Database Tuning

Rasmus Pagh and Srinivasa Rao
IT University of Copenhagen

Spring 2007

Introduction, practical information, and recap of database background

February 1, 2007

Partly based on [RG 1.5-1.8; 4.2] (not curriculum)

Today's lecture (2 hours)

- Course introduction
- Practical information
- Overview of lectures
- Quick recap of relevant database background

What is a database?

According to Webster's dictionary:

da·ta·base
a usually large **collection of data** organized especially for rapid search and retrieval (as by a computer)

What is a database management system?

Database management system (DBMS):

Software system used when implementing databases
more precisely
System for providing **efficient**, **convenient**, and **safe** storage of and **multi-user** access to, possibly **massive**, amounts of **persistent** data.

Problem session: (5 minutes, discuss in groups of 2-4 students)

In this course we focus on massive amounts of data. Think about examples of large data sets for which there is a need for some kind of DBMS (existing or still not existing).

What the course is about

Your main goals in the course should be to:

- Understand how relational database systems work and what influences their performance. Main focus is on large data sets.
- Get an overview of indexing methods, both ones that are presently common, and ones that are likely to play a role in the future.
- Be able to use the above knowledge for tuning, and for critically evaluating upcoming database technologies.

Why this course is important

- Tuning databases is an important activity for many database developers and administrators.
- Most common tasks of databases can be done fast when the amount of data is small, while it may be very time consuming for larger data sets.
- There is a need for people that are able to handle massive data sets! In many areas the amount of data grows faster than the size of internal memory, e.g. biological data, internet pages. Also, it is becoming common to store large amounts of historical data. This means that the data to process cannot be expected to fit into internal memory.

Why this course is interesting

- Quote from "Database Tuning": *"Tuning is difficult because the principles and knowledge underlying that common sense requires a broad and deep understanding of the application, the database software, the operating system, and the physics of the hardware."*
- You will see many smart and elegant algorithmic ideas that go into DBMS software.
- Tuning is like athletics — infinitely challenging, always a way of doing better.

Course focus

The focus will be on scalability aspects of database implementation:

- Efficient data structures for indexes
- Efficient implementation of relational operations

Big issue: Internal vs. external memory:

- Time to retrieve one word from internal memory: $<$ 100 ns
- Time to retrieve one word from external memory: $\approx$10 ms ($= 10^7$ ns)
- Time to perform 100 CPU operations: $<$ 40 ns

Conclusion: When data must be stored in external memory, the time to retrieve data from disk is most often the bottleneck.

I/O model

Block transfers:
Because of the large access time, every external memory access is used to transfer a whole *block* of adjacent data. (Disk blocks are typically 4-16 kilobytes.)

Question: Why do block transfers help?

A particularly simple model of external memory is the *I/O model*, which will be used for most of the material in the course:

- The complexity of an algorithm is the number of block reads and writes (I/Os) it makes.
- Complexity depends on block size (B) and size of internal memory (M).

Problem session (discuss in small groups)

Pretend you are a DBMS!

Now suppose that all the relations mentioned in the queries on the hand-out are very large (residing on external memory).

How would you process each one of the queries?

Next: Practical information

Course format

Lectures and problem sessions:

Mainly Thursdays 10.00-12.00 and 13.00-15.00.

Mix of lectures and problem sessions/exercises without preparation.

Project:

4 project deliverables to be handed in during the course, and one project report at the end of the course.

The project will be initiated on February 6, and most project activities (common sessions, supervision, feedback) will run on Tuesdays.

Exam: Report + oral on June 27 or 28, 2007.

Manning of course and course homepage

Lecturers:

Rasmus Pagh, `pagh@itu.dk`, office 3C.07. (Course responsible.)

Srinivasa Rao, `ssrao@itu.dk`, office 3C.04.

Project supervisor:

Milan Ruzic, `milan@itu.dk`, office 3C.10.

Homepage: `www.itu.dk/people/pagh/DBT07/`

- News.
- Reading directions for each lecture.
- Lecture slides, and other material.
- Material for the project.
- Intranet with material (password protected).

What we expect from you

- Basic course in databases, e.g.,
  - Relational data model / relational databases
  - SQL (and perhaps relational algebra)
- Basic course in algorithms and data structures, e.g.,
  - Search trees
  - Sorting algorithms
  - Hashing
  - Big-O notation
  - Basic algorithm analysis

Next: Course overview

(different slide set)

Recap part of the lecture

Goal: Refresh your memory, and agree on common terminology.

- Basic concepts in relational data model, like attribute, schemas, keys etc.
- Relational algebra
- Basic operations, like set operations, joins, selection etc.
- Bags (multisets)
- More operations, e.g., duplicate removal, grouping
- Indexes
- Transactions

What is a relational database?

All major general purpose DBMSs are based on the so-called **relational data model**. This means that all data is stored in a number of tables (with named columns), such as:

accountNo	*balance*	*type*
12345	1000.00	`savings`
67890	2846.92	`checking`
32178	-3210.00	`loan`
...	...	...

For historical, mathematical reasons such tables are referred to as **relations**.

SQL is a query language for relational databases and is based on **relational algebra**.

Relation instance

A **relation instance** is a two-dimensional table of data.

The order of rows and columns can be exchanged, and it is still the same relation instance.

An **attribute** is the name of a column in a relation instance.

Example:

title	*year*	*length*	*filmType*
Star Wars	1977	124	color
Mighty Ducks	1991	104	color
Wayne's World	1992	95	color

Tuple

A **tuple** is a row in a table. The values in the row are called components.

A relation (instance) can be seen as a set of tuples.

Example:

Movies

title	*year*	*length*	*filmType*
Star Wars	1977	124	color
Mighty Ducks	1991	104	color
Wayne's World	1992	95	color

Schema

A **schema** is a description of a class of relation instances with the same attributes. It consists of a name for the relation and a set of attributes.

(It may also contain data types for attributes.)

Example:

title	*year*	*length*	*filmType*
Star Wars	1977	124	color
Mighty Ducks	1991	104	color
Wayne's World	1992	95	color

Schema: Movies(title, year, length, filmType).

A set of schemas is called a **database schema**.

“Relation”

The word **relation** can refer both to a particular relation instance and to a schema (an “abstract relation instance”).

Saying “the relation R” is similar to saying “the integer x”. Depending on the context we may or may not be thinking of a concrete value/instance.

Keys of a relation

A **key** for a relation is a set of its attributes that satisfy:

- **Uniqueness.** The values of the attributes uniquely identify a tuple. (This should hold for all possible instances of the relation.)
- **Minimality.** No proper subset of the attributes has the uniqueness property.

If uniqueness is satisfied (but not necessarily minimality) the attributes are said to form a **superkey**.

title	*year*	*length*	*filmType*	*studioName*	*starName*
Star Wars	1977	124	color	Fox	Carrie Fisher
Star Wars	1977	124	color	Fox	Mark Hamill
Star Wars	1977	124	color	Fox	Harrison Ford
Mighty Ducks	1991	104	color	Disney	Emilio Estevez

Key: {*title, year, starName*} **Superkey**: {*title, year, length, starName*}

Relational algebra and SQL examples

Relational algebra is notation for expressing queries on relations.

The rest of the recap is about:

- Basic operations in relational algebra:
  - set operations (e.g. union)
  - selection and projection
  - join
- Bags (multisets). Why they are used and what the consequence is.
- More operations, e.g., duplicate removal, grouping
- Indexes
- Transactions

How many of the above red words do you recognize?

How many of them are you able to define now?

Set operations

Operations on two sets R and S, where R and S must have the same set of attributes. We have the three set operations:

- Union, $R \cup S$,
- Intersection, $R \cap S$, and
- Difference, $R \backslash S$.

Example:

$\underline{R}$

title	*year*	*filmType*
Star Wars	1977	color
Mighty Ducks	1991	color
Wayne's World	1992	color

$\underline{S}$

title	*year*	*filmType*
Star Wars	1999	color
Mighty Ducks	1991	color
Star Wars	2002	color

Projection

A **projection** of relation R on attributes $A_1, \ldots, A_n$ is denoted by

$$\pi_{A_1,\ldots,A_n}(R)$$

and is the relation R restricted to columns for attributes $A_1, \ldots, A_n$.

title	*year*	*length*	*filmType*	*studioName*	*starName*
Star Wars	1977	124	color	Fox	Carrie Fisher
Star Wars	1977	124	color	Fox	Mark Hamill
Star Wars	1977	124	color	Fox	Harrison Ford
Mighty Ducks	1991	104	color	Disney	Emilio Estevez

$\pi_{\text{title, length, studioName}}(\text{Movies}) =$

title	*length*	*studioName*
Star Wars	124	Fox
Mighty Ducks	104	Disney

Selection

A **selection** of tuples satisfying condition C from relation R is denoted by

$$\sigma_C(R)$$

and is the relation R restricted to tuples for which condition C is satisfied.

C can be any boolean expression, i.e. it may involve multiple attributes, constants, AND, OR, and NOT.

Example:

Movies

title	*year*	*filmType*
Star Wars	1977	color
Mighty Ducks	1991	color
Wayne's World	1992	color

$\sigma_{\text{year}>1981}(\text{Movies}) =$

title	*year*	*filmType*
Mighty Ducks	1991	color
Wayne's World	1992	color

Join (natural)

Natural-Join or Inner-Join:

Let R and S be two relations with attributes $R_1, \ldots, R_n$ and $S_1, \ldots, S_m$ respectively. The join of relations R and S, denoted

$$R \bowtie S$$

has attributes $\{R_1, \ldots, R_n\} \cup \{S_1, \ldots, S_m\}$.
If $r \in R$ and $s \in S$ agree on attributes $\{R_1, \ldots, R_n\} \cap \{S_1, \ldots, S_m\}$ then the joint tuple for r and s is in $R \bowtie S$.

There are other types of join, e.g., Theta-Join and Outer-Join.

Join example

Movies

title	*year*	*length*	*studioN.*
Star Wars	1977	124	Fox
Mighty D.	1991	104	Disney
Wayne's W.	1992	95	Param.

StarsIn

title	*year*	*starN.*
Star Wars	1977	Carrie F.
Star Wars	1977	Mark H.
Star Wars	1977	Harrison F.
Mighty D.	1991	Emilio E.
Wayne's W.	1992	Dana C.
Wayne's W.	1992	Mike M.

Movies $\bowtie$ StarsIn =

title	*year*	*length*	*studioN.*	*starN.*
Star Wars	1977	124	Fox	Carrie F.
Star Wars	1977	124	Fox	Mark H.
Star Wars	1977	124	Fox	Harrison F.
Mighty D.	1991	104	Disney	Emilio E.
Wayne's W.	1992	95	Param.	Dana C.
Wayne's W.	1992	95	Param.	Mike M.

Bags

Relational algebra is an algebra on sets, but most database systems do not (only) use sets, they (also) use bags.

A **bag** or a multiset, is a set where elements may appear more than once. (E.g., in a relation there may be two or more identical rows.)

The motivation for using bags instead of sets is that some operations can be implemented faster. E.g.,

- union
- projection

Operations on bags vs. sets

Some examples of the difference between operations on bags and sets.

- $R \cup S$: All rows in R and S, even if they appear in both or if they appear more than once in R or in S.
- $R \cap S$: if tuple t appears n times in R and m times in S, then it appears $\min(n, m)$ times in $R \cap S$.
- $\pi_{A_1,\ldots,A_n}(R)$ (projection): All tuples in R also appear in $\pi_{A_1,\ldots,A_n}(R)$, even if the rows become identical when some columns are removed.

Movies1

title	*year*	*filmType*
Star Wars	1977	color
Mighty Ducks	1991	color
Wayne's World	1992	color

Movies2

title	*year*	*filmType*
Star Wars	1999	color
Mighty Ducks	1991	color
Star Wars	2002	color

More operations

Other useful relational operations often used in languages like SQL:

- Duplicate elimination: When bags are used it is useful to be able to get rid of duplicates. $\delta(R)$
- Aggregation operators: E.g., sum, average, maximum in a column. $\gamma_{\mathtt{OP}(A)}(R)$, where OP is e.g., max.
- Grouping (not described in RG): Divide a relation up into groups of tuples depending on the values in one or more attributes. Used together with aggregation. $\gamma_{A_1,\ldots,A_n,\mathtt{OP}(A)}(R)$.
- Extended projection: Creation of new columns from existing columns by performing some kind of computation.

SQL

SQL is a language that can be used for expressing queries on relations. It is based on a "mixture" of relational algebra for sets and bags.

Some SQL examples:

- $\texttt{SELECT } A_1, \ldots, A_n \texttt{ FROM } R$ means $\pi_{A_1,\ldots,A_n}(R)$.
- $\texttt{SELECT * FROM } R \texttt{ WHERE } C$ means $\sigma_C(R)$.
- $R \texttt{ UNION } S$ means $R \cup S$ (set union, for bag union use `UNION ALL`).
- $R \texttt{ EXCEPT } S$ means $R \setminus S$.
- $R \texttt{ NATURAL JOIN } S$ means $R \bowtie S$.
- $\texttt{SELECT DISTINCT * FROM } R$ means $\delta(R)$.
- $\texttt{SELECT A, OP}(B) \texttt{ FROM } R \texttt{ GROUP BY } A$ means $\gamma_{A,\texttt{OP}(B)}(R)$

SQL update operations

SQL also supports the creation and modification of relations.

Some SQL examples:

- `CREATE TABLE` R `(<schema description>)`
- `INSERT INTO` R `VALUES` $(v_1, \ldots, v_n)$.
- `DELETE FROM` R `WHERE` C.
- `UPDATE` R `SET` $A = v$ `WHERE` C.

Selective queries

Consider the selection query:

```
SELECT *
FROM R
WHERE <condition>
```

- If we have to report 80% of the tuples in R, it makes sense to do a full table scan.
- On the other hand, if the query is very **selective**, and returns just a small percentage of the tuples we might hope to do better, by using an **index.**

Indexes

To be able to quickly find the first tuple with a specific value for an attribute, the DBMS may build an **index** on that attribute.

A database index is similar to an index in the back of a book:

1. For every piece of data you might be interested in (e.g., the attribute year=1977), the index says where to find it.
2. The index itself is organized such that one can quickly do the lookup.

Indexes

Some indexes are efficient for both **point queries** (year=1977) and **range queries** (1985<year<1999), while others only support efficient point queries.

Indexes are also used by the DBMS to speed up other operations, e.g., **join operations** are *sometimes* considerably faster when the join attributes are indexed.

In most DBMSs we can specify what indexes should be created, e.g.:

- `CREATE INDEX` I `ON` $R(A)$

Transactions

One or more updates in a database can be grouped into something called a **transaction**. This is a way to ensure correct updates of the database.

Ideal transactions are said to meet the **ACID** test:

- **A**tomicity – the all-or-nothing execution of transactions.
- **C**onsistency – transactions preserve database constraints.
- **I**solation – the appearance that transactions are executed one by one.
- **D**urability – the effect of a transaction is never lost once it has completed.

A good DBMSs should fully implement **A**, **C** and **D**, and will allow the user to specify the extent to which **I** should hold (for efficiency reasons).

However, **I** always applies to any *single* SQL statement in a transaction.

Summary of recap

This part of the lecture was about:

- the relational data model
- relational algebra
- relational algebra on bags
- some examples in SQL
- Properties of DBMSs

These concepts will underlie much of the course.