Gesteuerte Freigabe von Entitys und Attributen im Cache

**Zusammenfassung:**

Projektentwickler können durch lokale Worksets sowie durch die Funktion MakeStab() die Freigabe von Entitys auslösen.

**Hintergrund:**

Entitys und ihre Attribute werden gecached, damit sie bei wiederholtem Zugriff nur einmal von der Datenbank geladen werden müssen.

* Entitys durch Aufnahme in den Entity-Cache
* Attribute durch Speichern im Attribute-Cache am Entity

Sobald der Kern annimmt, dass ein Entity wahrscheinlich nicht mehr benötigt wird, wird es wieder freigegeben.

Diese Vorausschau ist schwierig, speziell bei Klassenformeln.

Werden Daten zu lange im Cache gehalten, führt dies zu einem erhöhten Speicherbedarf.

Werden sie zu früh freigegeben, muss wieder auf die Datenbank zugegriffen. Das Zeitverhalten wird verschlechtert und die Datenbank stärker belastet.

Welche Daten freigeben werden ist von der Situation abhängig.

Solange ein Entity noch referenziert wird (durch EntityListe, Klassenformel-Variable usw.), kann es nicht freigegeben werden. Falls aber ein Zugriff auf seine Attribute unwahrscheinlich ist, können diese freigegeben werden. Ein Entity ohne Attribute benötigt nur sehr wenig Speicher.

Falls wider Erwarten doch noch ein Zugriff erfolgt, hält das Entity noch genügend Informationen um die Attribute wieder aus der Datenbank zu laden.

Wird das Entity jedoch nicht mehr referenziert, kann es vollständig freigegeben und aus dem Cache entfernt werden.

Ein Entity befindet sich zu jedem Zeitpunkt in einem der folgenden Zustände:

* Stub nur Id, Klasse und Mandant sind bekannt, Attribute sind nicht geladen
* Clean Attribute sind geladen und wurde noch nicht verändert
* Dirty Attribute sind geladen und wurde verändert aber noch nicht in DB gespeichert.
* New Entity wurde neu angelegt, ist noch nicht in DB gespeichert.
* Deleted Entity ist als gelöscht markiert, alle Attribute wurden freigegeben

Die Attribute eines Entitys können nur im Zustand „Clean“ freigegeben werden.

Das Entity selbst nur im Zustand „Stub“ oder „Clean“.

Bei allen anderen Zuständen müssen die Änderungen am Entity durch Commit in die Datenbank geschrieben oder durch Rollback verworfen werden.

Rollback löst das Aufräumen des Cache aus, eine „garbage collection“ wird mittels „Mark&Sweep“-Verfahren durchgeführt. Commit gibt auch einige Entitys frei, u.a. speziell Entitys aus Datenbank-Views.

Der Kern kann nur wenige Annahmen über das Verhalten von Klassenformeln machen. Die Wahrscheinlichkeit eines zukünftigen Zugriffs auf ein Entity kann ohne zusätzliche Informationen nicht ermittelt werden.

**Implementierung:**

Für den Projektentwickler stehen zwei Werkzeuge zur Verfügung, um die Freigabe von Entitys auszulösen:

* Funktion „MakeStub“
* Worksets

MakeStub wird in Klassenformeln explizit aufgerufen. Die Funktion gibt jedes übergebene Entity frei, falls es sich im Zustand „Clean“ befindet.

Worksets automatisieren diesen Vorgang:

Worksets führen Buch über geladene Entitys und können diese wieder freigeben.

Solange ein Workset aktiv ist, merkt es sich alle neu geladenen Entitys.

Wird das Workset gelöscht, gibt es alle gemerkten Entitys wieder frei.

Falls eine Klassenformel an einer Stelle Entitys einliest, die nur hier benötigt werden, kann für diese Stelle ein neues Workset definiert und aktiviert werden, welches danach automatisch wieder gelöscht wird.

Intern wird die Menge der Worksets als Stapel (Prinzip LIFO, „last in, first out“) verwaltet. Neue Worksets werden auf den Stapel gelegt und beim Löschen wieder vom Stapel entfernt. Das oben auf dem Stapel liegende Workset ist immer das aktive Workset.

**Spracherweiterung:**

Die Programmiersprache der Klassenformeln wurde erweitert und unterstützt jetzt Worksets

Die Definition eines Workset hat die Form

*workset <optional name> statement*

Üblicherweise ist statement ein Block-Statement.

*workset DeterminBestEquipment*

*{*

 *LoadEntitys(….);*

 *LoadEntitys(….);*

*}*

Beim Ablauf der Klassenformel wird vor der Ausführung von *statement* ein neues Workset erzeugt, danach wird es wieder gelöscht. Dadurch werden die durch *statement* geladenen Entitys wieder freigegeben.

In bestimmten Fällen kann das kurzzeitige Aktivieren des vorherigen Worksets erwünscht sein ohne das aktuelle Workset zu beenden. Dazu dient die *parent\_workset*-Anweisung.

*parent\_workset statement*

*parent\_workset* nimmt für die Dauer des folgenden *statements* das aktive Workset vom Stapel ohne es zu löschen. Dadurch wird das vorherige Workset wieder aktiv.

„workset“- und „parent\_workset“-Anweisungen können beliebig ineinander verschachtelt werden.

**Achtung: Falls es kein vorheriges Workset gibt, kann die „parent\_workset“-Anweisung nicht durchgeführt werden und löst zur Laufzeit eine Ausnahme aus.**

*function void DoSomething()*

*{*

 *var entity $e1, $e2, $e3;*

 *var int $iLength;*

*LoadEntities ….; // -> Workset „A“*

 *workset B*

 *{*

*LoadEntities ….; // -> Workset „B“*

 *parent\_workset*

 *{*

*$e1 = LoadEntity … // -> Workset „A“*

*$e2 = LoadEntity … // -> Workset „A“*

 *}*

 *$e3 = LoadEntity … // -> Workset „B“*

*MakeStub($e2); // explizit in Zustand „Stub“ überführen*

*}*

 *$iLength = $e1.Length; // liegt im noch existierenden Workset „A“ und ist weiterhin gecached*

 *$iLength = $e2.Length; // wurde explizit in Zustand „Stub“ überführt -> Datenbankzugriff*

 *$iLength = $e3.Length; // lag in Workset „B“ und wurde implizit in Zustand „Stub“ gebracht -> Datenbankzugriff*

*}*