Introduction to Relational Databases

- Bachelor CS, Lille 1 University
- Nov. 16, 2011 (lecture 11/12)
- Today's lecturer: C. Kuttler
- Topic: Introduction to Database Tuning

Overview of today's lecture

- Introduction
- Query optimization
  - An example
- Tuning
  - Workload
  - Access structures
  - Operations (queries & modifications)
  - Modification of logical schema
  - Parameters of Architecture

Introduction

- Database design process
  - "logic design": from the conceptual schema, we derive:
    - a logical schema
    - the necessary external schemas
  - "physical design": starting from the logical schema, we derive:
    - a physical schema (access structures)
    - an optimized logical schema

Introduction

- Logic design
  - goal: design the database such that it avoids anomalies (normalization)
  - systematic algorithm exists for normalization
- Physical design
  - goal: all operations on the database are efficient
  - difficult to address systematically

Introduction

- Goal of physical design
  - is the performance
  - intervention on parameters that influence it
- Parameters with impact on the performance
  - organization of the files and access structures
  - logical schema
  - operations (queries and transactions)
  - parameters of the architecture (buffer, disks, etc)
- These aspects are difficult to plan

Introduction

- Typical approach to performance tuning
  - begin with the standard schema and the standard access structures
  - collect information on the actual use of the database, and evaluate the performance
  - based on these statistics, optimize the database's parameters ("tuning")
  - repeat this activity is periodically

Introduction

- In today's class
  - Overview of physical design and tuning
  - We will discuss the main parameters
  - We will describe guidelines
  - But we can't be exhaustive
- Starting point
  - Evaluation and optimization of a relational DBMS

Query optimization

- How a query is evaluated
  - The query is either interactively sent to the DBMS, or sent by an application
  - The DBMS analyzes the SQL code syntactically
  - The DBMS checks the access permission
  - The DBMS performs the query optimization

Query optimization

- Optimization process
  - Choice of an efficient strategy for the query's evaluation
- Execution plan for a query
  - fix the order in which the necessary algebraic operators are applied
  - fix the strategy to compute the result of each operator through the available access structures (data structures)

Query optimization

In order to perform the optimization

- Alternative execution plans are evaluated
- The optimizer uses statistics on the database's content
  - dimensions of the tables, dimensions of the records, dimensions of the indices, selectivity, etc
- The cost of each execution plan is estimated, based on these statistics
  - number of block accesses on the hard disk

Example: university database

```
CREATE TABLE teacher (
   t_id char(4) PRIMARY KEY,
   lastName varchar(20) NOT NULL,
   firstName varchar(20) NOT NULL,
   qualification char(15),
   school char(10) );


CREATE TABLE student (
   s_id integer PRIMARY KEY,
   lastName varchar(20) NOT NULL,
   firstName varchar(20) NOT NULL,
   program char(20),
   year integer,
   advisor char(4) REFERENCES teacher(t_id)
);
```

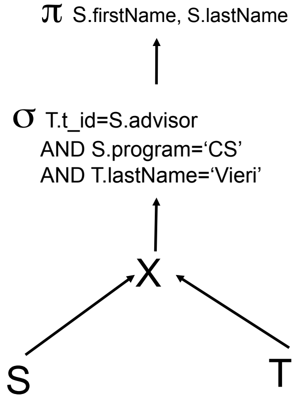

Our example

- Query: *"First and last names of Prof Vieri's students in Computer Science"*

```
SELECT student.firstName,
student.lastName
FROM teacher T, student S
WHERE T.t_id=S.advisor AND
      S.program = 'CS' AND
      T.lastName = 'Vieri';
```

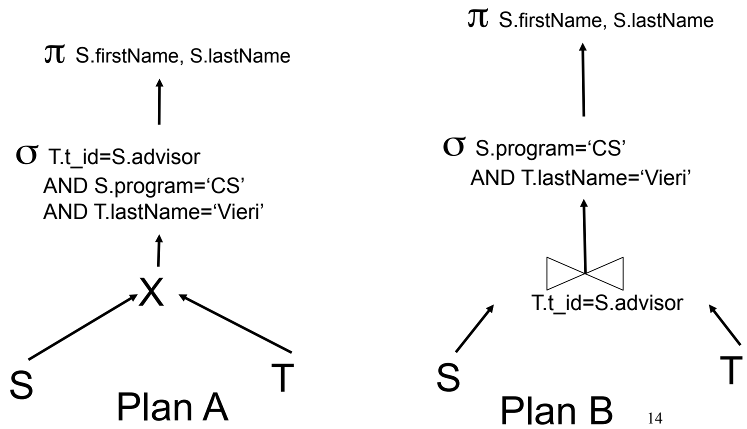
Operator tree

```
SELECT S.firstName,
       S.lastName
FROM teacher AS T,
     student AS S
WHERE T.t_id=S.advisor AND
      S.program = 'CS'
      AND T.lastName = 'Vieri';
```



$\pi_{\text{S.firstName, S.lastName}}\left(\sigma_{\text{T.t\_id=S.advisor AND S.program='CS' AND T.lastName='Vieri'}}(S \times T)\right)$

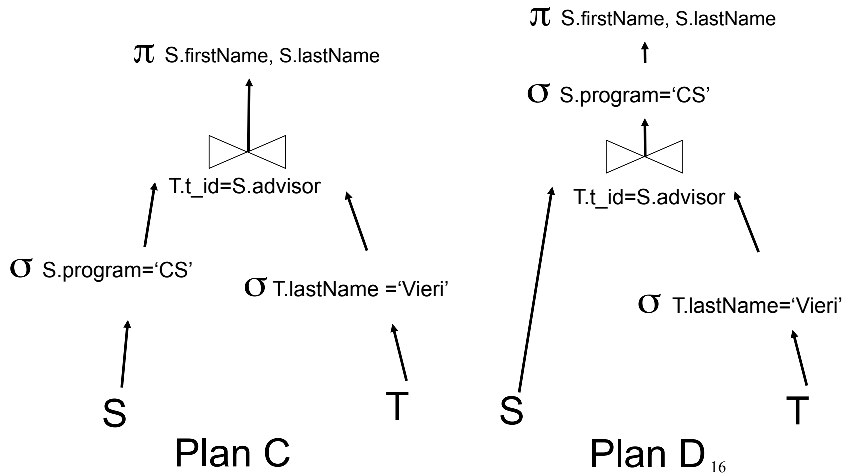
Same example – other trees



Which is better, A or B?

- Generally, the optimizer would prefer the plan B to the plan A
  - Cartesian products are expensive
- But in other cases
  - the operator tree alone isn't enough to know if one strategy is better than the other
  - the available access structures must be considered

Other execution plans



Which is better, C or D?

- Generally
  - plan C is better than plan D
  - but under certain conditions, plan D is better
- It depends on the strategy to evaluate the operators
  - in particular, the file organization
  - the data structures for access (indices)

Execution of algebraic operators

- Three main techniques
- Linear scan of the file
  - Inefficient, only applied to small files
- Access through index
  - Assumes the presence of indices (up to date)
- Temporary grouping
  - Creation of additional structures to group the tuples (example: order, hash table in main memory)

Execution of a selection: the data structure matters!

- Unordered file, no relevant index available
  - linear scan of the file
- File ordered by the attribute, no index
  - binary search in the file
- Index B+-tree on the attribute
  - B+-tree: data in external nodes, keys in internal nodes. Internal nodes have up to b children.
  - search in the index with complexity $O(\log_b n)$
- Hash index on the attribute
  - direct access through hash function, ideally

Execution of a join

- Basic strategy, without index
  - Embedded cycles
  - Pretty inefficient…
- Example: S JOIN T ON T.t_id=S.advisor
  - for each tuple of S
    - for each tuple of T
      - if T.t_id=S.advisor then output the resulting tuple

Execution of a join

- Embedded cycles with indices
  - Exploit an index on one of the join attributes
- Example: **S join T on S.advisor = T.t_id**
  - index on T.t_id
  - For each tuple of S
    - for each tuple of T such that S.advisor=T.t_id
      - include the tuple in result

index based access

Execution of a join

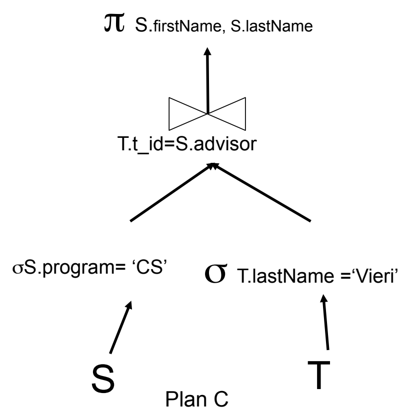
- "Sort-Merge" Join
  - idea: if both tables are ordered on the join attribute, the join is linear
- Strategy
  - Create an ordered copy of the table
  - Generate the result by a linear scan
  - Particularly efficient if table is already ordered

Execution of a join

- Hash Join
  - idea: hash on the join attribute for both tables
- Strategy
  - in main memory, build hash tables for both tables on the join attribute
  - Scan a table, and for each value, use the hash function to localize the corresponding bucket of tuples

Plan C

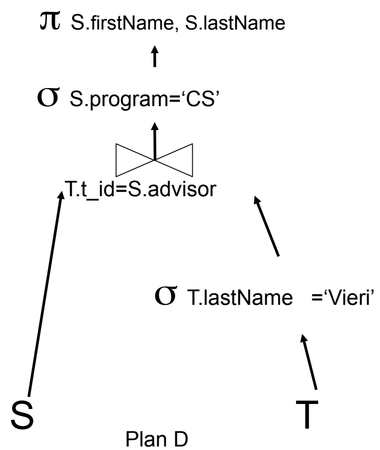
- Assume that:
  - no relevant indices
- Selection
  - linear scan
- Join
  - sort-merge
- Complete plan



Plan C

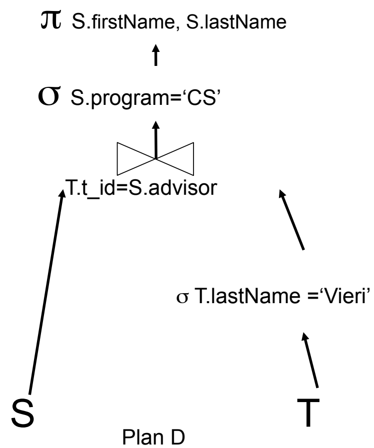
Plan D

- Suppose:
  - Hash index on T.lastName
  - Hash index on S. advisor
- Selection
  - hash
- Join
  - Cycles with index



Plan D

Complete plan D



Plan D

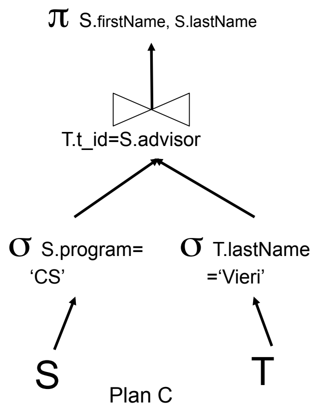
Calculated on the fly on the join result

Embedded program with index (use the secondary index on S.advisor)

Calculated using a hash index (materialization not required)

Plan C, other hypothesis

- Assume:
  - Hash index on T. lastName
  - Secondary index on S.program
- Not very useful for S
  - because the number of different programs is low
  - low selectivity of the index



Plan C

Optimization

- Top-end DBMS allow to consult the selected execution plan
- EXPLAIN command
  - Typical syntax: EXPLAIN <select>
  - Illustrates the execution plan and estimates its cost, by the optimizer
  - Available in PgSQL, MySQL, …

Tuning

- Typical scenario
  - After an initial phase, the actual use of a database is analyzed
  - The performance isn't good
  - An intervention is needed to improve the performance, by improving the parameters
- Starting point
  - Workload: usage statistics for the database

Workload

- Optimization isn't possible for all possible queries
- Only the most frequent and relevant operations are considered
- Workload
  - List of queries
  - List of updates
  - For each, expected performance (for example: <2s, or number of transactions per minute)

Tuning activities

1. Choice of access structures
   - Organization of files, indices, clustering
2. Interventions on operations
   - Re-structuring, isolation level
3. Interventions on the logical schema
   - Partitioning, aggregation, denormalization
4. Interventions on parameters of the architecture
   - buffer, disks, RAM

Access structures

- Main kind of intervention
  - Addition of indices
- Be careful
  - indices improve the performance
  - But slow down updates
  - Need disk space
  - A compromise is needed

Access structures

- Extreme case: read-only database
- Example
  - Archive of the immigrants in the USA between 1800 and 1900, with milions of tuples
  - Search by lastName, firstName and arrival year
- Many indices can be used
  - lastName, firstName, year, lastName and year, lastName and firstName

Access structures

- Guideline 1
  - It is only worth introducing an index, if it allows to improve the performance of more than one query of the workload
- Warning:
  - The optimizer doesn’t always manage to use an index
  - example: select * from Employees
    where AnnualSalary/12>3000
  - Check the execution plan before and after

Access structures

- Guideline 2
  - The attributes to intervene on are those that appear in joins and selections
  - For equality checks (ex: income=5000), hash indices are preferable
  - For conditions on intervals (ex: income>5000 and income<10000) B+-tree are preferable

Access structures

- Guideline 3
  - It is only worth introducing an index if the number of different values of an attribute is sufficienly high
- Example:
  - select * from employees
    where income = 10000
  - The index on income might not be useful, if many employees have the same income

Access structures

- Guideline 4
  - Pay attention to bottlenecks
- Example1:
  - Unordered relation with frequent insertions
  - The last block is a bottleneck
- Example 2:
  - Schema modifications (lock in writing into the catalogue)

Intervention on operations

- Two forms of interventions on operations
  - Restructuring queries
    - rewrite the query in a smarter way, such that its execution can be optimized
    - after some experience on the job, you should come back to this topic!
  - Choice of isolation level for transactions

Intervention on operations

- Example: restructuring a query
  - select * from Employee
    where yearlyIncome /12 > 3000
  - select * from Employee
    where yearlyIncome > 3000*12
- Other forms of restructuring
  - limit the level of embedding (difficult to optimize)
  - …

Intervention on operations

- Isolation level for transactions
  - The usual level is SERIALIZABLE
  - Often, READ COMMITTED is adequate
  - Generally, it is useful to separate interactive queries and updates

Isolation levels in SQL

Isolation level ↑

- Serializable (default) – avoids all anomalies
- Repeatable read – avoids all anomalies through phantom inserts
- Read committed – avoids "dirty reads"
- Read uncommitted – can present all anomalies

Modification of logical schema

- A normalized schema is not necessarily more efficient
  - normalization: formal method to reach "best" possible schema, that avoids most anomalies
- Four main forms of interventions
  - Partitionning tables
  - Aggregation of tables
  - Denormalization of tables
  - Adding redundant information

Modification of logical schema

- Be careful
  - When the logical schema is modified, some applications may not work correctly
- Two possible solutions
  - The modifications of the logical schema are decided very early (immediately after logical modeling)
  - Or, if possible, one creates an external schema equal to the old logical schema

Modification of logical schema

- Partitioning tables
  - Tables with many attributes can be split in two smaller tables
- Example: the table "student"
  - Primary key (s_id)
  - Personal data (firstName, lastName, s_id, social security number, address, family income etc.)
  - Academic attributes (program, year, advisor, internship, company etc.)

Modification of logical schema

- Can break the table into two tables
  - StudentPersonalData: s_id and all personal attributes
  - StudentUniversityData: s_id and all academic attributes
- Useful when
  - one rarely needs to access both kind of data
  - in those cases, a join will be needed

Modification of logical schema

- Warning
  - such restructuring must be done very early
  - views don't help
  - defining the view "student" as the join of the two tables, wouldn't improve the performance

Modification of logical schema

- Aggregation of tables: combining two tables into one
  - may avoid joins
- Example: student and internship
  - External key "s_id" of internship
  - If the internship data are frequently accessed, it makes sense to combine both tables into one
  - This avoids joins
  - The number of null values increases
  - Can define two views to present the schema

Modification of logical schema

- Denormalization of tables
  - Normalization avoids anomalies, but often enforces too many joins
- Example: teachers and numbers
  - Numbers(<u>number, teacher FK</u>)
  - If we often need to print the list of names and ids, then we could add the teacher's firstName to the table Number
  - However, this increases the complexity of updates

Modification of logical schema

- In this case
  - We generate (slight) update anomalies
  - example: each time that a teacher's lastName changes, need to intervene as well on teacher as on Numbers
  - To avoid to create inconsistent instances of the database, one needs to use transactions

Modification of logical schema

- Adding redundant information
  - Can sometimes avoid complex queries
- Example: number of exams that have been taken by a student
  - Can be derived by aggregation of the join between students and exams
  - Can be explicitly stored as an attribute of student
  - Forces to use transactions

Parameters of architecture

- Buffer
  - Increasing the buffer increases the "hit ratio"
  - Makes sense up to a certain limit
- Disks
  - Placing the files on several disks improves the performance
  - example: disk for the log (the log is a typical bottleneck)

Summary

- Introduction
- Query optimization
  - An example
- Tuning
  - Application workload
  - Access structures
  - Modification of logical schema
  - Architectural parameters