Proaktive modellbasierte Performance-Analyse und -Vorhersage von Datenbankanwendungen

Christoph Koch

Friedrich-Schiller-Universität Jena
Lehrstuhl für Datenbanken und Informationssysteme
Ernst-Abbe-Platz 2
07743 Jena
Christoph.Koch@uni-jena.de

DATEV eG
Abteilung Datenbanken
Paumgartnerstr. 6 - 14
90429 Nürnberg
Christoph.Koch@datev.de

KURZFASSUNG

Moderne (Datenbank-)Anwendungen sehen sich in der heutigen Zeit mit immer höheren Anforderungen hinsichtlich Flexibilität, Funktionalität oder Verfügbarkeit konfrontiert. Nicht zuletzt für deren Backend – ein meist relationales Datenbankmanagementsystem – entsteht dadurch eine kontinuierlich steigende Komplexität und Workload, die es frühestmöglich proaktiv zu erkennen, einzuschätzen und effizient zu bewältigen gilt. Die dazu nötigen Anwendungs- und Datenbankspezialisten sind jedoch aufgrund immer engerer Projektpläne, kürzerer Release-Zyklen und weiter wachsender Systemlandschaften stark ausgelastet, sodass für regelmäßige proaktive Expertenanalysen hinsichtlich der Datenbank-Performance kaum Kapazität vorhanden ist.

Zur Auflösung dieses Dilemmas stellt dieser Beitrag ein Verfahren vor, mit dessen Hilfe frühzeitig auf Grundlage der Datenmodellierung und synthetischer Datenbankstatistiken Performance-Analysen und -Vorhersagen für Anwendungen mit relationalem Datenbank-Backend durchgeführt und deren Ergebnisse auf leicht zugängliche Weise visualisiert werden können.

Kategorien und Themenbeschreibung

Data Models and Database Design, Database Performance

Allgemeine Bestimmungen

Performance, Design

Schlüsselwörter

Performance, Proaktivität, Statistiken, relationale Datenbanken, Modellierung, UML, Anwendungsentwicklung

In: G. Specht, H. Gamper, F. Klan (eds.): Proceedings of the 26 GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken),
21.10.2014 - 24.10.2014, Bozen, Italy, published at http://ceur-ws.org.

1. EINLEITUNG

Zur Erfüllung komplexerer Anforderungen und maximalen Benutzerkomforts ist gute Performance eine Grundvoraussetzung für moderne Datenbankanwendungen. Neben Anwendungs-Design und Infrastrukturkomponenten wie Netzwerk oder Anwendungs- beziehungsweise Web-Server wird sie maßgeblich durch die Performance ihres Datenbank-Backends – wir beschränken uns hier ausschließlich auf relationale Datenbankmanagementsysteme (DBMS) – bestimmt [1]. Dabei ist die Datenbank-Performance einer Anwendung selbst ebenfalls durch zahlreiche Faktoren beeinflusst. Während Hardware- und systemseitige Eigenschaften oftmals durch bestehende Infrastrukturen vorgegeben sind, können speziell das Datenbank-Design sowie die anwendungsseitig implementierten Zugriffe mittels SQL weitgehend frei gestaltet werden. Hinzu kommt als Einflussfaktor noch die Beschaffenheit der zu speichernden/gespeicherten Daten, die sich in Menge und Verteilung ebenfalls stark auf die Performance auswirkt.

Das **Datenbank-Design** entwickelt sich über unterschiedlich abstrakte, aufeinander aufbauende Modellstrukturen vom konzeptionellen hin zum physischen Datenmodell. Bereits bei der Entwicklung dieser Modelle können „Designfehler" wie beispielsweise fehlende oder „übertriebene" Normalisierungen gravierende Auswirkungen auf die späteren Antwortzeiten des Datenbanksystems haben. Der Grad an Normalisierung selbst ist jedoch nur als vager Anhaltspunkt für die Performance von Datenbanksystemen anzusehen, der sich ab einem gewissen Maß auch negativ auswirken kann. Eine einfache Metrik zur Beurteilung der Qualität des Datenbank-Designs bezüglich der zu erwartenden Performance (in Abhängigkeit anderer Einflussfaktoren, wie etwa der Workload) existiert nach vorhandenem Kenntnisstand nicht.

Etwas abweichend dazu verhält es sich mit dem Einfluss der **Workload** – repräsentiert als Menge von SQL-Statements und der Häufigkeit ihrer Ausführung, die von der Anwendung an das Datenbanksystem zum Zugriff auf dort gespeicherte Daten abgesetzt wird. Moderne DBMS besitzen einen kostenbasierten Optimierer zur Optimierung eingehender Statements. Dieser berechnet mögliche Ausführungspläne und wählt unter Zuhilfenahme von gesammelten Objekt-Statistiken den günstigsten Ausführungsplan zur Abarbeitung eines SQL-Statements aus. Mittels DBMS-internen Mechanismen – im Folgenden als

EXPLAIN-Mechanismen bezeichnet – besteht die Möglichkeit, noch vor der eigentlichen Ausführung von Statements den vom Optimierer bestimmten optimalen Ausführungsplan ermitteln und ausgeben zu lassen. Zusätzlich umfasst das EXPLAIN-Ergebnis eine Abschätzung der zur Abarbeitung des Ausführungsplans erwarteten Zugriffskosten bezüglich der CPU-und I/O-Zeit – fortan als Kosten bezeichnet. Anhand dieser Informationen können bereits frühzeitig in Hinblick auf die Datenbank-Performance (häufige) teure Zugriffe erkannt und gegebenenfalls optimiert werden. Voraussetzung für dieses Vorgehen ist allerdings, dass dem DBMS zur Berechnung der Ausführungspläne repräsentative **Datenbank-Statistiken** vorliegen, was insbesondere für neue Datenbankanwendungen nicht der Fall ist.

Auf der anderen Seite sehen sich sowohl Anwendungsentwickler- beziehungsweise -designerteams als auch Datenbankspezialisten mit immer komplexeren Anforderungen und Aufgaben konfrontiert. Kapazitäten für umfangreiche Performance-Analysen oder auch nur die Aneignung des dafür nötigen Wissens sind oft nicht gegeben. Nicht zuletzt deshalb geraten proaktive Performance-Analysen verglichen mit beispielsweise funktionalen Tests vermehrt aus dem Fokus.

Das im vorliegenden Beitrag vorgestellte modellbasierte Konzept setzt an diesen beiden Problemen an und stellt Mechanismen vor, um auf einfache Art und Weise eine repräsentative proaktive Analyse der Datenbank-Performance zu ermöglichen. Nachdem in **Kapitel 2** eine Abgrenzung zu alternativen/verwandten Ansätzen gegeben wird, rückt **Kapitel 3** den Entwicklungsprozess einer Datenbank-Anwendung in den Fokus. **Kapitel 4** beschäftigt sich mit dem entwickelten proaktiven Ansatz und stellt wesentliche Schritte/Komponenten vor. Abschließend fasst **Kapitel 5** den Beitrag zusammen.

2. VERWANDTE ARBEITEN

Das Ziel des im Beitrag vorgestellten proaktiven Ansatzes zur Performance-Analyse und -Vorhersage von Datenbankanwendungen ist die **frühzeitige Erkennung von potentiellen Performance-Problemen** auf Basis einer möglichst effizienten, leicht verständlichen Methodik. Dies verfolgt auch der Ansatz von [2], dessen Grundprinzip – Informationen über Daten und Datenzugriffe, die aus der Anforderungsanalyse einer Anwendung bekannt sind, zur frühzeitigen Optimierung zu nutzen – sich auch im vorliegenden Beitrag wiederfindet. Dabei gelangt [2] durch eine eigene, dem Datenbank-Optimierer nachempfundene Logik und dem verwendeten Modell des offenen Warteschlangennetzwerks frühzeitig zu Kostenabschätzungen bezüglich der Datenbank-Performance. Das in Kapitel 4 des vorliegenden Beitrags vorgestellte Konzept nutzt dagegen synthetisch erzeugte Statistiken und datenbankinterne EXPLAIN-Mechanismen, um eine kostenmäßige Performance-Abschätzung zu erhalten. Damit berücksichtigt es stets sowohl aktuelle als auch zukünftige Spezifika einzelner Datenbank-Optimierer und bleibt entgegen [2] von deren interner Berechnungslogik unabhängig. Ein weiterer Unterschied zwischen beiden Ansätzen besteht in der Präsentation der Analyseergebnisse. Während sich [2] auf tabellarische Darstellungen beschränkt, nutzt das im Beitrag vorstellte Konzept eine auf der Grundlage der Unified Modeling Language (UML) visualisierte Darstellungsform.

Ähnlich wie [2] basieren auch weitere **Ansätze zur Performance-Analyse und -Evaluation** auf dem Modell des Warteschlangennetzwerks. Ein Überblick dazu findet sich in [3]. Demnach zeigt sich für all diese Konzepte ein eher wissenschaftlicher Fokus und eine damit einhergehende weitgehend unerprobte Übertragbarkeit auf die Praxis. So fehlen Studien zur Integration in praxisnahe (Entwicklungs-)Prozesse, zur Benutzerfreundlichkeit sowie zum Kosten-Nutzen-Verhältnis der notwendigen Maßnahmen. Ein damit einhergehendes Defizit ist zusätzlich der mangelnde Toolsupport. Das in Kapitel 4 vorgestellte Konzept verfolgt diesbezüglich einen davon abweichenden Ansatz. Es baut direkt auf etablierten modellbasierten Praxisabläufen bei der Entwicklung von Datenbankanwendungen auf (vgl. Kapitel 3). Insbesondere durch die Verwendung von standardisierten UML-Erweiterungsmechanismen integriert es sich auch Tool-seitig nahtlos in bestehende UML-unterstützende Infrastrukturen.

Die Methodik der **synthetischen Statistiken** – also dem künstlichen Erstellen sowie Manipulieren von Datenbank-Statistiken – ist neben dem in Kapitel 4 vorgestellten Ansatz wesentlicher Bestandteil von [4]. Sie wird zum einen verwendet, um Statistiken aus Produktionsumgebungen in eine Testumgebung zu transferieren. Zum anderen sieht der Ansatz aber auch die gezielte manuelle Veränderung der Statistiken vor, um mögliche dadurch entstehende Änderungen in den Ausführungsplänen und den zu deren Abarbeitung benötigten Kosten mithilfe anschließender EXPLAIN-Analysen feststellen zu können. Dies kann beispielsweise bezogen auf Statistiken zur Datenmenge dafür genutzt werden, um Zugriffe auf eine (noch) kleine Tabelle mit wenigen Datensätzen bereits so zu simulieren, als ob diese eine enorme Menge an Daten umfasst. Weitere Einbettungen in den Entwicklungsprozess von Datenbankanwendungen sieht [4] gegenüber dem hier vorgestellten Ansatz allerdings nicht vor.

Ein weiterer Ansatzpunkt zur Performance-Analyse und -Optimierung existiert im Konzept des **autonomen Datenbank-Tunings** [5][6],[7] – also dem fortlaufenden Optimieren des physischen Designs von bereits bestehenden Datenbanken durch das DBMS selbst. Ein autonomes System erkennt anhand von erlerntem Wissen potentielle Probleme und leitet passende Optimierungsmaßnahmen ein, bevor sich daraus negative Auswirkungen ergeben. Dazu zählt beispielsweise die autonome Durchführung einer Reorganisierung von Daten, um fortwährend steigenden Zugriffszeiten entgegenzuwirken. Ähnlich können auch die mittlerweile je System vielseitig vorhandenen Tuning-Advisor wie beispielsweise [8] und [9] angesehen werden, die zwar nicht automatisch optimierend ins System eingreifen, dem Administrator aber Empfehlungen zu sinnvoll durchzuführenden Aktionen geben. Sowohl das autonome Tuning als auch die Tuning-Advisor sind nicht als Alternative zu dem im vorliegenden Beitrag vorgestellten Ansatz einzuordnen. Vielmehr können sich diese Konzepte ergänzen, indem die Anwendungsentwicklung auf Basis des in Kapitel 4 vorgestellten Konzepts erfolgt und für die spätere Anwendungsadministration/ -evolution verschiedene Tuning-Advisor und die Mechanismen des autonomen Tunings zum Einsatz kommen.

3. ENTWICKLUNGSPROZESS VON DATENBANKANWENDUNGEN

Der Entwicklungsprozess von Anwendungen lässt sich anhand des **System Development Lifecycle (SDLC)** beschreiben und in verschiedene Phasen von der Analyse der Anforderungen bis hin zum Betrieb/zur Wartung der fertigen Software gliedern [1].

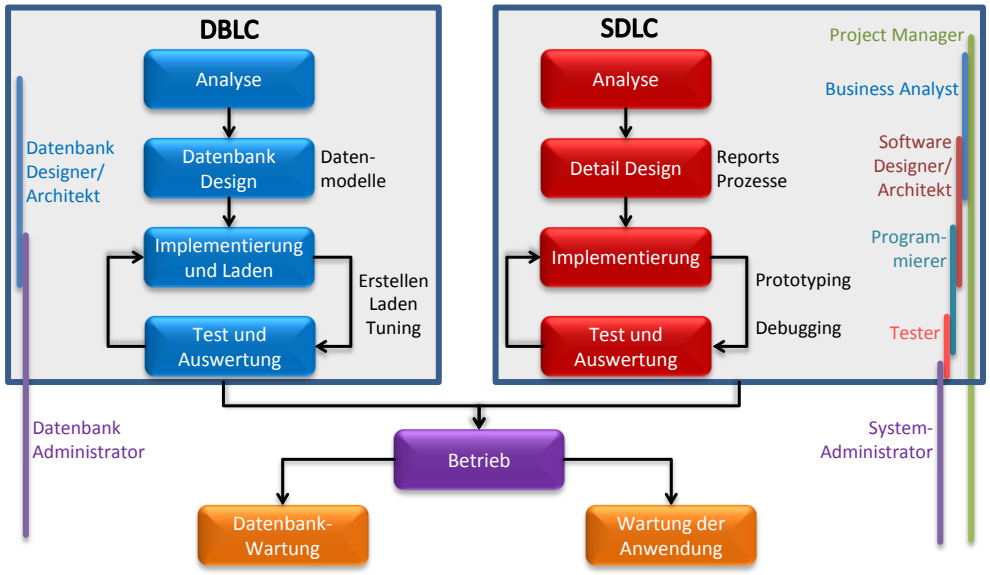

Abbildung 1: Phasen und Akteure im Database und Software Development Lifecycle (DBLC und SDLC)

Zusätzlich zur reinen Anwendungsentwicklung sind weitere Abläufe zur Planung und Bereitstellung einer geeigneten Infrastruktur nötig. Für Datenbankanwendungen wäre das unter anderem der Entwicklungsprozess der Datenbank, welcher sich nach [1] ebenfalls durch ein dem SDLC ähnliches Modell – dem Database Lifecycle (DBLC) – formalisieren lässt. Beide Entwicklungsprozesse verlaufen zeitlich parallel und werden insbesondere in größeren Unternehmen/Projekten durch verschiedene Akteure realisiert. Auf Grundlage von [1] liefert *Abbildung 1* eine Übersicht dazu. Sie visualisiert parallel ablaufende Entwicklungsphasen und eine Auswahl an zuständigen Akteuren, deren konkrete Zusammensetzung/Aufgabenverteilung aber stark abhängig von der Projektgröße und dem Projektteam ist. Wichtig sind hier besonders zwei Erkenntnisse. Zum einen finden ähnliche Entwicklungsprozesse bei Anwendung und Datenbank parallel statt – in etwa das Anwendungsdesign und das Datenbankdesign. Zum anderen können sehr viele Akteure am gesamten Entwicklungsprozess beteiligt sein, sodass Designer, Programmierer, Tester und Administratoren in der Regel disjunkte Personenkreise bilden.

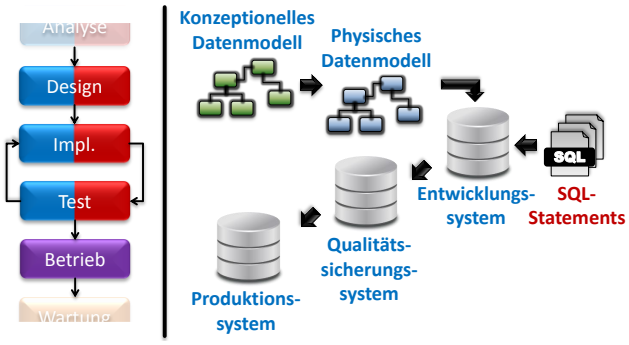

Abbildung 2: Performance-relevante Entwicklungsschritte

Aus dem Blickwinkel der Datenbank-Performance und der darauf einwirkenden bereits genannten Einflussfaktoren reduziert sich der Entwicklungsprozess von Datenbankanwendungen auf die in *Abbildung 2* visualisierten Aufgaben. Anhand der **analysierten Anforderungen** wird im **Datenbank-Design** ein konzeptionelles Datenmodell entwickelt, das anschließend hin zum physischen Datenmodell verfeinert wird. Da sich der Beitrag auf die in der Praxis vorherrschenden relationalen DBMS beschränkt, wird auf das in der Theorie gebräuchliche Zwischenprodukt des logischen Datenmodells (relationale Abbildung) verzichtet.

Nachdem die Design-Phase abgeschlossen ist, beginnt die **Implementierung**. Datenbankseitig wird dabei das physische Datenmodell mittels Data Definition Language (DDL) in ein Datenbankschema innerhalb eines installierten und geeignet konfigurierten DBMS umgesetzt und möglicherweise vorhandene Testdaten geladen. Anwendungsseitig erfolgt parallel dazu die Entwicklung von SQL-Statements zum Zugriff auf die Datenbank sowie die Implementierung der Anwendung selbst. Nach Fertigstellung einzelner Module finden mithilfe des Entwicklungs- und Qualitätssicherungssystems **kontinuierliche Tests** statt, die sich allerdings anfangs auf die Prüfung funktionaler Korrektheit beschränken. Performance-Untersuchungen, insbesondere bezogen auf die Datenbankzugriffe, erfolgen in der Regel erst gezielt zum Abschluss der Implementierungsphase mittels aufwändig vorzubereitender und im Qualitätssicherungssystem durchzuführender Lasttests.

Die Folgen aus diesem Vorgehen für die Erkennung und Behandlung von Performance-Problemen sind mitunter gravierend. Engpässe werden erst spät (im **Betrieb**) bemerkt und sind aufgrund des fortgeschrittenen Entwicklungsprozesses nur mit hohem Aufwand zu korrigieren. Basieren sie gar auf unvorteilhaften Design-Entscheidungen beispielsweise bezogen auf die Datenmodellierung, ist eine nachträgliche Korrektur aufgrund zahlreicher Abhängigkeiten (Anwendungslogik, SQL-Statements, Testdatenbestände, etc.), getrennten Zuständigkeiten und in der Regel engen Projektzeitplänen nahezu ausgeschlossen. Erfahrungen aus dem Arbeitsumfeld des Autors haben dies wiederholt bestätigt.

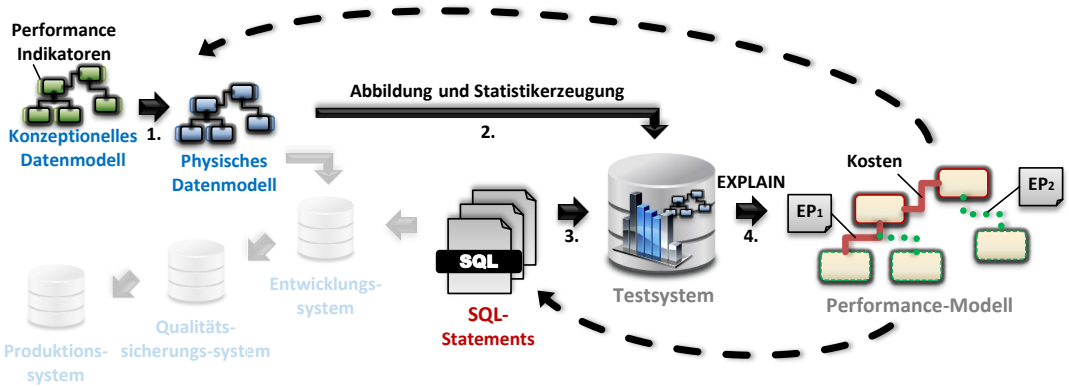

Abbildung 3: Ansatz zur proaktiven modellbasierten Performance-Analyse und -Vorhersage

4. PROAKTIVE MODELLBASIERTE PERFORMANCE-ANALYSE

Alternativ zur Performance-Analyse mittels Lasttests (vgl. Kapitel 3) bieten sich zur Kontrolle der SQL-Performance die eingangs erwähnten **EXPLAIN-Mechanismen** an. Mit deren Hilfe lassen sich bei vorliegendem physischen Datenbank-Design (inklusive Indexe, etc.) bereits in frühen Abschnitten der Implementierungsphase Auswertungen zu Ausführungsplänen und geschätzten Kosten für entwickelte SQL-Statements durchführen. Auf diese Weise gewonnene Erkenntnisse können vom Designer/Programmierer direkt genutzt werden, um Optimierungen in Hinblick auf die quasi grade entworfenen/implementierten SQL-Statements durchzuführen. Durch die gegebene zeitliche Nähe zum Anwendungs- und Datenbank-Design sind auch Performance-Optimierungen auf Basis von Datenmodellanpassungen (Normalisierung/Denormalisierung) ohne größeren Aufwand möglich.

Das beschriebene Vorgehen hat zwar den Vorteil, dass mögliche Performance-Probleme schon von den Akteuren (Designer/Programmierer) erkannt werden können, die diese durch Design-Änderungen am effektivsten zu lösen wissen. Demgegenüber erfordern die EXPLAIN-Analysen und das Verständnis der Ausführungspläne einen **Grad an Expertise**, den Designer/Programmierer in der Regel nicht besitzen. Ein Datenbank Administrator (DBA), der über diese verfügt, ist wiederum von den fachlichen Anforderungen zu distanziert, sodass er zwar mögliche Performance-Ausreißer erkennen, nicht aber fachlich bewerten kann. Führt eine Anwendung beispielsweise einmal monatlich eine sehr komplexe Auswertung mithilfe eines entsprechend Laufzeit-intensiven SQL-Statements durch, dann würde dem DBA diese Abfrage bei EXPLAIN-Analysen als kritisch erscheinen. Denn er weiß weder, dass damit ein fachlich aufwändiger Prozess durchgeführt wird, noch dass es sich dabei um eine einmalig pro Monat auszuführende Abfrage handelt. Um sich als DBA in einer Infrastruktur von nicht selten mehr als 100 unterschiedlichen Anwendungen über die fachlichen Anforderungen und speziellen Prozesse jeder einzelnen im Detail zu informieren beziehungsweise um sich als Designer/Programmierer das nötige Knowhow zur Ausführungsplanbewertung aufzubauen, ist personelle Kapazität vonnöten, die in der Regel nicht verfügbar ist.

Ein anderes Problem, dass sich in Zusammenhang mit frühzeitigen EXPLAIN-Analysen zeigt, begründet sich in dem dritten zuvor genannten Performance-Faktor: den Daten. Während diese bei Anwendungsweiterentwicklungen weitgehend vorliegen, existieren für neu zu entwickelnde Anwendungen im Normalfall keine **repräsentativen Datenbestände**. Somit fehlen auch geeignete Datenbankstatistiken zur Grundlage für die EXPLAIN-Auswertungen. Die Folge sind Ausführungspläne und Kostenabschätzungen, die mit denen eines späteren produktiven Einsatzes der Statements oftmals nur wenig gemeinsam haben und für eine proaktive Performance-Analyse somit (nahezu) unverwertbar sind.

Der im folgenden Kapitel vorgestellte proaktive modellbasierte Ansatz zur Performance-Analyse und -Vorhersage greift beide Probleme auf: die fehlende repräsentative Datenbasis für Datenbankstatistiken und die mangelnde Expertise zur Ausführungsplanbewertung durch Designer/Programmierer. Dabei sieht dieser Ansatz zur Bereitstellung geeigneter Datenbankstatistiken ein synthetisches Erzeugen anhand von Performance-Indikatoren vor. Das Problem der mangelnden Expertise wird durch eine einfache modellbasierte Darstellung von gewonnenen EXPLAIN-Ergebnissen adressiert. Wie diese gestaltet ist und mit den Performance-Indikatoren zusammenwirkt verdeutlichen die weiteren Ausführungen des Kapitels anhand *Abbildung 3*.

4.1 Performance-Indikatoren im Datenmodell

Als Performance-Indikatoren bezeichnet die vorliegende Arbeit ausgewählte **Metadaten zu Entitäten und deren Attributen** (beziehungsweise zu Tabellen und deren Spalten), die Aufschluss über die erwarteten realen Datenbestände geben und in Zusammenhang mit dem Datenbank-Design und der Infrastruktur erste Rückschlüsse auf die zukünftige Datenbank-Performance erlauben. Dazu zählen Informationen zu den erwarteten **Datenmengen** wie in etwa die erwartete Anzahl an Zeilen pro Tabelle und Kennzahlen zur **Datenverteilung** – beispielsweise in Form von Wertebereichsangaben, Einzelwertwahrscheinlichkeiten oder der Kardinalität pro Spalte. Viele dieser Informationen sind Teil des Ergebnisses der Anforderungsanalyse und somit frühzeitig im SDLC bekannt und vom Business Analyst erfasst worden. Dabei reicht die Dokumentation von rein textuellen Beschreibungen bis hin zu tief strukturierten Darstellungen. Eine einheitlich standardisierte Form zur Erfassung von Performance-Indikatoren im DBLC existiert jedoch bislang nicht, wodurch die Metadaten kaum bis gar nicht in den weiteren Entwicklungsprozess einfließen.

In der Praxis basiert die **Datenmodellierung** ähnlich wie weite Teile der Anwendungsmodellierung auf der Sprache UML. Dabei

wurde diese ursprünglich nicht zur Abbildung von Datenstrukturen im Sinn einer Entity-Relationship-Modellierung konzipiert, sodass die Verbindung beider Welten – und damit die Modellierung von Anwendung und Datenstrukturen mithilfe einer gemeinsamen Sprache in einem gemeinsamen Tool – erst durch Ansätze wie [10] oder auch den Entwurf zum IMM Standard der OMG [11] geschaffen wurde. Die Voraussetzung dafür bildet jeweils die UML-Profil-Spezifikation, die es ermöglicht, bestehende UML-Objekte über Neu-Stereotypisierungen zu erweitern.

Um die zuvor genannten Performance-Indikatoren für den weiteren Entwicklungsprozess nutzbar zu machen und sie innerhalb bestehender Infrastrukturen/Tool-Landschaften standardisiert zu erfassen, kann ebenfalls der UML-Profil-Mechanismus genutzt werden. So ließe sich beispielsweise mithilfe eines geeigneten Profils wie in *Abbildung 3* in 1. schematisch angedeutet aus einem UML-Objekt „entity“ ein neues Objekt „entity_extended“ ableiten, das in einem zusätzlichen Merkmal „cardinality“ Informationen über die produktiv erwartete Datenmenge zu einer Entität/Tabelle aufnehmen kann.

4.2 Synthetische Datenbankstatistiken

Eines der eingangs aufgezeigten Hindernisse für proaktive Performance-Analysen beziehungsweise -Vorhersagen bestand in der **fehlenden repräsentativen Datenbasis** für Datenbank-Statistiken. Diese Statistiken werden im Normalfall vom DBMS anhand der gespeicherten Daten selbst gesammelt. Dem entgegen verfolgt das hier vorgestellte Konzept den Ansatz, dem DBMS Statistiken vorzugeben, ohne dazu datenbankseitig repräsentative Datenbestände vorhalten zu müssen. Dafür bieten zwar die wenigsten DBMS vordefinierte Schnittstellen an, allerdings sind sämtliche Statistik-Informationen in der Regel innerhalb DBMS-interner manipulierbarer Tabellen gespeichert, wie dies beispielswiese auch bei DB2 oder Oracle der Fall ist [12].

Datenbankstatistiken enthalten Informationen über **Datenmengen und Datenverteilungen** sowie Kennzahlen zur physischen Speicherung wie beispielsweise die Anzahl der verwendeten Datenbankseiten pro Tabelle. Während erstere inhaltlich den zuvor beschriebenen Performance-Indikatoren entsprechen, sind die Statistikdaten zur physischen Speicherung interne DBMS-abhängige Größen. Mithilfe geeigneter, von den DBMS-Herstellern zur Unterstützung beim Datenbank-Design bereitgestellter Abschätzungsvorschriften lassen sich aber auch diese Kennzahlen auf Grundlage der Performance-Indikatoren approximieren. Somit ist es wie in *Abbildung 3* in 2. gezeigt möglich, anhand geeignet formalisierter Performance-Indikatoren frühzeitig im SDLC/ DBLC repräsentative Datenbankstatistiken künstlich zu erzeugen.

4.3 EXPLAIN und Performance-Modell

Auf Grundlage von synthetischen Datenbankstatistiken können wie in *Abbildung 3* in 3. und 4. zu sehen, mittels der vom DBMS bereitgestellten **EXPLAIN-Funktionalität**, der SQL-Workload und dem aus dem physischen Datenmodell ableitbaren Datenbankschema proaktive Performance-Vorhersagen durchgeführt werden. Die resultierenden, teils komplexen Ausführungspläne lassen sich allerdings nur mit ausreichend Expertise und vorhandenen personellen Kapazitäten angemessen auswerten, sodass diese Problematik vorläufig weiterbesteht. Eine Hauptursache, die das Verständnis von Ausführungsplänen erschwert, ist ihre hierarchische Darstellung als Zugriffsbaum. Demgegenüber denkt der Designer/Programmierer beim Modellieren oder dem Entwickeln von SQL-Statements auf relationale Weise. Die im vorliegenden Ansatz als Performance-Modell bezeichnete vereinfachte Präsentation von Ausführungsplänen versucht, diese Diskrepanz aufzulösen.

Das **Performance-Modell** basiert auf dem physischen Datenmodell und damit auf einer dem Designer/Programmierer bekannten Darstellungsform. Zusätzlich umfasst es die für diesen Personenkreis wesentlichen Informationen aus den EXPLAIN-Ergebnissen. Dazu zählen die vom DBMS abgeschätzten Kosten für die Ausführung des gesamten Statements sowie wichtiger Operatoren wie Tabellen- beziehungsweise Indexzugriffe oder Tabellenverknüpfungen mittels Join – jeweils skaliert um die erwartete Ausführungshäufigkeit des Statements. Weitere Detailinformationen innerhalb der Ausführungspläne wie beispielsweise die konkrete Abarbeitungsreihenfolge einzelner Operatoren oder Angaben zu abgeschätzten Prädikat-Selektivitäten werden vom Modell zum Zweck der Einfachheit und Verständlichkeit bewusst vernachlässigt. Für die gleichzeitige Analyse mehrerer Statements erfolgt eine Aggregation der jeweils abgeschätzten Kosten auf Objektebene.

Zentrale Komponente im Performance-Modell ist eine ebenfalls dem physischen Datenmodell angelehnte **Diagrammdarstellung**. Mithilfe farblicher Hervorhebung und geeigneter Bewertungsmetriken sollen sämtliche Objekte gemäß den vom DBMS geschätzten Zugriffskosten zur Abarbeitung der Workload klassifiziert und visualisiert werden. Auf diese Weise kann ein Designer/Programmierer frühzeitig Auskunft über aus Performance-Perspektive zu optimierende Bereiche im Datenbankschema beziehungsweise kritische, alternativ zu konzipierende SQL-Statements erhalten. *Abbildung 3* veranschaulicht exemplarisch ein visualisiertes Performance-Modell für zwei Statements/ Ausführungspläne (EP). Während der untere Bereich weitgehend grün/unkritisch markiert ist, befinden sich im oberen Diagrammteil mögliche Performance-kritische rot gekennzeichnete Zugriffe, die es gezielt zu untersuchen und an geeigneter Stelle (SQL-Statement, Datenbank-Design) zu optimieren gilt (vgl. gestrichelte Pfeile in *Abbildung 3*).

Die **technische Realisierung** des Performance-Modells sowie der dazugehörigen Diagrammdarstellung erfolgt analog zur Erfassung der Performance-Indikatoren über den UML-Profil-Mechanismus, wodurch auch in diesem Punkt die Kompatibilität des vorgestellten Ansatzes zu bestehenden Tool-Infrastrukturen gewährleistet ist.

4.4 Ablauf einer Analyse/Vorhersage

Für den Designer/Programmierer sieht der in *Abbildung 3* vorgestellte proaktive Ansatz folgende Vorgehensweise vor. Nachdem nach 1. ein Datenbank-Design-Entwurf fertiggestellt ist, initiiert er in 2. einen Automatismus zur Abbildung des Designs in ein Datenbank-Schema sowie zur Erstellung von synthetischen Datenbank-Statistiken anhand der von ihm modellierten Performance-Indikatoren. Mithilfe einer weiteren Routine startet der Designer/Programmierer in 3. und 4. anschließend einen Simulationsprozess, der auf Basis der EXPLAIN-Mechanismen Performance-Vorhersagen für eine gegebene Workload erstellt und diese als Performance-Modell aufbereitet. Von dort aus informiert er sich mithilfe der Diagrammdarstellung über mögliche kritische Zugriffe, die er daraufhin gezielt analysiert und optimiert.

5. ZUSAMMENFASSUNG

Datenbank-Performance ist ein wichtiger, oftmals jedoch vernachlässigter Faktor in der Anwendungsentwicklung. Durch moderne Anforderungen und dazu implementierte Anwendungen sehen sich speziell deren Datenbank-Backends mit kontinuierlich wachsenden Herausforderungen insbesondere betreffend der Performance konfrontiert. Diese können nur bewältigt werden, wenn das Thema Datenbank-Performance intensiver betrachtet und durch proaktive Analysen (beispielsweise mittels EXPLAIN-Mechanismen) kontinuierlich verfolgt wird. Doch auch dann sind einzelne Hindernisse unvermeidlich: fehlende repräsentative Daten(-mengen) und Expertise/Kapazitäten zur Analyse.

Der vorliegende Beitrag präsentiert zur Lösung dieser Probleme einen modellbasierten Ansatz, der auf Basis synthetisch erzeugter Statistiken proaktive Performance-Analysen sowie -Vorhersagen erlaubt und die daraus gewonnenen Ergebnisse in einer einfach verständlichen Form visualisiert. Die technologische Grundlage dafür bietet die in der Praxis vorherrschende Modellierungssprache UML mit ihrer UML-Profil-Spezifikation. Sie erlaubt es das hier vorgestellte Konzept und die dazu benötigten Komponenten mit vorhandenen technischen Mitteln abzubilden und nahtlos in bestehende UML-Infrastrukturen zu integrieren.

6. AUSBLICK

Bei dem im Beitrag vorgestellten Konzept handelt es sich um einen auf Basis wiederkehrender praktischer Problemstellungen und den daraus gewonnenen Erfahrungen konstruierten Ansatz. Während die technische Umsetzbarkeit einzelner Teilaspekte wie etwa die Erfassung von Performance-Indikatoren oder die Konstruktion des Performance-Modells auf Basis von UML-Profilen bereits geprüft wurde, steht eine prototypische Implementierung des gesamten Prozesses zur Performance-Analyse noch aus.

Zuvor sind weitere Detailbetrachtungen nötig. So ist beispielsweise zu klären, in welchem Umfang **Performance-Indikatoren** im Datenmodell vom Analyst/Designer sinnvoll erfasst werden sollten. Dabei ist ein Kompromiss zwischen maximalem Detailgrad und minimal nötigem Informationsgehalt anzustreben, sodass der Aufwand zur Angabe von Performance-Indikatoren möglichst gering ist, mit deren Hilfe aber dennoch eine repräsentative Performance-Vorhersage ermöglicht wird.

Weiterhin gilt es, eine geeignete **Metrik zur Bewertung/Kategorisierung der Analyseergebnisse** zu entwickeln. Hier steht die Frage im Vordergrund, wann ein Zugriff anhand seiner Kosten als schlecht und wann er als gut zu bewerten ist. Ein teurer Zugriff ist nicht zwangsweise ein schlechter, wenn er beispielsweise zur Realisierung einer komplexen Funktionalität verwendet wird.

Zuletzt sei noch die Erfassung beziehungsweise Beschaffung der für die EXPLAIN-Analysen notwendigen **Workload** erwähnt. Diese muss dem vorgestellten proaktiven Analyseprozess zugänglich gemacht werden, um anhand des beschriebenen Konzepts frühzeitige Performance-Untersuchungen durchführen zu können. Im einfachsten Fall könnte angenommen werden, dass sämtliche SQL-Statements (inklusive ihrer Ausführungshäufigkeit) vom Designer/Programmierer ebenfalls im Datenmodell beispielsweise als zusätzliche Merkmale von Methoden in der UML-Klassenmodellierung zu erfassen und kontinuierlich zu pflegen wären. Dies wäre jedoch ein sehr aufwändiges Verfahren, das der gewünschten hohen Praxistauglichkeit des proaktiven Ansatzes entgegensteht. Somit sind alternative Varianten zur Beschaffung der Workload für den Analyseprozess zu untersuchen und abzuwägen.

7. LITERATUR

[1] C. Coronel, S. Morris, P. Rob. *Database Systems: Design, Implementation, and Management*, Course Technology, 10. Auflage, 2011.

[2] S. Salza, M. Renzetti. *A Modeling Tool for Workload Analysis and Performance Tuning of Parallel Database Applications*, Proceedings in ADBIS'97, 09.1997 http://www.bcs.org/upload/pdf/ewic_ad97_paper38.pdf

[3] R. Osman, W. J. Knottenbelt. *Database system performance evaluation models: A survey*, Artikel in Performance Evaluation, Elsevier Verlag, 10.2012 http://dx.doi.org/10.1016/j.peva.2012.05.006

[4] Tata Consultancy Services. *System and method for SQL performance assurance services*, Internationales Patent PCT/IN2011/000348, 11.2011 http://dx.doi.org/10.1016/j.peva.2012.05.006

[5] D. Wiese. Gewinnung, Verwaltung und Anwendung von Performance-Daten zur Unterstützung des autonomen Datenbank-Tuning, Dissertation, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, 05.2011. http://www.informatik.uni-jena.de/dbis/alumni/wiese/pubs/Dissertation__David_Wiese.pdf

[6] S. Chaudhuri, V. Narasayya. *A Self-Tuning Database Systems: A Decade of Progress*, Proceedings in VLDB'07, 09.2007 http://research.microsoft.com/pubs/76506/vldb07-10yr.pdf

[7] N. Bruno, S. Chaudhuri. *An Online Approach to Physical Design Tuning*, Proceedings in ICDE'07, 04.2007 http://research.microsoft.com/pubs/74112/continuous.pdf

[8] Oracle Corporation. *Oracle Database 2 Day DBA 12c Release 1 (12.1) – Monitoring and Tuning the Database*, 2013. http://docs.oracle.com/cd/E16655_01/server.121/e17643/montune.htm#ADMQS103

[9] Microsoft Corporation. *SQL Server 2005 – Database Engine Tuning Advisor (DTA) in SQL Server 2005*, Technischer Artikel, 2006. http://download.microsoft.com/download/4/7/a/47a548b9-249e-484c-abd7-29f31282b04d/SQL2005DTA.doc

[10] C.-M. Lo. A Study of Applying a Model-Driven Approach to the Development of Database Applications, Dissertation, Department of Information Management, National Taiwan University of Science and Technology, 06.2012.

[11] Object Management Group. *Information Management Metamodel (IMM) Specification Draft Version 8.0*, Spezifikationsentwurf, 03.2009. http://www.omgwiki.org/imm/doku.php

[12] N. Burgold, M. Gerstmann, F. Leis. *Statistiken in relationalen DBMSen und Möglichkeiten zu deren synthetischer Erzeugung*, Projektarbeit, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, 05.2014.