Physical DB design and tuning: outline

• Designing the Physical Database Schema
  – Tables, indexes, logical schema

• Database Tuning
  – Index Tuning
  – Query Tuning
  – Transaction Tuning
  – Logical Schema Tuning

• DBMS Tuning
Relational DB design

• Database design phases:
  (a) Requirement Analysis,
  (b) Conceptual design
  (c) Logical design
  (d) Physical design

• Physical Design Goal: definition of appropriate storage structures for a specific DBMS, to ensure the application performance desired
Physical design: needed information

• Logical relational schema and integrity constraints
• Statistics on data (table size and attribute values)
• Workload
  – Critical queries and their frequency of use
  – Critical updates and their frequency of use
  – Performance expected for critical operations
• Knowledge about the DBMS
  – Data organizations,
  – Indexes and
  – Query processing techniques supported
Workload definition

• Critical operations: those performed more frequently and for which a short execution time is expected.
Workload definition

• For each critical query
  – Relations used
  – Attributes of the result
  – Attributes used in conditions (restrictions and joins)
  – Selectivity factor of the conditions

• For each critical update
  – Type (INSERT/DELETE/UPDATE)
  – Attributes used in conditions (restrictions and joins)
  – Selectivity factor of the conditions
  – Attributes that are updated
The use of ISUD table

**Insert, Select, Update, Delete**

<---- Employee Attributes ------>

<table>
<thead>
<tr>
<th>APPLICATION</th>
<th>FREQUENCY</th>
<th>%DATI</th>
<th>NAME</th>
<th>SALARY</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payroll</td>
<td>monthly</td>
<td>100</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>NewEmployee</td>
<td>quarterly</td>
<td>0.1</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>DeleteEmployee</td>
<td>quarterly</td>
<td>0.1</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>UpdateSalary</td>
<td>monthly</td>
<td>10</td>
<td>S</td>
<td>U</td>
<td></td>
</tr>
</tbody>
</table>
Decisions to be taken

• The physical organization of relations
• Indexes
• Index type
• ... Logical schema transformation to improve performance
Decisions for relations and indexes

• Storage structures for relations:
  – heap (small data set, scan operations, use of indexes)
  – sequential (sorted static data)
  – hash (key equality search), usually static
  – tree (index sequential) (key equality and range search)

• Choice of secondary index, considering that
  – they are extremely useful
  – slow down the updated of the index keys
  – require memory
How to choose indexes

• Use a DBMS application, such as DB2 Design Advisor, SQL Server DB Tuning Advisor, Oracle Access Advisor.
How to choose indexes

• Tips: don’t use indexes
  • on small relations,
  • on frequently modified attributes,
  • on non selective attributes (queries which returns ≥ 15% of data)
  • on attributes with values long string

• Define indexes on primary and foreign keys

• Consider the definition of indexes on attributes used on queries which requires sorting: ORDER BY, GROUP BY, DISTINCT, Set operations
How to choose indexes (cont.)

• Evaluate the convenience of indexes on attributes that can be used to generate index-only plans.

• Evaluate the convenience of indexes on selective attributes in the WHERE:
  – hash indexes, for equality search
  – $B^+$tree, for equality and range search, possibly clustered
  – multi-attributes (composite), for conjunctive conditions

• to improve joins with IndexNestedLoop or MergeJoin, grouping, sorting, duplicate elimination.

• Attention to disjunctive conditions
How to choose indexes (cont.)

– Local view: indexes useful for one query (simple)

– Global view: indexes useful for a workload

– Index subsumption: some indexes may provide similar benefit
  • An index $Idx_1(a, b, c)$ can replace indexes $Idx_2(a, b)$ and $Idx_3(a)$

– Index merging: two indexes supporting two different queries can be merged into one index supporting both queries.

– When there are a lot of data to load, create indexes after data loading
How to choose indexes: the global view

1. Identify critical queries
2. Create possible indexes to tune single queries
3. From set of all indexes remove subsumed indexes
4. Merge indexes where possible indexes
5. Evaluate benefit/cost ratio for remaining indexes
   (need to consider frequency of queries/index usage)
6. Pick optimal index configuration satisfying storage constraints.
Primary organizations and secondary indexes

CREATE [UNIQUE...] INDEX Nome [USING {BTREE | HASH | RTREE}] ON Table (Attributes+Ord)
Clustering indexes
Bitmap indexes

<table>
<thead>
<tr>
<th>Lastname</th>
<th>Firstname</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Müller</td>
<td>Heinrich</td>
<td>GER</td>
</tr>
<tr>
<td>Miller</td>
<td>Henry</td>
<td>UK</td>
</tr>
<tr>
<td>Magritte</td>
<td>René</td>
<td>FRA</td>
</tr>
<tr>
<td>Smith</td>
<td>John</td>
<td>US</td>
</tr>
<tr>
<td>Starkey</td>
<td>Richard</td>
<td>UK</td>
</tr>
<tr>
<td>Weiser</td>
<td>Bud</td>
<td>US</td>
</tr>
<tr>
<td>Röder</td>
<td>Hasso</td>
<td>GER</td>
</tr>
<tr>
<td>Hugo</td>
<td>Victor</td>
<td>FRA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>FRA</th>
<th>GER</th>
<th>UK</th>
<th>US</th>
</tr>
</thead>
<tbody>
<tr>
<td>Müller</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Miller</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Magritte</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Smith</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Starkey</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Weiser</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Röder</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hugo</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

...
DB schema for the examples

Lecturers(PkLecturer INT, Name VARCHAR(20), ResearchArea VARCHAR(1), Salary INT, Position VARCHAR(10), FkDepartment INT)
Departments(PkDepartment INT, Name VARCHAR(20), City VARCHAR(10))

<table>
<thead>
<tr>
<th></th>
<th>Departments</th>
<th>Lecturers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nrec</td>
<td>300</td>
<td>10 000</td>
</tr>
<tr>
<td>Npag</td>
<td>30</td>
<td>1 200</td>
</tr>
<tr>
<td>Nkey(IdxSalary)</td>
<td>50 (min=40, max=160)</td>
<td></td>
</tr>
<tr>
<td>Nkey(IdxCity)</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Primary organization definitions in SQL

CREATE TABLE Table (Attributes) < physical aspects >;

CREATE TABLE Table (Attributes + keys) ORGANIZED INDEX; (ORACLE)
CREATE TABLE Table (Attributes + keys) ORGANIZE BY DIMENSIONS (Att); (DB2)
The definition of indexes

Do not index!
Few records in this table

Do not index!
Possibly too many updates

The index is created implicitly on a PK
The definition of (multi-attributes) indexes

How many indexes on a Table?

A secondary index on **Position** or on **Salary**?
A secondary index on **Position** and another on **Salary**?
A secondary index on `<Position, Salary>`?

An index on **Position, ResearchArea** or **ResearchArea Position** to speed up GBY.
Better on **ResearchArea Position** because it also returns result sorted

How many indexes use the DBMS for AND?

```
SELECT Name
FROM Lecturers
WHERE Position = 'P'
  AND Salary BETWEEN 50 AND 60
```

```
SELECT ResearchArea,Position
  COUNT(*)
FROM Lecturers
GROUP BY Position, ResearchArea
ORDER BY ResearchArea
```
The creation of clustered index

With a not very selective predicate **Salary** > 70, a clustered index may still be useful

If there are a few lecturers with a **Salary** = 70, a clustered index on **Salary** can be more useful than an unclustered one

A clustered index on **FkDepartment** can be useful here
The creation of indexes for index-only plans

For Index-Only plans clustered indexes are not required.

Index on su
FkDepartment

Index on Salary

Index on FkDepartment for INL

Index on <FkDepartment,Salary>

SELECT DISTINCT FkDepartment
FROM Lecturers;

SELECT Salary, COUNT(*)
FROM Lecturers
GROUP BY Salary;

SELECT D.Name
FROM Lecturers L, Departments D
WHERE FkDepartment=PkDepartment;

SELECT FkDepartment, MIN(Salary)
FROM Lecturers
GROUP BY FkDepartment;
Index-only plans

Some DBMSs allow the definition of indexes on some attributes, and to include also others which are not part of the index key.

**DB2:** `CREATE UNIQUE INDEX Name ON Table (Attrs) INCLUDE (OtherAttrs);`

**SQL Server:** the clause `UNIQUE` is optional

Index on `FkDepartment` INCLUDE `Salary`

```
SELECT FkDepartment, MIN(Salary)
FROM Lecturers
GROUP BY FkDepartment;
```
Concluding remarks

• The implementation and maintenance of a database application to meet specific performance requirements is a very complex task

• Table organizations and index selections are fundamental to improve query performance, but ...
  – Indexes must be really useful because of their memory and update costs
  – An index should be useful for different queries

• Clustered indexes are very useful, but only one for table can be defined

• The order of attributes in multi-attribute (composite) indexes is important
Database tuning: what is the goal?

• Improve performance
• Database tuning mostly refers to query (or DB applications) performance.
What can be tuned?

- **Applications**
  - Queries, Transactions, ...

- **DBMS**
  - System configuration, DB schema, storage structures, ...

- **OS**
  - System parameters and configuration for IO, ...

- **HW**
  - Components for CPU, main memory, hard disks, backup solutions, network, ...

Here the focus is on Database Tuning
Needed knowledge

Applications
How is the DB used and what are the tuning goals?

DBMS
What possibilities of optimization does it provide?

OS
How does it manage HW resources and services by the overall system?

HW
What are suitable HW components to support the performance requirements?

Fully efficient DB Tuning requires deep knowledge about...

Schallehn: Database Tuning and Self-Tuning 2012
<table>
<thead>
<tr>
<th>Role</th>
<th>When</th>
<th>Knowledge Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB and application designers</td>
<td>During DB development (physical DB design) and initial testing.</td>
<td>DB designers must have knowledge about applications, and good knowledge about the DBMS, but may be only fair to no knowledge about OS and HW.</td>
</tr>
<tr>
<td>DB administrators (DBA)</td>
<td>During ongoing DBMS maintenance. Adjustment to changing requirements, data volume, new HW.</td>
<td>DBA have knowledge about DBMS, OS, and HW. And about applications? DBA knowledge about applications depends on the given organizational structure.</td>
</tr>
<tr>
<td>DB experts (consultants, in-house experts)</td>
<td>During system re-design, troubleshooting or fire fighting (emergency actions)</td>
<td>DB consultants usually have strong knowledge about DBMS, OS and HW, but have little knowledge about current applications.</td>
</tr>
</tbody>
</table>
Database tuning as a process

Overall system continuously changes
Data volume, # of user, # of queries, usage patterns, hardware, etc.

Requirements may change

Identify Existing Problem

Monitor system behavior and identify cause of problem

Apply changes to solve problem

Problem Solved

Current performance requirements not fulfilled

Observe and measure relevant quantities, e.g. queries time.

Adjust system parameters, storage structures etc.

Schallehn: Database Tuning and Self-Tuning 2012
Basic principles: controlling trade-offs

Database tuning very often is a process of decision about costs for a solution compared to its benefits.

**Costs:** monetary costs (for HW, SW), working hours or more technical costs (resource consumption, impact on other aspects)

**Benefits:** improved performance (monetary effects most often not easily quantifiable)

Examples

- Adding indexes -> **benefit:** better query performance
  - **costs:** more disk memory, more update time

- Denormalization -> **benefit:** better query performance
  - **costs:** need to control redundancy within tables

- Replace disk by RAID -> **benefit:** better I/O performance
  - **costs:** HW costs

Schallehn: Database Tuning and Self-Tuning 2012
Basic principles: Pareto principle

80/20 Rule: by applying 20% of the effort one can achieve 80% of the desired effects.

Consequences for DB tuning

100% effect = full optimized system

Full optimized system probably beyond necessary requirements.

“a little bit of DB tuning can help a lot”

Hence, one does not need to be an expert on all levels of the system to be able to implement a reasonable solution...

Schallehn: Database Tuning and Self-Tuning 2012
Database tuning

When a database has unexpected bad performances we must revise:

- **Physical Design**: the selection of indexes or their type, looking at the access plans generated by the optimizer

- **Query and Transaction Definitions**

- **DB Logical Design**

- **DBMS**: buffer and page size, disk use, log management.

- **Hardware**: number of CPU, disk types.
To begin

• Select the queries with low performance (either the critical one or those which do not satisfy the users)

• Analyze physical query plans looking at
  – physical operators for tables
  – sorting of intermediate results
  – physical operators used in the query plans
  – physical operators for the logical operators
Index tuning

One of the most often applied tuning measures

**Great benefits** with little effort (if applied correctly).

**Strong support** within all available DBMS
   (index structures, index usage controlled by optimizer)

**Storage cost** of additional disk memory most often acceptable.

**Cost** for index updates.

**Cost** for locking overhead and lock conflicts.
Query tuning

• SELECT with OR condition are rewritten as one predicate IN, or with UNION of SELECT
• SELECT with AND of predicates are rewritten as a condition with BETWEEN
• Rewrite SUBQUERIES as join
• Eliminate useless DISTINCT and ORDER BY from SELECT or SUBSELECTS
• Avoid the definition of temporary views with grouping and aggregations
• Avoid aggregation functions in SUBQUERIES
Query tuning (cont.)

Avoid expressions with index attributes (e.g. Salary*2=100)

Avoid useless HAVING or GROUP BY

\[
\sigma_\phi (A \gamma_F (E)) \equiv A \gamma_F (\sigma_\phi (E))
\]

SELECT Position, MIN(Salary)  
FROM Lectures  
GROUP BY Position  
HAVING Position IN(‘P1’, ‘P2’)

SELECT MIN(Salary)  
FROM Lectures  
GROUP BY Position  
HAVING Position = ‘P1’
Transactions tuning

• Do not block data during db loading, as well as during read-only transactions
• Split complex transactions in smaller ones
• Select the right block granularity, if possible (long T, table; medium T, page; short T, record)
• Select the right isolation level among those provided by SQL
  – SET TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
Isolation levels in SQL

READ UNCOMMITTED, record READ only without locks

Problem: dirty read (read data updated by other active T)

Account (No INTEGER PRIMARY KEY, Name CHAR(30), Balance FLOAT);

```
-- T1.begin:
UPDATE Account
SET Balance = Balance - 200.00
WHERE No = 123;
```

```
-- T2.begin: RU
SELECT AVG(Balance)FROM Account; COMMIT;
```
Isolation levels in SQL

**READ COMMITTED**, **shared read** locks are released immediately, **exclusive** locks until the T commit

**Problem**: avoid dirty read, but unrepeatable reads or loss of updates

```sql
-- T1.begin:
UPDATE Account
SET Balance = Balance - 200.00
WHERE No = 123;
COMMIT;

-- T2.begin:
RC
SELECT AVG(Balance)
FROM Account;
```
Isolation levels in SQL

**REPEATABLE READ**, shared and exclusive locks on records until the end of the transaction

**Problem**: avoid the previous problems, but not the “phantom records” problem:

---

**T1.begin:**

```sql
INSERT INTO Account
VALUES(1233,"xx",200.00);
COMMIT;
```

**T2.begin:**

```sql
RR
SELECT AVG(Balance)
FROM Account;
COMMIT;
```

---
Isolation levels in sql (cont.)

- SERIALIZABLE, multi-granularity locks: tables read by a T cannot be updated.
- Good, but the number of transactions which can be executed concurrently is considerably reduced.
DBMS isolation levels

• Commercial DBMS may
  – provide some isolation levels only,
  – not have the same isolation level by default
  – have other isolation levels (e.g. SNAPSHOT)
Logical schema tuning

• Types of logical schema restructuring:
  – Vertical Partitioning.
  – Horizontal Partitioning
  – Denormalization

• Unlike changes to the physical schema (physical independence), changes to the logical schema (schema evolution) require views creation for the logical independence.
Logical schema tuning

• Partitioning: splitting a table for performance
  – Horizontal: on a property
  – Vertical: R1(pk, Name, Surname) R2(pk, Address, …)

• Normalization: divide Students from Exams to avoid anomalies

• Denormalization: store Students and Exams into one table:
  – increases update time but makes join faster
Vertical partitioning (projections)

**Students**

<table>
<thead>
<tr>
<th>Name</th>
<th>StudentNo</th>
<th>City</th>
<th>BirthYear</th>
<th>BDegree</th>
<th>University</th>
</tr>
</thead>
</table>

**Exams**

<table>
<thead>
<tr>
<th>PkE</th>
<th>Course</th>
<th>StudentNo</th>
<th>Master</th>
<th>Date</th>
<th>Other</th>
</tr>
</thead>
</table>

**Critical Query:**
Find the number of exams passed and the number of students who have done the test by course, and by academic year.

**ExamForAnalysis**

<table>
<thead>
<tr>
<th>PkE</th>
<th>Course</th>
<th>Master</th>
<th>Date</th>
</tr>
</thead>
</table>

**ExamsOther**

<table>
<thead>
<tr>
<th>PkE</th>
<th>StudentNo</th>
<th>Other</th>
</tr>
</thead>
</table>

The decomposition must preserve data...
Horizontal partitioning (selections)

Exams

<table>
<thead>
<tr>
<th>PKE</th>
<th>Course</th>
<th>StudentNo</th>
<th>Degree</th>
<th>Date</th>
<th>...</th>
</tr>
</thead>
</table>

MasterExam

| FkM | FkE | ... | PKM | Title | President | ... |

Critical Query:
For a study program with title X and a course with less than 5 exams passed, find the number of exams, by course, and academic year

MasterXExams

| PKE | Course | StudentNo | Degree | Date | ... |

MasterYExams

| PKE | Course | StudentNo | Degree | Date | ... | ... |
Denormalization (attribute replication)

Critical Query:
For a student number N, find the student name, the master program title and the grade of exams passed

Finally: **Schema fusion** of relations (1:1) and **View materialization**
DBMS tuning

• Tune the transaction manager (log management and storage, checkpoint frequency, dump frequency)
• Tune the buffer size (as well as interactions with the operating system)
• Disk management (allocation of memory for tablespaces, filling factor for pages and files, number of preloaded pages, accesses to files)
• Use of distributed and parallel databases
Database self-tuning

Applications
Knowledge about from analyzing queries, TXNs, schema, etc.

DBMS
Naturally, it knows best about its functionality and tuning options

DBMS itself is best Tuning Expert!

OS
Knowledge about encoded in platform specific code + runtime inf via OS interfaces

HW
Knowledge about encoded in platform specific code + runtime inf via OS interfaces

Schallehn: Database Tuning and Self-Tuning 2012
Database self-tuning

• Databases are getting better and better at this
• This is clearly the way to go
• But there will be still space for a good DBA
Exercise

• Tables:
  – Sales(Date, FKShop, FKCust, FKProd, UnitPrice, Q, TotPrice)
  – Shops(PKShop, Name, City, Region, State)
  – Customer(PKCust, Nome, FamName, City, Region, State, Income)
  – Products(PKProd, Name, SubCategory, Category, Price)
Exercise

• Sales: 100.000.000, 1.000.000; Shops: 500, 2; Customers: 100.000, 1.000; Products: 10.000, 100

SELECT Sh.Region, Month(S.Date),Sum(TotPrice)
FROM Sales S join Shops Sh on FKShops=PKShops
GROUP BY Sh.Region, Month(S.Date)

• Propose a primary organization based on this query

• Compute the cost of an optimal access plan based on this organization

• Add a condition: WHERE 1/1/2017 < Date