Database Systems - The Complete Book

Text Notes and Summary

Original text by: H. Garcia-Molina, J. D. Ullman, J. Widom

Chapter 1 The Worlds of Database Systems

The power of databases comes from a body of knowledge and technology th at has developed over several decades and is embodied in specialized software called a *database management system* (DBMS), or "database system". A DBMS is a powerful tool for creating and managing large amounts of data efficiently and allowing it to persist over long periods of time, safely. These systems are among the most complex types of software available. We shall learn how to design databases, how to write programs in the various languages associated with a DBMS, and how to implement the DBMS itself.

1.2 Overview of a Database Management System

We preesent an outline of a complete DBMS. Single boxes represent system components, while double boxes represent in-memory data structures. The solid lines indicate control and data flow, while dashed lines indicate data flow only.

First we suggest there are two distinct sources of use to the DBMS:

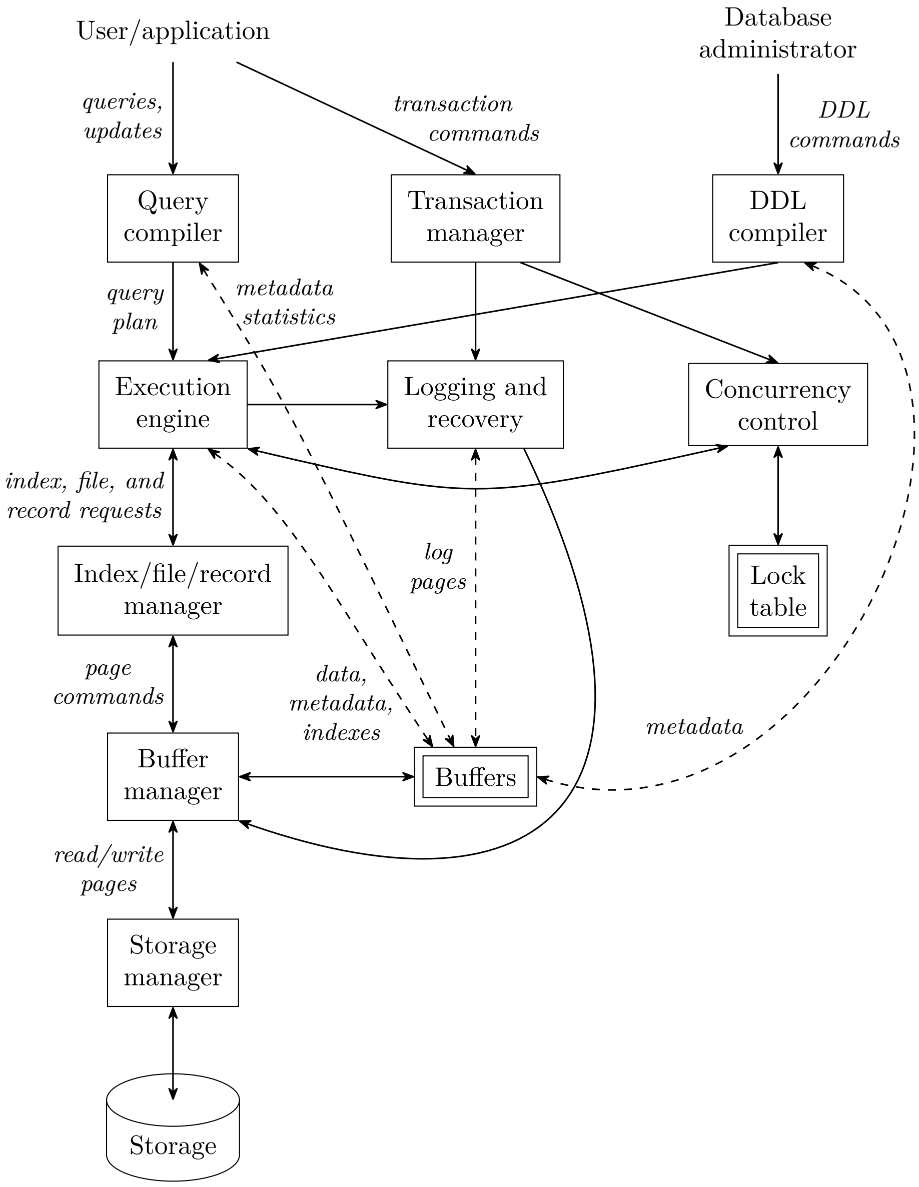
1. Conventional users and application programs that ask for data or modify data.
2. A *database administrator*, responsible for the structure or *schema* of the database.

1.2.1 Data-Definition Language Comands

The second kind of commands: shcema-altering *data-definition language* (DDL) commands, ara parsed by a DDL processor and passed to the execution engine, which then goes to the index/file/record manager to alter the *metadata* (schema information fot the database).

1.2.2 Overview of Query Processing

A user or an application program initiates some action using the *data-manipulation-language* (DML). This command cannot affect the schema of the database, but may affect its contents, or extract data from it. These commands are handled by two separate subsystems.



Answering the Query The query is parsed and optimized by a *query compiler*. The resulting *query plan* (sequence of actions the DBMS will perform to answer the query) is passed to the *execution engine*, which issues a sequence of requests for small pieces of data (typically records or tuples of a relation) to a *resource manager* that knows about data files (holding relations), the format and size of records in those files, and index files, which help find elements of data files quickly. The requests for data are passed to the *buffer manager*, which task is to bring appropriate portions of the data from secondary storage (disk) where it is kept permanently, to the main-memory buffers, communicating with a *storage manager* to get data from disk. The storage manager might involve operating-system commands, but more typically, the DBMS issues commands directly to the disk controller.

1.2.3 Storage and Buffer Manager

Data of a database normally resides in secondary storage (disk). However, to perform any useful operation on data, that data must be in main memory. The *store manager* controls the placement of data on disk and its movement between disk and main memory.

The storage manager keeps track of the location of files on the disk and obtains the blocks containing a file on request from the *buffer manager*. The buffer manager is responsible for partitioning the available main memory into buffers, which are page-sized regions into which disk blocks can be transferred.

So all DBMS components that need information from the disk will interact with the buffers or the buffer manager (directly or through execution engine).

1.2.4 Transaction Processing

Queries and other DML actions are grouped into *transactions*, which are units th at must be executed atomically and in isolation from other transactions. Any query or modification action can be a transaction by itself. The execution of transactions must be *durable*, the work of a completed transaction will never be lost.

The *transaction manager* accepts *transaction commands* from an application, which tell the transaction manager when transactions begin and end. The transaction manager performs the following tasks.

1. *Logging*: every change in the database is logged separately on disk, by a *log manager*. So when a system failure occurs, a *recovery manager* will be able to examine the log of changes and restore the database to some consistent state.
2. *Concurrency control*: Transactions must appear to execute in isolation. May not execute atomically, but in a way that the net effect is the same as if the transactions executed one at a time.
3. *Deadlock resolution*: resolve deadlocks produced by resource deadlocks between transactions.

1.2.5 The Query Processor

The portion of the DBMS that most affects the performance that the user sees is the *query processor*, represented by two componetnts.

1. *Query compiler*: translates query into internal form called *query plan*, a sequence of operations performed on data. The compiler may use metadata and statistics about the data to decide which sequence of operations is likely to be the fastest.

2. *Execution engine*: executes each of the steps in the chosen query plan.

Chapter 2 The Relational Model of Data

2.1 Data Models

A *data model* is a notation for describing data or information, which consists in three parts.

i) *Structure of data.* The data structures used to implement (in high level languages i.e. C, Java . . .) data in the computer are sometimes referred to as *physical data model.* In the database world, data models are higher level than data structures, thus called *conceptual model.*

ii) *Operations on the data.* Generally allowed to perform a limited set of operations: *queries* and *modifications.* This limitation makes possible to describe database operations at a very high level, yet the DBMS implement the operations efficiently.

iii) *Constraints on the data.*

We can differentiate three families of data models for database systems.

i) *Relational Model.* Based on tables, called *relations.* Operations associated are table-oriented and form the "relational algebra". Mostly used with SQL language.

ii) *Semistructured Model.* Resemble trees or graphs, rather than tables or arrays. Operations involve following paths from an element to its nested subelements, then to subelements nested within those. May use XML or JSON as a language to represent data.

iii) *Other Data Models.* Among those we find a trend to add object-oriented features to the relational model. Values can have structure, rather than being elementary types (integers, strings ...), and relations can have associated methods. Called *object-relational* model. Other fallen models include the *hierarchical model* and the *network model* which was a graph-oriented, physical level model.

2.1.1 Comparison of Modeling Approaches

2.2 Basics of the Relational Model

Definition. The *relational model* gives us a single way to represent data: as a two-dimensional table called *relation*, where each row represents an *entry* or *tuple* in the table, and the columns, called *attributes*, represent a property of the entries.

As an example consider the following relation called `Movies`.

title	*year*	*length*	*genre*
`Gone With the Wind`	1939	231	`drama`
`Star Wars`	1977	124	`sciFi`
`Wayne's World`	1992	95	`comedy`

Definition. The name of a relation and the set of attributes for a relation is called the *schema* for that relation. We can notate a schema with the name and a parenthesized list of attributes.

In the relational model, a database consists of one or more relations. The set of schemas for the relatios of a database forms the *relational database schema.*

For example for the `Movies` relation we can write the schema as `Movies(title, year, length, genre)`.

The relational model requires that each component of each tuple be atomic (some elementary type such as integer or string). It is not permitted for a value to be a compound type (structure, set, list, array ...). Associated with each attribute of a relation is a *domain*, a particular elementary type. The components of any tuple of the relation must have, in each component, a value that belongs to the domain of the corresponding column.

We can include the domain in the scheme notation. For example for the `Movies` relation, `Movies(title:string, year:integer, length:integer, genre:string)`.

Relation Instances. A relation is no static: it changes over time. We can insert tuples, delete, and modify them. It is less common, though possible, for the schema of a relaition to change.

Definition. A set of tubles for a given relation is an *instance* of that relation. The instance that a DBS maintains in present time is called the *current instance.*