Building Database Applications: Steps

1. Start with a conceptual model
   - “On paper” using certain techniques
   - E-R Model
   - ignore low-level details – focus on logical representation
   - “step-wise refinement” of design with client input

2. Design & implement schema
   - Design and codify (in SQL) the relations/tables
   - Refine the schema – normalization
   - Do physical layout – indexes, etc.

3. Import the data

4. Write applications using DBMS and other tools
   - Many of the hard problems are taken care of by other people
     (DBMS, API writers, library authors, web server, etc.)
   - DBMS takes care of Query Optimization, Efficiency, etc.

Database People

- There are several different types of database personnel:
  - **Database administrator (DBA)** - responsible for installing, maintaining, and configuring the DBMS software.
  - **Data administrator (DA)** - responsible for organizational policies on data creation, security, and planning.
  - **Database designer** - defines and implements a schema for a database and associated applications.
    - **Logical/Conceptual database designer** - interacts with users to determine data requirements, constraints, and business rules.
    - **Physical database designer** - implements the logical design for a data model on a DBMS. Defines indexes, security, and constraints.
  - **DBMS developer** - writes the DBMS software code.
  - **Application developer** - writes code that uses the DBMS.
  - **User** - uses the database directly or through applications.
Components of a DBMS

- A DBMS is a complicated software system containing many components:
  - **Query processor** - translates user/application queries into low-level data manipulation actions.
    Sub-components: query parser, query optimizer
  - **Storage manager** - maintains storage information including memory allocation, buffer management, and file storage.
    Sub-components: buffer manager, file manager
  - **Transaction manager** - performs scheduling of operations and implements concurrency control algorithms.

DBMS

- A database management system provides *efficient*, *convenient*, and *safe* multi-user storage and access to *massive* amounts of *persistent* data.
  - **Efficient** - Able to handle large data sets and complex queries without searching all files and data items.
  - **Convenient** - Easy to write queries to retrieve data.
  - **Safe** - Protects data from system failures and hackers.
  - **Massive** - Database sizes in gigabytes/terabytes/petabytes.
  - **Persistent** - Data exists after program execution completes.
  - **Multi-user** - More than one user can access and update data at the same time while preserving consistency….concept of transactions

DBMS Architecture
Next: Look inside a DBMS

- Storage Manager: File organization
  - How is data organized/stored in secondary memory
  - Memory management—more in operating systems

- Query processor
  - Very brief look at how queries are executed by the machine
  - Translation of SQL code to C code

- Transaction manager
  - Dealing with concurrency—abstract definition of scheduling primitives
    In operating systems you will work with implementation

Storage and Organization Overview

- A database system relies on the operating system to store data on storage devices.

- Database performance depends on:
  - Properties of storage devices
  - How devices are used and accessed via the operating system

- Quick look into techniques for storing and representing data
  - Important Note: These apply for SQL as well as NoSQL systems
  - Key in efficient storage and retrieval systems
    Including search engines

Review from architecture (?):

Memory Definitions

- Temporary memory retains data only while the power is on.
  - Also referred to as volatile storage.
  - e.g. dynamic random-access memory (DRAM) (main memory)

- Permanent memory stores data even after the power is off.
  - Also referred to as non-volatile storage or secondary storage
  - e.g. flash memory, SSD, hard drive, DVD, tape drives

- Cache is faster memory used to store a subset of a larger, slower memory for performance.
  - processor cache (Level 1 & 2), disk cache, network cache

Physical Storage: Memory Hierarchy

- Primary Storage: cache & main memory
  - Can be directly accessed by CPU
  - Currently used data

- Secondary Storage: flash, SSD, magnetic disks, optical disks, tapes
  - Larger capacity, low cost, slow access
  - Cannot be directly processed by CPU

- DB stores large amount, persist over time
  - Data is stored in secondary storage
  - Contrast with run-time data structures
  - Time taken to fetch data depends on how data is organized on disk/file
Why Not Store Everything in Main Memory?

- Costs too much.
- Main memory is volatile.
  - We want data to be saved between runs. (Obviously!)
  - Situations that cause permanent loss of data occur less frequently in disks than primary memory
  - Disk storage is non-volatile

Recall: Disks

- Secondary storage device of choice.
- Main advantage over tapes: random access vs. sequential.
- Data is stored and retrieved in units called disk blocks or pages.
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
  - Therefore, relative placement of pages on disk has major impact on DBMS performance!

Disk Geometry

- Disks consist of platters, each with two surfaces.
- Each surface consists of concentric rings called tracks.
- Each track consists of sectors separated by gaps.

Components of a Disk

The platters spin (say, 90rps). The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a cylinder (imaginary!).

Only one head reads/writes at any one time.

- Block size is a multiple of sector size (which is fixed).
**Disk Structure**
- For READ/WRITE operations
  - H/W address of block and address of buffer is supplied to disk IO hardware via disk controller
  - Buffer is contiguous reserved area in main memory that holds block (page)
- Actual H/W that reads blocks is disk head, part of disk drive
- Disk drives rotate disk pack
- Disk arm positions disk head over block read
  - When block passes under disk head, data transferred to buffer

**Logical Disk Blocks**
- Modern disks present a simpler abstract view of the complex sector geometry:
  - The set of available sectors is modeled as a sequence of b-sized logical blocks (0, 1, 2, ...)
- Mapping between logical blocks and actual (physical) sectors
  - Maintained by hardware/firmware device called disk controller.
  - Converts requests for logical blocks into (surface,track,sector) triples.
  - Block 200 mapped to disk location \((x,y,z)\)

**Accessing a Disk Page**
- Time to access (read/write) a disk block:
  - **seek time** (moving arms to position disk head on track)
  - **rotational delay** (waiting for block to rotate under head)
  - **transfer time** (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
- Key to lower I/O cost: reduce seek/rotation delays!

**Disk Access Time**
- H/W address of disk block: (surface #, track #, sector #)
- Average time to access a target sector approximated by:
  - \(T_{\text{access}} = T_{\text{avg seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}}\)
- **Seek time** \((T_{\text{avg seek}})\)
  - Time to position heads over cylinder containing target sector.
  - Typical \(T_{\text{avg seek}} = 9\) ms
- **Rotational latency** \((T_{\text{avg rotation}})\)
  - Time waiting for first bit of target sector to pass under r/w head.
  - Typical \(T_{\text{avg rotation}} = 1/2 \times 1\text{RPMs} \times 60\text{sec}/1\text{min}\)
- **Transfer time** \((T_{\text{avg transfer}})\)
  - Time to read the bits in the target sector.
  - Typical \(T_{\text{avg transfer}} = 1\text{RPM} \times 1/(\text{avg # sectors/track}) \times 60\text{sec}/1\text{min}\).
Summary: Accessing a Disk Page

- Time to access (read/write) a disk block:
  - seek time (moving arms to position disk head on track)
  - rotational delay (waiting for block to rotate under head)
  - transfer time (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
- Key to lower I/O cost: reduce seek/rotation delays!

Example

- SELECT * FROM EMP;
- Need to scan entire file
  - Read all records
- Access all blocks/pages of the file on the disk
  - Assume N pages
- How long does this take?
- Simple approach: \( N \times T_{access} \)
  - \( T_{access} = T_{avg \ seek} + T_{avg \ rotation} + T_{avg \ transfer} \)

Example: Arranging Pages on Disk

- "Next" block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- Blocks in a file should be arranged sequentially on disk (by "next"), to minimize seek and rotational delay.
- For a sequential scan, **pre-fetching** several pages at a time is a big win!

Disk Geometry and File Layout

File = \{ block 1, b2, \ldots \}
Need to read/scan entire file:
**Example: time using next block approach**

- Need to scan entire file
  - Read all records
- Time to read first block: $T_{\text{seek}} + T_{\text{rotation}} + T_{\text{transfer}}$
- Time to read next block: $T_{\text{transfer}}$
- ....
- Time to read all $N$ blocks: First block and then one block every $T_{\text{transfer}}$ cycles
  - $(T_{\text{seek}} + T_{\text{rotation}} + T_{\text{transfer}}) + (N-1) T_{\text{transfer}}$
- About $(N-1)(T_{\text{seek}} + T_{\text{rotate}})$ less than naive approach
  - $\sim O(N)$ faster since $T_{\text{seek}} >> T_{\text{transfer}}$

**So what does this tell us?**

- Time to process a query can be reduced by careful mapping of the pages to the physical disk blocks
- Page organization on disk affects Query performance
  - Database performance linked to physical organization of data

**File Interfaces**

- Besides the physical characteristics of the media and device, how the data is allocated on the media affects performance (**file organization**).

  - The physical device is controlled by the operating system. The operating system provides one or more interfaces to accessing the device.

**Block-Level Interface**

- A **block-level interface** allows a program to read and write a chunk of memory called a **block** (or **page**) from the device.

  - The page size is determined by the operating system. A page may be a multiple of the physical device's block size.

  - The OS maintains a mapping from logical page numbers (starting at 0) to physical blocks on the device.
**File and Data Organization**

- How is data stored on disk?
  - Records
  - file of records
- how to organize the files to enable fast processing of queries
  - how to measure speed - computation or I/O time?
- Study data organization techniques that lead to more efficient processing of queries

**File and Record Organizations**

- DB applications typically need small portion of database
  - when specific data needed:
    - located on disk
    - copied into main memory
    - rewritten into disk if data changed
- data stored on disk is organized as file of records
  - File is a sequence of records
    - Records mapped to disk blocks

**Files of Records**

- Page or block is OK when doing I/O, but higher levels of DBMS operate on records, and files of records.
- **FILE**: A collection of pages, each containing a collection of records. Must support:
  - insert/delete/modify record
  - read a particular record (specified using rid: record id)
  - scan all records (possibly with some conditions on the records to be retrieved)

**Mapping Relations to Files**

- Most DBMS store each relation in separate file
  - records correspond to rows
  - record fields correspond to columns
  - joins require accessing multiple files
Recap: Representing Data in Databases

- A database is made up of one or more files.
  - Each file contains one or more blocks.
  - Each block has a header and contains one or more records.
  - Each record contains one or more fields.
  - Each field is a representation of a data item in a record.

Organization of Records

- Record is collection of related information
  - Each tuple/row is a record
  - each value is one or more bytes, corresponds to a particular field of record
  - each field specifies some attribute
  - collection of field definitions and their types constitutes record type or format
    - data type associated with each field
  - blocks are fixed size, but record sizes vary
- Two main types of records:
  - Variable length: size of record varies
  - Fixed length: all records have fixed length

Record Types

- Fixed length vs Variable length records
  - fixed is easier to implement
  - fixed wastes space when block size not multiple of record size
- spanned vs unspanned
  - when parts of a record can be placed onto a block, need pointers to next block where remainder of record is placed
Record Formats: Fixed Length

- Information about field types same for all records in a file; stored in system catalogs.
- Finding $i$'th field requires scan of record.

Base address (B) $\Rightarrow$ Address = B + L1 + L2

Record Formats: Variable Length

- Two alternative formats (# fields is fixed):
  - Second offers direct access to $i$'th field, efficient storage of nulls (special don't know value); small directory overhead.

Field Count

File Management

- Support search, scan, and insert/delete
- When records deleted or inserted, need to move records to occupy space or mark empty space
- maintain file header
  - point to next record that is deleted
  - first record point to next empty record etc.
  - can have dangling pointer problem on delete
  - pinned records: avoid moving or deleting records that are pointed to by other records

Link between file organization and DBMS efficiency?
File Organizations
- File organization determines how records are physically placed on disk
  - heap file: no particular order
  - Sorted file
  - indexed file
    - hash index
    - tree indices
- Efficiency of file organization typically measured in terms of number of disk/SSD accesses to fetch data
  - Why?

Evaluation of File Organizations
- Time always measured in # disk accesses
- Access time or lookup time
  - time to find particular data item
- Insertion time
  - time to insert new record
  - time to find correct location and time to insert
- Deletion time
- Modification time
- Space overhead
  - additional space occupied by index structure

Evaluation of File Organizations
- Relation of size \( n \) records \( \rightarrow \) \( n \) rows/tuples
- disk block size \( b \) bytes \( \rightarrow \) page size
- record size \( r \) bytes
  - average size
- "blocking factor" \( p \), number of records/block
  - \( p = \frac{b}{r} \)
- number of disk blocks to store relation
  - \( \left\lceil \frac{n}{p} \right\rceil \)

Example
- File of 1,000,000 records
- record size 200 bytes
- blocks are 4096 bytes
  - \( n = 1,000,000 \)
  - \( r = 200 \)
  - \( b = 4096 \)
  - Blocking factor, \( p = \frac{b}{r} = \frac{4096}{200} = 20 \)
  - file size = \( N = \frac{n}{p} = \frac{1,000,000}{20} = 50,000 \) blocks
Evaluation of File Organizations

- Baseline we use Heap File
  - The do nothing approach!
- Derive performance for each type of file organization
  - note that query type plays a large role in determining efficiency

Heap File Implemented as a List

- The header page id and Heap file name must be stored someplace.
- Each page contains 2 `pointers` plus data.

Evaluation of Heap Files

- Lookup time: on average retrieve \( \frac{1}{2} (n/p) \) I/Os
  - worst case = \( n/p = N \)
- Insertion time: retrieve last record on heap, if no empty space then start new block
  - 2 disk I/O
- Deletion: find record and then delete
  - \( \frac{1}{2} (n/p) + 1 \) average, \( n/p + 1 \) worst case
- Modification: same as deletion

Example

- File of 1,000,000 records
- record size 200 bytes
- blocks are 4096 bytes
  - \( n = 1,000,000 \)
  - \( r = 200 \)
  - \( b = 4096 \)
  - Blocking factor, \( p = b/r = 4096/200 = 20 \)
  - file size = \( N = n/p = 1,000,000/20 = 50,000 \) blocks
Heap File Example

- Successful lookup?
- Insertion time?
- Deletion time?
- Modification?

Heap File: Example

- Successful lookup: average \( \frac{1}{2}(n/p) = 25,000 \)
  - worst case is \( n/p = 50,000 \) disk accesses
  - At 10ms disk access time, this is 500 seconds ~ 8 minutes!
- insertion = 2
- deletion = \( \frac{1}{2}(n/p) + 1 = 25,001 \)
  - worst case = 50,001
- header page of pointers can get large
- Heap file summary: not a smart solution!

Lesson 1: better organize the records on the file

- Heap file will not cut it!
- Need to organize physical records on the file in some “smart” manner
  - Sorted file
  - Hash file

Other approaches...

- Sorted File
  - Search time: \( \log \) (Number of disk blocks)
- What if you are searching by another field….
  - Sorted by Number, search by name
Lesson 2: do we really need to access the entire file to answer a query?

- Many queries reference small portion records
  - system should be able to locate these without having to search all records
  - Without having to search through the physical file of records?