Distributed Data Management

Labor 08

Marc-Antoine Brülhart

Simon Wächter

Carole Hug

Carlo Baumann

Inhaltsverzeichnis

[**Ausgangslage**](#_bgj07lhj750w) **3**

[**Lösungsstrategie**](#_gzvq63p2fi4j) **4**

[Änderungen auf einen lokalen Node](#_1ruc1om8ic5x) 4

[Synchronisieren der Nodes](#_ite0abcqobxx) 5

[**SQL Befehle für das Einrichten der Multi-Master Replication**](#_8m2z6v1hvst0) **6**

[**DB-Links erstellen**](#_bi14xgyp1wyi) **7**

[Für Orion](#_wltbk7r46viz) 7

[Für Titania](#_83siz7piwpf) 7

[Für Ganymed:](#_9xw5rxdn1ntd) 7

[**Synonymes erstellen für einfachere Namen**](#_nncy1goicf1p) **7**

[Für Orion](#_ejvzin7vfugs) 8

[Für Titania](#_4mzrbshlp687) 8

[Für Ganymed](#_sa35lzkiq33h) 9

[Tabellen auf allen Nodes erstellen](#_9v6c7jv01vvt) 9

[**Triggers zum Loggen und Löschen der Logs erstellen**](#_eqgl7hgaox1e) **11**

[**Prozedur zum Synchronisieren erstellen**](#_hpwxlcpf3vn8) **12**

[Execute\_on\_myself](#_62xvngwvjo28) 12

[View Erstellen um den Code lesbarer zu schreiben](#_ir4qrbyzvxxk) 13

[Für Helios und Ganymed](#_jz3v5btyeanr) 13

[Für Titania](#_mc2fffb07n51) 13

[Update\_Logs\_on\_collegues](#_sibng57j4wzm) 14

[Für Orion:](#_jvuo6aato49o) 14

[Für Ganymed:](#_854rtx9jwp2a) 14

[Für Titania:](#_biycfhkutpch) 14

[Trigger\_delete\_logs\_on\_collegues](#_7yig2n9kcaww) 15

[Sync\_node](#_z3r2wtxtubgd) 15

[**Sync aller Nodes**](#_xcd4116119py) **15**

[**Testkonzept und Testresultate**](#_oq0deuatc3kb) **16**

## 

# Ausgangslage

**Grundlage**: Situation 1 aus dem Lab02-DreamHome.

Um die lokale Autonomie der Filialen zu erhöhen, möchte die Immobilienfirma DreamHome

die Tabellen auf ihren Knoten Ganymed, Orion und Titania replizieren. Als erster Schritt

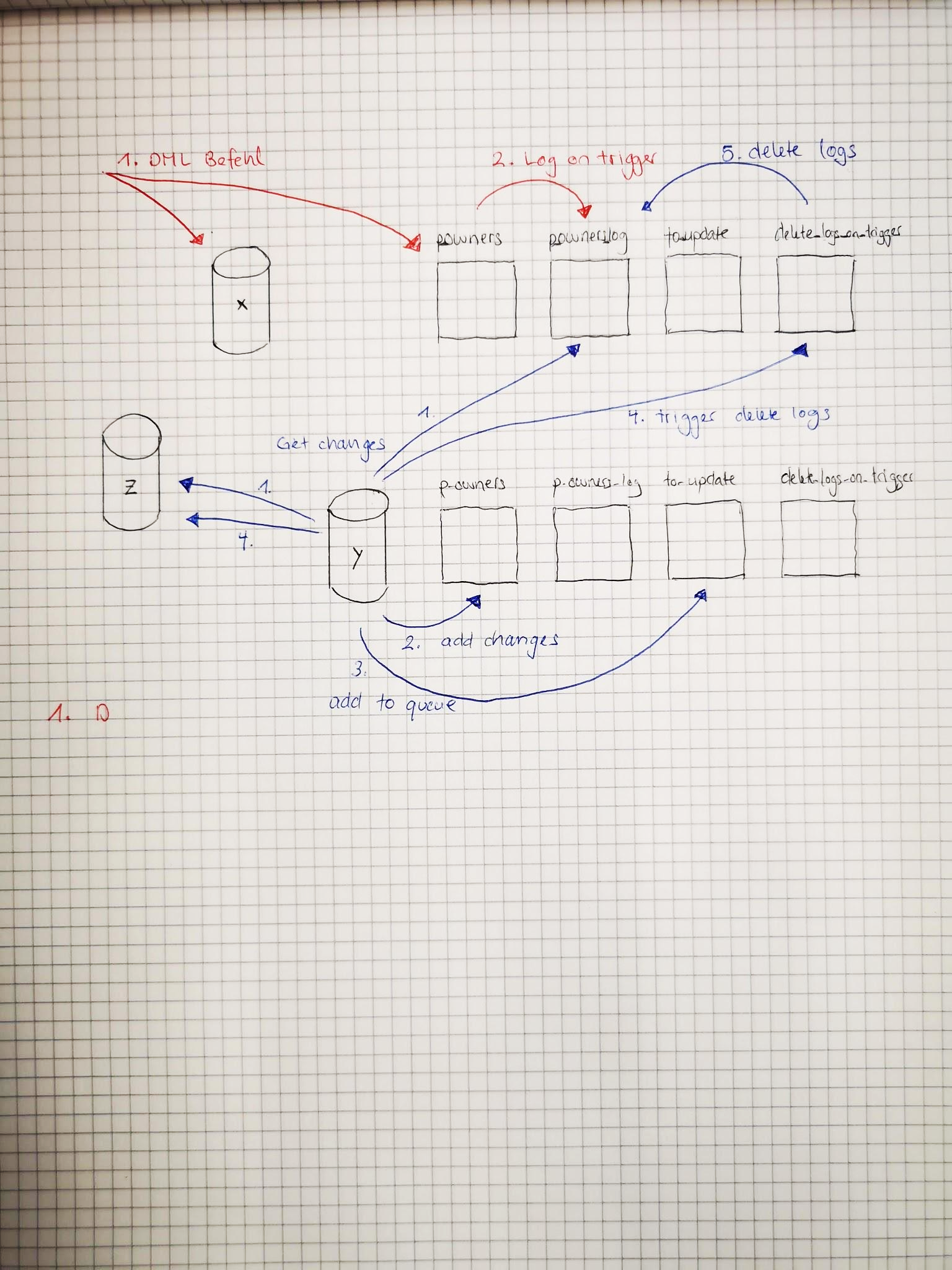
soll die Tabelle Privateowners, original auf Titania, auch auf den anderen Knoten repliziert

werden. Änderungen (Insert, Update, Delete) müssen auf allen Knoten möglich sein.

**Auftrag**: Entwerfen und realisieren Sie für diesen ersten Schritt eine Multi-Master Replication der Tabelle Privateowners

# Lösungsstrategie

Für das Synchronisieren der Nodes, verwenden wir 4 Tabellen, 2 Triggers, sowie mehrere Synonyme, Views und Prozeduren für die Les- und Wartbarkeit. Nachfolgend wird an einer Beispielskizze unsere Lösungsidee erläutert.



Im Bild sind mit rotem Stift die Schritte für das Einfügen von Änderungen auf einen lokalen Node hinterlegt. MIt blauem Stift, sind die Schritte und deren Abfolge für das Synchronisieren der Nodes niedergeschrieben.

## Änderungen auf einen lokalen Node

1. Schritt: Mit einem DML Befehl eine Änderung auf einen Node einfügen
2. Schritt: Die Änderung auf dem Node wird in eine Log Tabelle geschrieben

Der Trigger **“p\_owners\_log\_trigger”** sorgt dafür, dass Änderungen die auf einen Node in die **“p\_owners”** Tabelle per DML Befehl geschrieben, in die Log Tabelle**”p\_owners\_log”** übertragen werden. Damit die Logs später nacheinander der Reihenfolge repliziert werden können, wird jeweils ein “**timestamp”** und eine **“acknowledgment”** Nummer mitgeschrieben. Der “**timestamp”** ist für das zeitliche Sortieren der Befehle verantwortlich. Die **“acknowledgment”** Nummer ist für das Kennzeichnen von bereits replizierten Änderungen verantwortlich. Das Feld kann mit 0, 1, 2 oder 3 belegt sein. Die Bedeutungen der Nummern sind nachfolgend beschrieben:

* 0 die default Nummer - keine der Nachbar Nodes hat die Änderung bei sich repliziert
* 1 - einer der beiden nachbarn Nodes hat die Änderung bei sich lokal repliziert
* 2 - der andere der beiden Nachbarn Nodes hat die Änderung bei sich lokal repliziert
* 3 - beide Nachbarn Nodes haben die Änderungen bei sich lokal repliziert, bereit zum Löschen der Datensatz.

Bei Ausführen von Änderungen auf einem lokalen Node wie in Schritt 1, wird immer die 0 hinterlegt. Die Nummern 1 bis 3 werden von den Nachbarn Nodes beim Replizieren von Änderungen gespeichert.

## Synchronisieren der Nodes

1. Schritt: Laden der DML Änderungen von den Nachbar Nodes.
2. Schritt: Einfügen der DML Änderungen sortiert auf meinem Node.
3. Schritt: Hinzufügen aller Änderungen lokal übernommen wurden in eine Queue
4. Schritt: Auf den Nachbarn Nodes die Triggers zum Löschen der Logs triggern
5. Schritt: Löschen der synchronisierten Änderungen aus den Log Tabellen der Nachbarn

Zum Synchronisieren eines Nodes muss die **“sync\_node”** Prozedur ausgeführt werden. Wenn alle Nodes synchronisiert werden sollen müssen die **“sync\_node”** Prozeduren auf allen Nodes ausgeführt werden. Das genaue Synchronisieren wird nachfolgend detailiert erklärt.

Eine View **“collegue\_changes”** sorgt dafür, dass alle Änderungen der Nachbarn Nodes selektiert und nach dem Timestamp sortiert (nach chronologischer Reihenfolge) werden. Dazu wird jeweils die hinterlegte **“acknowledgment”** Nummer in der Tabelle **“p\_owners\_log”** geprüft. Danach wird mit dem Prozeduraufruf **“execute\_on\_myself”** alle Änderungen der Nachbarn Nodes lokal repliziert, nach chronologischer Reihenfolg. Zugleich werden alle Änderungen die lokal repliziert wurden, in eine Tabelle **“to\_update”** hinterlegt. Diese dient als Queue und sorgt dafür, dass mit einer Übertragung an die Nachbarn Nodes mitgeteilt werden kann welche Änderungen repliziert wurden. Wenn alle Änderungen lokal repliziert wurden, werden die Änderungen in der **“to\_update”** Tabelle mit dem Prozeduraufruf **“update\_logs\_on\_collegues”** an die Nachbarn weitergegeben und die **“acknowledgment”** Nummer entsprechend erhöht. Die **“to\_update”** Tabelle wird nach dem propagieren der Änderungen mit einem TRUNCATE Befehl geleert.

Damit die Logs in den **“p\_owners\_log”** Tabellen nicht überquellen, wird mittels Prozeduraufruf **“delete\_logs\_on\_trigger”,** auf den Nachbarn Nodes, die Datensätzte mit der **“acknowledgment”** Nummer 3 gelöscht. Diese zusätzliche Tabelle wird benötigt, weil keine Modifikationen innerhalb eines Triggers auf die Tabelle vorgenommen werden können, auf die der Trigger registriert ist. Damit die **“p\_owners\_log”** Datensätzte mit der **“acknoweledgment”** Nummer 3 auf den Nachbarn Nodes tatsächlich gelöscht werden, gibt es den Trigger **“delete\_logs\_on\_trigger\_trigger”.** Dieser sorgt mit einem DML Befehl dafür, dass alle Datensätzte mit der **“acknoweledgment”** Nummer 3 in der Tabelle **“p\_owners\_log”** gelöscht werden.

Während des Repliziervorgangs, werden die Triggers auf der **“p\_owners”** Tabelle deaktiviert. Damit wird verhindert, dass die replizierten Daten in die **“p\_owners\_log”** Tabelle gelangen. Es wird ein Infinity Loop verhindert.

Damit Orion und Ganymed wissen, welche Änderungen sie von Titania bereits repliziert haben und die **“p\_owners\_log”** Tabelle nicht überquellt, gibt es das Feld **“acknowledgment”.**

Für das bessere Verständnis der Replizierung mithilfe des Felds **“acknowledgment”** , wird nachfolgend anhand einer Skizze dies genauer erläutert:



Auf dem Bild sind die drei Nodes Orion, Ganymed und Titania und die **“p\_owners\_log”** Tabelle von letzterem Node zu sehen. Die bedeutungen der einzelnen Nummern wurden im vorherigen Kapitel bereits erwähnt. Damit die Nodes nicht mehrmals die gleichen Daten replizieren, werden sie anhand fest zugeteilter Nummern (1 und 2) von den Nachbarn eindeutig identifiziert. Als Beispiel erhält Titania beim Node Ganymed die Nummer 2 und beim Node Orion die Nummer 1. Umgekehrt sind die Nodes Ganymed mit der Nummer 1 und Orion mit der Nummer 2 bei Titania registriert. Sobald die Nachbarn Nodes Ganymed und Orion Änderungen von Titania replizieren, erhöhen sie in der Tabelle **“p\_owners\_log”**  das Feld **“acknowledgment”** mit ihrer Nummer und bestätigen die Replizierung des Datensatz. Nach der Replizierung wird der Trigger **“delete\_logs\_on\_trigger\_trigger”** getriggert, der alle Datensätze löscht, die im Feld **“acknowledgment”** die Nummer 3 eingetragen haben. Die Nummer drei bedeutet, dass beide Nachbarn Nodes die Änderung repliziert haben.

# 

# SQL Befehle für das Einrichten der Multi-Master Replication

Die SQL Befehle sind aufgeteilt in DB-Links, Synonyme, Tabellen, Triggers und Prozeduren erstellen und können nacheinander im Lesefluss ausgeführt werden.

## DB-Links erstellen

### Für Orion

|  |
| --- |
| CREATE DATABASE LINK ganymed.sirius.fhnw.ch CONNECT TO ddm20 IDENTIFIED BY ddm20 USING 'ganymed';  CREATE DATABASE LINK titania.sirius.fhnw.ch CONNECT TO ddm20 IDENTIFIED BY ddm20 USING 'titania'; |

### Für Titania

|  |
| --- |
| CREATE DATABASE LINK ganymed.sirius.fhnw.ch CONNECT TO ddm20 IDENTIFIED BY ddm20 USING 'ganymed';  CREATE DATABASE LINK orion.sirius.fhnw.ch CONNECT TO ddm20 IDENTIFIED BY ddm20 USING 'orion'; |

### Für Ganymed:

|  |
| --- |
| CREATE DATABASE LINK orion.helios.fhnw.ch CONNECT TO ddm20 IDENTIFIED BY ddm20 USING 'orion';  CREATE DATABASE LINK titania.sirius.fhnw.ch CONNECT TO ddm20 IDENTIFIED BY ddm20 USING 'titania'; |

## Synonymes erstellen für einfachere Namen

Die Synonyme werden für eine einfachere Verwaltung durch einfachere Namensgebung verwendet. Es gilt für jeden immer das selbe:

* x zeigt immer auf sich selbst
* y und z zeigen auf die zwei anderen Nodes

Für Multi-Master Replication mit 3 Nodes ist es nicht von Bedeutung allen einen eindeutigen Namen zu geben. Die Nodes müssen mit allen gemeinsam synchronisieren können.

### Für Orion

|  |
| --- |
| CREATE OR REPLACE SYNONYM p\_owners\_x FOR p\_owners; CREATE OR REPLACE SYNONYM p\_owners\_log\_x FOR p\_owners\_log;  CREATE OR REPLACE SYNONYM p\_owners\_y FOR [p\_owners@titania.janus.fhnw.ch](mailto:p_owners@titania.janus.fhnw.ch);  CREATE OR REPLACE SYNONYM p\_owners\_log\_y FOR p\_owners\_log@titania.janus.fhnw.ch; CREATE OR REPLACE SYNONYM delete\_logs\_on\_trigger\_y FOR delete\_logs\_on\_trigger@titania.janus.fhnw.ch;  CREATE OR REPLACE SYNONYM p\_owners\_z FOR p\_owners@ganymed.sirius.fhnw.ch; CREATE OR REPLACE SYNONYM p\_owners\_log\_z FOR p\_owners\_log@ganymed.sirius.fhnw.ch; CREATE OR REPLACE SYNONYM delete\_logs\_on\_trigger\_z FOR delete\_logs\_on\_trigger@ganymed.sirius.fhnw.ch; |

### 

### Für Titania

|  |
| --- |
| CREATE OR REPLACE SYNONYM p\_owners\_x FOR p\_owners; CREATE OR REPLACE SYNONYM p\_owners\_log\_x FOR p\_owners\_log;  CREATE OR REPLACE SYNONYM p\_owners\_y FOR p\_owners@orion.helios.fhnw.ch; CREATE OR REPLACE SYNONYM p\_owners\_log\_y FOR p\_owners\_log@orion.helios.fhnw.ch; CREATE OR REPLACE SYNONYM delete\_logs\_on\_trigger\_y FOR delete\_logs\_on\_trigger@orion.helios.fhnw.ch;   CREATE OR REPLACE SYNONYM p\_owners\_z FOR p\_owners@ganymed.sirius.fhnw.ch; CREATE OR REPLACE SYNONYM p\_owners\_log\_z FOR p\_owners\_log@ganymed.sirius.fhnw.ch; CREATE OR REPLACE SYNONYM delete\_logs\_on\_trigger\_y FOR delete\_logs\_on\_trigger@ganymed.sirius.fhnw.ch; |

### Für Ganymed

|  |
| --- |
| CREATE OR REPLACE SYNONYM p\_owners\_x FOR p\_owners; CREATE OR REPLACE SYNONYM p\_owners\_log\_x FOR p\_owners\_log;  CREATE OR REPLACE SYNONYM p\_owners\_y FOR p\_owners@orion.helios.fhnw.ch; CREATE OR REPLACE SYNONYM p\_owners\_log\_y FOR p\_owners\_log@orion.helios.fhnw.ch; CREATE OR REPLACE SYNONYM delete\_logs\_on\_trigger\_y FOR delete\_logs\_on\_trigger@orion.helios.fhnw.ch;  CREATE OR REPLACE SYNONYM p\_owners\_z FOR p\_owners@titania.janus.fhnw.ch; CREATE OR REPLACE SYNONYM p\_owners\_log\_z FOR p\_owners\_log@titania.janus.fhnw.ch; CREATE OR REPLACE SYNONYM delete\_logs\_on\_trigger\_y FOR delete\_logs\_on\_trigger@titania.janus.fhnw.ch; |

## 

## 

## Tabellen auf allen Nodes erstellen

|  |
| --- |
| DROP TABLE p\_owners; CREATE TABLE p\_owners (  ownerno VARCHAR2(5) NOT NULL,  fname VARCHAR2(15) NOT NULL,  lname VARCHAR2(15) NOT NULL,  city VARCHAR2(15) NOT NULL,  postcode VARCHAR2(8) NULL,  CONSTRAINT ownerno\_p\_owners\_pk PRIMARY KEY (ownerno)); |

|  |
| --- |
| DROP TABLE p\_owners\_log; CREATE TABLE p\_owners\_log (  transaction\_id RAW(16) NOT NULL,  transaction\_timestamp TIMESTAMP NOT NULL,  acknowledgment NUMBER(1) DEFAULT 0 NOT NULL,  action VARCHAR(3) NOT NULL,  old\_ownerno VARCHAR2(5) NOT NULL,  new\_ownerno VARCHAR2(5),  fname VARCHAR2(15),  lname VARCHAR2(15),  city VARCHAR2(15),  postcode VARCHAR2(8),  CONSTRAINT transaction\_id\_p\_owners\_log\_pk PRIMARY  KEY (transaction\_id)); |

|  |
| --- |
| DROP TABLE to\_update; CREATE TABLE to\_update (  transaction\_id RAW(16) NOT NULL,  node VARCHAR(1) NOT NULL,  CONSTRAINT transaction\_id\_to\_update\_pk PRIMARY  KEY (transaction\_id)); |

|  |
| --- |
| DROP TABLE delete\_logs\_on\_trigger; CREATE TABLE delete\_logs\_on\_trigger (  dummy VARCHAR(1) DEFAULT 0 NOT NULL,  CONSTRAINT delete\_logs\_on\_trigger\_pk PRIMARY KEY (dummy)); |

## Triggers zum Loggen und Löschen der Logs erstellen

|  |
| --- |
| CREATE OR REPLACE TRIGGER p\_owners\_trigger AFTER INSERT OR UPDATE OR DELETE ON p\_owners FOR EACH ROW DECLARE tmp\_action VARCHAR(3); old\_ownerno VARCHAR2(5) := :OLD.ownerno; new\_ownerno VARCHAR2(5) := :NEW.ownerno; new\_fname VARCHAR2(15) := :NEW.fname; new\_lname VARCHAR2(15) := :NEW.lname; new\_city VARCHAR2(15) := :NEW.city; new\_postcode VARCHAR2(8) := :NEW.postcode; BEGIN  IF INSERTING THEN  tmp\_action := 'INS';  ELSIF UPDATING THEN  tmp\_action := 'UPD';  ELSIF DELETING THEN  tmp\_action := 'DEL';  END IF;   IF old\_ownerno IS NULL THEN  old\_ownerno := new\_ownerno;  END IF;   INSERT INTO p\_owners\_log (transaction\_id, transaction\_timestamp,  action, old\_ownerno, new\_ownerno, fname, lname, city, postcode)  VALUES (SYS\_GUID(), systimestamp, tmp\_action, old\_ownerno,  new\_ownerno, new\_fname, new\_lname, new\_city, new\_postcode); END; |

|  |
| --- |
| CREATE OR REPLACE TRIGGER delete\_logs\_on\_trigger\_trigger AFTER INSERT OR UPDATE OR DELETE ON delete\_logs\_on\_trigger BEGIN  DELETE FROM p\_owners\_log WHERE acknowledgment = 3; END; |

## Prozedur zum Synchronisieren erstellen

### Execute\_on\_myself

Diese Prozedur wird verwendet um den SQL Code lesbarer zu gestalten. Sonst würde die **“sync\_node”** Prozedur sehr gross werden.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE execute\_on\_myself(action VARCHAR2, old\_ownerno VARCHAR2, new\_ownerno VARCHAR2, fname VARCHAR2, lname VARCHAR2, city VARCHAR2, postcode VARCHAR2) AS BEGIN  IF action = 'INS' THEN  INSERT INTO p\_owners\_x (ownerno, fname, lname, city,  postcode)  VALUES (new\_ownerno, fname, lname, city, postcode);  ELSIF action = 'UPD' THEN  UPDATE p\_owners\_x SET  ownerno = new\_ownerno,  fname = fname,  lname = lname,  city = city,  postcode = postcode  WHERE ownerno = old\_ownerno;  ELSIF action = 'DEL' THEN  DELETE FROM p\_owners\_x WHERE ownerno = old\_ownerno;  END IF; END; |

### 

### View Erstellen um den Code lesbarer zu schreiben

Diese View wird verwendet um den SQL Code lesbarer zu gestalten. Sonst würde die **“sync\_node”** Prozedur sehr gross werden.

#### Für Helios und Ganymed

|  |
| --- |
| CREATE OR REPLACE VIEW collegue\_changes AS (SELECT 1 as node, transaction\_timestamp, action, transaction\_id, old\_ownerno, new\_ownerno, fname, lname, city, postcode  FROM p\_owners\_log\_z WHERE acknowledgment != 1)  UNION  (SELECT 2 as node, transaction\_timestamp, action, transaction\_id, old\_ownerno, new\_ownerno, fname, lname, city, postcode FROM p\_owners\_log\_y WHERE acknowledgment != 2)  ORDER BY transaction\_timestamp ASC; |

#### Für Titania

|  |
| --- |
| CREATE OR REPLACE VIEW collegue\_changes AS (SELECT 1 as node, transaction\_timestamp, action, transaction\_id, old\_ownerno, new\_ownerno, fname, lname, city, postcode  FROM p\_owners\_log\_y WHERE acknowledgment != 1)  UNION  (SELECT 2 as node, transaction\_timestamp, action, transaction\_id, old\_ownerno, new\_ownerno, fname, lname, city, postcode FROM p\_owners\_log\_z WHERE acknowledgment != 2)  ORDER BY transaction\_timestamp ASC; |

## 

### Update\_Logs\_on\_collegues

Diese Prozeduren werden verwendet um den SQL Code lesbarer zu gestalten. Sonst würde die **“sync\_node”** Prozedur sehr gross werden.

#### Für Orion:

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_logs\_on\_collegues AS BEGIN  UPDATE p\_owners\_log\_y SET acknowledgment = acknowledgment + 2  WHERE transaction\_id IN  (SELECT transaction\_id FROM to\_update WHERE node = 2);  UPDATE p\_owners\_log\_z SET acknowledgment = acknowledgment + 1  WHERE transaction\_id IN  (SELECT transaction\_id FROM to\_update WHERE node = 1); END; |

#### Für Ganymed:

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_logs\_on\_collegues AS BEGIN  UPDATE p\_owners\_log\_y SET acknowledgment = acknowledgment + 2  WHERE transaction\_id IN  (SELECT transaction\_id FROM to\_update WHERE node = 2);  UPDATE p\_owners\_log\_z SET acknowledgment = acknowledgment + 1  WHERE transaction\_id IN  (SELECT transaction\_id FROM to\_update WHERE node = 1); END; |

#### Für Titania:

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_logs\_on\_collegues AS BEGIN  UPDATE p\_owners\_log\_y SET acknowledgment = acknowledgment + 1  WHERE transaction\_id IN  (SELECT transaction\_id FROM to\_update WHERE node = 1);  UPDATE p\_owners\_log\_z SET acknowledgment = acknowledgment + 2  WHERE transaction\_id IN  (SELECT transaction\_id FROM to\_update WHERE node = 2); END; |

### Trigger\_delete\_logs\_on\_collegues

|  |
| --- |
| CREATE OR REPLACE PROCEDURE trigger\_delete\_logs\_on\_collegues AS BEGIN  UPDATE delete\_logs\_on\_trigger\_y SET dummy = 0 WHERE dummy = 0;  UPDATE delete\_logs\_on\_trigger\_z SET dummy = 0 WHERE dummy = 0; END; |

### Sync\_node

|  |
| --- |
| CREATE OR REPLACE PROCEDURE sync\_node AS BEGIN  EXECUTE IMMEDIATE 'ALTER TABLE p\_owners DISABLE ALL TRIGGERS';   FOR rec IN (SELECT \* FROM collegue\_changes)  LOOP  execute\_on\_myself(rec.action, rec.old\_ownerno, rec.new\_ownerno,   rec.fname, rec.lname, rec.city, rec.postcode);  INSERT INTO to\_update (transaction\_id, node)   VALUES (rec.transaction\_id, rec.node);  END LOOP;   update\_logs\_on\_collegues();  trigger\_delete\_logs\_on\_collegues();  EXECUTE IMMEDIATE 'TRUNCATE TABLE to\_update';  EXECUTE IMMEDIATE 'ALTER TABLE p\_owners ENABLE ALL TRIGGERS'; END; EXECUTE sync\_node; |

## 

## 

## Sync aller Nodes

|  |
| --- |
| EXECUTE sync\_node@ganymed.sirius.fhnw.ch; EXECUTE sync\_node@titania.janus.fhnw.ch; EXECUTE sync\_node@orion.helios.fhnw.ch; |

# Testkonzept und Testresultate

Testszenarien, Testfälle und Testresultate mit SQL-Befehlen für Update, Insert und De-

Lete