Relational Database System Implementation

CS122 – Lecture 1

Winter Term, 2018-2019

Welcome!

- **How do relational databases work?**
  - Provide a *hands-on* opportunity to explore this topic
- This is a project course:
  - A sequence of programming assignments focusing on various areas of database system implementation
  - Also some reading and problem-solving, but not a lot
- No midterm, no final exam ☺

Prerequisites

- You should be familiar with:
  - The SQL query language (DDL and DML)
  - The relational model and relational algebra
  - I will post a link to the CS121 slides
- Also, familiarity with:
  - Algorithms (e.g. sorting, searching), trees, graphs, etc.
  - Basic time and space complexity
- All programming is in Java 11

Software Development

- Secondary focus of course is on software development
  - Automated build system: Maven
  - Documentation generation: Javadoc
  - Automated testing: TestNG (JaCoCo for test coverage)
  - Code linting / static analysis: ~~FindBugs~~ ???
  - Version control: Git
  - Integrated development environments (IntelliJ IDEA)

- You will work with all of these tools during the term

Software Development (2)

- Databases must be correct
  - A portion of each assignment's grade will depend on the correctness of your work
- All work must be clean and well-documented
  - Follow standard Java coding conventions
  - Use Javadoc to document the code you write
  - Clearly and concisely comment your code as well
  - Git commit-logs must also include clear documentation of your changes
  - Failure to do these things will result in point deductions

Teams and Collaboration

- You must work in teams of 2-3 people
  - Need to decide team composition relatively quickly
  - First assignment hopefully out sometime this week
- Each team's submission must be entirely its own work
- Feel free to help debug each other's programs, get tools or the debugger working, etc.
  - Follow the "50-foot rule" for helping other teams debug: help others with your brain, not your code.

Course Website

- Lectures, assignments, and other course details are available on the Caltech Moodle
  - https://courses.caltech.edu/course/view.php?id=3254
  - Enrollment key: planner
  - **Please enroll ASAP!** All course announcements are made through Moodle.

- Course policies, collaboration policy, contact info, etc. will all be posted there – make sure to review it all!

Late Policy

- Late penalties will be applied to assignments:
  - Up to 24 hours late: -10%
  - Up to 48 hours late: -30%
  - Up to 72 hours late: -60%
  - After 72 hours: don't bother ☹
- Extensions are available for extenuating circumstances
  - Talk to the Dean's office for extensions due to long-term wellness issues, family emergencies, etc.
  - Talk to me for extensions due to interviews, class/team trips, etc.
- Teams also have 4 tokens to use throughout the term
  - Each token worth a 24-hour extension, "no questions asked"
  - State in your submission how many tokens you are using

Course Overview

- This class focuses on the implementation of *relational databases*
- Tables (relations) populated with rows (tuples), manipulated with SQL queries

- Lots of other kinds of databases too, but (sadly) we won't discuss them this term
  - CS123 is a great venue to explore other data models, e.g. XML databases, graph databases, NoSQL, etc.

Relational DB Requirements (1)

- Need generalized query support
  - Use the SQL query language
- How to represent query plans internally?
- How to optimize query plans?
  - What plan transformations are allowed?
  - What data statistics do we need to guide this phase?
- How to execute query plans?
  - General issues: subqueries, correlated subqueries, inserts/updates/deletes, etc.
  - Advanced techniques (e.g. parallel execution)

Relational DB Requirements (2)

- Typically needs persistent storage of data
  - Some databases are in-memory only
  - (Makes things *much* simpler!)
- Problem: disks are slow
- How to improve data access performance?
  - Storage formats, data layouts, data access patterns
  - Alternate *access paths* (i.e. indexes)
  - Better hardware? (e.g. RAID, SSD)
- How do these improvements affect query planning and execution?

Relational DB Requirements (3)

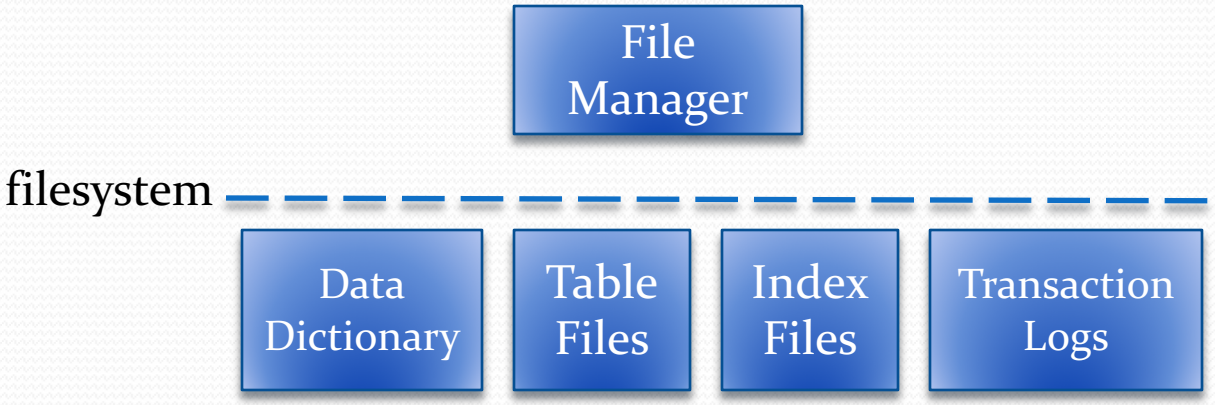
- Generally, databases should provide transacted operations
  - Satisfy specific properties (ACID)
- Example: Durability
  - I complete some critical piece of work
    - i.e. the database reports the transaction as committed
  - Then the power fails!
  - At restart, database should ensure my work is still there…
- Example: Consistency and Atomicity
  - I am performing a multiple-step transaction…
  - Along the way, I violate a database constraint!
    - DB must roll back all other changes in the transaction

Relational DB Requirements (4)

- Databases often need to support concurrent access from multiple clients
- How to ensure that client operations are isolated from each other?
  - *(And what do we mean by "isolated" anyway?)*
  - How to govern access to shared tables/records
  - What techniques can make this fast and efficient?
- How does concurrent access affect query planning, optimization, and execution?

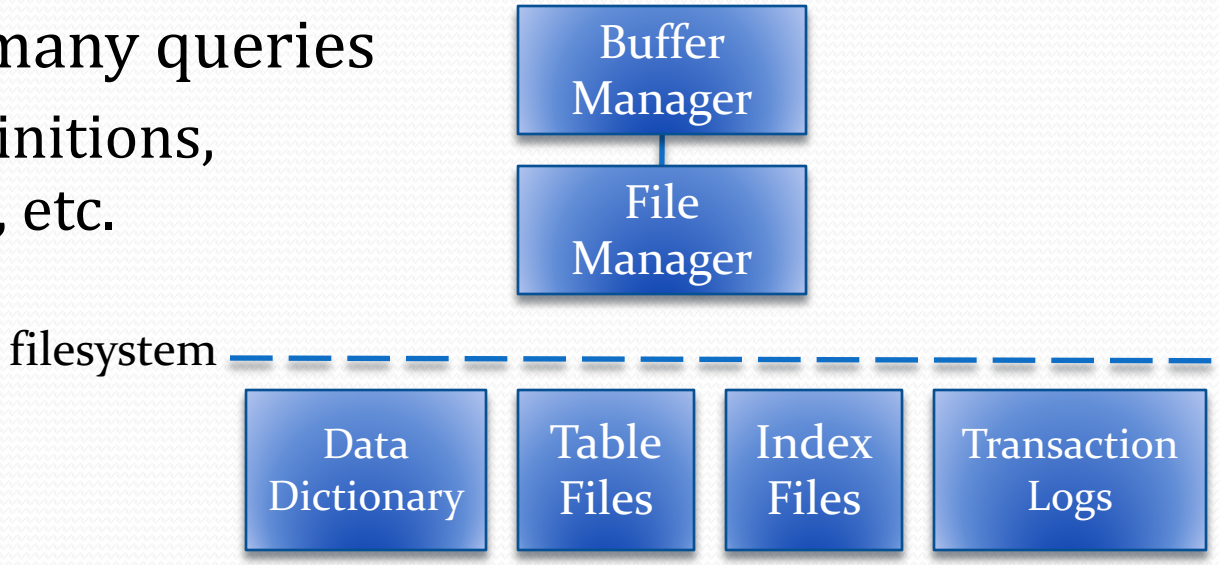
DB Architecture (1)

- Database data is stored in files on disks
  - Files are read and written in blocks to improve performance and simplify transaction processing
- The *file manager* exposes raw file data to the DB
  - Allows blocks or pages of data to be read from a file
  - Allows files to be created or deleted as well



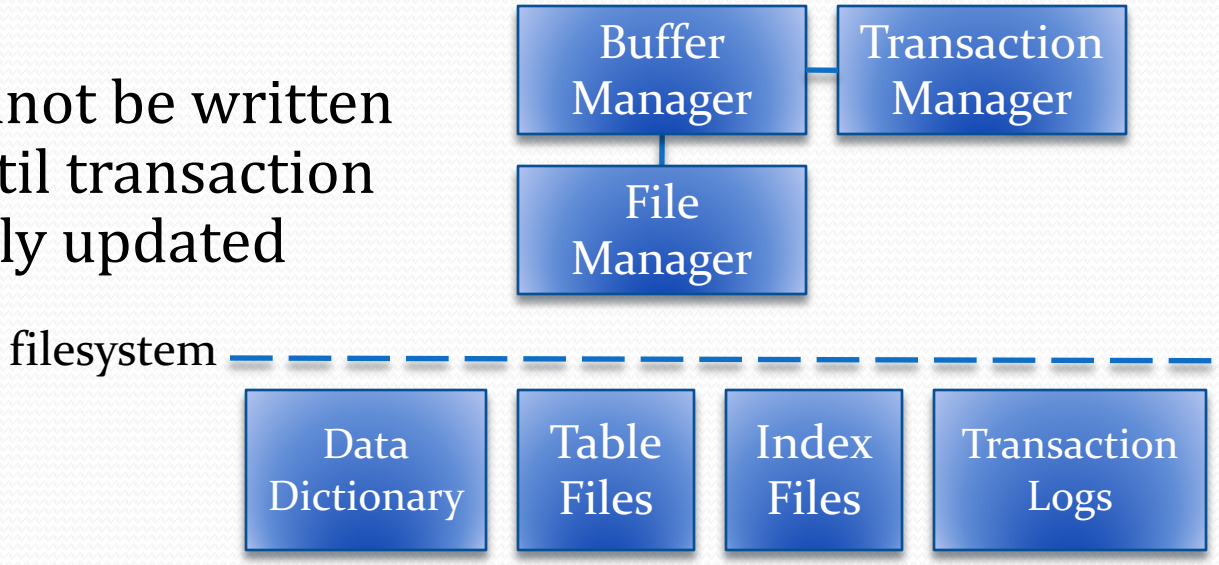
DB Architecture (2)

- The *buffer manager* caches disk pages in memory
  - Ensures that disk is only accessed when necessary
  - Allows very large data files to be managed by database
- Frequently, most widely used data pages remain cached in memory across many queries
  - e.g. schema definitions, critical indexes, etc.



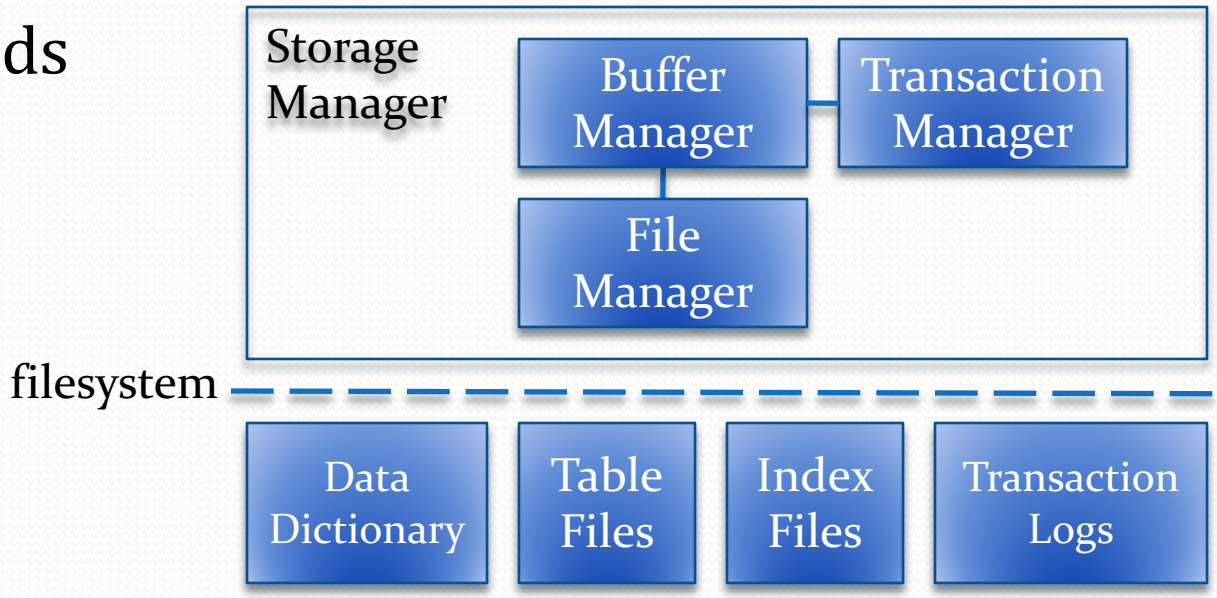
DB Architecture (3)

- The *transaction manager* provides transaction processing in the system
  - Must work closely with the buffer manager
  - Ensure pages are written back to disk in a way that also satisfies transaction properties
- Example:
  - Dirty pages cannot be written back to disk until transaction logs are properly updated



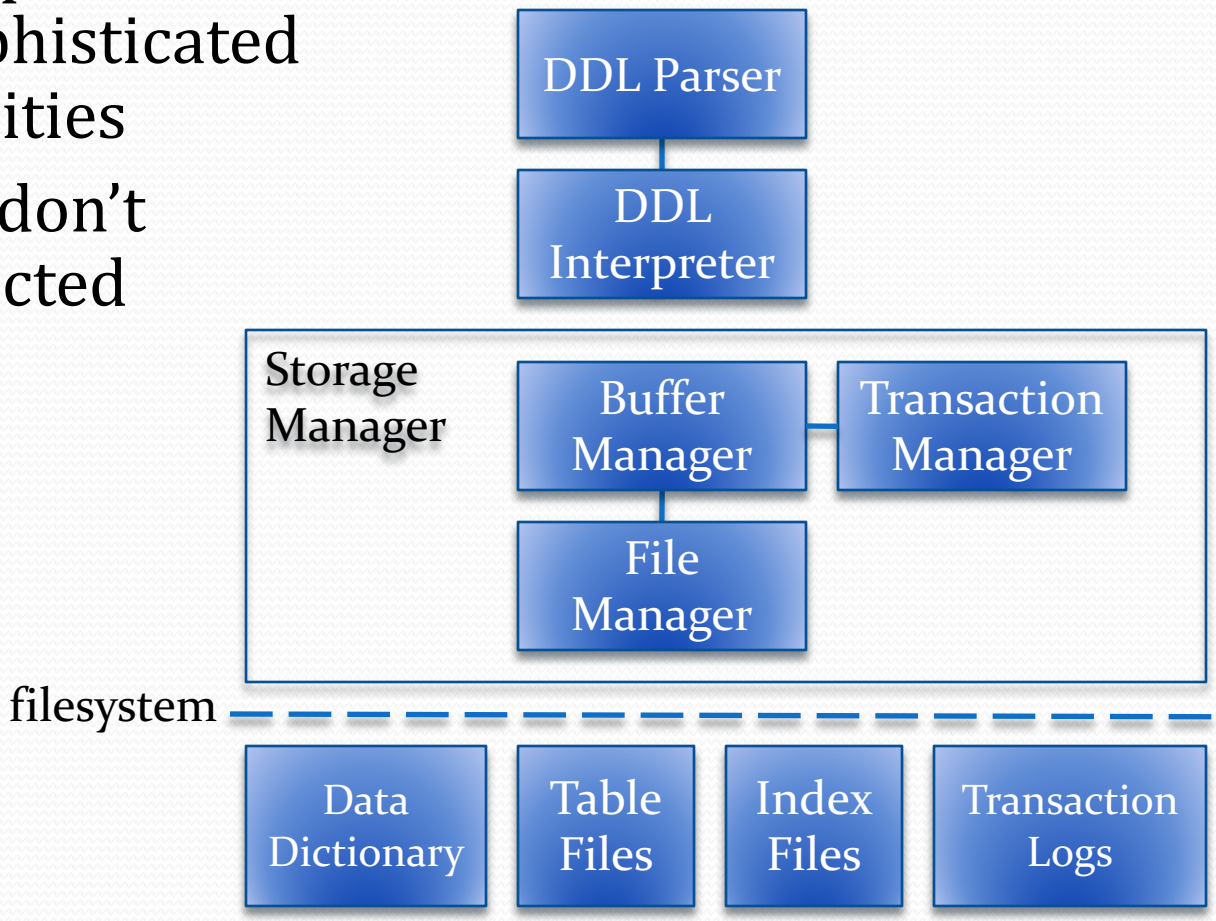
DB Architecture (4)

- These components collectively comprise the *storage manager*
- Provides access to schema information, table data, indexes, and statistics about table data
- Can also manipulate each of these kinds of data



DB Architecture (5)

- Data definition operations don't require sophisticated querying capabilities
- Many databases don't even have transacted DDL

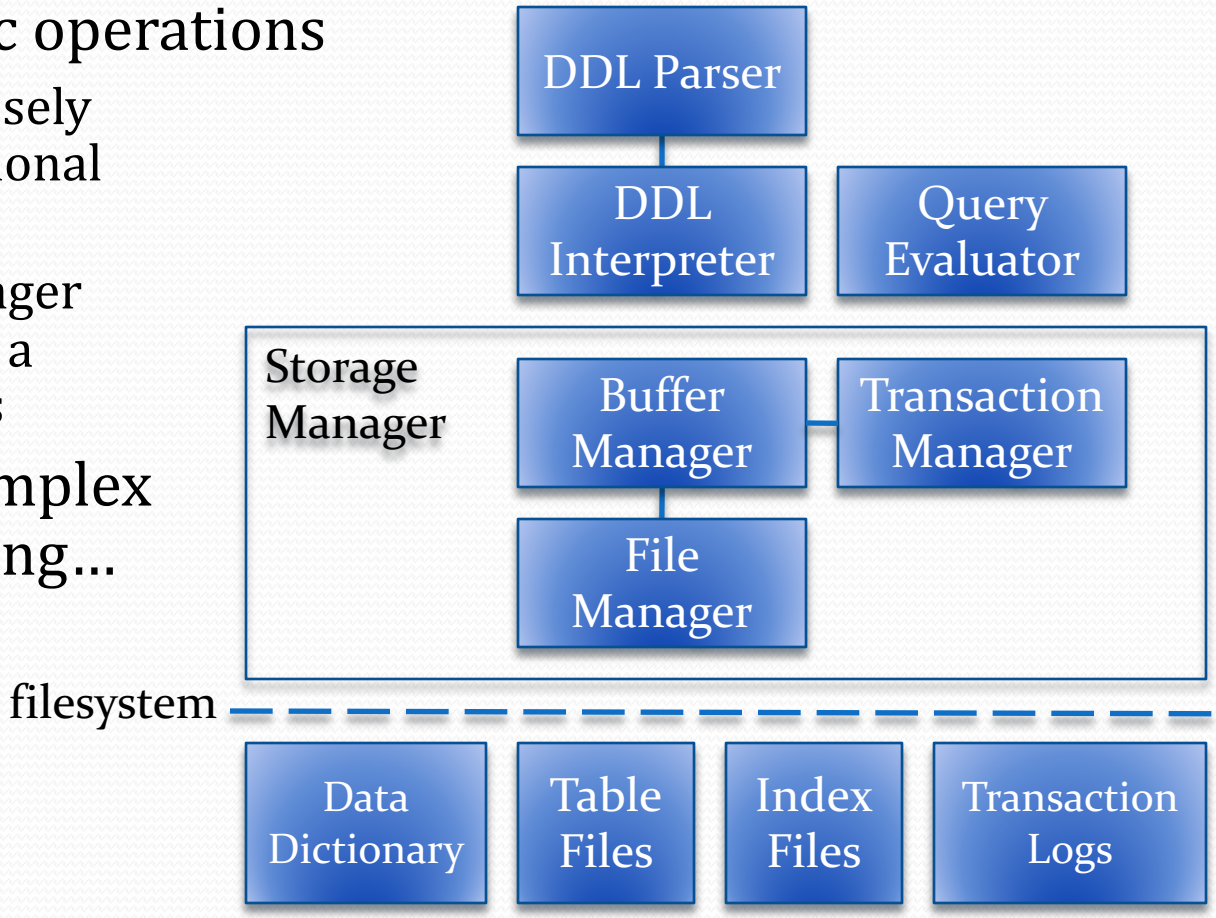


Data Definition

- Data definition operations require manipulation of data dictionary, table files, and index files
  - Create/alter/drop a table
  - Create/alter/drop an index
- The *data dictionary* holds all DB schema information
  - Most databases use another set of tables to store the data dictionary
    - Use same machinery to read/write tuples in data dictionary
    - Makes it easier to transact DDL operations too
    - Manipulation of these tables is performed more directly, without using query processing system

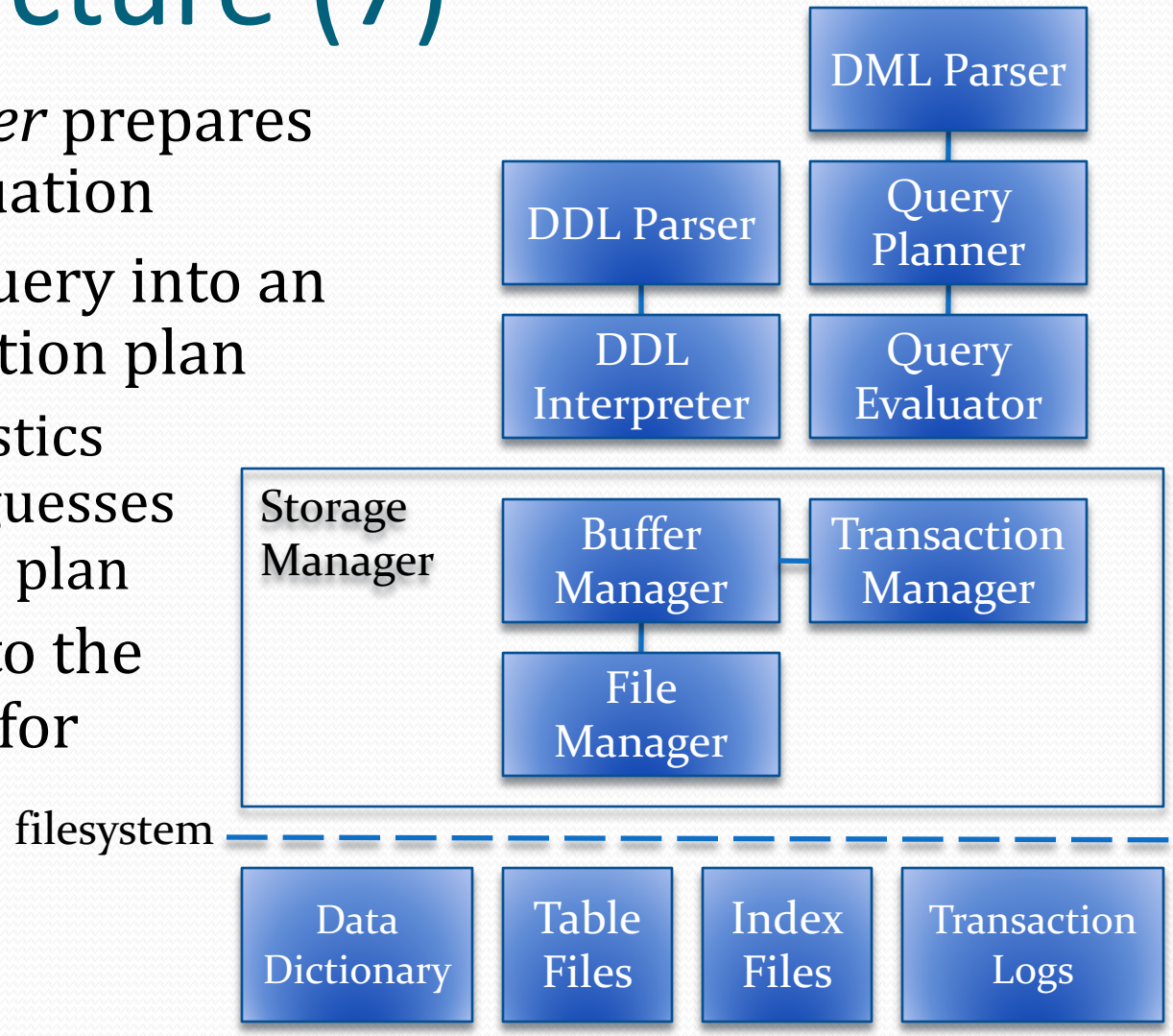
DB Architecture (6)

- The *query evaluator* executes plans comprised of basic operations
  - Operations are loosely based on the relational algebra operators
  - Uses storage manager to access tables as a sequence of tuples
- Queries can be complex and time-consuming…



DB Architecture (7)

- The *query planner* prepares a query for evaluation
- Translates the query into an optimized execution plan
  - Uses data statistics and educated guesses to optimize the plan
- Passes the plan to the query evaluator for execution



Introducing NanoDB

- Assignments and projects will be based on the NanoDB codebase
  - An increasingly sophisticated DB implementation in Java
- Primary goal is to be understandable and extensible
  - When deciding between performance and clarity, NanoDB generally chooses clarity
  - (Most open-source database projects go the other way.)
- Definitely still a work in progress!
  - Plenty of opportunity to make a lasting contribution to this project, and to future iterations of this class

Database Files

- The database stores data in files, and manipulates data in memory…
  - …this data is stored on physical devices in the computer
- Need to understand the characteristics of these devices to understand how to implement databases properly

Storage Hierarchy

- Wide variety of storage media available for use
- Can classify storage media based on:
  - Access speed
  - Cost per unit storage
  - Reliability
- Different media support different access patterns
  - Affects their usefulness for various tasks
- Examples:
  - Magnetic tapes are best suited to sequential access; random access is prohibitively slow
  - DRAM/flash memory are very good for random access

Storage Hierarchy (2)

- Primary storage:
  - Cache memory is usually very small, very fast
  - Main memory is much larger than cache, but usually still too small to hold an entire database
  - Supports random access (a.k.a. direct access)
  - Storage is *volatile*: data doesn't survive a power loss
- Secondary storage, a.k.a. online storage:
  - Much larger, cheaper, and slower than primary storage
  - Storage is *persistent*: data lasts through a power loss
  - Magnetic disks are most common form of secondary storage
  - Flash memory devices (Solid State Drives or SSD) are increasingly common, but are still smaller/more expensive

Storage Hierarchy (3)

- Tertiary storage, a.k.a. offline storage:
  - Optical storage (CDs, DVDs, etc.), tape storage
  - Primarily used for backup and archival storage
  - Access rates are *very* slow compared to primary and secondary storage
  - Tapes only support sequential access, not direct access
- Tapes/optical disks can be managed in a library
  - Robotic system to load specific tapes/disks into a drive
  - Called near-line storage
- Primary difference between offline & near-line storage:
  - A computer can access near-line storage and load it into online storage all by itself
  - A human being must make offline storage available

Internal vs. External Memory

- In database systems, primary storage is also often called *internal memory*
  - Memory the CPU can access quickly and efficiently
- Similarly, secondary storage (flash memory, magnetic disks) are called *external memory*
- Algorithms designed to work with data sets much larger than primary storage are called *external memory algorithms*
  - Must repeatedly load portions of data into internal memory, process them, then store them back to disk

Internal vs. External Memory (2)

- Data structures for organizing data on disk are called external memory structures
  - e.g. hash file organization also called *external hashing*
- Frequently encounter "external-memory [*whatever*]" in database literature
  - Simply means the algorithm/structure is designed to work with data stored on disk, not just in memory

Storage Hierarchy (4)