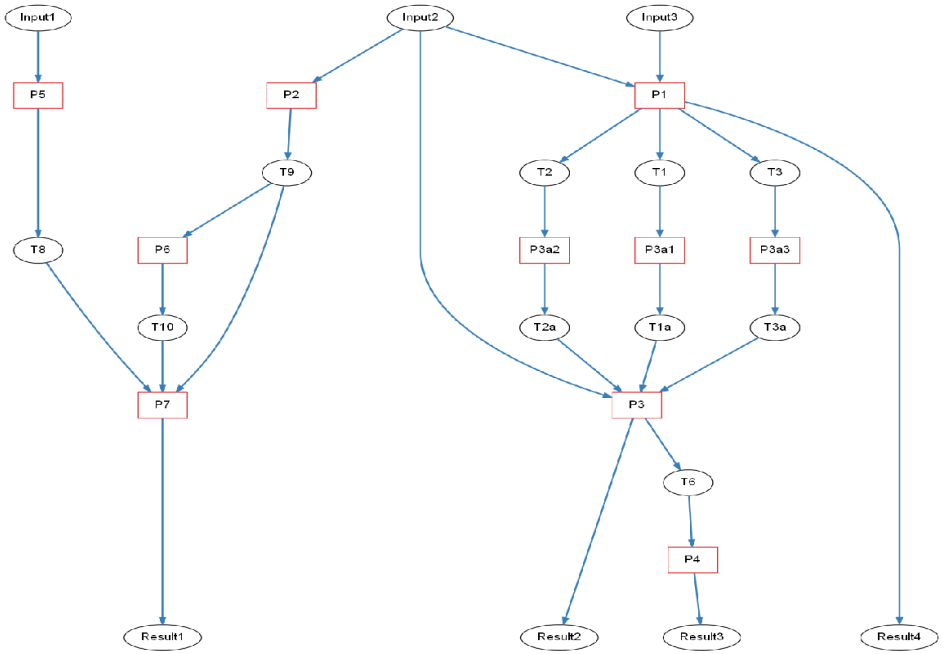
- Programme dürfen keine OUT-Parameter benutzen
- IN-Parameter sind umständlich
- Am günstigsten sind Steuertabellen zum Datenaustausch, eventuell in Wrappern realisiert

Anforderungen an Programme (2/2)

- Programme dürfen keine EXCEPTIONs verschlucken, Fehler müssen eskaliert werden
- RAISE oder RAISE_APPLICATION_ERROR einsetzen

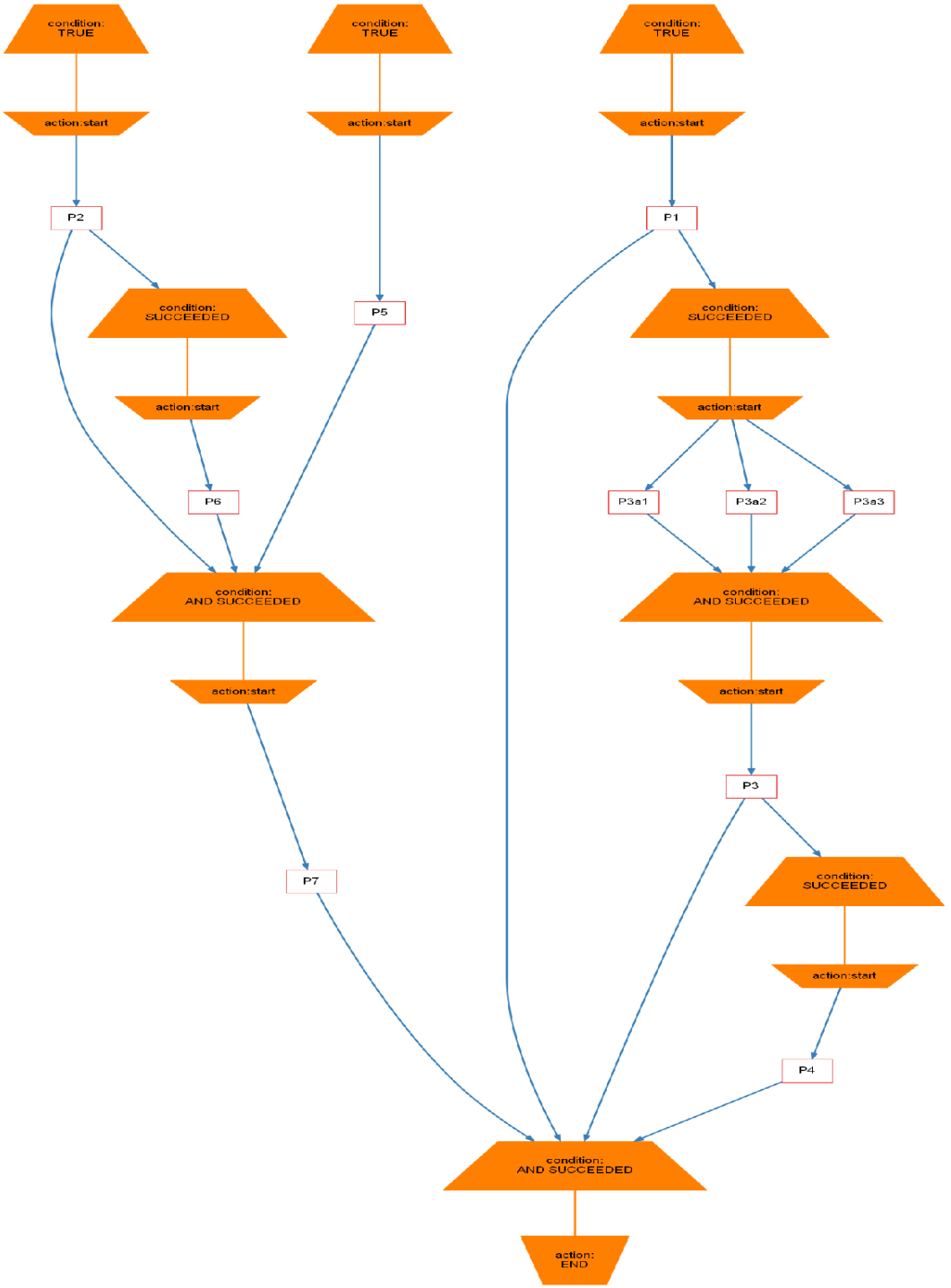
Vorgehensweise: Abhängigkeiten dokumentieren

- Es muß erkennbar werden, welches Programm welche Tabelle liest und schreibt



Vorgehensweise: Vom Programm aus rückwärts sehen:

- Welche Tabellen werden gelesen und bearbeitet ?
- Welche Programme bringen die Tabellen in den Zustand, der gebraucht wird ?
- Alle diese Programme müssen erfolgreich abgeschlossen sein , bevor das betrachtete Programm starten kann.


condition:
TRUE
condition:
TRUE
condition:
TRUE
action:start
action:start
action:start
P2
P1
condition:
SUCCEEDED
P5
condition:
SUCCEEDED
action:start
action:start
P6
P3a1
P3a2
P3a3
condition:
AND SUCCEEDED
condition:
AND SUCCEEDED
action:start
action:start
P3
P7
condition:
SUCCEEDED
action:start
P4
condition:
AND SUCCEEDED
action:
END

Weitere Regeln für CHAINs

- Eine RULE muß die Vorbedingung TRUE haben
- Eine RULE muß die Aktion END haben
- Keine STEPs erlauben, die keinen Nachfolger haben (lose Enden)
- Mehrfacher Start einer CHAIN sollte abgefangen werden

Wie werden Chains praktikabel ?

CHAINS können deutlich über 100 STEPs enthalten, da müssen zahlreiche Informationen zusammen passen, daraus ergibt sich :

- Man muß CHAINs bearbeiten können wie Sourcecode und sie reproduzierbar wieder herstellen können.
- Man muß die Richtigkeit der CHAIN-Logik kontrollieren können, ohne sie laufen zu lassen

Chain-'Sourcecode': CHAIN + STEPs

```
-- Anlegen aller Steps und Rules  der Chain  C_DB2, generiert am 29.09.11
-- Diesen Text als "C_DB2.sql" im Unterverzeichnis "scripts" speichern und mit svn versionieren
-- $HeadURL: xxxx $
-- $Id: c_db2.sql 1086 2011-10-19 10:00:19Z norbert.klamann $
DECLARE
L_CHAIN_NAME varchar2(30) := 'C_DB2';
  L_COMMENT       varchar2(200) := 'Berechnung der Deckungsbeiträge II';-- Anzahlen (vor diesen Änderungen) : steps 14 rules 15
BEGIN
PKG_UTIL_SCHEDULING.CREATE_OR_REPLACE_CHAIN(L_CHAIN_NAME, L_COMMENT, P_FORCE => TRUE);
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0000_START','P_AGG_PREPARE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0102_SERVICE','P_SERVICE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0103_RETOURE','P_RETOURE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0104_LOGISTIK','P_LOGISTIK');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0201_AGG_ORDER','P_AGG_ORDER');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0202_AGG_LOGI_UNIV','P_AGG_LOGI_UNIV');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0302_MERGE_SERVICE','P_AGG_SERVICE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0401_PAYMENT_START','P_PAYMENT_START');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0411_PAYM_FILL_BI','P_PAYMENT_FILL_BI_PCTRL_LOG');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0412_PAYM_FILL_ORDER','P_PAYMENT_FILL_ORDER');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0413_PAYM_FILL_NOTI','P_PAYMENT_FILL_PAYMENT_NOTI');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0414_PAYM_FILL_REMD','P_PAYMENT_FILL_REMINDER');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0420_PAYMENT_PHASE2','P_PAYMENT_PHASE2');
--
```

Nicht gezeigt:
die Definitionen
der
PROGRAMs

Chain-'Sourcecode': RULEs und JOB

STEPS müssen existieren

```
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'TRUE','START "S_0000_START"', comments=> 'V bereitung');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START "S_0401_PAYMENT_START"', comments=> 'kann
ohne Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START "S_0102_SERVICE"', comments=> 'kann ohne
Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START "S_0103_RETOURE"', comments=> 'kann ohne
Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START "S_0104_LOGISTIK"', comments=> 'kann ohne
Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START "S_0411_PAYM_FILL_BI"', comments=>
'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START "S_0412_PAYM_FILL_ORDER"',
comments=> 'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START "S_0413_PAYM_FILL_NOTI"', comments=>
'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START "S_0414_PAYM_FILL_REMD"', comments=>
'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0411_PAYM_FILL_BI SUCCEEDED AND S_0412_PAYM_FILL_ORDER SUCCEEDED AND
S_0413_PAYM_FILL_NOTI SUCCEEDED AND S_0414_PAYM_FILL_REMD SUCCEEDED ','START "S_0420_PAYMENT_PHASE2"', comments=> 'Nach den
Berechnungen aggregieren');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0420_PAYMENT_PHASE2 SUCCEEDED AND S_0102_SERVICE SUCCEEDED AND
S_0103_RETOURE SUCCEEDED AND S_0104_LOGISTIK SUCCEEDED ','START "S_0201_AGG_ORDER"', comments=> 'Nach den Berechnungen
aggregieren');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0104_LOGISTIK SUCCEEDED','START "S_0202_AGG_LOGI_UNIV"', comments=>
'Logistik-Universum kann parallel laufen');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0201_AGG_ORDER SUCCEEDED','START "S_0302_MERGE_SERVICE"', comments=>
'weitere parallele Merges');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0302_MERGE_SERVICE COMPLETED AND S_0202_AGG_LOGI_UNIV COMPLETED ','END ',
comments=> 'Chain fertig.');
DBMS_SCHEDULER.ENABLE(name => L_CHAIN_NAME);
 PKG_UTIL_SCHEDULING.CREATE_OR_REPLACE_SC_JOB(P_JOB_NAME =>'RUN_DB2',P_CHAIN_NAME=> l_chain_name, P_SCHED_NAME => '',
P_COMMENTS=> 'DB2-Prozess',P_FORCE => TRUE);
END;
```

Logik der Chain visualisieren

Kann man gut mit Graphviz (http://www.graphviz.org)

- Sourcecode für Graphviz mit SQL generieren
- Sourcecode in gvedit laden
- Graphik generieren
- Mit MSPAINT kann man große Graphiken auf mehrere Blatt Papier aufteilen

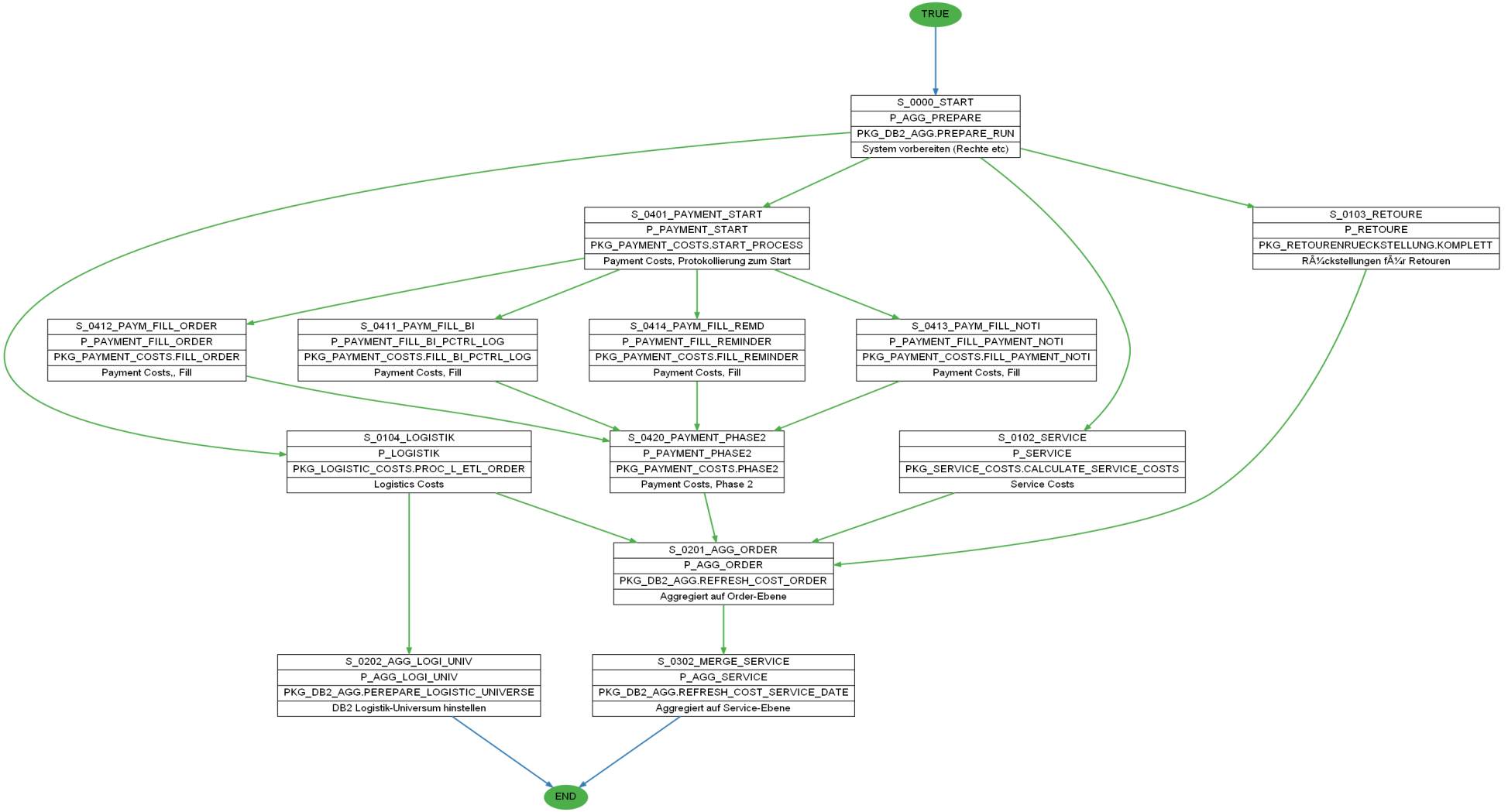
Graphviz-Sourcecode (1/2)

```
/* Graphviz/dot- Dokumentation der Chain xxx.C_DB2, generiert am 31.03.12
 speichern als Datei C_DB2.gv, dann mit Graphviz starten
*/
digraph  C_DB2  {
    ratio=auto; charset=latin1; center=true; fontname=Helvetica; pad=0.5;
    concentrate=true;ranksep="1.0 equally"; nodesep=1.0;colorscheme="set19";
    node [fontname=Helvetica, colorscheme="set19", color="black"];
    edge [fontname=Helvetica, colorscheme="set19", penwidth=2, color=2];
  label="Jobchain C_DB2, generiert am : 31.03.12";
TRUE [style=filled,color=3];
END  [style=filled,color=3];
S_0302_MERGE_SERVICE [shape="record",label="{S_0302_MERGE_SERVICE|P_AGG_SERVICE|PKG_DB2_AGG.REFRESH_COST_SERVICE_DATE|Aggregiert\ auf\
Service-Ebene}"] ;
S_0412_PAYM_FILL_ORDER [shape="record",label="{S_0412_PAYM_FILL_ORDER|P_PAYMENT_FILL_ORDER|PKG_PAYMENT_COSTS.FILL_ORDER|Payment\ Costs,,\
Fill}"] ;
S_0420_PAYMENT_PHASE2 [shape="record",label="{S_0420_PAYMENT_PHASE2|P_PAYMENT_PHASE2|PKG_PAYMENT_COSTS.PHASE2|Payment\ Costs,\ Phase\ 2}"] ;
S_0201_AGG_ORDER [shape="record",label="{S_0201_AGG_ORDER|P_AGG_ORDER|PKG_DB2_AGG.REFRESH_COST_ORDER|Aggregiert\ auf\ Order-Ebene}"] ;
S_0202_AGG_LOGI_UNIV [shape="record",label="{S_0202_AGG_LOGI_UNIV|P_AGG_LOGI_UNIV|PKG_DB2_AGG.PEREPARE_LOGISTIC_UNIVERSE|DB2\ Logistik-
Universum\ hinstellen}"] ;
S_0104_LOGISTIK [shape="record",label="{S_0104_LOGISTIK|P_LOGISTIK|PKG_LOGISTIC_COSTS.PROC_L_ETL_ORDER|Logistics\ Costs}"] ;
S_0401_PAYMENT_START [shape="record",label="{S_0401_PAYMENT_START|P_PAYMENT_START|PKG_PAYMENT_COSTS.START_PROCESS|Payment\ Costs,\
Protokollierung\ zum\ Start}"] ;
S_0411_PAYM_FILL_BI [shape="record",label="{S_0411_PAYM_FILL_BI|P_PAYMENT_FILL_BI_PCTRL_LOG|PKG_PAYMENT_COSTS.FILL_BI_PCTRL_LOG|Payment\
Costs,\ Fill}"] ;
S_0414_PAYM_FILL_REMD [shape="record",label="{S_0414_PAYM_FILL_REMD|P_PAYMENT_FILL_REMINDER|PKG_PAYMENT_COSTS.FILL_REMINDER|Payment\ Costs,\
Fill}"] ;
S_0102_SERVICE [shape="record",label="{S_0102_SERVICE|P_SERVICE|PKG_SERVICE_COSTS.CALCULATE_SERVICE_COSTS|Service\ Costs}"] ;
S_0413_PAYM_FILL_NOTI [shape="record",label="{S_0413_PAYM_FILL_NOTI|P_PAYMENT_FILL_PAYMENT_NOTI|PKG_PAYMENT_COSTS.FILL_PAYMENT_NOTI|Payment\
Costs,\ Fill}"] ;
S_0103_RETOURE [shape="record",label="{S_0103_RETOURE|P_RETOURE|PKG_RETOURENRUECKSTELLUNG.KOMPLETT|Rückstellungen\ für\ Retouren}"] ;
S_0000_START [shape="record",label="{S_0000_START|P_AGG_PREPARE|PKG_DB2_AGG.PREPARE_RUN|System\ vorbereiten\ (Rechte\ etc)}"] ;
TRUE -> S_0000_START;
```

Graphviz-Sourcecode (2/2)

```
TRUE -> S_0000_START;
S_0411_PAYM_FILL_BI -> S_0420_PAYMENT_PHASE2[color=3];
S_0412_PAYM_FILL_ORDER -> S_0420_PAYMENT_PHASE2[color=3];
S_0413_PAYM_FILL_NOTI -> S_0420_PAYMENT_PHASE2[color=3];
S_0414_PAYM_FILL_REMD -> S_0420_PAYMENT_PHASE2[color=3];
S_0420_PAYMENT_PHASE2 -> S_0201_AGG_ORDER[color=3];
S_0102_SERVICE -> S_0201_AGG_ORDER[color=3];
S_0103_RETOURE -> S_0201_AGG_ORDER[color=3];
S_0104_LOGISTIK -> S_0201_AGG_ORDER[color=3];
S_0104_LOGISTIK -> S_0202_AGG_LOGI_UNIV[color=3];
S_0201_AGG_ORDER -> S_0302_MERGE_SERVICE[color=3];
S_0302_MERGE_SERVICE -> END;
S_0202_AGG_LOGI_UNIV -> END;
S_0000_START -> S_0401_PAYMENT_START[color=3];
S_0000_START -> S_0102_SERVICE[color=3];
S_0000_START -> S_0103_RETOURE[color=3];
S_0000_START -> S_0104_LOGISTIK[color=3];
S_0401_PAYMENT_START -> S_0411_PAYM_FILL_BI[color=3];
S_0401_PAYMENT_START -> S_0412_PAYM_FILL_ORDER[color=3];
S_0401_PAYMENT_START -> S_0413_PAYM_FILL_NOTI[color=3];
S_0401_PAYMENT_START -> S_0414_PAYM_FILL_REMD[color=3];
}
```

Graphviz-Ausgabe



Namenskonventionen

"There are only two hard things in Computer Science: cache invalidation, naming things and off by 1 errors"

- PROGRAMS können nicht genauso heißen wie PACKAGE.PROZEDUR-Aufrufe
- STEP-Namen können nur 27 Zeichen lang sein
- Es werden **viele** Namen gebraucht

  Empfehlung : keine strukturierten Namen, fachlich benennen

Bewertung: Schwächen

- Falsche STEP-Namen in der RULE-Definition werden nicht zur 'Kompilezeit' abgefangen sondern führen zu einer Flut an Jobs zur Laufzeit.
- Es müssen sehr viele DB-Objekte eingerichtet werden
- Man sollte PROGRAMs nicht verwenden müssen, wenn man sie nicht fachlich braucht

Bewertung: Chains sind nützlich

Job-Chains waren eine gute Investition:

- Durch Parallelität haben wir Zeit gewonnen
- Durch Restartfähigkeit haben wir Zuverlässigkeit und damit Standing nach außen gewonnen
- Mit einigen kleineren Tools und etwas Erfahrung sind Chains auch verwaltbar
- Chains sind seit etwa 11 Monaten im Einsatz und steuern jede Nacht mehrere hundert Verarbeitungsschritte

Fragen ?

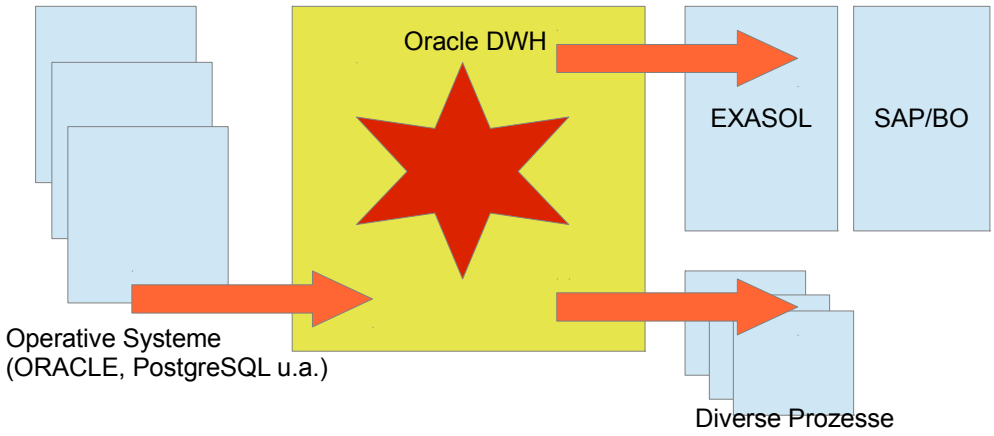
Präsentiert von

Norbert Klamann
Klamann Software Ltd.
http://klamann-software.de
Norbert.Klamann@klamann-software.de
0172/2797723

Der Einsatz von Oracle Job-Chains im ETL-Prozeß

Erfahrungsbericht eines Projekteinsatzes, daher keine umfassende Darstellung des Themas, aber praktisch funktionierende Lösungen. Der Vortrag wendet sich an Techniker.

Gesamtsituation beim Kunden



Nächtlich werden Daten aus den operativen Systemen in das DWH geladen
Dort werden sie in vielfältiger Weise verarbeitet.

Das ist unser Thema heute

Danach werden die Daten in eine EXASOL-DB kopiert, auf der die Abfragen aus dem SAP/BO-System aufsetzen.

Außerdem werden diverse andere Prozesse mit Daten beliefert

Ausgangslage

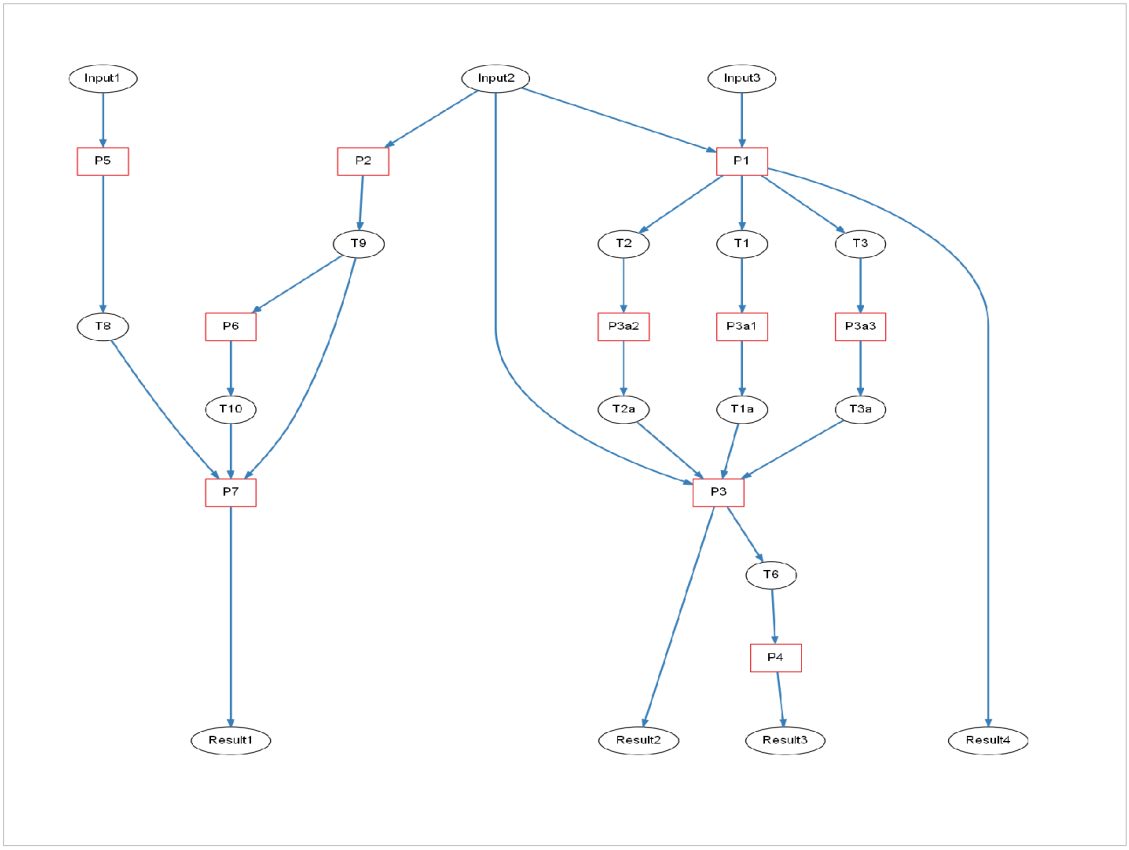
- Zeitfenster 01:00 bis 08:00
- Datenmenge wird stark steigen
- Zeitfenster wird schon mit der jetzigen Datenmenge nur eingehalten, wenn alles glatt geht
- Ca. 30 Verarbeitungsschritte wurden sequentiell abgearbeitet

Die sequentielle Verarbeitung nutzt die Maschinenkapazität nicht aus.

Restart ist schwierig:
•Fehlerstelle finden
•An den richtigen Stellen auskommentieren
•Wieder starten
•Änderungen wieder zurückdrehen

Durch Job-Chains haben wir erreicht:

- Verringerung der Durchlaufzeit im ersten Anlauf um 2 h
- Fehler sind leichter lokalisierbar
- Nach der Fehlerkorrektur kann leicht wieder gestartet werden
- Von Fehlern nicht betroffene Schritte laufen weiter



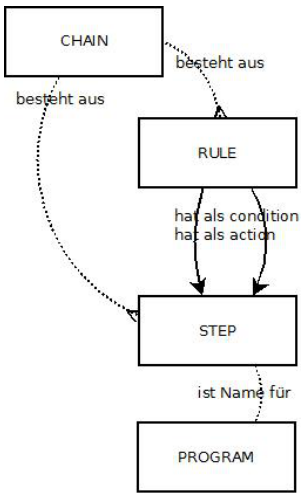
Dieses stark vereinfachte Beispiel zeigt die Ablaufstruktur in einem ETL-Prozeß

In das System fließen 3 Input-Datenbestände ein, es werden 3 Result-datenbestände erzeugt und dabei werden einige Tabellen Tx geschrieben und gelesen.

Problem sind die komplexen Abhängigkeiten.

Auf der anderen Seite kann vieles parallelisiert werden.

Job-Chains mit DBMS_SCHEDULER



Seit Version 10 in allen Versionen der Datenbank enthalten.
Über Systemviews und PL/SQL gut integrierbar

Die logischen Elemente oben werden mit Prozeduren des DBMS_SCHEDULER-Paketes verwaltet.
Die CHAIN ist die logische Struktur der Abhängigkeiten zwischen einzelnen STEPS.
Die RULE bildet die Logik der Chain ab, sie bsteht aus Vorbedingung und Aktion.
Ein STEP ist nur ein Name für ein PROGRAM (oder eine geschachtelte CHAIN)

RULE: CONDITION/ACTION

Vorbedingungen:

- TRUE ist immer wahr
- XXX SUCCEEDED
- XXX COMPLETED
- XXX FAILED

Verknüpfungen mit AND und OR

XXX steht für einen STEP-Namen

Mindestens eine RULE muß TRUE als Vorbedingung haben, sonst startet kein Verarbeitungsschritt

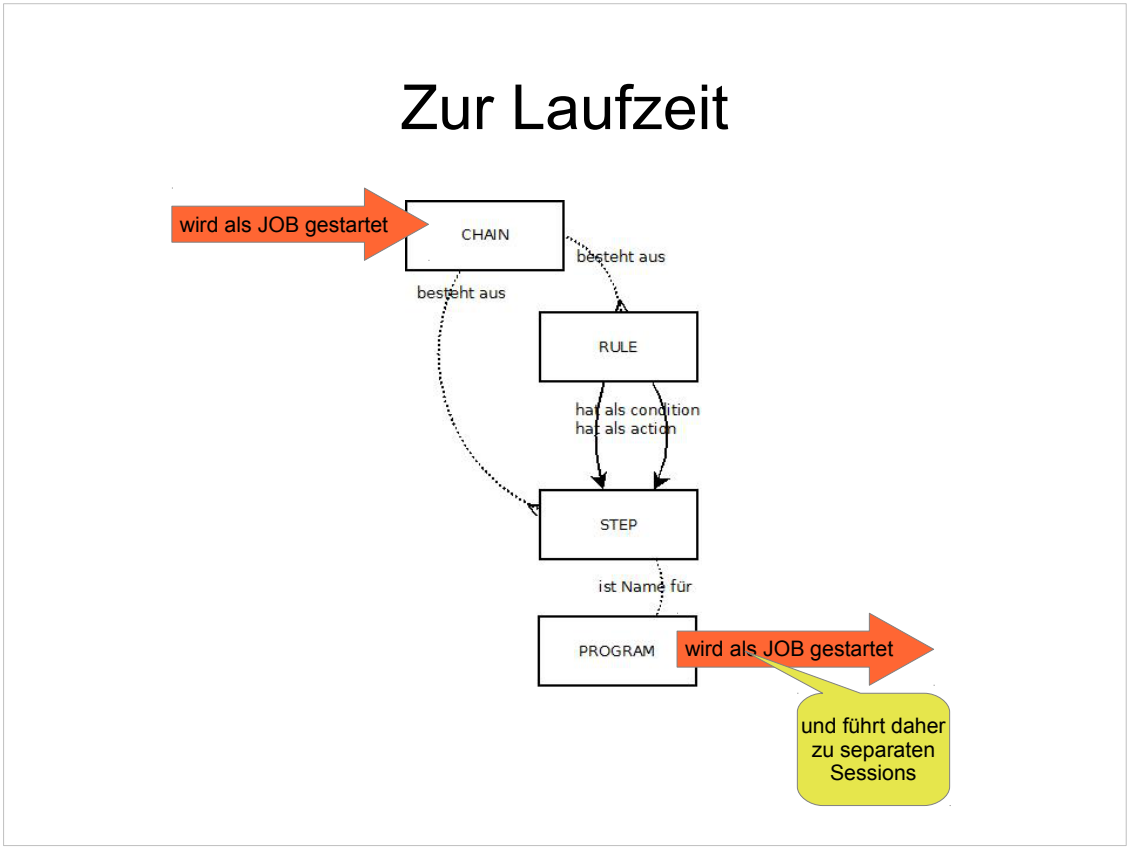
RULE: CONDITION/ACTION

Aktionen:

- END beendet die CHAIN
- START XXX

Es können auch mehrere Schritte gestartet werden

Genau ein Schritt sollte END als Aktion haben, sonst endet die CHAIN nicht.



Das Chain-System profitiert damit von der ganzen Infrastruktur des DBMS_SCHEDULER-Systems.

Für die praktische Umsetzung ist es wichtig, daß die einzelnen PROGRAMS als separete Sessions gestartet werden.

Die wichtigsten Subprozeduren von DBMS_SCHEDULER (1/4)

```
DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name              IN VARCHAR2,
   rule_set_name           IN VARCHAR2 DEFAULT NULL,
   evaluation_interval     IN INTERVAL DAY TO SECOND DEFAULT NULL,
   comments                IN VARCHAR2 DEFAULT NULL);
```

Die wichtigsten Subprozeduren von DBMS_SCHEDULER (2/4)

```
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name             IN VARCHAR2,
   program_type             IN VARCHAR2,
   program_action           IN VARCHAR2,
   number_of_arguments      IN PLS_INTEGER DEFAULT 0,
   enabled                  IN BOOLEAN DEFAULT FALSE,
   comments                 IN VARCHAR2 DEFAULT NULL);
```

Die wichtigsten Subprozeduren von DBMS_SCHEDULER (3/4)

```
DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
   chain_name              IN VARCHAR2,
   step_name               IN VARCHAR2,
   program_name            IN VARCHAR2);
```

Die wichtigsten Subprozeduren von DBMS_SCHEDULER (4/4)

```
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   chain_name              IN VARCHAR2,
   condition               IN VARCHAR2,
   action                  IN VARCHAR2,
   rule_name               IN VARCHAR2 DEFAULT NULL,
   comments                IN VARCHAR2 DEFAULT NULL);
```

Systemviews der statischen Struktur

- DBA_SCHEDULER_CHAIN_RULES
- DBA_SCHEDULER_CHAIN_STEPS
- DBA_SCHEDULER_JOBS
- DBA_SCHEDULER_PROGRAMS

Systemviews zur Laufzeitkontrolle

- DBA_SCHEDULER_RUNNING_CHAINS
- DBA_SCHEDULER_RUNNING_JOBS
- DBA_SCHEDULER_JOB_LOG
- DBA_SCHEDULER_JOB_RUN_DETAILS

(in dieser Tabelle ist die Log-Id der gesamten Chain in der Spalte ADDITIONAL_INFO zu finden.)

Laufzeitkontrolle

```
select
JOB_NAME, CHAIN_NAME, STEP_NAME, STATE, ERROR_CODE,START_DATE,END_DATE,
DURATION
from DBA_SCHEDULER_RUNNING_CHAINS
order by OWNER, JOB_NAME, START_DATE
```

JOB_NAME	CHAIN_NAME	STEP_NAME	STATE	ERROR_CODE	START_DATE	END_DATE	DURATION
RUN_DB2	C_DB2	S_0000_START	SUCCEEDED	0	31.03.12 14:25:04,539470 +02:00	31.03.12 14:25:10,601074 +02:00	+000000000
RUN_DB2	C_DB2	S_0102_SERVICE	SUCCEEDED	0	31.03.12 14:25:10,722009 +02:00	31.03.12 16:08:15,205578 +02:00	+000000000
RUN_DB2	C_DB2	S_0103_RETOURE	SUCCEEDED	0	31.03.12 14:25:10,743131 +02:00	31.03.12 16:22:12,860838 +02:00	+000000000
RUN_DB2	C_DB2	S_0104_LOGISTIK	FAILED	4063	31.03.12 14:25:10,827835 +02:00	31.03.12 14:25:33,762727 +02:00	+000000000
RUN_DB2	C_DB2	S_0401_PAYMENT_START	SUCCEEDED	0	31.03.12 16:32:21,919985 +02:00	31.03.12 16:32:21,920017 +02:00	+000000000
RUN_DB2	C_DB2	S_0411_PAYM_FILL_BI	SUCCEEDED	0	31.03.12 16:32:23,430245 +02:00	31.03.12 17:01:34,108099 +02:00	+000000000
RUN_DB2	C_DB2	S_0412_PAYM_FILL_ORDER	SUCCEEDED	0	31.03.12 16:32:23,529822 +02:00	31.03.12 16:56:04,032304 +02:00	+000000000
RUN_DB2	C_DB2	S_0413_PAYM_FILL_NOTI	SUCCEEDED	0	31.03.12 16:32:23,530043 +02:00	31.03.12 17:12:16,675682 +02:00	+000000000
RUN_DB2	C_DB2	S_0414_PAYM_FILL_REMD	SUCCEEDED	0	31.03.12 16:32:23,597860 +02:00	31.03.12 16:35:48,366374 +02:00	+000000000
RUN_DB2	C_DB2	S_0420_PAYMENT_PHASE2	SUCCEEDED	0	31.03.12 17:12:16,690766 +02:00	31.03.12 17:44:19,272098 +02:00	+000000000
RUN_DB2	C_DB2	S_0302_MERGE_SERVICE	NOT_STARTED				
RUN_DB2	C_DB2	S_0202_AGG_LOGI_UNIV	NOT_STARTED				
RUN_DB2	C_DB2	S_0201_AGG_ORDER	NOT_STARTED				

Auswirkungen eines gescheiterten STEPs in einer CHAIN

Nicht betroffene STEPs sind weiter gelaufen.

Weiterlauf nach Korrektur

```
begin
dbms_scheduler.alter_running_chain  (
job_name=>'RUN_DB2',
step_name=>'S_0104_LOGISTIK',
attribute=>'STATE',
value=>'SUCCEEDED'
);
end;
```

Zur Korrektur wird das PROGRAM des Schrittes S_0104_LOGISTIK separat nachgefahren und danach wird der Schritt mit folgendem Statement auf SUCCEEDED gesetzt:

Danach wird die Kette weiterlaufen und die letzten 3 Schritte abarbeiten.

Man sollte in solchen Fällen die Kette selber unbedingt weiterlaufen lassen. Oracle hält selber nach, welche Schritte getan werden können.

Anforderungen an Programme (1/2)

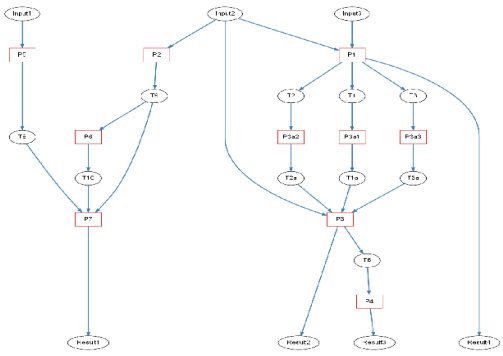
- Programme dürfen keine OUT-Parameter benutzen
- IN-Parameter sind umständlich
- Am günstigsten sind Steuertabellen zum Datenaustausch, eventuell in Wrappern realisiert

Anforderungen an Programme (2/2)

- Programme dürfen keine EXCEPTIONs verschlucken, Fehler müssen eskaliert werden
- RAISE oder RAISE_APPLICATION_ERROR einsetzen

Vorgehensweise: Abhängigkeiten dokumentieren

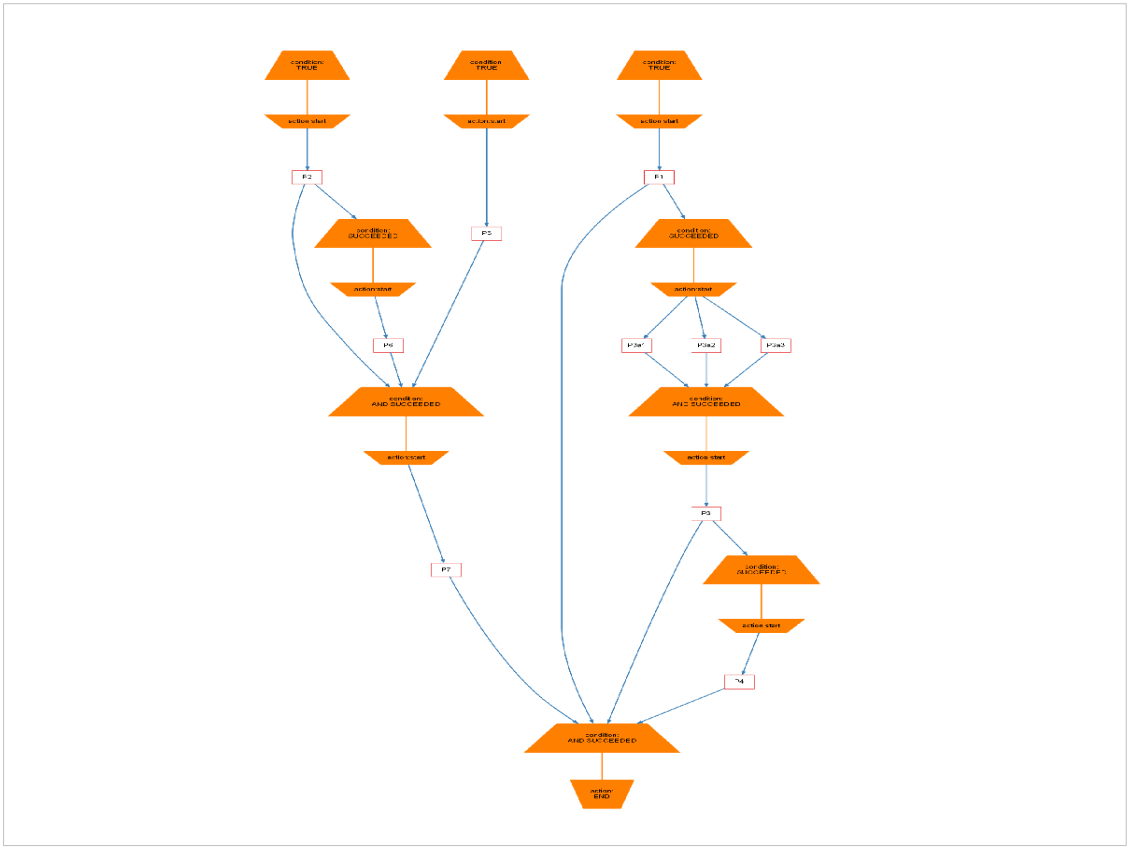
- Es muß erkennbar werden, welches Programm welche Tabelle liest und schreibt



Eine graphische Darstellung hilft, aber auch eine Tabelle ist nützlich

Vorgehensweise: Vom Programm aus rückwärts sehen:

- Welche Tabellen werden gelesen und bearbeitet ?
- Welche Programme bringen die Tabellen in den Zustand, der gebraucht wird ?
- Alle diese Programme müssen erfolgreich abgeschlossen sein , bevor das betrachtete Programm starten kann.



Durch die gestellten Fragen kann man ziemlich schematisch eine Graphik in eine Ansammlung von RULEs überführen

Die RULEs sind hier als die Orange-farbenen Formen dargestellt

Weitere Regeln für CHAINs

- Eine RULE muß die Vorbedingung TRUE haben
- Eine RULE muß die Aktion END haben
- Keine STEPs erlauben, die keinen Nachfolger haben (lose Enden)
- Mehrfacher Start einer CHAIN sollte abgefangen werden

Lose Enden führen nicht zu einem definierten Ende der Kette. Es ist nichts mehr zu tun, aber die RULE mit der Aktion END wird nicht erreicht.

Wie werden Chains praktikabel ?

CHAINS können deutlich über 100 STEPs enthalten, da müssen zahlreiche Informationen zusammen passen, daraus ergibt sich :

- Man muß CHAINs bearbeiten können wie Sourcecode und sie reproduzierbar wieder herstellen können.
- Man muß die Richtigkeit der CHAIN-Logik kontrollieren können, ohne sie laufen zu lassen

Die Prozeduren des DBMS_SCHEDULER-Packages bieten kein 'Create or replace', daher sollten sie gekapselt werden
=> Hilfspackage bauen

Zur Logik-Kontrolle sollte man Graphiken erzeugen können.

Der Enterprise Manager kann das theoretisch, er hat überhaupt rudimentären Support für Chains. Aber die Graphik läuft nur unter bestimmten Kombinationen von Browser und Plugin.

Chain-'Sourcecode': CHAIN + STEPs

Nicht gezeigt: die Definitionen der PROGRAMs

```
-- Anlegen aller Steps und Rules  der Chain  C_DB2, generiert am 29.09.11
-- Diesen Text als "C_DB2.sql" im Unterverzeichnis "scripts" speichern und mit svn versionieren
-- $HeadURL: xxxx $
-- $Id: c_db2.sql 1086 2011-10-19 10:00:19Z norbert.klamann $
DECLARE
L_CHAIN_NAME varchar2(30) := 'C_DB2';
  L_COMMENT       varchar2(200) := 'Berechnung der Deckungsbeiträge II';-- Anzahlen (vor diesen Änderungen) : steps 14 rules 15
BEGIN
PKG_UTIL_SCHEDULING.CREATE_OR_REPLACE_CHAIN(L_CHAIN_NAME, L_COMMENT, P_FORCE => TRUE);
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0000_START','P_AGG_PREPARE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0102_SERVICE','P_SERVICE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0103_RETOURE','P_RETOURE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0104_LOGISTIK','P_LOGISTIK');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0201_AGG_ORDER','P_AGG_ORDER');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0202_AGG_LOGI_UNIV','P_AGG_LOGI_UNIV');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0302_MERGE_SERVICE','P_AGG_SERVICE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0401_PAYMENT_START','P_PAYMENT_START');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0411_PAYM_FILL_BI','P_PAYMENT_FILL_BI_PCTRL_LOG');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0412_PAYM_FILL_ORDER','P_PAYMENT_FILL_ORDER');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0413_PAYM_FILL_NOTI','P_PAYMENT_FILL_PAYMENT_NOTI');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0414_PAYM_FILL_REMD','P_PAYMENT_FILL_REMINDER');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0420_PAYMENT_PHASE2','P_PAYMENT_PHASE2');
--
```

Dies ist ein SQL-Script, daß so ablaufen kann und eine komplette CHAIN definiert und mit einem startfähigen Job zusammen bereit stellt.

Chain-'Sourcecode': RULEs und JOB

STEPS müssen existieren

```
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'TRUE','START "S_0000_START"', comments=> 'V bereitung');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START "S_0401_PAYMENT_START"', comments=> 'kann
ohne Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START "S_0102_SERVICE"', comments=> 'kann ohne
Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START "S_0103_RETOURE"', comments=> 'kann ohne
Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START "S_0104_LOGISTIK"', comments=> 'kann ohne
Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START "S_0411_PAYM_FILL_BI"', comments=>
'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START "S_0412_PAYM_FILL_ORDER"',
comments=> 'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START "S_0413_PAYM_FILL_NOTI"', comments=>
'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START "S_0414_PAYM_FILL_REMD"', comments=>
'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0411_PAYM_FILL_BI SUCCEEDED AND S_0412_PAYM_FILL_ORDER SUCCEEDED AND
S_0413_PAYM_FILL_NOTI SUCCEEDED AND S_0414_PAYM_FILL_REMD SUCCEEDED ','START "S_0420_PAYMENT_PHASE2"', comments=> 'Nach den
Berechnungen aggregieren');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0420_PAYMENT_PHASE2 SUCCEEDED AND S_0102_SERVICE SUCCEEDED AND
S_0103_RETOURE SUCCEEDED AND S_0104_LOGISTIK SUCCEEDED ','START "S_0201_AGG_ORDER"', comments=> 'Nach den Berechnungen
aggregieren');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0104_LOGISTIK SUCCEEDED','START "S_0202_AGG_LOGI_UNIV"', comments=>
'Logistik-Universum kann parallel laufen');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0201_AGG_ORDER SUCCEEDED','START "S_0302_MERGE_SERVICE"', comments=>
'weitere parallele Merges');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0302_MERGE_SERVICE COMPLETED AND S_0202_AGG_LOGI_UNIV COMPLETED ','END ',
comments=> 'Chain fertig.');
DBMS_SCHEDULER.ENABLE(name => L_CHAIN_NAME);
 PKG_UTIL_SCHEDULING.CREATE_OR_REPLACE_SC_JOB(P_JOB_NAME =>'RUN_DB2',P_CHAIN_NAME=> l_chain_name, P_SCHED_NAME => '',
P_COMMENTS=> 'DB2-Prozess',P_FORCE => TRUE);
END;
```

Ein solches Skript selber wiederum wird aus den Systemviews generiert

Logik der Chain visualisieren

Kann man gut mit Graphviz (http://www.graphviz.org)

- Sourcecode für Graphviz mit SQL generieren
- Sourcecode in gvedit laden
- Graphik generieren
- Mit MSPAINT kann man große Graphiken auf mehrere Blatt Papier aufteilen

Graphviz ist Open Source Dieses programm generiert aus Strukturzusammenhängen Graphen mit einem zeimlich guten Layout. Vor allem aber respektiert es die Struktur:

Graphviz-Graphiken sind wahr, im Gegensatz zu manuell gezeichneten

Graphviz-Sourcecode (1/2)

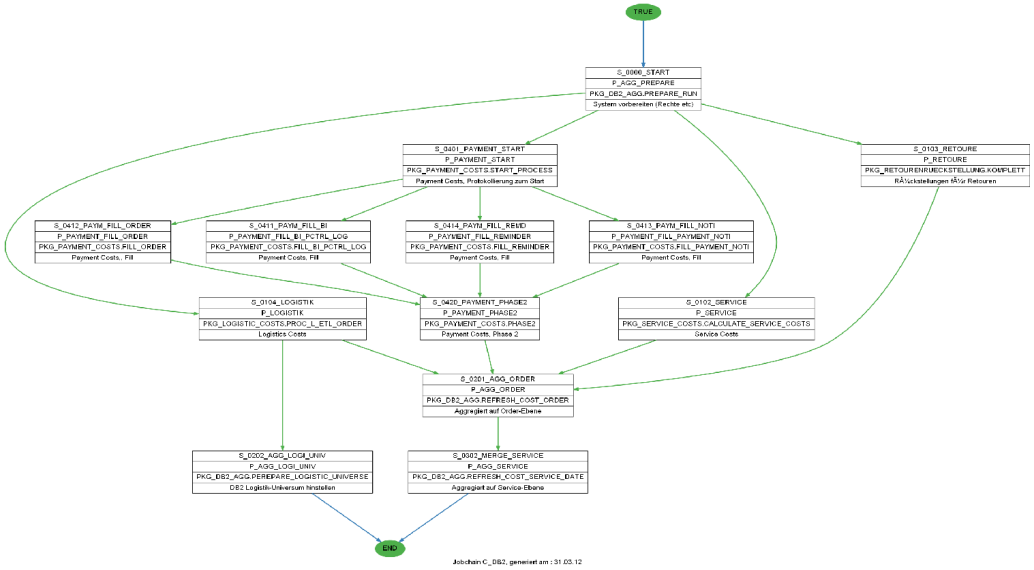
```
/* Graphviz/dot- Dokumentation der Chain xxx.C_DB2, generiert am 31.03.12
 speichern als Datei C_DB2.gv, dann mit Graphviz starten
*/
digraph  C_DB2  {
    ratio=auto; charset=latin1; center=true; fontname=Helvetica; pad=0.5;
    concentrate=true;ranksep="1.0 equally"; nodesep=1.0;colorscheme="set19";
    node [fontname=Helvetica, colorscheme="set19", color="black"];
    edge [fontname=Helvetica, colorscheme="set19", penwidth=2, color=2];
  label="Jobchain C_DB2, generiert am : 31.03.12";
TRUE [style=filled,color=3];
END  [style=filled,color=3];
S_0302_MERGE_SERVICE [shape="record",label="{S_0302_MERGE_SERVICE|P_AGG_SERVICE|PKG_DB2_AGG.REFRESH_COST_SERVICE_DATE|Aggregiert\ auf\
Service-Ebene}"] ;
S_0412_PAYM_FILL_ORDER [shape="record",label="{S_0412_PAYM_FILL_ORDER|P_PAYMENT_FILL_ORDER|PKG_PAYMENT_COSTS.FILL_ORDER|Payment\ Costs,,\
Fill}"] ;
S_0420_PAYMENT_PHASE2 [shape="record",label="{S_0420_PAYMENT_PHASE2|P_PAYMENT_PHASE2|PKG_PAYMENT_COSTS.PHASE2|Payment\ Costs,\ Phase\ 2}"] ;
S_0201_AGG_ORDER [shape="record",label="{S_0201_AGG_ORDER|P_AGG_ORDER|PKG_DB2_AGG.REFRESH_COST_ORDER|Aggregiert\ auf\ Order-Ebene}"] ;
S_0202_AGG_LOGI_UNIV [shape="record",label="{S_0202_AGG_LOGI_UNIV|P_AGG_LOGI_UNIV|PKG_DB2_AGG.PEREPARE_LOGISTIC_UNIVERSE|DB2\ Logistik-
Universum\ hinstellen}"] ;
S_0104_LOGISTIK [shape="record",label="{S_0104_LOGISTIK|P_LOGISTIK|PKG_LOGISTIC_COSTS.PROC_L_ETL_ORDER|Logistics\ Costs}"] ;
S_0401_PAYMENT_START [shape="record",label="{S_0401_PAYMENT_START|P_PAYMENT_START|PKG_PAYMENT_COSTS.START_PROCESS|Payment\ Costs,\
Protokollierung\ zum\ Start}"] ;
S_0411_PAYM_FILL_BI [shape="record",label="{S_0411_PAYM_FILL_BI|P_PAYMENT_FILL_BI_PCTRL_LOG|PKG_PAYMENT_COSTS.FILL_BI_PCTRL_LOG|Payment\
Costs,\ Fill}"] ;
S_0414_PAYM_FILL_REMD [shape="record",label="{S_0414_PAYM_FILL_REMD|P_PAYMENT_FILL_REMINDER|PKG_PAYMENT_COSTS.FILL_REMINDER|Payment\ Costs,\
Fill}"] ;
S_0102_SERVICE [shape="record",label="{S_0102_SERVICE|P_SERVICE|PKG_SERVICE_COSTS.CALCULATE_SERVICE_COSTS|Service\ Costs}"] ;
S_0413_PAYM_FILL_NOTI [shape="record",label="{S_0413_PAYM_FILL_NOTI|P_PAYMENT_FILL_PAYMENT_NOTI|PKG_PAYMENT_COSTS.FILL_PAYMENT_NOTI|Payment\
Costs,\ Fill}"] ;
S_0103_RETOURE [shape="record",label="{S_0103_RETOURE|P_RETOURE|PKG_RETOURENRUECKSTELLUNG.KOMPLETT|Rückstellungen\ für\ Retouren}"] ;
S_0000_START [shape="record",label="{S_0000_START|P_AGG_PREPARE|PKG_DB2_AGG.PREPARE_RUN|System\ vorbereiten\ (Rechte\ etc)}"] ;
TRUE -> S_0000_START;
```

Graphviz-Sourcecode (2/2)

```
TRUE -> S_0000_START;
S_0411_PAYM_FILL_BI -> S_0420_PAYMENT_PHASE2[color=3];
S_0412_PAYM_FILL_ORDER -> S_0420_PAYMENT_PHASE2[color=3];
S_0413_PAYM_FILL_NOTI -> S_0420_PAYMENT_PHASE2[color=3];
S_0414_PAYM_FILL_REMD -> S_0420_PAYMENT_PHASE2[color=3];
S_0420_PAYMENT_PHASE2 -> S_0201_AGG_ORDER[color=3];
S_0102_SERVICE -> S_0201_AGG_ORDER[color=3];
S_0103_RETOURE -> S_0201_AGG_ORDER[color=3];
S_0104_LOGISTIK -> S_0201_AGG_ORDER[color=3];
S_0104_LOGISTIK -> S_0202_AGG_LOGI_UNIV[color=3];
S_0201_AGG_ORDER -> S_0302_MERGE_SERVICE[color=3];
S_0302_MERGE_SERVICE -> END;
S_0202_AGG_LOGI_UNIV -> END;
S_0000_START -> S_0401_PAYMENT_START[color=3];
S_0000_START -> S_0102_SERVICE[color=3];
S_0000_START -> S_0103_RETOURE[color=3];
S_0000_START -> S_0104_LOGISTIK[color=3];
S_0401_PAYMENT_START -> S_0411_PAYM_FILL_BI[color=3];
S_0401_PAYMENT_START -> S_0412_PAYM_FILL_ORDER[color=3];
S_0401_PAYMENT_START -> S_0413_PAYM_FILL_NOTI[color=3];
S_0401_PAYMENT_START -> S_0414_PAYM_FILL_REMD[color=3];
}
```

Der Code wurde durch eine Routine unseres Hilfspackages aus der Datenbank generiert, seine Inhalte basieren auf den Systemtabellen und sind daher wahr. In der Routine wurden sie nur um Stücke der Graphviz-Syntax ergänzt.

Graphviz-Ausgabe



In der Chain C_DB2 wurde - wie man hier sieht - vor dem END kein Dummy-Schritt eingeführt, aber eine Sichtkontrolle des Sourcecodes oben zeigt, daß in der Tat nur eine RULE die Action END hat.

Namenskonventionen

"There are only two hard things in Computer Science: cache invalidation, naming things and off by 1 errors"

- PROGRAMS können nicht genauso heißen wie PACKAGE.PROZEDUR-Aufrufe
- STEP-Namen können nur 27 Zeichen lang sein
- Es werden **viele** Namen gebraucht

  Empfehlung : keine strukturierten Namen, fachlich benennen

Namenskonventionen , welche die Struktur ausdrücken sollen, scheitern an der Vielzahl der Objekte und am Verschieben

Für Präfixe ist kein Platz

Bewertung: Schwächen

- Falsche STEP-Namen in der RULE-Definition werden nicht zur 'Kompilezeit' abgefangen sondern führen zu einer Flut an Jobs zur Laufzeit.
- Es müssen sehr viele DB-Objekte eingerichtet werden
- Man sollte PROGRAMs nicht verwenden müssen, wenn man sie nicht fachlich braucht

Bewertung: Chains sind nützlich

Job-Chains waren eine gute Investition:

- Durch Parallelität haben wir Zeit gewonnen
- Durch Restartfähigkeit haben wir Zuverlässigkeit und damit Standing nach außen gewonnen
- Mit einigen kleineren Tools und etwas Erfahrung sind Chains auch verwaltbar
- Chains sind seit etwa 11 Monaten im Einsatz und steuern jede Nacht mehrere hundert Verarbeitungsschritte

Fragen ?

Präsentiert von

Norbert Klamann
Klamann Software Ltd.
http://klamann-software.de
Norbert.Klamann@klamann-software.de
0172/2797723