Anmerkung:

1. Die Lese-Schreib-Quoren-Frage kommt fast bei jeder Prüfung
2. Die Fragen sind nach Themengebieten geordnet (soweit das ging)
3. Es gibt nur einpaar „Praxis“-Fragen die falsch sind (siehe unten) -> die wahrscheindlichkeit dass eine „Praxis“-Frage falsch ist dürft Ihr Euch selber ausrechnen ;)
4. Wenn Fragen ähnlich sind (richtig/falsch), sind die unterschiede **fettgedruckt und unterstrichen**
5. Alle grünen fragen sind richtig und alle roten sind falsch

Lese- & Schreib-Quoren
Eine Relation R sei an 5 Netzwerkknoten materialisiert mit den Gewichten 5, 9, 7, 3 und 5. Dann sind Qr(R) = 10 und Qw(R) = 20 gülitge Lese- bzw. Schreib-Quoren.

Wenn die folgenden Formeln erfüllt sind, dann sind die Quoren gültig:
Qr(R) + Qw(R) > Summe der Gewichte
2 * Qw(R) > Summe der Gewichte

Theorie

ACID garantiert die Eigenschaften Atomicity, Consistency, Isolation, und Durability.

Callable Statements dienen bei JDBC unter anderem dazu, die Performance der Anfragen zu steigern.

Prepared Statements dienen bei JDBC unter Anderem dazu, die Performance der Anfragen zu steigern.

Will man in PL/pgSQL eine komplette Zeile einer Tabelle “table„ in einer Variable ablegen, so muss diese entweder vom Typ RECORD oder vom Typ table%ROWTYPE sein.

In PL/pgSQL-Blöcken haben in SQL-Statements Variablennamen immer Präferenz über andere Bezeichner wie Spalten- oder Tabellennamen.

Ein WITH RECURSIVE Statement in PostgreSQL erzeugt eine Common Table Expression, deren Inhalt in der darauffolgenden referenzierenden SQL-Query verwendet werden kann.

Wird der Wert einer Sequence in einer Transaktion verändert und danach ein Rollback durchgeführt, so bleibt der veränderte Wert der Sequence trotzdem erhalten.

Obwohl die Reihenfolge von **Natural Joins** für das Ergebnis irrelevant ist (da sie assoziativ sind), kann sie doch starke Auswirkungen auf die Laufzeit der Query haben.

Obwohl die Reihenfolge von **Joins** für das Ergebnis irrelevant ist (da Joins assoziativ sind), kann sie doch starke Auswirkungen auf die Laufzeit der Query haben.

Bei verletzten Deferred Constraints treten Fehler erst am Ende der Transaktion auf.

Falls das Rollback von Datenbankänderungen unterbrochen wird, muss das Datenbanksystem beim Wiederanlauf in der Lage sein, das Rollback abzuschließen.

Nehmen Sie an, dass eine relationale Datenbank um das objektrelationale Feature “Geschachtelte Relationen” erweitert wurde. Dann lassen sich unter Umständen Anfragen bezüglich 1:n-Relationen effizienter auswerten als ohne dieses Feature.

Wenn ein Join mittels Block Nested Loop Join realisiert wird, kann die Anzahl der benötigten I/O-Operationen im Idealfall weniger sein, als wenn derselbe Join mittels Nested Loop Join realisiert würde.

Bei Cross-Joins spielt es keine Rolle, ob Projektionen vor oder nach dem Join ausgeführt werden.

Beim Zweiphasen-Commit-Protokoll kann der **Koordinator** mittels Timer-Überwachung verhindern, dass beim Absturz eines Agenten die Beendigung einer Transaktion blockiert wird.

Eine Tupel-ID kann auch auf eine weitere Tupel-ID verweisen. Die wirklich Werte befinden sich dann in einer anderen Seite. Dies wird auch als “Forward” bezeichnet.

Bei der horizontalen Fragmentierung ergeben sich für n Zerlegungsprädikate 2n Fragmente, die durch eine **Vereinigung** wieder zusammengeführt werden können.

In der relationalen Algebra ist die Operation **σ** distributiv mit ∪, ∩, und \, die Operation **π** ist distributiv mit ∪.

Betrachten Sie die Kostenformel 2 · bR · (1 + l) mit l = ┌logm−1 (┌**bR**/m┐)┐ für das externe Sortieren. Dabei steht l für die Anzahl der “passes”.

Betrachten Sie die Kostenformel 2 ∗ bR ∗ (1 + l) mit l = ┌logm−1 (┌**bR**/m┐)┐ für das externe Sortieren. Dabei steht bR für die Anzahl der Seiten der Relation R und m für die Anzahl der verfügbaren Puffer-Seiten.

Praxis

Für X(ABC) und Y (BCD) ist X ◃▹ Y äquivalent zu ΠA,X.B,X.C,D(σX.B=Y.B,X.C=Y.C(X × Y)).

Betrachten Sie zwei Relationen R(ABC) und S (ABD). Dann gilt auf jeden Fall folgende Gleichheit: πAB(R ◃▹ S) = πAB(R) ∩ πAB(S).

Betrachten Sie zwei Relationen R(ABC) und S (ABD). Dann liefert der Ausdruck πAB(S) ◃▹ R ausschließlich Tupel, die auch in R enthalten sind.

Betrachten Sie zwei Relationen R(ABC) und S(BDE). Dann liefert der Ausdruck (πB(R)∪πB(S)) ◃▹ R ausschließlich Tupel, die auch in R enthalten sind.

Betrachten Sie zwei Relationen R(AB) mit 5000 Tupeln und S(AC) mit 3000 Tupeln, wobei A ein Fremdschlüssel von der Relation S auf den Primärschlüssel A in R ist. Dann ergibt der Ausdruck R ◃▹ S exakt 3000 Tupel.

Für Relation R(AB) mit 100 Tupeln und Relation S (AC) mit 10 Tupeln enthält ΠA(R) LEFT OUTER JOIN ΠA(S) maximal 10 Tupel.

Es gibt Relationen R(AB) mit **100** Tupeln und S(AC) mit **100** Tupeln, für die der Ausdruck R 1 S 10000 Tupeln ergibt.

Nehmen Sie an, dass eine Relation S 10000 Seiten umfasst und dass die Puffergröße 32 beträgt. Dann ist bei einem Hash- Join von R mit S in der Build-Phase **möglicherweise aber nicht notwendigerweise** ein Re-hashing von S erforderlich.

Nehmen Sie an, dass eine Relation R1 aus 6000 Seiten und eine Relation R2 aus 20000 Seiten besteht. Nehmen Sie weiters an, dass 8000 Pufferseiten für die Berechnung des Joins R1 ⋪⋫ R2 zur Verfügung stehen. In diesem Fall haben Sort Merge Join und Block Nested Loop Join (bei optimaler Wahl von innerer und äußerer Relation) exakt die gleichen I/O-Kosten.

Betrachten Sie drei Relationen U(AB), V (BC) und W(CD). Dann gilt auf jeden Fall folgende Gleichheit: (U⋪⋫V)⋪⋫W)=(U×W)⋉V)

Betrachten Sie drei Relationen R(AB), S(AB) und T(BC). Dann gilt auf jeden Fall folgende Gleichheit: πC((R ∩ S) ⋪⋫ T) = πC([R ⋪⋫ T] ⋪⋫ [S ⋪⋫ πB(T)]).

Nehmen Sie an, dass in einem verteilten DBMS das globale Relationenschema R(<u>A</u>BCDE) (d.h.: A ist der Primärschlüssel) in die vertikalen Fragmente R(AB), R(ABCD), und R(ADE) unterteilt wurde. Diese Fragmentierung ist vollständig und rekonstruierbar.

Die Historie r1(A), r2(B), w1(A), w1(B), w2(B), c2, c1 hat folgende Eigenschaften: Sie vermeidet kaskadierendes Rücksetzen, und sie ist nicht serialisierbar.

Die Historie r2(X), w2(X), r1(Y), c2, r1(X), w1(X), w1(Y), c1 ist sowohl beim Zwei-Phasen Sperrprotokoll als auch beim strengen Zwei-Phasen Sperrprotokoll möglich.

Die Historie r1(A), r2(B), w2(B), w1(A), w1(C), w2(C), c2, c1 ist zwar beim Zwei-Phasen Sperrprotokoll jedoch nicht beim strengen Zwei-Phasen Sperrprotokoll möglich.

Die Historie r1(A), w1(A), r2(B), c1, r2(A), w2(A), w2(B), c2 ist sowohl beim Zwei-Phasen Sperrprotokoll als auch beim strengen Zwei-Phasen Sperrprotokoll möglich.

Betrachten Sie folgende Historie, bei der nur BOT (= Transaktionsbeginn), ci (= Commit) und Sperranforderungen (aber keine Datenzugriffe oder Informationen über Warten bzw. Wecken einer Transaktion) gegeben sind: BOT1, BOT2, lockX1(A), lockX2(B), lockS1(B), c2, c1. Diese Historie ist beim Zeitstempel-Verfahren mit wait-die Strategie möglich.

Theorie:

RAID garantiert die Eigenschaften Redundancy, Atomicity, Isolation, und Durability.

Bei Verwendung von RAID Level 6 mit Reed-Solomon Code ist auf jeden Fall die durability Eigenschaft von ACID Transaktionen erfüllt.

Wird ein RAID-System zum Speichern der Daten verwendet, so wird dadurch immer auch die Ausfallsicherheit des Systems erhöht.

PL/(pg)SQL ist eine prozedurale Sprache und spezifiziert daher lediglich was, niemals jedoch wie selektiert werden soll.

Mit SQL-92 lassen sich Anfragen formulieren, die man mit PL/pgSQL nicht formulieren könnte.

JDBC ist eine Schnittstelle, die sowohl in Java (per JDBC-Driver) als auch direkt in der Datenbank implementiert sein muss, um zu funktionieren.

Verwendet man das JDBC DataSource Interface um Datenbankverbindungen aufzubauen, so wird dadurch auch immer Connection Pooling angewandt.

Wenn nur eine Leseoperation durchgeführt wird, kann die Datenbank auch direkt von der Festplatte lesen, ohne die komplette Seite in den Hauptspeicher zu laden.

Die Größe der Zwischenergebnisse spielt bei Pipelining keine Rolle. Solange also kein Materializing-Schritt dazwischen kommt, ist die Join- Reihenfolge egal.

Ein geballter Index kann nur mit Hilfe eines B+-Baumes erstellt werden, bei Hash-Indizierung ist ein geballter Index nicht möglich.

In einem verteilten DBMS benötigt ein einzelner Agent unter Umständen eine Kommunikation mit anderen Agenten, um nach einem Absturz die „winner“ und „looser“ Transaktionen zu erkennen.

Durch mehrere aufeinanderfolgende Forwards (auch „Chaining“ genannt) kann es zu hohen Zugriffszeiten kommen, da viele Seiten für nur ein Tupel eingelagert werden müssen.

Beim Zweiphasen-Commit-Protokoll können die **Agenten** mittels Timer-überwachung – durch geeignete Wahl des Timer-Intervalls – verhindern, dass sie bei einem Absturz des Koordinators blockiert werden.

Bei einem nicht perfekt balancierten B+ Baum vom Grad k mit n Elementen kann unter Umständen das Einfügen bzw. Löschen von einem Element länger als O(logk n) dauern.

Die Anzahl der Zyklen im Wartegraphen entspricht immer der Anzahl der Transaktionen, die zurückgesetzt werden müssen, um ein Deadlock aufzulösen.

Der Einsatz eines PERFORM-Statements in PL/pgSQL ist nur zulässig, wenn durch die Anfrage keine Seiteneffekte entstehen können.

Jede Bedingung eines FOREIGN KEY Constraints kann auch mittels eines CHECK Constraints überprüft werden.

Alle Constraints, die per ALTER TABLE Statement angelegt werden, könnten auch sofort im CREATE TABLE Statement angelegt werden.

Bei FOR EACH STATEMENT Triggern enthalten die Variablen OLD und NEW ein Array aller geänderten Datensätze.

Wenn in einem Auswertungsplan alle Block Nested Loop Joins durch Hash Joins ersetzt werden, dann kann dies unter Umständen zu einer Verringerung der Anzahl der Tupel in den Zwischenergebnissen führen.

Wenn ein Join mittels Hybrid Hash Join realisiert wird, dann kann das dadurch berechnete Ergebnis im Idealfall weniger Tupel umfassen, als wenn derselbe Join mittels Hash Join realisiert würde.

Bei der vertikalen Fragmentierung ergeben sich für n Zerlegungsprädikate 2n Fragmente, die durch einen **Join** wieder zusammengeführt werden können.

Die Gesamtkosten für die Sortierung einer Tabelle T sind 2bT (1 + ┌logm−1(┌m/**bT** ┐)┐), mit bT als Anzahl der Seiten von T am Hintergrundspeicher und m als Anzahl der Seitenrahmen im Datenbankpuffer.

In der relationalen Algebra ist die Operation **π** distributiv mit ∪, ∩, und \, die Operation **σ** ist distributiv mit ∪.

Praxis:

Nehmen Sie an, dass eine Relation R 16384 (= 1282) Seiten umfasst und die Puffergröße 128 beträgt. Dann ist bei einem Hash Join von R mit einer beliebigen Relation S in der Build-Phase **auf jeden Fall** ein Re-Hashing von R erforderlich.

Nehmen Sie an, dass eine Relation R aus 5000 Seiten besteht und dass die Puffergröße 100 beträgt. Dann ist bei einem Hash-Join von R mit einer beliebigen Relation S in der Build-Phase **möglicherweise** ein Re-hashing von S erforderlich.

Eine Datenbank enthalte die Relation R(AC) mit m Tupeln und die Relation S (BC) mit n Tupeln. Nehmen Sie an, dass die beiden Relationen bereits (nach dem Attribut C) sortiert wurden. Dann betragen die I/O-Kosten für den merge join (um den Ausdruck R ◃▹ S zu berechnen) m + n.

Für Relation R(AB) mit 100 Tupeln und Relation S(AC) mit 10 Tupeln enthält ΠA(R)−ΠA(S) mindestens 90 Tupel.

Betrachten Sie zwei Relationen R(AB) und S(AB). Dann gilt auf jeden Fall folgende Gleichheit: πA(R−S) = πAR−πAS

Es gibt Relationen R(AB) mit 10 Tupeln und S(AC) mit 100 Tupeln, für die der Ausdruck R 1 S 1000 Tupeln ergibt.

Die Historie w1(A), r2(A), r2(B), w1(B), w2(C), c1, c2 ist serialisierbar und nicht strikt.

Die Historie w2(C), r1(C), r1(D), w2(D), w1(A), c2, c1 ist serialisierbar und nicht strikt.