Goldene Regeln für schlechte Programmierung

Arne Hattendorf

Zusammenfassung

Sätze wie: "Die Datenbank ist egal, die brauche ich nur zum Persistieren" oder: "Performance ist ein Hardware Problem" hört man von Softwareentwicklern, -architekten und Projektleitern mit wenig Datenbankerfahrung gelegentlich. Es entsteht Code der eine extreme Belastung der Datenbank verursachen kann. Aus den Bereichen Datenbankabfrage, Programmcode und User Interface werden typische Fallen in der Softwareentwicklung aufgezeigt, die selbst leistungsstarken Datenbanken Probleme bereiten.

Nicht nur Anfänger stolpern gelegentlich in diese typischen Fallen, das zeigen Beispiele aus der Praxis, die durchweg aus der Feder erfahrener Programmierer stammen. Sie sind in SQL, PL/SQL und Pseudocode ausgeführt, aber so kurz und verständlich gehalten, dass sie leicht auf beliebige andere Programmiersprachen übertragen werden können.

Der Vortrag richtet sich an Anfänger und Fortgeschrittene. Für das Verständnis des Vortrags sind grundlegende Programmierkenntnisse (Array), Beherrschung der Grundlagen von SQL (Select, Where) sowie ein allgemeines Verständnis der Konzepte von relationalen Datenbanken (wie Index, Join, Key, View) hilfreich.

Datenbankdesign und Abfragen

Mindestanforderung Datenbankdesign

Beim Design der Datenbank ist es notwendig, die Möglichkeiten einer relationalen Datenbank zu nutzen. Insbesondere Primary Keys (ID), Unique Keys (zur Abbildung der fachlichen Eindeutigkeit) und Foreign Keys (für Beziehungen von Tabellen untereinander) sind wichtig. Sie gehören nicht zur Geschäftslogik, sondern zur Datenstruktur und müssen aus verschiedensten Gründen in der Datenbank angelegt werden. Auch über Indizierung muss man sich schon beim Design der Datenbank Gedanken machen, damit die wichtigsten Indices vorab mit Augenmaß angelegt sind. Sonst besteht die Gefahr, dass nach den ersten Performance Tests unter Druck viel zu viele Indices mit der Gießkanne verteilt werden.

Abfragen und SQL Fallstricke

Bei SQL bietet es sich auf den ersten Blick an, eine Abfrage wie einen Text herunterzuschreiben. Um unnötige Datenbanklast zu vermeiden ist es wichtig, INNER JOIN zu nutzen, falls möglich, und Funktionen auf Spalten so wie implizite Typkonvertierungen zu vermeiden. Falls nur ein beliebiger Datensatz benötigt wird oder auf Existenz mindestens eines Datensatzes geprüft werden muss, stehen mit ROWNUM und EXIST performante Möglichkeiten zur Verfügung.

Programmierung

Abfragen von Daten in Programmen

Eine häufige Quelle unnötiger Datenbanklast sind Daten, die von Programmen abgerufen, aber nicht genutzt werden. Wenn einzelne Datensätze benötigt werden, muss die Einschränkung schon im SQL erfolgen, anstelle erst alle Zeilen abzurufen und dann im Programm auszusortieren. Cursor, Funktionen und Prozeduren sollten nur dann wiederverwendet werden, wenn die genau die Daten abrufen, die benötigt werden. Gerade das gutgemeinte Wiederverwenden von Prozeduren kann sonst in der Datenbank für unnötige Last sorgen.

Datenverarbeitung in Schleifen

Bei der Verarbeitung größerer Datenmengen sollten die Daten nicht zeilenweise aus der DB abgerufen werden. Zunächst die Daten in ein Array holen, dann verarbeiten. Falls sehr große Datenmengen vorliegen, die Daten blockweise verarbeiten. BULK COLLECT, LIMIT und FORALL sind hier sehr hilfreich. Generell sollten Arrays eingesetzt werden, wo immer Gruppen von Datensätzen im Spiel sind, dabei die richtige Art von Array wählen, PLSQL z. B. bietet verschiedene Möglichkeiten.

User Interface

Suche als Problem

Wer auch immer die Oberfläche programmiert, sollte sie auf keinen Fall völlig losgelöst von der Programmlogik und dem Datenmodell betrachten. Eine einzelne Suche, insbesondere ein Teilstring Vergleich über mehrere Felder (in vielen Oberflächen Standard), kann in der Datenbank Millionen von Abfragen auslösen. Daher müssen Datenmodell, Programmlogik und Oberfläche aufeinander abgestimmt sein.

Fluch und Segen in Views

Views können in SQL Anweisungen auf die gleiche Art und Weise genutzt werden, wie Tabellen. Bei einer Tabelle ist der Aufwand eine einzelne Zeile abzufragen gering, bei Views hängt er stark von der unterliegenden Logik ab. Eine einzelne Zeile kann beliebig viele Datenbankzugriffe auslösen, die Art des Zugriffs ist ebenfalls entscheidend. Insbesondere in der Entwicklung von User Interfaces werden häufig Views genutzt, der Aufbau jeder genutzten View muss vor der Nutzung analysiert werden.

Beispiele

- DB-Engine verwirren
- Zu viele Daten abrufen
- Unachtsame Nutzung von Views

Fazit

Wenn man typische Fallen vermeidet, kann man zumindest mittelguten Code schreiben. Trotzdem gibt es nicht viele Regeln mit universeller Gültigkeit. Der Einsatz des eigenen Verstandes ist unverzichtbar.

Think before you Type! (Beim Tippen das Gehirn nicht abschalten.)