

Oracle-Views

auch nach umfassenden Änderungen per Mausklick kompilierbar.

Datenbanksysteme kompilieren Datenbank-Objekte wie Sichten, Funktionen, Prozeduren und Trigger in ihren eigenen Core, sodass sie sehr schnell ausgeführt werden können. Gerade für Oracle-Systeme gibt es hierfür sogar Tipps und Analysefunktionen, wie man die Laufzeit anhand einiger Regeln (z.B. Einsatz von Konstanten statt Variablen usw.) verbessern kann. Das Problem hierbei ist, dass man nach Änderungen an der „Basis“ wie dem Hinzufügen, Ändern oder Entfernen von Feldern an einer Tabelle die Objekte ungültig sind und nicht benutzt werden können. Ein stures Durchkompilieren „von oben nach unten“, wie es z.B. Oracles SQL-Developer tut, funktioniert nicht, wenn einzelne Objekte wie z. B. Sichten teilweise sogar mehrfach voneinander abhängen. Hier muss nach der Abhängigkeit kompiliert werden. Das Resultat ist, dass man nach einem regulären Update (beim Kunden) alle Sichten (momentan ca. 200) manuell kontrollieren muss, ob diese Daten zurückgeben. Wenn nicht, muss man die SQL-Quelle der betroffenen Sicht nach Datenbank-Objekten durchforsten und prüfen, ob diese ihrerseits vernünftig kompiliert werden können. Das Problem hierbei ist, dass es wenig Information darüber gibt, wo bei Oracle die Informationen über die Abhängigkeiten versteckt sind und wie man diese nutzen soll. Auch Internet-Recherchen helfen wenig weiter. Außerdem soll das Ganze in dem Schema ablaufen, in welchem die Datenbank des Kunden läuft, denn in der Regel bekommt man beim Kunden keine SYS-Privilegien.

Es hatte sich herauskristallisiert, dass Oracle-Sichten zwecks eingangs erwähnter Laufzeitoptimierung sämtliche Feldnamen in ihre Sichten hat einkompiliert, selbst wenn man ein bei Oracle untypisches „SELECT * FROM...“ in die SQL-Quelle geschrieben hat. (Die Diskussion darüber, ob „SELECT *“ sinnvoll ist und die Datenbank dadurch einigermaßen handhabbar macht oder nur die Faulheit der Datenbank-Designer unterstützt und inkonsistent ist, kann innerhalb der Programmiererguppen philosophische und epische Ausmaße annehmen.) Die DDL einer solchen View sieht damit folgendermaßen aus:

```
CREATE OR REPLACE VIEW
"SCHEMA"."MYVIEW" ("FELD1", "FELD2") AS
SELECT * FROM TABELLE
;
```

Hat die Tabelle „Tabelle“ mehr oder weniger als zwei Felder oder wurde eins davon umbenannt, so ist die Sicht ungültig. Eine Verschärfung des Problems lässt sich leicht konstruieren:

```
CREATE OR REPLACE VIEW
"SCHEMA"."MYVIEW2" ("FELD1", "FELD2") AS
SELECT FELD1, FELD2 FROM "SCHEMA"."MYVIEW"
;
```

Obwohl MYVIEW2 keinen Stern sondern definierte Feldnamen benutzt, wird diese Sicht nach der

Die Ausführung des Statements ergibt eine Liste mit allen Views des Schemas.

Zu Punkt 2: Eine Liste der Views, von denen unsere View abhängig sind. Es ist kein Geheimnis, dass sich diese Information in der Systemsicht

USER_DEPENDENCIES befindet, man muss nur die Information so aufbereiten, dass man sie auch nutzen kann. Für jede Sicht lässt sich somit eine Liste aller referenzierten Sichten zurückgeben:

```
SELECT REFERENCED_NAME FROM USER_DEPENDENCIES
WHERE TYPE = 'VIEW' AND REFERENCED_TYPE =
'VIEW' AND NAME = 'MYVIEW2' GROUP BY NAME,
REFERENCED_NAME
```

Obiges Statement gibt die Liste aller Views zurück, von denen MYVIEW2 abhängt, in unserem Beispiel also MYVIEW.

Zu Punkt 3: Die Möglichkeit, die DDL automatisiert aus der Datenbank zu bekommen. Das schafft das Package DBMS.METADATA, die die Prozedur GET_DDL anbietet:

```
SELECT DBMS_METADATA.GET_DDL('VIEW', 'MYVIEW')
FROM DUAL
```

Dieses Statement gibt exakt die DDL vom MYVIEW zurück, wie sie oben geschrieben wurde, inklusive CREATE und der Feldnamen. Für eine gültige Neuanlage von MYVIEW muss also alles zwischen dem Viewnamen und dem Schlüsselwort „AS“ gelöscht werden.

Zu Punkt 4: Der Kompilierbefehl für die Sicht:

```
ALTER VIEW MYVIEW COMPILE
```

Dieser Befehl kompiliert die Sicht.

Mit diesem Wissen bauten wir ein Tool, welches alle Sichten nach Abhängigkeiten durchforstet, die referenzierten Sichten zuerst und anschließend die abhängige Sicht neu erstellt und zum Schluss alles nochmal schön durchkompiliert.

Das Tool ist in C# im Visual Studio 2005 geschrieben worden und greift mit Hilfe von DLLs, die in unserem Haus entwickelt wurden, auf die Datenbank zu. Die DLLs sind kein Hexenwerk und kapseln den Zugriff lediglich transparent, sodass der Programmierer weniger Arbeit hat, wenn er mit .NET-Datenbankobjekten hantiert. Es gibt drei zentrale Prozeduren in der Worker-Klasse, die das Kompilieren übernehmen:

Die Kompilierfunktion compileFunction, die den

Tabellenänderung nicht funktionieren, da MYVIEW nicht funktioniert. Leider wird dies nirgends angezeigt, da der SQL-Code als solcher ja gültig ist. Auffallen tut dies erst, wenn man auf die View tatsächlich zugreift, dann kommt nämlich ein Fehler. Ein neues Kompilieren bringt auch nichts, da lediglich der vorhandene SQL-Code (gültig) durchkompiliert. Der Fehler bleibt. Es stellte sich schließlich heraus, dass sich die Gültigkeit wiederherstellen lässt, indem man Oracle zwingt, sich die Infos über die Tabelle neu zu holen. Das kann man tun, indem man die DDL **ohne** Feldbezeichnungen neu ausführen lässt:

```
CREATE OR REPLACE VIEW "SCHEMA"."MYVIEW" AS
SELECT * FROM TABELLE
;
```

Jetzt die die Sicht wieder gültig, und nach dem Kompilieren von MYVIEW2 kann auch diese wieder eingesetzt werden.

```
CREATE OR REPLACE VIEW
"SCHEMA"."MYVIEW" ("FELD1", "FELD2") AS
SELECT * FROM TABELLE
;
```

Um diesen Prozess zu automatisieren benötigt man also folgendes:

- Eine Liste mit allen Views
- Eine Liste mit Views, die diese referenzieren
- Eine Möglichkeit, aus der Datenbank die DDL einer Sicht zu bekommen
- Den Kompilierungsbehehl für die DDL

Zu Punkt 1: Die Liste aller im Schema vorhandenen Objekte versteckt sich in der Systemsicht „OBJ“. Grenzt man diese Objekte auf Sichten ein, ergibt sich folgendes Statement:

```
SELECT OBJECT_NAME FROM OBJ WHERE
OBJECT_TYPE = 'VIEW'
```

Viewnamen übergeben bekommt. Diese holt sich per DBMS_METADATA.GET_DDL die DDL der View, entfernt mittels Stringverarbeitung die Feldnamen und führt die DDL neu aus. Anschließend wird kompiliert.

Die Funktion selectViews, die mit zwei Datareadern arbeitet. Der erste Datareader holt alle Views aus der OBJ-Sicht. Für jede OBJ-Sicht werden im zweiten Datareader alle referenzierten Sichten geholt. Gibt es hier Datensätze, werden die referenzierten Sichten zuerst zum Kompilieren via compileFunction geschickt, anschließend die OBJ-Sicht selbst.

Damit hinterher auch im SQL-Developer keine roten Kreuzchen mehr auftauchen, gibt es noch die dritte Prozedur compileOnly, die nach dem kompletten DDL-Durchlauf alle invaliden Views noch einmal kompiliert. In dieser Funktion wurde auch die Falle bzw. der Oracle-Bug umschifft, in die Oracle-Entwickler gern hereinfallen, wenn sie schnell mal eben alles kompilieren wollen: Kompiliert man stur alle OBJ-Views (auch gesunde) durch, ändert sich am Zustand der Views gar nichts (warum auch immer). Kompiliert man aber nur alle invaliden Views durch, werden diese wundersam geheilt. Das Statement für den Datareader, der die Views zum Kompilieren liefert, muss also heißen:

```
SELECT OBJECT_NAME FROM OBJ WHERE OBJECT_TYPE =
'VIEW' AND STATUS = 'INVALID'
```

Der Status „invalid“ ist hier das entscheidende Kriterium.

Zur Sicherheit lassen wir die DDL-Kreation zweimal durchlaufen. So ist gewährleistet, dass auch die abstruseste Abhängigkeit aufgedröselt wird. Für das Kompilieren reicht ein einmaliger Durchlauf.

Nach einer Änderung an einer zentralen Stelle der Datenbank, die 85% aller unserer Views unbrauchbar gemacht hat, brauchte unser Tool ca. zwei Minuten, um die Datenbank wieder benutzbar zu machen. Dies ist natürlich von Komplexität der Datenbank sowie der allgemeinen Performance des Servers abhängig. Erwähnenswert wäre vielleicht noch, dass das Tool natürlich nicht von der Betriebssystem-Umgebung der Oracle-Installation abhängig ist und somit auch mit Linux/Unix-Clustern laufen sollte.

Seitenanfang 