Introduction to Databases

Madiba Hudson-Quansah

Contents

Chapter 1

Introduction

1.1 Uses of Databases

Retail - Department stores, supermarkets, mail order companies, etc.

Social Media - Store information about users, their friends, and their activities.

1.2 File Based Systems

Definition 1.2.1: File-Based Approach

A collection of application programs that perform services for the end-users, where each program defines and manages its own data

1.2.1 Disadvantages of File-Based Systems

- Space intensive
- Data redundancy / Duplication - Due to each program having and controlling its own data, if data is needed by multiple programs, it must be duplicated.
- Separation and isolation of data - When data is scattered across different files, it becomes difficult to access data that should be available.
- Data Dependence / Program-Data Dependence - The inherent relationship between the application program and the data it uses. This leads to:
  - Difficulty in updating data
- Incompatible file formats.
- Fixed Queries / Proliferation of application programs - Due to the nature of file-based systems creating new queries is difficult as a developer would have to write the new queries into the program.
- Difficulty in accessing data
- Inconsistent data

1.3 Database Management Systems (DBMS)

1.3.1 Database

Definition 1.3.1: Database

A *shared* collection of *logically* related data, and descriptions of this data, designed to meet the information needs of an organization. / A set of named relations.

Definition 1.3.2: System Catalogue / Data Dictionary

A description of the data in the database (metadata).

Definition 1.3.3: Entity / Record

A distinct real world object that is to be represented in a database / Row.

Definition 1.3.4: Attribute

A property that describes an entity. / Column.

Definition 1.3.5: Relationship

An association between entities. / Table

1.3.2 Database Management System (DBMS)

Definition 1.3.6: Database Management System (DBMS)

A software system that enables users to define, create, maintain, and control access to the database. A DBMS usually provides the following functions:

Data Definition Language (DDL) - Allows users to define a database by describing the data types, structures, and constraints on the data to be stored in the database.

Data Manipulation Language (DML) - Allows users to insert, update, delete, and retrieve data from the database. The DML, then provides a **query language**, which is used for inquiry and reporting.

Controlled Access -

- Ensures that only authorized users can access the database,
- Ensures the consistence of the stored data is maintained
- Manages concurrent access to the database
- Provides a recovery system to ensure consistency of data in the presence of system failures.

1.3.2.1 Properties of a DBMS

Massive Be able to manage large amounts of data.

Persistence Data should be available even after the application has been closed.

Safe Hardware and software failures should not cause data loss.

Multi-user / Concurrent Multiple users should be able to access the database at the same time>

Convenient Be able to declaratively specify queries and operations.

Efficient Be able to perform operations and process queries quickly.

Reliable Be able to recover from failures.

1.3.2.2 Difference between File-Based Systems and DBMS

1.3.2.2.1 The Self-Describing Nature of a Database System

The database system contains the complete definition of the database structure and constraints. This definition is

1.3.2.2.2 Multiple Views of Data

Each user may see a different view of the database, and the DBMS must control the access of these users to the database.

1.3.2.2.3 The Concurrent Access to the Data

Each user must be able to access the data concurrently, and the DBMS must include concurrency control techniques to ensure that the user's operations are correctly synchronized.

Definition 1.3.7: Database System

A DBMS together with the database itself.

1.4 (Database) Application Programs

Definition 1.4.1: (Database) Application Programs

Programs that interact with the DBMS to access the database, using the DBMS's DML and query language.

1.4.1 Views

Definition 1.4.2: View / View Mechanism

A subset of the database, defined using queries. Views provide an abstracted view of the database, hiding irrelevant information from the end-user.

Views not only serve as an abstraction layer but also provide:

A Level of Security - Views can be used to exclude sensitive data to unauthorized parties.

Customization - Views can be used to better present data for end-users

Consistency - Views can be used to preserve the state of the database even though the underlying data may have changed.

1.5 Comparison of File-Based Systems and DBMS

1.5.1 Advantages of Using the DBMS Approach

- Control of data redundancy - Data is stored in a central location, and is not duplicated unnecessarily across multiple programs. As some data will need to be duplicated for various reasons such as performance.
- Data Consistency - If a data item is stored only once in the database, an update to it will reflect in all places it is used, ensuring consistency.
- Data Sharing - Data can be shared across departments, and applications without the need for duplication.
- Improved Data Integrity - Database integrity refers to the validity and consistency of stored data. Integrity is defined in terms of constraints, which are consistency rules that the database is not allowed to violate. With DBMS, the Database Admin can define constraints that the DBMS will enforce.

- Economy of Scale - The cost of developing and maintaining the database is spread over all the applications that use the database.

1.5.2 Disadvantages of Using the DBMS Approach

- Complexity - The DBMS is a complex piece of software, and as such, it requires a high level of expertise to manage.
- Size - The complexity and breadth of functionality usually makes the DBMS large and resource-intensive.
- Cost of DBMS - The cost of acquiring a DBMS can be high.
- Additional hardware costs - The disk storage requirements of a DBMS and the database may require the purchase of additional storage space.
- Higher impact of a failure - The centralization of resources increases the vulnerability of the system to failure as a failure will affect all the applications that use the database which due to centralization, is a lot.
- Performance - The DBMS may not be as efficient as a custom-built file-based system.

1.6 Components of a Database System

- Users - Application programs, DBA, End-users
- Software - Controls the organization, storage, management and retrieval of data.
- Hardware - The physical devices used to store and process the data.
- Data - The data to be stored in the database.

Chapter 2

Database System Environment

2.0.1 Database Users / Roles

Database Administrators Responsible for authorizing access to the database for coordinating and monitoring its use. Acquiring software and hardware resources, controlling its use, and monitoring efficiency of operations.

Data Administrators Responsible for the management of the organization's data resources, including the database planning, development, maintenance of standards, polices, and procedures.

Database Designers Responsible for defining the content structure, constraints, and functions or transactions of the database. Database designers can be split into two categories:

- Logical Database Designers - Focus on the logical structure of the database, i.e. the constraints on data to be stored in the database (business rules).
- Physical Database Designers - Focuses on how the logical database is to be actually implemented. This involves:
  - Mapping the logical structure to a set of tables and integrity constraints
  - Selecting storage structures and access methods
  - Designing security measures required on the data.

Application Developers Responsible for developing the application programs that provide required functionality for the end users, using the implemented database.

End Users Clients of the database, which has been designed and implemented to serve their information needs. The can be classified according to the way the interact with the system.

- Naive Users - Unsophisticated users who interact with the system through application programs.
- Casual - Users who access the database occasionally, and may not be familiar with the system.
- Sophisticated - Users who interact with the system regularly and are familiar with the structure of the database and the facilities provided by the DBMS.
- Stand-alone - Users who maintain personal databases using ready-to-use packaged applications.

2.1 ANSI-SPARC Three-Level Architecture

Definition 2.1.1: ANSI-SPARC Three-Level Architecture

American National Standards Institute (ANSI) Standards Planning and Requirements Committee (SPARC). A database architecture that separates the user's view of the database from the way the data is physically stored.

This architecture is based on the idea that a database system should be divided into three levels of abstraction:

- External Level
- Conceptual Level
- Internal Level

2.1.1 External Level

Definition 2.1.2: External Level

The user's view of the database. Describes the part of the database that is relevant to the user.

This level consists of a number of external views of the database. Each external view includes only the entities, attributes and relationships that are relevant to a particular user group.

2.1.2 Conceptual Level

Definition 2.1.3: Conceptual Level

The community view of the database. Describes what data is stored in the database and the relationships between the data.

This level contains the logical structure of the database as seen by the Database Administrator, i.e., the complete view of the database without considering implementation details. The conceptual level represents:

- All entities, attributes, and relationships
- Data integrity constraints
- Semantic information about the data
- Security information.

"

2.1.3 Internal Level

Definition 2.1.4: Internal Level

The physical representation of the database on the computer. Describes how the data is stored in the database.

- Storage space allocation of data and indexes
- Record description for storage
- Record placement (pointers)
- Data compression and encryption techniques

2.1.4 Data Independence

2.1.4.1 Logical Data Independence

Definition 2.1.5: Logical Data Independence

Possibility for addition/removal of new entities, attributes or relationships. Indicates that the conceptual schema can be changed without affecting the existing external schemas.

This mainly affects the conceptual and external levels of the database, enabling a form of data abstraction from the external views and the conceptual structure of the database.

2.1.4.2 Physical Data Independence

Definition 2.1.6: Physical Data Independence

Possibility for changes to storage structures. Indicates that the physical storage structures or devices could be changed without affecting conceptual schema.

This mainly affects the conceptual and internal levels of the database, decoupling the conceptual description of the database from the actual physical implementation of the database.

2.2 Database Languages

Definition 2.2.1: Database Languages

A language that enables the user to create and maintain the database and provide ways to access and manipulate the data stored in the database.

- Data Definition Language (DDL)
- Data Manipulation Language (DML)

2.2.1 Data Definition Language (DDL)

Definition 2.2.2: Data Definition Language

A language used to define and name, entities, their attributes, and relationships required for the application, along with integrity constraints.

Database schema is specified by a set of definitions defined in a Data Definition Language. The result of statements made in a DDL is a set of tables stored in the system catalogue. The system catalogue couples the definition of these objects to metadata which makes it easier to access and manage the objects. This metadata includes definitions of records, data items, and other objects of interest.

2.2.2 Data Manipulation Language (DML)

Definition 2.2.3: Data Manipulation Language

A language that provides a set of operations to support the basic data manipulation operations on data in the database.

These operations include:

- Insertion
- Retrieval
- Modification
- Deletion

The part of the DML that handles retrieval is called a query language. A query language differs from the whole DML by their underlying retrieval constructs.

Definition 2.2.4: Query Language

High level special-purpose language used to satisfy diverse requests for the retrieval of data held in a database.

There are two types of DML:

- Procedural

- Non-procedural

With the main difference being that Procedural languages specify how the output of a DML statement is to be obtained but Non-procedural DMLs describe only what data is to be obtained.

2.2.2.1 Procedural DMLs

Definition 2.2.5: Procedural DMLs

A language that allows the user to tell the system what data is needed and exactly how to obtain the data

This means the user must express all the data access operations that are to be used by calling the corresponding procedures obtaining the required output. Typically Procedural DMLs process each record individually basing the next record it processes on the result of processing the previous record. This chain of retrievals continues until the data requested has been gathered.

2.2.2.2 Non-Procedural DMLs

Definition 2.2.6: Non-Procedural DMLs

A language that allows the user to state what data is needed rather than how it is to be retrieved.

The DBMS translates the DML statement into procedures that manipulate the required sets of data.

2.3 Data Models

Definition 2.3.1: Data Model

An integrated collection of concepts for describing and manipulation of data. The data model provides the necessary means to achieve data abstraction, which is the process of hiding the low-level details of the database and showing only the relevant data to the user.

A data model comprises of three components:

A Structural Part - Consisting of a set of rules according to which databases can be constructed.

A Manipulative Part - Defining the types of operations that are allowed on the data.

A Set of Integerity Constraints - Ensures that the data is accurate.

Following the ANSI-SPARC architecture, we can identify three related data models:

- An external data model / Universe of Discourse - To represent each user's view of the organization.
- A conceptual data model - To represent the logical view that is DBMS-independent.
- An internal data model - To represent the conceptual schema in a way that can be understood by the DBMS.

There are three main categories of data models:

- Object-Based
- Record-Based
- Physical Data Model

2.3.1 Object-Based Data Model

These categories of model use concepts such as entities, attributes, and relationships to represent data. In this model an *entity* is a representation of a distinct real world object we want to include in the database, an *attribute* is a property that describes some aspect of an entity, a *relationship* is an association between entities. Some of the most common object-based data models include:

- Entity-Relationship
- Semantic
- Functional
- Object-Oriented

2.3.2 Record-Based Data Model

These models are based on the concept of a record, which is a collection of fields, each of which contains one data item. There are three principal types of record-based logical data model:

- Relational Data Model
- Network Data Model
- Hierarchical Data Model.

2.3.3 Physical Data Model

2.4 The Relational Model

2.4.1 Terminology

2.4.1.1 Relational Data Structure

2.4.1.1.1 Database

Definition 2.4.1: Database

A set of named relations.

2.4.1.1.2 Relation

Definition 2.4.2: Relation / Table

A table in a relational database.

2.4.1.1.3 Attribute

Definition 2.4.3: Attribute

A column in a table. Each relation has a named set of attributes.

2.4.1.1.4 Tuple

Definition 2.4.4: Tuple

A row in a table. Each tuple has a value for each attribute.

2.4.1.1.5 Type

Definition 2.4.5: Type / Domain

A description of the types of operations and values an attribute can have.

2.4.1.1.6 Schema

Definition 2.4.6: Schema

A structural description of relations in a database / A structural description of the tables in a database. A database differs from a database by the fact that a database is the implementation of the schema.

2.4.1.1.7 Instance

Definition 2.4.7: Instance

The actual data in a database at a particular point in time.

2.4.1.1.8 Null values

Definition 2.4.8: Null

Unknown / Undefined

2.4.1.1.9 Key

Definition 2.4.9: Key

An attribute unique to each tuple, used to differentiate entities / tuples.

2.4.1.1.10 Domain

Definition 2.4.10: Domain

A domain D is a set of atomic values, that can be assigned to a specific attribute.

2.4.1.1.11 Atomic

Definition 2.4.11: Atomic

An atomic value is indivisible, i.e. at its most basic level.

2.4.1.1.12 Degree

Definition 2.4.12: Degree

The number of attributes in a relation.

2.4.1.1.13 Cardinality

Definition 2.4.13: Cardinality

The number of tuples in a relation.

2.4.1.1.14 Relational Database

Definition 2.4.14: Relational Database

A collection of normalized relations with distinct relation names.

2.4.1.2 Mathematical Relations

Definition 2.4.15: Mathematical Relation

A subset of the Cartesian product of a list of domains.

Domains specify all the possible values that an attribute can take. Suppose we have domains represented as sets D_1, and D_2, where $D_1 = \{2, 4\}$ and $D_2 = \{1, 3, 5\}$. The *Cartesian product* of these sets is:

$$D_1 \times D_2 = \{(2,1), (2,3), (2,5), (4,1), (4,3), (4,5)\}$$

Any subset of this product is a relation. For example, the subset $\{(2,1), (4,1)\}$ can be expressed as the relation R:

$$R = \{(2,1), (4,1)\}$$

Therefore generally for domains D_1 and D_2, a relation R is:

$$R = \{(x, y) \mid x \in D_1 \land y \in D_2\}$$

And due to the properties of n-tuples, a relation S can be defined using the same sets of domains as:

$$S = \{(y, x) \mid x \in D_1 \land y \in D_2\}$$

Therefore for n domains, $D_1, D_2, \ldots, D_n$, any subset of the Cartesian product of these domains is a relation:

$$D_1 \times D_2 \times \ldots \times D_n = \{(d_1, d_2, \ldots, d_n) \mid d_1 \in D_1 \land d_2 \in D_2 \land \ldots \land d_n \in D_n\}$$

Which is represented as

$$\prod_{j=1}^{n} D_j$$

2.4.1.3 Database Relations

2.4.1.3.1 Relation Schema

Definition 2.4.16: Relation Schema

A named relation defined by a set of attribute and domain name pairs.

Let $A_1, A_2, \ldots, A_n$ be attributes with domains $D_1, D_2, \ldots, D_n$. Then set

$$S = \{A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n\}$$

Is a relation schema.

A relation R defined by a relation schema S is a set of mappings from the attribute names to their corresponding domains, thus , R is a set of n-tuples:

$$R = \{(A_1 : d_1, A_2 : d_2, \ldots, A_n : d_n) \mid d_1 \in D_1 \land d_2 \in D_2 \land \ldots \land d_n \in D_n\}$$

2.4.1.3.2 Relational Database Schema

Definition 2.4.17: Relational Database Schema

A set of relation schemas, each with a distinct name

Therefore for relation schemas $R_1, R_2, \ldots, R_n$, a relation database schema S is:

$$S = \{R_1, R_2, \ldots, R_n\}$$

2.4.1.4 Properties of Relations

2.4.2 Relational Keys

2.4.2.1 Superkey

Definition 2.4.18: Superkey

An attribute, or set of attributes, that uniquely identifies a tuple within a relation.

A superkey may include extraneous attributes not necessary for identification.

2.4.2.2 Candidate Key

Definition 2.4.19: Candidate Key

A superkey such that such that no proper subset is a superkey within the relation. An attribute or set of attributes that can be used to uniquely identify a tuple within a relation without any extraneous attributes.

A candidate key K for a relation R has two properties:

Uniqueness In each tuple of R, the value of K uniquely identifies the tuple.

Irreducibility No proper subset of K has the uniqueness property.

Definition 2.4.20: Composite Key

A candidate key that consists of more than one attribute.

2.4.2.3 Primary Key

Definition 2.4.21: Primary Key

The candidate key that is chosen to uniquely identify tuples within a relation.

2.4.2.4 Foreign Key

Definition 2.4.22: Foreign Key

An attribute, or set of attributes, within one relation that matches the candidate key of some (possibly the same) relation.

2.4.3 Integrity Constraints

2.4.3.1 Domain Constraints

Definition 2.4.23: Domain Constraints

The set of allowable values for an attribute.

2.4.3.2 Null

Definition 2.4.24: Null

Represents a value for an attribute that is currently unknown or not applicable for the tuple.

2.4.3.3 Entity Integrity

Definition 2.4.25: Entity Integrity

In a base relation, no attribute of a primary key can be null.

By definition a primary key is a unique identify used to differentiate tuples from one another, this means no subset of the primary key is enough to uniquely identify a tuple. If any part of the primary key were null the implication is that not all the attributes of a tuple are needed to distinguish from other tuples contradicting the definition of primary key.

2.4.3.4 Referential Integrity

Definition 2.4.26: Referential Integrity

In a base relation, a tuple that refers to another relation must refer to an existing tuple in that relation.

This constraint is defined between two related relations to maintain the consistency of the data between the relations.

2.4.3.5 Key Integrity

Definition 2.4.27: Key Integrity

No two distinct tuples in a relation can have the same values for all their attributes.

By definition, a relation is a set of tuples, and a set by definition cannot contain duplicates. Therefore no two tuples in a relation can have the same values for all their attributes, as this would imply that the relation is not a set, as a set cannot contain duplicates.

2.4.3.6 Operations and Integrity Violations

2.5 Software Architecture

Definition 2.5.1: Presentation Layer

Medium of access.

Definition 2.5.2: Application Layer

Software that is accessed by the presentation layer.

Definition 2.5.3: Database Layer

The database

Note:-

Machine means server.

- One-Tier architecture - Presentation layer , Application layer , Database layer on the same machine.
- Two-Tier architecture - Database tier (Application and Database layers) on one machine and client tier (Presentation layer) on another.
- Three-Tier architecture - Each layer on its own machine
- N-Tier architecture - Multiple instances of tiers using replication procedures and load balancing.

Chapter 3

Relational Algebra and Relational Calculus

3.1 The Relational Algebra

Note:-

A relation is a table!

Definition 3.1.1: Relational Algebra

A theoretical language with operations that work on one or more relations to define another relation without changing the original relation. Thus both the input and output of the relational algebra are relations. This allows operation nesting and chaining. This property is known as **Closure**.

3.1.1 Unary Operations

Definition 3.1.2: Unary Operations

Operations that work on a single relation.

There are two types of unary operations:

- Selection / Restriction
- Projection

3.1.1.1 Selection

Definition 3.1.3: Selection

Works on a single relation R, and defines a relation that contains only those tuples of R that satisfy the specified condition called a predicate. The selection operation is denoted by:

$$\sigma_{\text{predicate}}(R)$$

Where R is the relation and the predicate is the condition that must be satisfied.

Question 1

List all staff that earn more than 1000.

Solution: Therefore the selection operation is:

$$\sigma_{\text{salary}>1000}(\text{Staff})$$

And in SQL this would be:

```
    SELECT * FROM Staff WHERE salary > 1000;
```

3.1.1.2 Projection

Definition 3.1.4: Projection

Also works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates. The projection operation is denoted by:

$$\Pi_{a_1,\ldots,a_n}(R)$$

Question 2

Produce a list of salaries for all staff, showing only the staffNo, fName, IName, and salary details.

Solution:

The projection operation is:

$$\Pi_{\text{staffNo, fName, lName, salary}}(\text{Staff})$$

And in SQL this would be:

```
  SELECT staffNo, fName, lName, salary FROM Staff
```

3.2 Set / Binary Operations

3.2.1 Cartesian Product

Definition 3.2.1: Cartesian Product

A binary operation that defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S. The Cartesian product is denoted by:

$$R \times S$$

The Cartesian product operation multiples two relations resulting in another relation consisting of all the possible pairs of the tuples from the two relations, therefore one relation has I tuple and N attributes and another has J tuples and M attributes, the resulting relation will have $I \times J$ tuples and $N + M$ attributes

Question 3

List the names and comments of all clients who have viewed a property for rent

Solution: The names of clients are stored in the Client relation and the details of viewings are stored in the Viewing relation. To obtain the list of clients and the comments on properties they have viewed we need to combine these two relations:

$$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \times (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$$

or

$$\Pi_{\text{Client.clientNo, Viewing.clientNo, fName, lName, propertyNo, comment}}(\text{Client} \times \text{Viewing})$$

In SQL:

```
SELECT Client.clientNo, Viewing.clientNo, fName, lName, propertyNo, comment FROM Client
, Viewing
```

But this operation contains more information than required, therefore to obtain the required list we need to perform as selection operation to extract only the tuples where the clientNo of the Client relation is equal to the clientNo of the Viewing relation:

$$\sigma_{\text{Client.clientNo=Viewing.clientNo}}\left((\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \times (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))\right)$$

or

$$\Pi_{\text{Client.clientNo, Viewing.clientNo, fName, lName, propertyNo, comment}}\left(\sigma_{\text{Client.clientNo=Viewing.clientNo}}(\text{Client} \times \text{Viewing})\right)$$

In SQL:

```sql
SELECT Client.clientNo, Viewing.clientNo, fName, lName, propertyNo, comment FROM Client
, Viewing WHERE Client.clientNo = Viewing.clientNo
```

3.3 Join Operations

3.3.1 Natural Join

Definition 3.3.1: Equi-Join

A join operation that combines tuples from two relations based on a related attribute. Can also be described in terms of Theta-Join, where θ is specifically =. Denoted by:

$$R \bowtie_{R.a=S.b} S = \sigma_{R.a=S.b}(R \times S)$$

Where R and S are relations and a is an attribute of R and b is an attribute of S.

Definition 3.3.2: Natural Join

An equi-join, of the two relations R and S over all common attributes x. One occurrence of each common attribute is retained in the result, i.e. removing duplicate attributes. Denoted by:

$$R \bowtie S$$

The degree of a natural join is the sum of the degrees of the relations R and S minus the number of common attributes x.

This means the example operation performed with the Cartesian product can be simplified to:

$$\Pi_{\text{clientNo, fName, lName, propertyNo, comment}}(\text{Client} \bowtie \text{Viewing})$$

And in SQL:

```sql
SELECT clientNo, fName, lName, propertyNo, comment FROM Client NATURAL JOIN Viewing
```