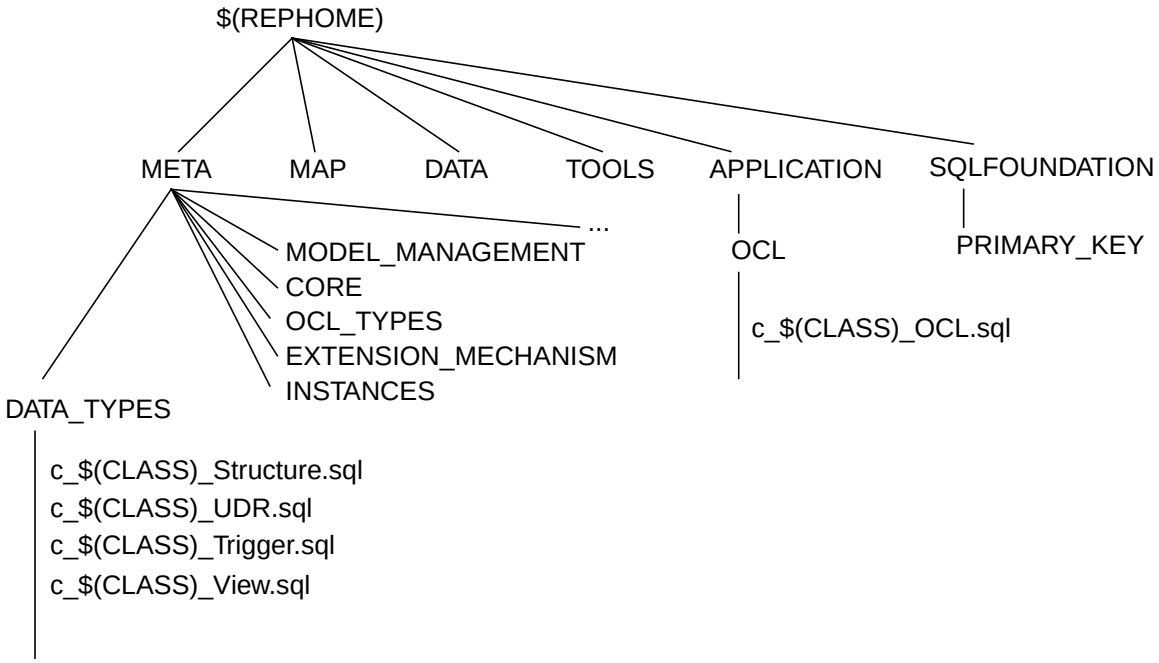
Kapitel 2 Installation und Wartung der Datenbank-Ebene

Dieses Dokument beschreibt die Verzeichnisstruktur der Dateien zur Installation der Datenbankebene für das UML-Repository. Weiterhin werden hier Namenskonventionen festgelegt, die im weiteren für Erweiterungen einzuhalten sind.

2.1 Die Verzeichnisstruktur

Im Hauptverzeichnis $(REPHOME) sind sechs Unterverzeichnisse mit den Namen ‘META’, ‘MAP’, ‘DATA’, ‘TOOLS’, ‘APPLICATION’ und ‘SQLFOUNDATION’ zu finden (Abbildung 1).

Abbildung 1 Verzeichnisstruktur



❑

Das Verzeichnis ‘META’ enthält für jedes Package des UML-Metamodells ein Verzeichnis, beispielsweise ‘CORE’. Die Verzeichnisnamen bestehen immer nur aus Groß-Buchstaben. Ein wei-

teres Subverzeichnis von ‘META’ mit den Namen ‘INSTANCE’ enthält die DML-Anweisungen zum Füllen des Schemas mit Daten. Hinzu kommt ein Verzeichnis mit den Erweiterungen für die OCL-Integration.

In den Verzeichnissen für die Packages liegen die Dateien mit den SQL-DDL-Anweisungen zum Anlegen des Schemas. Für jede Klasse gibt es mehrere Dateien, eine zum Anlegen der Typen und Tabellen, eine zum Anlegen der UDRs, eine zum Anlegen der Trigger und eine zum Anlegen der Views. Tabelle 1 gibt einen Überblick über die Namensregeln für Dateien. Die Namen für $(CLASS) sind identisch mit den Namen aus dem UML-Metamodell.

Tabelle 1 Namenskonvention für die DB-Ebene

Dateityp	Namenskonvention (Datei)	Namenskonvention (Objekt)
Struktur	c_$(CLASS)_Structure.sql	meta_$(TYPENAME)_ty
		meta_$(TYPENAME)_ta
UDRs	c_$(CLASS)_UDR.sql	meta_$(UDRNAME)_pr
Trigger	c_$(CLASS)_Trigger.sql	meta_$(TYPENAME)_$(TRIGGERNAME)_tr
Sichten	c_$(CLASS)_View.sql	meta_$(TYPENAME)_vi_$(X)
		mit X ∈ {_i, _u, _s, _d, _i_o, _u_o, _s_o, _d_o}
Constraint	c_$(CLASS)_OCL.sql	-

❑

Die Namenskonventionen für die Schemaelemente ist ebenfalls Tabelle 1 zu entnehmen. Weiterhin gilt, daß der Name für Typen und Klassen sich aus den Namen im UML-Metamodell ergeben. Da im DBMS keine Unterscheidung zwischen Groß- und Kleinschreibung gemacht wird, werden Großbuchstaben in Namen durch ein vorangestelltes Zeichen ‘_’ gekennzeichnet, beispielsweise der Name ‘model_element_ty’ für den Typ zur Klasse ModelElement im Metamodell. Sind im im Namen aus dem UML-Modell bereits Tiefstriche, so werden diese durch zwei Tiefstriche ersetzt. Damit ist die Namenkonvertierung eindeutig. Um Namenskonflikte zwischen dem Metadatenschema und dem Produktdatenschema in der gleichen Datenbank zu vermeiden wird dabei in allen Namen die Kennung ‘meta_’ vorangestellt.

Das Verzeichnis ‘DATA’ wird automatisch aus den Dateien im Verzeichnis ‘META’ generiert. In allen Namen wird dabei die Kennung ‘meta_’ durch ‘ data_’ ersetzt. Da das Verzeichnis, bzw. sein Inhalt automatisch generiert wird, sollten hier keine Änderungen vorgenommen werden.

Im Verzeichnis ‘MAP’ werden Dateien zur Erzeugung eines Schemas zur Speicherung der Abbildungsinformation ablegt. In dem hier (bald) definierten Schema wird gespeichert, welcher Zusammenhang zwischen dem UML-Modell im Metadatenschema und dem Produktatenschema besteht.

Die Komponenten in ‘META’ und ‘DATA’ benötigen gemeinsame Komponenten, wie beispielsweise die Implementierung des Primärschlüssels. Diese grundlegenden Komponenten werden im Verzeichnis ‘SQLFOUNDATION’ ablegt.

Manche Teile, die wir zur DB-Ebene zählen, beispielsweise einige der Integritätsbedingungen, beziehen sich sowohl auf die Metadaten als auf auf die Produktdaten. Diese werden in Unterverzeichnisse des Verzeichnisses ‘APPLICATION’ abgelegt. Augenblicklich gibt es dort nur das Unterverzeichnis ‘OCL’ für die Integritätbedinungen aus dem UML-Metamodell.

Ein weiteres Verzeichnis, ‘TOOLS’, ist angelegt, um Werkzeuge aufzunehmen. Hier liegt beispielsweise das Perl-Script ‘cm2d.pl’ mit dem rekursiv alle Namen in einem Verzeichnis von Metadatenschemanamen auf Produktdatenschemanamen umgestellt werden können.

2.2 Installation des DB-Schemas

Die Installationsdateien für die Datenbankebene des UML-Repository wird im SERUM-Repository verwaltet. Wer eine lokale Installation vornehmen will, muß zunächst die aktuelle Version aus dem Repository auslesen:

```
procject_co UMLREP2
```

Damit erhält der Aufrufer eine lokale Kopie der oben besprochenen Verzeichnishierarchie. Für die Installation des Repositories in der Datenbank stehen eine Sammlung von make-Dateien zur Verfügung. Um die Installation an die eigene Umgebung anzupassen, muß die Datei ‘global.mk’ im Verzeichnis UMLREP2 editiert werden. In den unten aufgeführten Zeilen sind die umgebungspezifischen Angaben einzutragen. Die Variable ‘BATCH’ wird auf den Namen des Programms zur Abarbeitung von Batch-Dateien gesetzt. In unserem Fall ist dies üblicherweise ‘dbaccess’. Der Name der (bereits existierenden) Datenbank wird in die Variable ‘DB’ eingetragen und der Name des Datenbankservers unter ‘SERVER’.

```
# DBS-spezifische Einstellungen
BATCH    = dbaccess
DB       = hpserum        // Name der Datenbank
SERVER   = ifmxdev2       // Name des Servers
```

Anlegen des Repository

Wenn die Einstellungen in ‘global.mk’ gemacht wurden kann die Installation des UML-Repositories mit

```
make
```

gestartet werden. Zunächst wird dabei das Schema für die Metadaten angelegt und mit Daten gefüllt, dann wird aus den Installationsdateien für dieses Schema das Schema für die Produktdaten erzeugt und auch dieses angelegt und mit Daten gefüllt.

Löschen des Repository

Das Repository läßt sich mit

```
make clean
```

wieder aus der Datenbank entfernen.

Fehler

Sollten bei der Installation oder dem Entfernen Fehler auftreten, so ist die folgende Vorgehensweise die einfachste, um wieder eine konsistente Datenbank zu erhalten. Man löscht zunächst die Datenbank, ruft dann ‘make clean’ auf um die Verzeichnisse aufzuräumen und legt die Datenbank wieder an. Dann sollte sich die Datenbankebene des Repositories mit ‘make’ wieder installieren lassen.

2.3 Wartung des DB-Schemas

Allgemeines zur Versionierung

2.3.1 Anlegen eines neuen Paketes

2.3.2 Löschen eines Paketes

2.3.3 Hinzufügen einer Klasse

Soll eine neue Klasse in ein bestehendes Package eingefügt werden, so müssen, je nach Klasse, bis zu vier Dateien angelegt werden.

Die erste Datei trägt den Namen ‘c_<Klassenname>_Structure.sql’. Sie enthält die DDL-Befehle zum Anlegen von Typen und Tabellen.

Die zweite Datei trägt den Namen ‘c_<Klassenname>_View.sql. Sie enthält die DDL-Befehle zum Anlegen der Sichten auf den Tabellen.

Die dritte Datei trägt den Namen ‘c_<Klassenname>_UDR.sql. Sie enthält die DDL-Befehle zum Anlegen der benutzerdefinierten Funktionen für die Typen.

Die vierte Datei trägt den Namen ‘c_<Klassenname>_Trigger.sql. Sie enthält die DDL-Befehle zum Anlegen von Triggern auf Tabellen.

Jedes Package enthält eine Datei ‘makefile’. Diese enthält eine Liste der Klassen in diesem Pakkage. Hier wird die neue Klasse eingetragen. Die Klassen werden in der Reihenfolge abgearbeitet, in der sie in der Liste stehen. Bei Abhängigkeiten zwischen Klassen muß dies beachtet und die neue Klasse an passender Stelle in der Liste eingefügt werden.

2.3.4 Ändern einer Klasse

Die Definitionsdateien für Klassen können einfach geändert werden.

2.3.5 Löschen einer Klasse