ADVANCED Database Systems

Edited by | **Waqas Ahmed**

TAP

ADVANCED DATABASE SYSTEMS

Edited by

Waqas Ahmed

Toronto Academic Press

Advanced Database Systems

Waqas Ahmed

Toronto Academic Press
224 Shoreacres Road
Burlington, ON L7L 2H2
Canada
www.tap-books.com
Email: orders@arclereducation.com

ISBN: 978-1-77956-173-2 (e-book)

ISBN: 978-1-77469-761-0

Toronto Academic Press publishes wide variety of books and eBooks. For more information about Toronto Academic Press and its products, visit our website at www.tap-books.com.

ABOUT THE EDITOR

Waqas Ahmed completed his Ph.D in Cyber security from East London University in 2022. He was awarded for academic excellence in his Master's degree. He qualified data scientist in January 2020. He has more than 5 years of teaching and research experience at different universities. Waqas Ahmed published 5 research papers, 3 conference papers, and 1 book chapter in well-established journals. He works in the area of computer networks, system security, and programming. He loves to read and share interesting aspects of computer science in books. In his free time, he loves to travel and explore and give talks on spirituality, ancient customs, and traditions.

Contents

List of Figures

List of Tables

List of Abbreviations

AML	anti-money-laundering
BI	business intelligence
CAD	computer-aided design
CMS	content management systems
DBaaS	database as a service
DBAs	database administrators
DBMS	database management system
DCL	data control language
DDL	data definition language
DML	data manipulation language
DQL	data query language
EDWs	enterprise data warehouses
ER	entity-relationship
ETL	extract, transform, load
GIS	geographic information systems
IDS	integrated data store
IMS	information management system

JSON	JavaScript object notation
MTSDBMS	Michigan terminal system database management system
OLTP	online transaction processing
OODBs	object-oriented databases
OOP	object-oriented programming
ORDBs	object-relational databases
RDBMS	relational database management system
SOA	service-oriented architecture
SOX	SarbanesOxley
SQL	structured query language

PREFACE

Advanced database systems refer to complex and specialized systems designed to handle large amounts of data while ensuring reliability, security, and efficiency. These systems incorporate advanced features such as distributed computing, in-memory processing, and cloud-based solutions to improve performance and scalability. They are essential for organizations dealing with large amounts of data and require efficient data management and processing solutions. Advanced database systems play a critical role in various industries, such as finance, healthcare, e-commerce, and social media, by providing a unified platform for data storage and management. As data continues to grow exponentially, advanced database systems will continue to evolve to meet the increasing demand for efficient and effective data management solutions. In today's data-driven world, Advanced Database Systems have become an essential topic for anyone working in the field of information technology. It provides the necessary skills and knowledge to design, develop, deploy, and maintain complex database systems that meet the demands of modern businesses and organizations.

The field of database systems has undergone a rapid transformation in recent years with the emergence of new technologies and innovations. The book "Advanced Database Systems" aims to provide a comprehensive overview of these advancements and the challenges they present. The book is divided into eight chapters, each of which covers a critical topic in modern database systems.

Chapter 2 provides an introduction to database systems and their role in modern business and society. Chapter 2 also discusses the different types of database systems, including relational, NoSQL, and object-oriented databases.

Chapter 3 provides an overview of various database models, including the hierarchical, network, and object-oriented models.

Chapter 4 delves into big data analytics and how it has changed the way we store and manage data. Chapter 5 covers stream processing systems, including the various algorithms used for data analysis in real time.

Chapter 6 focuses on cloud-based database systems, including the benefits and challenges of cloud computing. Chapter 7 examines the main memory database systems, which have become increasingly popular due to their speed and scalability.

Finally, Chapter 8 covers advanced database security, including the various techniques used to protect against cyber threats and unauthorized access.

This book is intended for students and professionals in the field of database systems, as well as anyone interested in the advancements and challenges of modern database technologies. We hope that this book provides a valuable resource to its readers and helps them gain a deeper understanding of advanced database systems.

CHAPTER 1

INTRODUCTION TO DATABASE SYSTEMS

UNIT INTRODUCTION

In today's digital age, we generate and consume vast amounts of data on a daily basis, ranging from personal information to financial transactions, social media interactions, and more. Database systems play a critical role in the management and storage of this data, making it easily accessible and organized for efficient retrieval and processing. Database systems can be defined as software systems that manage and store large amounts of data, allowing users to retrieve, add, update, and delete data as needed (Hellerstein & Stonebraker, 2005). They are used in a variety of industries, including e-commerce, healthcare, finance, education, logistics, and many more.

The concept of database systems has been around for several decades, with the first database management system (DBMS) developed in the 1960s. The early DBMSs were hierarchical and network models, which were eventually replaced by the relational model in the 1980s. The relational model is still widely used today and is considered the foundation of modern database systems. Database systems consist of several key components, including Data Definition Language (DDL), Data Manipulation Language (DML), Data Query Language (DQL), Data Control Language (DCL), and Database Management System (DBMS) (Yeung & Hall, 2007). DDL is used to define the structure of the database, including tables, relationships, and constraints. DML is used to manipulate the data in the database, including adding, updating, and deleting data. DQL is used

to retrieve data from the database, and DCL is used to control access to the data in the database. DBMS is the software system that manages the database, providing tools for data storage, retrieval, and processing. DBMSs can be classified into several types, including relational, NoSQL, object-oriented, and others (Jukic et al., 2014). Data storage and retrieval is a critical component of database systems. Primary storage refers to the use of main memory, which provides fast access to data but has limited storage capacity. Secondary storage refers to the use of hard disks or solid-state drives, which provide larger storage capacity but slower access speeds.

Indexing involves creating a data structure that enables swift access to data based on particular conditions, thereby enhancing data retrieval speed. On the other hand, query optimization is employed to boost the efficiency of data retrieval by selecting the most optimal query plan based on the complexity and size of the data. Query execution is the process of executing the selected query plan and returning the results to the user. Database systems have several key characteristics that make them critical for managing large amounts of data. These include data independence, concurrent access, data integrity, security, and scalability. Data independence refers to the ability to change the structure of the database without affecting the applications that use the data (Widom & Ceri, 1995). Concurrent access refers to the ability to allow multiple users to access the data at the same time. Data integrity refers to the ability to maintain the accuracy and consistency of the data in the database (Elmasri & Navathe, 2006). Security refers to the ability to control access to the data in the database to prevent unauthorized access. Scalability refers to the ability to handle increasing amounts of data without compromising performance.

The architecture of database systems can be classified into several types, including client-server architecture, tiered architecture, and distributed architecture. Client-server architecture involves a client that sends requests to a server, which processes the requests and returns the results to the client (Coronel & Morris, 2016). Tiered architecture involves multiple layers of servers, with each layer responsible for specific functions such as web serving, application serving, and database serving. Distributed architecture involves multiple servers distributed across different locations, with each server responsible for a subset of the data. Query processing and optimization are critical components of database systems. Query processing involves several phases, including parsing, translation, optimization, and execution.

Learning Objectives

At the end of this chapter, the readers will be able to:

- Understand the definition of database systems.
- Recognize the importance of database systems in the modern world.
- Trace the brief history of database systems.
- Identify the key components of database systems, including DDL, DML, DQL, DCL, and DBMS.

- Describe the process of data storage and retrieval, including primary storage, secondary storage, indexing, query optimization, and query execution.
- Explain the characteristics of database systems, such as data independence, concurrent access, data integrity, security, and scalability.
- Compare and contrast the different types of architecture in database systems, including client-server, tiered, and distributed architectures.
- Understand the process of query processing and optimization, including query processing phases, query optimization techniques, and query execution plans.
- Appreciate the importance of database systems in modern society and how they contribute to the management and processing of large amounts of data.

Key Terms

- Client-Server Architecture
- Data
- Data Definition Language (DDL)
- Data Independence
- Database
- Database
- Database Management System (DBMS)
- Query Optimization
- Scalability

1.1. BASICS OF DATABASE SYSTEMS

Following are the basics of database systems:

1.1.1. Data

Data is any distinct piece of information that is recorded or stored in a computer or other electronic device. It is a collection of unorganized facts, figures, and statistics that are meaningless until they are processed and turned into useful information. Data can be in different forms, including text, numbers, images, audio, and video. In computing, data is the foundation of any computer program, software, or system (Sokolinsky, 2004). It is the input that the computer uses to perform calculations, process instructions, or produce output. Data is stored in a variety of ways, such as databases, spreadsheets, text files, binary files, and multimedia files. The way in which data is stored and organized can affect its usefulness and accessibility.

Data is essential for decision-making, research, analysis, and problem-solving in many fields, including business, science, medicine, education, and government. For example, in business, data is used to monitor sales, track inventory, analyze customer behavior, and make strategic decisions. In medicine, data is used to study diseases, monitor patient health, and develop new treatments. In science, data is used to test hypotheses, make predictions, and create models. There are two types of data: structured and unstructured. Structured data is organized in a specific format, such as a spreadsheet or database, and can be easily processed and analyzed. Unstructured data, on the other hand, is not organized in a specific format and may include text, images, or multimedia (Maryanski et al., 1986). It is more difficult to process and analyze unstructured data, but it can provide valuable insights and information (Figure 1.1).

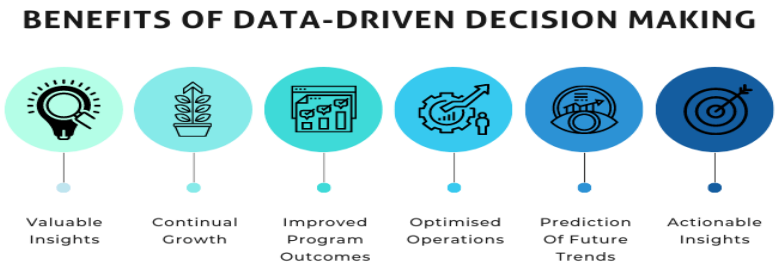


Figure 1.1. *Illustration of the data-driven decision making.*

Source: Jason Williams, creative commons license.

Data can also be classified based on its quality. High-quality data is accurate, complete, relevant, and timely (Chaudhri et al., 2003). It is free from errors, inconsistencies, and duplications. Low-quality data, on the other hand, may be incomplete, outdated, or contain errors, which can lead to incorrect conclusions or decisions. In recent years, the amount of data generated has increased exponentially due to the widespread use of technology and the internet. This has led to the development of big data, which refers to large datasets that are too complex or unstructured to be processed using traditional methods. Big data requires specialized tools and techniques, such as data mining, machine learning, and artificial intelligence, to extract insights and patterns.

1.1.2. Database

A database is an organized collection of structured data that is stored and managed in a computer or other electronic device. It is designed to store and retrieve information efficiently and accurately. Databases can be used for a wide range of applications, from small personal projects to large enterprise systems (Elmasri & Navathe, 2006). A database typically consists of one or more tables that are related to each other. Each table contains a set of records, and each record represents a specific instance of the data that is being stored. The columns in a table represent different attributes of the data, such as name, age, or address. Each column is given a data type that specifies the type of data that can be stored in it, such as text, number, or date (Figure 1.2).

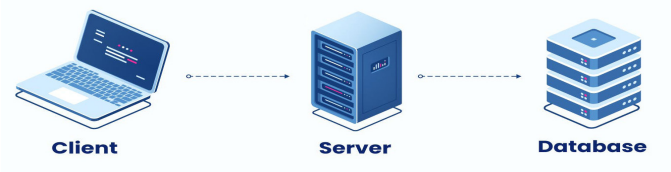


Figure 1.2. Schematic of a database.

Source: Night born, creative commons license.

The relationships between tables in a database are defined by a set of rules called database constraints. These constraints ensure that the data is accurate and consistent by enforcing rules such as unique values, data type restrictions, and referential integrity. Referential integrity ensures that the data in one table is consistent with the data in another table by enforcing a set of rules that dictate how data can be entered, updated, or deleted (Connolly & Begg, 2005). Databases can be categorized based on

their organization and management approach. The most common types of databases are relational databases, which organize data into tables with predefined relationships between them, and non-relational databases, which organize data in other ways such as key-value pairs or documents. Relational databases use a standardized language called Structured Query Language (SQL) to manipulate data (Bernstein & Goodman, 1981). SQL is a powerful language that allows users to create, modify, and query databases. It can be used to perform a wide range of operations, from simple queries to complex data transformations.

KEYWORD

An inventory management system (or inventory system) is the process by which you track your goods throughout your entire supply chain, from purchasing to production to end sales.

Non-relational databases, on the other hand, use a variety of languages and data models to manage data. Some of the most popular non-relational databases include MongoDB, Cassandra, and Redis (Ulusoy, 1995). Databases are used in a wide range of applications, from small-scale personal projects to large-scale enterprise systems. Some common applications of databases include e-commerce websites, inventory management systems, banking systems, healthcare systems, and social media platforms.

1.2. HISTORY OF DATABASE SYSTEMS

The history of database systems dates back to the 1960s, when the first electronic computers were developed. During this time, data was primarily stored on magnetic tapes and punched cards, and the processing of data was time-consuming and error-prone. In the 1970s, Edgar F. Codd, a researcher at IBM, introduced the concept of relational databases. Codd proposed that data should be organized into tables with a fixed number of columns, each with a defined data type, and that relationships between tables should be established using primary and foreign keys. This approach was designed to simplify data management and improve data integrity (Grad & Bergin, 2009).

The first commercially available relational database management system (RDBMS) was developed by IBM in the late 1970s. This system, called System R, was based on Codd's relational model and was used internally by IBM for several years before being released to the public (Miklau et al., 2007). In the 1980s and 1990s, the use of database systems became more widespread as the cost of computing hardware decreased and the need for efficient data management increased. During this time, several new RDBMSs were developed, including Oracle, Sybase, and Microsoft SQL Server. In the late 1990s and early 2000s, the emergence of

the internet and the growth of e-commerce led to the development of new database systems designed to handle large volumes of web-based data. These systems, known as NoSQL databases, were based on non-relational data models and were designed to be more flexible and scalable than traditional RDBMSs (Abadi et al., 2013).

Today, database systems continue to evolve and improve, with new technologies such as cloud computing, big data analytics, and machine learning playing an increasingly important role. The use of databases has become ubiquitous, with virtually every organization relying on some form of database system to manage their data.

1.3. IMPORTANCE OF DATABASE SYSTEMS

Database systems play a critical role in modern organizations by providing a centralized and efficient way to store, manage, and access large amounts of data. The importance of database systems can be understood from the following perspectives:

1.3.1. Data Integration and Data Quality

Database systems allow organizations to integrate data from multiple sources, ensuring data consistency and accuracy (Cai & Zhu, 2015). By eliminating data redundancy, database systems help to maintain data integrity and enhance data quality.

1.3.2. Decision Making

Database systems provide users with the ability to access and analyze data quickly and easily, allowing organizations to make informed decisions based on accurate and up-to-date information (Azeroual et al., 2018). This is particularly important for organizations operating in dynamic environments where quick decisions can make a significant difference.

1.3.3. Efficient Data Management

Database systems provide an efficient way to manage data, including indexing, searching, and updating data. This helps to reduce data entry errors, increase data consistency, and improve data security (Figure 1.3).

Did you Know?

According to a survey conducted by Forbes, over 90% of organizations rely on data-driven insights to make strategic business decisions, highlighting the critical role that databases play in decision making.

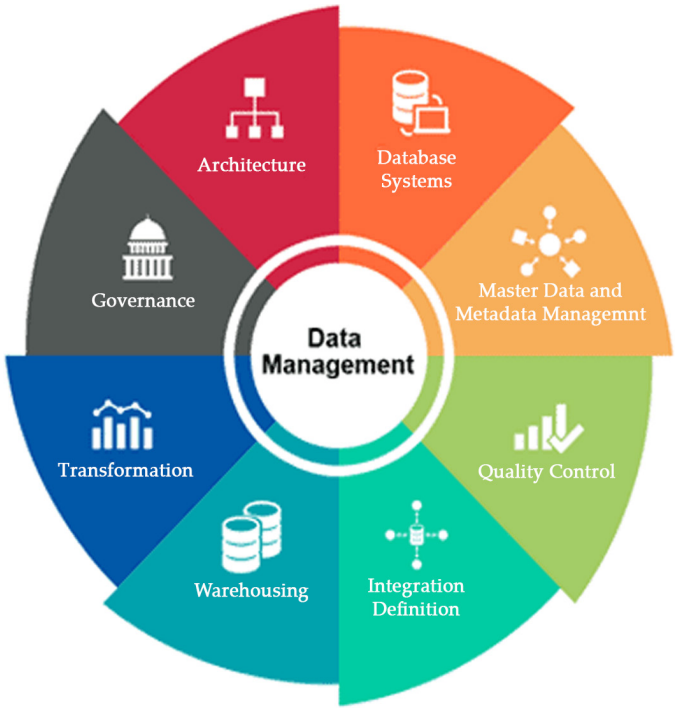


Figure 1.3. Schematic of the data management.

Source: Virtual Staffing, creative commons license.

1.3.4. Cost Savings

By centralizing data storage and management, database systems can reduce hardware and software costs, as well as administrative costs associated with maintaining and updating data (Wang & Strong, 1996).

1.3.5. Scalability

Database systems can be designed to scale up or down based on the needs of an organization, allowing them to handle large volumes of data without sacrificing performance.

1.3.6. Regulatory Compliance

Many organizations are subject to regulatory requirements that mandate the secure storage and management of sensitive data (DeFazio et al., 2001). Database systems provide the necessary features and controls to ensure compliance with these regulations.

1.3.7. Improved Customer Experience

Database systems can be used to store and manage customer data, allowing organizations to personalize their interactions with customers and provide a better overall customer experience (Sawyer & Mariani, 1995) (Figure 1.4).

Figure 1.4. Illustration of the importance of customer database.

Source: MBA Skool, creative commons license.

In conclusion, the importance of database systems in modern organizations cannot be overstated. By providing a centralized and efficient way to store, manage, and access data, database systems help organizations to make informed decisions, improve data quality, and reduce costs.

1.4. COMPONENTS OF DATABASE SYSTEMS

Database systems are complex software systems that include various components that work together to manage and store large amounts of data. The main components of a database system include (Figure 1.5):

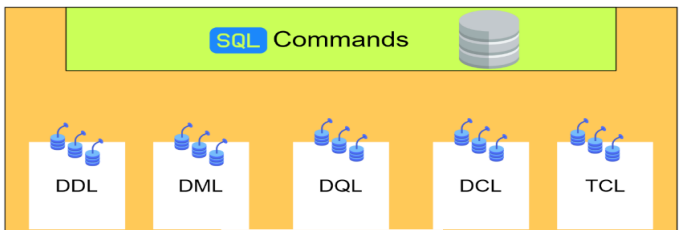


Figure 1.5. Components of the database components.

Source: Testing docs, creative commons license.

1.4.1. Data Definition Language (DDL)

Data Definition Language (DDL) is a set of SQL commands used to create and modify the structure of database objects, such as tables, indexes, and views. DDL is one of the core components of database management systems (DBMS) and is used to define and manage the database schema (Wells, 2001) (Figure 1.6).

Some common DDL statements include:

1. **CREATE:** This statement is used to create a new database object, such as a table or index. The syntax of the CREATE statement varies depending on the type of object being created.
2. **ALTER:** This statement is used to modify the structure of an existing database object, such as adding a new column to a table or modifying an index (Anwar, 2018).
3. **DROP:** This statement is used to delete an existing database object, such as a table or view. The DROP statement is a destructive operation and should be used with caution.
4. **TRUNCATE:** This statement is used to remove all the data from a table while keeping the table structure intact. The TRUNCATE statement is faster than using the DELETE statement to remove data (Wu, 2018).
5. **RENAME:** This statement is used to rename an existing database object, such as a table or column name. The RENAME statement is often used to improve the readability and maintainability of database objects.

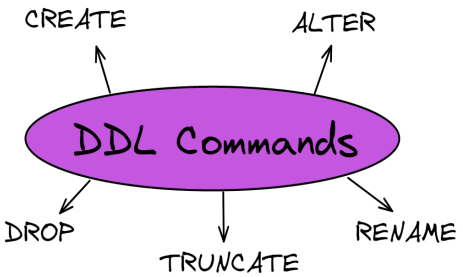


Figure 1.6. Schematic of the DDL commands.

Source: Algo daily, creative commons license.

It is important to note that DDL statements are not transactional, which means that once a DDL statement is executed, it cannot be rolled back like a transaction. Therefore, it is important to carefully review and test DDL statements before executing them to ensure that they do not cause any unintended consequences.

1.4.2. Data Manipulation Language (DML)

Data Manipulation Language (DML) is a set of SQL commands used to retrieve, insert, update, and delete data in a database. DML is an important component of DBMSs and is used to manage the data stored in the database (Ghali & Abu-Naser, 2019) (Figure 1.7).

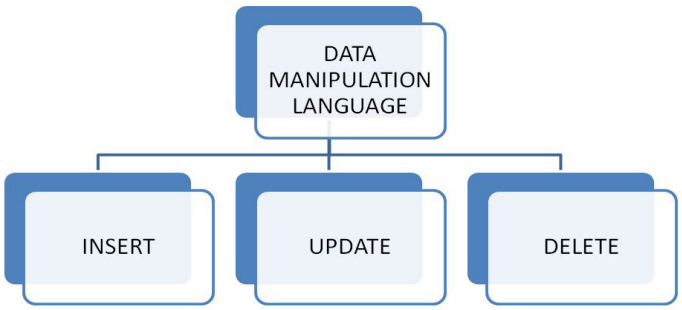


Figure 1.7. Illustration of the data manipulation language (DML).

Source: Anushree Goswami, creative commons license.

Some common DML statements include:

1. **INSERT:** This statement is used to insert new data into a table. The INSERT statement specifies the values to be inserted into each column of the table (Shaw et al., 2016).
2. **UPDATE:** This statement is used to modify existing data in a table. The UPDATE statement specifies the new values for each column that is being modified.
3. **DELETE:** This statement is used to delete data from a table. The DELETE statement specifies the rows to be deleted from the table (Litwin & Abdellatif, 1987).

It is important to note that DML statements are transactional, which means that they can be rolled back if an error occurs during execution. This allows for data consistency and helps ensure that the database remains in a valid state. ML is an important component of DBMSs that is used to manipulate the data stored in the database.

DML statements are used to retrieve, insert, update, and delete data from tables, and are executed by applications or users who need to manage the data in the database. DML statements are transactional and can be rolled back if an error occurs during execution.

1.4.3. Data Query Language (DQL)

Data Query Language (DQL) is a subset of SQL used to retrieve data from a database (Brodsky et al., 2009). DQL statements are designed to extract information from one or more database tables based on specific criteria, and are used extensively in DBMSs to retrieve data required by users or applications. There are several DQL statements used to retrieve data from a database, including:

KEYWORD

An **application developer** is a software engineer who designs, creates, tests, programs and updates applications for a particular device like mobile or web or a specific operating system.

1. **SELECT:** This is the most commonly used DQL statement. It allows the user to retrieve specific data from one or more tables in the database based on certain criteria. The SELECT statement can also be used to calculate values, group data, and join multiple tables.
2. **FROM:** This statement specifies the name of the table or tables from which data is to be retrieved.
3. **WHERE:** This statement specifies the conditions that must be met for the data to be retrieved. It allows users to filter data based on specific criteria, such as a particular date range or value.
4. **ORDER BY:** This statement is used to sort the retrieved data in a specific order, such as ascending or descending order (Jäkel et al., 2014).
5. **GROUP BY:** This statement is used to group data based on a specific column in the table. It is often used with aggregate functions such as SUM, AVG, and COUNT.
6. **HAVING:** This statement is used in conjunction with GROUP BY to filter the results of the group by statement based on a specific condition (Fikes et al., 2002).

In summary, DQL is a subset of SQL used to retrieve data from a database. DQL statements are used to extract information from one or more tables based on specific criteria, and are executed by the DBMS to extract the requested data. Common DQL statements include SELECT, FROM, WHERE, ORDER BY, GROUP BY, and HAVING. DQL is an essential component of DBMS and is used by application developers and database administrators (DBAs) to retrieve data for a variety of purposes.

1.4.4. Data Control Language (DCL)

Data Control Language (DCL) is a subset of SQL used to manage the security and access control of a database (Rao et al., 2019).

It allows DBAs to grant or revoke access to the database for users and to control the actions that users can perform on the database (Figure 1.8).

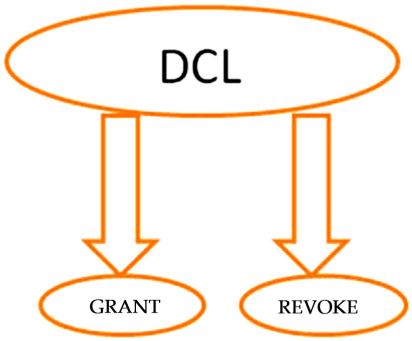


Figure 1.8. Schematic of the DCL statements.

Source: Developers tutorials, creative commons license.

There are several DCL statements used to control access to a database, including:

1. **GRANT:** This statement is used to give specific privileges to a user, such as the ability to read, write or execute database objects like tables, views, or procedures (Wyatt, 1994).
2. **REVOKE:** This statement is used to remove the previously granted privileges from the user.
3. **DENY:** This statement is used to explicitly deny a user access to a specific database object or system resource (El-Mehalawi & Miller, 2003).

DBAs use DCL statements to create security policies and enforce them in the database system. They can specify which users or groups of users have access to the database and which actions they are allowed to perform. This helps to ensure that data is only accessed and manipulated by authorized personnel. DCL is a subset of SQL used to manage the security and access control of a database. DCL statements such as GRANT, REVOKE, and DENY are used to grant or revoke access to specific database objects or system resources. DCL statements help to enforce security policies in a database system and protect data from unauthorized access (Yu & Zhou, 2019). DBAs use DCL statements to control access to the database and to ensure that data is only accessed and manipulated by authorized personnel.

Remember

Unauthorized access is when someone, internally or externally, gains access to a computer system, network, or data without permission.

1.4.5. Database Management System (DBMS)

A Database Management System (DBMS) is software that manages the storage, retrieval, and updating of data in a computer system. It is a collection of programs that enable users to create and maintain a database, and to access and manipulate data stored in the database. A DBMS provides an interface for users to interact with the database, and it acts as an intermediary between the user and the database (Rawat & Purnama, 2021). Users can use commands or queries to retrieve and manipulate data stored in the database. The DBMS is responsible for managing the storage of data, ensuring data integrity, and controlling access to the database (Figure 1.9).

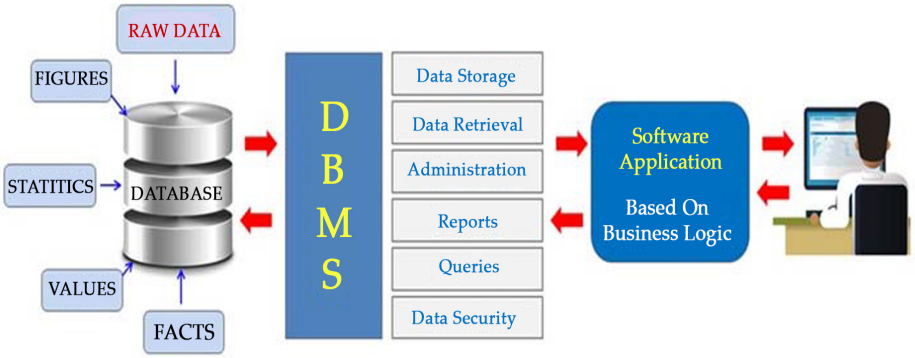


Figure 1.9. Illustration of the DBMS.

Source: Learn computer science, creative commons license.

There are several types of DBMS, including:

1. **Relational DBMS:** A type of DBMS that stores data in tables, with each table consisting of rows and columns. Relational DBMSs use SQL to access and manipulate data (Dittrich et al., 1995).
2. **Object-oriented DBMS:** A type of DBMS that stores data in objects, which can include data and the methods used to manipulate the data.
3. **Hierarchical DBMS:** A type of DBMS that organizes data in a hierarchical structure, with each record having a parent record and zero or more child records.
4. **Network DBMS:** A type of DBMS that stores data in a network structure, with each record having multiple parent and child records (Olle, 2003).

A DBMS is essential for managing large amounts of data in an organized and efficient manner. It allows multiple users to access

and manipulate the same data simultaneously, while maintaining data integrity and security. A DBMS also enables users to retrieve and manipulate data easily, using a variety of tools and interfaces. DBMS is software that manages the storage, retrieval, and updating of data in a computer system. It provides an interface for users to interact with the database, and it is responsible for managing the storage of data, ensuring data integrity, and controlling access to the database (Anjard, 1994). There are several types of DBMS, including relational, object-oriented, hierarchical, and network DBMS. A DBMS is essential for managing large amounts of data in an organized and efficient manner.

Remember

A DBMS is a software system designed to manage and control access to a database, ensuring data integrity, security, and efficient data retrieval and manipulation.

1.5. DATA STORAGE AND RETRIEVAL

Data storage and retrieval are two essential aspects of DBMSs. A DBMS must efficiently store and retrieve data to support the application's requirements (Levine, 1985). We will discuss the various techniques and methods used for data storage and retrieval in DBMS.

1.5.1. Primary Storage

Primary storage, also known as main memory or RAM, is the memory used to store the currently executing application and its data. Primary storage is volatile, meaning its contents are lost when the power is turned off. DBMS uses primary storage to cache frequently accessed data and improve query response time (Chessa & Maestrini, 2003).

1.5.2. Secondary Storage

Secondary storage is the non-volatile memory used to store data persistently. Secondary storage devices include hard disk drives, solid-state drives, and magnetic tapes. Secondary storage is used for long-term storage of the database, transaction logs, and backups.

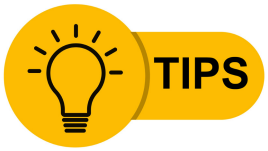
1.5.3. Indexing

Indexing is a technique used to improve the performance of data retrieval operations (Xue et al., 2019). It involves creating an index, which is a data structure that contains the values of one or more columns of a table, along with a pointer to the location of

the corresponding record in the table. When a query is executed, the index is searched instead of the table, resulting in faster data retrieval.

1.5.4. Query Optimization

Query optimization is the process of selecting the most efficient query execution plan to retrieve data from the database (Heanue et al., 1994). The query optimizer analyzes the query and the database's schema to generate an execution plan that minimizes the amount of disk I/O and CPU processing required to retrieve the data.



Ensure data reliability and accessibility by storing primary data in multiple locations with data backups.

1.5.5. Query Execution

Query execution is the process of retrieving data from the database based on the execution plan generated by the query optimizer. The DBMS retrieves data from the storage devices, applies any necessary sorting or aggregation operations, and returns the results to the application (Barrett, & Edgar, 2006). Query execution performance is critical for applications that require fast and efficient data retrieval.

In conclusion, data storage and retrieval are crucial components of a DBMS. Efficient data storage and retrieval techniques are essential for supporting fast and reliable application performance. Primary and secondary storage, indexing, query optimization, and query execution are some of the critical techniques used for efficient data storage and retrieval in a DBMS.

1.6. CHARACTERISTICS OF DATABASE SYSTEMS

A database system must possess certain characteristics to be considered efficient and reliable. In this section, we will discuss some of the key characteristics of a database system.

1.6.1. Data Independence

One of the most important characteristics of a database system is data independence. It means that the application programs should be independent of the physical storage structure of the data

(Naughton, 1985). It allows changes to be made to the database structure without affecting the application programs. There are two types of data independence: logical and physical.

1.6.2. Concurrent Access

Database systems must be designed to support multiple users accessing the database at the same time. This is known as concurrent access. The DBMS must ensure that each user sees a consistent view of the database, even if other users are modifying the data at the same time (Bernstein & Goodman, 1981). This is achieved through concurrency control mechanisms that allow multiple transactions to execute concurrently while ensuring the consistency of the database.

1.6.3. Data Integrity

Data integrity is the assurance of the accuracy and consistency of the data stored in the database. The DBMS must ensure that data is stored correctly and is consistent with other data in the database. The database system must enforce the integrity constraints that are specified in the database schema to ensure the data is accurate and consistent (Yesin et al., 2021) (Figure 1.10).

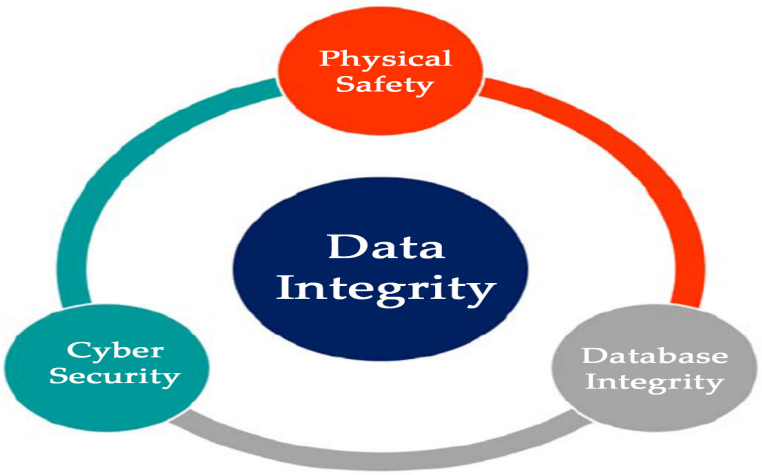


Figure 1.10. Data Integrity and its components.

Source: CFI Team, creative commons license.

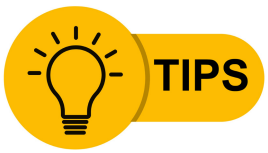
1.6.4. Security

Database systems must provide security mechanisms to protect the database from unauthorized access and ensure the confidentiality,

integrity, and availability of the data (Bertino & Sandhu, 2005). This includes authentication, authorization, and encryption mechanisms to protect the data from unauthorized access.

1.6.5. Scalability

Database systems must be designed to scale up or down to meet the changing demands of the application (Kuhlenkamp et al., 2014). The system must be able to handle increasing amounts of data and users without compromising performance or availability. This can be achieved through techniques such as partitioning, replication, and clustering. The characteristics of a database system include data independence, concurrent access, data integrity, security, and scalability. These characteristics ensure that the database system is efficient, reliable, and able to handle the demands of modern applications.



Concurrent access to a hardware component, such as a memory chip or memory bank, means that the circuits are built with two or more input or signaling channels.

1.7. ARCHITECTURE OF DATABASE SYSTEMS

Database systems can have different architectures, depending on the specific requirements of the application. The architecture of a database system determines how the different components of the system are organized, how they interact with each other, and how the data is processed and stored. There are three main types of database architectures:

1.7.1. Client-Server Architecture

The client-server architecture is a common architecture for database systems. In this architecture, the system is divided into two parts: the client and the server (McLeod & Heimbigner, 1980). The client is the user interface that interacts with the user, while the server is responsible for storing and managing the data. The client sends requests to the server, which processes the request and sends back the results.

1.7.2. Tiered Architecture

The tiered architecture is a multi-layered architecture that separates the presentation layer, application logic layer, and data storage layer. In this architecture, the application is divided into three tiers: the user interface, the business logic layer, and the data storage

layer (Hurson et al., 1989). The user interface communicates with the business logic layer, which in turn communicates with the data storage layer. This architecture provides scalability and flexibility to the system.

1.7.3. Distributed Architecture

The distributed architecture is a distributed database system that is spread across multiple locations or sites. In this architecture, the data is distributed across multiple servers, and each server is responsible for a specific set of data. The servers are connected through a network, and they work together to process user requests. This architecture provides scalability and fault tolerance to the system (Lam & Kuo, 2000) (Figure 1.11).

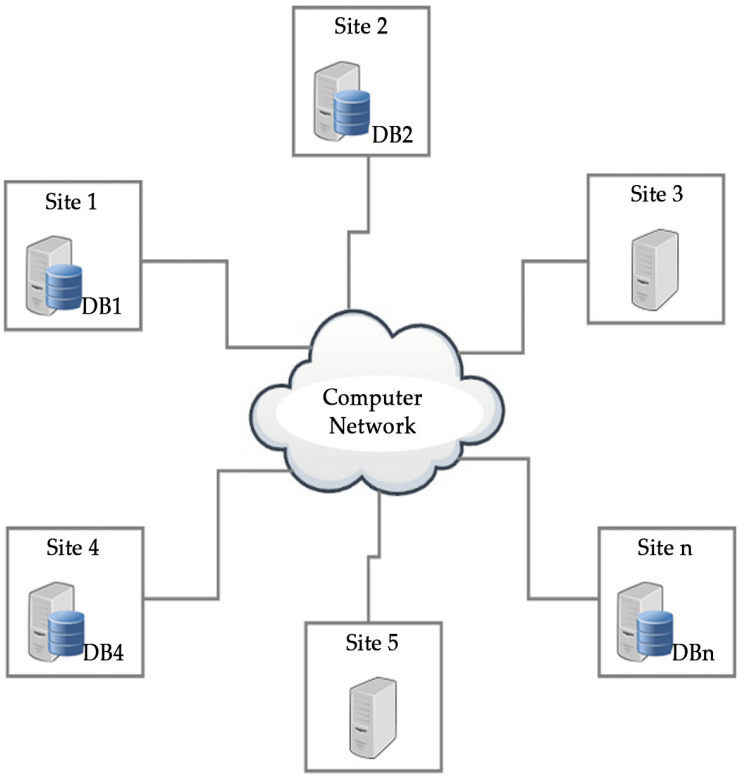


Figure 1.11. Illustration of the distributed architecture.

Source: Katembo Kituta Ezéchiel et al. creative commons license.

Each of these architectures has its own advantages and disadvantages, and the choice of architecture depends on the specific requirements of the application.

1.8. QUERY PROCESSING AND OPTIMIZATION

Query processing and optimization are essential components of database systems that ensure efficient and fast retrieval of data. Query processing refers to the translation of a user's query into an efficient form that can be executed by the system. Query optimization, on the other hand, involves selecting the most efficient execution plan for a given query.

1.8.1. Query Processing Phases

Remember

A database query is either an action query or a select query. A select query is one that retrieves data from a database. An action query asks for additional operations on data, such as insertion, updating, deleting or other forms of data manipulation.

The query processing phases are as follows:

i. **Parsing:** This is the first phase of query processing, where the system checks the syntax of the query and creates a parse tree (Park et al., 2013).

ii. **Optimization:** In this phase, the system generates different execution plans for the query and selects the most efficient one based on the query cost model.

iii. **Code generation:** Once the optimized execution plan is selected, the system generates the code to execute the query (Antoshenkov & Ziauddin, 1996).

1.8.2. Query Optimization Techniques

There are several query optimization techniques used in database systems, including:

i. **Cost-Based Optimization:** This technique uses a cost model to estimate the execution time and selects the plan with the lowest estimated cost (Alom et al., 2009).

ii. **Rule-Based Optimization:** This technique uses a set of predefined rules to generate execution plans.

iii. **Dynamic Programming:** This technique solves the query optimization problem by breaking it down into smaller sub-problems (Ouzzani & Bouguettaya, 2004).

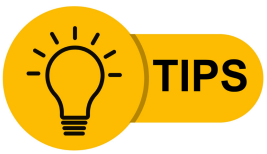
1.8.3. Query Execution Plans

The execution plan describes how the system will execute the query. It includes information on how the system will access the data, the order in which the operations will be performed, and how the results will be returned. The query execution plan is

generated during the query optimization phase and is critical to the performance of the system. The system will execute the query according to the plan, and any inefficiencies in the plan can lead to slow performance and decreased efficiency.

In conclusion, query processing and optimization are critical components of a database system. They ensure efficient and fast retrieval of data, and the performance of the system depends on their proper implementation. A well-designed query processing and optimization system can make a significant difference in the overall performance and efficiency of a database system.

1.9. IMPORTANCE OF DATABASE SYSTEMS IN MODERN WORLD



It is important to make sure that the information has not been changed or lost.

Database systems play a crucial role in the modern world where data has become an essential part of businesses, organizations, and individuals. Here are some of the key reasons why database systems are important in the modern world:

1.9.1. Data Management

Database systems help organizations manage large volumes of data efficiently (Chaudhri et al., 2003). It provides a structured way to store, manage, and retrieve data, ensuring data accuracy and consistency.

1.9.2. Decision Making

Database systems provide a platform to analyze and process data, which can help in making informed decisions (Wang et al., 2016). For instance, businesses can analyze sales data to determine which products are selling well and which ones need improvement.

1.9.3. Improved Efficiency

With a database system, data can be accessed and processed faster, reducing the time required to perform routine tasks. This improved efficiency can help organizations to be more productive and profitable.

1.9.4. Cost Savings

Database systems can help businesses save money by reducing the need for physical storage space, paper records, and manual data entry. It can also reduce the likelihood of errors and inconsistencies, which can be costly to correct.

1.9.5. Data Security

Database systems can help organizations secure sensitive data and prevent unauthorized access. By implementing user authentication and access control mechanisms, businesses can protect data from internal and external threats (Malik & Patel, 2016).

1.9.6. Compliance

Many organizations are required to comply with regulatory requirements such as GDPR, HIPAA, and PCI DSS. Database systems can help ensure compliance by providing features such as auditing, logging, and encryption (Wang, 2022).

1.9.7. Innovation

Database systems provide a foundation for innovation in areas such as machine learning, artificial intelligence, and the Internet of Things (Bradley et al., 2019). By leveraging data stored in a database, businesses can gain insights into new opportunities, products, and services.

Overall, the importance of database systems in the modern world cannot be overstated. They provide a structured way to manage and analyze data, enabling businesses and organizations to make informed decisions and stay competitive in a data-driven world.

ACTIVITY 1.1.

You are tasked with explaining the basic components of a database system to a group of stakeholders. Describe the various components, such as the data model, database schema, query language, and database management system, and explain how they work together to store, manage, and retrieve data.

SUMMARY

The chapter on Introduction to Database Systems provides an overview of the fundamentals of database systems. It starts by defining what database systems are and their importance in the modern world. The chapter then delves into the history of database systems, highlighting the major milestones that have led to the development of modern database systems. The chapter also discusses the different components of database systems, including DDL, DML, DQL, DCL, and DBMS. The discussion on each component provides a detailed explanation of what it does and its role in the database system.

The chapter also covers data storage and retrieval, with a focus on primary storage, secondary storage, indexing, query optimization, and query execution. Additionally, the chapter highlights the key characteristics of database systems, such as data independence, concurrent access, data integrity, security, and scalability. The architecture of database systems is also discussed in detail, with an emphasis on the client-server architecture, tiered architecture, and distributed architecture. Finally, the chapter provides an overview of query processing and optimization, covering query processing phases, query optimization techniques, and query execution plans.

MULTIPLE CHOICE QUESTIONS

1. Which of the following is not a component of database systems?

a. Data Definition Language
b. Data Control Language
c. Data Access Language
d. Database Management System

2. Which of the following is a characteristic of database systems?

a. Scalability
b. Flexibility
c. Reliability
d. Speed

3. Which of the following is a type of database architecture?

a. Centralized architecture
b. Decentralized architecture
c. Distributed architecture
d. All of the above

4. Which of the following is used to define the structure of a database?

a. Data Definition Language
b. Data Manipulation Language

c. Data Query Language
d. Data Control Language

5. Which type of storage is volatile in nature?
a. Primary storage
b. Secondary storage
c. Both A and B
d. None of the above

6. Which of the following is not a phase of query processing?
a. Parsing
b. Optimization
c. Execution
d. Storage

7. Which type of architecture allows for scalability in database systems?
a. Client-Server Architecture
b. Tiered Architecture
c. Distributed Architecture
d. All of the above

8. Which of the following is not a characteristic of database systems?
a. Scalability
b. Reliability
c. Compatibility
d. Security

Answers to Multiple Choice Questions

1. (a); 2. (d); 3. (b); 4. (c); 5. (a); 6 (b); 7. (c); 8. (d)

REVIEW QUESTIONS

1. Define database systems and explain their importance.
2. Describe the components of database systems and their roles.
3. Explain the different types of data storage in database systems.
4. Discuss the characteristics of database systems and their significance.
5. Compare and contrast the different types of database architecture.
6. Describe the phases of query processing and explain their importance.

7. Explain the techniques used for query optimization.
8. Discuss the role of query execution plans in database systems.

REFERENCES

1. Abadi, D., Boncz, P., Harizopoulos, S., Idreos, S., & Madden, S., (2013). The design and implementation of modern column-oriented database systems. *Foundations and Trends® in Databases, 5*(3), 197–280.
2. Alom, B. M., Henskens, F., & Hannaford, M., (2009). Query processing and optimization in distributed database systems. *IJCSNS, 9*(9), 143.
3. Anjard, R. P., (1994). The basics of database management systems (DBMS). *Industrial Management & Data Systems, 94*(5), 11–15.
4. Antoshenkov, G., & Ziauddin, M., (1996). Query processing and optimization in oracle Rdb. *The VLDB Journal, 5*, 229–237.
5. Anwar, I., (2018). *Penerjemahan Teks Bahasa Indonesia Menjadi Data Definition Language (Ddl) Dengan Penanganan Kalimat Majemuk* (2nd edn., pp. 4–10). Doctoral dissertation, Universitas Komputer Indonesia.
6. Azeroual, O., Saake, G., & Schallehn, E., (2018). Analyzing data quality issues in research information systems via data profiling. *International Journal of Information Management, 41*, 50–56.
7. Barrett, T., & Edgar, R., (2006). Gene expression omnibus: Microarray data storage, submission, retrieval, and analysis. *Methods in Enzymology, 411*, 352–369.
8. Bernstein, P. A., & Goodman, N., (1981). Concurrency control in distributed database systems. *ACM Computing Surveys (CSUR), 13*(2), 185–221.
9. Bertino, E., & Sandhu, R., (2005). Database security-concepts, approaches, and challenges. *IEEE Transactions on Dependable and Secure Computing, 2*(1), 2–19.
10. Bradley, D., Merrifield, M., Miller, K. M., Lomonico, S., Wilson, J. R., & Gleason, M. G., (2019). Opportunities to improve fisheries management through innovative technology and advanced data systems. *Fish and Fisheries, 20*(3), 564–583.
11. Brodsky, A., Bhot, M. M., Chandrashekar, M., Egge, N. E., & Wang, X. S., (2009). A decisions query language (DQL) high-level abstraction for mathematical programming over databases. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data* (Vol. 1, pp. 1059–1062).
12. Cai, L., & Zhu, Y., (2015). The challenges of data quality and data quality assessment in the big data era. *Data Science Journal, 14*(1), 4–10
13. Chaudhri, A. B., Rashid, A., & Zicari, R., (2003). *XML Data Management: Native XML and XML-Enabled Database Systems* (Vol. 4, No. 1, pp. 2–9). Addison-Wesley Professional.
14. Chessa, S., & Maestrini, P., (2003). Dependable and secure data storage and retrieval in mobile, wireless networks. In: *DSN* (Vol. 2003, pp. 207–216).

15. Connolly, T. M., & Begg, C. E., (2005). *Database Systems: A Practical Approach to Design, Implementation, and Management* (2nd edn., pp. 1–15). Pearson Education.
16. Coronel, C., & Morris, S., (2016). *Database Systems: Design, Implementation, & Management* (Vol. 3, No. 2, pp. 2–8). Cengage Learning.
17. DeFazio, S., Krishnan, R., Srinivasan, J., & Zeldin, S., (2001). The importance of extensible database systems for e-commerce. In: *Proceedings 17th International Conference on Data Engineering* (Vol. 1, pp. 63–70). IEEE.
18. DeWitt, D., & Gray, J., (1992). Parallel database systems: The future of high performance database systems. *Communications of the ACM, 35*(6), 85–98.
19. Dittrich, K. R., Gatziu, S., & Geppert, A., (1995). The active database management system manifesto: A rule base of ADBMS features. In: *Rules in Database Systems: Second International Workshop, RIDS'95 Glyfada, Athens, Greece, September 25–27, 1995 Proceedings 2* (Vol. 1, pp. 1–17). Springer Berlin Heidelberg.
20. Elmasri, R., & Navathe, S. B., (2006). *Database Systems: Models, Languages, Design, and Application Programming* (Vol. 4, No. 1, pp. 7–9). Pearson Education India.
21. El-Mehalawi, M., & Miller, R. A., (2003). A database system of mechanical components based on geometric and topological similarity. Part I: Representation. *Computer-Aided Design, 35*(1), 83–94.
22. Fikes, R., Hayes, P., & Horrocks, I., (2002). *DQL-a Query Language for the Semantic Web* (2nd edn., pp. 4–9). Knowledge Systems Laboratory.
23. Ghali, M. J. A., & Abu-Naser, S. S., (2019). *It's for Data Manipulation Language (DML) Commands Using Sqlite* (3rd edn., pp. 6–9).
24. Grad, B., & Bergin, T. J., (2009). Guest editors' introduction: History of database management systems. *IEEE Annals of the History of Computing, 31*(4), 3–5.
25. Heanue, J. F., Bashaw, M. C., & Hesselink, L., (1994). Volume holographic storage and retrieval of digital data. *Science, 265*(5173), 749–752.
26. Hellerstein, J. M., & Stonebraker, M., (2005). *Readings in Database Systems* (Vol. 1, pp. 2–5). MIT press.
27. Hurson, A. R., Miller, L. L., Pakzad, S. H., Eich, M. H., & Shirazi, B., (1989). Parallel architectures for database systems. In: *Advances in Computers* (Vol. 28, pp. 107–151). Elsevier.
28. Jäkel, T., Kühn, T., Voigt, H., & Lehner, W., (2014). RSQL-a query language for dynamic data types. In: *Proceedings of the 18th International Database Engineering & Applications Symposium* (Vol. 1, pp. 185–194).
29. Jukic, N., Vrbsky, S., Nestorov, S., & Sharma, A., (2014). *Database Systems: Introduction to Databases and Data Warehouses* (Vol. 1, p. 400). Pearson.
30. Kuhlenkamp, J., Klems, M., & Röss, O., (2014). Benchmarking scalability and elasticity of distributed database systems. *Proceedings of the VLDB Endowment, 7*(12), 1219–1230.

31. Lam, K. Y., & Kuo, T. W., (2000). *Real-Time Database Systems: Architecture and Techniques* (2nd edn., Vol. 593, pp. 4–10). Springer Science & Business Media.
32. Levine, H. G., (1985). Principles of data storage and retrieval for use in qualitative evaluations. *Educational Evaluation and Policy Analysis, 7*(2), 169–186.
33. Litwin, W., & Abdellatif, A., (1987). An overview of the multi-database manipulation language MDSL. *Proceedings of the IEEE, 75*(5), 621–632.
34. Malik, M., & Patel, T., (2016). Database security-attacks and control methods. *International Journal of Information, 6*(1, 2), 175–183.
35. Maryanski, F., Bedell, J., Hoelscher, S., Hong, S., McDonald, L., Peckham, J., & Stock, D., (1986). The data model compiler: A tool for generating object-oriented database systems. In: *Proceedings on the 1986 International Workshop on Object-Oriented Database Systems* (Vol. 1, pp. 73–84).
36. McLeod, D., & Heimbigner, D., (1980). A federated architecture for database systems. In: *Proceedings of the May 19–22, 1980, National Computer Conference* (Vol. 1, pp. 283–289).
37. Miklau, G., Levine, B. N., & Stahlberg, P., (2007). Securing history: Privacy and accountability in database systems. In: *CIDR* (2nd edn., pp. 387–396).
38. Naughton, J., (1985). Data independent recursion in deductive databases. In: *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (2nd edn., pp. 267–279).
39. Olle, T. W., (2003). Database management system (DBMS). In: *Encyclopedia of Computer Science* (Vol. 1, pp. 517–520).
40. Ouzzani, M., & Bouguettaya, A., (2004). Query processing and optimization on the web. *Distributed and Parallel Databases, 15*(1), 187–218.
41. Park, H., Pang, R., Parameswaran, A., Garcia-Molina, H., Polyzotis, N., & Widom, J., (2013). An overview of the deco system: Data model and query language; query processing and optimization. *ACM SIGMOD Record, 41*(4), 22–27.
42. Rao, T. R., Mitra, P., Bhatt, R., & Goswami, A., (2019). The big data system, components, tools, and technologies: A survey. *Knowledge and Information Systems, 60*(1), 1165–1245.
43. Rawat, B., & Purnama, S., (2021). MySQL database management system (DBMS) on FTP site LAPAN Bandung. *International Journal of Cyber and IT Service Management, 1*(2), 173–179.
44. Sawyer, P., & Mariani, J. A., (1995). Database systems: Challenges and opportunities for graphical HCI. *Interacting with Computers, 7*(3), 273–303.
45. Shaw, S., Vermeulen, A. F., Gupta, A., Kjerrumgaard, D., Shaw, S., Vermeulen, A. F., & Kjerrumgaard, D., (2016). Data manipulation language (DML). *Practical Hive: A Guide to Hadoop's Data Warehouse System, 1,* 77–98.

46. Sheth, A. P., & Larson, J. A., (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys (CSUR), 22*(3), 183–236.

47. Sokolinsky, L. B., (2004). Survey of architectures of parallel database systems. *Programming and Computer Software, 30,* 337–346.

48. Ulusoy, Ö., (1995). Research issues in real-time database systems: Survey paper. *Information Sciences, 87*(1–3), 123–151.

49. Wang, H., Xu, Z., Fujita, H., & Liu, S., (2016). Towards felicitous decision making: An overview on challenges and trends of big data. *Information Sciences, 367*, 747–765.

50. Wang, L., (2022). Providing compliance in critical computing systems. In: *System Dependability and Analytics: Approaching System Dependability from Data, System and Analytics Perspectives* (Vol. 1, pp. 191–206). Cham: Springer International Publishing.

51. Wang, R. Y., & Strong, D. M., (1996). Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems, 12*(4), 5–33.

52. Wells, G., (2001). Data definition language. In: *Code Centric: T-SQL Programming with Stored Procedures and Triggers* (Vol. 1, pp. 35–70). Berkeley, CA: Apress.

53. Widom, J., & Ceri, S., (1995). *Active Database Systems: Triggers and Rules for Advanced Database Processing* (Vol. 2, No. 1, pp. 5–10). Morgan Kaufmann.

54. Wu, M., (2018). *Property Graph Type System and Data Definition Language* (2nd edn., pp. 5–9). arXiv preprint arXiv:1810.08755.

55. Wyatt, J. C., (1994). Clinical data systems, part 2: Components and techniques. *The Lancet, 344*(8937), 1609–1614.

56. Xue, J., Xu, C., & Bai, L., (2019). DStore: A distributed system for outsourced data storage and retrieval. *Future Generation Computer Systems, 99*(1), 106–114.

57. Yesin, V., Karpinski, M., Yesina, M., Vilihura, V., & Warwas, K., (2021). Ensuring data integrity in databases with the universal basis of relations. *Applied Sciences, 11*(18), 8781.

58. Yeung, A. K., & Hall, G. B., (2007). *Spatial Database Systems: Design, Implementation and Project Management* (Vol. 87, pp. 4–8). Springer Science & Business Media.

59. Yu, J. H., & Zhou, Z. M., (2019). Components and development in big data system: A survey. *Journal of Electronic Science and Technology, 17*(1), 51–72.

CHAPTER 2

TYPES OF DATABASES

UNIT INTRODUCTION

Database systems are computer programs that allow users to store, organize, and retrieve large amounts of data (Güting & Schneider, 1993). There are several different types of database systems, each with its own strengths and weaknesses. Some of the most common types of database systems include:

1. **Hierarchical Databases:** These database systems organize data in a tree-like structure, with each record having one parent and multiple children. Hierarchical databases are often used in mainframe applications.
2. **Network Databases:** These database systems organize data in a more complex network-like structure, with each record having multiple parents and children. Network databases are useful for modeling complex relationships between data (Albano et al., 1989).
3. **Relational Databases:** These database systems are based on the relational data model, which organizes data into tables with rows and columns. RDBMSs are widely used for storing structured data and are popular in applications such as online transaction processing (OLTP) and business intelligence (BI) (Brodie, 1980).
4. **Object-Oriented Databases:** These database systems are designed to store

complex data types such as objects, classes, and inheritance hierarchies. OODBMSs are often used in object-oriented programming languages such as Java or C++.

5. **Graph Databases:** These database systems are designed to store and query data that has complex relationships, such as social networks or supply chain systems. Graph databases use nodes and edges to represent data and can perform complex queries quickly (Ohori, 1990).
6. **Document Databases:** These database systems store unstructured or semi-structured data in documents, such as JSON or XML. Document databases are often used in web applications and content management systems (CMS).
7. **NoSQL Databases:** These database systems are designed to handle large volumes of unstructured or semi-structured data, such as text, images, or social media posts. NoSQL database systems are often used in big data applications or real-time analytics (Schneider, 1997).

Each type of database system has its own unique features and benefits, and choosing the right system for a particular application requires careful consideration of factors such as data structure, scalability, performance, and cost.

Learning Objectives

- Define database systems and identify the different types of database systems, including hierarchical, network, relational, object-oriented, graph, document, and NoSQL databases.
- Understand the strengths and weaknesses of each type of database system and their suitability for different applications.
- Explain the relational data model and the basics of RDBMSs.
- Identify the benefits and limitations of using object-oriented databases in software development.
- Describe the unique features and benefits of graph databases and their use cases.
- Understand the purpose of document databases and their applications in web development and content management systems.
- Discuss the advantages and limitations of NoSQL databases in handling unstructured or semi-structured data and their use in big data applications and real-time analytics.
- Analyze the factors to consider when choosing the appropriate database system for a particular application, such as data structure, scalability, performance, and cost.

Key Terms

- Big Data
- Data

- Database
- Document
- E-Commerce
- Graph
- Hierarchical
- Management
- Model
- Network
- Object-Oriented
- Relational
- Social Network

2.1. HIERARCHICAL DATABASES

Hierarchical databases are a type of database management system (DBMS) that organizes data in a tree-like structure, with each record having one parent and multiple children (Tsichritzis & Lochovsky, 1976).). This structure resembles the organizational hierarchy of a company, hence the name "hierarchical." We will discuss the history, importance, applications, advantages, and disadvantages of hierarchical databases (Figure 2.1).

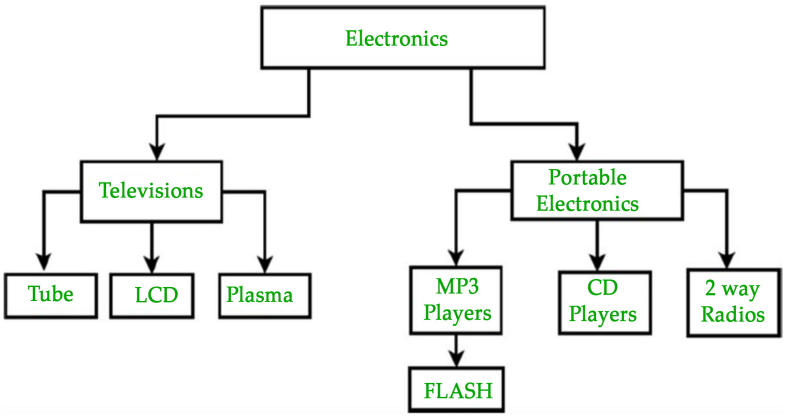


Figure 2.1. An example of the hierarchical database.

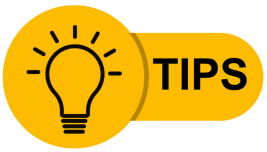
Source: Geeks for Geeks, creative commons license.

2.1.1. History of Hierarchical Databases

The hierarchical database model was developed in the 1960s and was one of the earliest forms of DBMSs. At that time, computers were large and expensive, and the data they processed was mostly structured and hierarchical in nature. The hierarchical database model was well-suited for this type of data and computing environment. The first hierarchical DBMS was IBM's Information Management System (IMS), which was introduced in the mid-1960s. IMS was widely used in large enterprises, particularly in the banking and insurance industries, where it was used to manage vast amounts of customer data (Deniša & Ude, 2015). IMS was based on a hierarchical data model, where data was organized in a tree-like structure with a single root node at the top and multiple child nodes branching out from it. Each node represented a data element or record, and the relationships between nodes were defined by parent-child relationships (Bouganim et al., 1996). This structure made it easy to represent data that had a fixed and predictable structure. The hierarchical database model was popular in the 1960s and 1970s, but it began to decline in popularity with

the advent of the relational database model in the 1970s. Relational databases offered more flexibility and scalability than hierarchical databases, making them better suited for complex and changing data environments (Deng et al., 2009).

Despite the decline in popularity, hierarchical databases are still used in some applications today, particularly in mainframe systems and other environments where the data is well-defined and predictable. They continue to be used in industries such as banking, insurance, and manufacturing, where there is a need to store large amounts of structured data. The hierarchical database model has a long and rich history, and it played an important role in the early days of DBMSs. While it has been largely replaced by more flexible and scalable database models, it continues to be used in certain applications today and remains an important part of the evolution of database technology.



Storage plays a crucial role for Database Management Systems particularly helps in storing and retrieving information stored.

2.1.2. Importance of Hierarchical Databases

Hierarchical databases were one of the earliest forms of DBMSs and played an important role in the development of modern database technology (Jindal & Bali, 2012). While they have been largely replaced by more flexible and scalable database models, they continue to be used in some applications today.

One of the key advantages of hierarchical databases is that they are well-suited for managing large amounts of structured data that has a predictable and fixed structure. This makes them ideal for use in applications such as banking, insurance, and manufacturing, where there is a need to store and manage vast amounts of data with a consistent and hierarchical structure.

Another advantage of hierarchical databases is that they are highly efficient and performant. Since data is organized in a tree-like structure with a single root node at the top and multiple child nodes branching out from it, it is easy to navigate and access data quickly. This makes hierarchical databases well-suited for applications where speed and performance are critical. Hierarchical databases also offer a high degree of data security and control (Shokoufandeh et al., 2005). Because the data is organized in a fixed and predictable structure, it is easier to control who has access to different parts of the database. This makes hierarchical databases ideal for applications where data security and access control are critical, such as in financial institutions.

Despite these advantages, hierarchical databases also have some limitations. They are not well-suited for managing complex or unstructured data, and they can be difficult to modify or update when the underlying data structure changes (Nicklin et al., 1985). This is why they have been largely replaced by more flexible and scalable database models, such as relational databases. Hierarchical databases played an important role in the development of modern database technology and continue to be used in some applications today. While they have some limitations, their strengths in managing structured data efficiently and securely make them well-suited for certain applications.

Remember

While structured data only accounts for around 20 percent of data world-wide, it is the current foundation of big data. This is because it is so easy to access, use, and the outcomes of using it are far more accurate.

2.1.3. Applications of Hierarchical Databases

Hierarchical databases have been widely used in a variety of applications, especially in early database systems. Some of the common applications of hierarchical databases include:

2.1.3.1. Banking and Financial Systems

Hierarchical databases have been widely used in banking and financial systems to manage customer data, transactions, and account information (Hsu & Madnick, 1983). Since hierarchical databases are highly efficient and performant, they can easily handle the large volume of structured data that is typically found in financial systems.

2.1.3.2. Inventory Management Systems

Hierarchical databases have been used in inventory management systems to manage product data, stock levels, and order information (Vargo et al., 1992). The predictable and fixed structure of hierarchical databases makes it easy to manage and track inventory data.

2.1.3.3. Manufacturing and Production Systems

Hierarchical databases have also been used in manufacturing and production systems to manage product data, bill of materials, and production schedules (Kearns & DeFazio, 1983). The hierarchical structure makes it easy to manage and track data related to different components and sub-assemblies in the production process.

2.1.3.4. Telecommunications Systems

Hierarchical databases have been used in telecommunications systems to manage customer data, billing information, and call records. The efficient and performant nature of hierarchical databases makes it easy to handle the large volume of structured data that is typically found in telecommunications systems (Figure 2.2).

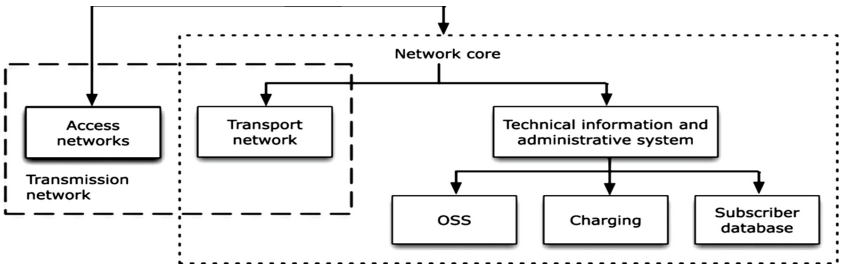


Figure 2.2. Hierarchy of networks and systems in a generic telecom infrastructure.

Source: Kjell Jørgen Hole, creative commons license.

2.1.3.5. Government Systems

Hierarchical databases have been used in various government systems to manage citizen data, tax records, and other administrative data (Banerjee et al., 1980). The predictable and fixed structure of hierarchical databases makes it easy to manage and track data related to different government departments and agencies.

Hierarchical databases have been used in a wide range of applications where structured data is the predominant data type. While they have been largely replaced by more flexible and scalable database models such as relational databases, they still remain a viable option for managing structured data in applications that require a fixed and predictable data structure.

2.1.4. Advantages of Hierarchical Databases

Hierarchical databases have several advantages that make them well-suited for managing large amounts of structured data with a predictable and fixed structure. Some of the key advantages of hierarchical databases include:

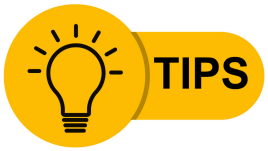
2.1.4.1. Efficient and Performant

Hierarchical databases are highly efficient and performant because data is organized in a tree-like structure with a single root node at

the top and multiple child nodes branching out from it (Tangorra & Chiarolla, 1995). This makes it easy to navigate and access data quickly, which is important in applications where speed and performance are critical.

2.1.4.2. Data Security and Control

Because the data in hierarchical databases is organized in a fixed and predictable structure, it is easier to control who has access to different parts of the database (Black et al., 2004). Hence, hierarchical databases is suitable for security and control of data.



Retrieving hierarchical data from a hierarchical database architecture requires traversing the entire tree, starting at the root node

2.1.4.3. Simple and Easy to Use

Hierarchical databases are simple and easy to use because they have a predictable and fixed structure. This makes it easy to understand and navigate the database, even for non-technical users (Ho & Akyildiz, 1997).

2.1.4.4. Low Maintenance

Because hierarchical databases have a fixed and predictable structure, they require less maintenance than other types of databases. This can be particularly important in applications where downtime or maintenance windows are limited.

2.1.4.5. Well-Established Technology

Hierarchical databases have been around for a long time and are a well-established technology. This means that there is a large community of developers and users who are familiar with them and can provide support and expertise (Abiteboul & Hull, 1986). Hierarchical database also has some disadvantages as well. They are not well-suited for managing complex or unstructured data, and they can be difficult to modify or update when the underlying data structure changes. Hence, flexible and scalable database models are required. However, for applications that require a fixed and predictable data structure, hierarchical databases can still be an effective and efficient solution.

2.1.5. Disadvantages of Hierarchical Databases

While hierarchical databases have several advantages in managing

structured data, they also have some limitations and disadvantages. Some of the key disadvantages of hierarchical databases include:

2.1.5.1. Limited Flexibility

Hierarchical databases have a rigid structure, which makes it difficult to accommodate changes to the data structure or schema. Adding new data fields or restructuring data requires modifying the entire database structure, which can be time-consuming and expensive (Ipeirotis et al., 2002).

Remember

Characteristics of hierarchical database models include their simplicity, but also their lack of flexibility.

2.1.5.2. Limited Query Capabilities

Hierarchical databases do not have the same level of query capabilities as other types of databases, such as relational databases. This can make it difficult to extract specific subsets of data or perform complex queries.

2.1.5.3. Limited Scalability

Hierarchical databases are not as scalable as other types of databases. As the database grows in size, it becomes more difficult to maintain and manage (Silberschatz & Kedem, 1980). This can result in slower performance and increased maintenance costs.

2.1.5.4. Limited Data Sharing

Hierarchical databases do not support the same level of data sharing capabilities as other types of databases. This can make it difficult to integrate data from multiple sources or share data with other systems.

2.1.5.5. Limited Data Modeling

Hierarchical databases do not support complex data modeling or relationships between data entities (Domdouzis et al., 2021). This can make it difficult to model data in a way that accurately reflects the real-world relationships between data entities. In conclusion, while hierarchical databases have advantages in managing structured data with a predictable and fixed structure, they also have limitations in terms of flexibility, scalability, and query capabilities. These limitations make them less suitable for managing complex or unstructured data, or for applications that require a high level of flexibility and scalability.

2.2. NETWORK DATABASES

A network database is a type of database model that is designed to manage complex data structures and relationships between data entities (Robinson, 2013). It is similar to the hierarchical database model, but it allows for more flexible relationships between data entities. In a network database, data is organized into records and sets, with each set containing one or more records. The relationships between sets are defined by a network structure, which allows for many-to-many relationships between data entities (Figure 2.3).

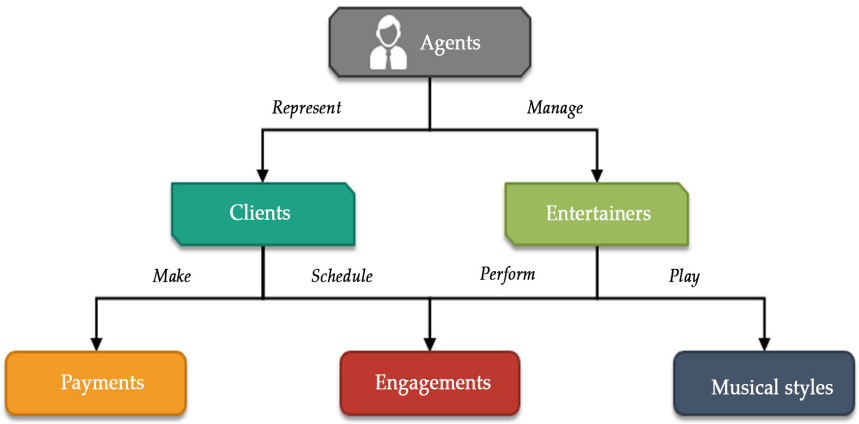


Figure 2.3. An example of the network database model.

Source: Sketch Bubble, creative commons license.

In a network database model, data is organized into a series of interconnected records, with each record containing one or more data fields. The records are organized into sets, with each set representing a different type of entity or object (Dunham & Helal, 1995). For example, in a database for a university, there might be sets for students, professors, courses, and departments. Each set would contain records with data fields for specific attributes or properties of that entity.

The network structure in a network database is defined by a set of pointers that connect related records together. Each record has one or more pointers that define its relationships to other records in the database. This allows for many-to-many relationships between data entities, which is not possible in the hierarchical database model (Lai & Hsia, 2007). For example, in a university database, a student record might be connected to multiple course records, and each course record might be connected to multiple student records. One of the main advantages of network databases is their ability to manage complex data relationships. They are well-suited to applications where there are many-to-many relationships

between data entities, or where the data structure is subject to frequent changes. Network databases are also highly efficient in terms of data retrieval, since they use pointers to quickly navigate between related records (Figure 2.4).

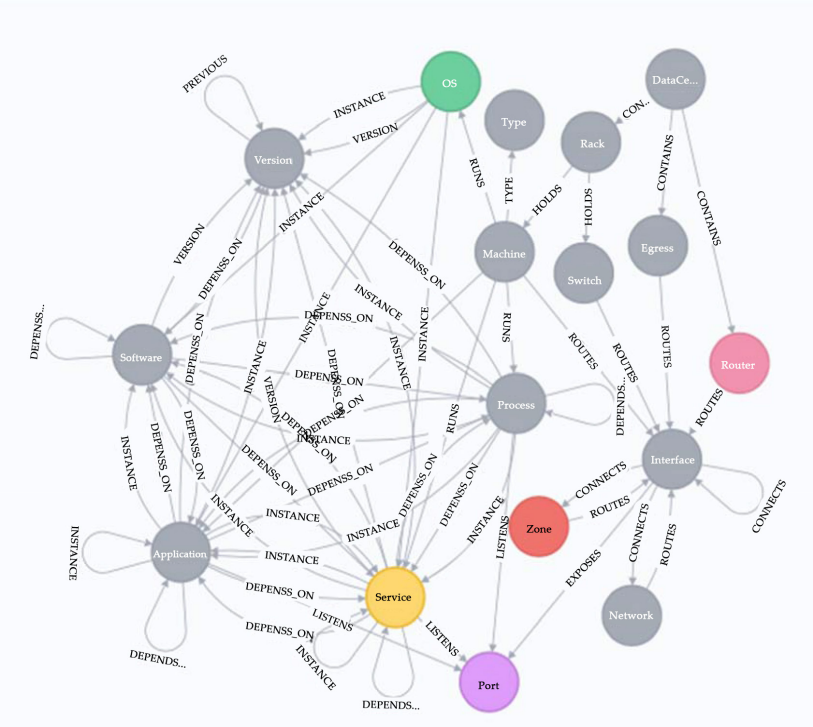


Figure 2.4. An example of the complex data relationship.

Source: Thomas Frisendal, creative commons license.

However, network databases also have some disadvantages. They can be complex and difficult to design and maintain, and they are not as widely used as other database models such as the relational database model. Additionally, they are not as scalable as other database models, since the network structure can become unwieldy as the database grows in size.

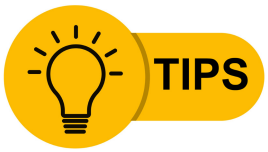
Overall, network databases are a powerful tool for managing complex data relationships, but they are best suited to applications where their advantages outweigh their disadvantages.

2.2.1. History of Network Databases

The network database model was first introduced in the late 1960s as an improvement over the hierarchical database model, which had limitations in terms of flexibility and scalability. The network model was developed by Charles Bachman, who was

working at General Electric at the time. Bachman was looking for a way to manage complex data relationships in a more flexible and efficient way than the hierarchical model allowed (Murty et al., 2011).

The first network DBMS was developed by the Integrated Data Store (IDS) project at the University of Michigan in the early 1970s. This system, called the Michigan Terminal System Database Management System (MTSDBMS), was used to manage data for a variety of research projects at the university (Alonso & Korth, 1993). It was later commercialized as Integrated Database Management System (IDMS) by Cullinane Corporation.



The program must still access the intervening segments, even though they are not needed.

Other companies, including IBM and Univac, also developed network database systems in the 1970s. IBM's System R project, which began in the mid-1970s, was the first to implement a relational database model, which eventually became the dominant database model. However, the network model continued to be used in certain specialized applications, such as airline reservation systems and financial transaction systems.

In the 1980s and 1990s, the popularity of network databases declined as the relational database model became the dominant database model. The relational model offered greater flexibility and scalability than the network model, and it was also easier to use and maintain (Wu & Chang, 1991). However, network databases continued to be used in certain specialized applications where the hierarchical structure was better suited to the data.

Today, network databases are still used in some legacy systems, but they have largely been replaced by relational databases and other database models that offer greater flexibility, scalability, and ease of use. However, the network model remains an important development in the history of database systems, as it paved the way for later innovations in data management and storage.

2.2.2. Importance of Network Databases

The network database model has played an important role in the development of database systems, particularly in applications that require the management of complex data relationships. Some of the key importance of network databases in database systems include:

2.2.2.1. Flexibility

The network database model offers greater flexibility than the hierarchical database model, which allows for more complex data relationships (Pham & Klamma, 2010). This makes it a good choice for applications where data entities have many-to-many relationships.

2.2.2.2. Efficiency

The network database model is highly efficient in terms of data retrieval, since it uses pointers to quickly navigate between related records. This makes it a good choice for applications that require fast access to large amounts of data.

Remember

A DBMS provides mechanisms for backing up and recovering the data in the event of a system failure.

2.2.2.3. Specialized Applications

Network databases are still used in some specialized applications, such as airline reservation systems and financial transaction systems, where the hierarchical structure is better suited to the data.

2.2.2.4. Historical Significance

The network database model played an important role in the development of database systems, and paved the way for later innovations in data management and storage (Dy-Liacco, 1994). It represented a significant improvement over the hierarchical model and paved the way for the development of the relational model. However, network databases also have some limitations and challenges.

Overall, network databases remain an important part of the history and development of database systems, and they continue to be used in some specialized applications where their advantages outweigh their limitations.

2.2.3. Applications of Network Databases

Network databases have been used in a variety of applications over the years, particularly in those that require the management of complex data relationships (Chung et al., 2013). Here are some of the key applications of network databases in database systems:

2.2.3.1. Financial Systems

Network databases have been used extensively in financial systems, where they are used to manage complex data relationships between financial transactions, accounts, and customers (Gavish & Pirkul, 1986). These systems require fast and efficient data retrieval, which the network database model is well-suited to (Figure 2.5).

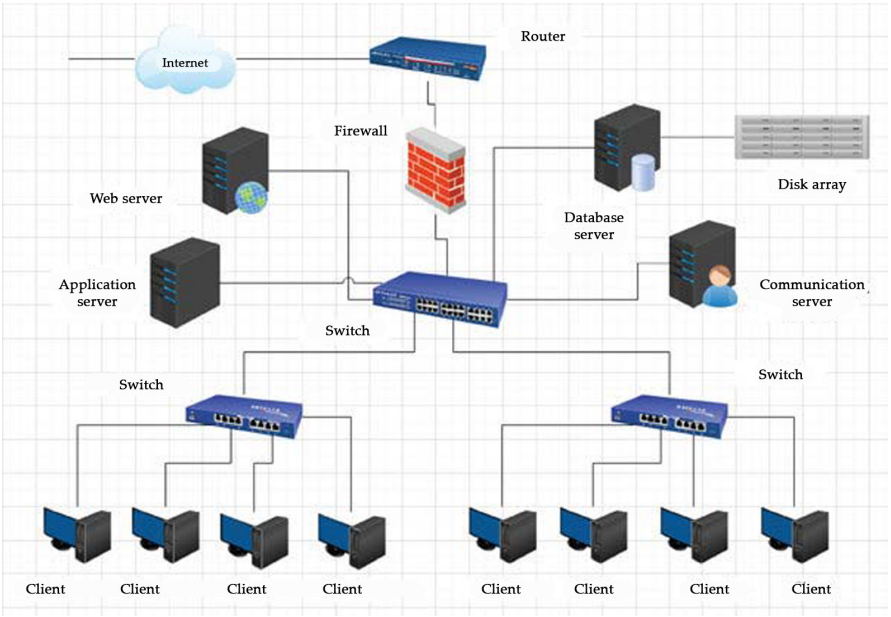


Figure 2.5. Network database in financial system.

Source: Jianwei Yan, creative commons license.

2.2.3.2. Inventory Systems

Inventory management systems often require the management of complex data relationships between inventory items, suppliers, and customers (Hesse et al., 1993). The network database model is well-suited to these applications, since it allows for efficient navigation between related records.

2.2.3.3. Airline Reservation Systems

The airline industry has used network databases for many years to manage flight schedules, reservations, and passenger data. These systems require fast and efficient data retrieval, as well as the ability to manage complex data relationships between flights, passengers, and destinations (Chou et al., 2008).

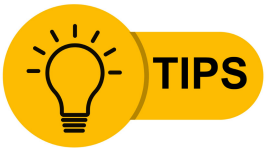
2.2.3.4. Telecommunications

Network databases have been used extensively in telecommunications systems, where they are used to manage complex data relationships between customers, phone numbers, and billing information. Network database is ideal for quick and efficient retrieval of data.

2.2.3.5. Manufacturing Systems

Manufacturing systems often require the management of complex data relationships between production processes, materials, and finished products (Nakada et al., 1999).

Overall, the network database model has been used in a variety of applications where complex data relationships need to be managed. While it has been largely replaced by the relational database model, it remains an important part of the history and development of database systems, and continues to be used in some specialized applications where its advantages outweigh its limitations.



However, the relational database model has started to win over both the network and the hierarchical models because its added flexibility and productivity has become more evident as hardware technology has become faster.

2.2.4. Advantages of Network Databases

The network database model offers several advantages over other database models, such as the hierarchical model, that make it well-suited to certain types of applications (Larson, 1983). Here are some of the key advantages of network databases in database systems:

2.2.4.1. Flexibility

One of the main advantages of the network database model is its flexibility. Unlike the hierarchical model, which can only represent one-to-many relationships, the network model can represent many-to-many relationships between data entities. This makes it a good choice for applications where data entities have complex relationships.

2.2.4.2. Efficient Data Retrieval

The network database model uses pointers to quickly navigate between related records (Sato et al., 1997). This makes it highly efficient in terms of data retrieval, and makes it a good choice for applications that require fast access to large amounts of data.

2.2.4.3. Scalability

The network database model is highly scalable, since it allows for the addition of new data entities without having to modify the entire database structure. This makes it well-suited to applications that require frequent updates and modifications.

2.2.4.4. Redundancy

The network database model allows for the redundancy of data, which can improve data integrity and system reliability (Bader et al., 2003). In the event that one data entity is lost or damaged, there may be other copies of the same entity elsewhere in the database.

KEYWORD

Data integrity is the overall accuracy, completeness, and consistency of data.

2.2.4.5. Data Independence

The network database model allows for data independence, which means that changes to the structure of the database do not necessarily require changes to the application programs that use the database (Kolahdouzan, & Shahabi, 2004). This can make it easier to update and modify the database over time.

Overall, the network database model offers several advantages over other database models, particularly in applications that require the management of complex data relationships. While it may not be as widely used as other database models such as the relational model, it remains an important part of the history and development of database systems, and continues to be used in some specialized applications where its advantages outweigh its limitations.

2.2.5. Disadvantages of Network Databases

While the network database model offers several advantages over other database models, it also has some limitations and drawbacks that can make it less suitable for certain types of applications. Here are some of the key disadvantages of network databases in database systems:

2.2.5.1. Complexity

The network database model can be more complex than other database models, such as the hierarchical model, due to the many-to-many relationships between data entities (Padmanabhan

et al., 2008). This complexity can make it more difficult to design and maintain a network database, particularly for applications with a large number of data entities.

2.2.5.2. Lack of Standards

The network database model lacks a standardized language for accessing and manipulating data, which can make it more difficult to work with compared to other database models that have standardized query languages, such as SQL.

2.2.5.3. Performance Issues

While the network database model is efficient in terms of data retrieval, it can suffer from performance issues when dealing with large amounts of data. This is because navigating the complex relationships between data entities can be computationally intensive and time-consuming (de Almeida & Güting, 2005).

KEYWORD

A data entity is an abstraction from the physical implementation of database tables.

2.2.5.4. Limited Support

The network database model is not as widely supported as other database models, such as the relational model. This can make it more difficult to find developers and database administrators (DBAs) with experience working with network databases.

2.2.5.5. Data Redundancy

While data redundancy can be an advantage in some cases, it can also be a disadvantage in terms of storage space and data management. Network databases can require significant amounts of storage space due to the duplication of data, and managing and updating redundant data can be complex and time-consuming (Papadias et al., 2003).

Overall, the network database model also has some limitations and drawbacks that can make it less suitable for certain types of applications. It is important to carefully consider the requirements of a particular application before choosing a database model.

2.3. RELATIONAL DATABASES

A relational database is a type of database model that stores data in tables that are related to each other through key fields

(Teorey et al., 1986). The relational database model is based on the relational algebra developed by Edgar F. Codd in the 1970s, and has become one of the most widely used database models in the world. In this model, data is organized into tables, with each table representing a specific type of data entity (e.g., customers, orders, and products). Each row in a table represents a single instance of that data entity, and each column represents a specific attribute of that entity (Outrata & Vychodil, 2012) (Figure 2.6).

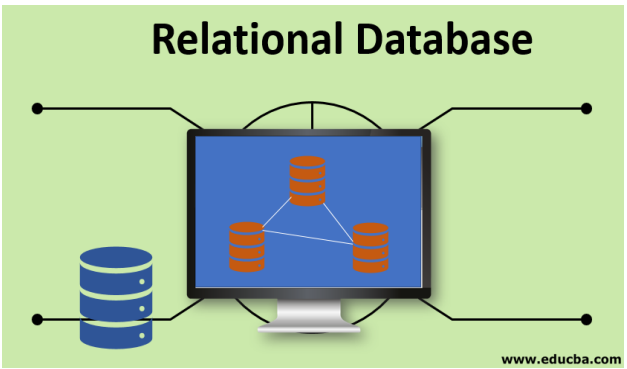


Figure 2.6. Schematic of a relational database.

Source: Educba, creative commons license.

For example, a customer table might have columns for customer ID, name, address, and phone number, with each row representing a different customer. An orders table might have columns for order ID, customer ID, order date, and order total, with each row representing a different order.

The relationships between tables in a relational database are defined through key fields. A key field is a field that uniquely identifies each row in a table. For example, in the customer and orders tables described above, the customer ID field would be a key field in the customer table, and the customer ID field in the orders table would be a foreign key that links each order to a specific customer in the customer table. Relational databases are known for their ability to handle complex relationships between data entities, as well as their ability to support complex queries and transactions. They are widely used in a variety of applications, from small business websites to large-scale enterprise systems.

2.3.1. History of Relational Databases

The history of relational databases dates back to the 1970s, when Edgar F. Codd, a researcher at IBM, introduced the concept of the relational database model in a paper titled “A Relational Model of

Data for Large Shared Data Banks" (Moszer et al., 1995). Codd's paper proposed a new way of organizing and storing data that differed from the hierarchical and network database models that were popular at the time.

The relational database model was based on the idea of using tables to store data, with each table representing a specific type of entity (e.g., customers, orders, products) and each row in the table representing a single instance of that entity. The relationships between tables were defined through key fields, which were used to link related data together.

Codd's paper sparked a revolution in the field of database management, and the relational database model quickly gained popularity among researchers and practitioners alike. In the following years, a number of commercial DBMSs were developed that implemented the relational model, including IBM's System R, Oracle, and Microsoft SQL Server (Bordoloi & Kalita, 2013).

Remember

The relational model in DBMS is an abstract model used to organize and manage the data stored in a database.

The widespread adoption of the relational database model was driven by several factors, including the growing complexity of data management tasks, the need for standardized data management techniques, and the desire for more flexible and powerful database systems (Meier et al., 1994).

Over the years, the relational database model has continued to evolve, with new features and enhancements being added to improve performance, scalability, and functionality. Today, the relational database model remains one of the most widely used database models in the world, and continues to be an important tool for managing complex data relationships in a wide range of applications.

2.3.2. Importance of Relational Databases

Relational databases are important in database systems for a number of reasons:

2.3.2.1. Structured Data Storage

Relational databases provide a structured way of storing data that is easy to understand and use (Roussopoulos, 1982). Tables with rows and columns make it simple to store data in a consistent and organized manner (Figure 2.7).

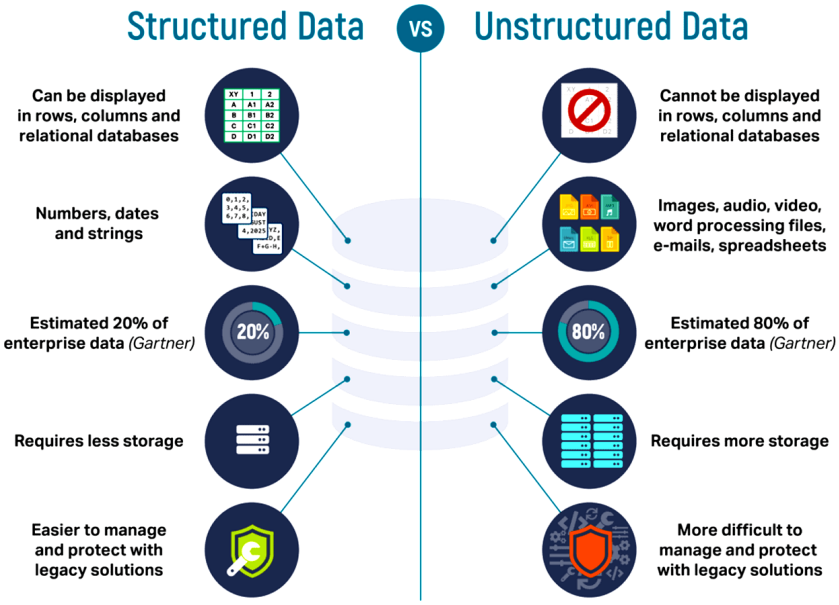


Figure 2.7. Comparison of the structured and unstructured data.

Source: Lawtomated, creative commons license.

2.3.2.2. Data Consistency

With the use of constraints and foreign key relationships, relational databases ensure that data is consistent and accurate (Blasgen & Eswaran, 1977). This is important in ensuring that data is reliable and can be used for decision-making.

2.3.2.3. Querying and Reporting

Relational databases allow for complex querying and reporting of data, making it possible to extract specific information from large datasets quickly and efficiently.

2.3.2.4. Scalability

Relational databases are highly scalable, allowing for the addition of new data and the ability to handle large datasets without a significant reduction in performance (Bhat & Jadhav, 2010).

2.3.2.5. Security

Relational databases provide robust security features that ensure data is protected from unauthorized access or modification. This is critical for organizations that store sensitive or confidential data.

2.3.2.6. Integration

Relational databases can be integrated with other systems and applications, making it easy to share data between different parts of an organization or with external partners (Stonebraker, 1986).

Overall, relational databases are important in database systems because they provide a reliable, scalable, and secure way of storing and managing data that is essential for many organizations to operate effectively.

2.3.3. Applications of Relational Databases

Relational databases have a wide range of applications in various industries and domains. Some common applications of relational databases include:

2.3.3.1. E-commerce

Online retailers use relational databases to manage customer information, orders, and inventory (Stefanidis et al., 2009). This allows them to easily track sales, manage stock levels, and provide a personalized shopping experience for customers.

KEYWORD

Inventory refers to a company's goods and products that are ready to sell, along with the raw materials that are used to produce them.

2.3.3.2. Banking and Finance

Banks and financial institutions use relational databases to store customer information, transactions, and account balances. This enables them to track and analyze financial data, detect fraudulent activity, and make informed decisions about lending and investment.

2.3.3.3. Healthcare

Healthcare providers use relational databases to store patient records, medical history, and treatment plans. This allows doctors and nurses to easily access patient information and provide personalized care.

2.3.3.4. Human Resources

Companies use relational databases to manage employee information, including personal details, job titles, and performance data (Imieliński & Lipski Jr, 1984). This helps HR departments

to make informed decisions about hiring, training, and career development.

2.3.3.5. Government

Government agencies use relational databases to manage citizen data, track public services, and analyze trends in social and economic data. This enables them to make evidence-based policy decisions and provide better services to citizens.

2.3.3.6. Education

Educational institutions use relational databases to manage student records, grades, and attendance data (Ogle & Stonebraker, 1995). This allows them to track student progress, identify areas for improvement, and provide support to students.

KEYWORD

An **informed decision** is a choice that individuals make once they have all the information related to the decision topic.

Overall, relational databases have a broad range of applications in various domains and industries, and their versatility and flexibility make them an important tool for managing and analyzing data.

2.3.4. Advantages of Relational Database

Relational databases offer several advantages over other types of databases, which make them a popular choice for managing data in various applications. Some of the key advantages of relational databases are:

2.3.4.1. Data Integrity and Consistency

Relational databases use various constraints like primary keys, foreign keys, and check constraints to ensure data integrity and consistency (Levene & Loizou, 2012). This means that data is accurate, complete, and consistent, which is essential for making informed decisions.

2.3.4.2. Easy to Use

Relational databases are easy to understand and use, with a simple structure of tables and relationships. This makes it easy for users to query and retrieve data from the database, even if they are not database experts.

2.3.4.3. Flexibility

Relational databases are highly flexible, allowing users to add new tables, fields, and relationships as needed (Stolte et al., 2002). This makes it easy to adapt to changing business needs or new requirements.

2.3.4.4. Scalability

Relational databases are highly scalable and can handle large amounts of data and users without a significant loss in performance. This makes it ideal for growing businesses that need to manage large amounts of data.

2.3.4.5. Security

Relational databases offer strong security features to protect data from unauthorized access, modification, and theft (Maier, 1983). This includes access control, data encryption, and data backup and recovery.

KEYWORD

Data encryption is a way of translating data from plaintext (unencrypted) to ciphertext (encrypted).

2.3.4.6. Data Sharing

Relational databases allow for easy data sharing between different applications and users (Padmanabhan et al., 2001). This makes it easy to integrate data from multiple sources and ensures that everyone has access to the same data.

Overall, the advantages of relational databases make them a popular choice for managing data in various applications, from e-commerce to healthcare and finance. Their flexibility, scalability, and strong security features make them an ideal choice for businesses and organizations of all sizes.

2.3.5. Disadvantages of Relational Database

While relational databases offer many advantages, they also have some disadvantages that should be considered before choosing them as a solution. Some of the key disadvantages of relational databases are:

2.3.5.1. Complexity

Relational databases can be complex to design, implement, and

maintain, especially for large-scale systems. (Codd, 2007) Managing the relationships between tables and ensuring data consistency can be challenging and time-consuming.

2.3.5.2. Performance

Relational databases can become slow and inefficient when dealing with large amounts of data. Complex queries or joins can take a long time to execute, leading to slower performance and decreased productivity (Mishra & Eich, 1992).

2.3.5.3. Cost

Relational databases can be expensive, both in terms of software licensing and hardware requirements. As the amount of data grows, businesses may need to invest in more powerful servers or storage solutions to maintain performance.

Remember

Object-oriented databases are designed to store and manage complex objects and data structures, allowing for efficient and flexible data retrieval and manipulation.

2.3.5.4. Limited Scalability

While relational databases are scalable, there are limitations to how much they can scale (Atzeni & De Antonellis, 1993). As the database grows larger and more complex, it can become more challenging to maintain and scale.

2.3.5.5. Data Redundancy

In relational databases, data can be duplicated across multiple tables, which can lead to data redundancy and inconsistencies. This can make it difficult to maintain data integrity and can lead to data quality issues.

2.3.5.6. Lack of Flexibility

Relational databases have a rigid structure, with tables and relationships predefined (Jatana et al., 2012). This can make it difficult to adapt to changing business needs or to add new data types or fields.

Overall, the disadvantages of relational databases should be carefully considered before choosing them as a solution. While they offer many advantages, businesses and organizations need to weigh the costs and complexity of implementing and maintaining them against their specific needs and requirements.

2.4. OBJECT-ORIENTED DATABASES

Object-oriented databases (OODBs) are a type of DBMS that are based on the principles of object-oriented programming (OOP). OODBs store data as objects, which consist of attributes (data) and methods (procedures). Each object is associated with a class, which defines its structure and behavior (De Caluwe, 1997). OODBs are used in applications that require complex data structures and relationships, such as scientific research, multimedia, and e-commerce (Figure 2.8).

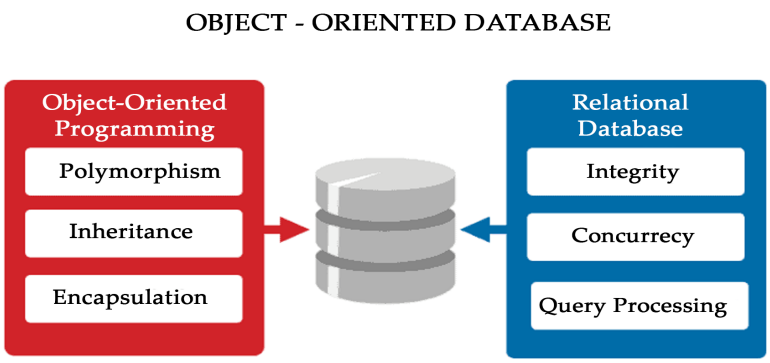


Figure 2.8. Illustration of the object-oriented database.

Source: PhoenixNAP, creative commons license.

The history of OODBs can be traced back to the 1970s and 1980s, when researchers began exploring ways to combine the benefits of OOP with database technology. The first OODBs were developed in the 1980s, and by the 1990s, they had become a popular alternative to traditional relational databases.

One of the key advantages of OODBs is that they provide a more natural and intuitive way to model complex data structures and relationships. OOP provides a powerful way to encapsulate data and behavior, which makes it easier to develop and maintain complex software applications. OODBs also support more flexible querying and indexing, which can lead to better performance and scalability.

OODBs are particularly well-suited for applications that require complex data modeling and relationships, such as scientific research, multimedia, and e-commerce (DeWitt et al., 1990). For example, an OODB could be used to store and manage multimedia content, such as images and videos, along with associated metadata and user information. OODBs are also used in scientific research, where they can be used to model complex relationships between data, such as in bioinformatics or genomics.

However, OODBs also have some disadvantages. One of the main challenges with OODBs is that they can be more difficult to implement and maintain than traditional relational databases. This is because they require more specialized knowledge of OOP principles and programming languages. Additionally, OODBs may not be as well-suited for applications that require simple data models and relationships, such as accounting or finance. Finally, OODBs may not be as widely supported or adopted as traditional relational databases, which could limit their usefulness in some contexts.

2.4.1. History of Object-Oriented Databases

The concept of object-oriented programming (OOP) emerged in the 1960s and 1970s, but it wasn't until the 1980s that researchers began exploring ways to combine the benefits of OOP with database technology. The first object-oriented databases (OODBs) were developed in the mid-1980s, and they quickly gained popularity as a powerful alternative to traditional relational databases (Wuu & Dayal, 1992).

Remember

The distinction between logical and physical also applies to database operations, which are clearly defined actions that enable applications to manipulate the data and structures of the database.

One of the earliest OODBs was the ObjectStore system, developed by Object Design Inc. in 1986. ObjectStore was based on the Smalltalk-80 programming language, which was one of the first languages to support OOP. Other early OODBs included GemStone and ONTOS, both of which were developed in the late 1980s.

By the early 1990s, OODBs had become a popular choice for applications that required complex data structures and relationships, such as scientific research, multimedia, and e-commerce (Cellary & Jomier, 1990). However, OODBs were also more complex and difficult to implement than traditional relational databases, and they required specialized knowledge of OOP principles and programming languages.

In the late 1990s and early 2000s, OODBs faced increasing competition from other types of databases, such as object-relational databases (ORDBs) and NoSQL databases. ORDBs were designed to provide some of the benefits of OODBs while still maintaining compatibility with traditional SQL databases. NoSQL databases, on the other hand, were designed to provide more flexible and scalable data storage solutions for web applications and other large-scale systems (Beeri, 1990).

Despite this competition, OODBs remain a valuable tool for applications that require complex data modeling and relationships. Today, OODBs are used in a variety of applications, including scientific research, multimedia, and e-commerce, as well as in specialized fields such as bioinformatics and genomics.

2.4.2. Importance of Object-Oriented Databases

OODBs are important in database systems because they provide a way to store and manage complex data structures that are difficult to represent using traditional relational databases. OODBs use the principles of OOP to create a more natural representation of the data, which makes it easier to work with and manipulate (Joseph et al., 1991).

Remember

Encapsulation can be used to hide both data members and data functions or methods associated with an instantiated class or object.

One of the key advantages of OODBs is that they allow developers to create complex data models that reflect the real-world relationships between objects. This makes it easier to represent complex structures such as hierarchies, networks, and graphs (Kim et al., 1990). For example, an OODB could be used to model a complex supply chain, with multiple levels of suppliers, manufacturers, and distributors, each with their own relationships and attributes.

Another advantage of OODBs is that they provide a more flexible way to query and manipulate the data. Unlike traditional relational databases, which require complex SQL queries to join tables together, OODBs allow developers to query the data using OOP principles such as inheritance, polymorphism, and encapsulation. This makes it easier to build complex applications that require sophisticated data access and manipulation.

OODBs are also important because they support the development of software systems that are more modular and reusable (Zand et al., 1995). By encapsulating the data and behavior of objects within the database, OODBs allow developers to create objects that can be easily reused across different applications and systems. This can help to reduce development time and improve the overall quality of software systems.

Finally, OODBs are important because they support the development of new applications and systems that require complex data modeling and analysis. This includes fields such as scientific research, multimedia, and e-commerce, as well as specialized fields such as bioinformatics and genomics. By providing a more

natural way to represent and manipulate complex data, OODBs enable developers to create new applications and systems that would be difficult or impossible to implement using traditional relational databases.

2.4.3. Applications of Object-Oriented Databases

OODBs are widely used in a variety of applications, particularly those that require the management and analysis of complex data structures. Some of the key applications of OODBs include:

2.4.3.1. Scientific Research

Remember

Multimedia is an engaging kind of media that offers a variety of effective ways to convey information to users.

OODBs are commonly used in scientific research, where complex data structures such as protein structures, molecular interactions, and genetic sequences must be managed and analyzed. OODBs provide a natural way to represent these data structures, making it easier to develop and maintain scientific applications.

2.4.3.2. Multimedia

OODBs are also used extensively in multimedia applications, such as image and video databases (Zdonik & Maier, 1990). These applications often require the management of large volumes of complex data, and OODBs provide a flexible and efficient way to store and access this data.

2.4.3.3. E-commerce

OODBs are increasingly used in e-commerce applications, where they are used to manage large volumes of customer data, product catalogs, and transactional data. OODBs provide a flexible and scalable platform for managing these data structures, making it easier to build and maintain e-commerce applications (Bernstein, 1998).

2.4.3.4. Finance

OODBs are also used in the finance industry, where they are used to manage complex financial instruments such as derivatives, options, and futures. These instruments have complex relationships and attributes, and OODBs provide a natural way to represent and manage this data.

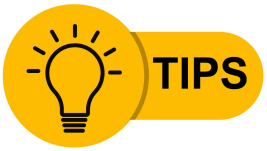
2.4.3.5. Content Management

OODBs are commonly used in content management systems (CMS), where they are used to manage large volumes of content such as articles, videos, and images. OODBs provide a flexible and scalable platform for managing this content, making it easier to build and maintain CMS (Gray et al., 1992).

2.4.3.6. Internet of Things (IoT)

OODBs are increasingly being used in IoT applications, where they are used to manage large volumes of data generated by connected devices. These data structures can be highly complex, with multiple levels of relationships and attributes, and OODBs provide a natural way to represent and manage this data (Orenstein, 1986).

In general, OODBs are used in applications that require the management and analysis of complex data structures, particularly those that have relationships and attributes that are difficult to represent using traditional relational databases.



The goal of database performance tuning is to minimize the response time of your queries by making the best use of your system resources.

2.4.4. Advantages of Object-Oriented Databases

OODBs offer a number of advantages over traditional relational databases. Some of the key advantages of OODBs include:

2.4.4.1. Support for Complex Data Structures

OODBs provide a natural way to represent and manage complex data structures, including hierarchical, network, and graph data (Hurson et al., 1993). This makes it easier to develop and maintain applications that require the management of complex data.

2.4.4.2. Improved Performance

Because OODBs are designed to work with complex data structures, they can often perform more efficiently than relational databases when dealing with these structures (Leavitt, 2000). This can result in faster query performance and improved overall system performance.

2.4.4.3. Flexibility

OODBs are highly flexible, allowing developers to easily add new data types and modify existing data structures as needed. This

makes it easier to adapt to changing business requirements and to evolve the database schema over time.

2.4.4.4. Object-Oriented Programming

OODBs are closely aligned with OOP languages, making it easier for developers to integrate database functionality into their application code (Kim et al., 1987). This can improve developer productivity and make it easier to develop and maintain complex applications (Figure 2.9).

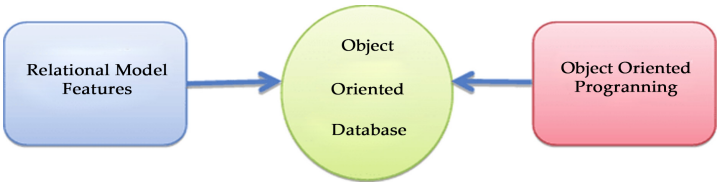


Figure 2.9. Illustration of the object-oriented database and object-oriented programming.

Source: Ebrary, creative commons license.

2.4.4.5. Support for Inheritance

OODBs support inheritance, which allows developers to reuse existing data structures and to create new data structures that inherit properties from existing ones. This can help to reduce development time and improve code quality by promoting code reuse.

2.4.4.6. Improved Data Integrity

OODBs provide better support for data integrity, including the ability to enforce referential integrity constraints and to perform complex validation checks (Wells et al., 1992). This can help to ensure that the data in the database is accurate and consistent.

Overall, OODBs offer a number of advantages over traditional relational databases, particularly for applications that require the management of complex data structures. By providing support for complex data structures, improved performance, flexibility, and better data integrity, OODBs can help developers to build more robust and scalable applications.

2.4.5. Disadvantages of Object-Oriented Databases

Despite their advantages, OODBs also have some disadvantages

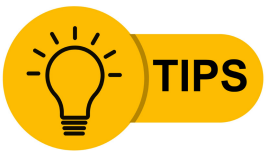
that need to be considered when deciding whether to use them for a particular application. Some of the key disadvantages of OODBs include:

2.4.5.1. Complexity

While OODBs are designed to work with complex data structures, they can also be more complex to design, develop, and maintain than traditional relational databases (Gotthard et al., 1992). This can require more specialized skills and expertise, which can be a challenge for some organizations.

2.4.5.2. Limited Tool Support

Compared to relational databases, there are fewer tools available for working with OODBs, which can make it more difficult to develop and maintain applications (Atkinson et al., 1990). This can be a particular challenge for organizations that rely heavily on third-party tools and applications.



Seamless integration of operating systems, databases, languages, spreadsheets, word processors, AI expert system shells.

2.4.5.3. Lack of Standardization

While there are standards for OOP languages, there are no widely accepted standards for OODBs. This can make it more difficult to move data between different OODBs or to integrate OODBs with other systems.

2.4.5.4. Performance Issues

While OODBs can offer improved performance for complex data structures, they can also be slower than relational databases for some types of queries (Bertino & Martino, 1991). This is because OODBs often have to traverse complex object graphs to retrieve data, which can be slower than using SQL to join tables in a relational database.

2.4.5.5. Scalability

While OODBs can be highly flexible and adaptable, they can also be more difficult to scale than relational databases. This is because the complex object graphs in an OODB can make it more difficult to partition data across multiple servers or to distribute queries efficiently.

2.4.5.6. Cost

Finally, OODBs can be more expensive to license and maintain than traditional relational databases (Kim, 1990). This is because there are fewer OODBs on the market and because they require more specialized skills and expertise to work with.

Overall, OODBs are not always the best choice for every application. While they offer some significant advantages over traditional relational databases, they also have some significant drawbacks that need to be considered. Organizations should carefully weigh the pros and cons of OODBs when making decisions about database management.

2.5. GRAPH DATABASES

A graph database is a type of NoSQL database that uses graph theory to store, map, and query relationships between data elements (Zhang, 2017). In a graph database, data is represented as nodes, edges, and properties, which can be used to model complex relationships between entities (Figure 2.10).

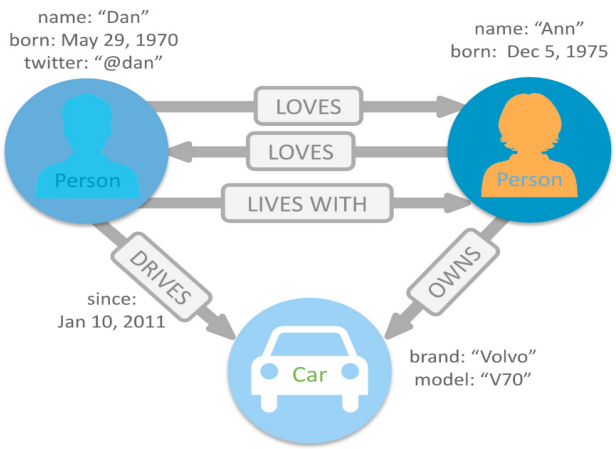
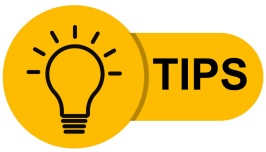


Figure 2.10. Illustration of a graph database with an example.

Source: Neo4j, Inc. creative commons license.

Nodes represent the entities in the database, while edges represent the relationships between those entities. For example, in a social network graph database, a node might represent a person, while an edge might represent a connection between two people, such as a friendship or a follow relationship. Properties provide additional information about the nodes and edges, such as a person's name or a connection's strength.

Graph databases are designed to handle complex, highly connected data sets, making them well-suited for use cases such as social networks, recommendation engines, and fraud detection (Singh & Kaur, 2015). Because graph databases can model relationships between data elements in a natural way, they can provide powerful insights into how different parts of a system are connected and how they influence one another.



Utilize graph databases for complex, interrelated data sets that require flexibility and high performance.

2.5.1. History of Graph Databases

Graph databases have their roots in graph theory, which is a branch of mathematics that studies networks of interconnected objects. In the 18th and 19th centuries, mathematicians began studying graphs as a way to model and understand complex systems, such as transportation networks and social relationships (Schindler, 2018).

The concept of graph databases as a computer science tool emerged in the 1960s and 1970s, when researchers began using graph theory to model and analyze data in computer systems. In the 1980s, graph databases became more widely used for specific applications such as geographic information systems (GIS) and computer-aided design (CAD) systems.

The first commercial graph database was released in the mid-2000s by a company called Neo Technology. Neo Technology's product, called Neo4j, was designed to provide a high-performance, scalable graph database for modern applications. Since then, other companies and open-source projects have emerged, offering a range of graph databases for different use cases and platforms (Jaiswal & Agrawal, 2013).

Today, graph databases are increasingly used in a wide range of applications, including social networks, recommendation engines, fraud detection, and bioinformatics. As more organizations recognize the value of modeling and analyzing complex relationships between data elements, the use of graph databases is likely to continue to grow in popularity.

2.5.2. Importance of Graph Databases

Graph databases are important for several reasons:

2.5.2.1. Relationship Modeling

Graph databases are designed to handle complex relationships between data elements (Ranu & Singh, 2009). This makes them well-suited for modeling and analyzing data that has many interconnected components. By using a graph database, organizations can better understand the relationships between different data points, which can lead to insights and better decision-making (Figure 2.11).

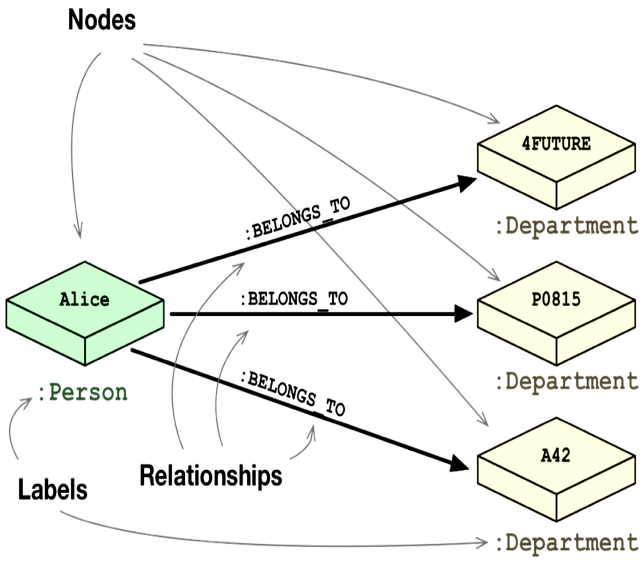


Figure 2.11. Relationship modeling in graph database.

Source: Neo4j, Inc. creative commons license.

2.5.2.2. Flexibility

Graph databases are highly flexible and can be used for a wide range of applications (Lee et al., 2012). They can handle unstructured data as well as structured data, which means they can be used for a wide range of use cases. This makes them a versatile tool for data management.

2.5.2.3. Performance

Graph databases are optimized for handling complex relationships and can deliver high performance for applications that require real-time query processing. This makes them well-suited for use cases such as fraud detection, recommendation engines, and social networks, where fast query processing is essential.

2.5.2.4. Scalability

Graph databases are designed to scale horizontally, which means that they can handle large amounts of data and can grow as data needs increase (Mendelzon & Wood, 1995). This makes them a good choice for organizations that need to store and manage large amounts of data.

Overall, graph databases are an important tool for data management and analysis, particularly in applications that involve complex relationships between data elements. By using a graph database, organizations can gain new insights into their data, improve decision-making, and deliver better results.

Remember

Object-oriented databases are a niche offering in the relational database management system (RDBMS) field and are not as successful or well-known as mainstream database engines.

2.5.3. Applications of Graph Databases

Graph databases have numerous applications across a wide range of industries. Some of the most common applications include:

2.5.3.1. Social Networks

Social networks are built on complex relationships between users, their friends, and their interactions. Graph databases are ideal for modeling and analyzing these relationships, making them a popular choice for social networking platforms (Pokorný et al., 2017).

2.5.3.2. Recommendation Engines

Recommendation engines rely on data about user behavior and preferences to make personalized recommendations. Graph databases can be used to model these relationships and deliver fast, accurate recommendations (Jouili & Vansteenberghe, 2013).

2.5.3.3. Fraud Detection

Graph databases can be used to detect fraudulent activities by analyzing relationships between different entities (Castellana et al., 2015). For example, a bank might use a graph database to identify fraudulent transactions by looking for patterns in the relationships between account holders, merchants, and transactions (Figure 2.12).

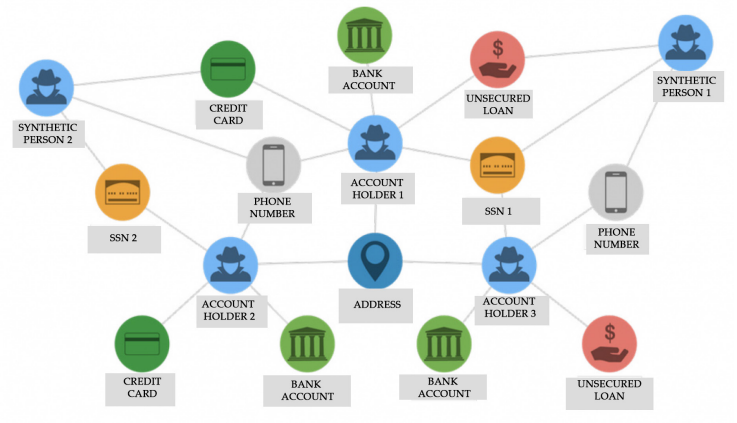


Figure 2.12. Fraud detection using graph database.

Source: Jim Webber, creative commons license.

2.5.3.4. Knowledge Management

Graph databases can be used to manage complex knowledge systems, such as those used in healthcare, scientific research, and engineering (Wood, 2012). By modeling the relationships between different data elements, graph databases can help organizations better understand complex systems and make more informed decisions.

2.5.3.5. Logistics and Supply Chain Management

Graph databases can be used to model complex supply chain networks and optimize logistics operations. By analyzing relationships between suppliers, customers, and products, organizations can improve efficiency and reduce costs.

2.5.3.6. Recommendation Engines

Graph databases can be used to model complex relationships between different products, such as books, movies, and music (Debrouvier et al., 2021). By analyzing these relationships, recommendation engines can make accurate recommendations to users based on their preferences and behavior.

Overall, graph databases have a wide range of applications across many industries. Their ability to handle complex relationships and deliver fast, accurate insights makes them a valuable tool for data management and analysis.

2.5.4. Advantages of Graph Databases

Graph databases offer several advantages over other types of databases:

2.5.4.1. Flexibility

Graph databases are highly flexible and can be used to model a wide variety of data structures, including complex relationships between entities (ShefaliPatil & Bhatia, 2014). This makes them ideal for use in applications where data is constantly changing or evolving.

2.5.4.2. Performance

Graph databases are designed to be highly performant, even when dealing with large and complex datasets. This is because graph databases use indexing and caching techniques to retrieve data quickly, and because they are optimized for querying relationships between entities.

Did you get it?

A small classic dataset from Fisher, 1936. One of the earliest datasets used for evaluation of classification methodologies.

2.5.4.3. Scalability

Graph databases are highly scalable, meaning that they can handle large amounts of data and users without sacrificing performance (Angles et al., 2017). This makes them ideal for use in applications with high growth potential.

2.5.4.4. Schema-Less Design

Graph databases do not require a fixed schema, meaning that the database can be easily adapted to changing business needs without requiring significant modifications to the underlying data model (Shimpi & Chaudhari, 2012).

2.5.4.5. Real-Time Analysis

Graph databases can be used to perform real-time analysis of data, making them ideal for use in applications where quick decision-making is required.

2.5.4.6. Natural Language Processing

Graph databases are well-suited for natural language processing

applications, where relationships between entities and concepts need to be identified and analyzed (Pokorný, 2015).

Overall, graph databases offer significant advantages over other types of databases when it comes to handling complex relationships between entities, delivering high performance, and providing flexibility and scalability.

2.5.5. Disadvantages of Graph Databases

While graph databases offer many advantages, they also have some disadvantages that should be taken into consideration:

Did you Know?

NoSQL databases offer flexibility, scalability, and high availability by utilizing non-relational data models and distributed architectures.

2.5.5.1. Complexity

Graph databases can be more complex to implement and maintain than other types of databases (Jin et al., 2010). This is because they require a deeper understanding of graph theory and specialized query languages, and may require more advanced technical expertise to manage.

2.5.5.2. Limited Support

Graph databases are a relatively new technology, and as such, there may be limited support available in terms of tools, frameworks, and third-party integrations.

2.5.5.3. Data Modeling Challenges

Graph databases require careful consideration of data modeling to properly represent relationships between entities (Kumar Kaliyar, 2015). This can be a challenge, especially when working with complex data structures or large datasets.

2.5.5.4. Cost

Some graph database solutions may be more expensive than other types of databases, which can be a barrier to adoption for smaller organizations or projects.

2.5.5.5. Performance Limitations

While graph databases are designed to be highly performant, their performance may be limited in some use cases (Batra & Tyagi,

2012). For example, when working with large or complex graphs, or when performing complex queries that require traversing multiple relationships.

Overall, while graph databases offer many advantages, they may not be the best fit for every use case. Organizations considering using graph databases should carefully evaluate the pros and cons and determine whether the benefits outweigh the potential drawbacks.

2.6. NOSQL DATABASES

NoSQL databases, as the name suggests, are databases that do not use the traditional structured query language (SQL) used in relational databases. They are designed to handle large volumes of unstructured or semi-structured data, which are increasingly becoming more common in modern applications (Liao et al., 2016). This data can include text, images, videos, and other types of unstructured data, as well as data from distributed systems, social networks, and big data analytics (Figure 2.13).

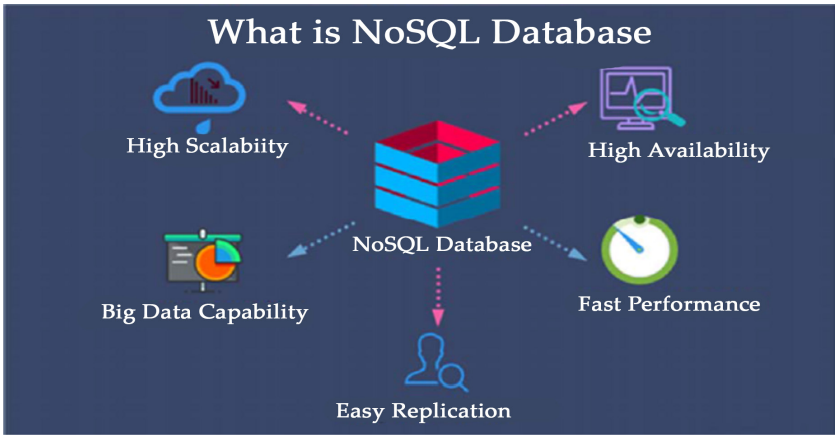


Figure 2.13. Illustration of the NoSQL database.

Source: Educba, creative commons license.

Unlike traditional relational databases, NoSQL databases use a variety of data models, including key-value stores, document-oriented databases, graph databases, and column-family stores, among others. Key-value stores, for example, are simple databases that store data as key-value pairs, making them ideal for caching and other performance optimizations. Document-oriented databases, on the other hand, store data as JSON or XML documents, making them ideal for handling complex data structures (Figure 2.14).

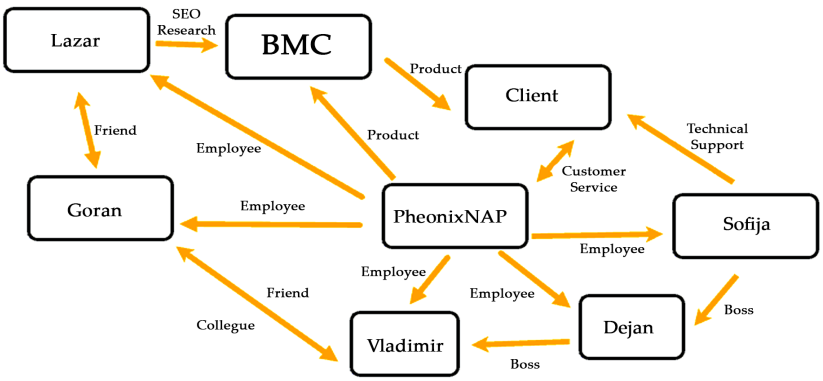


Figure 2.14. Illustration of an example of the NOSQL database.

Source: Vladimir Kaplarevic, creative commons license.

One of the key advantages of NoSQL databases is their high scalability and flexibility. They can handle large volumes of data and distribute data across multiple servers, making them ideal for applications that require high availability and horizontal scalability. NoSQL databases are also highly performant, as they can perform queries and updates quickly, without the overhead of traditional relational databases.

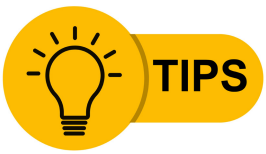
NoSQL databases are widely used in web applications, social networks, and big data analytics, as well as in other areas where large volumes of unstructured data are generated and need to be processed quickly. They are also popular in the cloud computing space, as they can be easily scaled up or down to meet changing demands. However, NoSQL databases also have their disadvantages, including a lack of standardization and a higher learning curve for developers who are used to traditional relational databases.

2.6.1. History of NoSQL Databases

NoSQL databases have been around for decades, but they only gained widespread popularity in the early 2000s. The term “NoSQL” was first used in 1998 to describe a lightweight, web-based database that didn’t use SQL. However, it wasn’t until the mid-2000s that NoSQL databases began to gain traction among large-scale web companies like Google, Amazon, and Facebook (Meier et al., 2019). These companies needed a way to handle massive amounts of data and to scale their systems horizontally, which is difficult to do with traditional relational databases.

In 2007, Amazon released Dynamo, a highly available key-value store designed for use in distributed systems. The following year, Google published a paper on Bigtable, a scalable, distributed data storage system. Both Dynamo and Bigtable are considered to be early examples of NoSQL databases. In 2009, a group of open-source developers created the first NoSQL database conference, which helped to popularize the term and the concept of NoSQL databases.

Since then, NoSQL databases have continued to evolve and have become an essential part of many modern software applications. Today, there are many different types of NoSQL databases, each with their own strengths and weaknesses. Some of the most popular types include document databases, graph databases, key-value stores, and column-family stores. NoSQL databases are now used by a wide range of organizations, from small startups to large enterprises, and are often used for big data and real-time applications.



While a variety of differences exist between relational database management systems (RDBMS) and NoSQL databases, one of the key differences is the way the data is modeled in the database.

2.6.2. Importance of NoSQL Databases

NoSQL databases have become increasingly important in modern computing due to the rise of big data, cloud computing, and web applications. The importance of NoSQL databases can be explained in the following ways:

2.6.2.1. Scalability

NoSQL databases are designed to handle large amounts of unstructured and semi-structured data, making them highly scalable (Zaki, 2014). They can scale horizontally by adding more nodes to a cluster, allowing them to handle high-traffic websites and applications with ease.

2.6.2.2. Flexibility

Unlike traditional relational databases, NoSQL databases are schema-less, which means that they do not require a predefined schema to store data. This makes it easier to store and retrieve complex data types, such as graphs, documents, and key-value pairs.

2.6.2.3. Performance

NoSQL databases are optimized for performance, making them ideal for high-traffic websites and applications (Moniruzzaman & Hossain, 2013). They use distributed architectures and caching techniques to improve query response times and minimize downtime.

2.6.2.4. Cost-Effective

NoSQL databases are typically open-source and do not require expensive licenses or hardware. This makes them a cost-effective option for businesses and organizations that need to store large amounts of data.

Remember

In order to retrieve all of the information about a user and their hobbies, information from the Users table and Hobbies table will need to be joined together.

2.6.2.5. Availability

NoSQL databases are designed to provide high availability and fault tolerance (Pokorny, 2011). They use techniques such as replication and sharding to ensure that data is always available, even in the event of hardware failure or network outages.

Overall, NoSQL databases have become increasingly important due to their ability to handle big data and scale to meet the needs of modern computing. They offer flexibility, performance, and availability, making them a popular choice for web applications, e-commerce sites, social networks, and other high-traffic websites.

2.6.3. Applications of NoSQL Databases

NoSQL databases are used in a wide range of applications and industries due to their flexibility, scalability, and performance advantages over traditional relational databases. Some of the common applications of NoSQL databases are:

2.6.3.1. Big Data

NoSQL databases are commonly used in big data applications, which involve processing and analyzing vast amounts of data (Haseeb & Pattun, (2017). These databases can handle large amounts of unstructured data, such as text, images, and videos.

2.6.3.2. E-commerce

NoSQL databases are used in e-commerce applications, where

high availability, scalability, and performance are essential. These databases can handle large volumes of transactions and provide real-time inventory and order management.

2.6.3.3. Social Networking

NoSQL databases are commonly used in social networking applications, which involve managing large volumes of user-generated content (Abramova et al., 2013). These databases can handle complex relationships between users and provide real-time updates.

Internet of Things (IoT)

NoSQL databases are used in IoT applications, where large amounts of data are generated from sensors and devices. These databases can handle high volumes of data and provide real-time analysis and insights.

KEYWORD

Real-time analytics is the discipline that applies logic and mathematics to data to provide insights for making better decisions quickly.

2.6.3.4. Content Management

NoSQL databases are used in content management applications, where flexible schema and high performance are essential. These databases can handle large volumes of unstructured data and provide real-time search and retrieval (Chandra, 2015).

2.6.3.5. Gaming

NoSQL databases are commonly used in gaming applications, which involve managing large amounts of user data, such as profiles, preferences, and game progress. These databases can provide real-time updates and ensure high availability and performance.

2.6.3.6. Mobile Applications

NoSQL databases are used in mobile applications, where high performance and scalability are essential. These databases can handle large volumes of data and provide real-time synchronization and offline support (Membrey et al., 2010).

2.6.4. Advantages of NoSQL Databases

NoSQL databases offer several advantages over traditional relational databases, including:

2.6.4.1. Scalability

NoSQL databases are designed to scale horizontally, which means that they can easily handle large amounts of data by adding more nodes to a cluster. This makes them ideal for applications that require high levels of scalability, such as social media platforms, e-commerce websites, and online gaming platforms.

KEYWORD

E-commerce is the activity of electronically buying or selling of products on online services or over the Internet.

2.6.4.2. Flexibility

NoSQL databases are highly flexible and can easily accommodate changes in data structures and data types (Han et al., 2011). This makes them ideal for applications that require frequent updates and modifications, such as CMS, mobile apps, and IoT devices.

2.6.4.3. High Performance

NoSQL databases are optimized for read and write performance, making them ideal for applications that require fast and efficient data access (Leavitt, 2010). This includes real-time analytics, financial trading systems, and high-speed trading platforms.

2.6.4.4. Cost-Effective

NoSQL databases are often more cost-effective than traditional relational databases, as they do not require expensive hardware or software licenses (Okman et al., 2011). They are also highly scalable, which means that they can handle large amounts of data without requiring additional infrastructure.

2.6.4.5. Availability

NoSQL databases are designed for high availability, which means that they can continue to operate even if some nodes in the cluster fail (Nayak et al., 2013). This makes them ideal for applications that require high levels of uptime and reliability, such as online banking and e-commerce platforms.

2.6.4.6. Schema-Less Design

NoSQL databases do not enforce a fixed schema, allowing for a more flexible data model. This makes it easier to store and access unstructured or semi-structured data, such as JSON documents.

Overall, the advantages of NoSQL databases make them an attractive option for many different types of applications, particularly those that require high levels of scalability, performance, and availability.

2.6.5. Disadvantages of NoSQL Databases

NoSQL databases also have certain disadvantages.

2.6.5.1. Limited Query Capability

NoSQL databases do not support complex queries as compared to SQL databases. They are designed to provide quick access to large amounts of data, but when it comes to data analysis, they may not be the best fit (Sicari et al., 2022).

KEYWORD

Query language is a language which is used to retrieve information from a database.

2.6.5.2. Limited Support for ACID Transactions

ACID stands for Atomicity, Consistency, Isolation, and Durability. While NoSQL databases provide high availability and scalability, they do not offer the same level of transactional consistency as SQL databases (Tauro et al., 2012).

2.6.5.3. Lack of Standardization

Unlike SQL databases, NoSQL databases lack a standard language for querying and accessing data. Each database has its own set of APIs and query languages, which can be difficult for developers to learn (Meier & Kaufmann, 2019).

2.6.5.4. Lack of Maturity

NoSQL databases are relatively new compared to SQL databases, which have been around for decades. This means that NoSQL databases may not have the same level of maturity or stability as SQL databases.

2.6.5.5. Limited Community Support

As NoSQL databases are relatively new, the community support around them may be limited (Han et al., 2011). This can make it difficult for developers to find resources and solutions when facing problems.

2.6.5.6. Security Concerns

As with any database, NoSQL databases can be vulnerable to security threats such as hacking and data breaches. However, NoSQL databases may be more vulnerable than SQL databases due to their lack of standardization and the fact that they often store large amounts of unstructured data.

2.7. DOCUMENT DATABASES

Document databases are a type of NoSQL database that are specifically designed to store and manage semi-structured and unstructured data (Indermühle et al., 2010). Unlike traditional relational databases, which rely on a rigid schema to organize data, document databases allow for flexible and dynamic data modeling.

In document databases, data is stored as JavaScript Object Notation (JSON) documents. These documents can be nested and can include arrays and other complex data structures. This allows for more natural and intuitive data modeling, since the structure of the data can be tailored to the specific needs of the application (Figure 2.15).

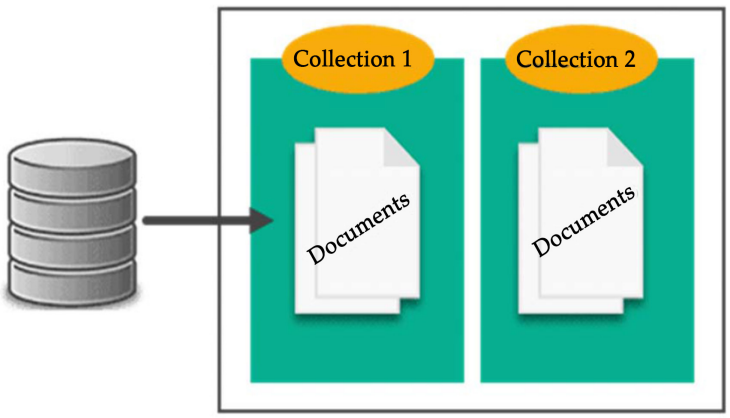


Figure 2.15. Illustration of the document database.

Source: Alex Williams, creative commons license.

Document databases have gained popularity in recent years due to the rise of web and mobile applications that generate and consume large amounts of unstructured data, such as social media posts, sensor data, and user-generated content (O'Gorman & Kasturi, 1995). Document databases provide a scalable and efficient way to store and manage this data, while also allowing for flexible querying and analysis.

Some popular examples of document databases include MongoDB, Couchbase, and Amazon DocumentDB. These databases are often used in e-commerce, content management, and real-time analytics applications. Overall, document databases offer a powerful and versatile tool for managing large and complex data sets in modern applications.

2.7.1. History Document Databases

Document databases, also known as document-oriented databases, are a type of NoSQL database that stores and manages unstructured data in the form of documents. Document databases have a relatively recent history, emerging in the early 2000s as an alternative to traditional relational databases for handling large volumes of unstructured data (Navarro & Baeza-Yates, 1997).

KEYWORD

A **document** is a piece of paper that contains official information.

The first document database, Lotus Notes, was developed by Lotus Development Corporation in 1989. It was initially designed as an email and messaging system, but it evolved to become a platform for building collaborative applications. Lotus Notes stored data in documents, which could contain rich text, images, and other types of data. Documents could be organized in a hierarchical structure, and users could search and retrieve documents using full-text search.

In the early 2000s, document databases began to emerge as a new type of NoSQL database. Apache CouchDB, one of the first popular document databases, was released in 2005. CouchDB was designed to be scalable, fault-tolerant, and easy to use, and it was particularly well-suited for web applications that needed to store and retrieve large volumes of JSON data (AbuSafiya & Mazumdar, 2004).

Since then, a number of other document databases have emerged, including MongoDB, RavenDB, and Couchbase. These databases have become popular for a variety of use cases, including content management, e-commerce, social media, and online gaming.

2.7.2. Importance Document Databases

Document databases are becoming increasingly popular due to their ability to store unstructured data in a scalable and efficient manner. In traditional databases, data is typically stored in tables with predefined schemas, which can make it difficult to store

and manage data that does not fit neatly into these structures. Document databases, on the other hand, are designed to store and manage data as flexible, schema-less documents, which can be easily updated and expanded as needed.

This flexibility makes document databases particularly well-suited for applications that handle large amounts of unstructured data, such as CMS, social media platforms, and e-commerce websites (Zantout & Marir, 1999). With document databases, developers can easily store and retrieve data in a way that is intuitive and natural, without having to worry about the limitations imposed by traditional database schemas.

KEYWORD

Distributed data is data that resides on a DBMS other than your local system.

Additionally, document databases offer excellent performance and scalability, thanks to their ability to distribute data across multiple nodes and clusters. This makes them an ideal choice for applications that need to handle large volumes of data and scale quickly as demand grows.

Overall, the importance of document databases lies in their ability to provide a flexible, scalable, and efficient way to store and manage unstructured data, which is increasingly becoming a critical part of modern application development.

2.7.3. Applications of Document Databases

Document databases are increasingly becoming popular due to their flexibility in handling complex and unstructured data (Kawell Jr et al., 1988). They are suitable for various applications that require fast and efficient data processing. Some of the applications of document databases include:

2.7.3.1. Content Management Systems (CMS)

CMSs rely heavily on document databases to manage, store and retrieve vast amounts of unstructured data such as images, videos, and text (Figure 2.16).

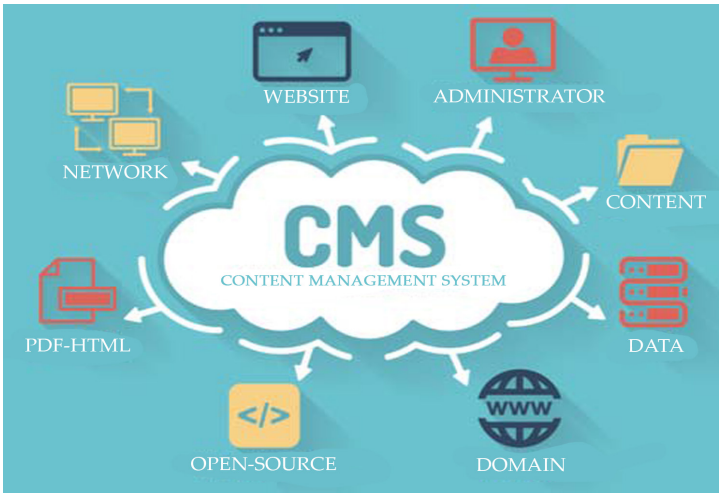


Figure 2.16. Illustration of the content management system.

Source: Pieter Arntz, creative commons license.

2.7.3.2. E-commerce Applications

Document databases are used to store and retrieve product data, customer orders, and transactional data, enabling e-commerce sites to provide a fast and responsive user experience (Moffat et al., 1997).

2.7.3.3. IoT Applications

Internet of Things (IoT) devices generate large volumes of data, which is typically unstructured. Document databases can be used to store and process this data, allowing IoT applications to provide real-time insights and responses.

2.7.3.4. Mobile Applications

Mobile apps often require fast and reliable access to data stored in a remote database. Document databases provide a flexible data model and fast read/write operations that are well-suited to the requirements of mobile apps (Joshi et al., 2002).

2.7.3.5. Real-Time Analytics

Document databases enable businesses to quickly process and analyze large amounts of unstructured data in real-time (Han et al., 2011). This capability is useful in applications such as fraud detection, social media analytics, and sentiment analysis.

2.7.3.6. Scientific Research

Document databases are used in scientific research to store and analyze complex data such as gene sequences, climate data, and astronomical data (Nayak et al., 2013).

Overall, document databases are suitable for applications that require flexibility in handling complex and unstructured data, fast read/write operations, and real-time data processing and analysis.

2.7.4. Advantages of Document Databases

Document databases offer several advantages, which include:

2.7.4.1. Flexible Data Modeling

KEYWORD

Indexing is a very useful technique which helps in optimizing the search time in database query.

Document databases allow flexible data modeling, enabling businesses to store complex and unstructured data in a structured format. This makes it easier to manage and process data and to scale the database system as the business grows.

2.7.4.2. Scalability

Document databases are highly scalable, both horizontally and vertically (Clifton & Garcie-Molina, 2000). Horizontal scalability is achieved by distributing data across multiple servers, while vertical scalability involves increasing the processing power of a single server.

2.7.4.3. High Performance

Document databases can handle large amounts of data and complex queries quickly and efficiently. They use advanced indexing techniques to improve query performance and support fast read and write operations (Walker, 1988).

2.7.4.4. Easy to Use

Document databases are often easier to use than traditional relational databases. They do not require a predefined schema, so developers can start building applications immediately without having to design a schema.

2.7.4.5. Developer Productivity

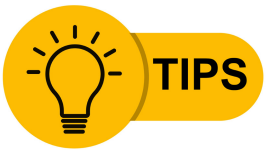
Document databases allow developers to work with data in the same format as their programming language, reducing the need for data mapping and conversion (Ha & Shichkina, 2022). This increases developer productivity and reduces development time.

2.7.4.6. Cost-Effective

Document databases can be less expensive to run than traditional databases. They require less hardware, and the open-source versions are often free to use.

2.7.4.7. Availability

Document databases are highly available, with many offering automatic failover and replication capabilities (Liansheng, 2000). This ensures that data is always accessible, even in the event of a server failure.



Schemas commonly use visual representations to communicate the architecture of the database, becoming the foundation for an organization's data management discipline.

2.7.4.8. Schema Evolution

Document databases allow for schema evolution, which means that the database schema can evolve over time as the data changes. This reduces the need for downtime or complex schema migrations (Bouaziz et al., 2019).

Overall, Document databases are a powerful and flexible solution for managing complex and unstructured data. They offer high performance, scalability, and availability, making them an excellent choice for modern applications that require a flexible data model.

2.7.5. Disadvantages of Document Databases

Document databases, like any other database system, have their own set of advantages and disadvantages. Some of the disadvantages of document databases are:

2.7.5.1. Lack of Standardized Query Language

Unlike relational databases, document databases lack a standardized query language. This can make it difficult for users to interact with

the database, as they must learn a new query language or use a specialized interface to interact with the data.

2.7.5.2. Limited Scalability

Document databases are generally less scalable than other types of databases, which can limit their use in large-scale applications (Popescul et al., 2000). This is because document databases often store data in a single machine or cluster, making it difficult to scale horizontally.

2.7.5.3. Limited Support for Transactions

Many document databases lack support for transactions, making it difficult to maintain data integrity and consistency. This can be particularly problematic for applications that require ACID compliance.

2.7.5.4. Limited Support for Analytics

Document databases are often less suitable for data analysis and reporting than relational databases. This is because document databases lack the structured data necessary for traditional analytics tools and reporting systems.

2.7.5.5. Data Duplication

Document databases often duplicate data across multiple documents, which can lead to redundancy and wasted storage space (Chen & Lynch, 1992). This can also make it more difficult to maintain data consistency and integrity.

ACTIVITY 2.1.

You work for a transportation company that manages a large fleet of vehicles and needs to track the routes, schedules, and maintenance history of each vehicle. Describe how you would use a graph database to store and manage this data, and how it would improve performance and efficiency compared to a relational database.

SUMMARY

The chapter provides an introduction to different types of database systems, including hierarchical, network, relational, object-oriented, graph, document, and NoSQL databases. Each database system has its strengths and weaknesses and is suitable for specific applications based on factors like data structure, scalability, performance, and cost. Hierarchical databases are often used in mainframe applications, network databases for modeling complex data relationships, relational databases for structured data in OLTP and BI, OODBs for complex data types, graph databases for complex relationship data, document databases for unstructured or semi-structured data in web applications, and NoSQL databases for handling large volumes of unstructured or semi-structured data in big data applications or real-time analytics. Selecting the appropriate database system requires careful consideration of these factors.

REVIEW QUESTIONS

1. What is a database system, and what is its purpose?
2. What are some common types of database systems, and how do they differ from one another?
3. What are some advantages of using a hierarchical database system, and in what types of applications is it commonly used?
4. What are some advantages of using a network database system, and in what types of applications is it commonly used?
5. What is a relational database system, and how does it differ from other types of database systems?
6. What are some advantages of using a relational database system, and in what types of applications is it commonly used?
7. What is an object-oriented database system, and in what types of applications is it commonly used?
8. What is a graph database system, and in what types of applications is it commonly used?
9. What is a document database system, and in what types of applications is it commonly used?
10. What is a NoSQL database system, and in what types of applications is it commonly used?

MULTIPLE CHOICE QUESTIONS

1. Which type of database system is often used in mainframe applications?

a. Hierarchical databases

b. Network databases

c. Relational databases
d. Object-oriented databases

2. Which type of database system is useful for modeling complex relationships between data?

a. Hierarchical databases
b. Network databases
c. Relational databases
d. Graph databases

3. Which type of database system organizes data into tables with rows and columns?

a. Hierarchical databases
b. Network databases
c. Relational databases
d. Object-oriented databases

4. Which type of database system is designed to store complex data types such as objects, classes, and inheritance hierarchies?

a. Hierarchical databases
b. Network databases
c. Relational databases
d. Object-oriented databases

5. Which type of database system is designed to store and query data that has complex relationships, such as social networks or supply chain systems?

a. Graph databases
b. Document databases
c. NoSQL databases
d. Relational databases

6. Which type of database system stores unstructured or semi-structured data in documents?

a. Graph databases
b. Document databases
c. NoSQL databases
d. Hierarchical databases

7. Which type of database system is often used in web applications and content management systems?

a. Graph databases
b. Document databases

c. NoSQL databases
d. Object-oriented databases

8. **Which type of database system is designed to handle large volumes of unstructured or semi-structured data?**
   a. Hierarchical databases
   b. Network databases
   c. Relational databases
   d. NoSQL databases
9. **Which type of database system is widely used for storing structured data and is popular in applications such as online transaction processing (OLTP) and business intelligence (BI)?**
   a. Hierarchical databases
   b. Network databases
   c. Relational databases
   d. Object-oriented databases
10. **Which database system uses nodes and edges to represent data and can perform complex queries quickly?**
   a. Graph databases
   b. Document databases
   c. NoSQL databases
   d. Hierarchical databases
11. **Which database system organizes data in a more complex network-like structure, with each record having multiple parents and children?**
   a. Hierarchical databases
   b. Network databases
   c. Relational databases
   d. Object-oriented databases
12. **Which type of database system is often used in OOP languages such as Java or C++?**
   a. Hierarchical databases
   b. Network databases
   c. Relational databases
   d. Object-oriented databases

Answers to Multiple Choice Questions

1. (a); 2. (b); 3. (c); 4. (d); 5. (a); 6. (b); 7. (b); 8. (d); 9 (c); 10. (a);

11. (b) 12. (d)

REFERENCES

1. Abiteboul, S., & Hull, R., (1986). Restructuring hierarchical database objects. *Theoretical Computer Science, 62*(1, 2), 3–38.
2. Abramova, V., & Bernardino, J., (2013). NoSQL databases: MongoDB vs Cassandra. In: *Proceedings of the International C* Conference on Computer Science and Software Engineering* (Vol. 1, pp. 14–22).
3. AbuSafiya, M., & Mazumdar, S., (2004). Accommodating paper in document databases. In: *Proceedings of the 2004 ACM Symposium on Document Engineering* (Vol. 1, pp. 155–162).
4. Albano, A., Ghelli, G., & Orsini, R., (1989). Types for databases: The Galileo experience. In: *Proceedings of the 2nd International Workshop on Database Programming Languages* (Vol. 1, pp. 196–206).
5. Alonso, R., & Korth, H. F., (1993). Database system issues in nomadic computing. In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (Vol. 1, pp. 388–392).
6. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., & Vrgoč, D., (2017). Foundations of modern query languages for graph databases. *ACM Computing Surveys (CSUR), 50*(5), 1–40.
7. Atkinson, M., Dewitt, D., Maier, D., Bancilhon, F., Dittrich, K., & Zdonik, S., (1990). The object-oriented database system manifesto. In: *Deductive and Object-Oriented Databases* (Vol. 1, pp. 223–240). North-Holland.
8. Atzeni, P., & De Antonellis, V., (1993). *Relational Database Theory* (Vol. 1, pp. 2–5). Benjamin-Cummings Publishing Co., Inc.
9. Bader, G. D., Betel, D., & Hogue, C. W., (2003). BIND: The biomolecular interaction network database. *Nucleic Acids Research, 31*(1), 248–250.
10. Banerjee, J., Hsiao, D. K., & Ng, F. K., (1980). Database transformation, query translation, and performance analysis of a new database computer in supporting hierarchical database management. *IEEE Transactions on Software Engineering*, (1), 91–109.
11. Batra, S., & Tyagi, C., (2012). Comparative analysis of relational and graph databases. *International Journal of Soft Computing and Engineering (IJSCE), 2*(2), 509–512.
12. Beeri, C., (1990). Formal models for object oriented databases. In: *Deductive and Object-Oriented Databases* (Vol. 1, pp. 405–430). North-Holland.

13. Bernstein, P. A., (1998). Repositories and object oriented databases. *ACM Sigmod Record, 27*(1), 88–96.
14. Bertino, E., & Martino, L., (1991). Object-oriented database management systems: Concepts and issues. *Computer, 24*(4), 33–47.
15. Bhat, U., & Jadhav, S., (2010). Moving towards non-relational databases. *International Journal of Computer Applications, 1*(13), 40–47.
16. Black, J., Ellis, T., & Makris, D., (2004). A hierarchical database for visual surveillance applications. In: *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No. 04TH8763)* (Vol. 3, pp. 1571–1574). IEEE.
17. Blasgen, M. W., & Eswaran, K. P., (1977). Storage and access in relational data bases. *IBM Systems Journal, 16*(4), 363–377.
18. Bordoloi, S., & Kalita, B., (2013). Designing graph database models from existing relational databases. *International Journal of Computer Applications, 74*(1), 2–7.
19. Bouaziz, S., Nabli, A., & Gargouri, F., (2019). Design a data warehouse schema from document-oriented database. *Procedia Computer Science, 159*, 221–230.
20. Bouganim, L., Florescu, D., & Valduriez, P., (1996). *Dynamic Load Balancing in Hierarchical Parallel Database Systems* (Vol. 3, No. 1, pp. 5–10). Doctoral dissertation, INRIA.
21. Brodie, M. L., (1980). The application of data types to database semantic integrity. *Information Systems, 5*(4), 287–296.
22. Castellana, V. G., Morari, A., Weaver, J., Tumeo, A., Haglin, D., Villa, O., & Feo, J., (2015). In-memory graph databases for web-scale data. *Computer, 48*(3), 24–35.
23. Cellary, W., & Jomier, G., (1990). Consistency of versions in object-oriented databases. In: *VLDB* (Vol. 90, pp. 432–441).
24. Chandra, D. G., (2015). BASE analysis of NoSQL database. *Future Generation Computer Systems, 52*(1), 13–21.
25. Chen, H., & Lynch, K. J., (1992). Automatic construction of networks of concepts characterizing document databases. *IEEE Transactions on Systems, Man, and Cybernetics, 22*(5), 885–902.
26. Chou, T. S., Yen, K. K., & Luo, J., (2008). Network intrusion detection design using feature selection of soft computing paradigms. *International Journal of Computer and Information Engineering, 2*(11), 3722–3734.
27. Chung, W. Y., Yu, P. S., & Huang, C. J., (2013). Cloud computing system based on wireless sensor network. In: *2013 Federated Conference on Computer Science and Information Systems* (Vol. 1, pp. 877–880). IEEE.
28. Clifton, C., & Garcie-Molina, H., (2000). The design of a document database. In: *Proceedings of the ACM Conference on Document Processing Systems* (Vol. 1, pp. 125–134).
29. Codd, E. F., (2007). Relational database: A practical foundation for productivity. In: *ACM Turing Award Lectures* (Vol. 1, p. 1981).

30. De Almeida, V. T., & Güting, R. H., (2005). Supporting uncertainty in moving objects in network databases. In: *Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems* (Vol. 1, pp. 31–40).
31. De Caluwe, R., (1997). *Fuzzy and Uncertain Object-Oriented Databases: Concepts and Models* (Vol. 13, pp. 2–9). World Scientific.
32. Debrouvier, A., Parodi, E., Perazzo, M., Soliani, V., & Vaisman, A., (2021). A model and query language for temporal graph databases. *The VLDB Journal, 30*(5), 825–858.
33. Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L., (2009). ImageNet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Vol. 1, pp. 248–255). IEEE.
34. Deniša, M., & Ude, A., (2015). Synthesis of new dynamic movement primitives through search in a hierarchical database of example movements. *International Journal of Advanced Robotic Systems, 12*(10), 137.
35. DeWitt, D. J., Futtersack, P., Maier, D., & Velez, F., (1990). *A Study of Three Alternative Workstation-Server Architectures for Object Oriented Database Systems* (Vol. 1, pp. 2–9). University of Wisconsin-Madison Department of Computer Sciences.
36. Domdouzis, K., Lake, P., & Crowther, P., (2021). Hierarchical databases. In: *Concise Guide to Databases: A Practical Introduction* (Vol. 1, pp. 205–212). Cham: Springer International Publishing.
37. Dunham, M. H., & Helal, A., (1995). Mobile computing and databases: Anything new? *ACM Sigmod Record, 24*(4), 5–9.
38. Dy-Liacco, T. E., (1994). Modern control centers and computer networking. *IEEE Computer Applications in Power, 7*(4), 17–22.
39. Gavish, B., & Pirkul, H., (1986). Computer and database location in distributed computer systems. *IEEE Transactions on Computers, 35*(07), 583–590.
40. Gotthard, W., Lockemann, P. C., & Neufeld, A., (1992). System-guided view integration for object-oriented databases. *IEEE Transactions on Knowledge and Data Engineering, 4*(1), 1–22.
41. Gray, P. M., Kulkarni, K. G., & Paton, N. W., (1992). *Object-Oriented Databases: A Semantic Data Model Approach* (Vol. 1, pp. 2–8). Prentice-Hall, Inc.
42. Güting, R. H., & Schneider, M., (1993). Realms: A foundation for spatial data types in database systems. In: *Advances in Spatial Databases: Third International Symposium, SSD'93 Singapore, June 23–25, 1993 Proceedings 3* (Vol. 1, pp. 14–35). Springer Berlin Heidelberg.
43. Ha, M., & Shichkina, Y., (2022). Translating a distributed relational database to a document database. *Data Science and Engineering, 7*(2), 136–155.
44. Han, J., Haihong, E., Le, G., & Du, J., (2011). Survey on NoSQL database. In: *2011 6th International Conference on Pervasive Computing and Applications* (Vol. 1, pp. 363–366). IEEE
45. Han, J., Song, M., & Song, J., (2011). A novel solution of distributed memory NOSQL

database for cloud computing. In: *2011 10th IEEE/ACIS International Conference on Computer and Information Science* (Vol. 1, pp. 351–355). IEEE.

46. Haseeb, A., & Pattun, G., (2017). A review on NoSQL: Applications and challenges. *International Journal of Advanced Research in Computer Science, 8*(1), 2–6.
47. Hesse, B. W., Sproull, L. S., Kiesler, S. B., & Walsh, J. P., (1993). Returns to science: Computer networks in oceanography. *Communications of the ACM, 36*(8), 90–101.
48. Ho, J. S., & Akyildiz, I. F., (1997). Dynamic hierarchical database architecture for location management in PCS networks. *IEEE/ACM Transactions on Networking, 5*(5), 646–660.
49. Hsu, M., & Madnick, S. E., (1983). Hierarchical database decomposition: A technique for database concurrency control. In: *Proceedings of the 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (Vol. 1, pp. 182–191).
50. Hurson, A. R., Pakzad, S. H., & Cheng, J. B., (1993). Object-oriented database management systems: Evolution and performance issues. *Computer, 26*(2), 48–58.
51. Imieliński, T., & Lipski, Jr. W., (1984). Incomplete information in relational databases. *Journal of the ACM (JACM), 31*(4), 761–791.
52. Indermühle, E., Liwicki, M., & Bunke, H., (2010). IAMonDo-database: An online handwritten document database with non-uniform contents. In: *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems* (Vol. 1, pp. 97–104).
53. Ipeirotis, P. G., & Gravano, L., (2002). Distributed search over the hidden web: Hierarchical database sampling and selection. In: *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases* (Vol. 1, pp. 394–405). Morgan Kaufmann.
54. Jaiswal, G., & Agrawal, A. P., (2013). Comparative analysis of relational and graph databases. *IOSR Journal of Engineering (IOSRJEN), 3*(8), 25–27.
55. Jatana, N., Puri, S., Ahuja, M., Kathuria, I., & Gosain, D., (2012). A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology, 1*(6), 1–5.
56. Jin, R., Hong, H., Wang, H., Ruan, N., & Xiang, Y., (2010). Computing label-constraint reachability in graph databases. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (Vol. 1, pp. 123–134).
57. Jindal, G., & Bali, S., (2012). Hierarchical model leads to the evolution of relational model. *International Journal of Engineering and Management Research (IJEMR), 2*(4), 11–14.
58. Joseph, J. V., Thatte, S. M., Thompson, C. W., & Wells, D. L., (1991). Object-oriented databases: Design and implementation. *Proceedings of the IEEE, 79*(1), 42–64.
59. Joshi, J. B., Li, Z. K., Fahmi, H., Shafiq, B., & Ghafoor, A., (2002). A model for secure multimedia document database system in a distributed environment. *IEEE Transactions on Multimedia, 4*(2), 215–234.
60. Jouili, S., & Vansteenberghe, V., (2013). An empirical comparison of graph databases.

In: *2013 International Conference on Social Computing* (Vol. 1, pp. 708–715). IEEE.

61. Kawell, Jr. L., Beckhardt, S., Halvorsen, T., Ozzie, R., & Greif, I., (1988). Replicated document management in a group communication system. In: *Proceedings of the 1988 ACM Conference on Computer-Supported Cooperative Work* (Vol. 1, p. 395).
62. Kearns, J. P., & DeFazio, S., (1983). Locality of reference in hierarchical database systems. *IEEE Transactions on Software Engineering*, (2), 128–134.
63. Kim, W., (1990). Object-oriented databases: Definition and research directions. *IEEE Transactions on Knowledge and Data Engineering, 2*(3), 327–341.
64. Kim, W., Banerjee, J., Chou, H. T., & Garza, J. F., (1990). Object-oriented database support for CAD. *Computer-Aided Design, 22*(8), 469–479.
65. Kim, W., Banerjee, J., Chou, H. T., Garza, J. F., & Woelk, D., (1987). Composite object support in an object-oriented database system. In: *Conference Proceedings on Object-oriented Programming Systems, Languages and Applications* (Vol. 1, pp. 118–125).
66. Kolahdouzan, M., & Shahabi, C., (2004). Voronoi-based k nearest neighbor search for spatial network databases. In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases* (Vol. 1, 30, pp. 840–851).
67. Kumar, K. R., (2015). Graph databases: A survey. In: *International Conference on Computing, Communication & Automation* (Vol. 1, pp. 785–790). IEEE.
68. Lai, Y. P., & Hsia, P. L., (2007). Using the vulnerability information of computer systems to improve the network security. *Computer Communications, 30*(9), 2032–2047.
69. Larson, J. A., (1983). Bridging the gap between network and relational database management systems. *Computer, 16*(09), 82–92.
70. Leavitt, N., (2000). Whatever happened to object-oriented databases? *Computer, 33*(08), 16–19.
71. Leavitt, N., (2010). Will NoSQL databases live up to their promise? *Computer, 43*(2), 12–14.
72. Lee, J., Han, W. S., Kasperovics, R., & Lee, J. H., (2012). An in-depth comparison of subgraph isomorphism algorithms in graph databases. *Proceedings of the VLDB Endowment, 6*(2), 133–144.
73. Levene, M., & Loizou, G., (2012). *A Guided Tour of Relational Databases and Beyond*. Springer Science & Business Media.
74. Liansheng, M., (2000). Document database construction in China in the 1990s: A review of developments. *The Electronic Library, 18*(3), 210–215.
75. Liao, Y. T., Zhou, J., Lu, C. H., Chen, S. C., Hsu, C. H., Chen, W., & Chung, Y. C., (2016). Data adapter for querying and transformation between SQL and NoSQL database. *Future Generation Computer Systems, 65*, 111–121.
76. Maier, D., (1983). *The Theory of Relational Databases* (Vol. 1, 11, pp. 2–4). Rockville: Computer science press.

77. Meier, A., & Kaufmann, M., (2019). *SQL & NoSQL Databases* (Vol. 1, pp. 6–9). Berlin/Heidelberg, Germany: Springer Fachmedien Wiesbaden.

78. Meier, A., Dippold, R., Mercerat, J., Muriset, A., Untersinger, J. C., Eckerlin, R., & Ferrara, F., (1994). Hierarchical to relational database migration. *IEEE Software, 11*(3), 21–27.

79. Meier, A., Kaufmann, M., Meier, A., & Kaufmann, M., (2019). NoSQL databases. *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management, 1*, 201–218.

80. Membrey, P., Plugge, E., Hawkins, T., & Hawkins, D., (2010). *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing* (Vol. 1, pp. 3–8). Springer.

81. Mendelzon, A. O., & Wood, P. T., (1995). Finding regular simple paths in graph databases. *SIAM Journal on Computing, 24*(6), 1235–1258.

82. Mishra, P., & Eich, M. H., (1992). Join processing in relational databases. *ACM Computing Surveys (CSUR), 24*(1), 63–113.

83. Moffat, A., Zobel, J., & Sharman, N., (1997). Text compression for dynamic document databases. *IEEE Transactions on Knowledge and Data Engineering, 9*(2), 302–313.

84. Moniruzzaman, A. B. M., & Hossain, S. A., (2013). *NoSQL Database: New Era of Databases for Big Data Analytics-Classification, Characteristics and Comparison* (Vol. 1, pp. 4–9).

85. Moszer, I., Glaser, P., & Danchin, A., (1995). SubtiList: A relational database for the *Bacillus subtilis* genome. *Microbiology, 141*(2), 261–268.

86. Murty, R., Chandra, R., Moscibroda, T., & Bahl, P., (2011). Senseless: A database-driven white spaces network. *IEEE Transactions on Mobile Computing, 11*(2), 189–203.

87. Nakada, H., Sato, M., & Sekiguchi, S., (1999). Design and implementations of Ninf: Towards a global computing infrastructure. *Future Generation Computer Systems, 15*(5, 6), 649–658.

88. Navarro, G., & Baeza-Yates, R., (1997). Proximal nodes: A model to query document databases by content and structure. *ACM Transactions on Information Systems (TOIS), 15*(4), 400–435.

89. Nayak, A., Poriya, A., & Poojary, D., (2013). Type of NOSQL databases and its comparison with relational databases. *International Journal of Applied Information Systems, 5*(4), 16–19.

90. Nicklin, P. J., Powell, G. H., & Hollings, J. P., (1985). Hierarchical data management for structural analysis. *Engineering with Computers, 1*, 45–54.

91. O'Gorman, L., & Kasturi, R., (1995). *Document Image Analysis* (Vol. 39, pp. 3–9). Los Alamitos: IEEE Computer Society Press.

92. Ogle, V. E., & Stonebraker, M., (1995). Chabot: Retrieval from a relational database of images. *Computer, 28*(9), 40–48.

93. Ohori, A., (1990). Semantics of types for database objects. *Theoretical Computer Science, 76*(1), 53–91.

94. Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., & Abramov, J., (2011). Security issues in NoSQL databases. In: *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications* (Vol. 1, pp. 541–547). IEEE.

95. Orenstein, J. A., (1986). Spatial query processing in an object-oriented database system. In: *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data* (Vol. 1, pp. 326–336).

96. Outrata, J., & Vychodil, V., (2012). Fast algorithm for computing fixpoints of Galois connections induced by object-attribute relational data. *Information Sciences, 185*(1), 114–127.

97. Padmanabhan, P., Gruenwald, L., Vallur, A., & Atiquzzaman, M., (2008). A survey of data replication techniques for mobile ad hoc network databases. *The VLDB Journal, 17*(1), 1143–1164.

98. Padmanabhan, S., Malkemus, T., Jhingran, A., & Agarwal, R., (2001). Block oriented processing of relational database operations in modern computer architectures. In: *Proceedings 17th International Conference on Data Engineering* (Vol. 1, pp. 567–574). IEEE.

99. Papadias, D., Zhang, J., Mamoulis, N., & Tao, Y., (2003). Query processing in spatial network databases. In: *Proceedings 2003 VLDB Conference* (Vol. 1, pp. 802–813). Morgan Kaufmann.

100. Pham, M. C., & Klamma, R., (2010). The structure of the computer science knowledge network. In: *2010 International Conference on Advances in Social Networks Analysis and Mining* (Vol. 1, pp. 17–24). IEEE.

101. Pokorny, J., (2011). NoSQL databases: A step to database scalability in web environment. In: *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services* (Vol. 1, pp. 278–283).

102. Pokorný, J., (2015). Graph databases: Their power and limitations. In: *Computer Information Systems and Industrial Management: 14th IFIP TC 8 International Conference, CISIM 2015, Warsaw, Poland, September 24–26, 2015, Proceedings 14* (Vol. 1, pp. 58–69). Springer International Publishing.

103. Pokorný, J., Valenta, M., & Kovačič, J., (2017). Integrity constraints in graph databases. *Procedia Computer Science, 109*, 975–981.

104. Popescul, A., Flake, G. W., Lawrence, S., Ungar, L. H., & Giles, C. L., (2000). Clustering and identifying temporal trends in document databases. In: *Proceedings IEEE Advances in Digital Libraries 2000* (Vol. 1, pp. 173–182). IEEE.

105. Ranu, S., & Singh, A. K., (2009). Graphsig: A scalable approach to mining significant subgraphs in large graph databases. In: *2009 IEEE 25th International Conference on Data Engineering* (Vol. 1, pp. 844–855). IEEE.

106. Robinson, J. M., (2013). Computer-assisted peer review. In: *Computer-Assisted Assessment in Higher Education* (Vol. 1, pp. 95–102). Routledge.

107. Roussopoulos, N., (1982). View indexing in relational databases. *ACM Transactions on Database Systems (TODS), 7*(2), 258–290.

108. Sato, M., Nakada, H., Sekiguchi, S., Matsuoka, S., Nagashima, U., & Takagi, H., (1997). Ninf: A network based information library for global world-wide computing infrastructure. In: *HPCN Europe* (Vol. 1225, pp. 491–502).

109. Schindler, T., (2018). *Anomaly Detection in Log Data Using Graph Databases and Machine Learning to Defend Advanced Persistent Threats* (Vol. 1, pp. 4–9).

110. Schneider, M., (1997). *Spatial Data Types for Database Systems: Finite Resolution Geometry for Geographic Information Systems* (Vol. 1, pp. 5–9). Berlin, Heidelberg: Springer Berlin Heidelberg.

111. ShefaliPatil, G., & Bhatia, A., (2014). *Graph Databases-an Overview* (Vol. 2, pp. 657–660). 1Student, ME Computers, Terna College of Engg, Navi Mumbai.

112. Shimpi, D., & Chaudhari, S., (2012). An overview of graph databases. In: *IJCA Proceedings on International Conference on Recent Trends in Information Technology and Computer Science* (Vol. 1, pp. 16–22).

113. Shokoufandeh, A., Macrini, D., Dickinson, S., Siddiqi, K., & Zucker, S. W., (2005). Indexing hierarchical structures using graph spectra. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*(7), 1125–1140.

114. Sicari, S., Rizzardi, A., & Coen-Porisini, A., (2022). Security & privacy issues and challenges in NoSQL databases. *Computer Networks, 1*, 108–828.

115. Silberschatz, A., & Kedem, Z., (1980). Consistency in hierarchical database systems. *Journal of the ACM (JACM), 27*(1), 72–80.

116. Singh, M., & Kaur, K., (2015). SQL2Neo: Moving health-care data from relational to graph databases. In: *2015 IEEE International Advance Computing Conference (IACC)* (Vol. 1, pp. 721–725). IEEE.

117. Stefanidis, K., Drosou, M., & Pitoura, E., (2009). You may also like results in relational databases. In: *Proceedings International Workshop on Personalized Access, Profile Management and Context Awareness: Databases, Lyon, France,* (Vol. 1, pp. 5–7).

118. Stolte, C., Tang, D., & Hanrahan, P., (2002). Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics, 8*(1), 52–65.

119. Stonebraker, M., (1986). *The Ingres Papers: Anatomy of a Relational Database System* (Vol. 1, pp. 8–10). Addison-Wesley Longman Publishing Co., Inc.

120. Tangorra, F., & Chiarolla, D., (1995). A methodology for reverse engineering hierarchical databases. *Information and Software Technology, 37*(4), 225–231.

121. Tauro, C. J., Aravindh, S., & Shreeharsha, A. B., (2012). Comparative study of the new generation, agile, scalable, high performance NOSQL databases. *International Journal of Computer Applications, 48*(20), 1–4.

122. Teorey, T. J., Yang, D., & Fry, J. P., (1986). A logical design methodology for relational

databases using the extended entity-relationship model. *ACM Computing Surveys (CSUR), 18*(2), 197–222.

123. Tsichritzis, D. C., & Lochovsky, F. H., (1976). Hierarchical data-base management: A survey. *ACM Computing Surveys (CSUR), 8*(1), 105–123.
124. Vargo, C. G., Brown, C. E., & Swierenga, S. J., (1992). An evaluation of computer-supported backtracking in a hierarchical database. In: *Proceedings of the Human Factors Society Annual Meeting* (Vol. 36, No. 4, pp. 356–360). Sage CA: Los Angeles, CA: SAGE Publications.
125. Walker, J. H., (1988). Supporting document development with Concordia. *Computer, 21*(1), 48–59.
126. Wells, D. L., Blakeley, J. A., & Thompson, C. W., (1992). Architecture of an open object-oriented database management system. *Computer, 25*(10), 74–82.
127. Wood, P. T., (2012). Query languages for graph databases. *ACM Sigmod Record, 41*(1), 50–60.
128. Wu, C. H., & Chang, T. C., (1991). Protein classification using a neural network database system. In: *Proceedings of the Conference on Analysis of Neural Network Applications* (Vol. 1, pp. 29–41).
129. Wuu, G. T., & Dayal, U., (1992). A uniform model for temporal object-oriented databases. In: *1992 Eighth International Conference on Data Engineering* (Vol. 1, pp. 584–585). IEEE Computer Society.
130. Zaki, A. K., (2014). NoSQL databases: New millennium database for big data, big users, cloud computing and its security challenges. *International Journal of Research in Engineering and Technology (IJRET), 3*(15), 403–409.
131. Zand, M., Collins, V., & Caviness, D., (1995). A survey of current object-oriented databases. *ACM SIGMIS Database: The DATABASE for Advances in Information Systems, 26*(1), 14–29.
132. Zantout, H., & Marir, F., (1999). Document management systems from current capabilities towards intelligent information retrieval: An overview. *International Journal of Information Management, 19*(6), 471–484.
133. Zdonik, S. B., & Maier, D., (1990). *Readings in Object-Oriented Database Systems* (Vol. 1, pp. 2–7). Morgan Kaufmann.
134. Zhang, Z. J., (2017). Graph databases for knowledge management. *IT Professional, 19*(6), 26–32.

CHAPTER 3

DATABASE MODELING

UNIT INTRODUCTION

Keeping detailed records is essential for every company. Due to the growing importance of data management in today's economy, a sizable portion of the global computing infrastructure is devoted to this task (Hull & King, 1987).

Almost every industry nowadays uses some form of database. There are databases for storing everything from emails and contacts to sales figures and bank details. The search continues for effective ways to archive less-structured data, such as domain expertise.

Relational databases and the language used in SQL will be covered in depth in a subsequent essay. In this introduction, we will cover some of the fundamentals of the technology, with a highlight on normalizing databases.

Edgar Frank Codd (nearly always referred to as E. F. Codd in practical literature) initially introduced the idea of relational databases in the IBM research report RJ599, on August 19, 1969. 1 However, "A Relational Model of Data for Massive Shared Data Banks," issued in Communications of the ACM, is the work that is typically regarded as the foundation of this technology (Angles & Gutierrez, 2008). Online access to the entire article is limited to its initial section.

For relational database implementations, additional writings by E. F. Codd from the 1970s and 1980s are still regarded as canon. On October 14, 1985, and October 21, 1985, respectively, two Computer world articles titled "Is Your DBMS Actually Relational?" and "Does Your DBMS Operate by the Rules?" presented his well-known "Twelve Rules for Relational Databases"2. The 12 rules, which he originally issued in his book "The Relational Model for Database Management, Version 2," have subsequently been expanded, and there are now 333 of them (Ma, 2007).

In order to describe, control, and question the data in the database, which is conveyed as a string of characters, it is necessary to have a language that complies with all twelve of Codd's rules. All of the top relational database providers have embraced the language, known as SQL, which was first created at the research division of IBM (originally in Yorktown Heights, New York, and then in San Jose, California) 3 (Teorey et al., 2011). Structured Query Language (SQL) was what the acronym SQL originally stood for. SEQUEL is short for Sequential English Query Language and was the name of the first commercially available language implementation, which was a component of IBM's SEQUEL/DS product. For legal concerns, the name was altered after sometime. The pronunciation "see-quell" is therefore used by many veteran database developers (Peckham & Maryanski, 1988).

As an ANSI/ISO standard, SQL has been accepted. Despite being updated in 1999 (sometimes referred to as SQL99 or SQL3), the majority of suppliers still do not adhere to the 1992 version of the customary in full. Because the 1992 standard is shorter and easier for users to reference, and because only few of the 1999 explicit necessities are usually applied at this time, it might be a preferable place to start while learning the language (Figure 3.1).

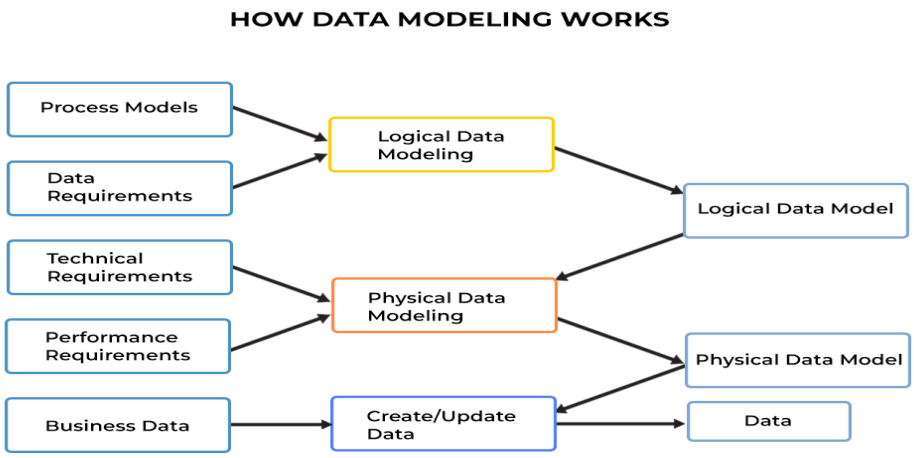


Figure 3.1. Illustration of working of database modeling.

Source: Spice works, creative commons license.

Learning Objectives

At the end of this chapter, readers will be able to:

- Define data modeling and its importance in database design.
- Explain the entity relationship model and how it represents the relationships between data objects.
- Describe the part of data modeling in the larger context of database design.
- Identify data objects and their associations using the entity association model.
- Develop a basic schema for a database using the entity relationship model.
- Refine the entity relationship diagram by adding detail and clarity.
- Understand the purpose and function of primary and foreign keys in a database schema.
- Add attributes to the entity relationship model to more accurately represent the data objects.
- Define generalization hierarchies and how they can be used to simplify complex data relationships.
- Implement data integrity rules to ensure data consistency and accuracy.
- Explain the relational model and how it relates to the entity relationship model.

Key Terms

- Adding Attributes
- Data Modeling
- Data Objects
- Database Design
- Diagram
- Entity Relationship Model
- Foreign Keys
- Generalization Hierarchies
- Primary Keys
- Refining
- Schema

3.1. OVERVIEW OF DATA MODELING

In a data model, important data structures for a database. The data objects, linkages between the items, and rules governing how the objects can be operated on are all included in the data structures (Petkovic & Jonker, 2000). As suggested by the name, the data model is more concerned with the types of data that are needed and how they should be organized than it is with the actual operations that will be carried out on the data. The model of data is comparable to the blueprints for a structure, to use a standard analogy. No hardware or software limitations apply to a data model. The data model emphases on displaying the statistics as the user perceives it in the "actual world," as a disparate to trying to denote the data as it would be seen by a database (Aracic et al., 2006). The physical illustration of those thoughts in a database and the ideas that make up actual processes and events are connected by this bridge.

3.1.1. Methodology

Entity-Relationship (ER) method and Object Model are the two main approaches used to design a data model. ER theory is applied in this text (Figure 3.2).

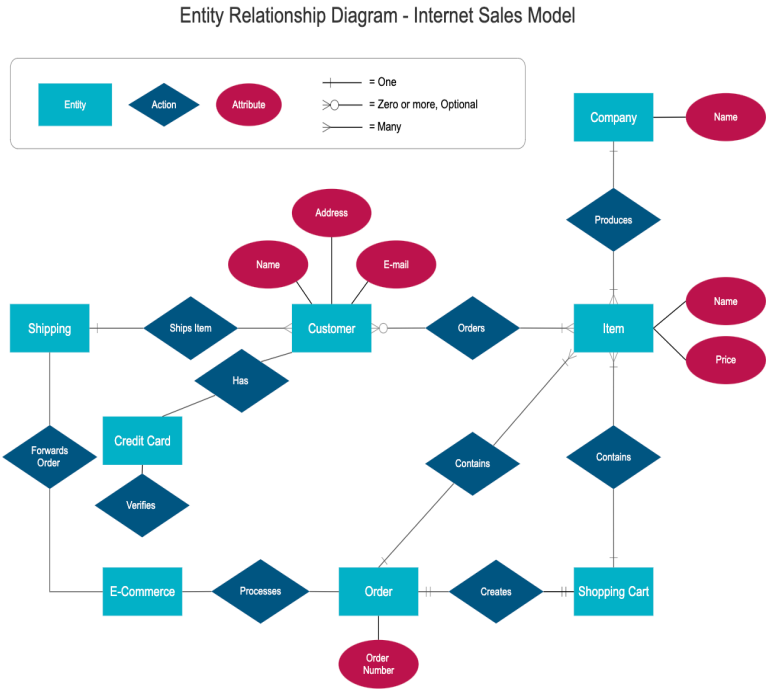


Figure 3.2. Flow diagram of entity-relationship (ER).

Source: Smartdraw, creative commons license.

3.1.2. Data Modeling in the Setting of Database Design

Manipulating the rational and physical organization of one or more than one databases to meet the info wants of operators in an institute for a certain set of bids is what is meant by the definition of database design. The five steps of the design process are following:

1. Preparation and evaluation;
2. Conceptual planning;
3. Logical layout;
4. Physical layout; and
5. Implementation (Kerre & Chen, 1995).

One step in the theoretical designing process is the model of data. The functional model is usually the other. The functional model addresses how to handle the data, while the data model concentrates on which data had it better be saved in the database. Relational tables are planned by means of the data model, to position this in the perspective of the relational databases. The enquiries that drive access and function on those tables are shaped with the functional model (Figure 3.3).

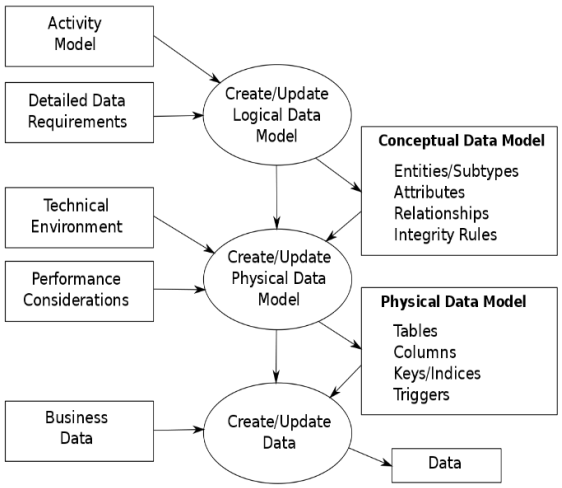


Figure 3.3. Schematic of data modeling.

Source: J.H. Bekke, creative commons license.

3.1.3. Constituents of a Data Model

Planning and investigation stage outputs are fed into the data

model (Ma & Yan, 2010). Here, the modeler and analysts examine the current documentation and speak with end users to gather knowledge about the database's requirements.

The data model produces two results. The 1st is a picture-based entity-relationship (ER) figure that shows the data structures (Modolo et al., 2018). Since the graphic is simple to understand, it is an effective tool for explaining the model to the last user. A data document is the second part. This paper provides a detailed description of the relationships, rules, and data items needed by the database. The vocabulary offers the information needed by the database designer to build the actual database.

KEYWORD

A foreign key (FK) is a column or combination of columns that is used to establish and enforce a link between the data in two tables to control the data that can be stored in the foreign key table.

3.1.4. Significance of Data Modeling

The most backbreaking and labor-intensive stage in the progress process is likely data modeling. Why bother, especially if time is of the essence? Practitioners who write about the topic frequently respond that you should know more to construct a database without an exemplary than you have to know to build a home without plans.

The data model's aim is to confirm that entire data objects that the database needs is totally and precisely represented (Deville et al., 2003). The data model may be examined and validated as accurate by the end-users since it use plain language and notations that are simple to understand.

Database designers can use the data model as a detailed "blueprint" to create the actual database from scratch. All interactive tables, primary keys, foreign keys, stored events, and triggers will be defined based on the data model. Inefficient database layouts will add more work in the long run. The database you build may lack the information you need to generate vital reports, return inaccurate or inconsistent findings, and be unable to adapt to your needs as they evolve if you don't give it enough thought beforehand.

3.2. THE ENTITY-RELATIONSHIP MODEL

In command to combine the network and relational database perspectives, Peter first suggested the ER model in 1976 (Chen, 1976). The ER model, to put it simply, is a conceptual data model that perceives the real life as entailing of entities and relations. ER diagrams, which are used to represent data items visually, are an

important part of models. Since Chen's article, the paradigm has expanded and is now extensively used for database design. The ER model is useful for the database developer because:

1. The relational model is well mapped to it. Relational tables can be formed quickly from the ER model's concepts (Moody et al., 2005).
2. It requires little training and is straightforward to grasp. In direction to explain the enterprise to the user, the database designer can use the model.
3. The model can also be utilized as a design strategy by the database designer to incorporate a data model in a particular database managing system (Gregersen & Jensen, 1999).

Remember

The Entity-Relationship Model is a conceptual model used to design and represent relationships between data entities in a database.

3.2.1. Basic Concepts of E-R Modeling

According to the ER model, the actual world is made up of associations between different items.

3.2.2. Entities

The main data object that has to have information collected on is an entity. Entities often consist of recognizable notions that might be either abstract or concrete, such as individuals, locations, objects, or events that are relevant to the database (Chen, 1977). Employees, projects, and invoices are a few instances of entities in more detail. The relational paradigm compares an entity to a table.

There are two categories of entities: independent and rely on. A self-identifying object is one that do not depend on another. A dependent unit is one whose identification depends on another.

An entity's individual occurrences are referred to as entity occurrences or instances. A row in a relational table and an occurrence are equivalent.

3.2.3. Special Entity Types

Association entities, often referred to as intersectional entities, are utilized to connect dual or more entities to make sense of a many-to-many relationship (Batini & Lenzerini, 1984). In generalization hierarchies, sub-types are used to characterize a subset of cases

of their parental item, known as the super type, but only for which certain characteristics or relations apply. Further information about generalization hierarchies and associative entities is provided Figure 3.4.

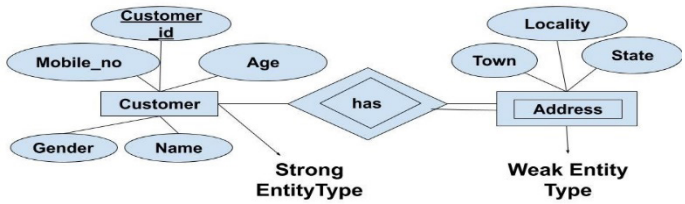


Figure 3.4. Flow chart of entity and set of entity.

Source: After academy, creative commons license.

3.3. DATABASE DESIGN IS A PART OF DATA MODELING

The abstract design process includes the data model as one component. The function model is another. In contrast to the function model, which focuses on how to handle the data and the data model emphasis on which data must be kept in the database. The relational tables in a relational database are intended with the data model, to put this into perspective. The queries that will contact those tables and carry out those activities are created using the functional model.

Planning and analysis come before data modeling (Câmara et al., 1996). The extent of the database determines how much effort is put into this stage. In comparison to a database designed to service the needs of a small workgroup, a database planned to aid the desires of an enterprise will need more planning and analysis.

During the requirements analysis, the data required to construct a data model is obtained. The requirements study and the ER diagramming stage of the data model are actually completed at the same time, even though they aren't formally included in the data modeling stage by some techniques.

3.3.1. Requirements Analysis

The objectives of the requirements analysis are:

- To identify the database's data necessities in the form of primeval objects (Bellatreche et al., 2006)

- To categorize and explain the data regarding these things.
- To define and organize the relations between the objects.
- To establish the kinds of transactions that will be carried out on the database and how the data and transactions will interact (Zhao & Roberts, 1988)
- To determine the laws controlling the accuracy of the data

To ascertain the data requirements for the database, the modelers or modelers, collaborates with the organization's end users. Many methods can be used to acquire the data needed for the requirements analysis:

- Review of already-existing documentation, such as memoranda, job descriptions, written policies and procedures, forms and reports, and personal narratives. A smart technique to get accustomed to the organization or activity you want to mimic is through paper documentation.
- Interviews with end users, which may combine one or both types of encounters. Strive to limit group meetings to no more than five or six persons. Try to hold a meeting where everyone who performs the same function is present. Take notes from the interviews using a blackboard, flip charts, or overhead transparencies.
- Examination of current automated systems – if the company currently has mechanized system, analysis the system enterprise documentation and requirements (Yu et al., 2000).

The requirements investigation is typically carried out along with data modeling. Once data is gathered, data objects are named, characterized using terminologies that are recognizable to end users, and categorized as entities, characteristics, or relationships. After that, an ER diagram is used to model and analyze the objects. The accuracy and completeness of the figure can be checked by both the modeler and the end operators. Incorrect models are changed, which occasionally necessitates the gathering of more data. Up till the model is declared to be accurate, evaluate and edit process is repeated.

During the requirements analysis, there are three things to keep in mind:

- Discuss data with end users in "real-world" terms. Users consider the authentic people, things, and activities they interact with on a regular basis rather than abstract

KEYWORD

A transaction is a completed agreement between a buyer and a seller to exchange goods, services, or financial assets in return for money.

concepts like entities, qualities, and relationships (Sibley & Kerschberg, 1977).

- Spend some time getting to know the organization and the operations you wish to imitate. It will be simpler to construct the model if you are aware of the procedures.
- Depending on their role within an organization, end users often approach and perceive data differently (Peckham & Maryanski, 1988). As a result, it's crucial to interview as many people as time will allow.

KEYWORD

Information gaps exist when either the buyer or seller does not have access to the information needed for them to make a fully-informed decision.

3.3.2. Phases in Building the Data Model

The steps for creating a data model are not standardized, despite the fact that the ER model lists and defines the necessary structures. Certain techniques, like IDEFIX, call for a bottom-up expansion approach somewhere the model is constructed incrementally. It is typical to model the entities and relations first, then the key characteristics, and finally the non-key attributes to complete the model (Worboys et al., 1990). Using a phased strategy, according to some experts, is unrealistic since it necessitates too many meetings with end users.

- Credentials of data items and relations
- Initial conscripting of the ER diagram containing entities and relations
- Improving the ER diagram (Kusiak et al., 1997)
- Including important attributes in the diagram
- Adding unimportant attributes
- Diagramming Generality Hierarchies
- Normalization serves to validate the model
- The Model's addition of business and integrity rules (Batra & Davis, 1992)

In reality, creating a model is not a strictly linear process. As was already said, the initial ER diagram draft and requirements analysis frequently happen at the same time. The diagram may need to be improved and validated in order to identify any issues or information gaps that call for additional data gathering and examination (Figure 3.5).

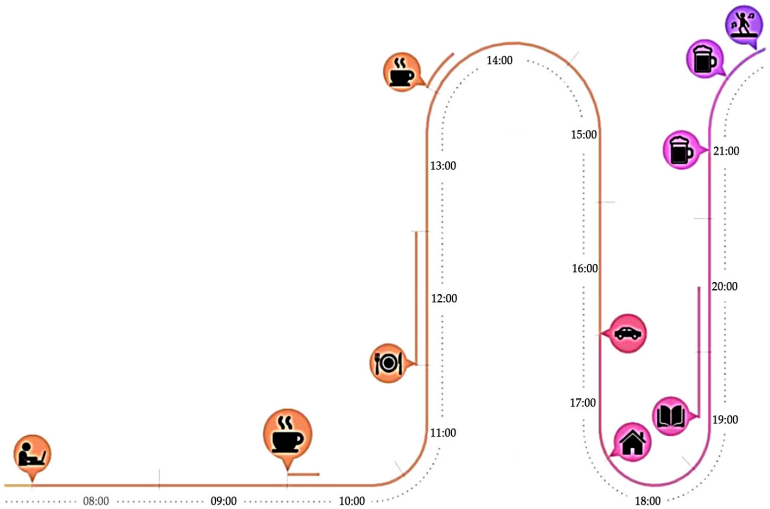


Figure 3.5. Example of linear process.

Source: Amcharts, creative commons license.

3.4. CLASSIFYING DATA OBJECTS AND RELATIONSHIPS

The modeler must evaluate the data obtained from the requirements analysis before starting to build the basic model in order to:

- Classifying data objects as entities or characteristics, for example
- Identification and clarification of links between entities
- defining and identifying the recognized entities, traits, and relationships
- recording this data in the data file (Fang et al., 2020)

The modeler must examine user narratives, meeting notes, plans and process documents, and if they're lucky, approach documents from the existing information system in order to achieve these aims.

The ER model's fundamental concepts are simple to define, but it might be challenging to determine how each plays a part in the construction of the data model (Bruns et al., 2008). What qualifies an item as an entity or an attribute? Take the statement that "workers effort on projects" as an example. Employees should they be considered an attribute or an entity? The needs of the database are frequently a determining factor in the right answer. Employee may be an attribute in some circumstances or an entity in others.

The ER Model has straightforward definitions for its structures, but it doesn't deal with the crucial problem of how to identify them. Here are a few suggestions that are frequently given:

Entities are made up of descriptive information, and characteristics either recognize or characterize them. Relations are relations between entities.

Below, we go into greater depth about these rules.

- Entities;
- Derived Attributes;
- Attributes;
- Code Values;
- Object Definition (Hashem et al, 2015);
- Naming Data Objects;
- Relationships; and
- Recording Data in Project Document.

KEYWORD

Descriptive Information allows users to locate information of potential interest, analyze that information, and order desired information.

3.4.1. Entities

An entity can be described in many different ways, such as "*any distinct place, thing, concept, or event about which info is conserved.*"

A distinguishing identifier for anything every distinct article that is to be exemplified in a database is referred to as *"any unique object that is to be characterized in a database"* (e.g., supplier, employee, machine tool, airline seat, utility pole etc.) (Sezer et al., 2017). Certain properties are kept for each entity type (Figure 3.6).

Common ideas concerning entities appear in these definitions:

1. A thing, notion, or item is an entity. The connections between two or more items can, however, occasionally be represented as entities. Associative entities are the name for this kind of thing (Bhavsar & Ganatra, 2012).
2. Entities are things that have a descriptive description. Your chosen data object is an entity if it can be termed by other objects. The item is not an entity if here is no accompanying description. Depending on the business or activity that is being modeled, a data object may be an entity or not.
3. Several things with similar properties are represented by an entity. They are not isolated entities. For instance, the

plays King Lear and Hamlet have similar traits in common, such as their names, authors, and casts of characters. King Lear and Hamlet are examples of the entity PLAY, which is the thing describing these things.

Figure 3.6. Picture of king Lear.

Source: Shakespeare, creative commons license.

4. Entities with shared characteristics are potential candidates for generalization hierarchies (See below)
5. It is improper to employ entities to discriminate between different time periods. One entity called Profits, for instance, should replace the entities 2nd Quarter Profits, 1st Quarter Profits, etc. (Nguyen & Rosen, 2017). To categorize by time, a time period attribute would be utilized.
6. Not all of the things the consumers want to gather data about will be entities. Several entities may be needed to represent a complex topic. Some "things" users consider significant may not actually be objects.

3.4.2. Attributes

Data objects known as attributes either describe or identify entities. Key qualities are those that allow for entity identification. The term "non-key attributes" refers to qualities that characterize an entity (Clarkson et al., 1998). In a later section, important characteristics will be covered in more detail. Similar steps can be taken to identify attributes, with the exception that this time you want to search for and excerpt names that seem to be noun expressions that describe something.

3.4.3. Validating Attributes

Values for attributes should only include one fact, or be atomic. Data that has been deaggregated makes programming simpler, increases

data reuse, and makes change implementation simpler. The "single fact" requirement must be fulfilled for normalization to occur. Typical transgressions include:

1. Straightforward concatenation, such as Person Tag, which combines the first name, central initial, and final name. The street address, city, and zip code are combined in another option called Address (Zhou et al., 2004). You must determine whether there are valid justifications for decomposing such attributes when dealing with them. For instance, do end users wish to utilize the subject's first name in a template letter? Would you like to kind by zip code?
2. Compound codes are characteristics whose tenets are codes made up of concatenated bits of data. The code found on cars and trucks is one instance (Chen et al., 2013). Over ten different facts regarding the car are represented by the code. These ciphers have no significance to the end operator unless they are a part of an industry standard. They are extremely challenging to practice and update.
3. Text blocks, which are open-ended text fields. Even if they have a place, relying too much on them could mean that the model isn't able to handle all the data.
4. Mixed domains, where a worth of an trait may have a distinct meaning in other contexts (Ma et al., 2018).

KEYWORD

An attribute is a quality or characteristic given to a person, group, or some other thing.

3.5. DERIVED ATTRIBUTES AND CODE VALUES

Experts in data modeling dispute on whether derived attributes and characteristics with code values should be allowed in the data model, to name two topics.

The characteristics that result from a summary or a formula process on other qualities are known as derived attributes. The foundation of arguments against derived data inclusion is that it shouldn't be kept in a database and, hence, shouldn't be a part of the data model (Sever et al., 2020). The supporting evidence is:

1. Resulting data is frequently crucial to users and managers, so it should be involved in the data model (Kononenko, 1995).
2. Documenting derived characteristics is equally as crucial as documenting other qualities, if not more so.

3. The data model's inclusion of derived attributes does not suggest how they will be used (Chang et al., 2011).

A fact is represented by one or more eruditions or integers in a coded value. For instance, the value Gender can utilize "M" and "F" instead of "Male" and "Female" as values. Opponents of this technique point out that codes complicate data processing and lack any clear significance for end users. Several businesses have long used coded attributes, according to proponents, who also claim that codes save space and improve flexibility by making it simple to add or change values using look-up tables.

3.5.1. Relationships

Relationships are linkages that connect different things. A verb connecting two or more elements typically denotes a relationship. Employees are given tasks to work on, for instance.

Once connections are found, they should be categorized according to cardinality, direction, dependence, optionality. The definition of the relations may lead to the removal of some relationships and the addition of others. The number of occurrences of one entity that are related to one instance of another is known as the cardinality, which quantifies the relationships between things (Van Winden et al., 2016). Think about the existence of a single instance of each entity to calculate the cardinality. Then, ascertain how many distinct examples of the second item may be connected to the first. Reverse the entities and run this analysis once more (Figure 3.7).

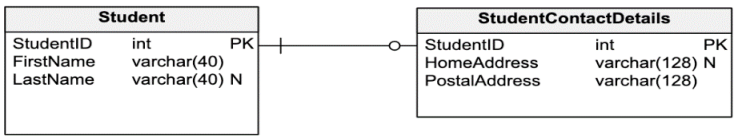


Figure 3.7. Illustration of cardinality.

Source: Leigh, creative commons license.

Consider this:

Each project has at least two employees allocated to it, and no employee may be dispensed to more than 3 projects at one time.

The cardinality of the association between the projects and the employees in this instance is two, but it is three for the relationship between the two. As a result, this relationship fits the definition of a many-to-many association.

Any affiliation that has a potential cardinality of nonentity is an elective one. The relationship is required if it essentially have a cardinality of at one. The conditional tense usually indicates relationships that are optional. An employee might be given a project, for instance. On the other side, words like must have suggest relationships that are mandatory.

For instance, a student needs to sign up for at least three classes per semester.

A parental object and a child entity exist in every instance of the particular relationship form (1:1 and 1:M). The parent in one-to-many connections is always the entity with cardinality one. The business being modeled must be taken into account when choosing the parent organization in one-to-one connections (Demšar, 2010). The selection is arbitrary if a result cannot be made.

Remember

The relationship resulting from affiliating one thing with another.

3.5.2. Naming Data Objects

These names ought to have the enlisted possessions:

- Unique
- Have significance to the enduser
- Cover the least digits of words required to exceptionally and precisely define the object (Klenosky & Perkins, 1992)

For attributes an entities, terms are single nouns though association names are normally the verbs.

Few of the authors guide against the use acronyms or abbreviations because they may led to misunderstanding about whatever they actually mean. Other rely on using acronyms or abbreviations are suitable providing that they are globally used and agreed within the institute.

You would also take caution to recognize and resolve substitutes for attributes and entities. This can take place in bulky projects where dissimilar departments use dissimilar terms for the similar thing.

3.5.3. Object Definition

Accurate and complete definitions are significant to make certain that all participants involved in the exhibiting of the data identify exactly what ideas the objects are demonstrating (Friedman &

Goldszmidt, 1996). Definitions must use terms acquainted to the user and must precisely describe what the object signifies and the character it is playing in the enterprise (Micci-Barreca, 2001). Few of the authors suggest having the end-users offer the definitions. If abbreviations, or terms not commonly understandable, are used in the description, then these have to be defined.

Although crucial objects, the modeler would be cautious to resolute any occurrences where a solitary entity is truly representing 2 distinct concepts (called homonyms) or where 2 distinct entities are essentially on behalf of the similar "thing" (called synonyms). This state of affairs usually arises because persons or administrations may consider about a process or an event in terms of their particular function.

An instance of a homonym will be a situation where the Marketing Section outlines the entity MARKETING in terms of topographical regions whereas the Sales Departments may think this entity in relations of demography. Unless determined, the outcome would be and unit with 2 unlike meanings and characteristics (Figure 3.8).

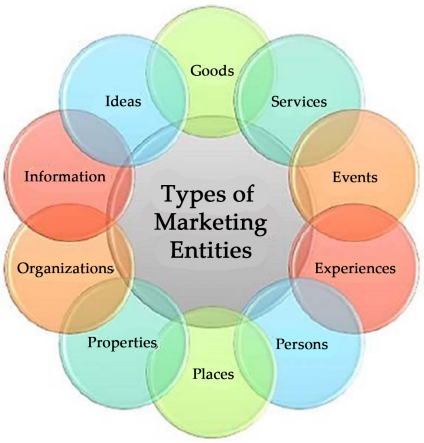


Figure 3.8. Illustration of types of marketing entities.

Source: Shridaddy, creative commons license.

On the contrary, an illustration of a synonym will be the Service Department might have acknowledged an entity entitled as CUSTOMER even though the Help Desk has acknowledged the entity CONTACT. In actual, they could mean the similar thing, an individual who calls or contacts the institute for aid with a badly behaved. The tenacity of synonyms is essential in order to evade idleness and to elude possible constancy or integrity hitches (Feng et al., 1988).

3.5.4. Recording Information in Designing Document

a. The policy file records thorough information roughly of each object utilized in the model (Kawakami & Kaneda, 2009). As per you define, describe, and name the objects, this evidence should be sited in this document. You are not utilizing an automatic design tool if, the document could be completed on paper or with a word processor system. Here is no standard for the association of this file, but the document must comprise material about definitions, terms, and for domains and attributes.

b. Two documents utilized in the IDEF1X technique of modeling are valuable for keeping trails of objects (Moudrý et al., 2019).

c. They are the ENTITY-ATTRIBUTE matrix, and the ENTITY-ENTITY matrix.

d. The ENTITY-ENTITY matrix is a 2D arrangement for representing relations between objects. The terms of all recognized entities are recorded alongside both axes (Menzies et al., 2006; Tan et al., 2010). As relations are first recognized, an "X" is positioned in the crossing points where each of the two axes encounter to specify a possible association among the objects involved. As the association is additionally categorized, the "X" is swapped with the notation representing cardinality.

e. The ENTITY-ATTRIBUTE matrix is utilized to point out the obligation of entities or characteristics (Sellitto et al., 2007). It is related in practice to the ENTITY-ENTITY matrix excluding attribute names are recorded on the rows (Figure 3.9).

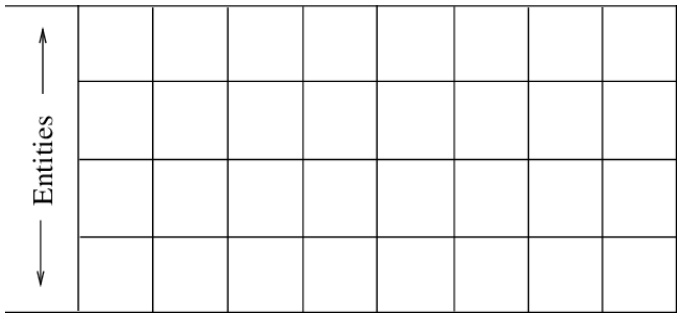


Figure 3.9. Example of entity attribute matrix.

Source: Aftab Hussian, creative commons license.

Table 3.1 displays models of an ENTITY-ENTITY matrix, and an ENTITY-ATTRIBUTE matrix.

Table 3.1. Tabular Representation of Models of an ENTITY-ENTITY Matrix, and an ENTITY-ATTRIBUTE Matrix

Employee	**An individual who work for as well as is salaried by the institute.**
Est_Time	The sum of hours a project director estimates that task will need to be completed. Predictable time is precarious for arranging a project and for following project time modifications
Assigned	Workers in the institute may be allocated to work on no extra than 3 schemes at a period. Each project will have as a minimum 2 workers allotted to it at any assumed time

Source: Wilson & Martinez, creative commons license.

Figure 3.10 displays models of an ENTITY-ENTITY matrix and an ENTITY-ATTRIBUTE matrix (Moser et al., 2008).

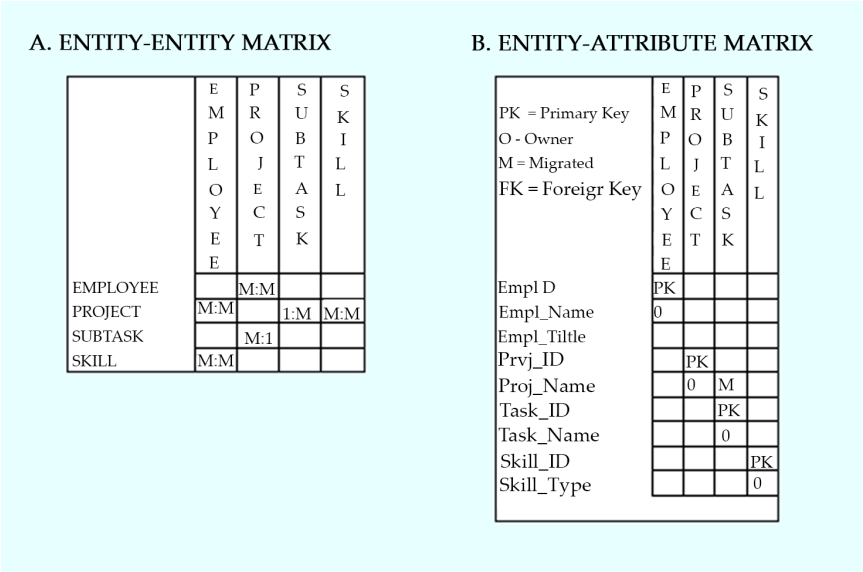


Figure 3.10. ENTITY-ENTITY matrix, and an, ENTITY-ATTRIBUTE matrix.

Source: Science direct, creative commons license.

3.6. DEVELOPING THE BASIC SCHEMA

Once entities and associations have been recognized and distinct, the primary draft of the object association diagram will be shaped. This unit presents the ER figure by validating how to figure binary associations (Nicolaos & Katerina, 2015). Recursive associations are also revealed.

3.6.1. Binary Relationships

Figure 3.11 displays samples of by what means to diagram one-to-one, many-to-many and one-to-many relationships (Sheth & Larson, 1990).

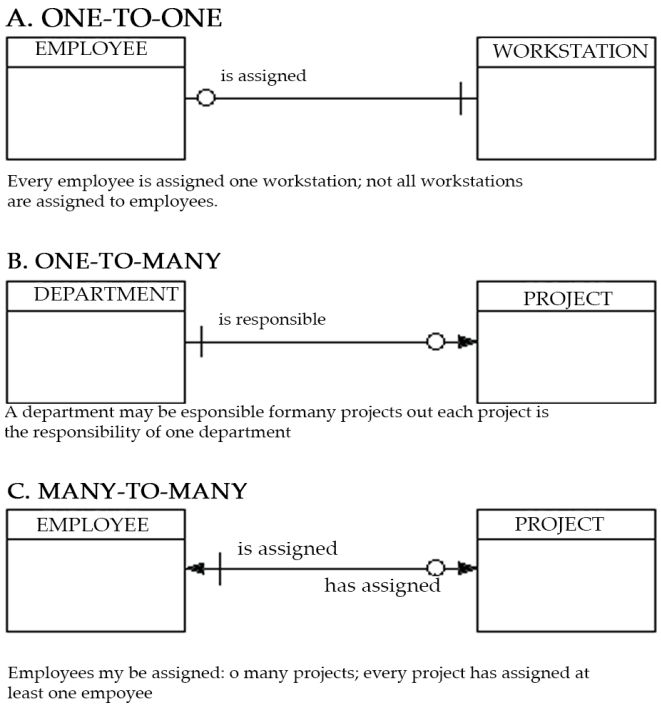


Figure 3.11. Schematic of binary relationship.

Source: Data base design, creative commons license.

3.6.2. One-To-One

Figure 3.11 Example of Binary Relationships Figure 3.11A demonstrate a case of a one-to-one figure (Baghdadi, 2006). Understanding the figure from left to rightward signify the association every worker is allotted a workstation. For the reason that every single employee must have a workstation, the sign for compulsory existence, in this situation the crossbar is positioned following to the WORKSTATION entity. Interpretating from right to left, the figure displays that not entire workstation are allotted to personnel. This state may reveal that more or less workstations are reserved for spares or for lends. Consequently, we custom the sign for noncompulsory existence, the circle, following to EMPLOYEE.

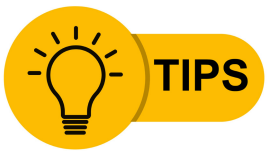
The cardinality and existence of a relationship must be derived from the “business rules” of the organization. For instance, if entire

workstations possessed by an institute were allotted to employees, at that time the circle would be substituted by a crossbar to signpost compulsory presence. One-to-one relations are infrequently perceived in "practical" data models (Han et al., 2011). Some PR actioners suggest that greatest one-to-one relations ought to be misshapen into a solitary entity or rehabilitated to a generality hierarchy.

3.6.3. One-To-Many

Figure 3.11 B expresses an illustration of a one-to-many association between PROJECT and DEPARTMENT (Burgin & Mikkilineni, 2021). In this diagram, a DEPARTMENT is deliberated the parent article though PROJECT is the child. Interpretation from leftward to right, the figure characterizes sections may be accountable for numerous projects. The optionality of the associations reveals the "business rule" that not every department in the institute will be answerable for handling projects. Reading from right to left, the diagram tells us that every project must be the responsibility of exactly one department.

3.6.4. Many-To-Many



Manage many-to-many data relations by implementing junction tables that efficiently link multiple data entities.

Figure 3.11C expresses a many-to-many association between PROJECT and EMPLOY (Brunette et al., 2013). A worker might be allotted to numerous projects; each project need many employee. Remind that the link among EMPLOYEE and PROJECT is voluntary as, at a certain time, and worker may not be allotted to a project. Conversely, the association between the PROJECT and EMPLOYEE is compulsory because a project requisite at least 2 employees allotted. Many-To-Many associations can be utilized in the preliminary recruiting of the model but ultimately must be converted into two one-to- numerous relationships (Soylu & De Causmaecker, 2009). The conversion is obligatory because many-to-many relations can't be signified by the interpersonal model. The procedure for determining many -to -many relationships is discoursed in the subsequent section.

3.6.5. Recursive Relations

A recursive link is an object is connected with this one. Figure 3.12 displays an sample of the recursive association (Wang & Ariguzo, 2004).

A worker may be able to manage numerous workers and each worker is managed by one worker.

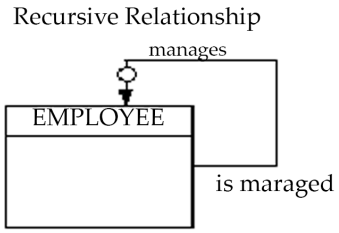


Figure 3.12. Diagram of instance of recursive association.

Employee may manage other employee. Each employee is managed by at most one employee

Source: Direct science, creative commons license.

3.7. REFINING – THE ENTITY-RELATIONSHIPS DIAGRAMS

This piece deliberates four rudimentary procedures for modeling interactions:

3.7.1. Entities Participation in Relationships

Entities can't be exhibited unconnected to one other entity (Jen-Yen & Yu-Shiang, 1994). Or else, while the model was converted to the interpersonal model, there will be noway to direct to that. table. The exclusion to this law is a database using a particular table.

3.7.2. Resolve Many-To-Many Relationships

Many-to-many interactions can't be utilized in the data. Model as they can't be exemplified by the interactive model. Consequently, many-to-many relations essentially be determined initially in the model development. The approach for determining many-to-many connection is to swap the relationship by an association object and then narrate the two innovative objects to the link entity. This policy is established below Figure 3.13 illustrate the many-to-many association (Campbell & Islam, 1992):

Employees might be assigned to numerous projects.

Each project need to allot it to more than an individual employee.

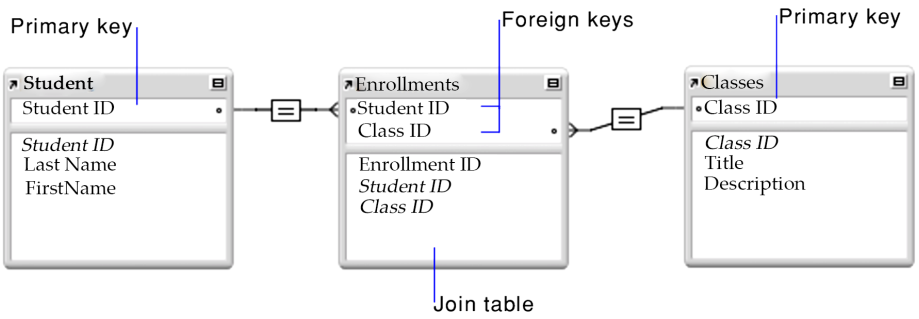


Figure 3.13. Illustration of many-to-many relationship.

Source: Filemaker, creative commons license.

In adding to the putting into practice problematic, this relationship offers other hitches. Presume we required to record info about member projects such as who allotted them, the starting date of the task, and the finishing date for the task. Given the current association, these features could not be epitomized in either PROJECT or EMPLOYEE deprived of repeating info. The first stage is to change the correlation given to a novel entity we will call an ASSIGNMENT. Formerly the unique entities, PROJECT and EMPLOYEE are connected to this new entity maintaining the optionality and cardinality of the original relations (Colomb & Dampney, 2005).

Notice that the plan deviate the semantics of the original relative to workers may be prearranged projects to tasks and assignments must be completed by more or one worker assignment.

A many to many recursive association is determined in alike fashion.

3.7.3. Transform Complex Relations into Binary Relationships

Complex relations are categorized as ternary; a relationship between 3 entities, or n-ray, an relationship between more than 3, where n is the figure of involved entities (Dragos et al., 2016). For instance, Figure 3.14A demonstrates the relationship.

Employees can use diverse abilities on any one or more projects. Each project use several employees with a number of skills. Complex associations can't be straightly applied in the interpersonal model so they must be determined initially in the exhibiting process. The policy for determining complex associations

is comparable to determining many-to-many relations. The difficult relationship swapped by an relationship entity and the novel entities are linked to this new object. Entity correlated over binary relations to apiece of the unique entities. (Sommestad et al., 2009).

3.7.4. Eliminate. Redundant. Relationships

A terminated connection is an association among two entities that is equal in sense to additional association among those similar two articles that may permit over an middle entity (Huang et al., 2017). For instance, Figure 3.14B displays the explanation which is to eliminate the terminated association DEPARTMENT allotted WORKSTATIONS.

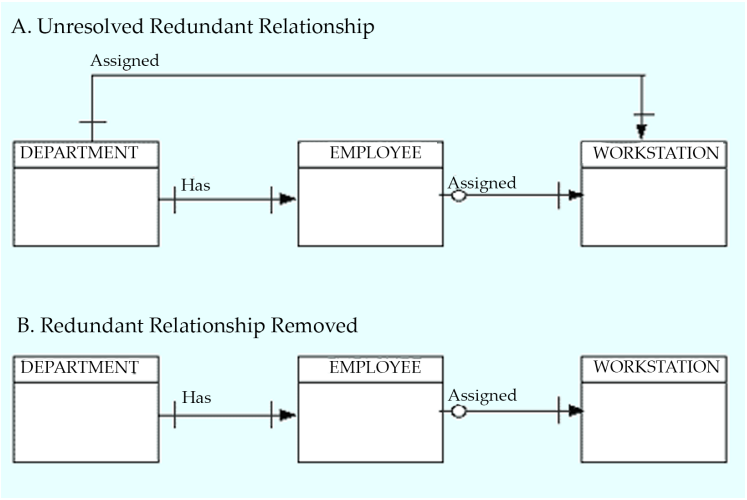


Figure 3.14. Schematic of removing redundant relationship.

Source: DW, creative commons license.

3.8. PRIMARY AND FOREIGN KEYS

Relational theory's two cornerstone components are primary and foreign keys. By permanently and unambiguously identifying individual instances of an entity, primary keys are a powerful tool for enforcing entity consistency. Relying on a foreign key to fully connect two entities is a surefire way to maintain their referential integrity. Step two in constructing the data model entails:

i. Determining which attributes of each entity will serve as "primary keys" and then defining those attributes.

ii. Check for correctness of primary keys and associations

iii. Foreign key establishment via migration of primary keys (Memari et al., 2015)

3.8.1. Primary Key Attributes

Entities can be described by their attributes, which are pieces of information. Simply put, an attribute example is a solitary value of a trait for a given case of an entity. An employee's name and start date are both examples of properties that can be applied to the entity EMPLOYEE. Illustrations of the qualities name and hiring date include "Jane Hathaway" and "3 March 1989," respectively.

Did you Know?

Primary keys uniquely identify a record in a table, while foreign keys establish relationships between records in different tables.

An entity's main key is the attribute (or group of attributes) that can be used to identify a single instance of that entity (Qian & Lunt, 1996). It is required that each object in the data ideal have a primary key, the values of which can be used to exclusively recognize each case of that entity.

Attributes must meet the following criteria in order to function as primary keys for entities:

1. First, it must always be non-null in all occurrences of the entity.
2. Second, each instances of an entity need to have a different value.
3. During the lifetime of an entity case, the values must remain constant and cannot be set to null (Markowitz & Makowsky, 1990).

Sometimes, an entity will have more than one characteristic that have action as the main key. A candidate key is a single key or a small usual of keys that could serve as the primary key. Once possible keys have been determined, pick a single main key that will be used consistently across all instances of that entity. Choose the identification that is most frequently used by the user provided it has the above-mentioned qualities. Unselected candidate keys are referred to as alternate keys.

Employee is a type of entity that exemplifies the possibility of many main keys (Zhuge, & Xing, 2005). Consider three potential keys for each worker at an organization: id, employee, name and social security number.

As qualifications go, "name" ranks towards the bottom. In a small team where everyone knows each other's names, this strategy would work, but it wouldn't be practical for a company

with hundreds or thousands of workers. In addition, an employee's name may change after getting married. Employee ID could work if it were used to identify each employee in a way that was both consistent and distinctive. Since all employers mandate a Social Security number, that's the most practical option.

3.8.2. Composite Keys

Several identifying characteristics may be necessary at times. A composite key. is a main key that contains of no more than 1 attribute. A sample composite key is shown in Table 3.2. A combination of the employee ID and the project ID is the sole way to identify a single case of the entity Work.

Table 3.2. Tabular Representation of Example of Composite Keys

Project .ID	Employee. ID	Hours Worked
01	01	200
02	01	120
01	02	50
03	02	120
03	03	100
04	03	200

Source: Haas, creative commons license.

3.8.3. Artificial Keys

Fake keys are those that have no significance for the company or organization. When either

1. No trait possesses all of the main key attributes.
2. The prime key is big and complex, artificial keys are allowed (Getoor & Grant, 2006).

3.8.4. Primary Key Migration

Dependent entities inherit the whole main key from the parent unit, meaning they depend on the presence of a new entity for their documentation (Premerlani & Blaha, 1993). The root generic entity's primary key is sent down to each entity in a generalization hierarchy.

3.8.5. Define Key Attributes

It is time to identify and describe the qualities that have been utilized as keys after the model's keys have been determined.

The representation of main keys in ER diagrams is not standardized. Inside the entity box for this file, the other name of the principal key is typed, trailed by the notation (PK) (Hunt, 2010).

Remember

A relational database must have only one primary key.

3.8.6. Validate Keys and Relationships

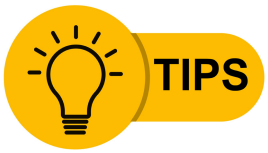
The following fundamental guidelines regulate the documents and relocation of main keys:

- Every object in the data model essentially have a main key, the ideals of which exclusively recognize instances of the entity.
- The primary key characteristic can't have an optional value (i.e., have null values).
- Repeating values are incompatible with the primary key (Levene & Loizou, 2012). That is, it is not permitted for an entity instance to have more than one value for an attribute at any given time. The No Repetition Rule is used to describe this.
- It is impossible to divide entities with multiple primary keys into several entities with more straightforward primary keys. The Minimum Key Rule applies here.
- Apart from entities contained within generalization hierarchies, two entities cannot have the same primary keys.
- The whole main key must be transferred between parent and child entities as well as between supertype subtypes, generic entities and category entities (Hamouda & Zainol, 2017).

3.8.7. Foreign Keys

By identifying the parent object, a foreign key is a characteristic that finalizes a relationship. Foreign keys offer a way to navigate between multiple examples of an entity and retain the referential integrity of the data (also known as foreign keys). There must be a foreign key supporting each relationship in the model.

3.8.8. Categorizing Foreign Keys



A table with a foreign key reference to itself is still limited to 253 foreign key references.

A foreign key is required for each connection in which a model's reliant on and class (subtype) entities take part (Waguespack & Waguespack, 2010). The whole chief key from the parental or generic object is migrated to dependent and subtype entities to create foreign keys. The primary key may not be divided if it is composite.

3.8.9. Foreign Key Ownership

Due to the fact that they are reflections of parent entity characteristics, foreign key attributes are not regarded as belonging to the entities to which they migrate. As a result, each feature in an object either belongs to that object or to a distant key in that object.

When all the properties in a foreign key are existing in the main key of a child entity, the child entity is said to be "identifier reliant" on the parent entity, and the relationship is referred to as a "identifying relationship." When any of the properties in a far-off key are not a part of the kid's main key, the connection is referred to as "non-identifying" and the kid is not identifier reliant on the parent.

3.8.10. Diagramming Foreign Keys

The symbol (FK) next to the characteristics of foreign keys designates them (Mäenpää & Wanne, 2015).

3.9. ADDING QUALITIES TO THE MODEL

Non-key characteristics give information about the entities they feel right. In this segment, we go over the guidelines for tagging entities with non-key properties and how to deal with multivalued attributes.

3.9.1. Relation of Attributes to Entities

There can only be one entity with non-key attributes. Non-key attributes, in contrast to key attributes, never pass from parent to child entities and only reside in one entity (Solomon & Brisini, 2019).

The first step in connecting attributes to entities is for the modeler to group characteristics with the entities that they seem to describe, with the help of the end users. Your choices should be noted in the entity characteristic matrix covered in the preceding section. The formal approach of normalization is then used to validate the assignments. The law is to include non-key characteristics in entities wherever the primary key's value influences the attributes' values before formal normalization is started. Generally speaking, entities that share a primary key ought to be consolidated into a single entity. The following list includes further recommendations for linking attributes to entities.

3.9.2. Parent-Child Relationships

1. When there is a parent-child relationship, characteristics should be added to the parent object whenever it makes sense to do so
2. Associate the parent and child entities if a parent entity lacks non-key attributes (Xu et al., 2014).

3.9.3. Multivalued Attributes

Categorize the characteristic as a new child entity if it depends on the chief key but is multivalued (more or one. value for a specific value of the key) (Ravald & Grönroos, 1996). The multivalued attribute becomes the primary key if it is exclusive to the new object. If not, shift the original, parent entity's primary key.

Consider, for instance, a PROJECT entity having the properties PROJ. ID (the key), Task. ID,PROJ Name and Task. Name.

PROJECT.

Proj_ID	Task_ID	Project Name	Task_Name
01	01	A	Analysis
01	02	A	Designing
01	03	A	Programming
01	04	A	Tuning
02	01	B	Analysis

Table 3.3. Tabular Representation of PROJECT entity

Source: Data base design, creative commons license.

Several values are available for the key property for Task ID and Task Name. The answer is to establish a novel entity—name lease's it TASK—and sort. it a subordinate of PROJECT. Transfer the fields Task ID, and Task Title. from PROJECT to the TASK. Migration of Proj ID to TASK would be the last step because neither characteristic can be used to uniquely identify a task.

3.9.4. Relations Described by Attributes

There are times when it seems like an attribute designates a relationship slightly than an actual object (Mattingly et al., 2014). A MEMBER might borrow books, for instance.

KEYWORD

Inclusion is the policy or practice of making sure that everyone in society has access to resources and opportunities.

The time the books were borrowed and their due date are potential qualities. A many-to-many relationship will typically result in such a problem, and the remedy is the similar. Re-classify the connection as a new entity that is a child of the two primary entities. The freshly produced thing is known as an associated entity in some techniques (Hung, 2007).

3.9.5. Code Values and Derived Attributes

The inclusion of derived characteristics and attributes with code values in the data model are two topics on which data modeling professionals can't agree.

A summary or a formula operation on additional attributes creates derived attributes, which are formed by such a process. Under the evidence that derived data shouldn't be kept in a database and, consequently, shouldn't be comprised in data model, influences against its inclusion are made. The justifications are as follows:

- The data model needs to incorporate resulting data because it is frequently crucial for managers and users alike.
- Documenting derived attributes should be treated in the same manner as other attributes, if not more so.
- The fact that derived attributes are included in the data model doesnot necessarily indicate how they will be used (Davis, 1985).

In a implicit value, a fact is represented by one or more letters or numbers. In place of "Male" and "Female," the values "M" and "F" can be used for the value "Gender," for instance. Codes don't

have a natural meaning to end users, according to many who object to this practice, and they complicate the data processing process. In support, proponents point out that numerous administrations have long used implicit properties, that coding save space, and that they increase flexibility by making it simple to add or change values using look-up tables.

3.9.6. Attributes in the ER Diagram

On whether characteristics belong in the ER diagram, there is dispute. Attributes must be added, according to the IDEF1X standard (Gambrel et al., 2016). However, many knowledgeable practitioners point out that including attributes, particularly if there are a lot of them, clogs the diagram and reduces its effectiveness in giving the enduser an outline of in what way the data is organized.

KEYWORD

A job title is a specific designation of a post within an organization, normally associated with a job description that details the tasks and responsibilities that go with it.

3.10. GENERALIZATION HIERARCHIES

Up until now, we have talked about labeling an object, or entity, through its communal traits, or attributes. An employee's name, employee id, job title, and skills setting are a few examples of how we might describe them.

Using both comparisons and modifications to describe entities is another strategy. Assume, for instance, that a company divides the projects it works on into internal and external ones. On behalf of a specific organizational unit, internal tasks are completed. For organizations outside the organization, external tasks are completed. We can see that both sorts of projects share a common characteristic in that they both require work to be completed by organization employees according to a set schedule. But, we also acknowledge that there are distinctions between them. A client identity and the amount the customer is charged are two distinctive characteristics of external projects (Reiter, 1989). Generalization is the process of classifying entities according to their shared characteristics and distinctive characteristics.

3.10.1. Description

A generality hierarchy is a logical arrangement of items with related characteristics. It is a potent and popular technique for conveying shared traits among things while maintaining their distinctions. It is the association of an entity with one or more abridged forms.

Each refined form is referred to as a subtype, while the entity being advanced is referred to as the supertype. When:

1. A lot of things seem to be of the same type and generalization hierarchies should be utilized.
2. When attributes are replicated across several entities.
3. When the model is dynamic (McCarthy, 1982).

Detailed hierarchies reduce the amount of entities in the model, which simplifies it and increases model stability by allowing modifications to only be made to objects relevant to the change.

3.10.2. Making a Generalization Hierarchy

Remember

The discriminator of a weak entity set is a set of attributes that differentiates it.

The supertype is given all common attributes in order to build a generalization hierarchy. A discriminator attribute, whose values specify the subtype categories, is also given to the supertype. A subtype is given the attributes that are specific to that category. The super type's primary key is also passed down to each subtype. Subtypes with a single primary key ought to be dropped. In a one-to-one relationship, subtypes and super types are related.

3.10.3. Types of Hierarchies

Both overlapping and disjoint generalization hierarchies are possible. An entity instance may belong to more than one subtype in a hierarchy that overlaps. The super type entity. PERSON, which. has the 3 subtypes STAFF, STUDENT and FACULTY, is one example of how you may symbolize people at a university. An individual could very well fall under more than one category, such as a staff employee who is simultaneously enrolled as a student. A single subtype can contain all instances of an entity in a disjoint hierarchy. Examples include the subtypes CLASSIFIED and WAGES for the entity EMPLOYEE. An employee may fall under either category, but not both (Ehrlich, 2001).

3.10.4. Rules

Generalization hierarchies' main tenet is that any instance of an entity belonging to the supertype must also occur in at least one instance of the subtype, and vice versa. There can only be one generalization hierarchy that includes subtypes. In other words, a subtype can only be associated to one supertype. The

supertype for one hierarchy might be the subtype for another, allowing generalization hierarchies to be layered.

Subtypes are not always the children in relationships; they can be the parent entity (Michalski, 1983). The subtype would receive two primary keys if this were permitted.

3.11. ADDING DATA INTEGRITY RULES

Any of the pillars of the interactive pattern is data reliability. Simply said, data integrity denotes to the constancy and correctness of the data standards in the data-base.

Entity and the referential. integrity rules in the interpersonal model are used to enforce data integrity (Zhang et al., 2019). Although not a component of the relational architecture, the majority of database software uses domain information to guarantee attribute integrity.

KEYWORD

Data integrity is the overall accuracy, completeness, and consistency of data.

3.11.1. Entity Integrity

The chief key value for each example of an object must be existing, dissimilar, and couldntt be null, according to the entity integrity rule (Bertossi & Milani, 2018). The primary. key purpose of exclusively identifying every instance of an entity would be compromised in the absence of entity integrity.

3.11.2. Referential Integrity

All distant key value need correspond to a main key values in a related table, according to the referential veracity rule. We can transfer between linked things with correctness thanks to referential integrity.

3.11.3. Inserting and Deleting Rules

Between two related entities, a foreign key establishes a hierarchical relationship (Becker et al., 2008). The crucial key table from which the distant key values are derived is the parental, and the entity comprising the distant key is the child, or reliant.

Some insert and delete rules must be taken into account when adding or removing data from the database in directive to ensure referential integrity among the parent and child.

3.11.4. Insert Rules

The following insert rules are frequently used:

1. **Dependent:** Only if the parent entity that equals the child entity instance can be inserted according to the dependent insert rule (Wang et al., 2006).
2. **Automatic:** A child entity instance may always be inserted under the automatic insert rule. If no instance of the matching parent object already exists, one is generated.
3. **Nullify:** The addition of a child entity case is always allowable by the nullify insert rule. The far-off key in the teen-ager entity case is set to null if a corresponding parent entity case is not present.
4. **Default:** Child entity instances may always be inserted according to the default insert rule. If there isn't a parent entity instance that matches, the foreign key in the child is agreed to the value that was originally specified.
5. **Customized:** Only under certain customized validity conditions is it possible to insert a child entity instance according to the custom-made insert rule (Thion & Coulondre, 2012).
6. **No Impact:** The introduction of a child entity case is always allowed, according to this rule. There is no requirement that a parent entity instance match, hence there is no validity check.

KEYWORD

A validity check is the process of ensuring that a concept or construct is acceptable in the context of the process or system that it is to be used in.

3.11.5. Delete Rules

The following insert rules are frequently used:

1. **Restrict:** Only if there are no identical child entity cases is it possible to delete the parent entity instance under the restrict delete rule.
2. **Cascade:** A parent entity case may always be deleted, and the cascade delete rule also removes all cases in the child entity that match.
3. **Nullify:** The deletion of a parent entity case is always permitted under the nullify delete rules (Berti-Equille, 2007). The standards of the foreign keys in any instances of matching child entities that exist are set to null.
4. **Default:** A parent entity case may never be deleted under

the default rule. The worth of the distant keys is fixe to a predetermined default value if there are any instances of matching child entities.

5. **Customized:** A parent entity case may only be deleted using the customized delete rule if certain validity requirements are satisfied (Hernandez, 2013).
6. **No Impact:** A parent entity case may never be deleted while using the no impact delete rule. There is no validity checking.

KEYWORD

Regulation is the management of complex systems according to a set of rules and trends.

3.11.6. Insert and Delete Guidelines

The decision of which regulation to apply is made by these are some fundamental recommendations for delete and insert rules.

1. Steer clear of rules that negate inserts or deletions. In a parent-child connection, the parent entity often has an obligatory existence. This rule would be broken if the null insert or delete rule was used.
2. For generalization hierarchies, use either the automated or dependent insert rule. The requirement that all instances in subtypes must also be in super types will only be preserved by these rules.
3. When working with generalization hierarchies, use the cascade delete rule. The restriction that only instances of the supertype may exist in subtypes will be enforced by this rule (Caroprese et al., 2008).

3.11.7. Domains

A legitimate collection of values for a characteristic is referred to as a domain, and it ensures that values from inserts or updates make sense. The following domain information should be assigned to each characteristic in the model:

1. Data Type—Integer, decimal, and character data types are the most basic data types. The majority of data bases allow variations of these in addition to unique date and time data types (Reiter, 1989).
2. Length—the length of the value is its digit or character count. a 5 digit or 40 character value, for instance.
3. Date Format—the set-up for dates using the notation dd/ mm /yy or yy/ mm /dd (Christiansen & Martinenghi, 2006).

4. Range – The array describes the lowest and upper limits of the values that an attribute may have legally.
5. Constrictions—are particular limitations on acceptable values. For instance, a new employee's Beginning Pay Date must always be the first workday of the month in which they are hired.
6. Null support—States if the attribute is compatible with null values.
7. Default value (if any): If no value is given, this value will be applied to an attribute instance (Kufoniyi, 1995).

Remember

An array is stored such that the position of each element can be computed from its index tuple by a mathematical formula.

3.11.8. Primary Key Domains

Most important key values cannot be null and essentially be distinct.

3.11.9. Foreign Key Domains

Primary keys must have the same data type, length, and format as the matching primary key (Bertossi & Rizzolo, 2016). The relationship type and the individuality property must match. An exclusive foreign key is implied by a one-to-one link, whereas an unexclusive foreign key is implied by a one-to-many link.

3.12. OUTLINE OF THE RELATIONAL MODEL

Dr. E. F. Codd officially proposed the relational model in 1970, and it has subsequently developed through a number of articles. The relational model offers a straightforward yet precisely defined understanding of how consumers view data. Two-dimensional tables are used in the relational paradigm to represent data. Each table characterizes a practical individual, location, object, or event that is the subject of data collection. Two-dimensional tables make up a relational database. The logical perspective of the database refers to how the data is arranged into relational tables (Van Zomeren, 2015). Specifically, this refers to the way a relational data-base displays data to the operator and the computer operator. The internal view refers to how the database program physically saves the information on a computer disc drive. The internal view varies from creation to creation and is not relevant to this discussion.

To use relational database software that is based on the relational model, such as Oracle, and Microsoft. SQL Server, or

even your own personal database systems like Access or Fox, successfully, one must have a fundamental understanding of the relational model.

This document serves as a loose introduction to relational principles, particularly as they apply to problems with relational database architecture. Relational theory is not entirely described in this way.

The fundamental ideas—data assemblies, relations, and data integrity—that form the foundation of the relational model are covered in this section.

1. Terminology and Data Structure
2. Relational Table Properties
3. Notation
4. Partnerships and Keys
5. Data Integrity
6. Relational Data Manipulation
7. Stabilization
8. Advancement of Stabilization (Sibley, 2007)

ACTIVITY 3.1.

You are tasked with designing a database system for a large hospital network that will store patient information, medical records, and billing data. Describe how you would use database modeling techniques, such as entity-relationship diagrams, to design an efficient and scalable database structure.

SUMMARY

Database modeling, or the act of developing a conceptual model of a database, is explored in length in the chapter. It starts with a brief introduction to data modeling and its significance in developing a database structure. The entity relationship model, which depicts connections between data objects, is next presented in this chapter. Data object and relationship discovery, schema construction, and ERD refinement are also discussed in this chapter. Later, the chapter dives into the value of primary and foreign keys in a database. The definition of generalization hierarchies and the addition of characteristics to the entity relationship model are also covered. Finally, a brief introduction to relational models and data integrity rules rounds off this chapter.

REVIEW QUESTIONS

1. What is the entity relationship model and how is it used in database modeling?
2. How do you identify data objects. and relationships in the entity relationship model?
3. What is the role of primary and foreign keys in a database schema?
4. How do you refine the entity relationship diagram?
5. What are generalization hierarchies and how are they used in database modeling?
6. What are data integrity rules and why are they important in a database schema?
7. What is the relational model and how does it relate to the entity relationship model?

MULTIPLE CHOICE QUESTION

1. Which of the following best defines data modeling?

a. The process of creating a visual representation of data objects and their relationships
b. The method of creating a physical representation of a database
c. The process of creating a logical representation of a database schema
d. The procedure of creating a report based on the database

2. What is the purpose of the entity-relationship model?

a. To provide a physical representation of a database
b. To provide a logical representation of a database schema
c. To define data integrity rules
d. To provide a report based on database data

3. What is the role of data modeling in database design?

a. To create a physical representation of a database

b. To create a logical representation of a database schema
c. To define data integrity rules
d. To create a report based on database data

4. What is the primary key in a database schema?
a. A field that uniquely recognizes each record in a table
b. A field that links records from one table to records in an additional table
c. A field that defines data integrity rules
d. A field that is used to create a report based on database data

5. What is the purpose of foreign keys in a database schema?
a. To uniquely identify each record in a table
b. To link records from one table to records in another table
c. To define data integrity rules
d. To create a report based on database data

6. What is the process of refining the entity relationship diagram?
a. Adding detail and clarity
b. Removing unnecessary attributes
c. Creating a physical representation of the database
d. Defining data integrity rules

7. What are generalization hierarchies?
a. A way to simplify complex data relationships
b. A way to define data integrity rules
c. A way to create a physical representation of the database
d. A way to create a report based on database data

8. What is the relational model?
a. A model that describes the relationships between data objects in a database
b. A physical representation of a database
c. A report based on database data
d. A set of data integrity rules

Answer to Multiple Choice Questions

1. (a); 2. (b); 3. (b); 4. (a); 5. (b); 6. (a); 7. (a); 8. (a)

REFERENCES

1. Angles, R., & Gutierrez, C., (2008). Survey of graph database models. *ACM Computing Surveys (CSUR), 40*(1), 1–39.
2. Aracic, I., Gasiunas, V., Mezini, M., & Ostermann, K., (2006). An overview of CaesarJ. *Transactions on Aspect-Oriented Software Development I, 1*, 135–173.
3. Baghdadi, Y., (2006). Reverse engineering relational databases to identify and specify basic web services with respect to service oriented computing. *Information Systems Frontiers, 8*(1), 395–410.
4. Batini, C., & Lenzerini, M., (1984). A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering, 4*(6), 650–664.
5. Batra, D., & Davis, J. G., (1992). Conceptual data modelling in database design: Similarities and differences between expert and novice designers. *International Journal of Man-machine Studies, 37*(1), 83–101.
6. Becker, J., Matzner, M., Müller, O., & Winkelmann, A., (2008). Towards a semantic data quality management-using ontologies to assess master data quality in retailing. *AMCIS 2008 Proceedings, 129*(1), 4–9.
7. Bellatreche, L., Dung, N. X., Pierra, G., & Hondjack, D., (2006). Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases. *Computers in Industry, 57*(8, 9), 711–724.
8. Berti-Equille, L., (2007). Measuring and modelling data quality for quality-awareness in data mining. *Quality Measures in Data Mining, 2*(1) 101–126.
9. Bertossi, L., & Milani, M., (2018). Ontological multidimensional data models and contextual data quality. *Journal of Data and Information Quality (JDIQ), 9*(3), 1–36.
10. Bertossi, L., & Rizzolo, F., (2016). *Contexts and Data Quality Assessment, 1*, 3–6.
11. Bhavsar, H., & Ganatra, A., (2012). A comparative study of training algorithms for supervised machine learning. *International Journal of Soft Computing and Engineering (IJSCE), 2*(4), 2231–2307.
12. Brunette, W., Sundt, M., Dell, N., Chaudhri, R., Breit, N., & Borriello, G., (2013). Open data kit 2.0: Expanding and refining information services for developing regions. In: *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications* (Vol. 1, pp. 1–6).
13. Bruns, E., Brombach, B., & Bimber, O., (2008). Mobile phone-enabled museum guidance with adaptive classification. *IEEE Computer Graphics and Applications, 28*(4), 98–102.
14. Burgin, M., & Mikkilineni, R., (2021). From data processing to knowledge processing: Working with operational schemas by autopoietic machines. *Big Data and Cognitive Computing, 5*(1), 13.
15. Câmara, G., Souza, R. C. M., Freitas, U. M., & Garrido, J., (1996). SPRING: Integrating remote sensing and GIS by object-oriented data modeling. *Computers*

& Graphics, 20(3), 395–403.

16. Campbell, R. H., & Islam, N., (1992). A technique for documenting the framework of an object-oriented system. In: *[1992] Proceedings of the Second International Workshop on Object Orientation in Operating Systems* (Vol. 1, pp. 288–300). IEEE.
17. Caroprese, L., Greco, S., & Zumpano, E., (2008). Active integrity constraints for database consistency maintenance. *IEEE Transactions on Knowledge and Data Engineering, 21*(7), 1042–1058.
18. Chang, J. R., Jheng, Y. H., Lo, C. H., & Chang, B., (2011). Attribute coding for the rough set theory based rule simplications by using the particle swarm optimization algorithm. In: *Intelligent Decision Technologies: Proceedings of the 3rd International Conference on Intelligent Decision Technologies (IDT'2011)* (Vol. 1, pp. 399–407). Springer Berlin Heidelberg.
19. Chen, P. P. S., (1976). The entity-relationship model—Toward a unified view of data. *ACM Transactions on Database Systems (TODS), 1*(1), 9–36.
20. Chen, P. P. S., (1977). The entity-relationship model: A basis for the enterprise view of data. In: *Proceedings of the June 13–16, 1977, National Computer Conference* (Vol. 1, pp. 77–84).
21. Chen, X., Shrivastava, A., & Gupta, A., (2013). Neil: Extracting visual knowledge from web data. In: *Proceedings of the IEEE International Conference on Computer Vision* (Vol. 1, pp. 1409–1416).
22. Christiansen, H., & Martinenghi, D., (2006). On simplification of database integrity constraints. *Fundamenta Informaticae, 71*(4), 371–417.
23. Clarkson, B., Sawhney, N., & Pentland, A., (1998). Auditory context awareness via wearable computing. *Energy, 400*(600), 20.
24. Colomb, R. M., & Dampney, C. N., (2005). An approach to ontology for institutional facts in the semantic web. *Information and Software Technology, 47*(12), 775–783.
25. Davis, F. D., (1985). *A Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory and Results* (Vol. 1, pp. 2–4). Doctoral dissertation, Massachusetts Institute of Technology.
26. Demšar, J., (2010). Algorithms for subsetting attribute values with relief. *Machine Learning, 78*(1), 421–428.
27. Deville, Y., Gilbert, D., Van, H. J., & Wodak, S. J., (2003). An overview of data models for the analysis of biochemical pathways. *Briefings in Bioinformatics, 4*(3), 246–259.
28. Dragos, V., Gatepaille, S., & Lerouvreur, X., (2016). Refining relation identification by combining soft and sensor data. In: *2016 19th International Conference on Information Fusion (FUSION)* (Vol. 1, pp. 2139–2146). IEEE.
29. Ehrlich, P., (2001). Number systems with simplicity hierarchies: A generalization of Conway's theory of surreal numbers. *The Journal of Symbolic Logic, 66*(3), 1231–1258.
30. Fang, W., Ma, L., Love, P. E., Luo, H., Ding, L., & Zhou, A. O., (2020). Knowledge graph for identifying hazards on construction sites: Integrating computer vision with

ontology. *Automation in Construction, 119*(1), 103310.

31. Feng, A., Sugiyama, Y., Fujii, M., & Torii, K., (1988). An attribute grammar with common attributes and its evaluator in prolog. *Systems and Computers in Japan, 19*(6), 97–107.
32. Friedman, N., & Goldszmidt, M., (1996). Discretizing continuous attributes while learning Bayesian networks. In: *ICML* (Vol. 1, pp. 157–165).
33. Gambrel, L. E., Faas, C., Kaestle, C. E., & Savla, J., (2016). Interpersonal neurobiology and couple relationship quality: A longitudinal model. *Contemporary Family Therapy, 38*(1), 272–283.
34. Getoor, L., & Grant, J., (2006). PRL: A probabilistic relational language. *Machine Learning, 62*(1), 7–31.
35. Gregersen, H., & Jensen, C. S., (1999). Temporal entity-relationship models: A survey. *IEEE Transactions on Knowledge and Data Engineering, 11*(3), 464–497.
36. Haas, L., (2006). Beauty and the beast: The theory and practice of information integration. In: *Database Theory–ICDT 2007: 11th International Conference, Barcelona, Spain, January 10–12, 2007. Proceedings 11* (Vol. 1, pp. 28–43). Springer Berlin Heidelberg.
37. Hamouda, S., & Zainol, Z., (2017). Document-oriented data schema for relational database migration to NoSQL. In: *2017 International Conference on Big Data Innovations and Applications (Innovate-Data)* (Vol. 1, pp. 43–50). IEEE.
38. Han, J., Haihong, E., Le, G., & Du, J., (2011). Survey on NoSQL database. In: *2011 6th International Conference on Pervasive Computing and Applications* (Vol. 1, pp. 363–366). IEEE.
39. Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U., (2015). The rise of "big data" on cloud computing: Review and open research issues. *Information Systems, 47*, 98–115.
40. Hernandez, M. J., (2013). *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design* (Vol. 1, pp. 2–5). Pearson Education.
41. Huang, J., Zhang, W., Zhao, S., Ding, S., & Wang, H., (2017). Learning to explain entity relationships by pairwise ranking with convolutional neural networks. In: *IJCAI* (Vol. 1, pp. 4018–4025).
42. Hull, R., & King, R., (1987). Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys (CSUR), 19*(3), 201–260.
43. Hung, C. J. F., (2007). Toward the theory of relationship management in public relations: How to cultivate quality relationships. *The Future of Excellence in Public Relations and Communication Management: Challenges for the Next Generation* (Vol. 1, pp. 443–476).
44. Hunt, T. D., (2010). Natural or artificial primary key? Using the Mifrenz childrens email application as a case study. *New Zealand Journal of Applied Computing & Information Technology, 14*(1), 4–9.

45. Jen-Yen, C., & Yu-Shiang, H., (1994). An integrated object-oriented analysis and design method emphasizing entity/class relationship and operation finding. *Journal of Systems and Software, 24*(1), 31–47.

46. Kawakami, T., & Kaneda, S., (2009). *A Real-Time Computing Approach for Derived Attribute Values in Object-Oriented Application Design* (Vol. 109, No. 196, pp. 25–28). IEICE Technical Report; IEICE Tech. Rep.

47. Kerre, E. E., & Chen, G., (1995). An overview of fuzzy data models. *Fuzziness in Database Management Systems, 1*, 23–41.

48. Klenosky, D. B., & Perkins, W. S., (1992). Deriving attribute utilities from consideration sets: An alternative to self-explicated utilities. *ACR North American Advances, 1*, 3–7.

49. Kononenko, I., (1995). On biases in estimating multi-valued attributes. In: *IJCAI* (Vol. 95, pp. 1034–1040).

50. Kufoniyi, O., (1995). *Spatial Coincidence Modeling, Automated Database Updating and Data Consistency in Vector GIS* (Vol. 1, pp. 1–5). Wageningen University and Research.

51. Kusiak, A., Letsche, T., & Zakarian, A., (1997). Data modeling with IDEF1x. *International Journal of Computer Integrated Manufacturing, 10*(6), 470–486.

52. Levene, M., & Loizou, G., (2012). *A Guided Tour of Relational Databases and Beyond* (Vol. 2, No. 1, pp. 6–10). Springer Science & Business Media.

53. Ma, L., Sacks, R., Kattel, U., & Bloch, T., (2018). 3D object classification using geometric features and pairwise relationships. *Computer‐Aided Civil and Infrastructure Engineering, 33*(2), 152–164.

54. Ma, Z. M., & Yan, L., (2010). A literature overview of fuzzy conceptual data modeling. *J. Inf. Sci. Eng., 26*(2), 427–441.

55. Ma, Z. M., (2007). A literature overview of fuzzy database modeling. *Intelligent Databases: Technologies and Applications, 1*, 167–196.

56. Mäenpää, T., & Wanne, M., (2015). Review of similarities between adjacency model and relational model. In: *Modeling, Computation and Optimization in Information Systems and Management Sciences: Proceedings of the 3rd International Conference on Modeling, Computation and Optimization in Information Systems and Management Sciences-MCO 2015-Part II* (Vol. 1, pp. 69–79). Springer International Publishing.

57. Markowitz, V. M., & Makowsky, J. A., (1990). Identifying extended entity-relationship object structures in relational schemas. *IEEE Transactions on Software Engineering, 16*(8), 777–790.

58. Mattingly, B. A., Lewandowski, Jr. G. W., & McIntyre, K. P., (2014). "You make me a better/worse person": A two‐dimensional model of relationship self‐change. *Personal Relationships, 21*(1), 176–190.

59. McCarthy, W. E., (1982). The REA accounting model: A generalized framework for accounting systems in a shared data environment. *Accounting Review, 1*, 554–578.

60. Memari, M., Link, S., & Dobbie, G., (2015). SQL data profiling of foreign keys. In:

Conceptual Modeling: 34th International Conference, ER 2015, Stockholm, Sweden, October 19–22, 2015, Proceedings 34 (Vol. 1, pp. 229–243). Springer International Publishing.

61. Menzies, T., Greenwald, J., & Frank, A., (2006). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(1), 2–13.
62. Micci-Barreca, D., (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter, 3*(1), 27–32.
63. Michalski, R. S., (1983). A theory and methodology of inductive learning. In: *Machine Learning* (Vol. 1, pp. 83–134). Morgan Kaufmann.
64. Modolo, R., Hess, S., Génot, V., Leclercq, L., Leblanc, F., Chaufray, J. Y., & Holmström, M., (2018). The LatHyS database for planetary plasma environment investigations: Overview and a case study of data/model comparisons. *Planetary and Space Science, 150*(1), 13–21.
65. Moody, D. L., & Shanks, G. G., (2005). What makes a good data model? Evaluating the quality of entity relationship models. In: *Entity-Relationship Approach—ER'94 Business Modelling and Re-Engineering: 13th International Conference on the Entity-Relationship Approach Manchester, United Kingdom, December 13–16, 1994 Proceedings* (Vol. 1, pp. 94–111). Berlin, Heidelberg: Springer Berlin Heidelberg.
66. Moser, R., Pedrycz, W., & Succi, G., (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: *Proceedings of the 30th International Conference on Software Engineering* (Vol. 1, pp. 181–190).
67. Moudrý, V., Lecours, V., Malavasi, M., Misiuk, B., Gábor, L., Gdulová, K., & Wild, J., (2019). Potential pitfalls in rescaling digital terrain model-derived attributes for ecological studies. *Ecological Informatics, 54*(1), 100987.
68. Nguyen, H., & Rosen, P., (2017). DSPCP: A data scalable approach for identifying relationships in parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics, 24*(3), 1301–1315.
69. Nicolaos, P., & Katerina, T., (2015). Simple-talking database development: Let the end-user design a relational schema by using simple words. *Computers in Human Behavior, 48*(1), 273–289.
70. Peckham, J., & Maryanski, F., (1988). Semantic data models. *ACM Computing Surveys (CSUR), 20*(3), 153–189.
71. Petkovic, M., & Jonker, W., (2000). An overview of data models and query languages for content-based video retrieval. In: *International Conference on Advances in Infrastructure for E-Business, Science, and Education on the Internet* (Vol. 1, pp. 2–4).
72. Premerlani, W. J., & Blaha, M. R., (1993). An approach for reverse engineering of relational databases. In: *[1993] Proceedings Working Conference on Reverse Engineering* (Vol. 1, pp. 151–160). IEEE.
73. Qian, X., & Lunt, T. F., (1996). A MAC policy framework for multilevel relational

databases. *IEEE Transactions on Knowledge and Data Engineering, 8*(1), 3–15.

74. Ravald, A., & Grönroos, C., (1996). The value concept and relationship marketing. *European Journal of Marketing, 30*(2), 19–30.
75. Reiter, R., (1989). Towards a logical reconstruction of relational database theory. In: *Readings in Artificial Intelligence and Databases* (Vol. 1, pp. 301–327). Morgan Kaufmann.
76. Sellitto, C., Burgess, S., & Hawking, P., (2007). Information quality attributes associated with RFID-derived benefits in the retail supply chain. *International Journal of Retail & Distribution Management, 35*(1), 69–87.
77. Sever, I., Verbič, M., & Klaric, S. E., (2020). Estimating attribute-specific willingness-to-pay values from a health care contingent valuation study: A best–worst choice approach. *Applied Health Economics and Health Policy, 18*(1), 97–107.
78. Sezer, O. B., Dogdu, E., & Ozbayoglu, A. M., (2017). Context-aware computing, learning, and big data in internet of things: A survey. *IEEE Internet of Things Journal, 5*(1), 1–27.
79. Sheth, A. P., & Larson, J. A., (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys (CSUR), 22*(3), 183–236.
80. Sibley, C. G., (2007). The association between working models of attachment and personality: Toward an integrative framework operationalizing global relational models. *Journal of Research in Personality, 41*(1), 90–109.
81. Sibley, E. H., & Kerschberg, L., (1977). Data architecture and data model considerations. In: *Proceedings of the June 13–16, 1977, National Computer Conference* (Vol. 1, pp. 85–96).
82. Solomon, D. H., & Brisini, K. S. C., (2019). Relational uncertainty and interdependence processes in marriage: A test of relational turbulence theory. *Journal of Social and Personal Relationships, 36*(8), 2416–2436.
83. Sommestad, T., Ekstedt, M., & Johnson, P., (2009). Cyber security risks assessment with Bayesian defense graphs and architectural models. In: *2009 42nd Hawaii International Conference on System Sciences* (Vol. 1, pp. 1–10). IEEE.
84. Soylu, A., & De Causmaecker, P., (2009). Merging model driven and ontology driven system development approaches pervasive computing perspective. In: *2009 24th International Symposium on Computer and Information Sciences* (Vol. 1, pp. 730–735). IEEE.
85. Tan, X., Hammad, A., & Fazio, P., (2010). Automated code compliance checking for building envelope design. *Journal of Computing in Civil Engineering, 24*(2), 203–211.
86. Teorey, T. J., Lightstone, S. S., Nadeau, T., & Jagadish, H. V., (2011). *Database Modeling and Design: Logical Design* (Vol. 3, No. 1, pp. 4–9). Elsevier.
87. Thion, R., & Coulondre, S., (2012). A relational database integrity framework for access control policies. *Journal of Intelligent Information Systems, 38*, 131–159.

88. Van, W. K., Biljecki, F., & Van, D. S. S., (2016). Automatic update of road attributes by mining GPS tracks. *Transactions in GIS, 20*(5), 664–683.

89. Van, Z. M., (2015). Collective action as relational interaction: A new relational hypothesis on how non-activists become activists. *New Ideas in Psychology, 39*(1), 1–11.

90. Waguespack, L. J., & Waguespack, L. J., (2010). Promoting life using the relational paradigm. *Thriving Systems Theory and Metaphor-Driven Modeling* (Vol. 1, pp. 139–151).

91. Wang, R. Y., Ziad, M., & Lee, Y. W., (2006). *Data Quality* (Vol. 23, pp. 6–19). Springer Science & Business Media.

92. Wang, S., & Ariguzo, G., (2004). Knowledge management through the development of information schema. *Information & Management, 41*(4), 445–456.

93. Wilson, D. R., & Martinez, T. R., (1996). Instance-based learning with genetically derived attribute weights. In: *Proceedings of the International Conference on Artificial Intelligence, Expert Systems, and Neural Networks* (Vol. 1, pp. 11–14).

94. Worboys, M. F., Hearnshaw, H. M., & Maguire, D. J., (1990). Object-oriented data modelling for spatial databases. *International Journal of Geographical Information System, 4*(4), 369–383.

95. Xu, L., Fu, P., Xi, Y., Zhang, L., Zhao, X., Cao, C., & Ge, J., (2014). Adding dynamics to a static theory: How leader traits evolve and how they are expressed. *The Leadership Quarterly, 25*(6), 1095–1119.

96. Yu, K., Froese, T., & Grobler, F., (2000). A development framework for data models for computer-integrated facilities management. *Automation in Construction, 9*(2), 145–167.

97. Zhang, R., Indulska, M., & Sadiq, S., (2019). Discovering data quality problems: The case of repurposed data. *Business & Information Systems Engineering, 61*, 575–593.

98. Zhao, L., & Roberts, S. A., (1988). An object-oriented data model for database modelling, implementation and access. *The Computer Journal, 31*(2), 116–124.

99. Zhou, L., Burgoon, J. K., Twitchell, D. P., Qin, T., & Nunamaker, Jr. J. F., (2004). A comparison of classification methods for predicting deception in computer-mediated communication. *Journal of Management Information Systems, 20*(4), 139–166.

100. Zhuge, H., & Xing, Y., (2005). Integrity theory for resource space model and its application. In: *Advances in Web-Age Information Management: 6th International Conference, WAIM 2005, Hangzhou, China, October 11–13, 2005; Proceedings 6* (Vol. 1, pp. 8–24). Springer Berlin Heidelberg.

CHAPTER 4

BIG DATA ANALYTICS

UNIT INTRODUCTION

Data is created uninterruptedly and at a rushing rate. Many of these and other, including social media, mobile devices, and imaging technology used to make medical diagnoses, produce fresh data that essentially be saved someplace for more or less reasons. Devices and sensors automatically produce diagnostic data, which must be recorded and processed right away. Maintaining this large input of data is tough enough, but examining vast amounts of it to uncover meaningful arrangements and abstract actionable evidence is even more difficult—particularly once the data doesn't adhere to traditional notions of data structure. There is a chance to alter business, government, research, and daily life as a result of these data deluge difficulties. Some sectors have taken the lead in improving their capacity to collect and use data (Tsai et al., 2015):

1. Using rules developed by analyzing billions of transactions, credit card firms can accurately identify fraudulent purchases made by their consumers by closely monitoring every purchase they make.
2. Phone companies look at who their customers are phoning most often to see if they are on a competing network. A mobile phone service provider can provide a retention incentive to a customer who is considering leaving their service because of a more alluring deal being offered by a competing network.

3. For businesses like LinkedIn and Facebook, data is their main offering. These companies' valuations heavily depend on the data they collect and host, which has increasing inherent value as the volume of data increases.
4. Three qualities that stand out as identifying Big Data traits are (Russom, 2011):
5. Big Data can have millions of columns and billions of rows, which is much more than the millions or thousands of rows used in traditional data sets.
6. The heterogeneity of data sources, forms, and architectures that Big Data reflects, from digital footprints leftward on the web to other digital archives for future analysis.
7. The rate at which new data is generated and grows: Big Data is high-velocity data that is ingested quickly and analyzed in close to real-time.

Big Data is often better defined by its diversity and velocity than by its volume, which receives much of the media's attention. The three vs.—volume, variety, and velocity—are commonly used to define big data. Big Data cannot be effectively evaluated using solely conventional databases or methodologies because of its enormous scope or structure. In order to store, handle, and reap the commercial advantage from big data challenges, new technologies and tools are needed (Singh & Reddy, 2015). These modern resources help with the production, modification, and administration of both huge datasets and their storage environments. A leads to a variety of big data can be found in the 2011 McKinsey Global report:

Big Data is a data that, because of its scale, diversity, dispersion, or/and timeliness, needs the application of novel technical frameworks and analytics to produce intuitions that unleash new sources of economic value.

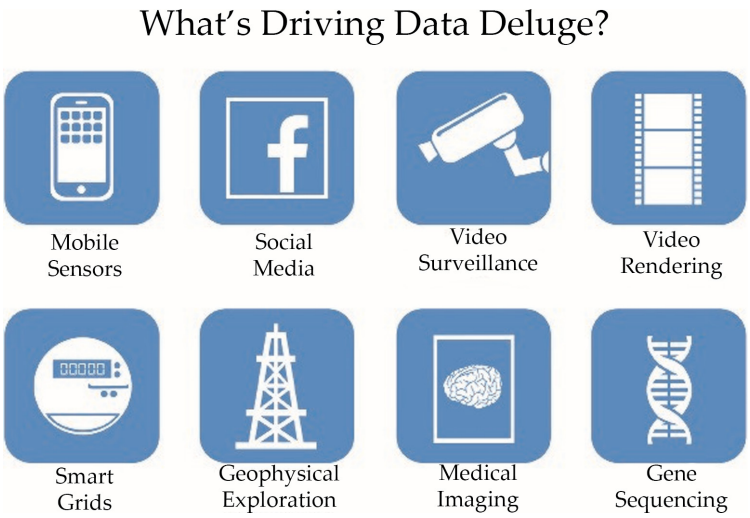


Figure 4.1. *Illustration of the causes of the data flood.*

Source: Charles Tappert, creative commons license.

According to McKinsey's description of big data, businesses will require novel data structures and analytical sandboxes, as well as tools, new systematic techniques, and an expansion of the data scientist's skill set. The Big Data flood has many sources, which are highlighted in Figure 4.1 (Kambatla et al., 2014).

Many of the factors in Figure 4.1 are contributing to an increase in the rate of data creation. Among the Big Data sources with the greatest growth rates and examples of non-conventional data sources utilized for analysis are social networking and genetic sequencing (Rajaraman, 2016).

Learning Objectives

After finishing this chapter, readers can learn:

- Data structures in Big Data
- Analyst perspectives on the data depositories
- State of the training in analytics
- Developing big data environment and novel methods to analytics including data devices, collection, aggregators.
- Main roles for the novel big data network

Key Terms

- Analytics
- Big Data
- Buyers
- Collectors
- Data
- Data Science
- Devices
- Drives
- Ecosystem
- Practice
- Users

4.1. DATA STRUCTURES

Big data can take many different types, comprising such text files, financial data, multimedia files, genetic maps, and other types of structured and unstructured data. Contrary to a large portion of the typical data analysis carried out by enterprises, most big information is either unorganized or semi-structured, requiring distinct processing and analysis methods and tools (Samet, 1984). The recommended method for processing such complicated data is to use parallel processing (MPP) architectures and distributed computing settings that allow parallel processing data ingest and analysis.

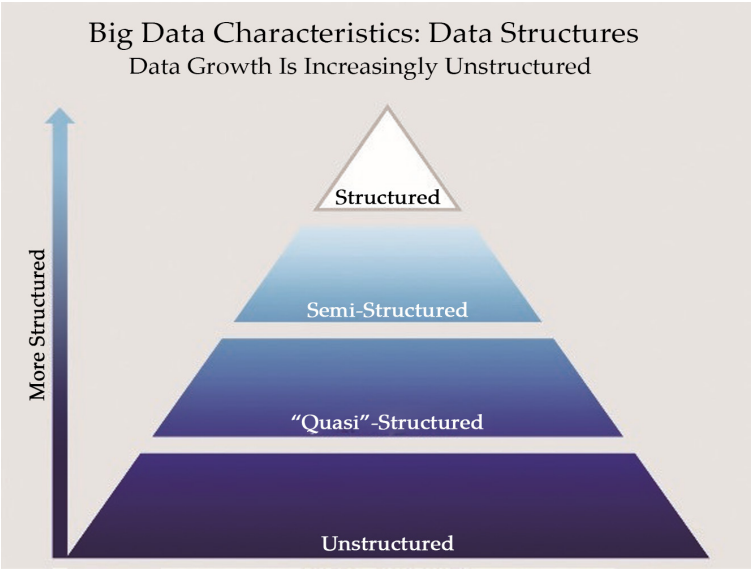


Figure 4.2. The growth of big data is becoming more unstructured.

Source: Palani Kumar, creative commons license.

Figure 4.2 depicts four distinct kinds of information structures, with non-structured data types expected to account for 80–90% of future data expansion (Nikhil & Pingali, 1989). The four are frequently combined despite being distinct. For instance, a traditional Relational Database Management System (RDBMS) might keep call records for a call center for software assistance. Machine types, time stamps, problem categories, and functioning systems are all examples of the kind of common structured data that an RDBMS may use to record details about support calls. Moreover, the system is likely to contain or semi-structured or quasi- data, unstructured data, for instance, free-form call log details derived from an email permit of the issue, client chat account, or a record of a telephone call outlining the practical issue and an elucidation,

or an acoustic recording of the handset call dialogue. The call center data's unstructured, semi-structured, and unstructured data can yield a wealth of new insights (Vuillemin, 1980).

Although structured data analysis is typically the most used technique, it is more difficult to examine semi-structured data (represented as XML), quaasi-structured data (represented as a click-stream), plus unstructured information (Malcolm, 1990).

Below are a few illustrations of each of the 4 major categories of data assemblies:

- Organized data Information with a specific schema and format.
- Semi-structured data consists of text files that follow a predictable format, making parsing them possible
- Data that is neither completely nor completely unstructured is called "quasi-structured"
- Unstructured data, which includes written documents, PDFs, photos, and video, is data that lacks an underlying framework (McKinney, 2010).

KEYWORD

A spreadsheet is a computer application for computation, organization, analysis and storage of data in tabular form.

4.2. ANALYST OUTLOOK ON DATA REPOSITORIES

With the advent of worksheets, corporate users could now easily apply simple logic to data that was organized into rows and columns and produce their own evaluations of operational issues. Spreadsheets can be quickly and easily configured to perform a wide variety of tasks, and their creation does not necessitate the expertise of a database administrator. Spreadsheets are simple to share, and the users are in charge of the underlying logic (Wu et al., 2019). However, their widespread use may lead to "many interpretations of the truth." To put it another way, it can be difficult to tell which user is using the most up-to-date worksheet with the most accurate logic and data. However, the spreadsheet's data and logic could be lost if a laptop is misplaced or a file is corrupted. Due to the continued widespread usage of spreadsheet products like Microsoft Excel, this problem will never go away. The demand for data centralization has never been greater due to the rise of data islands (also known as spreadmarts) (Smith et al., 2019).

Scalable data warehousing systems increased in number as data needs grew. These innovations made it possible to centrally

manage data, which resulted in improved security, redundancy, and a centralized location from which users could reliably retrieve the "official" source data required for tasks such as financial reporting. Access to a predefined set of dimensions within an RDBMS was made much simpler thanks to the framework that allowed for the development of OLAP cubes and BI analytical tools. Regression analysis and neural networks might be performed thanks to more sophisticated features. Enterprise Data Warehouses (EDWs) are essential for reporting and business intelligence (BI) jobs and address many of the issues that multiplying spreadsheets bring about, such as determining which of several spreadsheet versions is accurate. EDWs, along with a solid BI strategy, offer direct data feeds from centralized, managed, and protected sources (Lochmiller, 2021).

KEYWORD

An enterprise data warehouse (EDW) is a relational data warehouse containing a company's business data, including information about its customers.

EDWs and BI are advantageous, but they often limit the flexibility required to carry out comprehensive or experimental data analysis. Data analysts must rely on IT for access and modifications to the data schemas under the EDW model because data is owned and controlled by IT departments and database administrators (DBAs). Because of this, analysts must wait longer to receive data because much of the time is spent awaiting endorsements rather than beginning productive work. Moreover, the EDW guidelines frequently prevent analysts from creating datasets (Baker, 2009). As a result, it is frequent for new systems to emerge that are locally maintained by power users and include crucial data for building analytic datasets. In contrast to an EDW, unmanaged, insecure, and unbacked-up datasets are often frowned upon by IT departments. EDW and BI, from the standpoint of an analyst, address issues with data availability and accuracy. EDW and BI, however, create fresh issues with flexibility and agility that weren't as noticeable while working with spreadsheets (Kay & Kummerfeld, 2019).

The analytic sandbox is an attempt to overcome this problem by bridging the gap between EDW and more formally regulated corporate data, where analysts and data scientists sometimes find themselves at odds. Under this architecture, the analytic sandboxes may still be managed by the IT department, but they will be specifically created to support robust analytics while being centralized, controlled and guarded. These "workspaces," often known as "sandboxes," are intended to let teams study a variety of datasets under restricted conditions; they are not typically utilized for enterprise-level financial reporting or sales dashboards (Tok et al., 2011). High-performance computing is frequently made possible by analytic sandboxes using in-database processing,

where the analyzes take place inside the database itself. Instead of taking the data to an external analytical tool, it is thought that the analysis will perform better if it is conducted within the database itself. Advanced Analytics—Technology and Tools: In-Database Analytics, which is covered in more detail in Chapter 11, establishes connections to various data sources within an organization and saves time by eliminating the need to generate these data feeds one at a time. Although decreasing, though not eliminate, the costs involved with data stored on local, "shadow" file systems, in-database processing for profound analytics offers quicker turnaround time for designing and executing new analytic models. Moreover, analytic sandboxes can hold a wider range of information, such as raw data, textual data, as well as other types of unstructured data, without compromising with crucial production databases, as opposed to the often structured data in the EDW. The traits of the data repositories addressed in this section are summarized in Table 4.1 (Jarke et al., 2019).

Table 4.1. Categories of Data Repositories, According to Analyst Viewpoint

Data Sources	Features
Spreadsheets and spreadmarts	Low-volume databases and spreadsheets for recordkeeping Data extracts are necessary for analysts.
Data Warehouses	centrally located data storage in a dedicated area BI and reporting are supported, but comprehensive analysis is not. Analysts that need to access data and make schema modifications rely on IT and DBAs
Analytic Sandbox (workspaces)	To obtain consolidated and disaggregated data extracted from numerous sources, analysts must invest a lot of time. Data assets obtained for analysis from various sources and technologies adaptable, high-performance analysis that can take use of in-database processing in a non-production context Data repetition into "shadow" file schemes that are possessed by analysts slightly than DBAs is less expensive and risky.

Source: Manyika, creative commons license.

To make sure the strategy matches the required aims, there are a number of factors to take into account when working on big data analytics projects. These initiatives are well-suited to supporting high-value, tactical policymaking with increased handling complexity thanks to the properties of big data. Due to the volume and complexity of the data, the analytical procedures utilized in this situation must be iterative and flexible (Gray & Durcikova, 2005). Rapid and intricate analysis needs high amount system connections, and the tolerable level of delay must be taken into account. For example, creating an actual produce recommendation system for a website places more strains on the system than creating a near-actual-time recommendation system, which might still deliver usable presentation but has a little bit more delay and might be easier to implement. These factors necessitate a distinct way of approaching analytics difficulties, which will be discussed in more detail in the following section (Huerta & Jensen, 2017).

4.3. FORMAL PRACTICE IN ANALYTICS

As demonstrated in Table 4.2, firms have various chances to grow more logical and data driven as a result of current business difficulties.

Table 4.2. Business Drivers for Progressive Analytics

Business Driver	Instances
Optimization of business processes	Sales, rating, productivity, proficiency
Corporate risk identification	Customer-churn, scam, default
Prediction of latest business prospects	Cross-sell, Upsell, top new customer views
Complying with rules or governing necessities	Anti-Money-Laundering, Reasonable Loaning, Basel II-III, SarbanesOxley (SOX)

Source: Manyika, creative commons license.

Table 4.2 lists four types of typical business issues that businesses face and where they may be able to use forward-thinking analytics to gain an advantage. Administrations can use progressive analytical approaches to streamline operations and get greater value from these typical duties rather than just providing conventional reporting on these areas. The first three instances don't show any brand-new issues (Batarseh & Latif, 2016). For

many years, businesses have worked to lower customer turnover, boost sales, and cross-sell clients. The possibility to combine sophisticated analytical methods with Big Data to deliver more effective insights for these age-old issues is what's novel. The last illustration shows new legal requirements. Although many adherence and regulatory regulations have been in place for many years, new ones are always being added, adding to the complexity and data needs for enterprises. Advanced analytical approaches are necessary to adequately manage and comply with laws relating to anti-money-laundering (AML) and deception anticipation (Jiang & Fu, 2018).

4.3.1. Business Intelligence vs. Data Science

Did you Know?

According to a survey, 47% of organizations have both BI and data science teams.

To fully handle the four main business drivers listed in Table 4.2, a number of analytical methodologies are needed. There is a lot published about analytics in general, but the difference between BI and Data Science needs to be made clear. There are various approaches to compare these groupings of analytical procedures, as shown in Figure 4.3 (Larson & Chang, 2016).

Examining the timeframe as well as the analytical approaches being employed are two ways to gauge the kind of investigation being done. BI often offers reports, dashboards, and questions on current or historical business questions. With the use of BI tools, it is simple to respond to inquiries about quarter-to-date income, advancement to quarterly goals, and knowledge of the volume of a specific product vended during a preceding quarter or time. These queries are often closed- ended and clarify present or past conduct by gathering and organizing ancient data. BI offers some foresight and insight and typically responds to inquiries about "where" and "when" events happened (Fan et al., 2015).

Data science, on the other hand, focuses on comprehending the present and allowing knowledgeable choices about the forthcomings. It employs statistical data in a more inquisitive, prospective way. In order to predict upcoming invention sales and income more precisely than just ranging a drift line, a squad may service Data Science methods like time series analysis, which is additionally discussed in Chapter 8, "Advanced- Analytical -Theory and Methods; Time-series-Analysis." In lieu of integrating historical data, use this method to determine how many units of a specific product were purchased in the preceding quarter. Also, data science tends to be more experimental and can use scenario optimization to handle

more open-ended subjects. This approach focuses primarily on "how" and "why" questions while also providing information on ongoing actions and a view into the future (Sun et al., 2018).

In contrary to BI problems, which often ask for correctly formatted data organized in rows and columns for reliable reporting, data science initiatives frequently use a range of data sources, particularly large or atypical datasets. A corporation may elect to start a BI project if its main focus is on creating dashboards, performing basic imaginings or reporting, or a Data Science program if its main focus is on doing a more comprehensive analysis on disaggregated or diverse datasets, depending on its requirements (Campos et al., 2017).

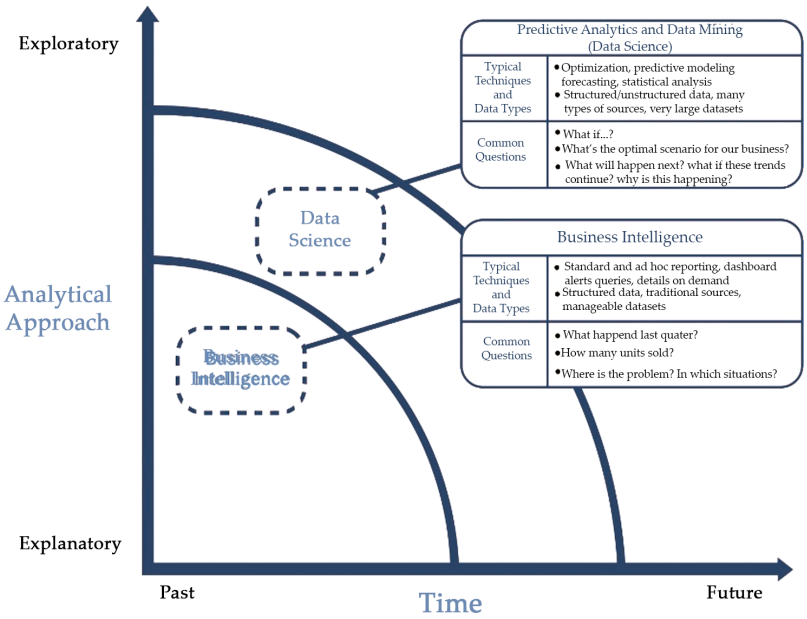


Figure 4.3. *Comparison of BI with data science.*

Source: Pouya Ataei, creative commons license.

4.3.2. Present Analytical Architecture

Workspaces designed specifically for data experimentation are essential for Data Science initiatives, as are data architectures that are both flexible and agile. The majority of firms always have data warehouses that perform traditional reporting and straightforward data analysis tasks very well, but sadly struggle to support more thorough studies. This section investigates a typical organizational analytical data architecture (Drakopulos et al., 2022).

A typical data structure is illustrated in Figure 4.4, along with some of the difficulties it poses for data scientists and other people attempting advanced analytics. In this analysis, we look at the Data Scientist's role in the process of acquiring data for analysis and how that data is delivered to them (Larson & Chang, 2016).

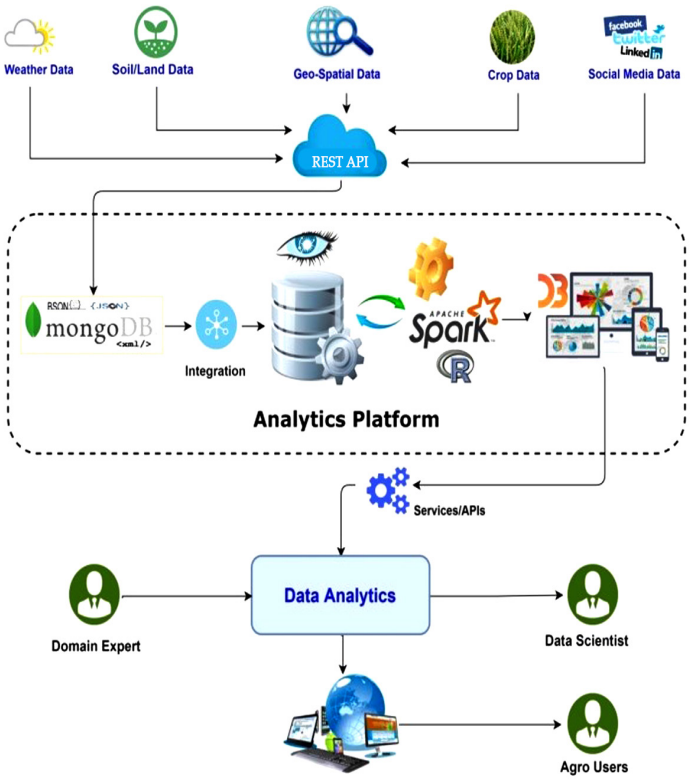


Figure 4.4. *Big data analytical architecture.*

Source: Purnima Shah, creative commons license.

1. Data must be fully understood, formatted, and standardized with the required data type definitions before data bases can be fed into the data storeroom. The backup, security, and failback of highly important data are made possible by this type of centralization, but it also means that data must typically undergo extensive pre – processing and entry points before it can arrive this type of skillful environment, which is not conducive to data discovery and iterative analytics (Spruit & Lytras, 2018).
2. As a consequence of this degree of switch over the EDW, added neighborhood systems, such as departmental storerooms plus local data markets made by business operators to meet their essential for supple analysis, may develop. The security and architectural restrictions that

apply to the main EDW may not apply to these local data marts, which gives users the ability to perform some deeper analysis. Yet, these standalone systems don't typically sync or connect with other data storage, and they might not even be backed up. They also exist in isolation.

3. After entering the data warehouse, information can be accessed by other enterprise-wide applications for BI and reporting. These crucial data feeds are obtained by high-priority operational processes from data repositories and warehouses.

Remember

With in-memory computing, data is stored directly in system memory.

4. Analysts receive data allocated for their upstream analytics at the conclusion of this workflow. Analysts make data excerpts from the EDW to perform off analyzes in R or additional native analytical tools since operators are typically not permitted to perform tradition or intense analytics on manufacture databases. These tools frequently don't have the capacity to analyze the complete population of a dataset; instead, they're restricted to in-memory analytics on the desktops and sampling data. These investigations are predicated on data extracts, thus they are performed in a different place, and the findings—along with any information on the data's quality or anomalies-rarely get transmitted rear into the primary data storehouse (Chen et al., 2016).

Data moves slowly into the EDW, and changes to the data schema take time since new data sources slowly assemble there thanks to the strict verification and data structuring procedures.

Although departmental data granaries could have been created initially with a specific goal and set of business requirements in mind, they have evolved over time to hold an increasing amount of data, some of which may have been pushed into pre-existing schemas to support BI and the development of OLAP cubes for analysis and reporting. Despite the fact that the EDW accomplishes the reporting goal and occasionally the generation of dashboards, EDWs typically limit the capacity of researchers to repeat on the data in a different nonproduction setting where they can perform in-depth analytics or analyze unstructured data (Szymańska, 2018).

The usual data architectures that were just discussed are intended to serve enterprise applications, provide corporate reporting functions, and store and analyze mission-critical data. While corporations still value reports and dashboards, most conventional data architectures prevent data discovery and more

in-depth analysis. Traditional data architectures can have various other effects on data scientists (Górriz et al., 2020).

1. High-value data is difficult to access and use, and it is often overlooked for use in activities like prediction analytics and data mining. The EDWs' focus on central data administration and reporting means that requests for data for analysis typically come second in importance to operational procedures.
2. Data is transferred from EDW to nearby analysis tools in batches. Because of this procedure, data scientists can only execute in-memory analytics (using R, SAS, SPSS, or Excel, for example), which limits the number of datasets they may use. As a result, sample restrictions may affect analysis, which could distort model correctness (Kalidindi & De Graef, 2015).
3. Data Science projects won't be centrally controlled; they will continue to be discrete and ad hoc. Because of this isolation, the organization will never be able to scale up the use of sophisticated analytics, and Data Science projects will always be unconventional endeavors that frequently don't correspond with corporate business objectives or strategy (Lowndes et al., 2017).

Remember

Data warehouse systems provide online analytical processing (OLAP) tools for interactive analysis of multidimensional data at varied granularity levels.

Many of these signs of the traditional information architecture contribute to a slow "time to insight" and less significant business impact than would be possible if the data were more easily accessible and supported by a setting that encouraged sophisticated analytics. As previously mentioned, creating analytical sandboxes would allow data scientists to carry out advanced analyzes in a regulated and approved manner. Current Data Warehousing systems, meanwhile, maintain a focus on providing reporting and BI capabilities to back up management and crucial processes (Farley et al., 2018).

4.3.3. Drivers of Big Data

It is useful to first comprehend some of the historical development of data stores, as well as the types of sources and tools used to govern these data stores, in order to better comprehend the market factors relating to big data.

Throughout the 1990s, the amount of information was frequently measured in terabytes, as seen in Figure 4.5. The majority of firms employed relational databases and data centers to handle enormous

volumes of enterprise information and analyze structured data in rows and columns. In the decade that followed, a variety of data sources proliferated, mostly production and publishing tools like web content repositories and network attached storage systems (Alles, 2015). This type of information also started to grow in quantity and be measured at petabyte scales. In the 2010s, companies' attempts to manage information have expanded to encompass a variety of new types of data. Everyone and everything these days is leaving a digital trace (Côrte-Real et al., 2019). An overview of the sources of Big Data produced by new applications, as well as the size and growth rate of the data, is shown in Figure 4.5. These applications produce exabyte-scale amounts of data, opening up possibilities for novel analytics and creating new value for businesses.

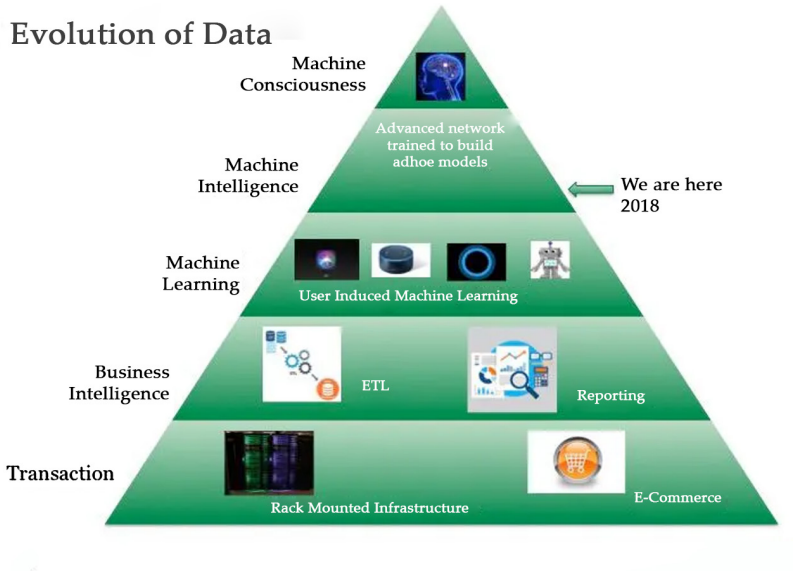


Figure 4.5. Data evolution and the growth of vast data resources.

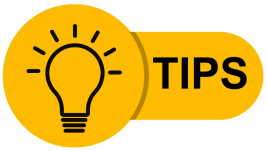
Source: Karpagam Narayanan, creative commons license.

Data now originates from an array of sources, including the following (Alles, 2015):

- Medical data including genetic sequencing and imaging diagnostics.
- Postings of images and video to the Internet.
- Video surveillance, such the tens of thousands of cameras positioned all over a city.
- Mobile devices that offer geospatial location information of their users as well as metadata on texts, phone calls, and smart phone application usage.

- Smart gadgets offer sensor-based data collecting from smart buildings, smart electric grids, and numerous other public and industrial infrastructures.
- Non-conventional IT tools, such as the utilization of GPS navigation systems, RFID readers, and seismic processing.

A large amount of information is being produced by the Big Data trend from numerous new sources. In order to benefit from these prospects and innovative market dynamics—discussed in the following subdivision—this data flood necessitates better analytics and new market participants (Liu et al., 2017).



Unlike purchased retail games, online games have the problem of not being permanently playable, as they require special servers in order to function.

4.4. NEW APPROACHES TO ANALYZING BIG DATA

A new economy is developing as a result of businesses and data collectors discovering the intrinsic value of the data they can acquire from people. The market is seeing the entry of data sellers and data cleaners that employ crowdsourcing (like Mechanical Turk and Galaxy Zoo) to test the results of machine learning methods as this new digital economy endures to develop. By simplifying the packaging of open source products and distributing them, other vendors contribute value. This value-add has been offered for the open source Hadoop framework by vendors including Cloudera, Hortonworks, and Pivotal (Curry, 2016).

There are four primary categories of stakeholders in this intricate web as the new ecology emerges.

4.4.1. Data Devices

The “Sensornet” collects data from many sources and generates new data about the collected data on a continuous basis. There is an incremental petabyte of data created about each new gigabit of data (Singh, 2019). Take the case of someone using a computer, gaming console, or smartphone to play an online video game (Singh, 2019). In this instance, the video game service records information regarding the player’s proficiency and degree of achievement. Intelligent systems keep track of the user’s game play and record it. As a result, the game developer can adjust the game’s difficulty, recommend further games that are similar and are most probably to interest the player, and give extra equipment and character improvements based on the user’s age, sex, and hobbies. This

data may be kept on-site or transferred to the game publisher's cloud in order to evaluate user behavior, spot up-sell and cross-sell opportunities, and create archetypal personas for particular types of players (Faroukhi et al., 2020).

Remember

Data devices such as GPUs and FPGAs are increasingly being used in analyzing big data due to their parallel processing capabilities and ability to handle large amounts of data, enabling faster and more efficient data processing.

Another comprehensive data source is smartphones. They store and send information regarding Internet, SMS, and real-time location usage in additional to messaging and standard phone use. By examining the concentration of smartphones in a given area, this metadata may be utilized to analyze traffic patterns and track vehicle speeds or levels of relative congestion on popular highways. In this manner, GPS systems in vehicles can provide drivers with real-time updates and suggest detours to escape traffic (Nagorny et al., 2017).

Major shopping loyalty cards keep track of not just how much a person spends, but also the stores they visit, the products they buy, the stores they frequent the most, and the pairings of things they buy. The gathering of this information offers insights into purchasing and travel patterns as well as the probability of effective ad aiming for particular kinds of wholesale promotions (Chaudhari & Sinha, 2021).

4.4.2. Data Gatherers

This includes illustrations of entities that gather user and device data. The TV shows a person sees, the TV channels they would and won't be paying for to view on request, and the fees they are keen to shell out for premium TV content are all tracked by a cable TV provider. Using geo location data gathered from the RFID chips, retailers track a customer's movement throughout their supply while moving a shopping cart to determine which products receive the most foot traffic.

4.4.3. Data Aggregators

This clarifies the information gathered from the various "SensorNet" or "Internet of Things" organizations. These businesses collate information from devices and usage trends gathered by governmental bodies, businesses, and websites. They can then decide to change the data and package it as goods to sell and listed operators who may want to create marketing lists of potential targets for particular advertising campaigns (Martin, 2016).

4.4.4. Data Users and Consumers

The data gathered and combined by others along the data supply series directly benefits these entities. As a data purchaser, retail banks may be interested in learning which clients are most likely to submit an application for a new debt or a line of credit based on their home equity. Marketing banks may buy data from a facts aggregator to offer input for this research. This type of information may include demographics about local residents, people who show up to have a certain amount of debt but still have strong credit scores (or other traits like on-time bill payment and reserves accounts that may be employed to infer credit worthiness), and people who search the internet for advice on debt repayment or home improvement projects. Big Data will make it easier to create marketing campaigns that are more specifically focused than would have been possible in the past owing to an absence of knowledge or high-level technologies (Lohse et al., 2000).

Users can evaluate public opinion on topics like presidential elections by applying natural language handling to unstructured text data culled from social media platforms using tools like Hadoop. For instance, someone might want to examine relevant blogs and internet comments to ascertain how the general public feels about a candidate. Similar to this, data users may wish to monitor and get ready for natural catastrophes by figuring out which regions a storm strikes first and how it progresses grounded on which geographical areas are tweeting or talking about it on social media (Moon & Tikoo, 2002) (Figure 4.6).

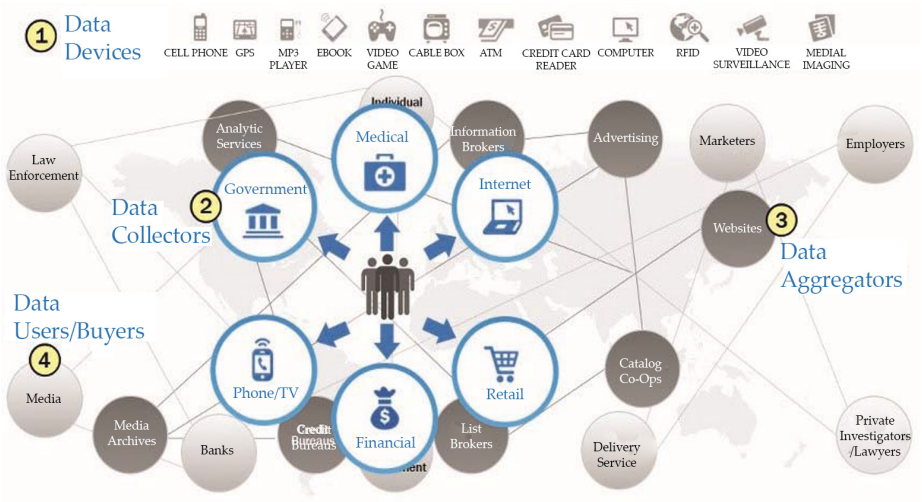
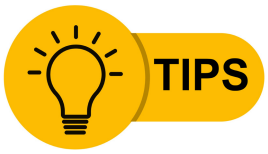


Figure 4.6. *Developing big data ecosystem.*

Source: Jordan Nash, creative commons license.



When analyzing big data, it's important to consider who will be using the data and what their needs are.

This developing Vast Data ecosystem serves as an example of the wide variety of data types and associated market dynamics. These datasets may comprise social media, structured datasets, text, and sensor data. In light of this, it is important to keep in mind that these data will not function effectively within conventional EDWs, which were designed to be centrally maintained and to simplify reporting and dashboards. Instead, in order to be successful, big data initiatives and problems involve various methods (Jung et al., 2018).

To obtain the data they require within an analytical sandbox, analysts must collaborate with IT and DBAs. Data with various types of organization, aggregated data, and raw data are all common components of an analytical sandbox. The sandbox facilitates thorough data exploration and calls for a knowledgeable user to make use of and benefit from facts in the sandbox setting (Kuan et al., 2014).

4.5. IMPORTANCE OF DIFFERENT IMMENSE DATA ECOSYSTEM

Fresh players have appeared to curate, save, produce, clean, and manage data. New jobs, new technological platforms, and novel analytical approaches have also emerged as a result of the requirement to apply more sophisticated analytical methodologies to more complicated company challenges. The succeeding chapters examine a few of the analytical techniques and technological platforms after this part examines the different roles that fulfill these needs.

As Figure 4.7, the Immense Data ecosystem requires three different types of roles. These positions were detailed in the McKinsey Global research on Big Data, from May 2011 (Yang et al., 2016). The very first group—Profound Analytical Talent—has high analytical abilities and is technologically adept. Members are equipped with a variety of abilities to manage unstructured, raw data and to use sophisticated analytical methods at scale. Our team has advanced knowledge in quantitative fields like machine learning, statistics, and math. Members must have access to a solid analytical sandbox or workspace in which to do extensive analytical data experiments in order to accomplish their duties. The new position of the Data Scientist is one example of a modern profession falling into this category, along with statisticians, economists, and mathematicians (Shibuya et al., 2022).

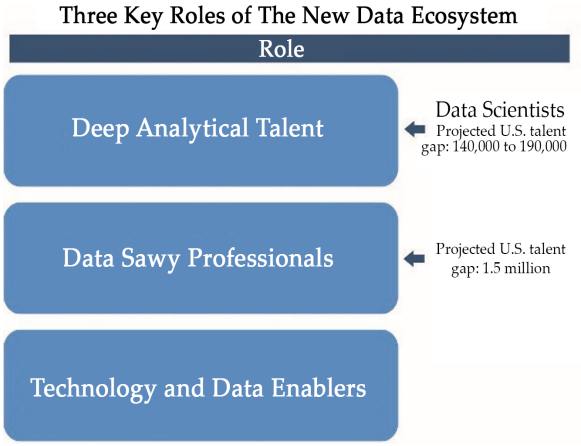


Figure 4.7. *Important parts of the different big data ecosystem.*

Source: Jordan Nash, creative commons license.

According to the McKinsey study, there will be an unavailability of 140,000–190,000 profound analytical talent in the United States by 2018. This range instead shows the discrepancy among what will be accessible in the workforce and what will be needed. It does not reflect the amount of people required with profound analytical talent. However, these projections just take into account skills shortage in the United States; on a global scale, the figure would be substantially higher (Li et al., 1999).

Data Savvy Professionals, the second type, are less technically savvy but have a foundational understanding of statistics or deep learning and can identify the major queries that can be addressed by progressive analytics. These individuals typically have a basic understanding of dealing with data or a respect for some of the tasks carried out by data scientists and other individuals with strong analytical skills. Financial analysts, market research experts, life scientists, management teams, and business and functional managers are a few examples of data-savvy professionals (Cui et al., 2020).

The McKinsey report estimates that by 2018, there would be a 1.5 million person talent gap in the United States for this group. On a broad scale, this suggests that the need for Data Savvy Workers will be ten times greater than that for Data Scientist profiles. Becoming data savvy is a crucial step in extending managers, directors, and leaders' perspectives because it gives them an understanding of the kind of questions that may be answered with data (Orenga-Roglá & Chalmeta, 2018).

The survey mentions technology and data facilitators as the third group of people. This group is made up of individuals that offer their technical skills to help analytical projects, like setting up and administering analytical sandboxes and overseeing large-scale data structures that permit extensive analytics across businesses and organizations. Computer engineering, programming, and database management expertise are needed for this position. To overcome challenging Big Data problems, these three groups must closely collaborate. The last two kinds of individuals are well known to most businesses, whereas the first group, Deep Analytical Talent, is typically the modern and least understood function for most. This talk concentrates on the growing function of the data scientist for the sake of simplicity. It gives a more thorough understanding of the skills required to perform that function and outlines the various tasks needed to fulfill that role (Shah et al., 2021).

ACTIVITY 4.1

An e-commerce business is considering using big data analytics to personalize its product recommendations. How can it ensure the privacy and security of its customers' data?

Data scientists regularly engage in three categories of tasks:

1. Recast business difficulties as analytics difficulties. This ability enables one to diagnose business issues, think about the underlying causes of a problem, and choose the potential analytical approaches that could be used to address it.
2. Create, put into practice, and use statistical models as well as data mining methods on big data. When people consider the duties of a data scientist, they typically picture this group of tasks: leveraging data to apply sophisticated analytical techniques to a range of business issues.
3. Create knowledge that results in practical advice. It is important to remember that applying cutting-edge techniques to data challenges does not always result in new economic value. Instead, it's critical to develop your ability to explain data insights clearly.

In general, there are five key skill sets and personality traits associated with data scientists (Hernandez-Almazan et al., 2022) (Figure 4.8).

1. Quantitative expertise, such as in statistics or mathematics.
2. Technological prowess, specifically expertise in software engineering, machine learning, and programming.
3. A doubtful mindset and critical thinking skills: Data scientists must be able to critically evaluate their work rather than viewing it from only one perspective.

4. Inquisitive and inventive: Data scientists are enthusiastic about data and seek for inventive means to address issues and present information.
5. Communication and teamwork skills: Data scientists should be able to clearly communicate the business significance and work cooperatively with other organizations, especially project sponsors and important stakeholders.

Figure 4.8. Responsibilities of a data scientist.

Source: Tech Vidvan, creative commons license.

The majority of data scientists feel at ease by this combination of abilities to gather, organize, evaluate, and display information and present meaningful stories about it.

SUMMARY

Social media, sensors, the IoT, and video observation are just a few examples of the various types of data that contribute to Big Data. Some organizations are coming up with innovative methods to use big data to address their expanding business requirements and more difficult issues as they battle to keep up with shifting market demands. Organizations tend to go beyond standard BI activities, including using data to create reports and dashboards, and toward Data Science-driven projects that seek answers to more unrestricted and complicated queries as their processes change and they become aware of the benefits that Big Data may offer.

To take advantage of the opportunities offered by Big Data, new data framework, notably analytical sandboxes, new methods of operation, and personnel with innovative skill sets are necessary. These factors are motivating businesses to create data science teams and analytical sandboxes. The majority of firms do not have data scientists, despite the fact that some are fortunate enough to have them, as a result of the difficulty in quickly discovering and employing data scientists due to the widening skill gap. Yet, businesses are starting to use Big Data in inventive and original ways, including those in web retail, healthcare, genomics, innovative IT infrastructures, and social media.

REVIEW QUESTIONS

1. What are Big Data's three distinguishing features, and what factors must be taken into account when processing Big Data?
2. Describe an analytical sandbox and explain its significance.
3. Describe how BI and data science are different.
4. Identify the difficulties with the data scientists' current analytical architecture.
5. What core competencies and personality traits define a data scientist?

MULTIPLE CHOICE QUESTIONS

1. Which of the following refers to the organization and storage of large data sets?

a. Big Data
b. Data Aggregation
c. Data Analytics
d. Data Devices

2. What are the two main perspectives in analyzing Big Data?

a. Statistical and Predictive
b. Technical and Managerial

c. Business and Financial

d. Analyst and Developer

3. Which of the following is a challenge when training individuals in analytics for Big Data?

a. Limited access to data

b. Limited access to software tools

c. Limited funding for training programs

d. Limited interest in the field

4. What is the main purpose of data aggregators in Big Data?

a. To collect data from different sources

b. To organize and structure large data sets

c. To filter out irrelevant data

d. To perform statistical analysis on the data

5. Which of the following is not a main role for the novel Big Data network?

a. Data processing and analysis

b. Data storage and retrieval

c. Data aggregation and filtering

d. Data visualization and reporting

6. Which of the following best describes the term "data devices" in the context of Big Data?

a. Software tools used for data analysis

b. Hardware used for data storage and retrieval

c. Algorithms used for data processing

d. Techniques used for data visualization

Answers to Multiple Choice Questions

1. (a); 2. (d); 3. (a); 4. (b); 5. (c); 6. (b)

REFERENCES

1. Alles, M. G., (2015). Drivers of the use and facilitators and obstacles of the evolution of big data by the audit profession. *Accounting Horizons, 29*(2), 439–449.

2. Baker, K. S., (2009). Data stewardship: Environmental data curation and a web of repositories. *Digital Discourse: The E-volution of Scholarly Communication, 1*(1), 4–19.

3. Batarseh, F. A., & Latif, E. A., (2016). Assessing the quality of service using big data analytics: With application to healthcare. *Big Data Research, 4*, 13–24.

4. Campos, J., Sharma, P., Gabiria, U. G., Jantunen, E., & Baglee, D., (2017). A big data analytical architecture for the asset management. *Procedia CIRP*, *64*, 369–374.
5. Chaudhari, S. L., & Sinha, M., (2021). A study on emerging trends in Indian startup ecosystem: Big data, crowd funding, shared economy. *International Journal of Innovation Science, 13*(1), 1–16.
6. Chen, Y., Chen, H., Gorkhali, A., Lu, Y., Ma, Y., & Li, L., (2016). Big data analytics and big data science: A survey. *Journal of Management Analytics, 3*(1), 1–42.
7. Cho, S., Vasarhelyi, M. A., & Zhang, C., (2019). The forthcoming data ecosystem for business measurement and assurance. *Journal of Emerging Technologies in Accounting, 16*(2), 1–21.
8. Côrte-Real, N., Ruivo, P., Oliveira, T., & Popovič, A., (2019). Unlocking the drivers of big data analytics value in firms. *Journal of Business Research, 97*, 160–173.
9. Cui, Y., Kara, S., & Chan, K. C., (2020). Manufacturing big data ecosystem: A systematic literature review. *Robotics and Computer-Integrated Manufacturing, 62*, 101861.
10. Curry, E., (2016). The big data value chain: Definitions, concepts, and theoretical approaches. *New Horizons for a Data-Driven Economy: A Roadmap for Usage and Exploitation of Big Data in Europe, 1*(1), 29–37.
11. Drakopulos, L., Havice, E., & Campbell, L., (2022). Architecture, agency and ocean data science initiatives: Data-driven transformation of oceans governance. *Earth System Governance, 12*, 100140.
12. Fan, S., Lau, R. Y., & Zhao, J. L., (2015). Demystifying big data analytics for business intelligence through the lens of marketing mix. *Big Data Research, 2*(1), 28–32.
13. Farley, S. S., Dawson, A., Goring, S. J., & Williams, J. W., (2018). Situating ecology as a big-data science: Current advances, challenges, and solutions. *BioScience, 68*(8), 563–576.
14. Faroukhi, A. Z., El Alaoui, I., Gahi, Y., & Amine, A., (2020). Big data monetization throughout big data value chain: A comprehensive review. *Journal of Big Data, 7*(1), 1–22.
15. Górriz, J. M., Ramírez, J., Ortíz, A., Martinez-Murcia, F. J., Segovia, F., Suckling, J., & Ferrandez, J. M., (2020). Artificial intelligence within the interplay between natural and artificial computation: Advances in data science, trends and applications. *Neurocomputing, 410*, 237–270.
16. Gray, P. H., & Durcikova, A., (2005). The role of knowledge repositories in technical support environments: Speed versus learning in user performance. *Journal of Management Information Systems, 22*(3), 159–190.
17. Hernandez-Almazan, J. A., Chalmeta, R., Roque-Hernández, R. V., & Machucho-Cadena, R., (2022). A framework to build a big data ecosystem oriented to the collaborative networked organization. *Applied Sciences, 12*(22), 11494.
18. Huerta, E., & Jensen, S., (2017). An accounting information systems perspective

on data analytics and big data. *Journal of Information Systems, 31*(3), 101–114.

19. Jarke, M., Jeusfeld, M. A., Quix, C., & Vassiliadis, P., (1999). Architecture and quality in data warehouses: An extended repository approach. *Information Systems, 24*(3), 229–253.
20. Jiang, M., & Fu, K. W., (2018). Chinese social media and big data: Big data, big brother, big profit? *Policy & Internet, 10*(4), 372–392.
21. Jung, T., Li, X. Y., Huang, W., Qiao, Z., Qian, J., Chen, L., & Hou, J., (2018). AccountTrade: Accountability against dishonest big data buyers and sellers. *IEEE Transactions on Information Forensics and Security, 14*(1), 223–234.
22. Kalidindi, S. R., & De Graef, M., (2015). Materials data science: Current status and future outlook. *Annual Review of Materials Research, 45*, 171–193.
23. Kambatla, K., Kollias, G., Kumar, V., & Grama, A., (2014). Trends in big data analytics. *Journal of Parallel and Distributed Computing, 74*(7), 2561–2573.
24. Kay, J., & Kummerfeld, B., (2019). From data to personal user models for life-long, life-wide learners. *British Journal of Educational Technology, 50*(6), 2871–2884.
25. Kuan, K. K., Zhong, Y., & Chau, P. Y., (2014). Informational and normative social influence in group-buying: Evidence from self-reported and EEG data. *Journal of Management Information Systems, 30*(4), 151–178.
26. Larson, D., & Chang, V., (2016). A review and future direction of agile, business intelligence, analytics and data science. *International Journal of Information Management, 36*(5), 700–710.
27. Li, H., Kuo, C., & Rusell, M. G., (1999). The impact of perceived channel utilities, shopping orientations, and demographics on the consumer's online buying behavior. *Journal of Computer-Mediated Communication, 5*(2), JCMC521.
28. Lindman, J., Kinnari, T., & Rossi, M., (2015). Business roles in the emerging open-data ecosystem. *IEEE Software, 33*(5), 54–59.
29. Liu, Y., Teichert, T., Rossi, M., Li, H., & Hu, F., (2017). Big data for big insights: Investigating language-specific drivers of hotel satisfaction with 412,784 user-generated reviews. *Tourism Management, 59*, 554–563.
30. Lochmiller, C. R., (2021). Conducting thematic analysis with qualitative data. *Qualitative Report, 26*(6), 6–10.
31. Lohse, G. L., Bellman, S., & Johnson, E. J., (2000). Consumer buying behavior on the internet: Findings from panel data. *Journal of Interactive Marketing, 14*(1), 15–29.
32. Lowndes, J. S. S., Best, B. D., Scarborough, C., Afflerbach, J. C., Frazier, M. R., O'Hara, C. C., & Halpern, B. S., (2017). Our path to better science in less time using open data science tools. *Nature Ecology & Evolution, 1*(6), 0160.
33. Malcolm, G., (1990). Data structures and program transformation. *Science of Computer Programming, 14*(2, 3), 255–279.
34. Martin, K., (2016). Data aggregators, consumer data, and responsibility online: Who

is tracking consumers online and should they stop? *The Information Society, 32*(1), 51–63.

35. Mazumdar, S., Seybold, D., Kritikos, K., & Verginadis, Y., (2019). A survey on data storage and placement methodologies for cloud-big data ecosystem. *Journal of Big Data, 6*(1), 1–37.
36. McKinney, W., (2010). Data structures for statistical computing in python. In: *Proceedings of the 9th Python in Science Conference* (Vol. 445, No. 1, pp. 51–56).
37. Moon, J., & Tikoo, S., (2002). Buying decision approaches of organizational buyers and users. *Journal of Business Research, 55*(4), 293–299.
38. Nagorny, K., Lima-Monteiro, P., Barata, J., & Colombo, A. W., (2017). Big data analysis in smart manufacturing: A review. *International Journal of Communications, Network and System Sciences, 10*(3), 31–58.
39. Nathan, R., Monk, C. T., Arlinghaus, R., Adam, T., Alós, J., Assaf, M., & Jarić, I., (2022). Big-data approaches lead to an increased understanding of the ecology of animal movement. *Science, 375*(6582), eabg1780.
40. Nikhil, R. S., & Pingali, K. K., (1989). I-structures: Data structures for parallel computing. *ACM Transactions on Programming Languages and Systems (TOPLAS), 11*(4), 598–632.
41. Orenga-Roglá, S., & Chalmeta, R., (2018). Framework for implementing a big data ecosystem in organizations. *Communications of the ACM, 62*(1), 58–65.
42. Pappas, I. O., Mikalef, P., Giannakos, M. N., Krogstie, J., & Lekakos, G., (2018). Big data and business analytics ecosystems: Paving the way towards digital transformation and sustainable societies. *Information Systems and e-Business Management, 16*, 479–491.
43. Rajaraman, V., (2016). Big data analytics. *Resonance, 21*, 695–716.
44. Russom, P., (2011). *Big Data Analytics* (Vol. 19, No. 4, pp. 1–34). TDWI best practices report, fourth quarter.
45. Samet, H., (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR), 16*(2), 187–260.
46. Shah, S. I. H., Peristeras, V., & Magnisalis, I., (2021). Government big data ecosystem: Definitions, types of data, actors, and roles and the impact in public administrations. *ACM Journal of Data and Information Quality, 13*(2), 1–25.
47. Shibuya, Y., Lai, C. M., Hamm, A., Takagi, S., & Sekimoto, Y., (2022). Do open data impact citizens' behavior? Assessing face mask panic buying behaviors during the COVID-19 pandemic. *Scientific Reports, 12*(1), 17607.
48. Shin, D. H., (2016). Demystifying big data: Anatomy of big data developmental process. *Telecommunications Policy, 40*(9), 837–854.
49. Singh, D., & Reddy, C. K., (2015). A survey on platforms for big data analytics. *Journal of Big Data, 2*(1), 1–20.

50. Singh, N., (2019). Big data technology: Developments in current research and emerging landscape. *Enterprise Information Systems, 13*(6), 801–831.
51. Smith, M., Turner, K., Bond, R., Kawakami, T., & Roos, L. L., (2019). The concept dictionary and glossary at MCHP: Tools and techniques to support a population research data repository. *International Journal of Population Data Science, 4*(1), 1–20.
52. Spruit, M., & Lytras, M., (2018). Applied data science in patient-centric healthcare: Adaptive analytic systems for empowering physicians and patients. *Telematics and Informatics, 35*(4), 643–653.
53. Sun, Z., Sun, L., & Strang, K., (2018). Big data analytics services for enhancing business intelligence. *Journal of Computer Information Systems, 58*(2), 162–169.
54. Szymańska, E., (2018). Modern data science for analytical chemical data: A comprehensive review. *Analytica Chimica Acta, 1028*, 1–10.
55. Taylor, L., (2016). The ethics of big data as a public good: Which public? Whose good? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 374*(2083), 20160126.
56. Tok, A. Y., Zhao, M., Chow, J. Y., Ritchie, S., & Arkhipov, D., (2011). Online data repository for statewide freight planning and analysis. *Transportation Research Record, 2246*(1), 121–129.
57. Tsai, C. W., Lai, C. F., Chao, H. C., & Vasilakos, A. V., (2015). Big data analytics: A survey. *Journal of Big Data, 2*(1), 1–32.
58. Vuillemin, J., (1980). A unifying look at data structures. *Communications of the ACM, 23*(4), 229–239.
59. Wu, M., Psomopoulos, F., Khalsa, S. J., & De Waard, A., (2019). Data discovery paradigms: User requirements and recommendations for data repositories. *Data Science Journal, 18*(1), 5–10.
60. Yang, J., Sia, C. L., Liu, L., & Chen, H., (2016). Sellers versus buyers: Differences in user information sharing on social commerce sites. *Information Technology & People, 29*(2), 444–470.

CHAPTER 5

STREAM PROCESSING SYSTEMS

UNIT INTRODUCTION

The chapter on focuses on the use of stream processing systems to enable real-time data processing and analytics in advanced database systems. The chapter provides an overview of stream processing systems and their importance in handling continuous and real-time data streams, as well as their use in-memory processing, distributed processing, scalability, and fault tolerance. The chapter also discusses the major stream processing frameworks, including Apache Kafka, Apache Storm, Apache Flink, and Apache Spark Streaming, and explores their unique features and capabilities (Cherniack et al., 2003). Furthermore, the chapter highlights the major applications of stream processing systems, including real-time analytics, fraud detection, IoT, and social media analysis. The chapter also discusses the integration of stream processing systems with database management systems (IDMSs), including data integration techniques, real-time data warehousing, data archiving and retrieval, and data security and privacy. Finally, the chapter explores the challenges and future directions for stream processing systems, including scalability and performance, data consistency and quality, integration with machine learning, and future trends in stream processing systems (Stephens, 1997) (Figure 5.1).

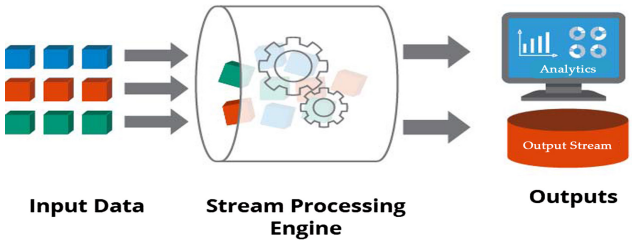


Figure 5.1. Schematic of the stream processing.

Source: Hazel Cast, creative commons license.

Stream processing systems are a class of distributed computing systems designed to process continuous data streams in real-time. Unlike traditional batch processing systems, stream processing systems are capable of processing data as soon as it becomes available and can produce results in near real-time. Stream processing systems typically employ in-memory processing techniques and are designed to scale horizontally across a cluster of machines (Cardellini et al., 2022).

Advanced database systems are designed to support complex data processing requirements beyond the capabilities of traditional relational database systems. These systems are typically designed to handle large-scale, distributed datasets and support a wide range of data processing workflows including real-time analytics, machine learning, and graph processing. Advanced database systems typically employ a variety of data processing techniques including batch processing, real-time processing, and stream processing (Shukla et al., 2017). Stream processing systems are a critical component of many advanced database systems, as they provide the ability to process and analyze real-time data streams at scale. By processing data in real-time, stream processing systems enable organizations to make informed decisions faster, detect and respond to emerging trends and issues quickly, and improve overall operational efficiency. Stream processing systems are particularly important in applications such as financial fraud detection, IoT analytics, social media analysis, and real-time monitoring of system logs and sensor data.

In addition to supporting real-time data processing, stream processing systems can also integrate with other data processing workflows in advanced database systems. For example, stream processing systems can be used to feed real-time data into a data warehouse for historical analysis, or to train machine learning models in real-time. As such, stream processing systems are an important tool for organizations looking to derive insights and value from their data in real-time (Botan et al., 2010).

Learning Objectives

At the end of this chapter, readers will be able to:

- Understanding the characteristics of stream processing systems, including their ability to process continuous and real-time data streams, use in-memory processing

techniques, support distributed processing, and provide scalability and fault tolerance.

- Familiarizing with the major stream processing frameworks, including Apache Kafka, Apache Storm, Apache Flink, and Apache Spark Streaming, and understanding their unique features and capabilities.
- Exploring the major applications of stream processing systems, including real-time analytics, fraud detection, IoT, and social media analysis.
- Understanding the integration of stream processing systems with database management systems, including data integration techniques, real-time data warehousing, data archiving and retrieval, and data security and privacy.
- Identifying the challenges and future directions for stream processing systems, including scalability and performance, data consistency and quality, integration with machine learning, and future trends in stream processing systems.
- Understanding the implications of stream processing systems for industry and research, including their potential to transform the way organizations derive value from their data.

Key Terms

- Apache Flink
- Apache Kafka
- Apache Spark Streaming
- Apache Storm
- Artificial Intelligence
- Data Consistency
- Data Quality
- Distributed Processing
- Edge Computing
- Fault Tolerance
- Fraud Detection
- In-memory Processing
- IoT
- Performance
- Real-time Analytics
- Real-time Data Processing
- Scalability
- Stream Processing Systems

5.1. CHARACTERISTICS OF STREAM PROCESSING SYSTEMS

Following are the characteristics of the stream processing systems:

5.1.1. Continuous and Real-time Data Processing

Stream processing systems are designed to handle continuous and real-time data streams that are generated at high velocity. They can process data as it is generated, without waiting for the data to be collected and stored in a database. This makes stream processing systems ideal for applications that require real-time data analysis and decision-making (Kaya & Safak, 2015) (Figure 5.2).

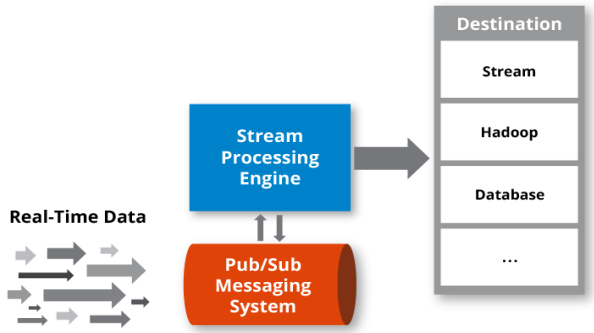


Figure 5.2. Illustration of the continuous and real-time data processing.

Source: Hazel cast, creative commons license.

5.1.2. In-memory Processing

Stream processing systems typically use in-memory processing techniques to enable high-speed data processing. In-memory processing means that data is stored in memory rather than on disk, which reduces latency and enables faster data processing.

5.1.3. Distributed Processing

Stream processing systems are typically designed as distributed systems that can process data across multiple nodes in a cluster. Distributed processing enables stream processing systems to scale horizontally and handle large volumes of data. It also provides fault tolerance, as data can be replicated across multiple nodes

in the cluster, ensuring that data is not lost if a node fails (Fox & Friston, 2012).

5.1.4. Scalability and Fault Tolerance

Stream processing systems are designed to be highly scalable and fault-tolerant. They can handle large volumes of data and can be scaled up or down as needed. They are also designed to handle node failures gracefully, by replicating data across multiple nodes in the cluster. This ensures that data is not lost if a node fails and that the system can continue to operate smoothly. Scalability and fault tolerance are critical characteristics of stream processing systems, as they enable these systems to process data reliably at scale (Abd-El-Malek et al., 2005).

5.2. STREAM PROCESSING FRAMEWORKS

Following is the stream processing framework:

5.2.1. Apache Kafka

Apache Kafka is an open-source distributed streaming platform that is used for building real-time data pipelines and streaming applications. Kafka is designed to handle high-throughput, low-latency data streams and provides reliable, fault-tolerant data delivery. It is widely used for applications such as data ingestion, real-time analytics, and data processing (Wang et al., 2015) (Figure 5.3).

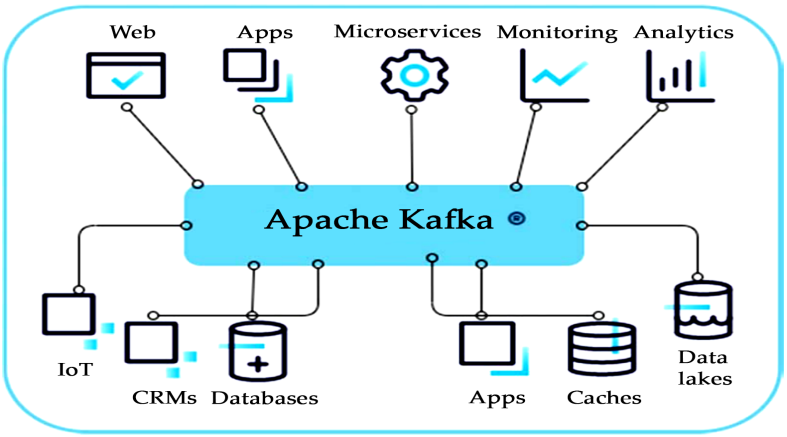


Figure 5.3. Schematic of the Apache Kafka.

Source: Confluent, Inc, creative commons license.

5.2.2. Apache Storm

Apache Storm is an open-source distributed real-time computation system that is used for processing streaming data. Storm is designed to handle real-time data processing at scale and provides fault tolerance, reliability, and scalability. It is widely used for applications such as real-time analytics, online machine learning, and fraud detection.

5.2.3. Apache Flink

Did you Know?

According to a report, the in-memory computing market is projected to grow from $2.72 billion in 2018 to $7.05 billion by 2023.

Apache Flink is an open-source distributed stream processing framework that is used for building real-time data processing applications. Flink is designed to handle continuous data streams and provides low-latency, high-throughput data processing. It is widely used for applications such as real-time analytics, event-driven applications, and machine learning (García et al., 2017).

5.2.4. Apache Spark Streaming

Apache Spark Streaming is an open-source distributed stream processing system that is used for building real-time data processing applications. Spark Streaming provides fault tolerance, scalability, and high throughput data processing capabilities. It is widely used for applications such as real-time analytics, fraud detection, and data processing (Salloum et al., 2016).

Each of these stream processing frameworks provides unique features and capabilities that make them suitable for different use cases. The choice of stream processing framework depends on the specific requirements of the application, including data volume, latency requirements, fault tolerance, and scalability.

5.3. STREAM PROCESSING APPLICATIONS

Following are the applications of the stream processing:

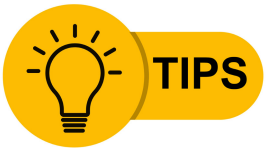
5.3.1. Real-time Analytics

Real-time analytics is a common application of stream processing systems. Stream processing systems enable organizations to perform real-time analysis on streaming data, which can provide

immediate insights and enable rapid decision-making. Real-time analytics applications include financial trading, online advertising, and supply chain management (Gulisano et al., 2015).

5.3.2. Fraud Detection

Stream processing systems are also used for fraud detection applications. By analyzing real-time data streams, stream processing systems can detect fraudulent transactions or activities in real-time. Fraud detection applications include credit card fraud detection, insurance fraud detection, and network security.



Fraud detection can be effectively implemented in streaming processing applications by using real-time analytics and machine learning algorithms.

5.3.3. Internet of Things (IoT)

Stream processing systems are an important tool for processing and analyzing data from IoT devices. By processing and analyzing data in real-time, stream processing systems can enable real-time decision-making and provide valuable insights into the behavior of IoT devices. IoT applications include smart homes, industrial automation, and smart cities (Elsaleh et al., 2020).

5.3.4. Social Media Analysis

Stream processing systems are also used for analyzing social media data in real-time. By processing social media data streams, stream processing systems can provide valuable insights into social media trends, sentiment analysis, and customer behavior. Social media analysis applications include marketing analytics, reputation management, and social media monitoring. Each of these stream processing applications provides unique insights and value to organizations that leverage real-time data processing. By using stream processing systems, organizations can enable real-time decision-making, gain competitive advantage, and provide superior customer experiences (Batrinca & Treleaven, 2015).

5.4. INTEGRATION WITH DATABASE MANAGEMENT SYSTEMS

5.4.1. Data Integration Techniques

Stream processing systems can integrate with traditional database management systems (IDMSs) using data integration techniques

such as extract, transform, load (ETL) processes, data virtualization, and data synchronization. These techniques enable stream processing systems to integrate with existing databases and data warehouses, ensuring that data is processed and stored in a consistent and reliable manner (Tang et al., 2019).

5.4.2. Real-Time Data Warehousing

Real-time data warehousing is a technique that involves integrating stream processing systems with data warehouses to enable real-time analytics on streaming data. Real-time data warehousing enables organizations to process and analyze streaming data in real-time, while also storing the data for historical analysis and reporting (Figure 5.4).

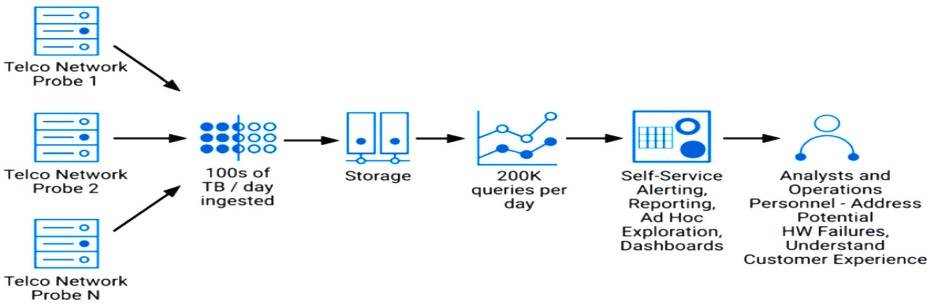


Figure 5.4. *Illustration of the real time data warehousing.*

Source: Justin Hayes, creative commons license.

5.4.3. Data Archiving and Retrieval

Stream processing systems can be integrated with data archiving and retrieval systems to ensure that data is stored and retrieved in a reliable and efficient manner. Data archiving and retrieval systems enable organizations to store large volumes of data over extended periods of time, while also providing fast and efficient retrieval of data when needed (Devaraj et al., 2020).

5.4.4. Data Security and Privacy

Data security and privacy are critical concerns for organizations that use stream processing systems. Stream processing systems can integrate with security and privacy tools such as encryption, access controls, and data masking techniques to ensure that sensitive data is protected and secure. These tools can be used to ensure that only authorized users have access to data, and

that data is stored and transmitted securely (Chen & Zhao, 2012). Integration with DBMSs is critical for stream processing systems, as it enables organizations to store, process, and analyze data in a consistent and reliable manner. By integrating with DBMSs, stream processing systems can provide valuable insights and enable real-time decision-making, while also ensuring that data is stored and processed securely and efficiently (Yang et al., 2020).

5.5. CHALLENGES OF THE STREAM PROCESSING

Following are the challenges of the stream processing:

5.5.1. Scalability and Performance

Scalability and performance are critical challenges for stream processing systems. As data volumes continue to grow, stream processing systems must be able to scale horizontally across multiple nodes in a cluster. Stream processing systems must also provide high-performance processing capabilities to enable real-time analytics and decision-making (García-Gil et al., 2017).

5.5.2. Data Consistency and Data Quality

Data consistency and data quality are critical challenges for stream processing systems. Stream processing systems must ensure that data is consistent and accurate across multiple nodes in the cluster, and that the data is of high quality. This requires robust data validation and cleansing techniques to ensure that the data is reliable and trustworthy.

5.5.3. Integration with Machine Learning

Integration with machine learning is an important future direction for stream processing systems. By integrating with machine learning techniques, stream processing systems can enable advanced analytics and predictive modeling on real-time data streams. This will enable organizations to gain deeper insights into their data and make more informed decisions (Elkhoukhi et al., 2020).

Remember

Apache Kafka is a distributed streaming platform that is widely used for building real-time data pipelines and streaming applications, providing high-throughput, low-latency messaging and the ability to process data in real-time.

5.6. FUTURE TRENDS IN STREAM PROCESSING SYSTEMS

The future of stream processing systems is focused on enabling real-time analytics and decision-making on large-scale data sets. Key future trends include the integration of stream processing systems with artificial intelligence and machine learning techniques, the adoption of edge computing for real-time data processing, and the development of new technologies for processing and analyzing data at scale (Stephanopoulos & Reklaitis, 2011) (Figure 5.5).

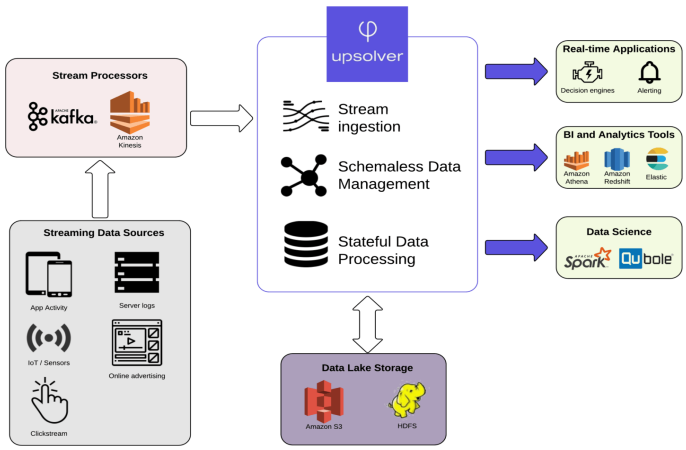


Figure 5.5. Streaming data processing trends.

Source: Upsolver, creative commons license.

As organizations continue to generate and collect large volumes of data, stream processing systems will become increasingly important for enabling real-time data processing and analytics. However, to fully realize the benefits of stream processing systems, organizations must overcome key challenges related to scalability, data quality, and integration with other technologies. By addressing these challenges, organizations can unlock the full potential of stream processing systems for real-time data processing and analytics (Ali & Abdullah, 2018).

5.7. IMPLICATIONS FOR INDUSTRY AND RESEARCH

Stream processing systems have significant implications for both industry and research. In industry, stream processing systems enable real-time data processing and analytics, which can provide organizations with valuable insights and enable rapid decision-

making. The ability to process and analyze data in real-time is particularly important in industries such as finance, healthcare, and manufacturing, where timely decision-making can have a significant impact on business outcomes.

In research, stream processing systems are an important area of study, as they enable the processing and analysis of large-scale data sets in real-time. Research in this area is focused on developing new algorithms and techniques for processing and analyzing data streams, as well as developing new architectures and systems for scaling stream processing to handle large volumes of data. There are also implications for the development of new technologies and standards for stream processing systems. For example, the development of new technologies for integrating stream processing systems with machine learning algorithms and the adoption of new data formats for stream processing. The development of new standards and technologies in this area will enable more efficient and effective stream processing, and will enable organizations to derive greater value from their data (Jämsä-Jounel, 2007).

Overall, the implications of stream processing systems for industry and research are significant, and the continued development and adoption of these systems will have a major impact on the way organizations process and analyze data (Kazemitabar et al., 2010).

ACTIVITY 5.1

An e-commerce platform wants to use a stream processing system to monitor customer behavior and personalize its offerings. What challenges might it face in doing so, and how can it overcome them?

SUMMARY

The chapter on "Stream Processing Systems in Advanced Database Systems" provides an overview of stream processing systems and their importance in advanced database systems. The chapter describes the characteristics of stream processing systems, including their ability to process continuous and real-time data streams, use in-memory processing techniques, support distributed processing, and provide scalability and fault tolerance. The chapter also discusses the major stream processing frameworks, including Apache Kafka, Apache Storm, Apache Flink, and Apache Spark Streaming, and describes their unique features and capabilities. In addition, the chapter explores the major applications of stream processing systems, including real-time analytics, fraud detection, IoT, and social media analysis. The chapter also discusses the integration of stream processing systems with DBMSs, including data integration techniques, real-time data warehousing, data archiving and retrieval, and data security and privacy. Finally, the chapter explores the challenges and future directions for stream processing systems, including scalability and performance, data consistency and quality, integration with machine learning, and future trends in stream processing systems. The chapter concludes with a discussion of the implications of stream processing systems for industry and research, highlighting the potential for real-time data processing and analytics to transform the way organizations derive value from their data.

REVIEW QUESTIONS

1. What are the characteristics of stream processing systems, and why are they important in advanced database systems?
2. Describe the major stream processing frameworks, including their unique features and capabilities.
3. What are some of the applications of stream processing systems, and how do they enable real-time data processing and analytics?
4. How can stream processing systems be integrated with DBMSs, and what are the benefits of this integration?
5. What are some of the challenges faced by stream processing systems, and how can they be addressed?
6. What are some of the future trends in stream processing systems, and how are they expected to impact the field of data processing and analytics?

MULTIPLE CHOICE QUESTIONS

1. Which of the following is a characteristic of stream processing systems?

a. Batch processing of data

b. Reliance on disk storage

c. Ability to handle continuous and real-time data streams
d. Limited scalability and fault tolerance

2. Which of the following stream processing frameworks is designed for processing real-time data streams at scale?
a. Apache Cassandra
b. Apache Kafka
c. Apache Hadoop
d. Apache Spark

3. Which of the following is an application of stream processing systems?
a. Social media marketing
b. Document management
c. Inventory control
d. Financial reporting

4. Which of the following techniques is used for integrating stream processing systems with data warehouses?
a. Real-time data synchronization
b. Extract, transform, load (ETL) processes
c. Data virtualization
d. All of the above

5. Which of the following is a challenge for stream processing systems?
a. Real-time data processing
b. In-memory processing
c. Data quality and consistency
d. Distributed processing

6. What is a future trend in stream processing systems?
a. Integration with voice recognition technologies
b. Use of blockchain for data storage
c. Adoption of edge computing for real-time data processing
d. Increased reliance on batch processing

Answers to Multiple Choice Questions

1. (c); 2. (b); 3. (a); 4. (d); 5. (c); 6. (c)

REFERENCES

1. Abd-El-Malek, M., Ganger, G. R., Goodson, G. R., Reiter, M. K., & Wylie, J. J., (2005). Fault-scalable byzantine fault-tolerant services. *ACM SIGOPS Operating Systems Review, 39*(5), 59–74.
2. Ali, A. H., & Abdullah, M. Z., (2018). Recent trends in distributed online stream processing platform for big data: Survey. In: *2018 1st Annual International Conference on Information and Sciences (AiCIS)* (pp. 140–145). IEEE.
3. Batrinca, B., & Treleaven, P. C., (2015). Social media analytics: A survey of techniques, tools and platforms. *AI & Society, 30*, 89–116.
4. Botan, I., Derakhshan, R., Dindar, N., Haas, L., Miller, R. J., & Tatbul, N., (2010). SECRET: A model for analysis of the execution semantics of stream processing systems. *Proceedings of the VLDB Endowment, 3*(1, 2), 232–243.
5. Cardellini, V., Lo Presti, F., Nardelli, M., & Russo, G. R., (2022). Runtime adaptation of data stream processing systems: The state of the art. *ACM Computing Surveys, 54*(11), 1–36.
6. Chen, D., & Zhao, H., (2012). Data security and privacy protection issues in cloud computing. In: *2012 International Conference on Computer Science and Electronics Engineering* (Vol. 1, pp. 647–651). IEEE.
7. Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Cetintemel, U., Xing, Y., & Zdonik, S. B., (2003). Scalable distributed stream processing. In: *CIDR* (Vol. 3, pp. 257–268).
8. Devaraj, A. F. S., Murugaboopathi, G., Elhoseny, M., Shankar, K., Min, K., Moon, H., & Joshi, G. P., (2020). An efficient framework for secure image archival and retrieval system using multiple secret share creation scheme. *IEEE Access, 8*, 144310–144320.
9. Elkhoukhi, H., NaitMalek, Y., Bakhouya, M., Berouine, A., Kharbouch, A., Lachhab, F., & Essaaidi, M., (2020). A platform architecture for occupancy detection using stream processing and machine learning approaches. *Concurrency and Computation: Practice and Experience, 32*(17), 5651.
10. Elsaleh, T., Enshaeifar, S., Rezvani, R., Acton, S. T., Janeiko, V., & Bermudez-Edo, M., (2020). IoT-Stream: A lightweight ontology for internet of things data streams and its use with data analytics and event detection services. *Sensors, 20*(4), 953.
11. Fox, P. T., & Friston, K. J., (2012). Distributed processing; distributed functions? *Neuroimage, 61*(2), 407–426.
12. García-Gil, D., Ramírez-Gallego, S., García, S., & Herrera, F., (2017). A comparison on scalability for batch big data processing on Apache spark and Apache flink. *Big Data Analytics, 2*(1), 1–11.
13. Gulisano, V., Nikolakopoulos, Y., Walulya, I., Papatriantafilou, M., & Tsigas, P., (2015). Deterministic real-time analytics of geospatial data streams through scale

gate objects. In: *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems* (Vol. 1, pp. 316, 317).

14. Jämsä-Jounela, S. L., (2007). Future trends in process automation. *Annual Reviews in Control, 31*(2), 211–220.
15. Kaya, Y., & Safak, E., (2015). Real-time analysis and interpretation of continuous data from structural health monitoring (SHM) systems. *Bulletin of Earthquake Engineering, 13*, 917–934.
16. Kazemitabar, S. J., Demiryurek, U., Ali, M., Akdogan, A., & Shahabi, C., (2010). Geospatial stream query processing using Microsoft SQL server StreamInsight. *Proceedings of the VLDB Endowment, 3*(1, 2), 1537–1540.
17. Liu, X., & Buyya, R., (2020). Resource management and scheduling in distributed stream processing systems: A taxonomy, review, and future directions. *ACM Computing Surveys (CSUR), 53*(3), 1–41.
18. Mishra, L., & Varma, S., (2020). Performance evaluation of real-time stream processing systems for internet of things applications. *Future Generation Computer Systems, 113*, 207–217.
19. Salloum, S., Dautov, R., Chen, X., Peng, P. X., & Huang, J. Z., (2016). Big data analytics on Apache spark. *International Journal of Data Science and Analytics, 1*, 145–164.
20. Shukla, A., Chaturvedi, S., & Simmhan, Y., (2017). Riotbench: An IoT benchmark for distributed stream processing systems. *Concurrency and Computation: Practice and Experience, 29*(21), 4257.
21. Stephanopoulos, G., & Reklaitis, G. V., (2011). Process systems engineering: From Solvay to modern bio-and nanotechnology.: A history of development, successes and prospects for the future. *Chemical Engineering Science, 66*(19), 4272–4306.
22. Stephens, R., (1997). A survey of stream processing. *Acta Informatica, 34*, 491–541.
23. Tang, S., Shelden, D. R., Eastman, C. M., Pishdad-Bozorgi, P., & Gao, X., (2019). A review of building information modeling (BIM) and the internet of things (IoT) devices integration: Present status and future trends. *Automation in Construction, 101*, 127–139.
24. Wang, G., Koshy, J., Subramanian, S., Paramasivam, K., Zadeh, M., Narkhede, N., & Stein, J., (2015). Building a replicated logging system with Apache Kafka. *Proceedings of the VLDB Endowment, 8*(12), 1654–1655.
25. Yang, P., Xiong, N., & Ren, J., (2020). Data security and privacy protection for cloud storage: A survey. *IEEE Access, 8*, 131723–131740.

CHAPTER 6

CLOUD-BASED DATABASE SYSTEMS

UNIT INTRODUCTION

The cloud database service provider customers can access database over internet, & the database can be made available to the users whenever they request it. To put it another way, a cloud database is intended for use in a virtual computer environment. Cloud database is created through cloud computing, which entails using software and hardware resources offered by a cloud computing service provider. Well, computing in the cloud is experiencing rapid expansion in the information technology sector around the world. A significant number of companies have just started the process of migrating to cloud computing and are currently accessing their data through cloud databases. According to a poll, over one-third of businesses already use cloud services to host their application servers (Agrawal et al., 2015). The phrase “cloud computing” may refer to new dimension in information technology because of the cost savings and increased speed of application performance. This trend among enterprises demonstrates that businesses will begin to rely on apps hosted in the cloud in the not-too-distant future. The cloud database is typically utilized in the form of a service. In some circles, it’s known as “Database as a Service” (DBaaS) (Mahfoud & Nouali-Taboudjemat, 2019).

A cloud database is quickly becoming one of the most popular technologies among businesses worldwide for storing massive amounts of data. Taking a relational database and installing it on a cloud server is not as straightforward as it may first appear. There is

much more to it than that. This means that new database nodes will be added online when necessary, and the overall performance of the database will be improved. It is necessary to disperse the data across a variety of data centers which are located in a variety of different geographic areas. A database should be available for the users every time to retrieve data whenever required. The database stored on the cloud ought to be simple to administer and bring down costs. To recovering information after a disaster has occurred in a database, cloud computing method is considered to be quite effective (Deka, 2013). Since both users' needs and the underlying technology's capabilities continue to evolve, new usage patterns for cloud-based databases are being developed. The customer that accessed the cloud database initially had access to the reading facility when the cloud database was first launched. Writing queries was also engaged in fulfilling the demands of the client requests. Because of the development of Web 2.0, all of this was made feasible (Samaraweera & Chang, 2019). One observation is that there are still more read requests in the database than write requests. On the other hand, in the not-so-distant future, there will be a rise in the total number of reads performed on the cloud database. Business applications are fully dependent on cloud computing. This pattern has begun to result in narrowing the gaps between write and read queries sent to cloud databases (Bagheri & Shaltooki, 2015).

Learning Objectives

At the end of this chapter, readers will be able to learn:

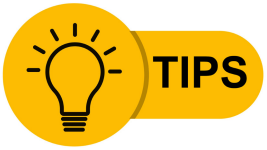
- Structure of the cloud database
- Working of nodes, nodes splitting and distributed queries
- Various services cloud database
- Concepts of data sizing, profitability, transaction capabilities, configurability and accessibility
- Challenges to the cloud database including internet speed, multi-tenancy, elastic scalability and privacy.

Key Terms

- Cloud Database
- Configurability
- Data Sizing
- Internet
- Multi-Tenancy
- Nodes
- Privacy
- Queries
- Scalability

6.1. STRUCTURE OF CLOUD DATABASE

The information regarding the locations of the various data centers is saved in the cloud database. As a result of this difference, the rational database management system (IDMS) can be differentiated from the cloud database structure. Because of this, the structure of the cloud database is somewhat complicated. Many servers, or nodes, spread out over a cloud database provide query services for on-premises and remote data centers. This connection is essential to ensure full and easy database access via cloud services. The database can be accessed in several different ways using cloud services. A user has the option of doing so by using an internet or computer, or by utilizing a mobile phone and either 3G or 4G services. To demonstrate cloud database structure, we'll take an example of BI application. Businesses can store vast volumes of data because they use BI software to store client data (Al Shehri, 2013).



Cloud databases can be offered as a managed database-as-a-service (DBaaS) or deployed on a cloud-based virtual machine (VM) and self-managed by an in-house IT team.

In this case, we'll assume that user connects to cloud database via internet and computer. The data centers, cloud data centers, and the user viewing the data are all linked by the internet, which serves as a bridge between all of these organizations. It is essential to emphasize that, in contrast to conventional databases, the cloud database makes use of a network of multiple nodes rather than a single one (Bhatti & Rad, 2017). For this application, peer-to-peer communication is preferred. Because a single node may process every user-implemented query, peer-to-peer communication is used. Despite its appearance, this type of node system has a straightforward solution: Each cloud database node has a map of the data it stores. The data for the specific query is easier to retrieve thanks to this map to the stored data (Ferretti et al., 2013).

6.1.1. Overview

After receiving a question from user via a computer, node first assesses query type and the best node to answer it. Once it has identified it, the query is sent to that specific node. The specific node responds to user and processes the request after that. For instance, after the question is submitted, it might be sent to Node 1 first, which will decide which Node is best equipped to answer it. Node 1 will send the query to Node 7, and after reviewing the data map, Node 7 may decide to store the data. Data is supplied to the user quickly and without further delay once the query has been delivered to the particular query. The cloud database's

fundamental architecture may be seen in the graphic below, which can be regarded an overview (Ellison et al., 2018) (Figure 6.1).

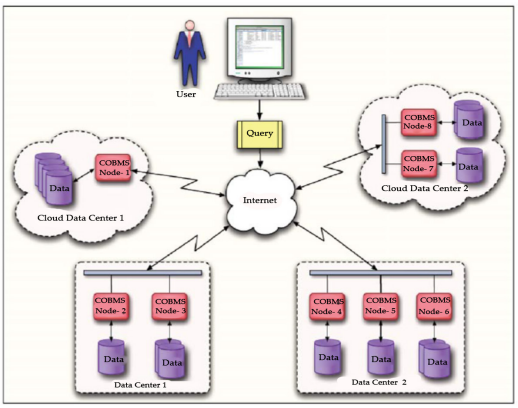


Figure 6.1. Illustration of the structure of cloud.

Source: Waleed Al Shehri, creative commons license.

6.1.2. Working of Nodes

This section will discuss how a node in the cloud DBMS functions. When a query is sent to a node, the node has two possibilities: it can either obtain the data directly from the database or from a replicating database. Both of these options are available to it. This is how a CDBMS node accesses data stored in database. It is not usually utilized because the duplicated database is only accessed in times of crisis, such as when the primary database experiences technical difficulties (Cserni et al., 2003). The node mostly uses database access to speed up data retrieval. In CDBMS, the applications store the application data. The data from files are directly accessed by CDBMS. When a node retrieves information via direct access, it logs a file's metadata as a "map." The following diagram illustrates how a node functions within a cloud-based database administration system (Berney et al., 2011) (Figure 6.2).

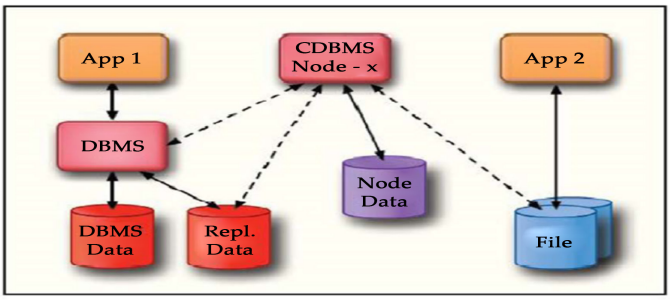


Figure 6.2. Schematic of the working of nodes.

Source: Macquarie University Sydney, creative commons *license.*

Above diagram shows how node operates to receive data from files and DBMS data. Also, CDBMS will continue to maintain its database for storing information that the nodes frequently use. Performance of CDBMS is improved.

6.1.3. Node Splitting

Accessing local data in a single-location data center with a capacity for terabytes of data is straightforward for BI applications without any compromise in database performance. On the other hand, it will be incredibly difficult to handle the increasing volume of enquiries if the cloud system is supposed to handle this much volume. For this, the cloud DBMS may employ a complex technique. A data center, on the other hand, has several nodes, and each data center at a distinct geological position may have multiple nodes (Valkenhoef et al., 2016).

Mostly larger firms, like supermarkets, use cloud services for their database. Although it is a wonderful strategy, it has scaling issues. Because the cloud database must process a huge number of queries, the CDBMS may face performance issues. Despite the fact that the cloud DBMS is known to have a huge number of nodes, this number may not always be adequate. The majority of time, total number of searches keeps climbing upward. It's essential that a response be given to the recent onslaught of questions. In order to accomplish this, CDBMS swiftly starts a new node that distributes the workload associated with database querying (Wynne-Jones, 1993).

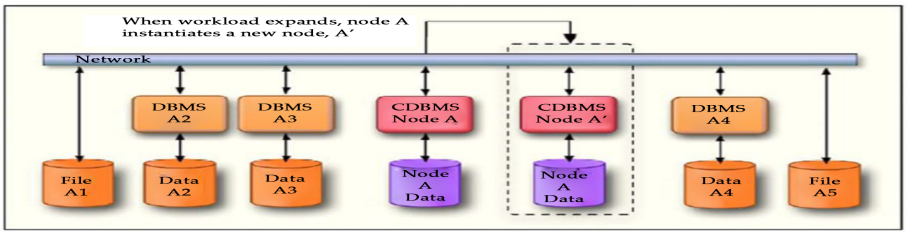
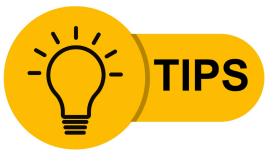


Figure 6.3. Illustration of the nodes splitting.

Source: Al Shehri, creative commons license.

The following example should be used to further illustrate this concept. Node A in the diagram maintains the database's data along with Nodes A2, A3, & A4, whereas Nodes A2 & A3 handle the files of Nodes A1 & A5. When a node receives an excessive number of inquiries and becomes bothersome, it divides into new node that'll share queries with original node. Following splitting

process, original Node A will handle data for file A1 and databases A2 and A3. The new Node A' will take care of A4 & A5, on the other hand. This is a really clever technique, and the ability of cloud databases to scale makes it possible to handle enormous volumes of data (Janssen & Corporaal, 1997) (Figure 6.3).



Split nodes are used to split the previous incoming node into multiple outgoing nodes (or outputs), as opposed to flowing into a single output.

Thanks to this node-splitting approach, the CDBMS continues to function well despite the growing number of queries. As the total number of requests processed by a node increases, a greater proportion of it will be comprised of splitting nodes, which help to more evenly spread the workload imposed by those questions. When Node A divides into the identical Node A', it will keep a record of all of the tasks that were distributed before the division. This record makes it simpler to distribute questions to the appropriate parties (Liu et al., 2014).

The splitting function of the cloud database makes it simpler to manage many queries and maintain database speed. Once the questions have been resolved, the split requests are reconnected to the Main Node A.

6.1.4. Distributed Queries

A large company's database might contain information about its products, clients, employees, and corporate policies. Many searches could be used to find this information (Kossmann, 2000). These many CDBMS entities could be used in a wide range of applications. Answer all questions; many nodes may be involved. There are several methods for storing data in DBMSs in CDBMS, including query-oriented databases and column store databases. Yet, most effective database management strategy is to use distributed queries.

The term "distributed query" refers to inquiries that are networked and each contact a separate distributed node to retrieve data. There could be a large number of responses given the number of inquiries that have been made (Yu & Chang, 1984). Towards the conclusion, each one of them is joined together, much like the dispersed solutions.

Consider a distributed query that has been further subdivided into sub-queries. TA computer generates the inquiry and sends it over the internet. It is then divided into smaller enquiries, each of which is routed to a different node. In the following example, Sub Query 1 & Sub Query 2 are routed to CDBMS Node 2. Subquery

Node 3 is moved to Node 5, while Subquery Node 8 is moved to Node 6. After nodes are resolved, the answers are likewise returned in dispersed form. The responses from the nodes are then consolidated and returned to user (Cai & Shi, 2020) (Figure 6.4).

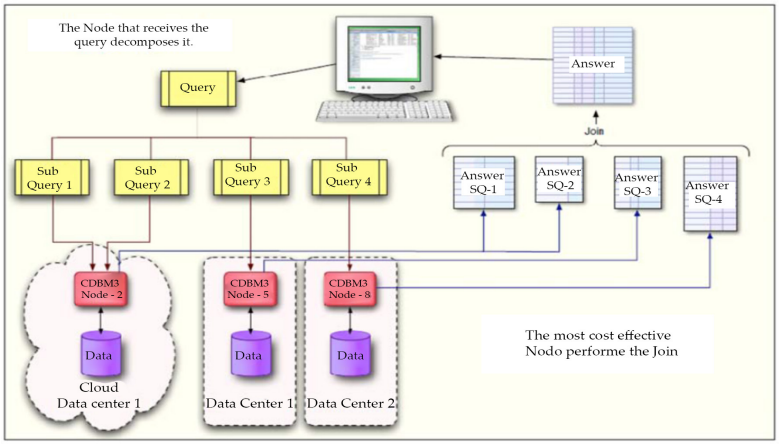


Figure 6.4. Schematic of the splitted queries.

Source: Al Shehri, creative commons license.

6.2. CLOUD DATABASE SERVICE

Several cloud database service providers at work provide databases as a service, which are further classified into three types. Virtual machines with locally installed database applications like SQL are available in three different database types: logical, irrational, and operational databases.

Several companies, including Microsoft SQL Azure, Amazon RDS, Google AppEngine Datastore, & Amazon SimpleDB, offer Database as a Service (DBaaS) (Tan et al., 2017). Depending on the quality and type of services given, each service provider is different. To select the ideal service for your company, a few criteria may be used. This is not limited to a particular business; depending on the requirements of any firm, these criteria may help in choosing the best service provider.

6.2.1. Choosing Best DBaaS

The requirements of the organization and the services offered by the company have an impact on the decision to use DBaaS. To choose the best DBaas, a few things can be referred to.

6.2.2. Data Sizing

Depending on the DBaaS provider, a database can hold a range of data sizes. The size of the data is critical since the company needs to be certain that the data will fit in its database. For instance, Amazon RDS enables users to store up to 1TB of data in a single database, whereas SQL Azure only gives 50GB for one database (Ferretti et al., 2018).

Did you Know?

According to a report, the global cloud database market is expected to grow from $2.13 billion in 2016 to $14.05 billion by 2023.

6.2.3. Portability

Users need to have access to the database at all times; hence it needs to be portable. If the service provider goes out of business, the database and the data it contains can be destroyed. If such occurrences take happen, an emergency plan should be in place. Using cloud services from other companies can solve this problem and guarantee that the database is accessible even during an emergency.

6.2.4. Transaction Capabilities

Because it is essential for the user to finish the transaction, the cloud database's transaction capabilities are its most important feature. The success or failure of the transaction must be communicated to the user. Some companies deal largely in financial transactions, in which case entire read-&-write tasks must be performed. When a user requests guarantee for a transaction he made, that transaction is known as an ACID transaction (Al Shehri, 2013). If a guarantee is not necessary, transactions can be carried out using non-ACID transactions. Along with being quicker.

6.2.5. Configurability

Many databases can be readily changed by the user because the service provider handles the majority of setup. The database administrator has limited options and may easily manage the database using this technique.

6.2.6. Database Accessibility

There are many different ways to access databases since there are so many of them. The first strategy involves making an RDBMS

accessible using drivers compliant with industry standards, such as Java Database Connectivity. The purpose of this driver is to make it possible for an external connection to access services via a regular connection. The second approach to database accessibility involves using interfaces or protocols as Service-Oriented Architecture (SOA), SOAP, & rest (Bhatti & Rad, 2017). These interfaces leverage HTTP & a novel API concept.

KEYWORD

Cloud storage is a model of computer data storage in which the digital data is stored in logical pools, said to be on "the cloud".

6.2.7. Certification and Accreditation

It is recommended to use a cloud database supplier with certification and accreditation on their resume. It helps the company lower service risks to avoid any disruptions. Businesses with certifications like FISMA might be viewed as respectable when compared to other DBaaS suppliers.

6.2.8. Data Integrity, Security, and Storage Location

Security has been biggest threat to data stored in cloud storage. A security also depends on places where data is stored and encryption methods used (Sun et al., 2014). Data is housed in many locations at data centers.

6.3. CHALLENGES TO CLOUD DATABASE

A cloud database's implementation raises various challenges for its successful use. Cloud databases are becoming the ideal option for enterprises despite these challenges. The difficulties with cloud computing are outlined below.

6.3.1. Internet Speed

The data transfer speed is relatively very quick when compared to internet speed utilized to connect to data center. Performance of cloud database is hampered by this. This has an effect on how well cloud database functions . How quickly queries are sent to database and data is received from the data center depend on the internet's speed. The solution to this issue is faster speed lines, however these are extremely expensive and defeat the purpose of using a cloud database (Januzaj et al., 2015).

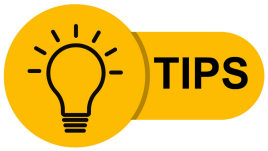
6.3.2. Query and Transactional Workloads

Remember

Low internet speeds can pose a challenge to cloud database systems, affecting the speed and reliability of data transmission and access.

The effort associated with queries and transactions are extremely dissimilar. While discussing the transactional burden, we can determine how long it will take. We are unable to do so while talking about the query workload, though. The burden is determined by the amount of queries, however it is unknown how many people will be using the database.

6.3.3. Multi-Tenancy



Multi-tenancy can pose challenges to cloud databases, especially when it comes to ensuring data privacy and security for different users sharing the same infrastructure.

The most important question is how to maximize the performance of the available system, despite the fact that a workload and database may need to be handled. In this regard, it is necessary to decrease the number of machines while retaining efficiency. An amount of hardware resources required for each task should be understood by system. The devices on which the workloads are distributed and the joining method used can be identical. The ideal solution is to create virtual machines for each database and to build several virtual machines on the same system (Abd Elmonem et al., 2016). Additional equipment is required. Assume that the same task needs to be split between two or three machines. The performance and speed ultimately suffer a 6–10-fold decline as a result. Each virtual machine has its own operating system and database, which results in the decreased performance. When these two essential components are split, each virtual machine has its own buffer loop. It would be desirable to use the same database server across numerous machines since this would increase performance.

6.3.4. Elastic Scalability

ACTIVITY 6.1

A large corporation is migrating its legacy database system to the cloud. What steps should it take to ensure the security and privacy of its data during and after the migration?

When we're talking about cloud databases, any workload can be managed by a good cloud database. Yet, the problem in a cloud database arises when workload exceeds system capacity. Cloud database must be able to scale out on its own as workload grows. The database's scaling out helps cloud database achieve its best performance and efficiency.

6.3.5. Privacy

Privacy has been the primary issue with cloud computing. Cloud

computing is more advanced in terms of accessibility to consumers as well as hackers who desire to break into the system. It's critical that the cloud database, which houses client records for organizations, is secure. The companies cannot risk having their database's data leak. If the database has data encryption, securely storing data is relatively easy to do (Narasayya & Chaudhuri, 2021).

SUMMARY

The idea of a cloud database has been introduced in this chapter, along with some of its key features. For a number of reasons, businesses have started utilizing cloud computing. Cloud computing services are becoming increasingly popular as an alternative to the establishment of individual data centers by each company or organization. These services allow for improved and more rapid access to information. Businesses are often looking for new approaches that may meet their needs while remaining cost-effective. The method is utilized across the database. Before it, every firm maintained its own data center as well as a traditional database. Database Administration as a Service is a recent innovation in the field of cloud database (DBaaS). This makes it possible for companies and organizations to make use of resources provided by DBaaS providers while at the same time continuing to make investments in and maintain software and hardware for their own data centers, which are responsible for housing the entirety of database. They use the services of DBaaS suppliers and value independence of a continually available database. There are advantages and disadvantages to using a cloud database, but adoption has proved that the advantages outweigh the disadvantages. The provision of database services via the cloud has brought about a number of benefits, among which various enterprises are competing. The business went with the solution that was the most suitable to its requirements.

REVIEW QUESTIONS

1. What is a cloud database and how does it differ from traditional databases?
2. Describe the structure of a cloud database and how it enables distributed queries.
3. What are some of the services provided by a cloud database and how can they benefit businesses?
4. Explain the concepts of data sizing, profitability, transaction capabilities, configurability, and accessibility in relation to a cloud database.
5. What are some of the challenges to implementing a cloud database and how can they be addressed?
6. How can businesses ensure the privacy and security of their data in a cloud database?

MULTIPLE CHOICE QUESTIONS

1. What is the structure of a cloud database?

a. It is a hierarchical structure
b. It is a network structure
c. It is a relational structure
d. It is a distributed structure

2. **What is nodes splitting in a cloud database?**
   a. Dividing the database into smaller databases
   b. Dividing the database into smaller tables
   c. Dividing the database into smaller chunks of data
   d. Dividing the database into smaller servers
3. **Which of the following is not a service provided by a cloud database?**
   a. Data backup and recovery
   b. Data warehousing
   c. Data analysis
   d. Data destruction
4. **What is data sizing in a cloud database?**
   a. The amount of data a database can store
   b. The speed at which data can be retrieved from the database
   c. The cost of storing data in the database
   d. The security of the data stored in the database
5. **What is a challenge to the cloud database related to multi-tenancy?**
   a. Ensuring that data is accessible only to authorized users
   b. Ensuring that data is stored securely
   c. Ensuring that resources are shared fairly among tenants
   d. Ensuring that the database can scale as needed

Answers to Multiple Choice Questions

1. (d); 2. (c); 3. (d); 4. (a); 5. (c)

REFERENCES

1. Abd, E. M. A., Nasr, E. S., & Geith, M. H., (2016). Benefits and challenges of cloud ERP systems–a systematic literature review. *Future Computing and Informatics Journal, 1*(1, 2), 1–9.
2. Agrawal, D., El Abbadi, A., & Salem, K., (2015). A taxonomy of partitioned replicated cloud-based database systems. *IEEE Data Eng. Bull., 38*(1), 4–9.
3. Al Shehri, W., (2013). Cloud database as a service. *International Journal of Database Management Systems, 5*(2), 1.
4. Bagheri, H., & Shaltooki, A. A., (2015). Big data: Challenges, opportunities and cloud based solutions. *International Journal of Electrical and Computer Engineering, 5*(2), 340.

5. Berney, D. M., Wheeler, T. M., Grignon, D. J., Epstein, J. I., Griffiths, D. F., Humphrey, P. A., & Srigley, J. R., (2011). International society of urological pathology (ISUP) consensus conference on handling and staging of radical prostatectomy specimens. Working group 4: Seminal vesicles and lymph nodes. *Modern Pathology, 24*(1), 39–47.
6. Bhatti, H. J., & Rad, B. B., (2017). Databases in cloud computing: A literature review. *International Journal of Information Technology and Computer Science, 9*(4), 9–17.
7. Cai, Z., & Shi, T., (2020). Distributed query processing in the edge-assisted IoT data monitoring system. *IEEE Internet of Things Journal, 8*(16), 12679–12693.
8. Cserni, G., Amendoeira, I., Apostolikas, N., Bellocq, J. P., Bianchi, S., Bussolati, G., & European Working Group for Breast Screening Pathology, (2003). Pathological work-up of sentinel lymph nodes in breast cancer. Review of current data to be considered for the formulation of guidelines. *European Journal of Cancer, 39*(12), 1654–1667.
9. Deka, G. C., (2013). A survey of cloud database systems. *IT Professional, 16*(2), 50–57.
10. Ellison, M., Calinescu, R., & Paige, R. F., (2018). Evaluating cloud database migration options using workload models. *Journal of Cloud Computing, 7*, 1–18.
11. Ferretti, L., Colajanni, M., & Marchetti, M., (2013). Distributed, concurrent, and independent access to encrypted cloud databases. *IEEE Transactions on Parallel and Distributed Systems, 25*(2), 437–446.
12. Ferretti, L., Marchetti, M., Andreolini, M., & Colajanni, M., (2018). A symmetric cryptographic scheme for data integrity verification in cloud databases. *Information Sciences, 422*, 497–515.
13. Janssen, J., & Corporaal, H., (1997). Making graphs reducible with controlled node splitting. *ACM Transactions on Programming Languages and Systems (TOPLAS), 19*(6), 1031–1052.
14. Januzaj, Y., Ajdari, J., & Selimi, B., (2015). DBMS as a cloud service: Advantages and disadvantages. *Procedia-Social and Behavioral Sciences, 195*, 1851–1859.
15. Kossmann, D., (2000). The state of the art in distributed query processing. *ACM Computing Surveys (CSUR), 32*(4), 422–469.
16. Liu, X., Song, M., Tao, D., Liu, Z., Zhang, L., Chen, C., & Bu, J., (2014). Random forest construction with robust semisupervised node splitting. *IEEE Transactions on Image Processing, 24*(1), 471–483.
17. Mahfoud, Z., & Nouali-Taboudjemat, N., (2019). Consistency in cloud-based database systems. *Informatica, 43*(3), 3–10.
18. Narasayya, V., & Chaudhuri, S., (2021). Cloud data services: Workloads, architectures and multi-tenancy. *Foundations and Trends® in Databases, 10*(1), 1–107.
19. Samaraweera, G. D., & Chang, J. M., (2019). Security and privacy implications on database systems in big data era: A survey. *IEEE Transactions on Knowledge and Data Engineering, 33*(1), 239–258.

20. Sun, Y., Zhang, J., Xiong, Y., & Zhu, G., (2014). Data security and privacy in cloud computing. *International Journal of Distributed Sensor Networks, 10*(7), 190903.
21. Tan, D. P., Li, L., Zhu, Y. L., Zheng, S., Ruan, H. J., & Jiang, X. Y., (2017). An embedded cloud database service method for distributed industry monitoring. *IEEE Transactions on Industrial Informatics, 14*(7), 2881–2893.
22. Van, V. G., Dias, S., Ades, A. E., & Welton, N. J., (2016). Automated generation of node-splitting models for assessment of inconsistency in network meta-analysis. *Research Synthesis Methods, 7*(1), 80–93.
23. Wynne-Jones, M., (1993). Node splitting: A constructive algorithm for feed-forward neural networks. *Neural Computing & Applications, 1*, 17–22.
24. Yu, C. T., & Chang, C. C., (1984). Distributed query processing. *ACM Computing Surveys (CSUR)*, *16*(4), 399–433.

CHAPTER 7

MAIN MEMORY DATABASE SYSTEM

INTRODUCTION

Data is permanently stored in the main physical memory in an (MMDB) main memory database system, whereas it is disc resident in a traditional database system (DRDB). An MMDB may contain a backup copy of the memory-resident data on a disc, but a DRDB may cache disc data into memory for access. An item can therefore have copies in both memory and disc in both scenarios. The fundamental distinction is that in MMDB, the primary copy stays in memory permanently. This has significant ramifications regarding how structure and access will be addressed (Garcia-Molina & Salem, 1992) (Figure 7.1).

As chip densities rise and semiconductor memory becomes more affordable, it becomes practical to store ever-larger databases within memory, making massively multi-dimensional databases (MMDBs) a reality. Compared to DRDBs, MMDBs can offer substantially faster response times and transaction throughputs because data can be retrieved directly in memory. This is crucial for real-time applications since transactions must be performed by their given timeframes (Larson & Levandoski, 2016).

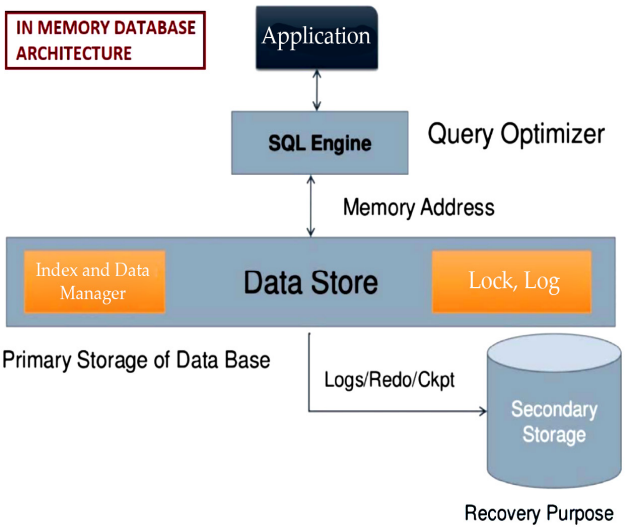


Figure 7.1. Illustration of the main memory database system.

Source: Kodamasimham, creative commons license.

Learning Objectives

At the end of this chapter, readers will be able to learn:

- Difference between the main memory and magnetic disks
- Applications of the partition of the data
- Frequency of backups in main memory database
- Impact of memory resident data including concurrency control, commit processing, access methods etc.

Key Terms

- Access Methods
- Backups
- Commit Processing
- Concurrency Control
- Data Partitioning
- Indexing
- Locking
- Magnetic Disks
- Main memory
- Querying

7.1. DIFFERENCE BETWEEN MAIN MEMORY AND MAGNETIC DISKS

Magnetic discs and main computer memory have different characteristics, and these distinctions significantly impact the layout and functionality of database systems. However, these distinctions are generally known, but it is still important to review them (Mittal & Vetter, 2015).

1. Main memory has a ten times quicker access time than disc storage.
2. Disk storage is typically not volatile, although main memory typically is. Nevertheless, the nonvolatile main memory can be created (at a price).
3. The cost of accessing a disc is high and fixed, not based on the volume of data retrieved. Disks have block-oriented storage devices as a result. Block-oriented memory is not the main memory.
4. Because sequential access to such a disc is faster than random access, the data structure on a disc is significantly more important than the layout of information in the main memory. In major memory, sequential access is less crucial.
5. Disks are typically not directly accessible by processor(s), but the main memory is. Main memory data is more rendered to software faults than disk-resident data.

These variances impact nearly all facets of database management, including concurrency control and application interfaces. This study will address these consequences and briefly overview various newly developed or put-into-use MMDBs. But before we begin, it's crucial to answer three often-asked issues about MMDBs (Mittal & Vetter, 2015).

Presume that the main memory can hold the complete database. For some applications, yes. In some circumstances, the database has a finite size or is expanding more slowly than memory can support. For instance, it is realistic to anticipate because memory can store hundreds of thousands of bytes per worker or client, regardless of the firm's success. The database volume may also be proportionate to the number of customers or employees in a company. The database will naturally be smaller than accessible memory in some real-time applications where the data has to be memory residence to fulfill the real-time requirements. Real-time applications include those in telecommunications (where 8()0

Did you get it?

The first commercial digital disk storage device was the IBM 350 which shipped in 1956 as a part of the IBM 305 RAMAC computing system.

telephone digits must be converted to actual numbers), radar tracking (where object signatures must be compared to a database of known planes), & securities trading (e.g., trading opportunities must be discovered and executed before they disappear) (Litwin & Risch, 1992).

There are, however, situations in which the database cannot be made to fit in memory, such as an application that uses satellite image data. DRDB will be crucial in these situations. However, even in these extremely large applications, it is typical to explore various classes of data, including hot data—which is frequently accessed, typically low volume, and with strict timing requirements—cold data, which is accessed infrequently but is more substantial, as well as various intermediate degrees. If so, the data can be divided into one or even more logical databases, with the most active one being kept in the main memory. Then, we have a group of databases, some of which are handled by an MMDB and others by a DRDB (Chou, 1997). The database systems may be closely integrated into facilities for fully automated data migration through one database system to another as the frequency of data access changes, or they may be completely disjointed, in which case applications access them similarly to how they would access a loose federation of database systems. Be aware that this migration involves more than just storing values. A temporary duplicate of an object is created through caching at a different level in the storage hierarchy. When an object is migrated, a new management system is used, & its structure and access controls may change (Ng, 1992).

Did you Know?

Data partitioning can help improve query performance and reduce resource contention in databases.

7.2. APPLICATIONS OF DATA PARTITION

This data partitioning naturally occurs in many applications. For instance, account records (such as those holding balances) are frequently hot, while customer records (such as address and mother's surname) are typically colder. Bank activity data from the past are generally sparse. In some network switching applications, route tables are cold data for monthly customer statements and hot data for mapping 800 phone numbers into actual numbers (Ke et al., 2015) (Figure 7.2).

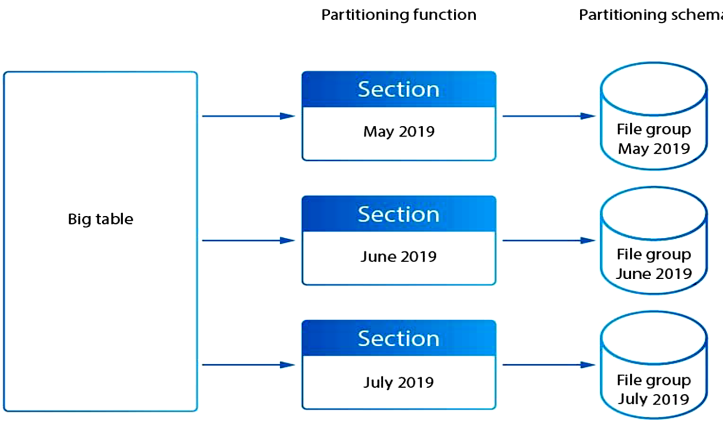


Figure 7.2. Schematic of the data partition.

Source: Data Sunrise, creative commons license.

For many years, IMS, one of the first database systems, has offered two systems: Fast Route for memory-resident data and traditional IMS for everything else. This is because IMS recognized these access disparities early on. A recent study also covers some of the problems associated with data movement and multilevel database systems. Whenever we refer to "in database" in the remaining sections of this work, we mean the portion of the total amount of data that MMDB will permanently manage (Panda et al., 2000).

What distinguishes an MMDB from a DRDB with only a sizable cache? Copies of data will always be present in memory if the DRDB's cache is large enough. Even while such a system may function adequately, it is not fully utilizing the RAM. Even though the data is in memory, the index structures (like B-trees) will be created for disc access. Moreover, apps might need a buffer manager to access data as if on a disc. For instance, whenever a particular tuple needs to be accessed, its disc address must be determined. The buffer manager must then be called to see whether the associated block is in memory. The tuple would be copied through an applications tuple buffer and examined there after the block has been located. It is more effective to refer to a record by its memory address if it remains in memory (Mahmud et al., 2020).

One of several potential in-memory enhancements. Certain DRDB and some (OOSS) object-oriented storage systems are starting to adopt some of the in-memory optimizations of MMDB as they realize that with big caches, part of their data would frequently remain in memory. For instance, some modern systems allow apps a direct pointer to a tuple or object's in-memory representation. (This

process is known as “swizzling”) The in-memory improvements that DRDB performs bring them closer to MMDB. Anticipate that the distinctions between an MMDB & DRDB will no longer exist because any competent database administration system will acknowledge and use that certain data will remain in memory permanently and must be managed as such (Khaleghzadeh et al., 2018) (Figure 7.3).

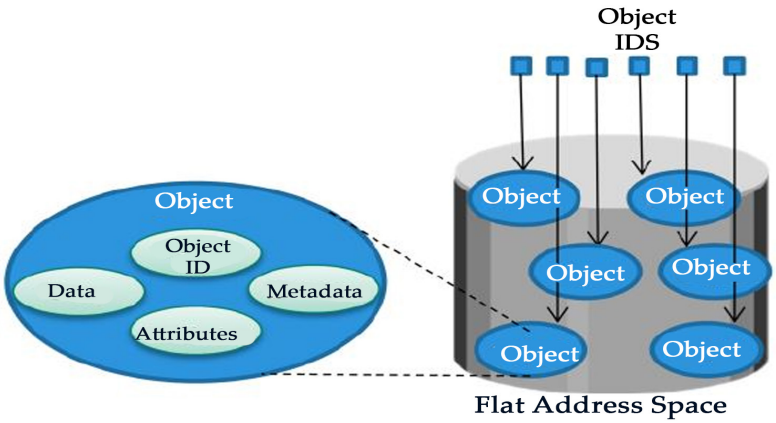


Figure 7.3. Illustration of the object-oriented storage systems.

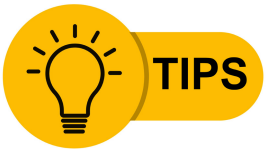
Source: Anil Ommi, creative commons license.

Can we rely on reliable and nonvolatile main memory by installing specialized hardware? This assumption is alluring since it would significantly streamline MMDB architecture and boost performance (there would be zero crash recovery code at all!). There isn’t a “yes” or “no” response to this. Simple methods like battery-backed memory boards, uninterruptible power supplies, error-detecting, even correcting memory, & triple modular redundancy can increase the reliability of memory, which is merely a storage medium. Yet, this does not eliminate the possibility of media failure; it simply lowers it. As a result, one will always need a database backup, most likely on a disc. Remember that backups for DRDB are also necessary, potentially to tape other drives (Morrison et al., 2013).

7.3. FREQUENCY OF BACKUPS

The number of backups may decrease along with the likelihood of media failure, which can also have a negative impact on performance. For instance, tape backup for good drives may be sufficient once a week. The cost of once-weekly scanning and copying the full database (perhaps while doing other things) shouldn’t be too high. Several circumstances increase the necessity of backups for memory resident data (Li et al., 2020).

1. Operating system mistakes are more likely to occur in memory because it is available directly by the CPU. The memory content will occasionally be lost whenever the system "crashes," even if the hardware is extremely reliable.
2. Repairing a failed disc without damaging the data on any other discs is possible. Only a portion of the database (upon the disc) must be restored from backup after recovery (and logs). Furthermore, the remainder of the database might still be available during recovery. The whole machine must normally be turned off when a memory board breaks, meaning the entire database is lost. A recent backup is preferable since recovering the data will take significantly longer. (The earlier the backup, the longer it requires to bring up-to-date out from the log)
3. Compared to discs, battery-backed memory or uninterruptible (UPS) power supplies are "active" devices with a higher data loss risk. Disks are "passive," meaning they don't need any action to preserve data. A UPS may overheat or run out of power. Batteries could leak or stop holding their charge (Qin et al., 2002).



Regardless of the repository model that is used, the data has to be copied onto an archive file data storage medium.

7.4. IMPACT OF MEMORY RESIDENT DATA

Information kept in a computer's memory and easily accessed by the CPU for processing is called memory resident data. Depending on the situation, memory-resident data may have different effects. Having data stored in memory permanently can considerably increase system performance, which is one of its key benefits. Data that is kept in memory is easily accessed by the CPU, which shortens the processing time. This could lead to quicker application responses, more fluid video playback, and enhanced system performance (Bakerjian et al., 2020).

When the system has to retrieve commonly used information rapidly, memory-resident data could be helpful. For instance, often requested data can be cached in memory in the database management system (IDMS) to minimize the amount of disc reads needed. This may result in quicker query reaction times and enhanced database performance (Salem & Garcia-Molina, 1990).

The drawback of having resident data in memory is that it consumes some memory, which might impact system resources. The performance may be slowed or crash if resident data takes too much memory. Moreover, improper memory management by

the system can lead to memory leaks, which can make the system unstable over time (Manegold et al., 2000).

7.4.1. Concurrency Control

Transactions may anticipate finishing more rapidly in a main memory system because main memory access is much faster than disc access. Lock contention might not be as significant in systems that use lock-based concurrency management as if the data is stored on a disc because locks won't be held for as long. (In this section, we concentrate on locked concurrency control as it has been utilized in MMDB prototypes and is the most widely employed in practice. Other kinds of mechanisms, like optimistic or time-stamp-type things, could benefit from improvements comparable to the ones that will be detailed.) Systems utilize small lock granules (records or fields) to lessen conflict. The main benefit of using small lock granules is eliminated if there is already little contention because data is memory resident. Subsequently suggested that very big lock granules, such as relations, are best suited for memory-resident data because of this (Ulusoy & Buchmann, 1998) (Figure 7.4).

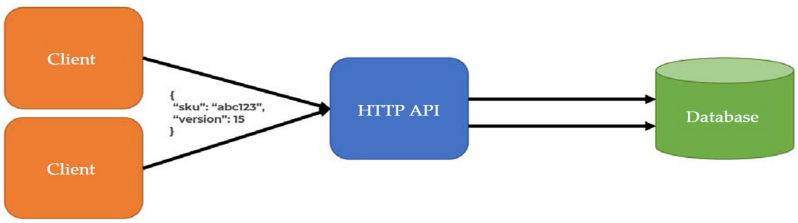


Figure 7.4. Schematic of the optimistic concurrency control.

Source: Derek Comartin, creative commons license.

In extreme cases, the entire database could be selected as the lock granule. Transactions are being executed serially as a result. Since the expenses of access controls (releasing and setting locks, dealing with a deadlock) are now almost avoided, serial processing data is extremely desirable. Additionally, there are far fewer CPU cache flushes. (If a transaction is held up while awaiting a lock, a payment update must be started, and the information in the CPU cache should be updated.) Just one flush is necessary for each transaction during serial execution.) Gains can be substantial on high-performance machines where cache flushes are comparable to thousands of instructions. When lengthy transactions (such as conversational transactions) are involved, serial transactions are likely impractical. It should be possible to perform brief transactions alongside long-lived ones fairly.

Furthermore, even when all transactions are brief, multiprocessor systems may still need some concurrency control (Garcia-Molina & Salem, 1992).

The actual implementation of the locking method can be tailored to the objects that will be locked's memory location. Locks are implemented in a typical system using a hash table with entries for currently closed objects. The items (on the disc) don't have any lock information. We can spare just a few bits to describe the objects' lock status if they are in memory (Gruenwald & Liu, 1993).

Remember

Concurrency control mechanisms are crucial for ensuring consistency and reliability in main memory databases, where multiple transactions may be accessing the same data concurrently.

7.4.2. Commit Processing

Maintaining a backup copy (see Part I) or keeping track of transaction activity is essential to guard against media failures. This log must be stored in stable media because memory is typically volatile (e.g., redundant disks). Activity entries for a transaction must be added to the log before it can commit.

The performance benefits obtained with memory-resident data are in danger of being undermined by the requirement for a steady log. Because every transaction should wait for at least one reliable write before committing, logging may slow down response times. If logging becomes a bottleneck, performance may also be impacted. The logging reflects the only disc operation each transaction would need. Therefore, even if these issues exist when data is disc resident, they are worse in main memory systems (Levy & Silberschatz, 1992).

This issue has received several remedies. Initially, a component of the log can be stored in a tiny quantity of stable main memory. A transaction is completed by quickly writing the transaction log information into stable memory. Then, a unique procedure or processor is responsible for copying data from stable memory to a log drive. Although a log bottleneck cannot be fixed, the response time issue can be resolved because transactions no longer need to wait for disc operations when using stable memory. Research indicates that even in high-performing systems, just a modest amount of stable memory (e.g., less than 100 log pages) is required to hold a log tail (Kallman et al., 2008).

Transactions could be committed if stable memory isn't available for log tail. By relieving a total transaction locks as rapidly as its log entry is added to the log, rather than waiting for information to be transmitted to the disc, pre-committing is done. Because

the log is sequential, transactions that depend on others cannot commit before those that rely on them. Recommitting a transaction sometimes doesn't speed up a transaction's response time, but it can speed up other concurrent transactions' responses by reducing blocking delays (Pandis et al., 2010).

The group might be used to alleviate a log bottleneck. A transaction's log records don't have to be delivered to the log disc immediately after it commits when using group commit. Instead, permitted the memory to build with the records of numerous transactions. All are flushed towards the log disc in a single disc transaction once enough have collected (for example, when a webpage is full). Because a single operation committed many transactions, group commit lowers the overall number of operations the log discs must do (Bhide & Stonebraker, 1988).

7.4.3. Access Methods

Remember

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done.

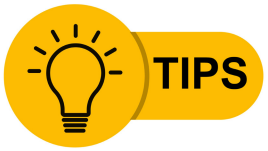
Index structures made for block-oriented storage, such B-Trees, lose much of their attractiveness in a main memory database. For main memory databases, a wide range of index structures have already been put forth and assessed. They include several kinds of trees and hashes. Although hashing offers quick lookup and update, it can't be as space-efficient as a tree and is incompatible with range queries. For memory-resident databases, specific trees like the T-Tree have been created. While traversing deeper trees in main memory is significantly faster than on a disc, main memory trees do not necessarily require the short, bushy structure of B-Tree.

The data values from which an index is created may not necessarily need to be stored as in the index itself, just like the case with B-Trees, which is a common observation throughout all main memory space methods. Pointers can be tracked rapidly in main memory since random access is quick. As a result, rather than storing the actual data, index structures can hold pointers to indexed data instead. As long as the pointers are less than the data they refer to, this solves the issue of putting variable length fields through an index (Lindström et al., 2013).

Pointers show that inverting the relation upon this index field is arguably the easiest way to give an index. The inverted "relation" in a main memory database could be nothing more than a list of sorted tuple pointers. Although updates are slow, these indexes are relatively space-efficient and reasonably quick for exact-match queries and range searches.

7.4.4. Data Representation

Databases that store data in main memory can also benefit from effective pointer following. A collection of pointers with data values can describe relational tuples. When large numbers appear several times within the database, using pointers saves space because only one copy of the value has to be saved. Because variable-length data could be represented via pointers together in a heap, pointers also facilitate the processing of variable-length fields (Mastykash et al., 2018).



Efficient data representation is crucial for optimizing performance in main memory database systems.

7.4.5. Query Processing

Memory-resident database, sequential access is not noticeably faster than random access. Therefore, query processing strategies that benefit from faster sequential access lack that advantage. For instance, sort-merge join processing sorts the joined relations to create sequential access first. Although pointer lists might be used to represent the sorted relations in the main memory database conveniently, there is no need for this as most of the sorting's benefits have already been neglected (Faerber et al., 2017).

Certain relational operations could be carried out relatively quickly whenever relational tuples are constructed as a collection of pointers to data values. For instance, combining relations R and S is needed because they share the attribute A. Scan the smaller relations, let's say R, to complete the join. Follow each tuple's A pointer to the actual value, referred to as at. T race it back pointers to every S tuple using as from that value. Combining the initial R tuple with these S tuples produces the final product. Enough pointers are required; to effectively guide us toward the S tuple, which utilizes these values for its A value for this to work. Although some extra storage will be needed, as stated in, substantial performance advantages may exist. The main concept is that since data is still in memory, it is easy to create suitable, compact data structures which can accelerate queries (Kemper et al., 2012).

While most conventional systems aim to reduce disc access, query processors for memory-resident data must concentrate on processing costs. One issue is that it might be challenging to quantify processing costs with a complicated data management system. It is necessary to first identify costly actions (such as building a copying data or an index) before developing techniques to lessen

their frequency. Operating costs can differ greatly amongst systems. Therefore, an optimization method that performs well in 1 design may not perform well in another (Friedrichs & Wolynes, 1990).

7.4.6. Data Clustering and Migration

Data items (such as tuples and fields) frequently retrieved together are clustered in a DRDB. For instance, if a "department" and its "workers" are commonly viewed by queries, the personnel records can be kept on the same disc page. Naturally, there is no requirement to cluster objects in an MMDB. The parts of an item may be distributed in memory (e.g., tuples only have pointers towards data values stored elsewhere).

A challenge is raised that does not exist in traditional systems: how and where should an object be saved when it is to move to a disc (or be "vacuum cleaned")? There are several options for dealing with this, from those where users designate how items should be clustered in case of migration to those where the system figures out the access patterns and clusters on its own. One major argument is that dynamic clustering and migration are unique MMDB features (Ramon-Gonen & Gelbard, 2017).

ACTIVITY 7.1

A social media platform wants to use a main memory database system to provide real-time analytics of its user activity. What benefits can it expect from this approach, and what challenges might it face?

SUMMARY

This chapter covers various aspects of main memory databases, including their differences from magnetic disks, the applications of partitioning data, the frequency of backups, and the impact of memory resident data on concurrency control, commit processing, and access methods. Main memory databases have several advantages, including fast access times, but also some disadvantages, such as limited storage capacity. Understanding these aspects of main memory databases is important for effectively using them in various applications.

REVIEW QUESTIONS

1. What are the primary differences between main memory and magnetic disks?
2. What are some of the applications of partitioning data in a database?
3. What factors influence the frequency of backups in a main memory database?
4. What is the impact of memory resident data on concurrency control, commit processing, and access methods?
5. What are some of the advantages and disadvantages of using main memory databases?

MULTIPLE CHOICE QUESTIONS

1. Which of the following is not a difference between main memory and magnetic disks?

a. Access time
b. Capacity
c. Volatility
d. Cost

2. What is the primary application of partitioning data in a database?

a. Increasing storage capacity
b. Enhancing data security
c. Improving data access efficiency
d. Simplifying data management

3. Which of the following factors does not influence the frequency of backups in a main memory database?

a. Data volatility
b. Available storage space
c. Recovery time objectives
d. Data retention policies

4. **What is the impact of memory resident data on concurrency control?**
   a. It reduces the need for concurrency control
   b. It increases the likelihood of deadlocks
   c. It makes concurrency control unnecessary
   d. It simplifies concurrency control
5. **What is the impact of memory resident data on commit processing?**
   a. It speeds up commit processing
   b. It slows down commit processing
   c. It has no effect on commit processing
   d. It makes commit processing unnecessary
6. **Which of the following is not an access method impacted by memory resident data?**
   a. Hashing
   b. Indexing
   c. Sorting
   d. Sequential

Answers to Multiple Choice Questions

1. (c); 2. (c), 3. (b); 4. (b); 5. (a); 6. (c)

REFERENCES

1. Bakerjian, D., Bettega, K., Cachu, A. M., Azzis, L., & Taylor, S. (2020). The impact of music and memory on resident level outcomes in California nursing homes. *Journal of the American Medical Directors Association*, *21*(8), 1045-1050.
2. Bhide, A., & Stonebraker, M., (1988). An analysis of three transaction processing architectures. In: *VLDB* (Vol. 14, pp. 339–350).
3. Chou, S. Y., (1997). Patterned magnetic nanostructures and quantized magnetic disks. *Proceedings of the IEEE, 85*(4), 652–671.
4. Faerber, F., Kemper, A., Larson, P. Å., Levandoski, J., Neumann, T., & Pavlo, A., (2017). Main memory database systems. *Foundations and Trends® in Databases, 8*(1, 2), 1–130.
5. Friedrichs, M. S., & Wolynes, P. G., (1990). Molecular dynamics of associative memory Hamiltonians for protein tertiary structure recognition. *Tetrahedron Computer Methodology, 3*(3, 4), 175–190.
6. Garcia-Molina, H., & Salem, K., (1992). Main memory database systems: An overview. *IEEE Transactions on Knowledge and Data Engineering, 4*(6), 509–516.

7. Gruenwald, L., & Liu, S., (1993). A performance study of concurrency control in a real-time main memory database system. *ACM Sigmod Record, 22*(4), 38–44.
8. Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S., & Abadi, D. J., (2008). H-store: A high-performance, distributed main memory transaction processing system. *Proceedings of the VLDB Endowment, 1*(2), 1496–1499.
9. Ke, H., Li, P., Guo, S., & Guo, M., (2015). On traffic-aware partition and aggregation in MapReduce for big data applications. *IEEE Transactions on Parallel and Distributed Systems, 27*(3), 818–828.
10. Kemper, A., Neumann, T., Funke, F., Leis, V., & Mühe, H., (2012). HyPer: Adapting columnar main-memory data management for transactional AND query processing. *IEEE Data Eng. Bull., 35*(1), 46–51.
11. Khaleghzadeh, H., Manumachu, R. R., & Lastovetsky, A., (2018). A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous HPC platforms. *IEEE Transactions on Parallel and Distributed Systems, 29*(10), 2176–2190.
12. Larson, P. Å., & Levandoski, J., (2016). Modern main-memory database systems. *Proceedings of the VLDB Endowment, 9*(13), 1609–1610.
13. Levy, E., & Silberschatz, A., (1992). Incremental recovery in main memory database systems. *IEEE Transactions on Knowledge and Data Engineering, 4*(6), 529–540.
14. Li, J., Lee, P. P., Tan, C., Qin, C., & Zhang, X., (2020). Information leakage in encrypted deduplication via frequency analysis: Attacks and defenses. *ACM Transactions on Storage (TOS)*, *16*(1), 1–30.
15. Lindström, J., Raatikka, V., Ruuth, J., Soini, P., & Vakkila, K., (2013). IBM solidDB: In-memory database optimized for extreme speed and availability. *IEEE Data Eng. Bull.*, *36*(2), 14–20.
16. Litwin, W., & Risch, T., (1992). Main memory orientated optimization of OO queries using typed datalog with foreign predicates. *IEEE Transactions on Knowledge and Data Engineering, 4*(6), 517–528.
17. Mahmud, M. S., Huang, J. Z., Salloum, S., Emara, T. Z., & Sadatdiynov, K., (2020). A survey of data partitioning and sampling methods to support big data analysis. *Big Data Mining and Analytics, 3*(2), 85–101.
18. Manegold, S., Boncz, P. A., & Kersten, M. L., (2000). Optimizing database architecture for the new bottleneck: Memory access. *The VLDB Journal, 9*, 231–246.
19. Mastykash, O., Peleshchyshyn, A., Fedushko, S., Trach, O., & Syerov, Y., (2018). Internet social environmental platforms data representation. In: *2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)* (Vol. 1, pp. 199–202). IEEE.
20. Mittal, S., & Vetter, J. S., (2015). A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Transactions on Parallel and Distributed Systems, 27*(5), 1537–1550.

21. Morrison, R. E., Bryant, C. M., Terejanu, G., Prudhomme, S., & Miki, K., (2013). Data partition methodology for validation of predictive models. *Computers & Mathematics with Applications, 66*(10), 2114–2125.

22. Ng, R., (1992). Memory-fast computer memories. *IEEE Spectrum, 29*(10), 36–39.

23. Panda, P. R., Dutt, N. D., & Nicolau, A., (2000). On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, *5*(3), 682–704.

24. Pandis, I., Johnson, R., Hardavellas, N., & Ailamaki, A., (2010). Data-oriented transaction execution. *Proceedings of the VLDB Endowment, 3*, 928–939.

25. Qin, Z. S., Niu, T., & Liu, J. S., (2002). Partition-ligation–expectation-maximization algorithm for haplotype inference with single-nucleotide polymorphisms. *The American Journal of Human Genetics, 71*(5), 1242–1247.

26. Ramon-Gonen, R., & Gelbard, R., (2017). Cluster evolution analysis: Identification and detection of similar clusters and migration patterns. *Expert Systems with Applications, 83*, 363–378.

27. Salem, K., & Garcia-Molina, H., (1990). System M: A transaction processing testbed for memory resident data. *IEEE Transactions on Knowledge & Data Engineering, 2*(01), 161–172.

28. Ulusoy, Ö., & Buchmann, A., (1998). A real-time concurrency control protocol for main-memory database systems. *Information Systems, 23*(2), 109–125.

CHAPTER 8

ADVANCED DATABASE SECURITY

INTRODUCTION

Any organization can benefit from having access to information or data. Nearly all organizations, including social, governmental, and educational, have standardized information systems or other administrative tasks. The databases that hold the important data have been kept up to date by them. Hence, database safety is an important consideration. Before moving on, let's first talk about what database security entails. Database security aims to safeguard private or sensitive information in a repository. It deals with creating database security from any unauthorized access or threat. Permitting or preventing user access to the database and also the objects inside of it is necessary for database security. Successful businesses demand that databases remain confidential. Unauthorized individuals are not allowed to access their data or information (Bertino & Sandhu, 2005).

Additionally, it is reassurance that the data is shielded from unauthorized or unintentional change. Security issues include data security and privacy. The characteristics of database security, which are availability, confidentiality, and integrity, are shown in Figure 8.1.

Learning Objectives

By the end of this chapter, readers will be able to:

- Define the concept of database security and explain its importance in protecting data from unauthorized access.
- Identify the different types of threats that can compromise database security, such as excessive privilege abuse, privilege elevation, and SQL injection.
- Evaluate various techniques and strategies used to secure databases, including cryptography, steganography, and access control.
- Explain the role of encryption in database security and describe different encryption techniques.
- Analyze the impact of denial of service (DoS) attacks on databases and discuss techniques to mitigate them.
- Explain the concept of backup data exposure and evaluate different approaches to prevent it.
- Understand the importance of database platform vulnerabilities and database communication protocol vulnerabilities in database security.
- Assess the role of weak authentication in database security and describe various techniques to strengthen it.

Key Terms

- Access Control
- Audit Trails
- Breaches
- Cryptography
- DoS
- Encryption
- Excessive Privilege Abuse
- Rootkits
- Security
- SQL
- Threats

8.1. DEVELOPMENT OF ADVANCED DATABASE SECURITY

Confidentiality restricts the retrieval of secure data, preventing unauthorized access to the information. Integrity refers to the fact that the data won't be compromised in any way. Data timely availability is a feature of secure databases (Jajodia, 1996).

Inference control, access control, information flow control, and cryptographic flow control are listed as the four types of controls that can be employed to secure databases (Kumar & Rana, 2016).

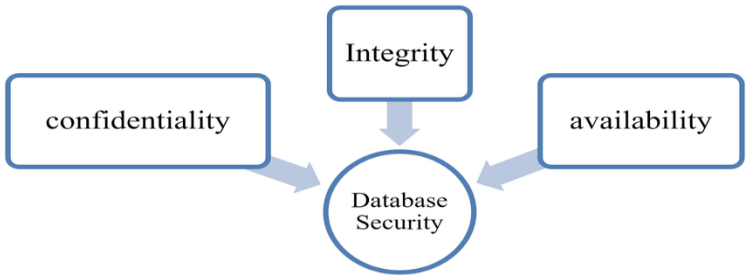


Figure 8.1. Properties of database security.

Source: Sandeep, creative commons license.

All straight accesses to the system are made approved by access controls. Access controls the objects of the system. Frequently, inappropriate information flows rather than poor access control results in the leakage or misuse of crucial information or data. Data in the system is less safeguarded when information flow policies are improperly defined. Data is encrypted and controlled through cryptographic control (Bertino, 1998). The databases have been secured using a different strategy. It has been proposed that different organizational policies might be used to make databases secure. Any organization's most valuable asset is data/information, whose security can't be violated. Technology advancements heighten the risk to such priceless assets. Hence, ensuring their security is difficult. Figure 8.2 illustrates the database security layers. The layers are database administrators (DBAs), system administrators, security officers, developers, and employees. Some specified security policies were anticipated for each layer (Pernul, 1994).

These rules protect users' integrity, confidentiality, and privacy. This research concentrates mainly on database security flaws and the measures to fix them. For non-public, public, and commercial enterprises, protecting sensitive information from theft, unauthorized access, and forgeries is a major concern. When sending data across several parties server-side, client-side encryption is insufficient.

The key challenge is assessing the security of a semi-trusted database (Widiasari, 2012).

A new theory for data encryption is put out, according to which apps could be given unified access to secure databases while still receiving database encryption as a service. Applications can focus on their core competencies while protecting data privacy from malevolent outsiders and users of untrusted database services using a data encryption management approach, all without being aware of the encryption details (Toshniwal et al., 2015).

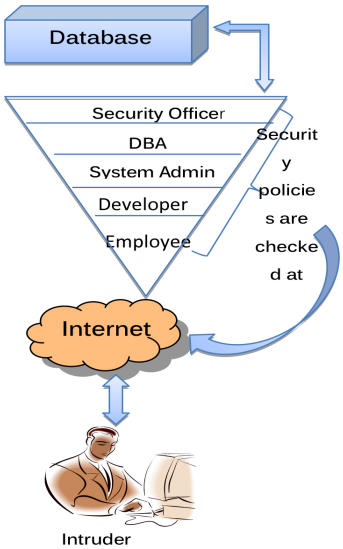


Figure 8.2. Security layer at organizational level.

Source: Anil Dixit, creative commons license.

The following section explains the measures taken to lessen or completely eradicate security concerns and how database security has been improved. And what must be done to protect a priceless resource, the organizations' databases (Kulkarni & Urolagin, 2012).

8.2. SECURITY RISKS TO DATABASES

The initiative database organization faces a vast array of dangers. The document predicts certain significant threats. The Application Defense Center at Imperva provided this list in a white paper.

8.2.1. Excessive Privilege Abuse

When users are granted access rights that let them carry out additional tasks beyond the scope of their work, those tasks may

reveal malicious intent that results in abusing those capabilities. A university example can be used, where an administrator has access to databases and has the authority to alter any student's information. Hence results in abuse, which could modify a student's grade, mark, or the amount of the fee imposed on them. As a result, default permission levels that provide excessive access are assigned to all users that carry out various actions (Malik & Patel, 2016) (Figure 8.3).

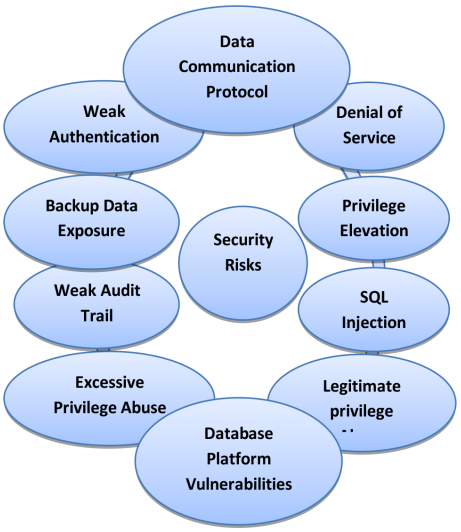


Figure 8.3. Database security risks.

Source: Anil Dixit, creative commons license.

8.2.2. Legitimate Privilege Abuse

Abuse of legitimate privileges might take the shape of a system manager, administrator, or database user engaging in illegal or unethical behavior. It is not restricted to any wrongful use of privileges or misuse of sensitive information (Deepika, 2015).

8.2.3. Privilege Elevation

Excessive exposure results in the discovery of defects exploited by hackers and may lead to a change in privileges, such as giving an ordinary user access to administrator powers. Loss of which might lead to fake accounts, money transfers, and incorrect interpretations of certain important analytical data. Similar instances are also discovered in SQL statements, protocols, and database functions (Singh & Rai, 2014).

8.2.4. Database Platform Vulnerabilities

Inadequacies in previous operating systems may lead to data loss, corruption, or denial of service in a database like Windows 2000 or Windows 98. For instance, the Windows 2000 vulnerability used by the blaster worm resulted in denial of service circumstances.

8.2.5. SQL Injection

The server is subjected to a series of malicious SQL queries from an attacker. The input for this attack is a string identifying preceded by SQL statement that has been server-verified. If it's not validated, it might be carried out. Attackers can access the entire database thanks to these unfettered rights (Asmawi et al., 2008).

Did you Know?

According to a report, the average cost of a data breach in 2020 was $3.86 million.

8.2.6. Weak Audit Trail

A database audit strategy guarantees all database transactions are logged promptly, accurately, and automatically. Since all significant database transactions possess an automatic record, its absence poses a significant threat to the company's databases. Such a policy must be incorporated within database security concerns as it could result in operational instability (Duncan & Whittington, 2017).

8.2.7. Denial of Service

An authorized user of the software, pieces of data, or applications cannot use or access that particular service due to the attack. DOS can happen in many different ways. Attackers may access databases and try to bring servers offline. They may provide the ideal environment for a (DoS) denial-of-service attack by overusing resources, flooding the network, and corrupting data. Any organization that uses it poses a serious threat.

8.2.8. Database Communication Protocol Vulnerabilities

Every database retailer's database communication method has several security flaws that are being found. Deceptive conduct targeting these vulnerabilities can take many forms, including unauthorized data access (Geneiatakis et al., 2006).

8.2.9. Weak Authentication

The databases become more exposed to attackers when there is a weak authentication technique. Suppose authentication is improperly implemented or is inadequate. In that case, it facilitates data theft by allowing an attacker to modify data or gain sensitive information via stealing the identities of database users or obtaining their login credentials from some source.

8.2.10. Backup Data Exposure

The release of data backup is a grave risk that requires an immediate response., Backups on DVDs, tapes, and other external media should be protected against theft and destruction because they are subject to significant risks. Some significant dangers are covered in database security (Alain et al., 2016).

Now examine what might be done to reduce these dangers and hazards.

Remember

The increasing prevalence of cyberattacks has led to the development of advanced database security measures such as encryption, access control, and intrusion detection and prevention systems, aimed at protecting sensitive data and ensuring data privacy.

8.3. DATABASE SECURITY CONSIDERATIONS

Each organization must establish a security plan to remove security risks. Also, this security strategy must be followed properly. A robust security policy requires clearly defined security characteristics. Figure 8.4 illustrates a few crucial areas that require consideration and is followed by an explanation. Access Control guarantees that almost all communications involving databases and other system objects follow predefined policies and constraints. Prevents intervention from outside or internal attackers and safeguards the databases against mistakes that could have a significant impact, such as halting a company's operations. The dangers that could directly affect the safety of databases on the primary servers are reduced with access control. For instance, the outcomes of any table deletion or access modification can be rolled back, and access control can prevent the deletion of particular files (Shmueli et al., 2010).

8.3.1. Inference Policy

A certain amount of data protection necessitates the use of inference policies. It happens when a higher level of security is needed to secure the conclusions drawn from specific information that takes

the shape of analysis or facts. Additionally, it decides how to prevent the data from becoming revealed (Basharat et al., 2012).

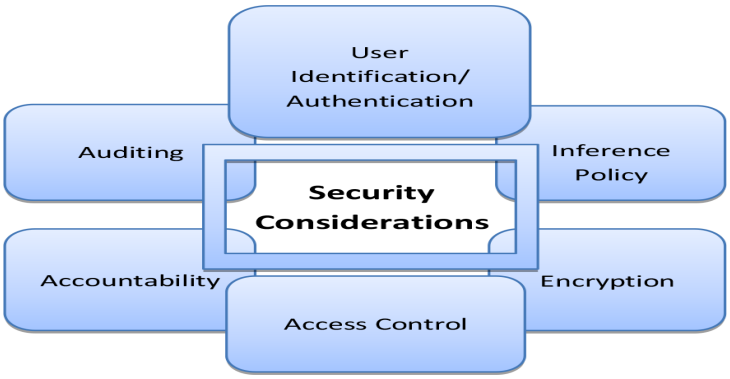


Figure 8.4. Critical areas under consideration.

Source: Suchithra, creative commons license.

8.3.2. User Identification/Authentication

Authentication and User identification is a fundamental security need since it establishes a group of individuals permitted access to data and offers a comprehensive accessibility mechanism. The identification is verified to assure safety and protects sensitive data from being altered by any common user (Chang et al., 2004).

8.3.3. Accountability and Auditing

Audit checks and accountability are necessary to protect the physical integrity of data. These checks require defined database access, which is controlled by auditing and record keeping. Additionally, it aids in analyzing data stored on servers for user access, accounting, and authentication.

8.4. ENCRYPTION

Encryption is the process of hiding or altering information to use a code or cipher so that only those having access to information can read it. Therefore, encoded information is referred to as encrypted information (Bhanot & Hans, 2015).

Data constitutes one of a company's most valuable assets. Hence, the security of an organization remains a formidable obstacle. Recent cryptographic examinations have focused on the security of public datasets. Mixed Cryptography Database is

a new technology in which many parties use separate keys to encrypt databases in diverse formats (MCDB). Sensitive data from numerous governmental, non-governmental, private, and other organizations are stored on web servers and must be shielded from hackers or intrusion. Several security methods were created to make databases secure. Techniques for encryption are one of them. Even while encryption increases protection, the choices made during implementation are crucial. For example, the what, how, when, and where of encryption. Figure 8.5 illustrates the location of encryption. Developing encryption techniques raises vital issues, such as when and how the encryption would be carried out (Davis, 1978).

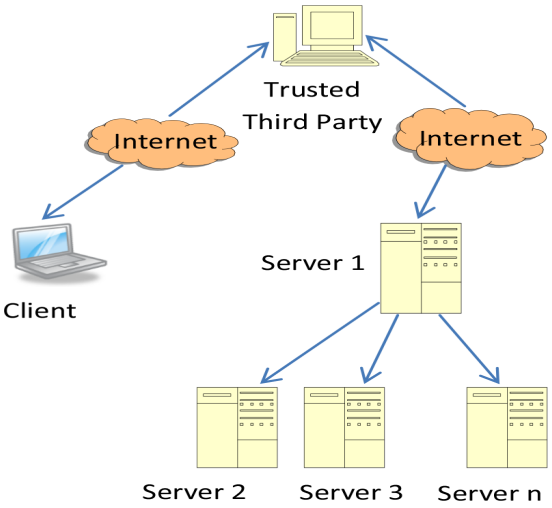


Figure 8.5. Three levels where encryption is performed.

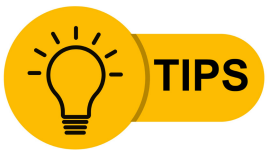
Source: Suchithra, creative commons license.

This approach does not discuss symmetric or asymmetric encryption algorithms. Algorithms have a detrimental effect on query processing performance. The encryption techniques impact the rate of query processing and security analysis. The optimum encryption algorithm to utilize within access control, mixed cryptography database strategies to govern access for all users, and indexing and joining among different databases are the other critical research challenges associated with this framework from a security and performance standpoint (Daemen & Rijmen, 2001).

Researchers assert that irrespective of the access control approach employed, it is impossible to circumvent the authorization requirements specified by the database server. For example, a stalker attempting to retrieve database information on a disc could access the information system. Database service providers (DSPs)

embrace threats and are hired to manage databases. The only option left to the database owner is to believe the DSPs. Also, the database administrator can abuse his powers and delete the database.

There are three distinct encryption levels. Database-level encryption, application-level encryption, and encryption at the storage level. Data within the storage subsystem is encrypted at the storage level. As it is transparent, the danger of any change to the current application is reduced. It isn't easy to select encrypt the files, such as those in temporary files, log files, etc., because storage level encryption requires the assurance that there won't be a single copy left unencrypted. Database-level encryption is used when data is recovered or saved from the database. Encryption can be performed on a row-by-row, column-by-column, or table-by-table basis. The encryption keys must be accessible on the server side to decrypt the data for both database-level and storage-level encryption techniques. The application executes the third application-level encryption. The greatest degree of flexibility is provided when the choice of keys and encryption granularity is determined using application logic (Liu & Koenig, 2010).



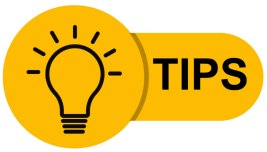
User identification and authentication are key components of database security, and should be implemented using strong encryption and access control policies.

The parameters that guarantee security are the encryption algorithm, key size, and key protection. The higher the security stronger the encryption algorithm is. Choosing the right operation mode is crucial when using a powerful encryption technique. Two ideas were put out to address the issue of obtaining keys without authorization. Using a security server with HSM. The database is still exposed to dangers even with the addition of a secure server and HSM that reduces the leakage of encryption keys.

Most databases are secured by using encryption techniques. It is not easy to implement cryptography on databases. Yet, it is widely recognized as one of the most significant problems with data security. While preserving data privacy and encouraging greater data sharing, an innovative encryption approach is proposed. Secure data is protected, and key management is efficiently done. This makes sharing encrypted data simpler. Encryption protects secrecy in databases.

A model has also been put forth showing all the database's threats. The user encrypts the data using functioning keys that they have randomly generated. To view the encrypted data, the Private key must first be decrypted (Davida et al., 1981). The technological evolution trend has erased the concept of boundaries

preventing access to any data media. With the click of a mouse, this limitless accessibility has shrunk the world and brought it closer together. Still, it has also raised the risk of security breaches, particularly in international business. Transparent Data Encryption has grown and now offers secure solutions to such problems. The information encoded and made readable and decodable only by the designated recipients is said to have been encrypted. This study examines transparent data encryption to protect against data fraud and theft. Encrypting or Encoding databases on networks, hard drives, and/or other backup media to enable highly flexible, transparent, advanced security for application development is the technical meaning of clear encrypting data (Smid & Branstad, 1988). This technology is used by Microsoft SQL Server 2008 to encrypt data files that are stored on every network, disc, or backup device along with the Master Key creation process. This covers key creation, certificate protection, and how to configure the database for Microsoft SQL Server encryption. This study looks into the benefits of the Microsoft SQL Server configurable environment for developers in terms of data security, application development, and data safety (Selent, 2010).



A Data Warehouse provides integrated, enterprise-wide, historical data and focuses on providing support for decision-makers for data modeling and analysis.

A brand-new, lightweight encryption technique is used with columns kept on trustworthy servers in data warehouses. The brand-new approach is known as (FCE) Fats Comparison Encryption. The comparison is both efficient and fat because of its overhead. The comparability of the earlier research will be presented in the following section.

8.4.1. Comparative Analysis

A comparison analysis is carried out in this part using three aspects.

The information in table 8.1 about the techniques, algorithms, and locations where encryption is applied in databases is provided. Table 8.1 lists many ways or procedures to secure the information. The comparison of such methods/techniques is then shown in Table 8.2.

Table 8.1. Encryption in Databases

Methods/Techniques	**Algorithm**	**Where Encryption can be Performed**
Using mixed cryptography and data classification techniques	You can use any symmetric encryption algorithm.	Encryption is done at -Client-side -Untrusted database -Server
Technique for Hash Security Module Encryption	Modern algorithms and modes of operation ought to be employed.	Encryption can be at: -Storage Level -Database Level -Application Level
Combining traditional encryption with public key encryption, taking advantage of both the convenience and speed of conventional encryption.	X	X
Master database key uses transparent data encryption	X	Page Level
Quick Comparative Encryption	algorithm for symmetric encryption	Dataware houses

Source: Anil Dixit, creative commons license.

Table 8.2. Comparison of Encryption Methods/Techniques

Methods/Techniques	**Advantages**	**Disadvantages / Limitations**
Mixed Cryptography Technique based on data classification methods	Because numerous keys belong to different parties, sensitive data is safeguarded from attacks Sensitive data is protected to the fullest extent through secure data transmission and storage.	Because of encryption techniques, queries' performance and security analysis's performance are both impacted. Methods of access control are not specified.

Hash Security Module Encryption Strategy	The security server is unaltered. Keys for encryption are never made public.	Complex
Transparent Data Encryption used by the Master database key	secures critical data on backup media and disc devices against unauthorized access. User management costs are decreased. Control privacy provision.	There is no provision for encryption across communication channels. If a certificate is unavailable and no backup of the certificate and private key is kept, the database cannot be opened. After changing the certificate to be password secured, the database becomes inaccessible.
Fast Comparison Encryption	quick indexing process with low overhead for decryption	

Source: Anil Dixit, creative commons license.

An empirical analysis will be provided in the section after this.

8.4.2. Empirical Analysis

The literature is carefully analyzed for this empirical investigation, after which conclusions are drawn.

8.4.2.1. Frequency

The quantity of repeated commonness occurrences is known as its frequency. The frequency is computed so that the paper with an issue that is unique to that paper is given a frequency of "1." Still, the paper with the issue shared by many papers is given a frequency proportional to the number of papers that share that issue (Standard, 1999).

8.4.2.2. Criticality

Criticality factor is split into four categories—Medium, Moderate, High, and Extremely High—to determine the occurrence rate of a problem. This is a definition of the criticality % range.

ACTIVITY 8.1

A healthcare organization wants to implement advanced database security measures to protect sensitive patient information. What are some key considerations in doing so, and what measures can it take to ensure compliance with regulations?

SUMMARY

Any organization's most valuable asset is its data. Any level of business faces a significant issue when protecting sensitive data. Databases are susceptible to numerous threats in the technological world of today. The main security problems that databases face are addressed, and some encryption techniques that can help to lessen attack risks and safeguard sensitive data are discussed. Conclusion: Encryption can provide confidentiality but not integrity unless we also use a digital signature and hash function. Strong encryption algorithms slow down the system. Further research could be done to improve the effectiveness and efficiency of encryption.

REVIEW QUESTIONS

1. What is the importance of database security, and what are some common threats to database security?
2. How does cryptography enhance database security, and what are some common encryption techniques used in databases?
3. What is the role of audit trails in database security, and how can you design and implement an effective audit trail?
4. What are the different types of privilege abuse in database security, and how can you prevent them?
5. What is backup data exposure in database security, and what are some techniques to prevent it?
6. What are some common database vulnerabilities and how can you mitigate them?

MULTIPLE CHOICE QUESTIONS

1. Which of the following is a common threat to database security?

a. Effective encryption
b. Strong authentication
c. Denial of service attacks
d. Reliable backup systems

2. What is the primary goal of cryptography in database security?

a. To prevent unauthorized access to the database
b. To identify vulnerabilities in the database
c. To improve the efficiency of the database
d. To simplify the administration of the database

3. What is the main purpose of audit trails in database security?

a. To detect and prevent SQL injection attacks

b. To monitor database activity and identify potential security breaches
c. To encrypt sensitive data in the database
d. To backup and restore the database in case of a disaster

4. Which of the following is not a type of privilege abuse in database security?

a. Excessive privilege abuse
b. Legitimate privilege abuse
c. Privilege elevation
d. Rootkit privilege abuse

5. What is the primary technique used to prevent backup data exposure in database security?

a. Encryption
b. Access control
c. Steganography
d. Denial of service attacks

Answers to Multiple Choice Questions

1. (c); 2. (a); 3. (b); 4. (d); 5. (a)

REFERENCES

1. Alain, N., Ann, K. I. B. E., & Cheruiyot, W. K., (2016). Use of enhanced transparent data encryption to protect database against exposure of backup data. *International Journal of Scientific Engineering and Technology, 5*(10), 477–481.
2. Arockiam, L., & Monikandan, S., (2013). Data security and privacy in cloud storage using hybrid symmetric encryption algorithm. *International Journal of Advanced Research in Computer and Communication Engineering, 2*(8), 3064–3070.
3. Asmawi, A., Sidek, Z. M., & Abd, R. S., (2008). System architecture for SQL injection and insider misuse detection system for DBMS. In: *2008 International Symposium on Information Technology* (Vol. 4, pp. 1–6).
4. Basharat, I., Azam, F., & Muzaffar, A. W., (2012). Database security and encryption: A survey study. *International Journal of Computer Applications, 47*(12), 5–10.
5. Bertino, E., & Sandhu, R., (2005). Database security-concepts, approaches, and challenges. *IEEE Transactions on Dependable and Secure Computing, 2*(1), 2–19.
6. Bertino, E., (1998). Data security. *Data & Knowledge Engineering, 25*(1–2), 199–216.
7. Bhanot, R., & Hans, R., (2015). A review and comparative analysis of various encryption algorithms. *International Journal of Security and its Applications, 9*(4), 289–306.

8. Chang, C. C., Chen, K. L., & Hwang, M. S., (2004). End-to-end security protocol for mobile communications with end-user identification/authentication. *Wireless Personal Communications, 28*, 95–106.
9. Daemen, J., & Rijmen, V., (2001). Reijndael: The advanced encryption standard. *Dr. Dobb's Journal: Software Tools for the Professional Programmer, 26*(3), 137–139.
10. Davida, G. I., Wells, D. L., & Kam, J. B., (1981). A database encryption system with subkeys. *ACM Transactions on Database Systems (TODS), 6*(2), 312–328.
11. Davis, R., (1978). The data encryption standard in perspective. *IEEE Communications Society Magazine, 16*(6), 5–9.
12. Deepika, N. S., (2015). Database security: Threats and security techniques. *International Journal of Advanced Research in Computer Science and Software Engineering, 5*(5), 6–10.
13. Duncan, B., & Whittington, M., (2017). Creating and configuring an immutable database for secure cloud audit trail and system logging. *International Journal on Advances in Security, 10*(3&4), 155–166.
14. Fernando, Y., Chidambaram, R. R., & Wahyuni-TD, I. S., (2018). The impact of big data analytics and data security practices on service supply chain performance. *Benchmarking: An International Journal, 25*(9), 4009–4034.
15. Geneiatakis, D., Dagiuklas, T., Kambourakis, G., Lambrinoudakis, C., Gritzalis, S., Ehlert, K. S., & Sisalem, D., (2006). Survey of security vulnerabilities in session initiation protocol. *IEEE Communications Surveys & Tutorials, 8*(3), 68–81.
16. George, B., & Valeva, A., (2006). A database security course on a shoestring. *ACM SIGCSE Bulletin, 38*(1), 7–11.
17. Harn, L., & Lin, H. Y., (1990). A cryptographic key generation scheme for multilevel data security. *Computers & Security, 9*(6), 539–546.
18. Jajodia, S., (1996). Database security and privacy. *ACM Computing Surveys (CSUR), 28*(1), 129–131.
19. Kulkarni, S., & Urolagin, S., (2012). Review of attacks on databases and database security techniques. *International Journal of Emerging Technology and Advanced Engineering, 2*(11), 253–263.
20. Kumar, P., & Rana, S. B., (2016). Development of modified AES algorithm for data security. *Optik, 127*(4), 2341–2345.
21. Liu, F., & Koenig, H., (2010). A survey of video encryption algorithms. *Computers & Security, 29*(1), 3–15.
22. Malik, M., & Patel, T., (2016). Database security-attacks and control methods. *International Journal of Information, 6*(1, 2), 175–183.
23. Manogaran, G., Thota, C., & Kumar, M. V., (2016). MetaCloudDataStorage architecture for big data security in cloud computing. *Procedia Computer Science, 87*, 128–133.
24. Pernul, G., (1994). Database security. In: *Advances in Computers* (Vol. 38, pp. 1–72).

25. Rao, R. V., & Selvamani, K., (2015). Data security challenges and its solutions in cloud computing. *Procedia Computer Science, 48*, 204–209.
26. Selent, D., (2010). Advanced encryption standard. *Rivier Academic Journal, 6*(2), 1–14.
27. Shmueli, E., Vaisenberg, R., Elovici, Y., & Glezer, C., (2010). Database encryption: An overview of contemporary challenges and design considerations. *ACM SIGMOD Record, 38*(3), 29–34.
28. Singh, S., & Rai, R. K., (2014). A review report on security threats on database. *International Journal of Computer Science and Information Technologies, 5*(3), 3215–3219.
29. Smid, M. E., & Branstad, D. K., (1988). Data encryption standard: Past and future. *Proceedings of the IEEE, 76*(5), 550–559.
30. Standard, D. E., (1999). *Data Encryption Standard* (Vol. 112, pp. 3–10). Federal Information Processing Standards Publication.
31. Toshniwal, R., Dastidar, K. G., & Nath, A., (2015). Big data security issues and challenges. *International Journal of Innovative Research in Advanced Engineering (IJIRAE), 2*(2), 5–20.
32. Widiasari, I. R., (2012). Combining advanced encryption standard (AES) and one time pad (OTP) encryption for data security. *International Journal of Computer Applications, 57*(20), 1–12.
33. Wu, C. P., & Kuo, C. C., (2005). Design of integrated multimedia compression and encryption systems. *IEEE Transactions on Multimedia, 7*(5), 828–839.
34. Zhang, Y., Le, X., Jian, Y., Lu, W., Zhang, J., & Chen, T., (2019). 3D fluorescent hydrogel origami for multistage data security protection. *Advanced Functional Materials, 29*(46), 1905514.

INDEX

D

E

F

G

O

P

Q

R

S

About the Editor

Waqas Ahmed completed his Ph.D in Cyber security from East London University in 2022. He was awarded for academic excellence in his Master's degree. He qualified data scientist in January 2020. He has more than 5 years of teaching and research experience at different universities. Waqas Ahmed published 5 research papers, 3 conference papers, and 1 book chapter in well-established journals. He works in the area of computer networks, system security, and programming. He loves to read and share interesting aspects of computer science in books. In his free time, he loves to travel and explore and give talks on spirituality, ancient customs, and traditions.

ISBN 978-1-77956-173-2

Advanced Database Systems

Advanced database systems refer to complex and specialized systems designed to handle large amounts of data while ensuring reliability, security, and efficiency. These systems incorporate advanced features such as distributed computing, in-memory processing, and cloud-based solutions to improve performance and scalability. They are essential for organizations dealing with large amounts of data and require efficient data management and processing solutions. Advanced database systems play a critical role in various industries, such as finance, healthcare, e-commerce, and social media, by providing a unified platform for data storage and management. As data continues to grow exponentially, advanced database systems will continue to evolve to meet the increasing demand for efficient and effective data management solutions. In today's data-driven world, Advanced Database Systems have become an essential topic for anyone working in the field of information technology. It provides the necessary skills and knowledge to design, develop, deploy, and maintain complex database systems that meet the demands of modern businesses and organizations.

The field of database systems has undergone a rapid transformation in recent years with the emergence of new technologies and innovations. The book "Advanced Database Systems" aims to provide a comprehensive overview of these advancements and the challenges they present. The book is divided into eight chapters, each of which covers a critical topic in modern database systems. Chapter 2 provides an introduction to database systems and their role in modern business and society. Chapter 2 discusses the different types of database systems, including relational, NoSQL, and object-oriented databases. Chapter 3 provides an overview of various database models, including the hierarchical, network, and object-oriented models. Chapter 4 delves into big data analytics and how it has changed the way we store and manage data. Chapter 5 covers stream processing systems, including the various algorithms used for data analysis in real time. Chapter 6 focuses on cloud-based database systems, including the benefits and challenges of cloud computing. Chapter 7 examines the main memory database systems, which have become increasingly popular due to their speed and scalability. Finally, Chapter 8 covers advanced database security, including the various techniques used to protect against cyber threats and unauthorized access.

This book is intended for students and professionals in the field of database systems, as well as anyone interested in the advancements and challenges of modern database technologies. We hope that this book provides a valuable resource to its readers and helps them gain a deeper understanding of advanced database systems.

TAP
Toronto Academic Press