Database applications are usually partitioned into two or three parts

- **Two-tier architecture** -- the application resides at the client machine, where it invokes database system functionality at the server machine.
- **Three-tier architecture** -- the client machine acts as a front end and does not contain any direct database calls.
  - The client end communicates with an application server, usually through a forms interface.
  - The application server in turn communicates with a database system to access data.
Two-tier and three-tier architectures

(a) Two-tier architecture

(user

application

network

database system

server)

(b) Three-tier architecture

(user

application client

network

application server

database system

client)

Database Users

naive users
(tellers, agents, web users)

use

application interfaces

write

application programs

use

sophisticated users
(analysts)

use

query tools

use

administration tools

use

database administrators

write

DML queries

DML compiler
and optimizer

query evaluation engine

compiler and linker

object code

query processor

Database System Concepts - 7th Edition ©Silberschatz, Korth and Sudarshan

© 2021 Christopher W. Clifton
Physical Data Independence

• Physical Data Independence: the ability to modify the physical schema without changing the logical schema
  – Applications depend on the logical schema
  – In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

• We’ve talked about the *logical* schema
  – But what goes underneath?

Database Engine

- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.
- The functional components of a database system can be divided into
  • The storage manager,
  • The query processor component,
  • The transaction management component.
Isn’t Implementing a Database System Simple?

Relations ➔ Statements ➔ Results

Introducing the MEGATRON 3000
Database Management System
Megatron 3000 Implementation Details

- Relations stored in files (ASCII)
  - e.g., relation R is in /usr/db/R

<table>
<thead>
<tr>
<th>Smith # 123 # CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones # 522 # EE</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Megatron 3000 Implementation Details

- Directory file (ASCII) in /usr/db/directory

\[
\begin{align*}
\text{R1} & \# A \# \text{INT} \# B \# \text{STR} \\
\text{R2} & \# C \# \text{STR} \# A \# \text{INT} \\
& \vdots
\end{align*}
\]

Megatron 3000
Sample Sessions

```
% MEGATRON3000
   Welcome to MEGATRON 3000!
&
 &
 & quit
%```
& select A,B
from R,S
where R.A = S.A and S.C > 100 #

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>123</td>
<td>CAR</td>
</tr>
<tr>
<td>522</td>
<td>CAT</td>
</tr>
</tbody>
</table>

Megatron 3000

• To execute “select * from R where condition”:
  1) Read dictionary to get R attributes
  2) Read R file, for each line:
     a) Check condition
     b) If OK, display
Megatron 3000

• To execute “select * from R where condition | T”:
  1) Process select as before
  2) Write results to new file T
  3) Append new line to dictionary

Megatron 3000

• To execute “select A,B from R,S where condition”:
  1) Read dictionary to get R,S attributes
  2) Read R file, for each line:
     a) Read S file, for each line:
        i. Create join tuple
        ii. Check condition
        iii. Display if OK
What’s wrong with the Megatron 3000 DBMS?

- Tuple layout on disk
  - Change string from ‘Cat’ to ‘Cats’ and we have to rewrite file
  - ASCII storage is expensive
  - Deletions are expensive
- Search expensive; no indexes
  - Cannot find tuple with given key quickly
  - Always have to read full relation

What’s wrong with the Megatron 3000 DBMS?

- No buffer manager
  - Need caching
- Brute force query processing
  - select *
    from R,S
    where R.A = S.A and S.B > 1000
  - Do select first?
  - More efficient join?
What’s wrong with the Megatron 3000 DBMS?

• No concurrency control
• No reliability
  – Can lose data
  – Can leave operations half done
• No security
  – File system insecure
  – File system security is coarse

What’s wrong with the Megatron 3000 DBMS?

• No application program interface (API)
  – How can a payroll program get at the data?
• No GUI
• Cannot interact with other DBMSs.
• Poor dictionary facilities
  – How do we know what is in the database?
• Lousy salesman!!
What do we need to know?

- **File & System Structure**
  - Records in blocks, dictionary, buffer management,…

- **Indexing & Hashing**
  - B-Trees, hashing,…

- **Query Processing**
  - Query costs, join strategies,…

- **Crash Recovery**
  - Failures, stable storage,…

What do we need to know?

- **Concurrency Control**
  - Correctness, locks,…

- **Transaction Processing**
  - Logs, deadlocks,…

- **Security & Integrity**
  - Authorization, encryption,…

- **Distributed Databases**
  - Interoperation, distributed recovery,…
A program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

The storage manager is responsible to the following tasks:

- Interaction with the OS file manager
- Efficient storing, retrieving and updating of data

The storage manager components include:

- Authorization and integrity manager
- Transaction manager
- File manager
- Buffer manager
The storage manager implements several data structures as part of the physical system implementation:

- Data files -- store the database itself
- Data dictionary -- stores metadata about the structure of the database, in particular the schema of the database.
- Indices -- can provide fast access to data items. A database index provides pointers to those data items that hold a particular value.
Processor
Fast, slow, reduced instruction set, with cache, pipelined...
Speed: 100 → 1000 → 1,000,000 MIPS

Memory
Fast, slow, non-volatile, read-only,...
Access time: $10^{-6} \rightarrow 10^{-8}$ sec.
$1 \, \mu s \rightarrow 10 \, ns$

Secondary storage
Many flavors:
- Disk: Floppy (hard, soft) Removable Packs Winchester Ram disks Optical, CD-ROM...
- Tape: Reel, cartridge Robots

Arrays Solid State

Robots
Focus on: “Typical Disk”

Terms: Platter, Head, Actuator
Cylinder, Track
Sector (physical),
Block (logical), Gap

“Typical” Numbers
Diameter: 1 inch → 15 inches
Cylinders: 100 → 2000
Surfaces: 1 (CDs) →
(Tracks/cyl) 2 (floppies) → 30
Sector Size: 512B → 50K
Capacity: 360 KB (old floppy) → TB
**Disk Access Time**

I want block X → block x in memory

\[ \text{Time} = \text{Seek Time} + \text{Rotational Delay} + \text{Transfer Time} + \text{Other} \]
Seek Time

**Average Random Seek Time**

\[
S = \frac{\sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \text{SEEKTIME}(i \to j)}{N(N-1)}
\]

“Typical” $S$: 5 ms $\rightarrow$ 10 ms

SSD 0.1ms
Rotational Delay

Head Here

Block I Want

Average Rotational Delay

R = 1/2 revolution

“typical” R = 2 ms (15000 RPM)
Transfer Rate: $t$

- "typical" $t$: 100 → 200 MB/second
- transfer time: block size / $t$
- **SSD**: up to 3500 MB/s
  - But this exceeds architecture transfer limits, so often limited to 300MB/second
  - *Tape drives can match this!*

Other Delays

- CPU time to issue I/O
- Contention for controller
- Contention for bus, memory

"Typical" Value: 0
• So far: Random Block Access
• What about: Reading “Next” block?

If we do things right
(e.g., Double Buffer, Stagger Blocks…)

Time to get = Block Size + Negligible
block t
- skip gap
- switch track
- once in a while, next cylinder
Rule of Thumb

Random I/O: Expensive
Sequential I/O: Much less

• Ex: 1 KB Block
  » Random I/O: ~ 10 ms.
  » Sequential I/O: ~ 1 ms.

Curve Balls

• Buffering
  – Disks typically “read ahead” into a buffer
  – Buffer transfer rates typically 300MB/s

• “Moving” blocks
  – Disk controllers mask hardware failures by moving blocks around
  – Sequential reads may not actually be sequential…
To Modify a Block?

To Modify Block:
(a) Read Block
(b) Modify in Memory
(c) Write Block
[(d) Verify?]
Query Processor

- The query processor components include:
  - DDL interpreter -- interprets DDL statements and records the definitions in the data dictionary.
  - DML compiler -- translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
    - The DML compiler performs query optimization; that is, it picks the lowest cost evaluation plan from among the various alternatives.
  - Query evaluation engine -- executes low-level instructions generated by the DML compiler.
### Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation

![Diagram of query processing](image)

### Transaction Management

- **Transaction** is a collection of operations that performs a single logical function in a database application.
- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.
Database Architecture

- Centralized databases
  - One to a few cores, shared memory
- Client-server,
  - One server machine executes work on behalf of multiple client machines.
- Parallel databases
  - Many core shared memory
  - Shared disk
  - Shared nothing
- Distributed databases
  - Geographical distribution
  - Schema/data heterogeneity
Storage Hierarchy (Cont.)

- **primary storage**: Fastest media but volatile (cache, main memory).
- **secondary storage**: next level in hierarchy, non-volatile, moderately fast access time
  - Also called **on-line storage**
  - E.g., flash memory, magnetic disks
- **tertiary storage**: lowest level in hierarchy, non-volatile, slow access time
  - also called **off-line storage** and used for **archival storage**
  - e.g., magnetic tape, optical storage
  - Magnetic tape
    - Sequential access, 1 to 12 TB capacity
    - A few drives with many tapes
    - Juke boxes with petabytes (1000’s of TB) of storage

Storage Interfaces

- Disk interface standards families
  - **SATA** (Serial ATA)
    - SATA 3 supports data transfer speeds of up to 6 gigabits/sec
  - **SAS** (Serial Attached SCSI)
    - SAS Version 3 supports 12 gigabits/sec
  - **NVMe** (Non-Volatile Memory Express) interface
    - Works with PCIe connectors to support lower latency and higher transfer rates
    - Supports data transfer rates of up to 24 gigabits/sec
  - Disks usually connected directly to computer system
  - In **Storage Area Networks (SAN)**, a large number of disks are connected by a high-speed network to a number of servers
  - In **Network Attached Storage (NAS)** networked storage provides a file system interface using networked file system protocol, instead of providing a disk system interface
SSD (Flash) Storage

- NOR flash vs NAND flash
  - NAND flash
    - used widely for storage, cheaper than NOR flash
    - requires page-at-a-time read (page: 512 bytes to 4 KB)
      - 20 to 100 microseconds for a page read
      - Not much difference between sequential and random read
    - Page can only be written once
      - Must be erased to allow rewrite
  - Solid state disks
    - Use standard block-oriented disk interfaces, but store data on multiple flash storage devices internally
    - Transfer rate of up to 500 MB/sec using SATA, and up to 3 GB/sec using NVMe PCIe

SSD Storage (Cont.)

- Erase happens in units of erase block
  - Takes 2 to 5 millisecs
  - Erase block typically 256 KB to 1 MB (128 to 256 pages)
- Remapping of logical page addresses to physical page addresses avoids waiting for erase
- Flash translation table tracks mapping
  - also stored in a label field of flash page
  - remapping carried out by flash translation layer

- After 100,000 to 1,000,000 erases, erase block becomes unreliable
  - wear leveling
RAID

- **RAID: Redundant Arrays of Independent Disks**
  - disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
    - **high capacity** and **high speed** by using multiple disks in parallel,
    - **high reliability** by storing data redundantly, so that data can be recovered even if a disk fails
  - The chance that some disk out of a set of $N$ disks will fail is much higher than the chance that a specific single disk will fail.
    - E.g., a system with 100 disks, each with MTTF of 100,000 hours (approx. 11 years), will have a system MTTF of 1000 hours (approx. 41 days)
    - Techniques for using redundancy to avoid data loss are critical with large numbers of disks

---

**Hardware: Key Takeaways**

- Database must reside on non-volatile storage
  - Can cache in faster storage
- Non-volatile storage **slow**
  - But accessing a lot not much different than accessing a little
  - Therefore we read/write as large blocks (typically 4kb)
- Abstract performance as: $\alpha + \beta b$
  - $\alpha$ is *seek time* (abstraction of read/write setup overhead)
  - $\beta$ is *transfer rate*
  - $b$ is *block size*
- Rotating media: seek can dominate (but caching, sequential reads reduce this)
- Solid state: transfer dominates
  - but erasure, protocol overheads make “seek” more than you’d expect
- Writes typically worse than reads
  - *Not “done” until safe in non-volatile storage, so reduces caching benefits*