Database tuning and Course review

Lecture 11
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

10 June 2015

Agenda

- Mini-lecture on DB Tuning
- Review of course and prep for final exam

DB tuning

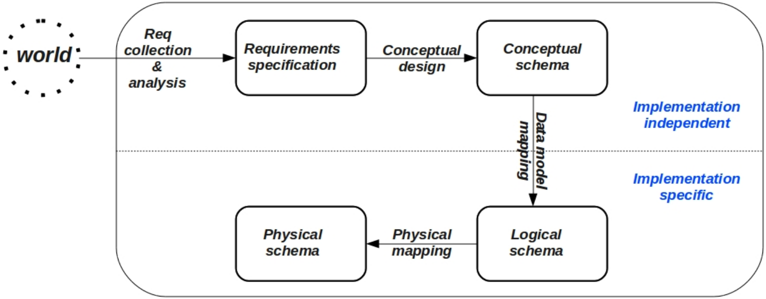
DB tuning: performance improvement, short of more/better hardware, in terms of response time and throughput.

DB tuning

DB tuning: performance improvement, short of more/better hardware, in terms of response time and throughput.

DB tuning is not about changing data semantics.

DB design



DB design process

DB design

“Goodness” in DB design

- Conceptual. Accurately reflect the semantics of use in the modeled domain.
- Logical. Accurately reflect conceptual model and disallow redundancies and update anomalies, as best possible.

DB design

"Goodness" in DB design

- Conceptual. Accurately reflect the semantics of use in the modeled domain.
- Logical. Accurately reflect conceptual model and disallow redundancies and update anomalies, as best possible.
- Physical. Accurately reflect logical model and efficiently and reliably support use of data.

DB design

"Goodness" in DB design

- Conceptual. Accurately reflect the semantics of use in the modeled domain.
- Logical. Accurately reflect conceptual model and disallow redundancies and update anomalies, as best possible.
- Physical. Accurately reflect logical model and efficiently and reliably support use of data.

The distinction between design and tuning is somewhat fuzzy

- design is about *semantics*
- tuning is about *efficient use*

Job of the DBA: design, tuning, protection, maintenance

DB tuning

Tuning is driven by observed and/or expected workload

- list of queries and updates, and their frequencies
  - relations involved
  - attributes involved
  - selection and join predicates

DB tuning

Tuning is driven by observed and/or expected workload

- list of queries and updates, and their frequencies
  - relations involved
  - attributes involved
  - selection and join predicates

Workload is never truly random, and hence we can leverage and support regular patterns of use

DB tuning

Tuning is driven by observed and/or expected workload

- list of queries and updates, and their frequencies
  - relations involved
  - attributes involved
  - selection and join predicates

Workload is never truly random, and hence we can leverage and support regular patterns of use

Let's consider tuning the basic components: buffers, indexes, schemas, queries

Tuning buffers and indexes

Six guidelines.

(1) Two types of cache:

- page cache
- procedure cache, for recent query plans

DBA can usually tweak cache size and replacement policy

Tuning buffers and indexes

Six guidelines.

(1) Two types of cache:

- page cache
- procedure cache, for recent query plans

DBA can usually tweak cache size and replacement policy

(2) To index or not to index: don't create indexes unnecessarily

- wasted space
- confuses query optimizer
- cost of maintenance can outweigh benefits

Tuning buffers and indexes

(3) Choice of search key: based on workload

- exact match vs. range selection
- multi-attribute

Tuning buffers and indexes

(3) Choice of search key: based on workload

- exact match vs. range selection
- multi-attribute

(4) Hash-based vs. tree-based

- usually B+tree
- except when
  - frequent join – index on join attributes of inner relation
  - equality selections in query

Tuning buffers and indexes

(5) Balance the cost of index maintenance: if extremely dynamic, may not be worth it.

Tuning buffers and indexes

(5) Balance the cost of index maintenance: if extremely dynamic, may not be worth it.

(6) To cluster or not to cluster: not always necessary on primary key (often the default) as this grouping might not give you much

- look at workload

Tuning buffers and indexes

Example.

```
SELECT M.ID, M.DeptID
FROM Managers M
WHERE M.Name = "Pointy-Haired Boss"
```

Tuning buffers and indexes

Example.

```
SELECT M.ID, M.DeptID
FROM Managers M
WHERE M.Name = "Pointy-Haired Boss"
```

Default is a clustered index on primary key `Managers.ID`

- not useful for this query
- if manager names are fairly unique, build unclustered index on `Managers.Name`, or, if warranted, build clustered index

Tuning buffers and indexes

Example.

```
SELECT M.ID, M.DeptID
FROM Managers M
WHERE M.Name = "Pointy-Haired Boss"
```

Default is a clustered index on primary key `Managers.ID`

- not useful for this query
- if manager names are fairly unique, build unclustered index on `Managers.Name`, or, if warranted, build clustered index
- also holds for `Managers.Age` and `Managers.Salary`
- Hash-based or Ordered index?

Tuning buffers and indexes

Example.

ProjectPart(projectID, partID)
PartSupplier(partID, supplierID)

```
SELECT P.projID, S.supplierID
FROM ProjectPart P, PartSupplier S
WHERE P.partID = S.partID
```

Tuning buffers and indexes

Example.

ProjectPart(projectID, partID)
PartSupplier(partID, supplierID)

```
SELECT P.projID, S.supplierID
FROM ProjectPart P, PartSupplier S
WHERE P.partID = S.partID
```

It seems natural to build an index on `PartSupplier.partID` to help out index nested loops join with `PartSupplier` as inner relation.

Tuning buffers and indexes

Example.
ProjectPart(projectID, partID)
PartSupplier(partID, supplierID)

```
SELECT P.projID, S.supplierID
FROM ProjectPart P, PartSupplier S
WHERE P.partID = S.partID
```

It seems natural to build an index on `PartSupplier.partID` to help out index nested loops join with `PartSupplier` as inner relation.

However, potentially many rows of `PartSupplier` will join with each row of `ProjectPart`, leading to large result size!

- need to use some other join algorithm
- hence, we should not build this index!

Tuning schemas

Three basics.

1: denormalize for read-heavy queries, and pay a higher price for maintaining data consistency

Tuning schemas

Three basics.

1: denormalize for read-heavy queries, and pay a higher price for maintaining data consistency

- for example, if the the join on `partID` is very common, join `ProjectPart` and `PartSupplier` as one relation

  ProjectPartSupplier(projectID, partID, supplierID)

Tuning schemas

Three basics.

1: denormalize for read-heavy queries, and pay a higher price for maintaining data consistency

- for example, if the the join on `partID` is very common, join `ProjectPart` and `PartSupplier` as one relation

  ProjectPartSupplier(projectID, partID, supplierID)

2: vertical partitioning. place infrequently used columns in a separate table (smaller tuples = smaller pages = more records per I/O)

Tuning schemas

Three basics.

1: denormalize for read-heavy queries, and pay a higher price for maintaining data consistency

- for example, if the the join on `partID` is very common, join `ProjectPart` and `PartSupplier` as one relation
  ProjectPartSupplier(projectID, partID, supplierID)

2: vertical partitioning. place infrequently used columns in a separate table (smaller tuples = smaller pages = more records per I/O)

- for example, place rarely accessed attributes such as `hireDate` and `hireReferral` in separate table from `Managers`
- cf. column stores such as MonetDB

Tuning schemas

3: horizontal partitioning. place infrequently used rows in a separate table (fewer tuples = smaller tables = fewer I/O's)

Tuning schemas

3: horizontal partitioning. place infrequently used rows in a separate table (fewer tuples = smaller tables = fewer I/O's)

- for example, place fired or retired managers in a separate table from `Managers`
- cf. google file system; distributed DB design

Tuning queries

Five heuristics.

(1) avoid sorts: `DISTINCT`, `UNION`, `INTERSECT`, `EXCEPT`.

Tuning queries

Five heuristics.

(1) avoid sorts: `DISTINCT`, `UNION`, `INTERSECT`, `EXCEPT`.
for example, in the query

```
SELECT DISTINCT M.ID, M.DeptID
FROM Managers M
WHERE M.Name = "Pointy-Haired Boss"
```

the `DISTINCT` is unnecessary, as `ID` is a key for `Managers`, so is also a key for any subset of `Managers`

Tuning queries

(1, cont.) consider next

```
SELECT DISTINCT M.ID, D.Phone
FROM Managers M, Departments D
WHERE M.DID = D.DID
```

Tuning queries

(1, cont.) consider next

```
SELECT DISTINCT M.ID, D.Phone
FROM Managers M, Departments D
WHERE M.DID = D.DID
```

Is `DISTINCT` necessary?

Tuning queries

(1, cont.) consider next

```
SELECT DISTINCT M.ID, D.Phone
FROM Managers M, Departments D
WHERE M.DID = D.DID
```

Is `DISTINCT` necessary? No, since `DID` is a key for `Departments` (and `ID` is a key for `Managers`)

Tuning queries

(1, cont.) consider next

```
SELECT DISTINCT M.ID, D.Phone
FROM Managers M, Departments D
WHERE M.DID = D.DID
```

Is `DISTINCT` necessary? No, since `DID` is a key for `Departments` (and `ID` is a key for `Managers`)

The relationship among `DISTINCT`, keys and joins can be generalized:

- Call a table T *privileged* if the fields returned by the `SELECT` contain a key of T.

Tuning queries

(1, cont.) consider next

```
SELECT DISTINCT M.ID, D.Phone
FROM Managers M, Departments D
WHERE M.DID = D.DID
```

Is `DISTINCT` necessary? No, since `DID` is a key for `Departments` (and `ID` is a key for `Managers`)

The relationship among `DISTINCT`, keys and joins can be generalized:

- Call a table T *privileged* if the fields returned by the `SELECT` contain a key of T.
- Let R be an unprivileged table. Suppose that R is joined on equality by its key field to some other table S, then we say R *reaches* S.
- Now, define reaches to be transitive. So, if R_1 reaches R_2 and R_2 reaches R_3, then say that R_1 reaches R_3.

Tuning queries

(1, cont.) Reaches Theorem.

There will be no duplicates among the records returned by a selection, even in the absence of `DISTINCT`, if one of the two following conditions hold:

- Every table mentioned in the `FROM` clause is privileged.
- Every unprivileged table reaches at least one privileged table.

Tuning queries

(2) avoid unnecessary scans: "$\neq$" in selection condition can often be rewritten

- for example, `credits != 3` might be rewritten as `credits = 1 OR ... credits = 4`

Tuning queries

(2) avoid unnecessary scans: "$\neq$" in selection condition can often be rewritten

- for example, `credits != 3` might be rewritten as `credits = 1 OR ...  credits = 4`

(3) instead of creating new indexes, check for usage of existing indexes

- for example, `Name = ‘‘Fred’’ OR Salary = 4000`
- if separate indexes exist on `Name` and `Salary`, compiler might not catch this
- instead, rewrite as union of two queries

Tuning queries

(4) help the optimizer reuse procedure cache. for example,

```
SELECT P.Name
FROM Professor P
WHERE P.DeptID = ‘Math’
```

and we will also want profs from EE, ME, IE, ...

Tuning queries

(4) help the optimizer reuse procedure cache. for example,

```
SELECT P.Name
FROM Professor P
WHERE P.DeptID = ‘Math’
```

and we will also want profs from EE, ME, IE, ...

The optimizer most likely won't reuse the plan for the first query in these later queries, even though that plan is cached. Instead, we should use a host variable:
...`WHERE P.DeptID = :deptID` ...

Tuning queries

(5) decorrelate and unnest complex queries.

Tuning queries

(5) decorrelate and unnest complex queries. for example,

```
SELECT P.Name
FROM Professor P
WHERE EXISTS ( SELECT *
               FROM department D
               WHERE D.Name = ‘Mathematics’
                     AND
                     P.DeptID = D.ID )
```

Tuning queries

(5) decorrelate and unnest complex queries. for example,

```
SELECT P.Name
FROM Professor P
WHERE EXISTS ( SELECT *
               FROM department D
               WHERE D.Name = ‘Mathematics’
                     AND
                     P.DeptID = D.ID )
```

Typical optimizer will generate a plan which executes the inner correlated query for each Profs tuple! We should decorrelate if possible.

Tuning queries

(5) decorrelate and unnest complex queries. for example,

```
SELECT P.Name
FROM Professor P
WHERE EXISTS ( SELECT *
               FROM department D
               WHERE D.Name = ‘Mathematics’
                     AND
                     P.DeptID = D.ID )
```

Typical optimizer will generate a plan which executes the inner correlated query for each Profs tuple! We should decorrelate if possible.

```
SELECT P.Name
FROM Professor P
WHERE P.DeptID IN ( SELECT D.ID
                    FROM department D
                    WHERE D.Name = ‘Mathematics’ )
```

Tuning queries

(5, cont.) However, optimizer won't recognize the implicit join here, and won't make use of available indexes. Hence, we should unnest if possible

```
SELECT P.Name
FROM Professor P, Department D
WHERE P.DeptID = D.ID AND D.Name = 'Mathematics'
```

Tuning queries

(5, cont.) However, optimizer won't recognize the implicit join here, and won't make use of available indexes. Hence, we should unnest if possible

```
SELECT P.Name
FROM Professor P, Department D
WHERE P.DeptID = D.ID AND D.Name = 'Mathematics'
```

In general, optimizers don't recognize equivalence of such plans, and so it is up to the client/DBA to tune.

DB tuning: tools

All major systems provide tools and means for tuning (and, third-party support as well).

DB tuning: tools

All major systems provide tools and means for tuning (and, third-party support as well).

Example. SQLite DB tuning.

- SQLite analyzer, from SQLite.
- SQLite Database Browser, open source.
- SQLite Manager, commercial product.
- SQLite Expert, commercial product.

DB tuning: tools

All major systems provide tools and means for tuning (and, third-party support as well).

Example. SQLite DB tuning.

- SQLite analyzer, from SQLite.
- SQLite Database Browser, open source.
- SQLite Manager, commercial product.
- SQLite Expert, commercial product.

Configuration:

- `PRAGMA cacheSize`
- `PRAGMA cacheSpill` to set steal/no-steal cache policy.
- `PRAGMA walAutocheckpoint`, to set checkpoint policy for the write-ahead log.
- `ANALYZE`, `REINDEX`, and `VACUUM`, to rebuild/clean the DB and stats
- `EXPLAIN`, to view query plan in virtual machine code

DB tuning: recap

- DB tuning is not about changing data semantics.
- Tuning is driven by observed and/or expected workload which is never truly random
- basic tuning components: buffers, indexes, schemas, queries
- every DB system has many, many tuning knobs, and lots of tool support for tuning

DB tuning: recap

- DB tuning is not about changing data semantics.
- Tuning is driven by observed and/or expected workload which is never truly random
- basic tuning components: buffers, indexes, schemas, queries
- every DB system has many, many tuning knobs, and lots of tool support for tuning

Nice book on DB tuning. *Database Tuning: principles, experiments, and troubleshooting techniques.* Shasha and Bonnet, 2002.

Exercise

SalesPerson(<u>personID</u>, regionID, pName, address, age, hireDate, taxCode, healthCode, parkingSpot, ...)
Region(<u>regionID</u>, rName, area, rCode, history, taxOffice, ...)
Sales(<u>personID</u>, <u>regionID</u>, amount)

Exercise

SalesPerson(<u>personID</u>, regionID, pName, address, age, hireDate,
taxCode, healthCode, parkingSpot, ...)
Region(<u>regionID</u>, rName, area, rCode, history, taxOffice, ...)
Sales(<u>personID</u>, <u>regionID</u>, amount)

Very common query:

```
SELECT DISTINCT P.personID, P.address, P.pName, R.rName
FROM SalesPerson P, Region R, Sales S
WHERE P.personID = S.personID AND R.regionID = S.regionID
      AND P.regionID = R.regionID
```

Can we tune the schema and/or query to improve response time?

Course review

Recap

Major topics

Relational data in external memory (lectures 1-4)

- storage, sorting, indexing

Recap

Major topics

Relational data in external memory (lectures 1-4)

- storage, sorting, indexing

Query processing (lecture 5)

- Implementing σ, π, $\bowtie$, $\cup$, $-$

Recap

Major topics

Relational data in external memory (lectures 1-4)

- storage, sorting, indexing

Query processing (lecture 5)

- Implementing σ, π, $\bowtie$, $\cup$, $-$

Views (lecture 6)

- Histograms, reasoning about views

Recap

Major topics

Relational data in external memory (lectures 1-4)

- storage, sorting, indexing

Query processing (lecture 5)

- Implementing σ, π, $\bowtie$, $\cup$, $-$

Views (lecture 6)

- Histograms, reasoning about views

Query optimization (lecture 7)

- logical plans, physical plans, dynamic programming

Recap

Major topics, cont.

Distributed query processing (lecture 8)

- parallel, distributed, and mediator systems; acyclic queries

Recap

Major topics, cont.

Distributed query processing (lecture 8)

- parallel, distributed, and mediator systems; acyclic queries

Transaction management (lecture 9)

- ACID, 2PL, S2PL

Recap

Major topics, cont.

Distributed query processing (lecture 8)

- parallel, distributed, and mediator systems; acyclic queries

Transaction management (lecture 9)

- ACID, 2PL, S2PL

NoSQL, graphs, and linked data (lecture 10)

Recap

Major topics, cont.

Distributed query processing (lecture 8)

- parallel, distributed, and mediator systems; acyclic queries

Transaction management (lecture 9)

- ACID, 2PL, S2PL

NoSQL, graphs, and linked data (lecture 10)

Tuning and recap (lecture 11)

Old final exam

Five questions, three hours. Let's practice the first three or four.

Old final exam

Five questions, three hours. Let's practice the first three or four.

(1) Consider the following conjunctive query Q:

$$result(A, B, C, D, E, F) \quad \leftarrow \quad r(A, B, C), s(A, F, E), t(E, D, C), u(A, C, E)$$

1. Give the hypergraph representation of Q.
2. Is Q acyclic? If so, can it be made cyclic by removing one hyperedge? Otherwise, can it be made acyclic by removing one hyperedge?

Old final exam

(2) Given a schedule S over transactions $\{T_1, \ldots, T_n\}$, the "strong graph" associated with S is the directed graph $\mathrm{sg}(S)$ having exactly one node for each transaction of S and an edge from T_i to T_j (for $i \neq j$) if and only if in S, for some object X there is an action $\alpha_i(X)$ of T_i on X which appears before an action $\alpha_j(X)$ of T_j on X.
Prove or disprove the following claims.

1. If $\mathrm{sg}(S)$ is acyclic, then S is conflict serializable.
2. If S is conflict serializable, then $\mathrm{sg}(S)$ is acyclic.

Old final exam

(3) Consider the following conjunctive queries.

$$
\begin{array}{lrcl}
Q_1: & \mathit{result}(A) & \leftarrow & r(A,B), r(A,C), s(B,D,E), s(B,F,F) \\
Q_2: & \mathit{result}(X) & \leftarrow & r(X,Y), r(X,W), s(Y,W,W), t(X)
\end{array}
$$

Is it the case that $Q_2 \subseteq Q_1$? Prove your answer.

Old final exam

(4) Consider the "semi-difference" relational algebra operator, defined as

$$
\begin{aligned}
R \rhd S &= \{r \in R \mid \neg\exists s \in S (r \bowtie s \in R \bowtie S)\} \\
&= R - (R \ltimes S).
\end{aligned}
$$

Formally prove or disprove the following proposals for relational algebra equivalences.

1. $\sigma_\theta(R \rhd S) = \sigma_\theta(R) \rhd S$, where θ is a standard single-table selection condition which mentions only attributes in R (i.e., $atts(\theta) \subseteq atts(R) - atts(S)$).
2. $R \ltimes S = R \rhd (R \rhd S)$.