

Systematische Rasterfahndung nach Performance-Antipattern



Otto Group Solution Provider Dresden GmbH

www.osp.de



Gründung:

März 1991

Muttergesellschaft:

OTTO Group

Standorte:

Dresden, Hamburg, Burgkunstadt, Bangkok, Taipeh

Mitarbeiterzahl:

Ca. 220

Geschäftsführer:

Dr. Stefan Borsutzky, Alexander Hauser

zur Person



Peter Ramm

Teamleiter strategisch-technische Beratung bei OSP Dresden

> 20 Jahre Historie in IT-Projekten

Schwerpunkte:

- Entwicklung von OLTP-Systemen auf Basis von Oracle-Datenbanken
- Architektur-Beratung bis Trouble-Shooting
- Performance-Optimierung bestehender Systeme

Varianten der Performance-Optimierung

Ereignis-getrieben

- Aufwand zur Optimierung dann, wenn produktive Notwendigkeit der Verbesserung
- Hoher Effekt der einzelnen Maßnahmen
- Potential zur Optimierung wird nur punktuell genutzt
- Evtl. unnötig hohe Kosten für Hardware-Ressourcen und Lizenzen
- Geringere Kosten für Performance-Optimierung

Vorbeugend

- Permanente Optimierung des produktiven Systems
- Sukzessive geringerer Effekt der einzelnen Maßnahmen
- Potential zur Optimierung wird weitgehend genutzt
- Effektive Nutzung der Hardware-Ressourcen, optimale Lizenzkosten
- Zunehmend höhere Aufwände für weitere Performance-Optimierung

Systematische Rasterfahndung mit Panorama

Motivation

- Erkennung und Bewertung aller weiteren Vorkommen einer einmal analysierten Problemstellung im System, gewichtet nach Relevanz / Potential
- Möglichst einfache Umsetzbarkeit der Lösungsvorschläge ohne Eingriff in die Architektur und Design
- Fixen erkannter leicht lösbarer Problemstellungen systemweit statt step by step nach Eskalation
- Komfortable Einbettung in weiteren Analyse-Workflow

Lösungsansatz

- Formulierung der Problem-Identifikation als SQL-Statement
- Nutzung des Dictionaries, der dynamischen SGA-Views sowie der AWR-Historiendaten der DB
- Knapp 100 unterschiedliche bewertete Aspekte, von Potential in Datenstrukturen und Indizierung bis zu suboptimalen SQL-Statements
- Limitierung: Adressiert nur die Themen, mit denen ich persönlich in Projekten konfrontiert wurde

Folgeschritte der Analyse

Panorama bietet Funktionen sowohl für die unmittelbare Selektion der Lösungsvorschläge als auch für die Umfeldbetrachtung zur individuellen Bewertung der Relevanz

- Die Rasterfahndungs-Selektionen sind kein automatisierter ToDo-Listen-Generator
- Zwingend notwendig ist die fachkundige Bewertung der Vorschläge sowie die Erkennung und der Ausschluss von ermittelten Vorschlägen ohne konkrete Relevanz einer Umsetzung
- Für die einfache Bewertung der Vorschläge sind SQL-IDs und DB-Objekte verlinkt und erlauben u.a. :
 - Detaillierte Struktur-Info zu angemerkt DB-Objekten
 - Verwendung von DB-Objekten in SQL-Statements aus aktueller SGA und AWR-Historie
 - Analyse der Ausführungspläne der genannten SQLs
 - Analyse der Active-Session-History und Historie der SQLs
 - Diverseste weitere Aspekte

Anwendung der Suchen: Tool-basiert

- In meinem Analyse-Tool „Panorama“ sind die Rasterfahndungs-Funktionen zu finden im Menü „Spez. Erweiterungen“ / „Rasterfahndung“
- Panorama ist frei verfügbar unter: <https://rammpeter.github.io/panorama.html>

Rasterfahndung nach Performance-Bottlenecks und Nutzung von Anti-Pattern

Auswahl des Rasterfahndungs-SQL

Filter: Include description Search

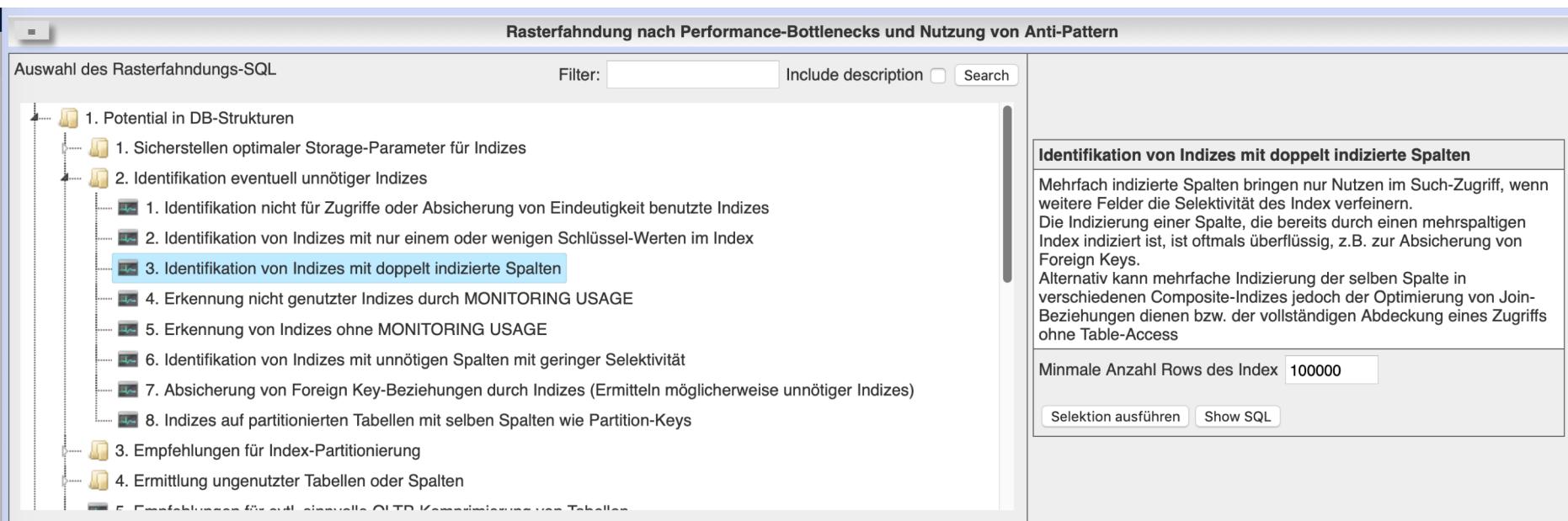
1. Potential in DB-Strukturen

- 1. Sicherstellen optimaler Storage-Parameter für Indizes
- 2. Identifikation eventuell unnötiger Indizes
 - 1. Identifikation nicht für Zugriffe oder Absicherung von Eindeutigkeit benutzte Indizes
 - 2. Identifikation von Indizes mit nur einem oder wenigen Schlüssel-Werten im Index
 - 3. Identifikation von Indizes mit doppelt indizierte Spalten**
 - 4. Erkennung nicht genutzter Indizes durch MONITORING USAGE
 - 5. Erkennung von Indizes ohne MONITORING USAGE
 - 6. Identifikation von Indizes mit unnötigen Spalten mit geringer Selektivität
 - 7. Absicherung von Foreign Key-Beziehungen durch Indizes (Ermitteln möglicherweise unnötiger Indizes)
 - 8. Indizes auf partitionierten Tabellen mit selben Spalten wie Partition-Keys
- 3. Empfehlungen für Index-Partitionierung
- 4. Ermittlung ungenutzter Tabellen oder Spalten

Identifikation von Indizes mit doppelt indizierte Spalten

Mehrfach indizierte Spalten bringen nur Nutzen im Such-Zugriff, wenn weitere Felder die Selektivität des Index verfeinern. Die Indizierung einer Spalte, die bereits durch einen mehrspaltigen Index indiziert ist, ist oftmals überflüssig, z.B. zur Absicherung von Foreign Keys. Alternativ kann mehrfache Indizierung der selben Spalte in verschiedenen Composite-Indizes jedoch der Optimierung von Join-Beziehungen dienen bzw. der vollständigen Abdeckung eines Zugriffs ohne Table-Access

Minmale Anzahl Rows des Index



Zugriff auf die komplette Liste der SQLs

Des weiteren findet sich die komplette Liste der auch in Panorama verfügbaren Selektionen unter: https://rammpeter.github.io/oracle_performance_tuning.html.

Oracle performance tuning

Systematic dragnet investigation for performance bottlenecks and usage of performance anti-pattern

Here I'll provide you a set of nearly 100 SQL-statements.

Each SQL considers a special aspect of performance anti-pattern or pitfalls. It selects for you a list of occurrences and gives you recommendations to possibly fix this issue.

All lists are sorted by relevance so you can start your consideration at top.

Be aware of your own brain. All these findings are recommendations for your personal consideration only. May be they are dramatically worth to develop activities for fixing. May also be you consider them as: Exactly this way my application should work.

If you plan to execute this SQL against your database:

Try my application "Panorama", available at this site. Panorama contains all these SQLs inside and it's much more convenient to execute them in Panorama and instantly drill down more from result into deep.

- 1. Potential in DB-structures
 - 1. Ensure optimal storage parameter for indexes
 - 2. Detection of possibly unnecessary indexes
 - 1. Detection of indexes not used for access or ensurance of uniqueness
 - 2. Detection of indexes with only one or little key values in index
 - 3. Detection of indexes with multiple indexed columns
 - 4. Detection of unused indexes by system monitoring
 - 5. Detection of indexes with unnecessary columns because of pure selectivity

Detection of indexes with multiple indexed columns

Multiple indexed columns are useful for data access only if additional index-columns improve selectivity of index.

Indexing on column that is already indexed as first column of another multi-column index is often unnecessary, e.g. for coverage of foreign key.

Otherwise multiple indexing same column in different composite indexes may be used for optimization of joins or for access on table data without accessing table itself.

```
SELECT /* DB-Tools Ramm doppelt indizierte Spalten*/ d.* , i.Index_Name , ix.Num_Rows ,  
(  
    SELECT Constraint_Name  
    FROM DBA_Constraints c  
    WHERE c.Table_Name = d.Table_Name  
    AND c.Owner = ix.Owner  
    AND c.Index_Name = ix.Index_Name  
    AND c.Constraint_Type='P'  
) UsedForPkey ,  
(SELECT SUM(bytes)/(1024*1024) MBytes  
FROM DBA_SEGMENTS s  
WHERE s.SEGMENT_NAME = ix.Index_Name  
AND s.Owner = ix.Owner
```

Beispiel „Potential in DB-Strukturen“ Punkt 1.2.2.

Identifikation eventuell unnötiger Indizes

Identifikation von Indizes mit nur einem oder wenigen Schlüssel-Werten im Index

- Für Indizes mit nur einem oder wenigen Schlüsseln kann die Sinnfrage gestellt werden.
Ausnahme: Index auch mit nur einem Schlüssel kann sinnvoll sein zur Differenzierung zwischen NULL und NOT NULL.
- Indizes mit nur einem Schlüssel und keinen NULLs in indizierten Feldern können i.d.R. entfernt werden.
- Bei Nutzung des Index zur Absicherung von Foreign Keys kann oftmals trotzdem auf den Index verzichtet werden:
 - der resultierende FullTableScan auf der referenzierenden Tabelle bei eventuellem Delete auf der referenzierten Tabelle kann billigend in Kauf genommen werden kann.
 - Gleiches gilt für Lock-Propagierung über ForeignKey-Constraint.
Wenn die referenzierte Tabelle nur wenige Records enthält und selten bis nie DML-Operationen darauf stattfinden, dann stellt das Fehlen eines den shared Lock abfangenden Index kein Problem dar

Beispiel „Potential in DB-Strukturen“ Punkt 1.1.3.

Sicherstellen optimaler Storage-Parameter für Index

Empfehlungen für Index-Komprimierung, Test auf Leaf-Blocks

- Index-Kompression (COMPRESS) bringt sinnvolle Ergebnisse durch Reduktion der physischen Größe für OLTP-Indizes mit geringer Selektivität.
- Bei geringer Selektivität der Indizes ist Reduzierung der Größe durch Komprimierung um 1/4 bis 1/3 möglich.
- Bei komprimiertem Index sollte die Anzahl Leaf-Blocks per Key sinken, im Optimum passen alle Referenzen auf Datenblöcke eines Keys in nur einen Leaf Block.

Beispiel „Potential in DB-Strukturen“ Punkt 1.2.3.

Identifikation eventuell unnötiger Indizes

Identifikation von Indizes mit doppelt indizierten Spalten

- Mehrfach indizierte Spalten bringen nur Nutzen im Such-Zugriff, wenn weitere Felder die Selektivität des Index verfeinern.
- Die Indizierung einer Spalte, die bereits durch einen mehrspaltigen Index indiziert ist, ist oftmals überflüssig, z.B. zur Absicherung von Foreign Keys.
- Alternativ kann mehrfache Indizierung der selben Spalte in verschiedenen Composite-Indizes jedoch der Optimierung von Join-Beziehungen dienen bzw. der vollständigen Abdeckung eines Zugriffs ohne Table-Access

Beispiel „Potential in DB-Strukturen“ Punkt 1.2.4.

Identifikation eventuell unnötiger Indizes

Erkennung nicht genutzter Indizes durch System-Monitoring

- Die DB protokolliert die Nutzung (Zugriff) von Indizes die vorab durch 'ALTER INDEX ... MONITORING USAGE' deklariert wurden.
- Das Ergebnis der Nutzungsprotokollierung ist je Schema in Tabelle v\$Object_Usage einsehbar.
- Schemaübergreifend ist die Nutzung mit dieser Selektion sichtbar.
- Achtung: Auch die Ausführung von GATHER_INDEX_STATS zählt als Nutzung, selbst wenn der Index nie durch andere Selects genutzt wurde.

Beispiel „Potential in DB-Strukturen“ Punkt 1.2.8.

Identifikation eventuell unnötiger Indizes

Indizes auf partitionierten Tabellen mit selben Spalten wie Partition-Keys

- Wenn ein Index auf einer partitionierten Tabelle die selben Spalten indiziert wie die Partition-Keys und die Partitionierung selbst selektiv genug ist durch Partition Pruning, dann kann der Index möglicherweise entfernt werden.

Beispiel „SQLs mit suboptimalen Plan“ Punkt 2.2.8.



Eventuelle Probleme mit parallel query

Statements mit geplanter paralleler Execution forced to serial

- PX COORDINATOR FORCED SERIAL im Execution-Plan zeigt, dass der Optimizer parallele Ausführung annimmt, dann aber Hinderungsgründe für parallele Ausführung feststellt (z.B. stored functions ohne PARALLEL_ENABLE).
- Die Operationen unter der Zeile des Ausführungsplans werden in der Realität nicht parallel ausgeführt auch wenn der Optimizer sie für parallele Ausführung markiert hat!

Beispiel „SQLs mit suboptimalen Plan“ Punkt 2.6.

Identifikation von Statements mit wechselndem Ausführungsplan aus Historie

- Mit dieser Selektion lassen sich aus den AWR-Daten Wechsel der Ausführungspläne unveränderter SQL's ermitteln.
- Betrachtet wird dabei die aufgezeichnete Historie ausgeführter Statements

Beispiel „Potential in DB-Strukturen“ Punkt 1.11.

Möglicherweise aufwändiger TABLE ACCESS BY INDEX ROWID mit zusätzlichen Filterbedingungen nach dem Zugriff auf die Tabelle

- Wenn in einem SQL eine Tabelle zusätzliche Filterbedingungen hat die nicht vom Index abgedeckt werden dann kann überlegt werden, den Index um diese Filterbedingungen zu erweitern.
- Dies sorgt dass der teurere TABLE ACCESS BY ROWID nur ausgeführt wird, wenn die Zeile allen Filterbedingungen entspricht, die durch den Index geprüft werden.
- Diese Abfrage betrachtet die aktuelle SGA.

Einbinden eigener Abfragen

Panorama erlaubt die Einbindung eigener parametrisierbarer Abfragen zusätzlich zu den vorhandenen

- Damit wird auch aus dem Ergebnis eigener Abfragen sofort weiterverlinkt zu SQLs und DB-Objekten
- Angabe der Abfrage in JSON-Notation, Bindevariablen per „?“ + Deklaration der Parameter
- Alternativ Ablage der eigenen Abfragen als File auf Server-Seite, nähere Details unter diesem [Link](#)

The screenshot shows the Panorama software interface. At the top, there is a search bar with the placeholder "Rasterfahndung nach Performance-Bottlenecks und Nutzung von Anti-Pattern". Below the search bar, there is a tree view with two main items: "1. Potential in DB-Strukturen" and "2. Ermittlung von SQL-Statements mit suboptimalem Ausführungsplan". To the right of the tree view, there is a button labeled "Selektion ausführen" and another labeled "Show SQL". Below the search bar, there is a section titled "Erweitern der Rasterfahndungs-SQLs um persönliche Abfragen". This section contains a JSON configuration snippet:

```
{  
    name: "Select table attributes for one table",  
    desc: "Select attributes from DBA_Tables for your selected table",  
    sql: "SELECT * FROM DBA_Tables WHERE Owner = UPPER(?) AND Table_Name = UPPER(?)",  
    parameter: [  
        {  
            name: "Owner",  
            title: "Name of table owner",  
            size: 30,  
            default: "SYS"  
        },  
        {  
            name: "Table name",  
            title: "Name of table",  
            size: 30  
        }  
    ]  
}
```

Letztes Wort



Panorama greift nur rein lesend auf Ihre Datenbank zu und benötigt keine eigenen PL/SQL-Objekte.

Sie können die Funktionen also ohne Risiko testen und verstehen.
Probieren sie es gern aus.

Beschreibung inkl. Download-Link:
Blog zum Thema:

<http://rammpeter.github.io/panorama.html>

<https://rammpeter.wordpress.com/category/panorama-news-and-hints>

Vielen Dank für Ihr Interesse

Otto Group Solution Provider (OSP) Dresden GmbH
Freiberger Str. 35 | 01067 Dresden
T +49 (0)351 49723 0 | F +49 (0)351 49723 119
osp.de