CS 4604: Introduction to Database Management Systems

Virginia Tech CS 4604 Sprint 2021

Instructor: Yinlin Chen

Today’s Topics

- Introduction to database systems
- Architecture & Classification

What is a Database?

- A **Database** is a large, organized collection of related data
- A **Database Management System** (DBMS) is the software designed to store, manage, and facilitate access to large collections of related data
- The combination of a DBMS and a database is then often called a **database system**

Features of a DBMS

- Supports massive amounts of data
  - Giga/tera/petabytes
  - Far too big for main memory
- Persistent storage
  - Programs update, query, manipulate data.
  - Data continues to live long after program finishes.
- Efficient and convenient access
  - Efficient: do not search entire database to answer a query.
  - Convenient: allow users to query the data as easily as possible.
- Secure, concurrent, and atomic access
  - Allow multiple users to access database simultaneously.
  - Allow a user access to only to authorized data.
  - Provide some guarantee of reliability against system failures.

Relational DBMSs

- Traditionally DBMS referred to relational databases
  - Called **Relational Database System** (RDBMS)
  - A DBMS designed specifically for relational databases
  - Oracle, SQL Server, PostgreSQL, SQLite, MySQL, etc.
- SQL data description and manipulation language
- **A**tomicity, **C**onsistency, **I**solation, **D**urability (**ACID**) transaction consistency
- Durable writes (prevent data loss)
- Mature technologies …

Applications of Database Technology

- Storage and retrieval of data in an inventory application
- Multimedia applications (e.g., YouTube, Spotify)
- Biometric applications (e.g., fingerprints, retina scans)
- Wearable applications (e.g., FitBit, Apple Watch)
- Geographical Information Systems (GIS) applications (e.g., Google Maps)
- Sensor applications (e.g., nuclear reactor)
- Big Data applications (e.g., Walmart, Target, Amazon)
- Internet of Things (IoT) applications (e.g., Telematics)

Database Services

Database type	Use cases	AWS service
Relational	Traditional applications, ERP, CRM, e-commerce	Amazon Aurora, Amazon RDS, Amazon Redshift
Key-value	High-traffic web apps, e-commerce systems, gaming applications	Amazon DynamoDB
In-memory	Caching, session management, gaming leaderboards, geospatial applications	Amazon ElastiCache for Memcached Amazon ElastiCache for Redis
Document	Content management, catalogs, user profiles	Amazon DocumentDB (with MongoDB compatibility)
Wide column	High scale industrial apps for equipment maintenance, fleet management, and route optimization	Amazon Keyspaces (for Apache Cassandra)
Graph	Fraud detection, social networking, recommendation engines	Amazon Neptune
Time series	IoT applications, DevOps, industrial telemetry	Amazon Timestream
Ledger	Systems of record, supply chain, registrations, banking transactions	Amazon QLDB

File System versus a DBMS

- Layout for student records using a file?
  - CSV (‘comma-separated-values’)

    Hermione Grainger,123,Potions,A

    Draco Malfoy,111,Potions,B

    Harry Potter,234,Potions,A

    Ron Weasley,345,Potions,C

File System versus a DBMS

- File approach
  - duplicate or redundant information will be stored
  - danger of inconsistent data
  - strong coupling between applications and data
  - hard to manage concurrency control
  - hard to integrate applications aimed at providing cross-company services

File System versus a DBMS

- Database approach
  - superior to the file approach in terms of efficiency, consistency and maintenance
  - loose coupling between applications and data
  - facilities provided for data querying and retrieval

File System versus a DBMS

- File approach

```
Procedure FindStudent;
begin
    open file Student.txt;
    Read(Student)
    While not EOF(Student)
        If Student.name='Bart' then
            display(Student);
        EndIf
        Read(Student);
    EndWhile;
End;
```

- Database approach (SQL)

```
SELECT *
FROM Students
WHERE
name = 'Bart'
```

Elements of a Database System

- Database model versus instances
- Data Model
- The Three Layer Architecture
- Catalog
- Database Users
- Database Languages

Database model versus instances

- Database **model** or database **schema** provides the description of the database data at different levels of detail and specifies the various **data items**, their **characteristics** and **relationships**, **constraints**, **storage** details, etc.
  - specified during database design and not expected to change too frequently
  - stored in the catalog
- Database **state** represents the data in the database at a particular moment
  - also called the **current set of instance**
  - typically changes on an ongoing basis

Database model versus instances

- Database model

Student (number, name, address, email)

Course (number, name)

Building (number, address)

Database model versus instances

- Database state

STUDENT			
Number	**Name**	**Address**	**Email**
0165854	Bart Baesens	1040 Market Street, SF	Bart.Baesens@kuleuven.be
0168975	Seppe vanden Broucke	520, Fifth Avenue, NY	Seppe.vandenbroucke@kuleuven.be
0157895	Wilfried Lemahieu	644, Wacker Drive, Chicago	Wilfried.Lemahieu@kuleuven.be

COURSE	
Number	**Name**
D0I69A	Principles of Database Management
D0R04A	Basic Programming
D0T21A	Big Data & Analytics

BUILDING	
Number	**Address**
0600	Naamsestraat 69, Leuven
0365	Naamsestraat 78, Leuven
0589	Tiensestraat 115, Leuven

Data Model

- A database model is comprised of different data models, each describing the data from different perspectives
- A data model provides a clear and unambiguous description of the data items, their relationships and various data constraints from a particular perspective

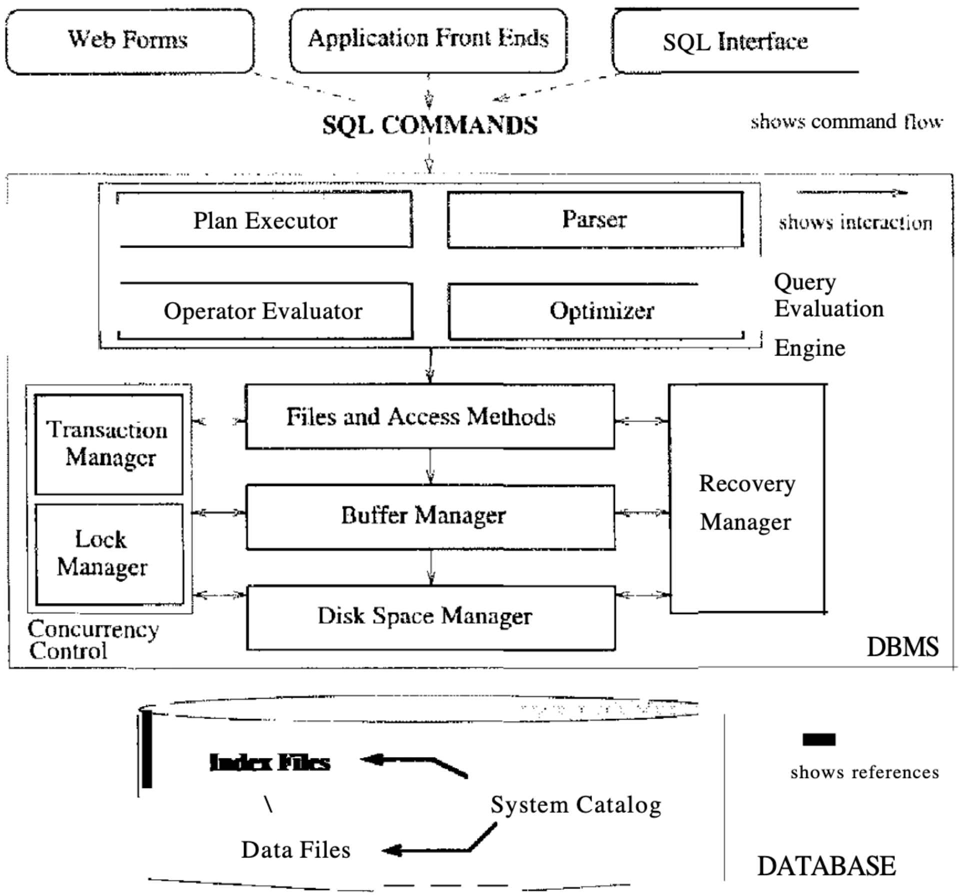
Data Model

- A conceptual data model provides a high-level description of the data items with their characteristics and relationships
  - Communication instrument between information architect and business user
  - Should be implemented as independent, user-friendly, and close to how the business user perceives the data
  - Usually represented using an Enhanced-Entity Relationship (EER) model, or an object-oriented model
- Logical data model is a translation or mapping of the conceptual data model towards a specific implementation environment
  - Can be a hierarchical, CODASYL, relational, object-oriented, extended relational, XML or NoSQL model

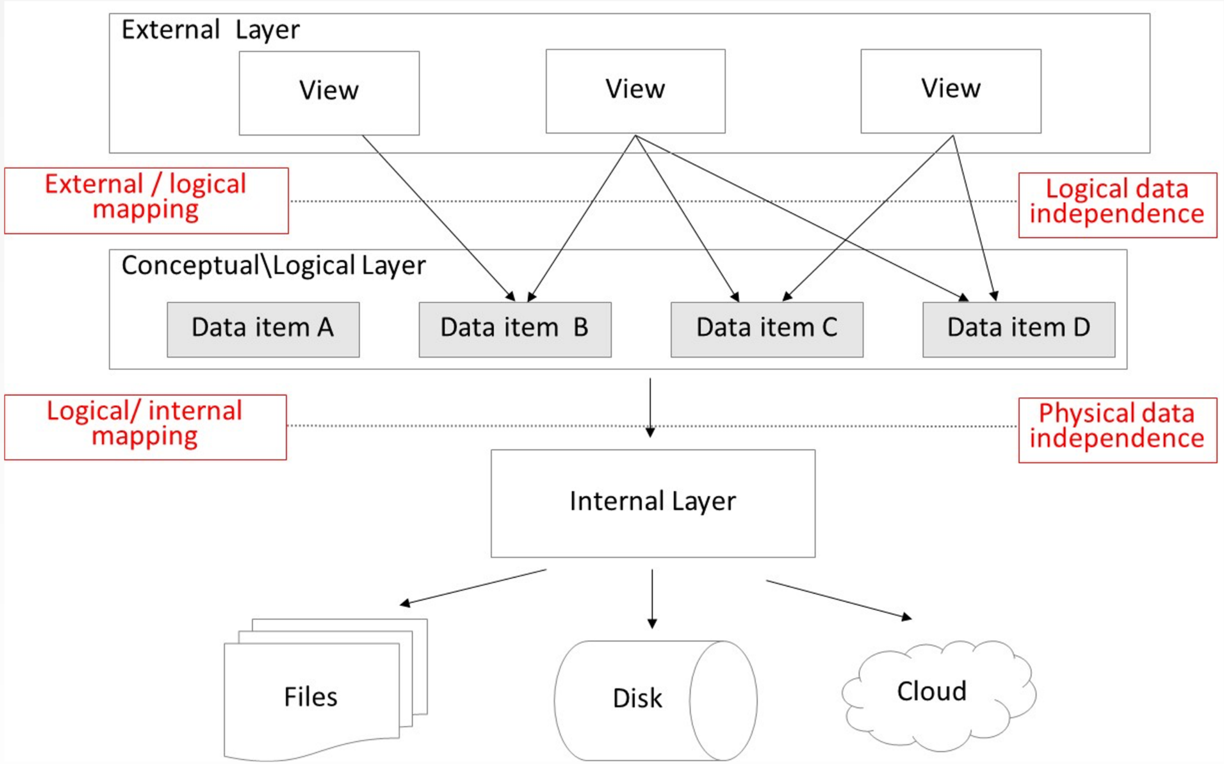
Data Model

- Logical data model can be mapped to an internal data model that represents the data's physical storage details
  - Clearly describes which data is stored where, in what format, which indexes are provided to speed up retrieval, etc.
  - Highly DBMS specific
- External data model contains various **subsets** of the data items in the logical model, also called **views**, tailored towards the needs of specific applications or groups of users

DBMS Architecture



The Three Layer Architecture

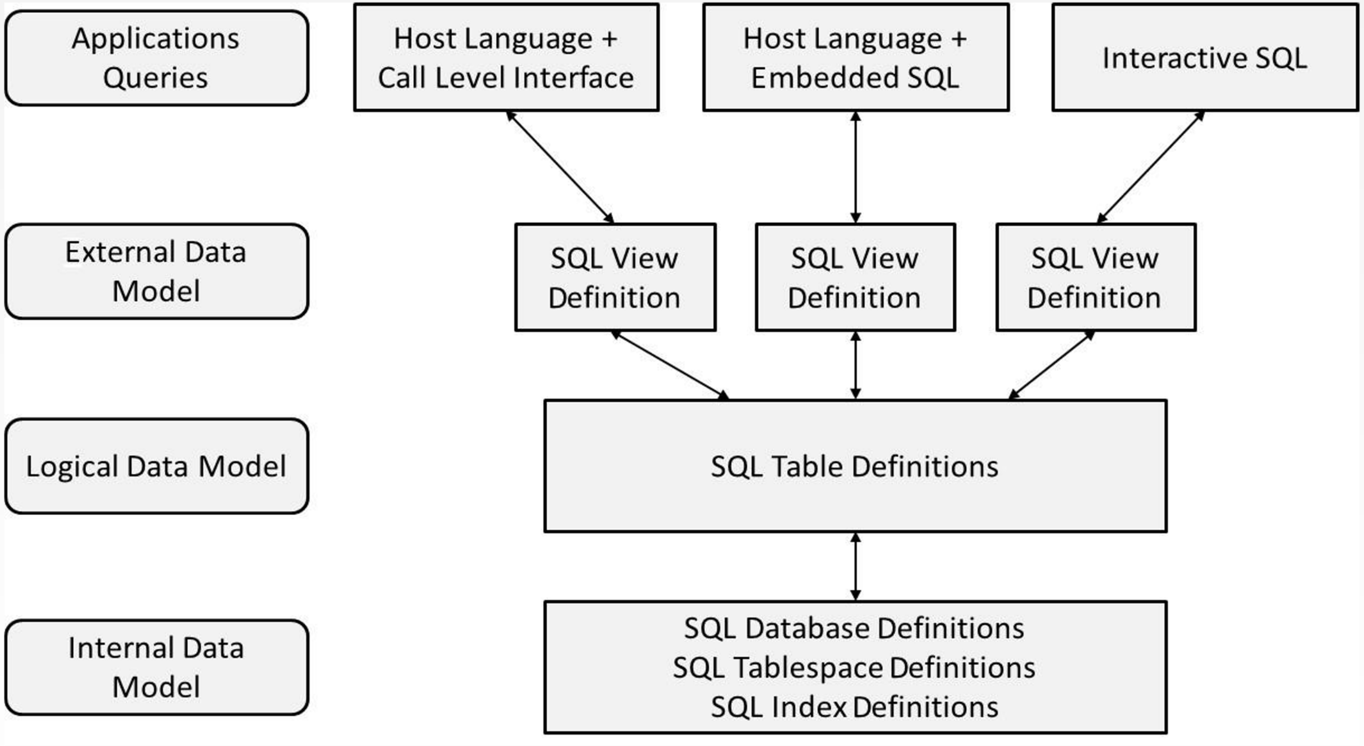


physical data + logical data independence!

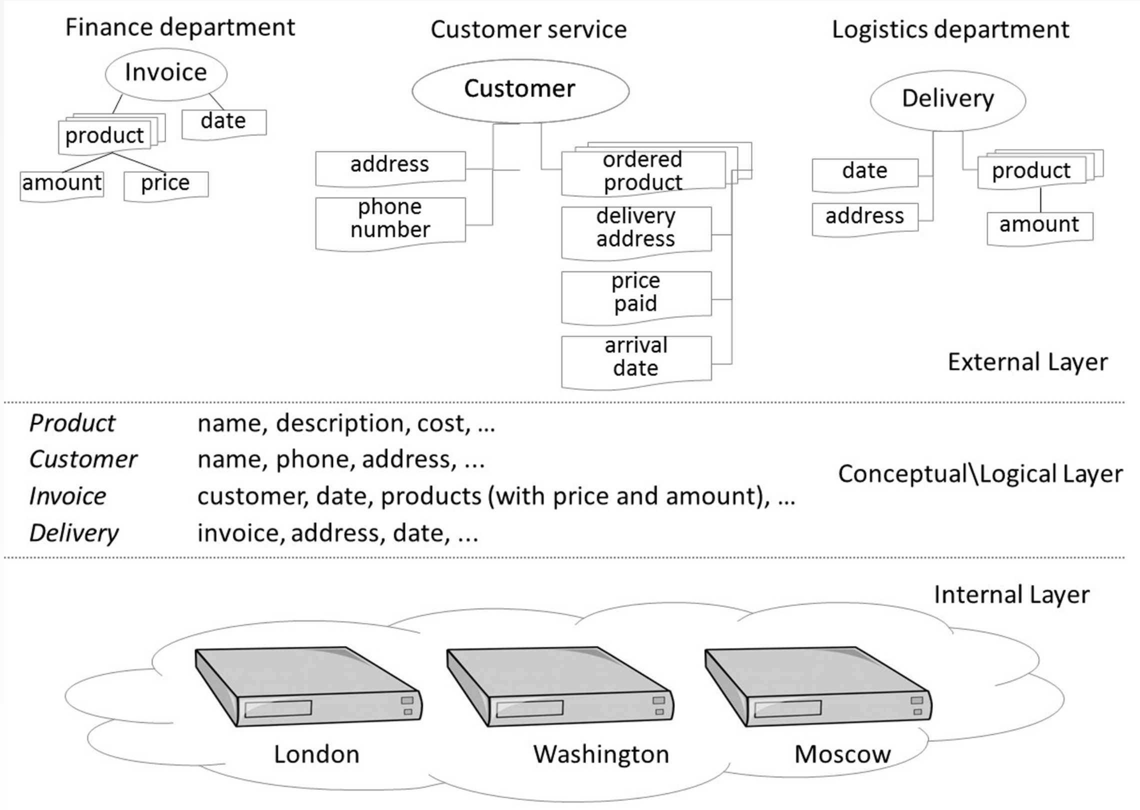
The Three Layer Architecture

- External layer (View level)
  - External data model which includes views
  - Used to control data access and enforce security
- Conceptual\logical layer (Logical level)
  - Contains the conceptual and logical data models
  - E.g., tables
- Internal layer (Physical level)
  - Includes the internal data model
  - E.g., Index
- Changes in one layer should have no to minimal impact on the others
  - Physical data independence
  - Logical data independence

The Three Layer Architecture



The Three Layer Architecture



Catalog

- Contains the data definitions, or metadata, of your database application
- Stores the definitions of the views, logical and internal data models, and synchronizes these three data models to make sure their consistency is guaranteed

Database Users

- Information architect designs the conceptual data model
  - closely interacts with the business user
- Database designer translates the conceptual data model into a logical and internal data model
- Database administrator (DBA) is responsible for the implementation and monitoring of the database
- Application developer develops database applications in a programming language such as Java or Python
- Business user will run these applications to perform specific database operations

Database Languages

- Data Definition Language (DDL) is used by the DBA to express the database's external, logical and internal data models
  - definitions are stored in the **catalog**
- Data Manipulation Language (DML) is used to retrieve, insert, delete, and modify data
  - DML statements can be embedded in a programming language, or entered interactively through a front-end querying tool
- Structured Query Language (SQL) offers both DDL and DML statements for relational database systems

Advantages of DB Systems and DB Management

- Data Independence
- Database Modelling
- Managing Structured, Semi-Structured and Unstructured Data
- Managing Data Redundancy
- Specifying Integrity Rules
- Concurrency Control
- Backup and Recovery Facilities
- Data Security
- Performance Utilities

Data Independence

- Data independence implies that changes in data definitions have minimal to no impact on the applications
- Physical data independence implies that neither the applications, nor the views or logical data model must be changed when changes are made to the data storage specifications in the internal data model
  - DBMS should provide interfaces between logical and internal data models
- Logical data independence implies that software applications are minimally affected by changes in the conceptual or logical data model
  - views in the external data model will act as a protective shield
  - DBMS must provide interfaces between conceptual/logical and external layer

Database Modeling

- A data model is an explicit representation of the data items together with their characteristics and relationships
- A conceptual data model should provide a formal and perfect mapping of the data requirements of the business process and is made in collaboration with the business user
  - translated into logical and internal data model
- Important that a data model's assumptions and shortcomings are clearly documented

Managing Structured, Semi-Structured and Unstructured Data

- Structured data
  - can be described according to a formal logical data model
  - ability to express integrity rules and enforce correctness of data
  - also facilitates searching, processing and analyzing the data
  - E.g., number, name, address and email of a student
- Unstructured data
  - no finer grained components in a file or series of characters that can be interpreted in a meaningful way by a DBMS or application
  - E.g., Invoices, records, emails, audio, weather data, sensor data
  - Note: volume of unstructured data surpasses that of structured data

Managing Structured, Semi-Structured and Unstructured Data

- Semi-structured data
  - data which does have a certain structure, but the structure may be very irregular or highly volatile
  - E.g., individual users' webpages on a social media platform, or resume documents in a human resources database

Managing Data Redundancy

- Duplication of data can be desired in distributed environments to improve data retrieval performance
- DBMS is now responsible for the management of the redundancy by providing synchronization facilities to safeguard data consistency
- Compared to the file approach, the DBMS guarantees correctness of the data without user intervention

Integrity Rules

- Integrity rules are specified as part of the conceptual\logical data model and stored in the catalog
  - Directly enforced by the DBMS instead of applications
  - Syntactical rules specify how the data should be represented and stored
    - E.g., customerID is an integer; birthdate should be stored as month, day and year
  - Semantical rules focus on the semantical correctness or meaning of the data
    - E.g., customerID is unique; account balance should be > 0; customer cannot be deleted if he/she has pending invoices

Concurrency Control

- DBMS has built-in facilities to support concurrent or parallel execution of database programs
- Key concept is a database transaction
  - sequence of read/write operations considered to be an atomic unit in the sense that either all operations are executed or none at all
- Read/write operations can be executed at the same time by the DBMS
- DBMS should avoid inconsistencies!

Concurrency Control

- Lost update problem

Time	T1	T2	balance
t1		Begin transaction	$100
t2	Begin transaction	read(balance)	$100
t3	read(balance)	balance=balance+120	$100
t4	balance=balance-50	write(balance)	$220
t5	write(balance)	End transaction	$50
t6	End transaction		$50

Concurrency Control

- DBMS must support ACID (Atomicity, Consistency, Isolation, Durability) properties
  - Atomicity requires that a transaction should either be executed in its entirety or not all
  - Consistency assures that a transaction brings the database from one consistent state to another
  - Isolation ensures that the effect of concurrent transactions should be the same as if they would have been executed in isolation
  - Durability ensures that the database changes made by a transaction declared successful can be made permanent under all circumstances

Backup and Recovery Facilities

- Backup and recovery facilities can be used to deal with the effect of loss of data due to hardware or network errors, or bugs in system or application software
- Backup facilities can either perform a **full** or **incremental** backup
- Recovery facilities allow to restore the data to a previous state after loss or damage occurred

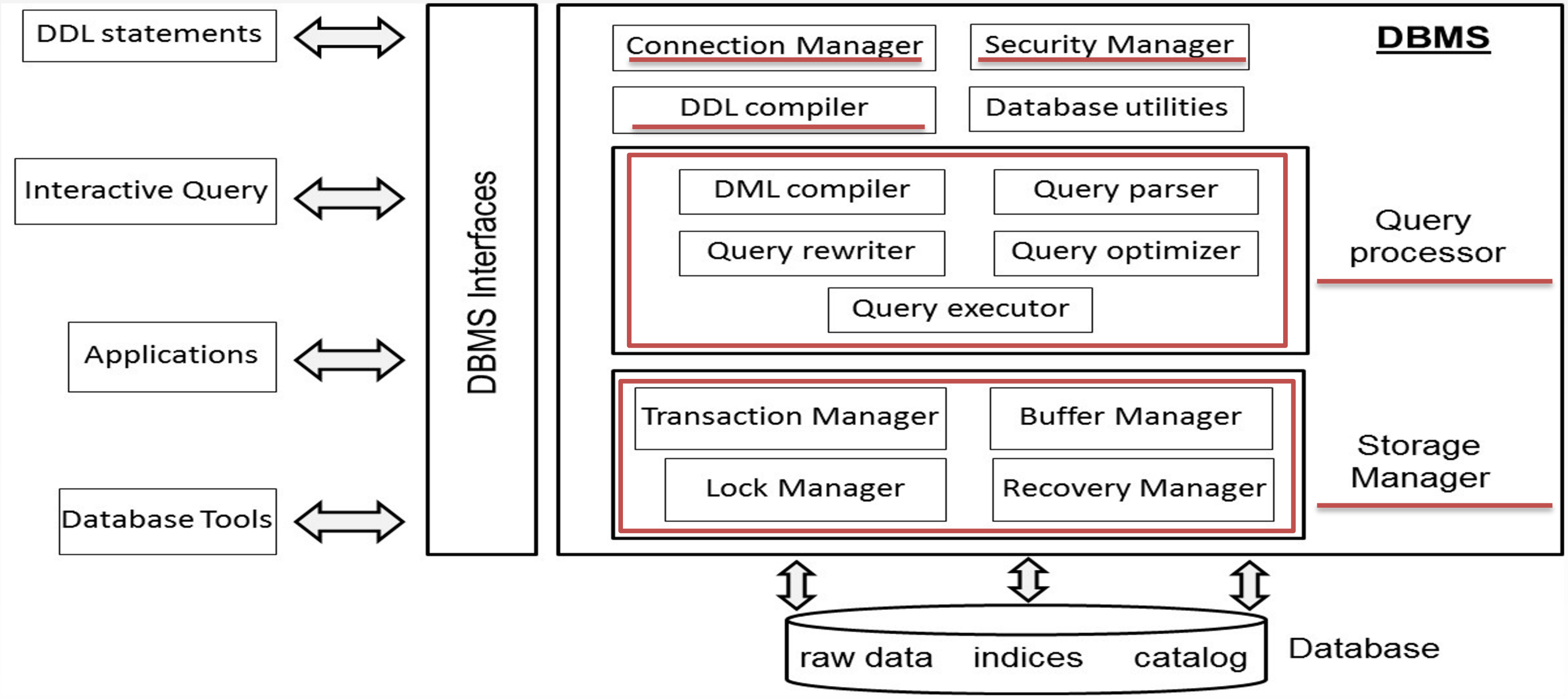
Data Security

- Data security can be enforced by the DBMS
- Some users have read access, whilst others have write access to the data (role-based functionality)
  - E.g., vendor managed inventory (VMI)
- Data access can be managed via logins and passwords assigned to users or user accounts
- Each account has its own authorization rules that can be stored in the catalog

Performance Utilities

- Three KPIs of a DBMS are
  - Response time denoting the time elapsed between issuing a database request and the successful termination thereof
  - Throughput rate representing the transactions a DBMS can process per unit of time
  - Space utilization referring to the space utilized by the DBMS to store both raw data and metadata
- DBMSs come with various types of utilities aimed at improving these KPIs
  - E.g., utilities to distribute and optimize data storage, to tune indexes for faster query execution, to tune queries to improve application performance, or to optimize buffer management

Architecture of a DBMS



Architecture of a DBMS

- Connection and Security Manager
- Data Definition Language (DDL) Compiler
- Query Processor
- Storage Manager

Connection and Security Manager

- Connection manager provides facilities to setup a database connection (locally or through a network)
  - Verifies logon credentials and returns a **connection handle**
  - Database connection can either run as **single process** or as **thread within a process**
- Security manager verifies whether a user has the right privileges
  - read versus write access

DDL Compiler

- Compiles the **data definitions** specified in DDL
- DDL compiler first parses the DDL definitions and checks their syntactical correctness
- DDL compiler then translates the data definitions to an internal format and generates errors if required
- Upon successful compilation, DDL compiler registers the data definitions in the catalog

Query processor

- Query processor assists in the execution of database queries such as retrieval, insertion, update or removal of data
- Key components:
  - Data Manipulation Language (DML) compiler
  - Query parser
  - Query rewriter
  - Query optimizer
  - Query executor

Query Parser and Query Rewriter

- Query parser parses the query into an internal representation format
- Query parser checks the query for syntactical and semantical correctness
- Query rewriter optimizes the query, **independently** of the current database state

Query Optimizer

- Query optimizer optimizes the query based upon the current database state (based upon e.g. predefined indexes)
- Query optimizer comes up with various query **execution plans** and evaluates their **cost** in terms of estimated
  - Number of I/O operations
  - CPU processing cost
  - Execution time
- Estimates based on catalog information combined with statistical inference
- Query optimizer is a key competitive asset of a DBMS

Query Executor

- Result of the query optimization is a final execution plan
- Query executor takes care of the actual execution by calling on the storage manager to retrieve the data requested

Storage manager

- Storage manager governs physical file access and supervises the correct and efficient storage of data
- Storage manager consists of
  - transaction manager
  - buffer manager
  - lock manager
  - recovery manager

Transaction manager

- Transaction manager supervises execution of database transactions
  - a database transaction is a sequence of read/write operations considered to be an atomic unit
- Transaction manager creates a schedule with interleaved read/write operations
- Transaction manager guarantees **ACID** properties
- COMMIT a transaction upon successful execution and ROLLBACK a transaction upon unsuccessful execution

Buffer Manager

- Buffer manager manages buffer memory of the DBMS
- Buffer manager intelligently caches data in the buffer
- Example strategies:
  - Data locality: data recently retrieved is likely to be retrieved again
  - **20/80 law**: 80% of the transactions read or write only 20% of the data
- Buffer manager needs to adopt smart replacement strategy in case buffer is full
- Buffer manager needs to **interact with lock manager**

Lock Manager

- Lock manager provides concurrency control which ensures data integrity at all times
- Two types of locks: **read** and **write** locks
- Lock manager is responsible for assigning, releasing, and recording locks in the catalog
- Lock manager makes use of a *locking protocol* which describes the locking rules, and a lock table with the lock information

Recovery Manager

- Recovery manager supervises the correct execution of database transactions
- Recovery manager keeps track of all database operations in a **log file**
- Recovery manager will be called upon to undo actions of aborted transactions or during crash recovery

Categorization of DBMSs

- Categorization based on data model
- Categorization based on degree of simultaneous access
- Categorization based on architecture
- Categorization based on usage

Categorization based on data model

- Hierarchical DBMSs
  - adopt a tree like data model
  - DML is procedural and record oriented
  - no query processor (logical and internal data model intertwined)
  - E.g., IMS (IBM)
- Network DBMSs
  - use a network data model
  - CODASYL DBMSs
  - DML is procedural and record oriented
  - no query processor (logical and internal data model intertwined)
  - CA-IDMS (Computer Associates)

Categorization based on data model

- Relational DBMSs
  - use the relational data model
  - currently the most popular in industry
  - SQL (declarative and set oriented)
  - query processor
  - strict separation between the logical and internal data model
  - E.g., MySQL (open source, Oracle), Oracle DBMS (Oracle), DB2 (IBM), Microsoft SQL (Microsoft), MariaDB

Categorization based on data model

- Object-Oriented DBMSs (OODBMS)
  - based upon the OO data model
  - No impedance mismatch in combination with OO host language
  - E.g., db4o (open source, Versant), Caché (Intersystems) GemStone/S (GemTalk Systems)
  - only successful in niche markets, due to their complexity

Categorization based on data model

- Object-Relational DBMSs (ORDBMSs)
  - also referred to as extended relational DBMSs (ERDBMSs)
  - use a relational model extended with OO concepts
  - DML is SQL (declarative and set oriented)
  - E.g., Oracle DBMS (Oracle), DB2 (IBM), Microsoft SQL (Microsoft)

Categorization based on data model

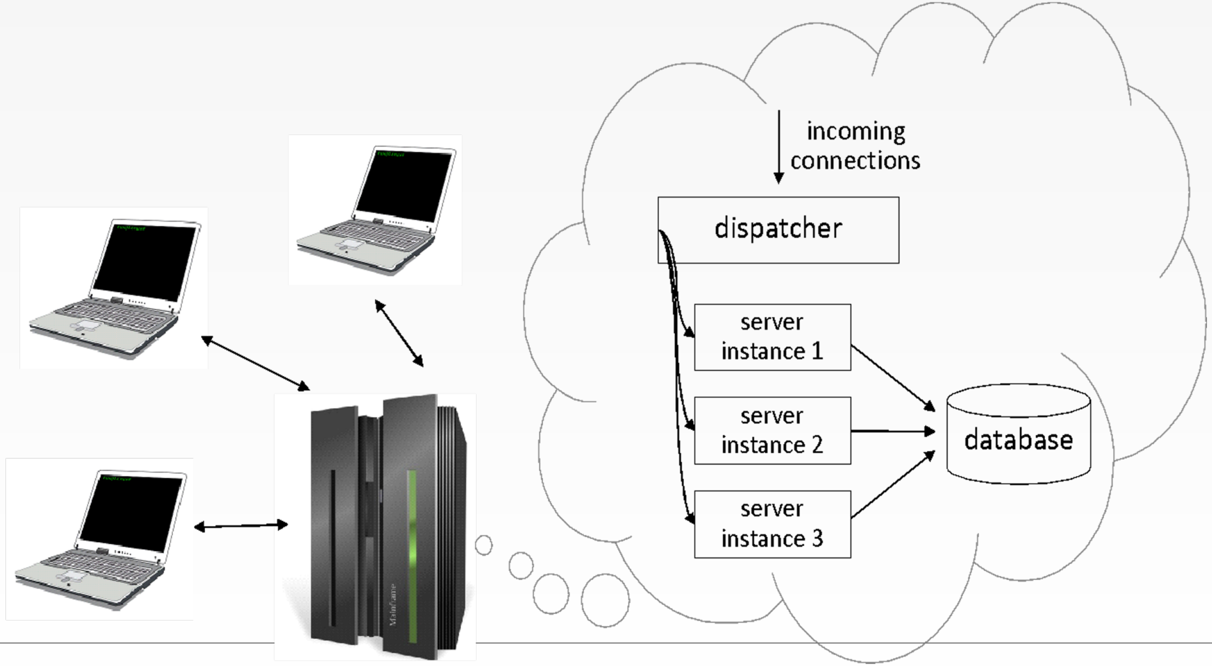
- XML DBMSs
  - use the XML data model to store data
  - Native XML DBMSs (e.g., BaseX, eXist) map the tree structure of an XML document to a physical storage structure
  - XML-enabled DBMSs (e.g., Oracle, IBM DB2) are existing DBMSs that are extended with facilities to store XML data

Categorization based on data model

- NoSQL DBMSs
  - targeted at storing big and unstructured data
  - can be classified into key-value stores, column-oriented databases and graph databases
  - focus on scalability and the ability to cope with irregular or highly volatile data structures
  - E.g., Apache Hadoop, MongoDB, Neo4j

Categorization based upon degree of simultaneous access

- Single user versus multi user systems



Categorization based on architecture

- Centralized DBMS architecture
  - data is maintained on a centralized server (mainframe)
- Client server DBMS architecture
  - active clients request services from passive servers
  - fat server versus fat client variant
- n-tier DBMS architecture
  - client with GUI functionality, application server with applications, database server with DBMS and database, and web server for web based access

Categorization based on architecture

- Cloud DBMS architecture
  - DBMS and database are hosted by a third-party cloud provider
  - E.g., AWS RDS, Microsoft Azure SQL, Apache Cassandra project and Google's BigTable
- Federated DBMS
  - provides a uniform interface to multiple underlying data sources
  - hides the underlying storage details to facilitate data access

Categorization based on architecture

- In-memory DBMS
  - stores all data in internal memory instead of slower external storage (e.g., disk)
  - often used for real-time purposes
  - E.g., Redis, Memcached, and HANA (SAP)

Categorization based on usage

- On-line transaction processing (OLTP)
  - focus on managing operational or transactional data
  - database server must be able to process lots of simple transactions per unit of time
  - DBMS must have good support for processing a high volume of short, simple queries
- On-line analytical processing (OLAP)
  - focus on using operational data for tactical or strategical decision making
  - limited number of users formulates complex queries
  - DBMS should support efficient processing of complex queries which often come in smaller volumes

Categorization based on usage

- Big Data & Analytics
  - NoSQL databases
  - focus on more flexible, or even schema-less, database structures
  - store unstructured information such as emails, text documents, Twitter tweets, Facebook posts, etc.
- Multimedia
  - Multimedia DBMSs provide storage of multimedia data such as text, images, audio, video, 3D games, etc.
  - should also provide content-based query facilities

Categorization Based on Usage

- Spatial applications
  - Spatial DBMSs support storage and querying of spatial data (both 2D and 3D)
  - Geographical Information Systems (GIS)
- Sensoring
  - Sensor DBMSs manage sensor data such as biometric data from wearables, or telematics data

Summary

- Applications of Database Technology
- Key definitions
- File versus Database Approach to Data Management
- Elements of a Database System
- Advantages of Database Systems and Database Management
- Architecture of a DBMS
- Categorization of DBMSs

Reading and Next Class

- Introduction to database systems
- Architecture & Classification
- Reading: Ch1, Ch2
- Next class:
  - The Relational Model and Relational Algebra
  - Ch3, Ch4