Oracle Performance Tuning
Musterlösung zur Prüfung WS 2001/2002

Alex Schröder

2002-10-12

1 Speicherstrukturen

1. Hier drei Beispiele für eine sinnvolle Aufteilung auf zwei Platten: 1. Temporary Segmente auf eine eigene Platte, 2. Programm und Daten auf separate Platten, 3. Tabellen und Indexe auf separate Platten, 4. RAID mit zwei Platten für Striping oder Spiegelung. Als Begründung muss entweder Performancesteigerung oder Ausfallsicherheit aufgeführt werden. (10 Punkte)

   - Daten verteilen, die gleichzeitig verwendet werden. (6 Punkte)
   - Daten verteilen, RAID oder Ähnliches erwähnt, aber im falschen Zusammenhang oder unpräzis. (5 Punkte)
   - Daten verteilen, aber nur vague begründet. (4 Punkte)
   - Daten verteilen, aber Begründung fehlt oder falsch, beispielsweise werden Redundanz und Ausfallsicherheit durcheinander gebracht, Parallel Server wird bemüht oder Tabellen und Indexe sollen auf dieselbe Platte. (3 Punkte)
   - I/O oder Plattenzugriffe müssen verteilt werden, ohne Details. (3 Punkte)

2. Auf dem Diagram müssen die verlangten Datenstrukturen ersichtlich sein: Datafile, Tablespace, Segment, Extent, Tabelle, Index. Wichtig ist, dass ersichtlich ist, wie die beiden Platten verwendet werden. (10 Punkte)

   - Verständlich und klar, Zusammenhang mit den Festplatten ersichtlich, aber ein oder zwei Begriffe unklar. (9 Punkte)
   - Die beiden Platten fehlen oder ein Begriff nicht ersichtlich. (6 Punkte)
   - Bilder im Detail richtig, doch Zusammenhang untereinander unklar, Bezug zu den zwei Festplatten fehlt. (5 Punkte)
   - Diverse Begriffe fehlen, es hat Fehler oder grobe Lücken. (3 Punkte)
   - Aufteilung auf zwei Platten erkenntlich, sonst nichts. (1 Punkt)

2 Verteilte Datenbanken

1. Materialized Views und DB Links sind beides akzeptable Lösungen. DB Links wären einfach zu installieren und kein Nachteil, wenn das Netzwerk schnell ist. Materialized views können entweder als *Real-Time Snapshots* oder als gewöhnliche *Snapshots* implementiert werden. Im erster Fall ist das Lesen der Daten schnell, das Schreiben der Daten allerdings langsam. Im zweiten Fall sind sowohl Lesen als auch Schreiben der Daten schnell, die Daten sind allerdings nicht auf dem neuesten Stand. (10 Punkte)

    - Fehlender Entscheid aber entsprechender Fragenkatalog mit Entscheidungskriterien. (10 Punkte)
    - Fehler in der Argumentation, aber richtige Begründung. Beispielsweise seien Snapshots langsam, deswegen werden DB-Links empfohlen (falsch), zudem seinen DB-Links einfacher zu warten (richtig). (8 Punkte)
    - Lösung in Ordnung, aber mit falscher Begründung, zB. DB-Link, damit die Daten auch bearbeitet werden können. Das geht mit Materialized View auch. (6 Punkte)

2. Konflikte treten auf, wenn ein Datensatz an zwei Orten bearbeitet wird, und zu einem späteren Zeitpunk repliziert wird. Es ist dann nicht klar, welcher Datensatz der richtige ist. Eine Variante wäre, dass immer der spätere gewinnt und den früheren Überschreibt. Dies führt zu Unsicherheiten Seiten der User. Eine weitere Möglichkeit wäre, die Konflikte manuel zu bearbeiten. Dies kann sehr Aufwendig sein. (10 Punkte)

    - Schlechter als die vorgestellten Lösungen: Eine der Datenbanken wird nur read-only verwendet. (9 Punkte)
    - Auch schlechter: Mittels Berechtigungen wird verhindert, dass Konflikte auftreten können. (9 Punkte).
    - Die Bedingen in der Frage werden ignoriert und es wird doch Real-Time Replication vorgeschlagen. (6 Punkte)
    - Problembeschreibung fehlt, aber es wird trotzdem eine richtige Massnahme vorgeschlagen. (5 Punkte)
    - Problembeschreib korrekt, aber keine korrekte Lösung vorgeschlagen. (4 Punkte)
    - Problembeschreib fehlt und eine Real-Time ähnliche Lösung wird vorgeschlagen, beispielsweise ein DB-Link. (4 Punkte)
    - Konfuse Beschreibung eines Problems beim Abgleichen. (2 Punkte)
    - Allgemeine Beschreibung von nicht Real-Time Replication. (1 Punkte)

3 Indexe

1. Bitmap ist ideal, weil das Attribut nicht selektiv ist. (5 Punkte)

    - Ein B*-Index wird vorgeschlagen, weil die Daten sich oft ändern (unwahrscheinlich). (4 Punkte)

- Ein B-Index wird aus demselben Grund vorgeschlagen. (3 Punkte)
- Unter der falschen Annahme, dass die Stati-Stammdaten in der Tabelle liegen (dh.ñur vier Datensätze), wurde vorgeschlagen keinen Index oder einen B*-Index zu verwenden. (1 Punkt)

2. Für jeden Wert in der Spalte wird ein Bitmap angelegt, welche für jeden Datensatz anzeigt, ob dieses Attribut den entsprechenden Wert hat. Diese Spalten (mindestens zwei) sind in der Zeichnung ersichtlich. (5 Punkte)

   - B*-Index wird korrekt beschrieben, weil im ersten Teil auch B-* empfohlen wurde. (5 Punkte)
   - Korrektes Diagram für Bitmap Index mit Beispieldaten aus der Aufgabe, aber ohne Text. (4 Punkte)
   - Nur ein allgemeines Diagram (zB.ãus dem Skript), ohne Text. (2 Punkte)
   - Falls im ersten Teil ein Bitmap Index empfohlen wurde, hier aber ein B*-Index korrekt beschrieben wird. (2 Punkte)
   - Ein allgemeine Vorstellung von Indexen wird geliefert mit einer Zuordnung von Schlüsselwert zu Block Adresse. (1 Punkt)

3. In einem Bitmap Index sind die Datensätze mit NULL Werten auch enthalten. Oracle pflegt diese NULL Spalte automatisch mit. Diese müssen in einem eigenen Schritt wieder entfernt werden, es sei denn, NULL Werte sind sowieso nicht möglich. Dies ist nur relevant, wenn eine Abfrage mit NOT gemacht wird, zB.s̃ucht man nach allen Projekten, welche noch nicht geliefert wurden. Dann liefert der Bitmap Index alle Projekte, welche in der entsprechenden Spalte eine 0 haben – inklusive der Projekte, die einen Status von NULL haben. Bei einem normalen B*-Index wären diese Projekte allerdings nicht dabei. (5 Punkte)

   - Es wird beschrieben, dass NULL Werte indexiert werden, und dass man die Spalte als NOT NULL deklarieren soll (aber ohne Hinweis darauf, warum die NULL Werte problematisch sind). (3 Punkte)
   - Es wird nur beschrieben, dass NULL Werte indexiert werden. (2 Punkte)
   - Allgemeine Hinweise zur Datenintegrität, zB.d̃ass damit Projekte ohne Status verhindert werden. (1 Punkt)

4. Die Index-organisierte Tabelle spart Platz, weil der Primärschlüssel nicht separat indexiert werden muss. Sie kombiniert Tabelle mit Index. Im vorliegenden Fall wird aber nicht der Primärschlüssel indexiert, sondern ein anderes Attribut. (5 Punkte)

   - Es wird erklärt, warum Platz gespart wird (eine Kopie des Schlüssels und eine ROWID weniger), und dass die Index-organisierte Tabelle sich vor allem für selektive Daten handelt, oder dass der Index “wenig bringt” oder “grosse Suchresultate” liefert. (4 Punkte)
   - Es wird erklärt, warum Platz gespart wird, und dass beim Ändern der indexierten Daten die Tabelle vielleicht umorganisiert werden muss. (4 Punkte)

- Die Index-organisierte Tabelle eignet sich nur für selektive Daten, ohne Erklärung des kleineren Platzverbrauchs. (3 Punkte)
- Beim Ändern der indexierten Daten muss die Tabelle vielleicht umorganisiert werden. (2 Punkte)
- Tabellen und Index werden zusammengelegt, ohne Erklärung warum das Platz spart, oder nur mit minimaler Erklärung ("ROWD gibt es nicht mehr"). (2 Punkte)

4 Tuning

1. Richtig sind Vorschläge, die in folgende Richtungen gehen: Überprüfen von SQL Statement, Execution Plan (Joins, Sorts), Indexe (Existenz, Spaltenreihenfolge bei kombinierten Indexen, Platzverbrauch bei sehr dynamischen Daten), Verteilung der Datenstrukturen auf den Festplatten, Denormalisierung mittels Materialized Views um Joins zu eliminieren. Falsch wären aufwändige Vorschläge: Eine zweite Datenbank mit anderen Storage Parametern, Parallel Server Option. (20 Punkte)

5 Joins

1. Der Bonus ist CHF 5250. (5 Punkte)
   (a) CHF 7350 im Jahr 2001. (5 Punkte)
   (b) CHF 7350 ohne Kommentar. (2 Punkte)
   (c) Allgemeine Formel. (0 Punkte)
2. *Nested Loops* joinen zwei Tabellen. Für jeden Datensatz aus der *Driving Table* wird der passende Datensatz aus der anderen Tabelle gesucht. (5 Punkte)
   (a) Verwendet, wenn kein equijoin möglich ist. (2 Punkte)
   (b) Performance ist schlechter als andere joins. (2 Punkte)
   (c) Hat etwas mit joins oder Daten suchen zu tun. (1 Punkte)
3. *Sort Merge* joint zwei Tabellen. Beide Tabellen werden sortiert, und dann werden wird jeder Datensatz aus der *Driving Table* mit den Datensätzen aus der anderen Tabelle verglichen. Solange die Werte gleich sind, wird gejoint; sobald die Werte verschieden sind, wird der Vergleich abgebrochen, die bisher gejointen Datensätze der*Driving Table* entfernt, und der nächste Datensatz wird dran genommen. (5 Punkte)
   (a) Daten werden sortiert und dann zusammengefügt. (3 Punkte)
   (b) Daten werden sortiert. (1 Punkte)
   (c) Hat etwas mit joins oder Daten suchen zu tun. (1 Punkte)
4. Ein *Sort Merge* funktioniert nur bei einem *equijoin*, nicht bei einem BETWEEN. (5 Punkte)
   (a) Ein between bedeuted, dass die Daten nicht "gleich" sind. (4 Punkte)
   (b) Ein between verhindert dies. (3 Punkte)