Database Management System
CSE 2221

Noortaz Rezoana

April 2024

Chapter 1: Introduction

Data Definition Language (DDL)

Specification notation is a formal way to define the structure of a database schema. In the example provided, we use SQL (Structured Query Language) to specify the creation of a table named "instructor" with four columns: ID, name, dept_name, and salary. Each column has a specified data type and length.

```
create table instructor (
    ID                  char(5),
    name                varchar(20),
    dept_name           varchar(20),
    salary              numeric(8,2)
)
```

When we execute this SQL statement, a Data Definition Language (DDL) compiler generates a set of table templates stored in a data dictionary. This data dictionary serves as a repository for metadata, which is essentially data about the data.

The data dictionary includes crucial information such as:

- **Database Schema**: The overall structure of the database, including tables, columns, and their data types.
- **Integrity Constraints**: Rules that enforce the validity and consistency of data, such as ensuring that certain fields cannot be left blank or that values in one table match values in another.
- **Primary Key**: A unique identifier for each record in a table. In our example, the "ID" column serves as the primary key, ensuring that each instructor has a unique identifier.
- **Authorization**: Specifies who can access what data within the database. This includes permissions for reading, writing, updating, and deleting data, ensuring data security and privacy.

By utilizing specification notation and maintaining a data dictionary, database administrators can effectively manage the structure, integrity, and security of their databases.

Data Manipulation Language (DML)

Data Manipulation Language (DML) is a language used for accessing and updating the data organized by the appropriate data model. DML is also commonly referred to as a query language.

There are two primary types of Data Manipulation Language:

1. **Procedural DML**: Procedural DML requires a user to specify what data are needed and how to get those data. In other words, it involves step-by-step instructions on how to manipulate the data.
2. **Declarative DML**: Declarative DML, on the other hand, requires a user to specify what data are needed without specifying how to get those data. Instead of giving detailed instructions, declarative DML focuses on stating the desired outcome. Declarative DMLs are usually easier to learn and use than procedural DMLs. They are also referred to as non-procedural DMLs.

θ The portion of a DML that involves information retrieval is called a query language. Query languages allow users to retrieve specific data from a database based on defined criteria.

θ Procedural DMLs provide a detailed approach to data manipulation, requiring users to specify each step of the process. While this can offer more control, it often requires a deeper understanding of the underlying mechanisms.

θ In contrast, declarative DMLs abstract away the implementation details, allowing users to focus on the desired outcome rather than the specific steps needed to achieve it. This makes them more intuitive and user-friendly, particularly for those without extensive programming experience.

θ By understanding the distinctions between procedural and declarative DMLs, database users can choose the most appropriate approach based on their specific needs and expertise.

SQL Query Language

SQL (Structured Query Language) is a nonprocedural query language commonly used for managing and manipulating relational databases. Unlike procedural languages, SQL focuses on specifying what data is needed rather than how to retrieve it.

A SQL query takes as input one or more tables and always returns a single table as output. For example, to find all instructors in the Computer Science department, we can use the following SQL query:

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

SQL is not a Turing machine equivalent language, meaning it does not have the computational power to perform arbitrary computations like a Turing machine. Instead, SQL is designed specifically for querying and manipulating relational databases.

To perform complex computations, SQL is often embedded within higher-level programming languages. Application programs typically access databases through one of two methods:

1. **Language Extensions**: Some programming languages provide extensions to embed SQL directly within the code. This allows seamless integration of database operations with application logic.
2. **Application Program Interface (API)**: APIs such as ODBC (Open Database Connectivity) and JDBC (Java Database Connectivity) provide a standardized way for applications to communicate with databases. These APIs allow SQL queries to be sent to a database and retrieve results programmatically.

By leveraging SQL and appropriate interfaces, developers can efficiently interact with databases and incorporate data manipulation capabilities into their applications.

Database Design

Database design is the process of designing the general structure of a database to efficiently store and manage data. It involves making decisions about the logical and physical organization of the database.

Logical Design

Logical design focuses on deciding the database schema, which involves determining the structure of the database at a conceptual level. This stage of database design requires finding a "good" collection of relation schemas that accurately represent the data and support the required functionality.

During logical design, several key decisions are made:

- **Business Decision**: What attributes should be recorded in the database? This involves understanding the requirements of the business or organization and identifying the essential data elements to be stored.
- **Computer Science Decision**: What relation schemas should be created, and how should the attributes be distributed among them? This decision involves translating business requirements into database structures, determining the entities, attributes, and relationships that need to be represented.

Physical Design

Physical design focuses on deciding the physical layout of the database, including storage structures, indexing, and optimization techniques. This stage aims to optimize the performance and efficiency of database operations by organizing data in a way that minimizes storage requirements and maximizes access speed.

During physical design, considerations include:

- Storage organization, such as file organization and indexing methods.
- Data compression and encryption techniques for security and resource optimization.

- Optimization of query performance through indexing, partitioning, and caching strategies.

By carefully designing the logical and physical structure of the database, database designers can ensure that the database meets the requirements of the organization while optimizing performance and resource utilization.

Query Processor

The query processor is a crucial component of a database management system responsible for interpreting and executing queries. It consists of several key components:

- **DDL Interpreter**: The Data Definition Language (DDL) interpreter interprets DDL statements, such as creating or modifying database objects (e.g., tables, indexes), and records the definitions in the data dictionary. This ensures that the database schema remains consistent and up-to-date.
- **DML Compiler**: The Data Manipulation Language (DML) compiler translates DML statements written in a query language (e.g., SQL) into an evaluation plan consisting of low-level instructions that the query evaluation engine understands. This process is known as query compilation.
  - **Query Optimization**: The DML compiler also performs query optimization, which involves selecting the most efficient evaluation plan from among the various alternatives. Optimization aims to minimize the time and resources required to execute queries by considering factors such as access paths, join algorithms, and indexing strategies.
- **Query Evaluation Engine**: The query evaluation engine executes the low-level instructions generated by the DML compiler. It processes data retrieval, manipulation, and other operations specified in the query. The evaluation engine interacts directly with the underlying storage and indexing structures to retrieve and update data efficiently.

By coordinating the interpretation, compilation, optimization, and execution of queries, the query processor plays a critical role in ensuring the efficient and accurate operation of a database management system.

Query Processing

Query processing is the process of executing a query in a database management system. It involves several stages, including:

1. **Parsing and Translation**:

   Parsing involves breaking down the query into its constituent parts, such as keywords, identifiers, and operators. The parsed query is then translated into an internal representation that the system can understand and process. This translation step ensures that the query is syntactically and semantically correct.

2. **Optimization**:

   Optimization involves finding the most efficient execution plan for the query. This includes selecting appropriate access paths, join methods, and indexing strategies to minimize the time and resources required to execute the query. Optimization aims to improve query performance by reducing the overall cost of query execution.

3. **Evaluation**:

   Evaluation is the final stage of query processing, where the optimized query plan is executed to retrieve the requested data. This involves accessing data from storage, applying any necessary operations (such as filtering, sorting, and aggregation), and returning the results to the user. Evaluation ensures that the query is executed correctly and produces the desired output.

By systematically processing queries through parsing, optimization, and evaluation stages, the database management system can efficiently handle user queries and retrieve relevant data from the database.

Database Architecture

Database architecture encompasses various structures tailored to specific needs:

- **Centralized Databases**:
  - One to a few cores, shared memory.
  - All data stored on a single server.
  - Simple management but may lack scalability.
- **Client-Server**:
  - One server serves multiple client machines.
  - Clients send requests to the server for data processing.
  - Better scalability and workload distribution.
- **Parallel Databases**:
  - Utilizes multiple cores and shared memory.
  - Includes shared-disk and shared-nothing architectures.
  - Allows simultaneous processing of data across multiple units.
- **Distributed Databases**:
  - Handles data storage and processing across multiple geographical locations.
  - Addresses challenges of geographical distribution and schema/data heterogeneity.
  - Uses techniques like data replication and distributed query processing.

Each architecture offers unique advantages and considerations based on factors such as scalability, performance, and geographical distribution.

Database Applications

Database applications are typically structured into two or three parts:

- **Two-tier Architecture**:
  - The application resides at the client machine.
  - It directly invokes database system functionality at the server machine.
  - This architecture simplifies communication but may lead to performance issues on the client side.
- **Three-tier Architecture**:
  - The client machine acts as a front end and does not contain any direct database calls.
  - The client communicates with an application server, often through a forms interface.
  - The application server, in turn, communicates with a database system to access data.
  - This architecture separates the user interface, application logic, and data storage layers, allowing for scalability and flexibility.

Each architecture offers different trade-offs in terms of performance, scalability, and maintenance. The choice depends on factors such as application complexity, user requirements, and scalability needs.

Two-tier and Three-tier Architectures

In the Two-tier architecture:

- The user can directly manipulate the database through the application.

In the Three-tier architecture:

- The user manipulates an application client.
- The application client communicates with an application server.
- The application server manipulates the data in the database.

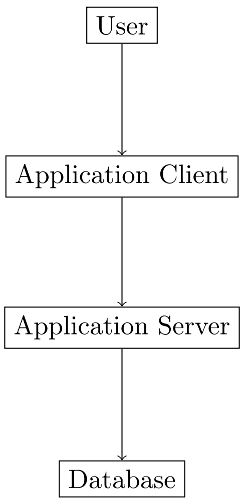

Figure 1: Three-tier Architecture

Database Administrator

A person who has central control over the system is called a database administrator (DBA). The functions of a DBA include:

- **Schema Definition**: Defining the logical structure of the database, including tables, columns, and relationships.
- **Storage Structure and Access-Method Definition**: Determining how data is stored on disk and accessed by users and applications.
- **Schema and Physical-Organization Modification**: Making changes to the database schema or physical organization to accommodate evolving requirements.
- **Granting of Authorization for Data Access**: Managing user permissions and access levels to ensure data security and integrity.
- **Routine Maintenance**: Performing regular maintenance tasks to keep the database running smoothly, such as index rebuilds and statistics updates.
- **Periodically Backing up the Database**: Creating backups of the database to protect against data loss due to hardware failures, disasters, or human error.
- **Ensuring Enough Free Disk Space**: Monitoring disk space usage and ensuring that sufficient space is available for normal operations. Upgrading disk space as required to accommodate growing data volumes.
- **Monitoring Jobs Running on the Database**: Monitoring and managing background processes, queries, and other jobs running on the database to ensure optimal performance and resource utilization.

By fulfilling these functions, database administrators play a critical role in maintaining the stability, security, and performance of the database system.