Relational Databases Overview

Themis Palpanas
Paris Descartes University

Data Intensive and Knowledge Oriented Systems

- thanks for slides to
  - Christoph Freytag

Relational Data Base Management Systems

Relational Model

- Relations are the main data structure
- Relation: Tuples, Attributes, Attribute Names
- Example
  - CUSTOMER (Name, Address, Account#)

Name	Address	Account#
Adams	Munich	003
George	London	001
....	....	....
Bond	World	007

  - ACCOUNT (Account#, Checking, Savings)
  - JOURNAL (Account#, Sequence#, Kind of TA, ...)

Query Processing

- User Languages:
  – SQL, QUEL, Embedded-SQL, 4GL
- *Data Definition* Language (DDL):
  – Create, Delete, Change of Relations
  – Authorization
- *Data Manipulation* Language (DML):
  – Access, Create, Change of Tuples
- Query (declarative):

```
SELECT Name, Address, Checking, Savings
FROM CUSTOMER C, ACCOUNT A
WHERE Name = "Bond" and C.Account# = A.Account#
```


Query Processing (cont.)

- Generate a "Query Execution Plan" (QEP):

```
FOR EACH c in CUSTOMER DO
IF k.Name = "Bond" THEN
    FOR EACH  a IN ACCOUNT DO
        IF a.Account# = c.Account# THEN
            Output ("Bond", c.Address,
                            a.Checking, a.Savings)
```

- QEP
  – Procedural Specification
  – Semantically "equivalent" to user query

Transactions (TA)

- Transaction: “Logical unit of work”

```
Begin_Transaction
    UPDATE ACCOUNT
        SET Savings = Savings + 1M
        SET Checking = Checking - 1M
    WHERE Account# = 007;
    INSERT JOURNAL <007, NNN, "Transfer", ...>
End_Transaction
```

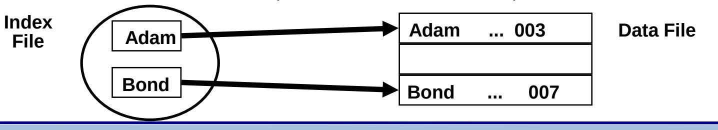
- Desirable properties (ACID Properties):
  - **Atomic** Execution
  - **Consistency**: Consistent DB state after successful updates
  - **Isolation**: No influence of result by concurrent executions
  - **Durability**: Updates are reflected in the database

Data Independence

- External Schema: Views
- Example: View Customer-Account

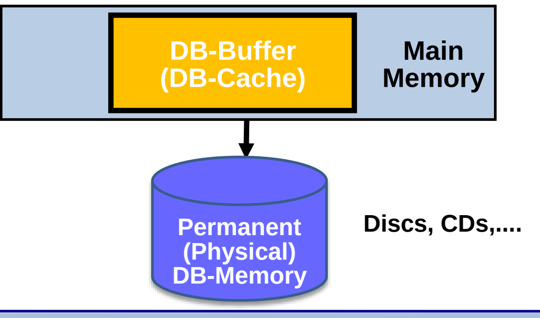
Name	Address	Checking	Savings
Adams	Boston	€ 1000	€ 300
...	...	...	...
Bond	World	€ 1M	€ 2M

- Conceptual Schema: Relations
  - CUSTOMER (Name, Address, Account#)
  - ACCOUNT (Account#, Checking, Savings)
- Internal Schema: Tables (Data Files, Index Files)

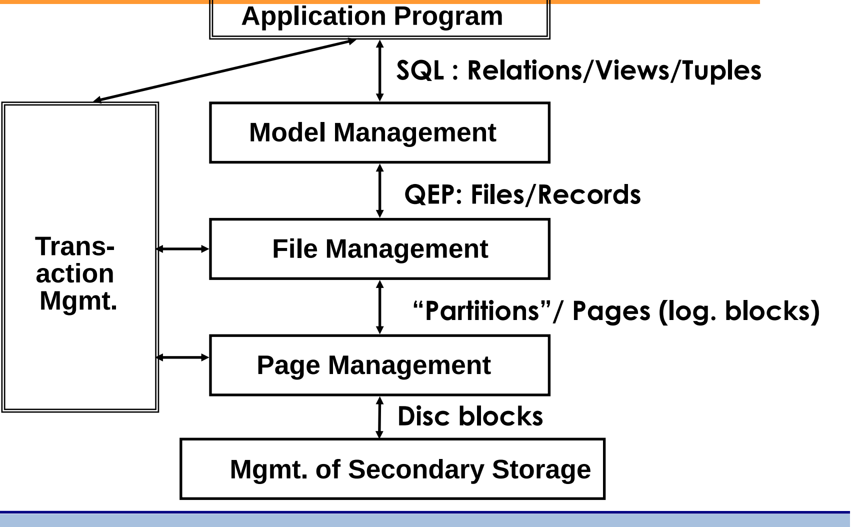


Persistence of Data

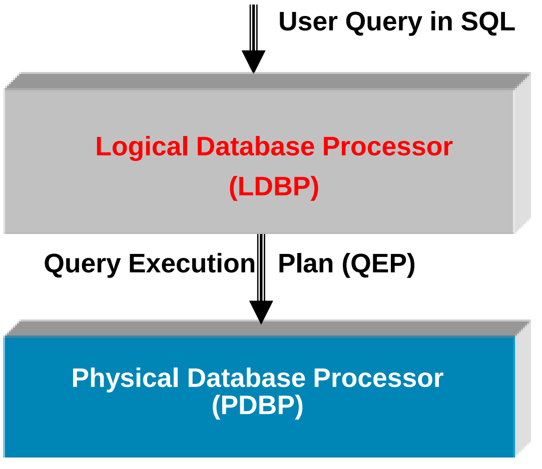
- Not all Data of DB fit into main memory
- “Data exchange” between MM and Discs necessary
- Optimization Problem: Which data when to load??
- DB together with MM und Discs is called **Standard DB**



Abstraction Levels

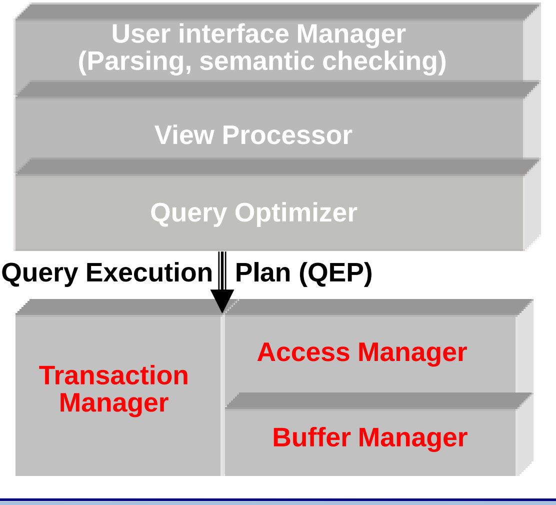


System Architecture (general)



1.11

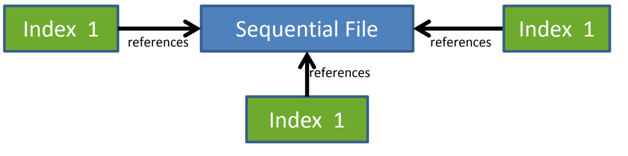
System Architecture (Details)



1.12

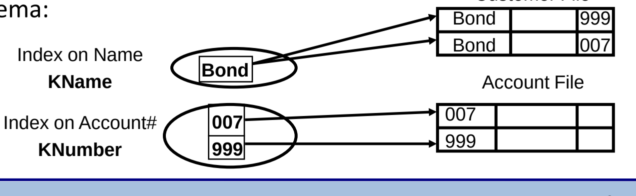
Physical Data Model/Access Manager

- Function/Goal:
  - Manage different access methods and their relationships
- Access Methods
  - Sequential File, B+-Tree, Hash based files; …
- Common physical data model:



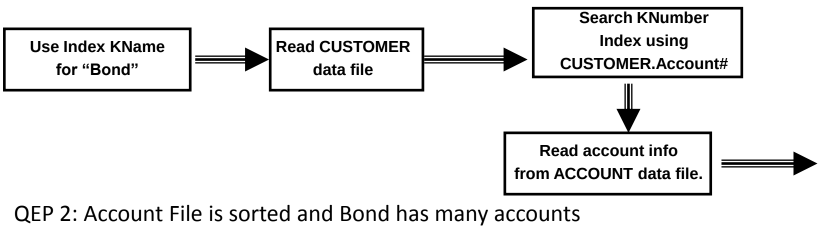
Query Optimizer

- Task: Generate Query Execution Plans (QEPs)
  - User query: Declarative
  - QEP: Procedural program
- Generate a QEP which is *efficient*
- Catalog info is important and necessary:
  - Info about internal schema
  - *Statistical Information*
- Example: Print Bond's account info
  - Internal schema:

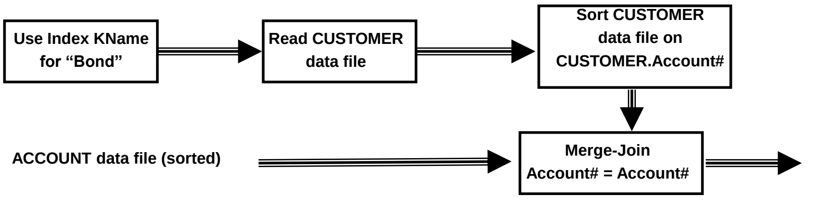


Query Optimizer (cont.)

QEP 1: Use both Indexes



QEP 2: Account File is sorted and Bond has many accounts



Optimization (cont.)

- Query Optimization for DBMS =

  Search strategy

  + “Generation rules” for alternative “search states”

  + Cost function for comparing alternative (partial) QEPs

Query Optimization

- Main Problem
  - Join order, join strategy (i.e. join algorithm)
- Problem:
  - Heuristic possible
  - Preferably: look at all alternatives and compare:
  - Optimization necessary
  - In general: number of possible join orders on n relations:
    $O(n!)$, i.e. large number
  - Lower complexity for special cases:
    for example chain queries: $O(n^2)$
- Optimization methods/algorithms:
  - Not new, known from AI, operations research, etc.

Transaction Manager

- Responsible for
  1. Concurrency Control: Concurrent access to data objects
  2. Recovery: Compensating for system and transaction errors
- Concurrency Control Manager (CCMgr):
  1. Correct access protocols
  2. Deadlock detection and deadlock resolution
- Recovery Manager (RecMgr.):
  - Usually based on log file
  - Follows error recovery protocols
  - Close cooperation with Buffer Mgr. and CC Mgr.

Observations on current Architecture

- Resembles 2-Phase Execution
  1. Simplify and optimize
  2. Execute (concurrently and error free)

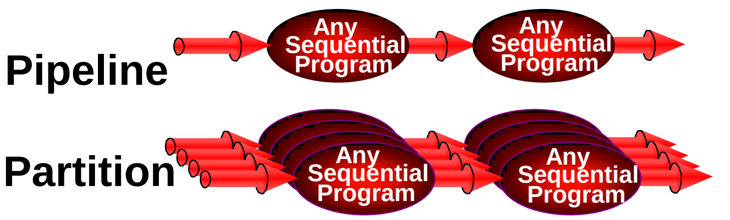
- Optimized for minimizing access to secondary storage
  - Optimize access to secondary storage (disk)
  - Keep "hot" data in caches as long as possible

- Optimized for Online-Transaction Processing (OLTP)

Parallel Data Base Management Systems

Parallel DBMS: Intro

- Parallelism is natural to DBMS processing
  - *Pipeline parallelism:* many machines each doing one step in a multi-step process.
  - *Partition parallelism:* many machines doing the same thing to different pieces of data.
  - **Both are natural in DBMS!**



outputs split N ways, inputs merge M ways

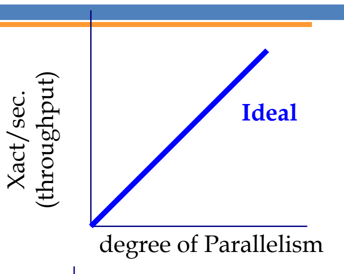
DBMS: The "Parallelism Success Story"

- DBMSs are the most (only?) successful application of parallelism.
  - Every major DBMS vendor has some parallel DBMS
    - Teradata, Tandem, Oracle, IBM DB2, IBM Informix, Microsoft SQLServer
- Reasons for success:
  - Bulk-processing (= Partitioned Parallelisms)
  - "Natural" pipelining
  - Inexpensive hardware is available
  - Users/app-programmers don't need to think "in parallel"

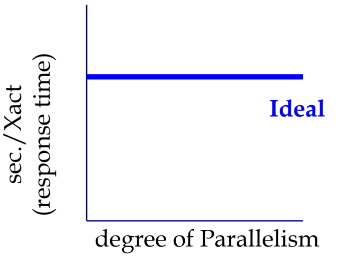
DBMS are the most successful parallel systems

Some || Terminology

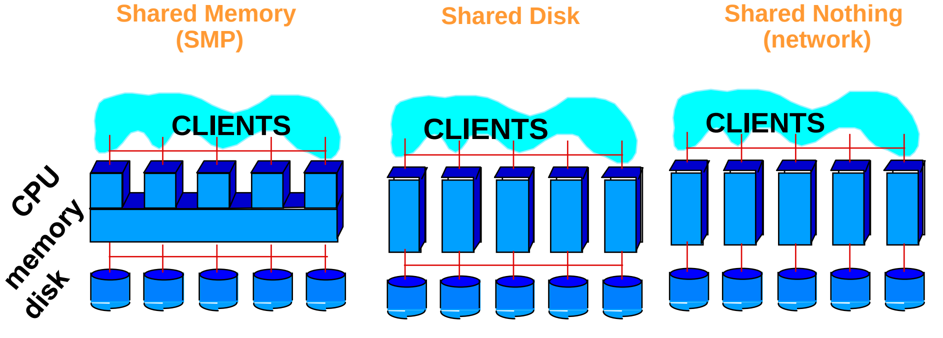
- Speed-Up
  - More resources means proportionally less time for given amount of data.



- Scale-Up
  - If resources increased in proportion to increase in data size, time is constant.



Architectural Issue: Shared What?



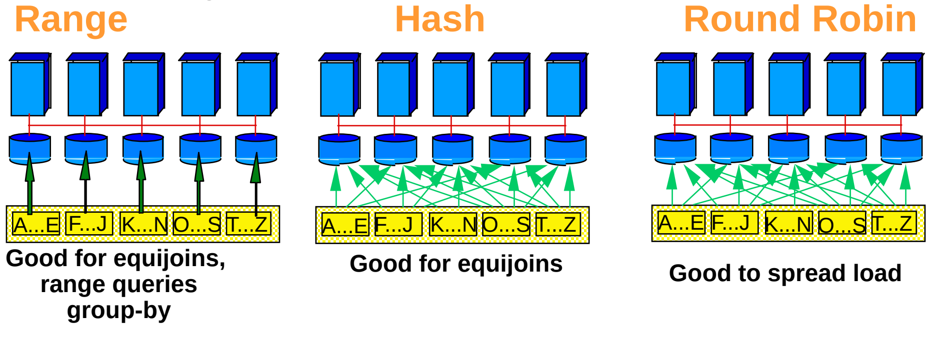
Shared Memory (SMP)	Shared Disk	Shared Nothing (network)
Easy to program Expensive to build Difficult to scaleup		Hard to program Cheap to build Easy to scale-up
IBM informix, Sun/Oracle	VMScluster, Sysplex Oracle, Oracle Rdb	Teradata, IBM DB2

Different Types of DBMS Parallelism

- **Intra-operator parallelism**
  - get all machines working to compute a given operation (scan, sort, join)
- **Inter-operator parallelism**
  - each operator may run concurrently on a different node (exploits pipelining)
- **Inter-query parallelism**
  - different queries run on different nodes
- We'll focus on ***intra-operator parallelism***

Automatic Data Partitioning

Partitioning a table:

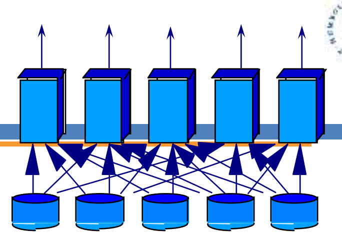


- **Shared disk and shared memory less sensitive to partitioning**
- **Shared nothing benefits from "good" partitioning**
  - **otherwise utilization of nodes may vary widely**

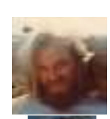
Parallel Scans

- *Scan* in parallel, and then *merge* results.
- Selection may not require all nodes
  - Advantage of range partitioning and hash partitioning
    - selections are localized on certain nodes (Round Robin spreads them out)
- Indexes:
  - can be built for each partition
  - challenges:
    - Differences in indexes for different partitioning schemes (why?)

Parallel Sorting

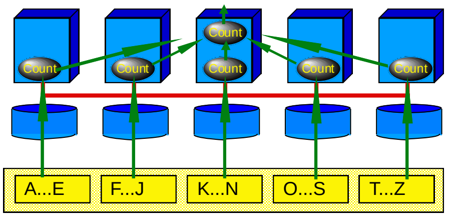


- Idea:
  - Scan in parallel, and range-partition as you go.
  - As tuples come in, begin “local” sorting on each
  - Resulting data is sorted, and range-partitioned.
  - Problem: ***skew!***
  - Solution: “sample” the data at start to determine partition points.
- Competition on sorting:
  - Initiated by J. Gray (died 2007)
    - http://research.microsoft.com/en-us/um/siliconvalley/projects/sortbenchmark/default.htm
  - See http://sortbenchmark.org/



Parallel Aggregates

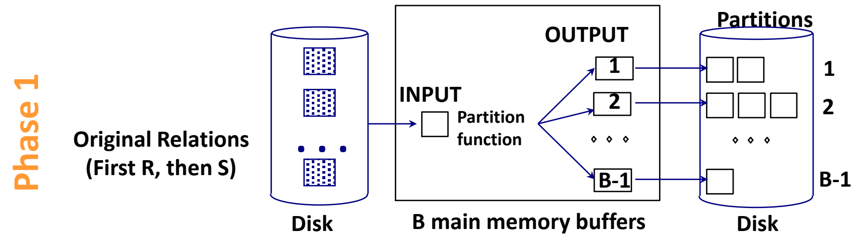
- For each aggregate function, need a decomposition:
  - count(S) = Σ count(s(i)), ditto for sum()
  - avg(S) = (Σ sum(s(i))) / Σ count(s(i))
  - and so on...
- For groups:
  - Sub-aggregate groups close to the source.
  - Pass each sub-aggregate to its group's node.
    - Chosen via a hash function



Parallel Joins

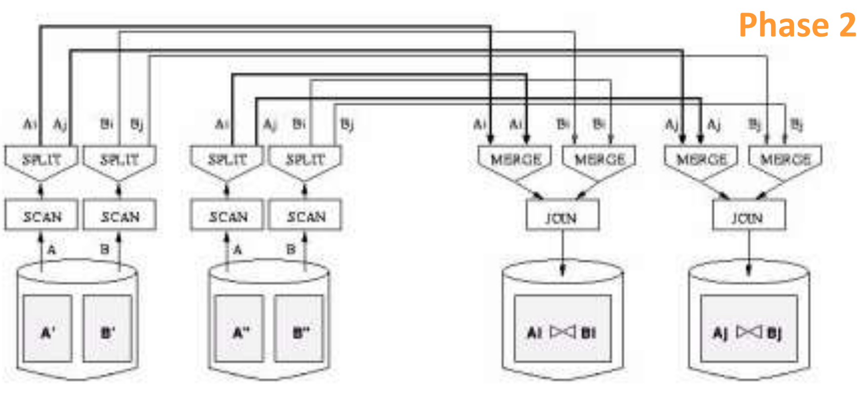
- Nested loop:
  - Each outer tuple must be compared with each inner tuple that might join.
  - Easy for range partitioning on join columns, hard otherwise!
- Sort-Merge Join (or plain Merge-Join):
  - Sorting gives range partitioning
  - Merging partitioned tables is local

Parallel Hash Join



- In **Phase 1**, partitions get distributed to different nodes:
  - A good partition function *automatically* distributes work evenly!
- **In Phase 2**, Do at each node "local" joins
  - Whatever your favourite join is
  - Independent from every other node
- Almost always the winner for equi-join

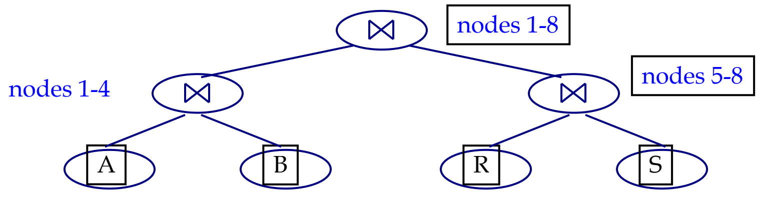
Dataflow Network for parallel Join



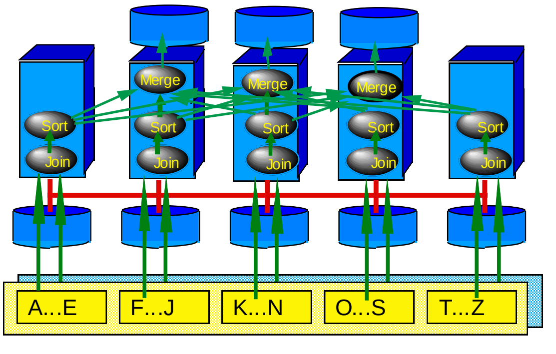
- Good use of split/merge makes it easier to build parallel versions of sequential join code.

Complex Parallel Query Plans

- Complex Queries: Inter-Operator parallelism
  - Pipelining between operators:
    - note that sort and Phase 1 of hash-join block the pipeline!!
  - Bushy Trees



N×M-way Parallelism



- N inputs, M outputs ⟶ no bottlenecks.
- **Partitioned Data**
- **Partitioned and Pipelined Data Flows**

Query Optimizer for parallel QEPs

- It is relatively easy to build a fast parallel query evaluation system
- It is hard to write a robust parallel query optimizer
  - There are many tricks.
  - One quickly hits the complexity barrier
- Common approach: **"2 phase" optimization**
  - **Phase 1:** Pick best sequential plan
  - **Phase 2:** Turn sequential plan into parallel plan
    - Pick degree of parallelism based on current system parameters
    - Granularity for parallelism: Operator
- "Bind" operators to processors/cores/threads
  - Take query tree, "decorate" as in previous picture

What's Wrong With That?

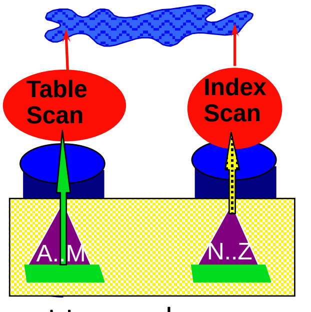
- Best serial plan != Best parallel plan
- Trivial counter example:

  SELECT *
  FROM telephone_book
  WHERE name < "NoGood";

What's Wrong With That?

- Best serial plan != Best parallel plan
- Trivial counter example:

  SELECT *
  FROM telephone_book
  WHERE name < "NoGood";



- Better execution:
  - Table partitioned with local secondary index at two nodes
  - Range query: all tuples of Node 1, and 1% of Node 2.
  - Node 1 should do a scan of its partition.
  - Node 2 should use secondary index

Parallel DBMS Summary

- Parallelism natural to query processing:
  - Both pipeline and partition Parallelism!
- Shared-Nothing vs. Shared-Memory
  - Shared-disk too, but less standard
  - Shared-memory easy, costly. Doesn't scale-up.
  - Shared-nothing cheap, scales well, harder to implement

Parallel DBMS Summary, cont.

- Data layout choices important!
- Most DB operations can be done by partition parallelisms
  - Sorting
  - Sort-merge join, hash-join
- Complex query execution plans
  - Often pipeline parallelism possible
  - but sorts, hashes block the pipeline.
  - Partition parallelism achieved via bushy execution trees

Parallelism DBMS Summary, cont.

- Hardest part of the equation: **optimization**.
  - 2-phase optimization simplest, but can be wrong
- Left out: transactions, transactional semantics, logging, failure recovery
  - Easy in shared-memory architecture
  - More complex in shared-nothing (2 phase commit)
  - Beyond this lecture

dbis

Questions ??