2023

Database Design and Implementation

Weiru Chen
Arkansas Tech University, wchen7@atu.edu

Database Design and Implementation

By: Dr. Weiru Chen

Arkansas Tech University

Table of Contents

Introduction

The book of Database Design and Implementation is a comprehensive guide that provides a thorough introduction to the principles, concepts, and best practices of database design and implementation. It covers the essential topics required to design, develop, and manage a database system, including data modeling, database normalization, SQL programming, and database administration.

The book is designed for students, database administrators, software developers, and anyone interested in learning how to design and implement a database system. It provides a step-by-step approach to database design and implementation, with clear explanations and practical examples. It also includes exercises and quizzes at the end of each chapter to help reinforce the concepts covered. The book begins by introducing the fundamental concepts of database systems and data modeling. It then discusses the process of database design and normalization, which is essential for creating a well-structured and efficient database system. The book also covers SQL programming, which is used for querying, updating, and managing data in a database. Additionally, it includes a comprehensive discussion on database administration, including security, backup and recovery, and performance tuning.

The book of Database Design and Implementation is written in a clear and concise manner, making it accessible to readers with little to no prior experience in database design and implementation. It is also suitable for those who already have experience in this field but wish to deepen their knowledge and improve their skills.

Chapter 0. Introduction to Information Systems

Learning Objective

Explain why it is essential to learn about information systems nowadays.

Define an information system and describe its primary operations.

List some important milestones of the development of information systems.

Identify the major parts of information systems.

Discuss the societal impact of information systems.

Reasons to learn about information systems nowadays

Learning about information systems is essential today because we live in a digital age where technology plays a significant role in our daily lives, both at home and at work. Information systems are the backbone of modern organizations, and they are used to support various functions, such as data storage and retrieval, decision-making, communication, and collaboration.

Figure 0.1 Information Systems

Understanding information systems is critical because it can help individuals and organizations to:

- Improve productivity and efficiency: Information systems provide tools and technologies that can help individuals and organizations to streamline their work processes, automate routine tasks, and increase productivity.
- Make better decisions: Information systems can provide access to real-time data and analytics, which can be used to make informed and data-driven decisions.
- Enhance communication and collaboration: Information systems can facilitate communication and collaboration among individuals and teams, regardless of their physical location.
- Gain a competitive advantage: Information systems can provide a competitive advantage by enabling organizations to differentiate themselves from their competitors through innovation and agility.
- Ensure data security and privacy: Information systems can help to protect sensitive data and ensure that it is not compromised or stolen.

Overall, learning about information systems is critical in today's digital age, as it can help individuals and organizations to harness the power of technology to achieve their goals and objectives.

An information system and describe its primary operations

An information system (IS) is a set of interconnected components that work together to collect, process, store, and distribute information to support decision-making, coordination, control, analysis, and visualization of an organization's activities.

The primary operations of an information system are as follows:

1. Input: The process of collecting data and information from various sources, such as users, sensors, databases, and other systems. The input can be in the form of text, images, audio, or video, and it needs to be captured, verified, and validated before it can be processed.
2. Processing: The process of converting raw data into meaningful information by applying various algorithms, rules, and procedures. This operation involves transforming, aggregating, filtering, and analyzing data to generate insights and knowledge that can be used to support decision-making.
3. Storage: The process of storing information in a secure and organized manner for future use. This operation involves selecting appropriate storage technologies, such as databases, cloud storage, or file systems, and designing a data schema and access controls that meet the organization's requirements.
4. Output: The process of presenting information to users in a useful and understandable format. This operation involves designing user interfaces, reports, dashboards, and visualizations that communicate insights and knowledge to users effectively.
5. Feedback: The process of monitoring the performance of the information system and providing feedback to users to improve its effectiveness and efficiency. This operation involves measuring key performance indicators, such as response time, accuracy, availability, and user satisfaction, and using this feedback to optimize the system's performance.

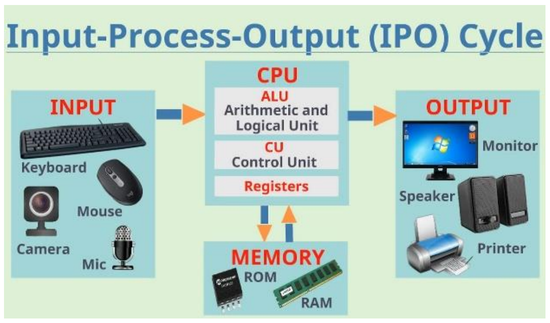


Figure 0.2 IPOS Cycle

Overall, an information system is a complex and dynamic system that performs various operations to collect, process, store, and distribute information to support an organization's activities and achieve its goals and objectives.

List some important milestones of the development of information systems

The development of information systems has been a long and ongoing process that spans several decades. Here are some of the most important milestones in the history of information systems:

1940s-1950s: The first computers were developed, which were large, expensive, and used mainly for scientific and military purposes.

1960s: The development of the first commercial computer systems, which were used for business applications such as payroll and inventory management.

1970s: The development of the first database management systems (DBMS), which made it easier to store and retrieve large amounts of data.

1980s: The development of the first personal computers (PCs) and the widespread adoption of local area networks (LANs), which enabled individuals and small groups to access and share information more easily.

1990s: The emergence of the internet and the World Wide Web, which revolutionized the way people communicate and access information.

2000s: The development of mobile devices and the proliferation of wireless networks, which enabled people to access information on the go.

2010s: The rise of cloud computing and big data analytics, which enabled organizations to store and process large amounts of data more efficiently and cost-effectively.

2020s: The increasing use of artificial intelligence (AI) and machine learning (ML) to automate and optimize business processes and decision-making.

The development of information systems has been a continuous process of innovation and evolution, driven by advances in computing, networking, and software technologies, and shaped by the needs and demands of organizations and individuals.

Major parts of information systems.

An information system is composed of several components that work together to collect, process, store, and distribute information. The major parts of an information system are:

- Hardware: The physical components of the system, such as computers, servers, printers, scanners, and other peripheral devices that provide input and output capabilities.
- Software: The programs and applications that run on the hardware and enable the system to perform various tasks, such as data processing, storage, and retrieval.
- Data: The raw facts and figures that are collected, stored, and processed by the system. Data can be structured (in a database) or unstructured (in documents, images, videos, etc.).
- Procedures: The rules, guidelines, and protocols that govern how the system operates and how users interact with it. Procedures can include data entry, processing, security, backup, recovery, and maintenance.
- People: The individuals who interact with the system, such as users, administrators, developers, and support staff. People are an essential part of the system and can influence its design, development, and implementation.
- Network: The communication infrastructure that connects the hardware components and enables data to be transmitted and received across the system. Networks can be local (LAN), wide (WAN), or global (Internet), and they can use wired or wireless technologies.

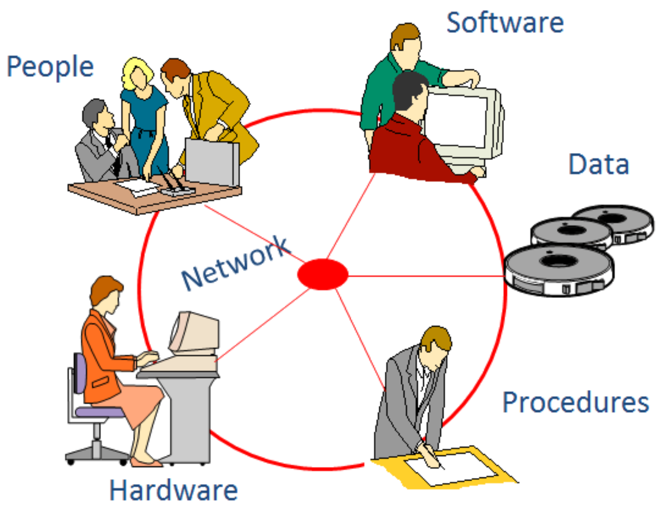


Figure 0.3 Major Components of Information Systems

Societal impact of information systems

Information systems have had a profound impact on society, affecting virtually every aspect of human life. Some of the key societal impacts of information systems are:

- Increased productivity and efficiency: Information systems have enabled individuals and organizations to automate and streamline many tasks, reducing the time and effort required to perform them. This has led to increased productivity and efficiency in areas such as manufacturing, healthcare, education, and transportation.
- Improved communication and collaboration: Information systems have made it easier for people to communicate and collaborate with each other, regardless of their location or time zone. This has facilitated the sharing of ideas, knowledge, and resources, and has enabled people to work together more effectively.
- Enhanced decision-making: Information systems have enabled individuals and organizations to collect, analyze, and interpret data more effectively, leading to better decision-making. This has been particularly beneficial in areas such as healthcare, finance, and marketing, where decisions can have significant consequences.
- Increased access to information and services: Information systems have made it easier for people to access information and services, regardless of their location or socioeconomic status. This has democratized access to education, healthcare, financial services, and other essential services, and has enabled people to participate more fully in society.
- Cybersecurity risks: Information systems have also introduced new risks and challenges, such as cybercrime, hacking, identity theft, and privacy breaches. These risks can have significant societal and economic impacts, and require ongoing efforts to mitigate and manage.

Information systems have had a transformative impact on society, enabling new forms of communication, collaboration, and innovation. While there are challenges and risks associated with these systems, their potential benefits are significant and far-reaching.

Chapter 1(a). Introduction to Database - Information Systems

Learning Objective

Identify the nature and characteristics of databases.

Gain a general understanding of tables and relationships.

Describe the components of a database system and explain the functions they perform.

Characteristics of databases

Databases are collections of data that are organized in a particular way to enable efficient retrieval, manipulation, and storage of information. Some of the key characteristics of databases include:

Data structure: Databases have a specific data structure, which is designed to organize data in a way that is easy to access and manipulate. This structure may be hierarchical, network, relational, or object-oriented, depending on the type of database.

Data integrity: Databases have built-in mechanisms to ensure data integrity, which means that the data stored in the database is accurate, consistent, and complete. This is achieved through the use of constraints, such as unique keys and referential integrity constraints.

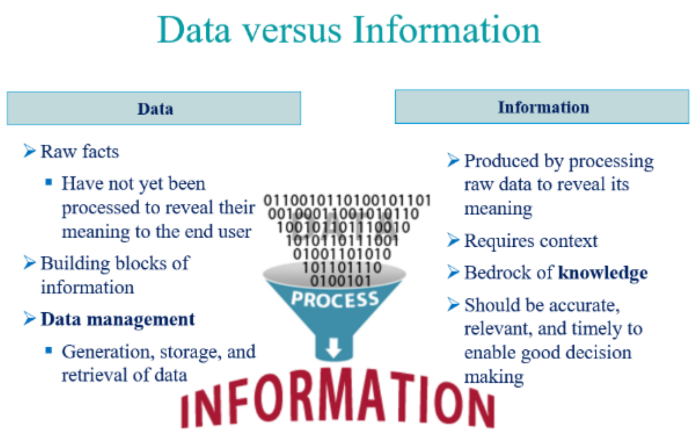
Data independence: Databases provide a layer of abstraction between the data and the applications that use the data. This means that applications can access and manipulate the data without needing to know how the data is stored or organized.

Concurrent access: Databases support concurrent access by multiple users, which means that multiple users can access and manipulate the data simultaneously without interfering with each other.

Security: Databases provide security mechanisms to protect the data from unauthorized access, modification, or deletion. This is achieved through the use of access control mechanisms, such as user accounts, roles, and permissions.

Scalability: Databases are designed to scale to handle large volumes of data and high levels of user traffic. This is achieved through the use of techniques such as partitioning, clustering, and replication.

Performance: Databases are optimized for performance, which means that they are designed to handle queries and transactions efficiently and quickly. This is achieved through the use of indexing, caching, and query optimization techniques.


PROCESS
INFORMATION

Data versus Information

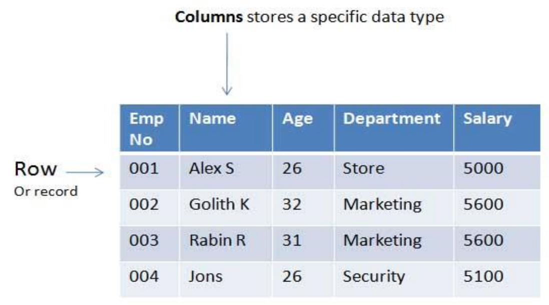
Data	Information
➢ Raw facts ▪ Have not yet been processed to reveal their meaning to the end user ➢ Building blocks of information ➢ **Data management** ▪ Generation, storage, and retrieval of data	➢ Produced by processing raw data to reveal its meaning ➢ Requires context ➢ Bedrock of **knowledge** ➢ Should be accurate, relevant, and timely to enable good decision making

Figure 1.1 The Difference Between Data and Information

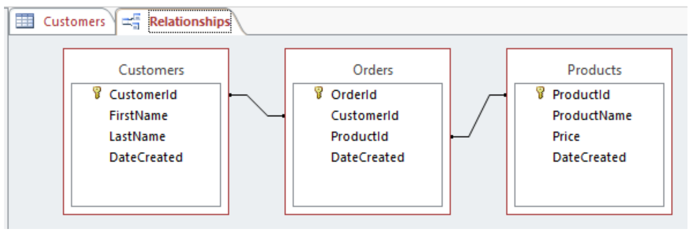
Tables and relationships

Tables and relationships are key components of a relational database. A table is a collection of related data, organized in rows and columns, where each row represents a single record or instance of the data, and each column represents a specific attribute or field of the data. Relationships, on the other hand, define how tables are related to each other.

Tables and relationships are fundamental concepts in database design and implementation, and understanding them is essential for building effective and efficient database systems.


Columns stores a specific data type
Row
Or record

Emp No	Name	Age	Department	Salary
001	Alex S	26	Store	5000
002	Golith K	32	Marketing	5600
003	Rabin R	31	Marketing	5600
004	Jons	26	Security	5100



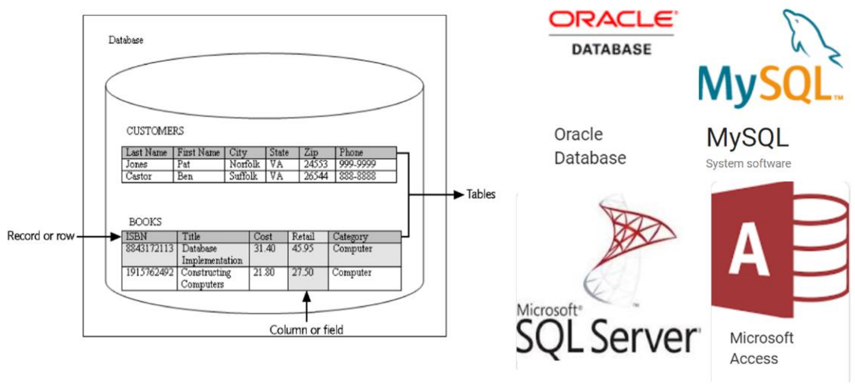
Customers	Orders	Products
CustomerId	OrderId	ProductId
FirstName	CustomerId	ProductName
LastName	ProductId	Price
DateCreated	DateCreated	DateCreated

Figure 1.2 Table and Relationships

Components and functions of a database system

A database system is composed of several components that work together to store, retrieve, and manage data. The main components and functions of a database system are:

- Hardware: This includes the physical components of the computer system such as the CPU, memory, and storage devices that are used to store and process data.
- Software: This includes the database management system (DBMS) software that is used to create, manage, and manipulate data in the database. The DBMS provides the interface for users and applications to interact with the database.
- Data: This is the actual data that is stored in the database, organized in tables or other structures depending on the type of database.
- Procedures: This includes the processes and procedures used to manage and maintain the database, including backup and recovery procedures, security procedures, and performance tuning.
- Users: This includes the people or applications that interact with the database system to perform various tasks such as querying, updating, and analyzing data.



CUSTOMERS

Last Name	First Name	City	State	Zip	Phone
Jones	Pat	Norfolk	VA	24553	999-9999
Castor	Ben	Suffolk	VA	26544	888-8888

BOOKS

ISBN	Title	Cost	Retail	Category
8843172113	Database Implementation	31.40	45.95	Computer
1915762492	Constructing Computers	21.80	27.50	Computer

Figure 1.3 Example of Database Systems

Chapter 1(b). Introduction to Database - Database Systems

Learning Objective

Understand concept of database systems.

Understand the development of database systems.

Understand database management system (DBMS) and identify the functions of a DBMS.

Explain the functions of Microsoft Access database system.

Explain the functions of an enterprise-class database system.

Concept of database systems

A database system is a software system designed to manage and organize large amounts of data. It provides a mechanism for storing, retrieving, and manipulating data in an efficient and secure manner. A database system consists of several components, including hardware, software, data, procedures, and users.

The concept of a database system is based on the idea that data is a valuable asset and should be managed like any other asset. By storing data in a central location and providing controlled access to it, a database system can help ensure data consistency, accuracy, and security. A database system provides a mechanism for sharing data among multiple users and applications, while maintaining data integrity and security.

The primary goal of a database system is to provide an efficient and effective way to store and retrieve data. This is achieved through the use of a database management system (DBMS) which provides a set of tools and functions for creating, maintaining, and querying the database. The DBMS serves as an interface between the user or application and the database, providing a mechanism for managing data storage, retrieval, and manipulation.

The concept of a database system is closely related to the concept of a database, which is a collection of related data organized in a particular way. A database system provides a means of managing and manipulating data in a database, while ensuring its consistency and security

Development of database systems

The history of database systems dates back to the early 1960s when the need for a more efficient and organized method of storing and accessing data arose. The following are some of the significant milestones in the evolution of database systems:

- Early database systems: In the 1960s, the first database management systems were developed. The first commercial database system, Integrated Data Store (IDS), was released in 1965 by General Electric. The system used a hierarchical model to organize data.
- Relational database systems: In the 1970s, Edgar F. Codd developed the relational data model, which formed the basis for the development of the relational database management system (RDBMS). The first RDBMS, called System R, was developed by IBM in the 1970s.
- SQL: In the 1980s, the Structured Query Language (SQL) was developed. SQL is a standard language for managing relational databases and has become the standard for database management.
- Object-oriented database systems: In the 1990s, object-oriented database management systems (OODBMS) were developed. OODBMS combines the principles of object-oriented programming and database management.
- NoSQL: In the 2000s, NoSQL databases emerged. NoSQL databases are non-relational databases designed to handle large volumes of unstructured data.
- Cloud-based database systems: In recent years, cloud-based database systems have become popular. Cloud-based database systems allow users to access and manage data from any location and at any time, making it easier to collaborate and share data.

Database management system (DBMS)

A Database Management System (DBMS) is a software system designed to manage and organize data in a database. The DBMS serves as an interface between the user or application and the database, providing a mechanism for managing data storage, retrieval, and manipulation.

DBMS software typically provides the following functionality:

Data Definition Language (DDL): DBMS provides DDL commands to define the structure of the database. DDL commands allow users to create, modify, and delete database objects such as tables, views, and indexes.

Data Manipulation Language (DML): DBMS provides DML commands to manipulate the data in the database. DML commands allow users to insert, update, and delete data in the database.

Query Language: DBMS provides a query language, which allows users to retrieve data from the database. The most common query language used in DBMS is SQL (Structured Query Language).

Data Integrity: DBMS enforces data integrity by ensuring that data is accurate and consistent. DBMS provides mechanisms such as constraints, triggers, and rules to enforce data integrity.

Security: DBMS provides security features to ensure that data is protected from unauthorized access. Security features include authentication, authorization, and encryption.

Concurrency Control: DBMS ensures that multiple users can access the database simultaneously without corrupting the data. DBMS provides mechanisms such as locking and transaction management to ensure concurrency control.

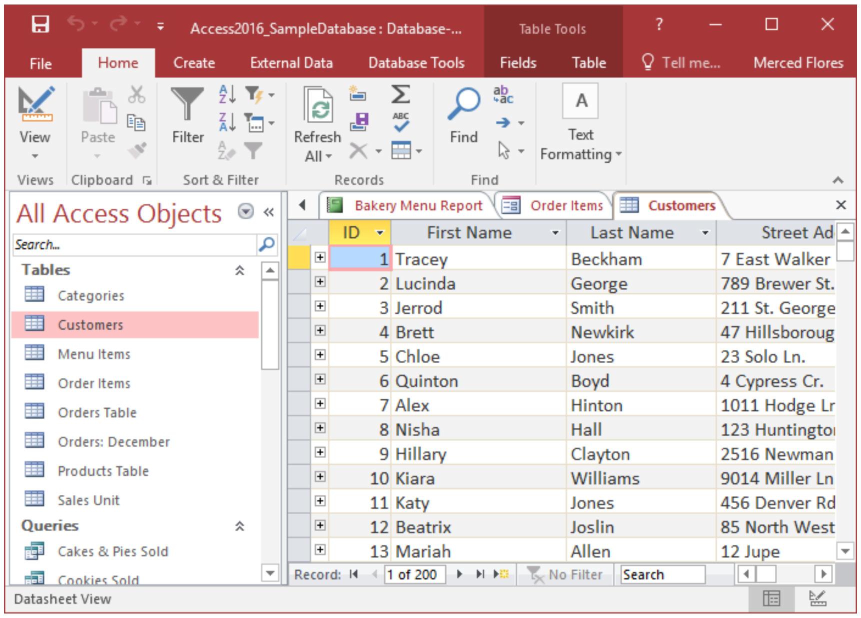
DBMS is an essential tool for managing large amounts of data efficiently and securely. The use of DBMS has revolutionized the way we store and manage data, making it easier for users to access and manipulate data in a consistent and secure manner.

Table 1.1 Functions of DBMS

FUNCTIONS OF A DBMS
Create database
Create tables
Create supporting structures (e.g., Indexes)
Modify (insert, update, or delete) database data
Read database data
Maintain database structures
Enforce rules
Control concurrency
Perform backup and recovery

Microsoft Access Database

Microsoft Access is a relational database management system (RDBMS) developed by Microsoft. It is a popular database management tool used by individuals and organizations to manage data in a simple and effective manner. Access provides a graphical user interface (GUI) for creating and managing databases, making it an easy-to-use tool for users with little or no programming experience.



ID	First Name	Last Name	Street Ad
1	Tracey	Beckham	7 East Walker
2	Lucinda	George	789 Brewer St.
3	Jerrod	Smith	211 St. George
4	Brett	Newkirk	47 Hillsboroug
5	Chloe	Jones	23 Solo Ln.
6	Quinton	Boyd	4 Cypress Cr.
7	Alex	Hinton	1011 Hodge Lr
8	Nisha	Hall	123 Huntingtoi
9	Hillary	Clayton	2516 Newman
10	Kiara	Williams	9014 Miller Ln
11	Katy	Jones	456 Denver Rd
12	Beatrix	Joslin	85 North West
13	Mariah	Allen	12 Jupe

Figure 1.4 GUI of Microsoft Access

Oracle Database

Oracle Database is a relational database management system (RDBMS) developed by Oracle Corporation. It is one of the most widely used enterprise-level database management systems in the world, offering powerful features and capabilities for managing large amounts of data in a secure and efficient manner.

Some of the features and capabilities of Oracle Database include:

Scalability: Oracle Database is designed to manage large amounts of data and can scale to support thousands of users, transactions, and applications.

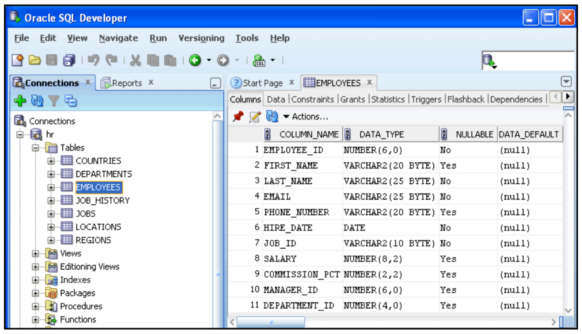
High Availability: Oracle Database provides features such as automatic backup and recovery, automatic storage management, and clustering to ensure high availability and data redundancy.

Security: Oracle Database provides a range of security features such as user authentication, data encryption, access control, and auditing to ensure the confidentiality, integrity, and availability of data.

Performance: Oracle Database is optimized for high performance, with features such as in-memory column store, advanced indexing, and query optimization.

Manageability: Oracle Database provides tools for database administration, monitoring, and tuning, making it easy to manage and maintain the database.

Integration: Oracle Database integrates with a wide range of applications and technologies, including Java, .NET, and XML.



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT
1	EMPLOYEE_ID	NUMBER(6,0)	No	(null)
2	FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)
3	LAST_NAME	VARCHAR2(25 BYTE)	No	(null)
4	EMAIL	VARCHAR2(25 BYTE)	No	(null)
5	PHONE_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)
6	HIRE_DATE	DATE	No	(null)
7	JOB_ID	VARCHAR2(10 BYTE)	No	(null)
8	SALARY	NUMBER(8,2)	Yes	(null)
9	COMMISSION_PCT	NUMBER(2,2)	Yes	(null)
10	MANAGER_ID	NUMBER(6,0)	Yes	(null)
11	DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)

Figure 1.5 GUI of Oracle SQL Developer

Oracle Database is used by many large organizations, including governments, financial institutions, and multinational corporations, to manage their mission-critical data. It is a robust, reliable, and secure database management system that provides users with the flexibility and scalability they need to manage their data efficiently and effectively.

Chapter 2(a). Introduction to SQL

Learning Objective

Learn the concept and significance of Structured Query Language (SQL)

Understand SQL commands and queries to retrieve data from a single table.

Concept of SQL

SQL is a programming language that is used to manage and manipulate data stored in RDBMS. It is used to create, modify and query databases, as well as manage the access to them. SQL is a standard language used by most relational databases, such as MySQL, Oracle, and Microsoft SQL Server.

SQL is a declarative language, which means that you specify what you want to do, and the system will determine how to do it. SQL commands are divided into categories, such as data definition language (DDL), data manipulation language (DML), and data control language (DCL). Here are some key concepts in SQL:

Data definition language (DDL) statements: Used for creating tables, relationships, and other structures.

Data manipulation language (DML) statements: Used for:

- Queries – SQL SELECT statement
- Inserting data – SQL INSERT statement
- Modifying data – SQL UPDATE statement
- Deleting data – SQL DELETE statement

Transaction control language (TCL) statements: Used to mark transaction boundaries and control transaction behavior.

Basic SQL Commands

SQL is a powerful programming language used to manage and manipulate data stored in RDBMS. It is a standard language used by most relational databases and is essential for working with databases. In the next chapter, we will explore how to create and manage databases using SQL.

SELECT: This command is used to retrieve data from one or more tables.

INSERT: This command is used to add data to a table.

UPDATE: This command is used to modify existing data in a table.

DELETE: This command is used to remove data from a table.

CREATE: This command is used to create a new table.

ALTER: This command is used to modify the structure of an existing table.

DROP: This command is used to delete an existing table.

To use SQL in practice, you will need to have access to an RDBMS. There are many RDBMS options available, including MySQL, Oracle, and Microsoft SQL Server. Once you have access to an RDBMS, you can use a client application or web interface to interact with the database using SQL commands.

Chapter 2(b). SQL Operation

Learning Objective

Create SQL queries that use SELECT, WHERE keywords.

Create SQL queries that use the SQL logical operators, including AND, OR, and NOT

Create SQL queries that use the SQL built-in aggregate functions of SUM, COUNT, MIN, MAX, and AVG with and without the SQL GROUP BY clause.

Data Manipulation Language

The Data Manipulation Language (DML) commands are used to retrieve, insert, update, and delete data in a database. Here are some commonly used DML commands:

INSERT: The INSERT command is used to add data to a table. You can specify the values for each column in the table or use a subquery to insert data from another table.

SELECT: The SELECT command is used to retrieve data from one or more tables. You can use the WHERE clause to filter the data based on specific criteria.

UPDATE: The UPDATE command is used to modify existing data in a table. You can specify the column to update and the new value, and use the WHERE clause to filter the data to be updated.

DELETE: The DELETE command is used to remove data from a table. You can use the WHERE clause to filter the data to be deleted.

INSERT Statement

The INSERT statement is used to add data to a table. The syntax of the INSERT statement is as follows:

```
INSERT INTO table_name (column1, column2, ...)

VALUES (value1, value2, ...);
```

The INSERT INTO clause specifies the name of the table and the columns into which data is to be inserted. The VALUES clause specifies the values to be inserted into the specified columns. For example, the following INSERT statement adds a new record to the "employees" table:

```
INSERT INTO employees (employee_id, first_name, last_name, salary)

VALUES (101, 'John', 'Doe', 60000);
```

SELECT Statement

The SELECT statement is used to retrieve data from one or more tables. The syntax of the SELECT statement is as follows:

```
SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

The SELECT clause specifies the columns to be retrieved from the table. The FROM clause specifies the table or tables from which the data should be retrieved. The WHERE clause is used to filter the data based on specific criteria. For example, the following SELECT statement retrieves all columns from the "employees" table where the salary is greater than 50000:

```
SELECT *

FROM employees

WHERE salary > 50000;
```

UPDATE Statement

The UPDATE statement is used to modify existing data in a table. The syntax of the UPDATE statement is as follows:

```
UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;
```

The UPDATE clause specifies the name of the table to be updated. The SET clause specifies the columns to be updated and their new values. The WHERE clause is used to filter the data to be updated. For example, the following UPDATE statement changes the salary of the employee with the employee_id of 101 in the "employees" table to 65000:

```
UPDATE employees

SET salary = 65000

WHERE employee_id = 101;
```

DELETE Statement

The DELETE statement is used to remove data from a table. The syntax of the DELETE statement is as follows:

```
DELETE FROM table_name

WHERE condition;
```

The DELETE FROM clause specifies the name of the table from which data is to be deleted. The WHERE clause is used to filter the data to be deleted. For example, the following DELETE statement removes all records from the "employees" table where the salary is less than 40000:

```
DELETE FROM employees

WHERE salary < 40000;
```

Aggregation Functions

Aggregation functions are used to perform calculations on multiple rows of a table and return a single value. These functions can be used with the SELECT statement and the GROUP BY clause to group the data based on specific criteria.

SUM Function

The SUM function is used to calculate the sum of values in a column. The syntax of the SUM function is as follows:

```
SELECT SUM(column_name)

FROM table_name;
```

For example, the following query returns the sum of all salaries in the "employees" table:

```
SELECT SUM(salary)

FROM employees;
```

AVG Function

The AVG function is used to calculate the average value of a column. The syntax of the AVG function is as follows:

```
SELECT AVG(column_name)

FROM table_name;
```

For example, the following query returns the average salary of all employees in the "employees" table:

```
SELECT AVG(salary)

FROM employees;
```

MAX Function

The MAX function is used to retrieve the maximum value in a column. The syntax of the MAX function is as follows:

```
SELECT MAX(column_name)

FROM table_name;
```

For example, the following query returns the highest salary in the "employees" table:

```
SELECT MAX(salary)

FROM employees;
```

MIN Function

The MIN function is used to retrieve the minimum value in a column. The syntax of the MIN function is as follows:

```
SELECT MIN(column_name)

FROM table_name;
```

COUNT Function

The COUNT function is used to retrieve the number of rows in a table or the number of rows that meet a specific condition. The syntax of the COUNT function is as follows:

```
SELECT COUNT(*)

FROM table_name;
```

For example, the following query returns the number of employees in the "employees" table:

```
SELECT COUNT(*)

FROM employees;
```

GROUP BY Clause

The GROUP BY clause is used to group the data in a table based on specific criteria. The GROUP BY clause is often used in conjunction with aggregation functions to perform calculations on grouped data.

For example, the following query groups the data in the "employees" table by department and returns the total salary for each department:

```
SELECT department, SUM(salary)

FROM employees

GROUP BY department;
```

Chapter 3(a). Relational Model

Learning Objective

Recognize basic relational terminology

Explain the characteristics of relations.

Classify alternative terminology used in describing the relational model

Concept of Relational Model

The Relational Model is a conceptual model that describes how data can be organized in a relational database. It was developed by E.F. Codd in the 1970s and has since become the most widely used database model.

The Relational Model is based on the principles of mathematics and set theory. It consists of tables, also known as relations, which contain data organized in rows and columns. Each row represents a unique record, and each column represents a specific attribute of the record.

The relational model allows for relationships between tables, which are defined by foreign keys. A foreign key is a column in one table that refers to the primary key of another table.

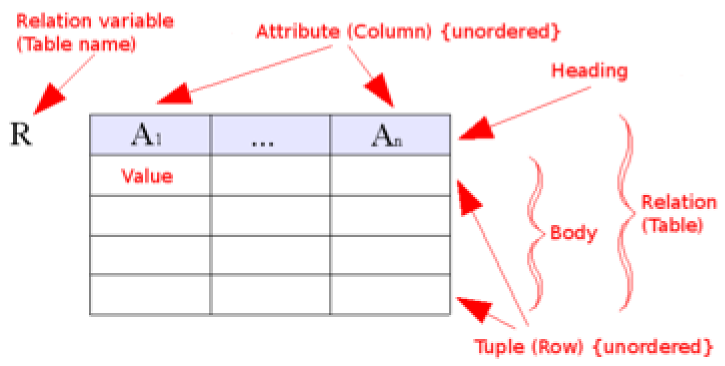


Figure 3.1 Demonstration of Relational Model

Terms of Relational Model

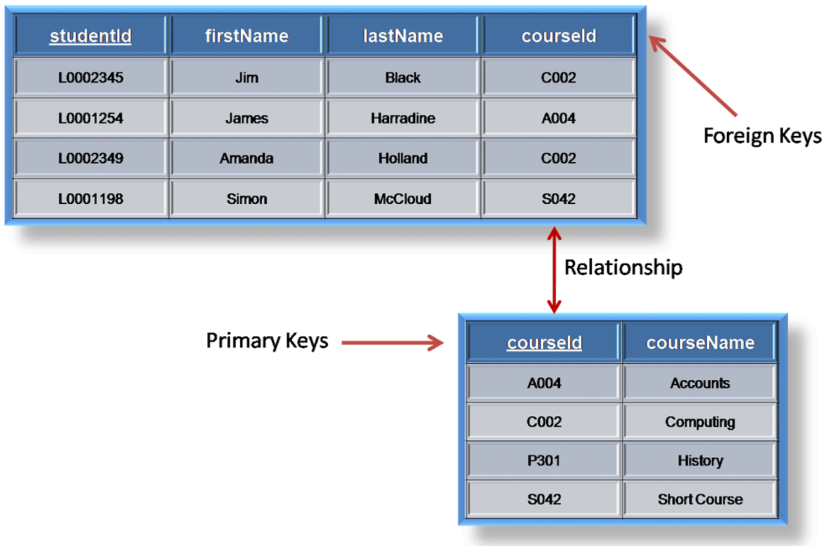
Tables (Relations) Tables, also known as relations, are the fundamental component of the relational model. They are used to store data in rows and columns.

Attributes (Columns) Attributes, also known as columns, represent specific characteristics of the data stored in a table. Each attribute has a name and a data type.

Records (Rows) Records, also known as rows, represent a single instance of data stored in a table. Each row contains data for each attribute defined in the table.

Primary Key The primary key is a unique identifier for each row in a table. It is a column or a combination of columns that uniquely identify each record in a table.

Foreign Key A foreign key is a column in one table that refers to the primary key of another table. It is used to establish relationships between tables.



studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

Figure 3.2 Primary Key and Foreign Key

Features of Relational Model

Flexibility: The relational model is flexible and allows for changes to be made to the data structure without affecting the entire system. Tables can be added or removed as needed, and data can be modified without disrupting other data in the system.

Consistency: The relational model ensures data consistency by enforcing referential integrity. This means that data in one table must match the data in another table, ensuring that the data is accurate and consistent.

Security: The relational model allows for access control, ensuring that only authorized users can access or modify data in the database.

Scalability: The relational model can easily handle large amounts of data, making it a scalable solution for data management.

Chapter 3(b). Normalization

Learning Objective

Identify possible insertion, deletion, and update anomalies in a relation.

Place a relation into BCNF normal form.

Recognize the special importance of domain key normal form.

Concept of Normalization

Normalization is the process of organizing data in a relational database in a way that minimizes data redundancy and avoids data anomalies. The objective of normalization is to make the database efficient, flexible, and scalable while ensuring data consistency and accuracy. In this chapter, we will explore the concept of database normalization, its different forms, and the benefits of normalization.

To understand database normalization, it is important to first understand the concept of a relational database. A relational database is a collection of tables, where each table represents a distinct entity in the database, such as customers, orders, or products. Each table consists of rows and columns, with each row representing a single instance of the entity, and each column representing a specific attribute of that entity, such as name, age, or price. When data is entered into a database, it is often necessary to store the same data in multiple places to ensure that it can be accessed from different parts of the database. However, this redundancy can lead to data inconsistencies, as updates to one instance of the data may not be reflected in all instances. This can result in data anomalies, where the data does not accurately reflect the state of the system.

Functional Dependency

Functional dependency is a concept in database management that describes the relationship between two sets of attributes in a table. It refers to the fact that the value of one attribute, called the dependent attribute, is determined by the value of another attribute, called the determinant attribute. In this chapter, we will explore the concept of functional dependency and its importance in database design.

In a relational database, a table consists of rows and columns. Each column represents a specific attribute of the data, while each row represents a single instance of that data. A functional dependency between two attributes exists when the value of one attribute uniquely determines the value of another attribute. For example, consider a table that stores information about products sold by a company. The table has the following attributes: product_id, product_name, category_id, and category_name. The category_name attribute is functionally dependent on the category_id attribute, because the value of category_name is uniquely determined by the value of category_id. In other words, if we know the value of category_id, we can determine the value of category_name.

We can represent functional dependency using an arrow notation. For example, the functional dependency between category_id and category_name can be represented as follows:

category_id → category_name.

This notation indicates that the value of category_name is functionally dependent on the value of category_id.

Normal Forms

There are several levels of normalization, called normal forms, that describe the level of data redundancy and anomalies present in a database. The most commonly used normal forms are first normal form (1NF), second normal form (2NF), and third normal form (3NF).

1NF requires that each column in a table contains only atomic values, meaning that each value in the column is indivisible. This means that each row in the table must have a unique identifier, known as a primary key, that distinguishes it from all other rows in the table.

2NF builds on 1NF by ensuring that all non-key attributes in a table are dependent on the entire primary key, rather than just a part of it. This ensures that each table contains only one type of data, and that each data item is represented only once.

3NF goes one step further, by removing any data that is not directly related to the primary key, and storing it in a separate table. This ensures that the data is not duplicated across multiple tables, and that each table contains only information that is directly relevant to the primary key.

BCNF, or Boyce-Codd Normal Form, is a higher level of normalization in database management that ensures that a table is free from certain types of anomalies that may occur due to functional dependencies between attributes. A table is said to be in BCNF if and only if every determinant in the table is a candidate key.

Importance of Normalization

Normalization offers several benefits to database design and management. By reducing data redundancy, normalization makes databases more efficient, as less storage space is required to store the data. It also makes databases more flexible, as changes to the database structure can be made more easily, without affecting other parts of the database. Additionally, normalization helps ensure data accuracy and consistency, as each piece of data is stored in only one place and can be easily updated.

Database normalization is a crucial part of designing and managing a relational database. By breaking down tables into smaller, more specialized tables, and linking them together through relationships, normalization reduces data redundancy and anomalies, making databases more efficient, flexible, and accurate.

Chapter 4. Database Design by Normalization

Learning Objective

Design updatable databases to store data received from another source.

Use SQL to access table structure.

Compare the advantages and disadvantages of normalization.

Process of Database Design by Normalization

Database design by normalization is the process of organizing data in a database to minimize redundancy and ensure data integrity and consistency. Normalization is a technique used to eliminate data redundancy by breaking down a large table into smaller tables and establishing relationships between them. In this chapter, we will discuss the principles of normalization and the different normalization forms that are used in database design.

The goal of normalization is to ensure that each piece of data is stored in only one place in the database. This eliminates redundancy and minimizes the potential for errors and inconsistencies in the data. Normalization is achieved by breaking down a large table into smaller tables and establishing relationships between them. The process of normalization involves several steps, each of which is called a normal form.

The first normal form (1NF) requires that each attribute in a table be atomic, meaning it cannot be broken down into smaller pieces. For example, a customer's name should be stored as separate first and last name attributes rather than as a single full name attribute.

The second normal form (2NF) requires that each non-key attribute in a table be fully functionally dependent on the primary key. In other words, each non-key attribute should depend on the entire primary key, not just part of it.

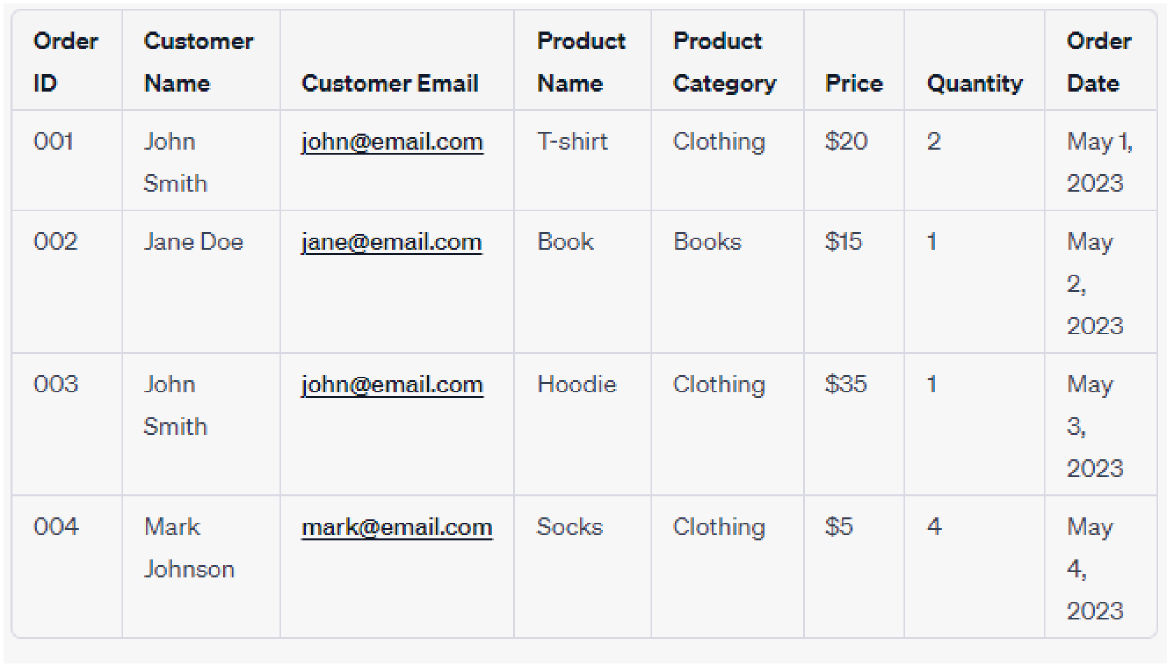
The third normal form (3NF) requires that each non-key attribute be dependent only on the primary key, and not on any other non-key attributes. This eliminates transitive dependencies, which are dependencies between non-key attributes.

The fourth normal form (4NF) requires that each multi-valued dependency in a table be eliminated. A multi-valued dependency is a relationship between two sets of attributes where the value of one set of attributes determines the values of another set of attributes.

Each normal form builds on the previous one, with each higher level of normalization providing greater data consistency and integrity. However, it is important to note that not all tables need to be fully normalized to 5NF. The level of normalization required depends on the specific needs of the database and the application using it.

Example of Database Design by Normalization

Think about a real-world example of table normalization using a customer order management system for an online retailer. Suppose we have the following data in a single table named "Orders":



Order ID	Customer Name	Customer Email	Product Name	Product Category	Price	Quantity	Order Date
001	John Smith	john@email.com	T-shirt	Clothing	$20	2	May 1, 2023
002	Jane Doe	jane@email.com	Book	Books	$15	1	May 2, 2023
003	John Smith	john@email.com	Hoodie	Clothing	$35	1	May 3, 2023
004	Mark Johnson	mark@email.com	Socks	Clothing	$5	4	May 4, 2023

There are some issues of above table:

The "Customer Name" and "Customer Email" attributes are not atomic, meaning they can be further broken down into smaller pieces (first normal form violation).

The "Product Name" and "Product Category" attributes are repeated for each order, leading to data redundancy (first normal form violation).

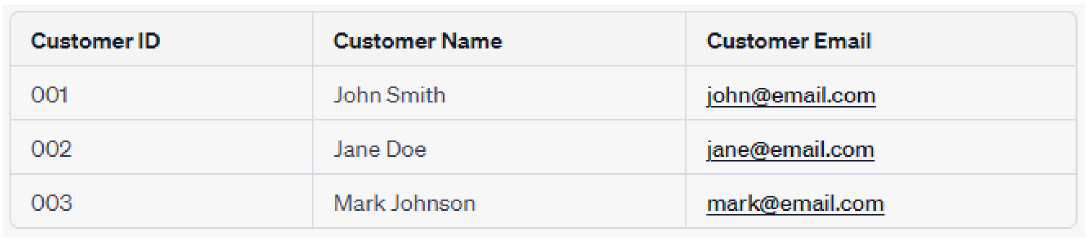
The "Price" attribute is dependent on the "Product Name" and "Product Category" attributes, rather than solely on the primary key "Order ID" (second normal form violation).

The "Quantity" attribute is also dependent on the "Product Name" and "Product Category" attributes (second normal form violation).

The "Order Date" attribute is not dependent on any other attributes, but it is not fully dependent on the primary key "Order ID" (second normal form violation).

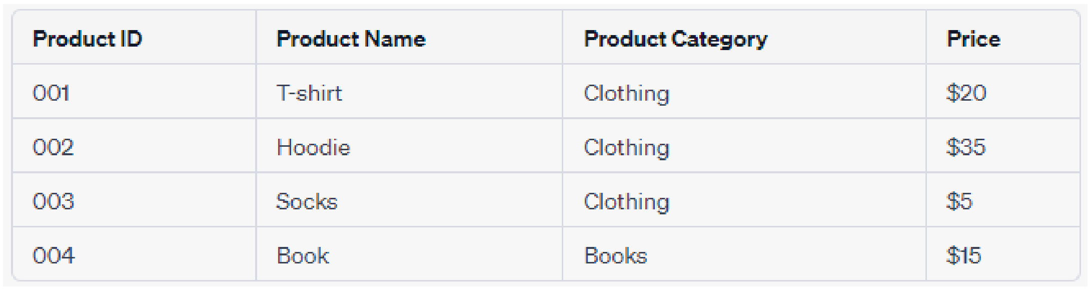
To normalize this table, we can create three separate tables: "Customers", "Products", and "Orders".

Customer Table:



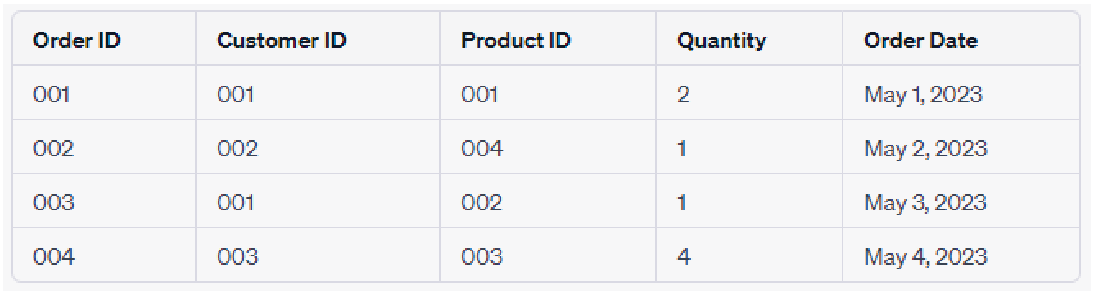
Customer ID	Customer Name	Customer Email
001	John Smith	john@email.com
002	Jane Doe	jane@email.com
003	Mark Johnson	mark@email.com

Product Table:



Product ID	Product Name	Product Category	Price
001	T-shirt	Clothing	$20
002	Hoodie	Clothing	$35
003	Socks	Clothing	$5
004	Book	Books	$15

Order Table:



Order ID	Customer ID	Product ID	Quantity	Order Date
001	001	001	2	May 1, 2023
002	002	004	1	May 2, 2023
003	001	002	1	May 3, 2023
004	003	003	4	May 4, 2023

After normalization, the Customers table contains unique customer data, the Products table contains unique product data, and the Orders table establishes a relationship between the two. This design eliminates redundancy and ensures data consistency.

Chapter 5. Data Modeling with Entity-Relationship Model

Learning Objective

Recognize the two-phase data modeling/database design process.

Infer the purpose of the data modeling process.

Execute entity-relationship (E-R) diagrams.

Data Modeling by Entity-Relationship Model

Data modeling is the process of creating a conceptual representation of data and its relationships to help design and develop effective databases. The Entity-Relationship (ER) model is a widely used data modeling technique that helps developers to identify and describe the entities, attributes, and relationships between them.

The Entity-Relationship model consists of three main components: Entities, Attributes, and Relationships.

Entities: An entity is a real-world object or concept that can be identified and described in a database. Examples of entities include customers, products, orders, and employees. Entities are represented by rectangles in an ER diagram.

Attributes: Attributes describe the characteristics or properties of an entity. For example, the customer entity might have attributes such as name, address, phone number, and email address. Attributes are represented by ovals in an ER diagram.

Relationships: Relationships define the connections or associations between entities. For example, a customer might place an order for one or more products. The relationship between the customer and order entities is represented by a diamond-shaped connector in an ER diagram.

Process of Creating an ERD

The process of creating an ER diagram involves the following steps:

1. Identify Entities and Attributes The first step in creating an ER diagram is to identify the entities and their attributes. This involves understanding the business requirements and the data that needs to be stored in the database. Entities and their attributes can be identified by asking questions such as:

- What are the main objects or concepts in the business?
- What information needs to be stored about each object or concept?
- What are the important characteristics or properties of each object or concept?

2. Identify Relationships The next step is to identify the relationships between the entities. Relationships can be one-to-one, one-to-many, or many-to-many. One-to-one relationships exist when each entity in one entity set is related to only one entity in the other entity set. One-to-many relationships exist when each entity in one entity set is related to one or more entities in the other entity set. Many-to-many relationships exist when each entity in one entity set is related to one or more entities in the other entity set, and vice versa.

3. Create the ER Diagram Once the entities, attributes, and relationships have been identified, the ER diagram can be created. The ER diagram visually represents the entities, attributes, and relationships using rectangles, ovals, and diamonds, respectively.

4. Refine the ER Diagram The final step in creating an ER diagram is to refine it. This involves reviewing the diagram to ensure that it accurately represents the business requirements and the data that needs to be stored in the database. If necessary, changes can be made to the diagram to improve its accuracy and completeness.

Case Analysis of ERD

Let’s analyze an example ERD for a simple online bookstore.

Entities: We can identify the following entities in our online bookstore:

- Customer
- Book
- Author
- Publisher
- Order

Attributes: We can identify the following attributes for each entity:

Customer:

- Customer ID (primary key)
- Name
- Email
- Phone Number

Book:

- Book ID (primary key)
- Title
- ISBN
- Publication Year
- Price

Author

- Author ID (primary key)
- Name

Publisher

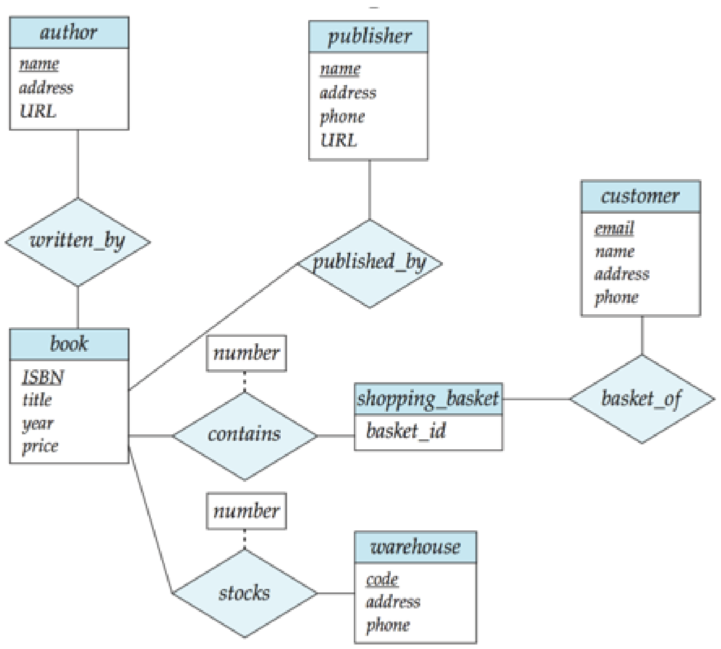
- Publisher ID (primary key)
- Name

Order

- Order ID (primary key)
- Customer ID (foreign key)
- Order Date
- Total Price

Relationships We can identify the following relationships between the entities:

- A customer can place zero or more orders.
- An order must be placed by one and only one customer.
- An order can include one or more books.
- A book can be included in zero or more orders.
- A book must be written by one or more authors.
- An author can write one or more books.
- A book must be published by one and only one publisher.
- A publisher can publish one or more books.



In the ER diagram, each entity is represented by a rectangle, and each attribute is represented by an oval. The relationships between the entities are represented by lines connecting the rectangles, with the relationship type indicated by a diamond shape at the end of the line. The primary key of each entity is underlined, and the foreign key is indicated by a dashed underline.

This ERD illustrates the structure of our online bookstore's database and can be used as a blueprint for the actual database design and implementation.

Chapter 6. Transforming Data Models into Database Design

Learning Objective

Recognize how to transform data models into database designs.

To be able to identify primary keys and understand when to use a surrogate key.

Execute referential integrity constraints.

Process of Transforming ERD to Database

Transforming a data model into a database design is an essential step in the development of any database application. A data model provides a conceptual view of the data, while a database design provides a physical implementation of the data model in the form of a database schema. In this chapter, we will discuss the process of transforming a data model into a database design.

The first step in transforming a data model into a database design is to map the entities and relationships in the data model to tables in the database schema. Each entity in the data model becomes a table in the database schema, and each relationship in the data model becomes a foreign key constraint between the corresponding tables in the database schema.

For example, let's consider a data model for a library management system. The data model may have entities such as Books, Authors, and Publishers, and relationships such as "A Book is written by an Author" and "A Book is published by a Publisher." To transform this data model into a database design, we would create tables for each entity and establish relationships between the tables using foreign keys.

The Books entity would become a table in the database schema with columns for attributes such as BookID, Title, and PublicationDate. The Authors entity would become another table in the database schema with columns for attributes such as AuthorID, FirstName, and LastName. To establish the "A Book is written by an Author" relationship, we would add a foreign key column for AuthorID in the Books table, referencing the Authors table.

Similarly, the Publishers entity would become a table in the database schema with columns for attributes such as PublisherID, Name, and Address. To establish the "A Book is published by a Publisher" relationship, we would add a foreign key column for PublisherID in the Books table, referencing the Publishers table.

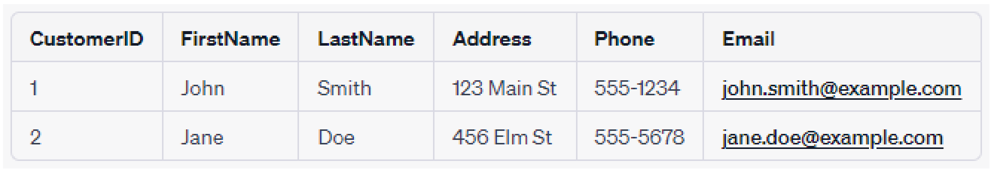
Once we have mapped the entities and relationships in the data model to tables and foreign keys in the database schema, we need to define the constraints and indexes on the tables. Constraints ensure that the data in the tables is valid and consistent, while indexes improve the performance of queries on the tables.

For example, we may define a primary key constraint on the BookID column in the Books table to ensure that each book has a unique identifier. We may also define a unique constraint on the combination of FirstName and LastName columns in the Authors table to ensure that each author has a unique name.

Finally, we may define indexes on the foreign key columns to improve the performance of queries that join the tables. For example, we may define an index on the AuthorID column in the Books table to improve the performance of queries that retrieve books written by a particular author.

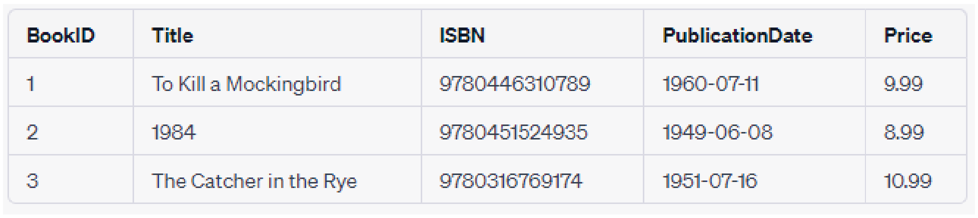
Example of Database Design

Customer Table:



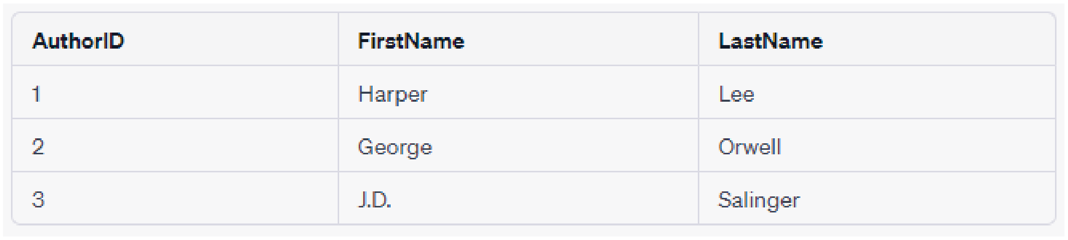
CustomerID	FirstName	LastName	Address	Phone	Email
1	John	Smith	123 Main St	555-1234	john.smith@example.com
2	Jane	Doe	456 Elm St	555-5678	jane.doe@example.com

Book Table:



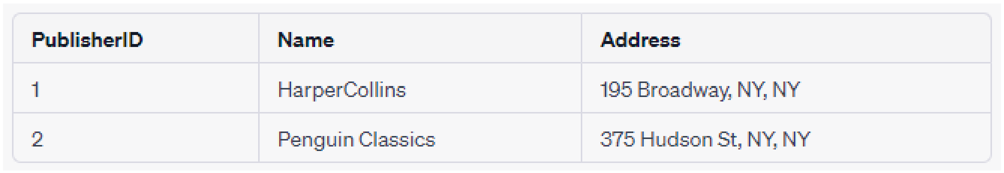
BookID	Title	ISBN	PublicationDate	Price
1	To Kill a Mockingbird	9780446310789	1960-07-11	9.99
2	1984	9780451524935	1949-06-08	8.99
3	The Catcher in the Rye	9780316769174	1951-07-16	10.99

Author Table:



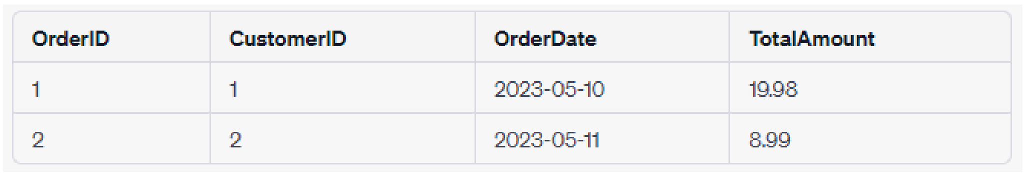
AuthorID	FirstName	LastName
1	Harper	Lee
2	George	Orwell
3	J.D.	Salinger

Publisher Table:



PublisherID	Name	Address
1	HarperCollins	195 Broadway, NY, NY
2	Penguin Classics	375 Hudson St, NY, NY

Order Table:



OrderID	CustomerID	OrderDate	TotalAmount
1	1	2023-05-10	19.98
2	2	2023-05-11	8.99

Referential Integrity Constraints:

Referential integrity constraints are a set of rules that ensure that the relationships between tables in a relational database are maintained. These constraints prevent the creation of invalid data by ensuring that each foreign key value in a child table matches a primary key value in a parent table. In this chapter, we will discuss the importance of referential integrity constraints and how they are implemented in a database.

Referential integrity constraints are important for maintaining the accuracy and consistency of data in a database. Without these constraints, it would be possible to create orphaned records or data that references non-existent values in a parent table. This can lead to data inconsistencies and errors in the application that relies on the database.

Referential integrity constraints ensure that data is always consistent and accurate by enforcing the following rules:

- A foreign key value in a child table must match a primary key value in a parent table.
- A record cannot be deleted from a parent table if it is referenced by a foreign key in a child table.
- If a primary key value is updated in a parent table, all foreign key values that reference that primary key value must be updated as well.

Chapter 7. Transact-SQL

Learning Objective

Learn the concept and significance of T-SQL

Understand T-SQL commands and queries

Concept of T-SQL

T-SQL (Transact-SQL) is a programming language used to manage and manipulate data in Microsoft SQL Server databases. T-SQL is an extension of SQL (Structured Query Language) that adds procedural programming constructs such as control flow statements, variables, and functions. In this chapter, we will discuss the basics of T-SQL and how it is used to interact with SQL Server databases.

T-SQL is a high-level programming language that allows developers to write code that interacts with SQL Server databases. It is used to create, modify, and retrieve data from database tables, and can also be used to define and manipulate other database objects such as views, stored procedures, and triggers.

The syntax of T-SQL is similar to that of other programming languages, with statements separated by semicolons (;) and comments indicated by two hyphens (--). T-SQL statements are executed one at a time, and the results are returned as a dataset.

T-SQL Transact SQL Language

T-SQL Code:

```
-- Create a new table called "Employees"

CREATE TABLE Employees (

    EmployeeID INT PRIMARY KEY,

    FirstName VARCHAR(50),

    LastName VARCHAR(50),
    Title VARCHAR(50),

    HireDate DATE,

    Salary MONEY

);


-- Insert some data into the "Employees" table

INSERT INTO Employees (EmployeeID, FirstName, LastName, Title, HireDate, Salary)

VALUES (1, 'John', 'Doe', 'Manager', '2020-01-01', 75000),

       (2, 'Jane', 'Smith', 'Assistant Manager', '2021-02-01', 50000),

       (3, 'Bob', 'Johnson', 'Sales Rep', '2019-03-01', 40000);


-- Retrieve the data from the "Employees" table

SELECT EmployeeID, FirstName, LastName, Title, HireDate, Salary

FROM Employees;
```

Key Features of T-SQL

T-SQL is used to interact with SQL Server databases through the use of SQL Server Management Studio (SSMS). SSMS is a graphical user interface that allows developers to write T-SQL code, execute queries, and view the results. Some key features of T-SQL include:

- Variables: Variables can be used to store values that can be used in T-SQL statements. Variables are declared using the DECLARE keyword, and their values can be assigned using the SET keyword.
- Control flow statements: T-SQL supports control flow statements such as IF/ELSE, WHILE, and CASE that allow developers to execute code based on certain conditions.
- Functions: T-SQL includes a large number of built-in functions that can be used to manipulate data, perform calculations, and convert data types.
- Cursors: Cursors can be used to iterate through a dataset and perform operations on each row.

Some of the most common tasks performed using T-SQL include:

- Creating and modifying database objects such as tables, views, stored procedures, and triggers.
- Inserting, updating, and deleting data in database tables.
- Retrieving data from database tables using SELECT statements.
- Joining multiple tables together to retrieve data.
- Aggregating data using functions such as SUM, AVG, and COUNT.

T-SQL is a powerful programming language that is used to manage and manipulate data in SQL Server databases. It allows developers to create complex queries and perform advanced data manipulation tasks. By mastering T-SQL, developers can build robust and efficient applications that interact with SQL Server databases.

Chapter 8. NoSQL

Learning Objective

Understand the concept of NoSQL.

Learn the key features of NoSQL.

Recognize the applications of NoSQL database systems.

Definition of NoSQL

NoSQL (not only SQL) is a term used to describe a class of database management systems that do not use the traditional relational database model based on tables and SQL (Structured Query Language). Instead, NoSQL databases use different models to store and manage data, often with greater flexibility and scalability. In this chapter, we will discuss the basics of NoSQL databases, their key features, and use cases.

NoSQL databases are designed to handle large volumes of unstructured or semi-structured data, such as documents, videos, images, and social media posts, which may not fit neatly into tables and columns. NoSQL databases use different models to organize and query data, such as key-value, document, column-family, and graph models.

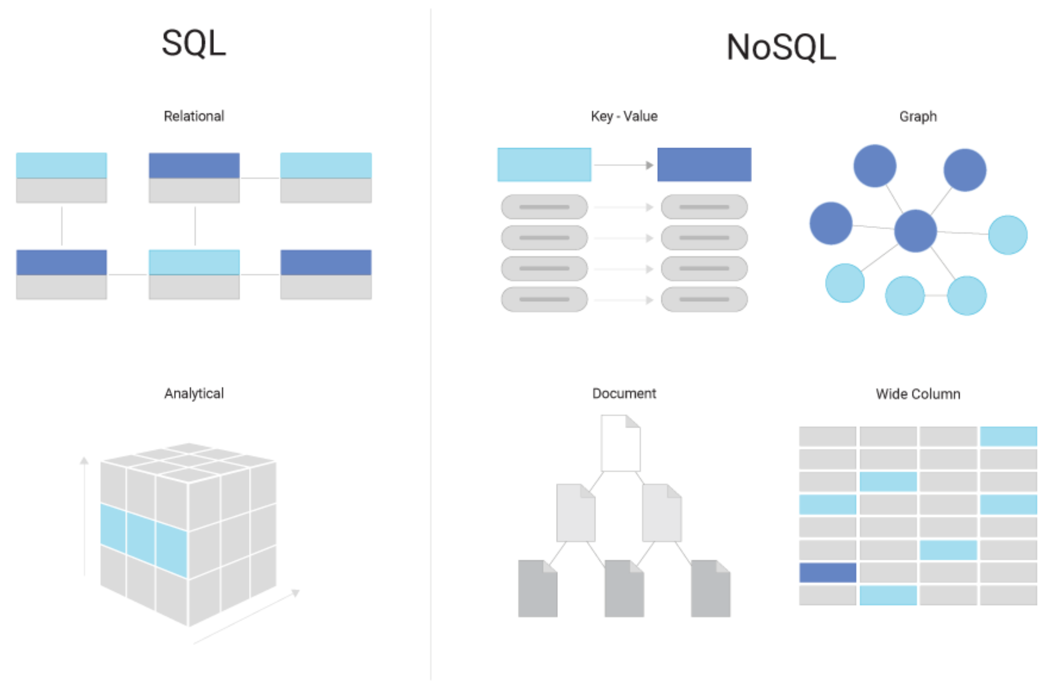


Figure 8.1 SQL and NoSQL

NoSQL databases have the following features:

Schemaless: NoSQL databases do not have a fixed schema like traditional SQL databases. Instead, data can be added or modified without first defining a schema, which allows for greater flexibility and scalability.

Horizontal Scalability: NoSQL databases are designed to scale horizontally, meaning that they can add more servers to increase capacity and handle larger volumes of data.

Distributed Architecture: NoSQL databases often use a distributed architecture, in which data is spread across multiple servers or nodes. This allows for greater fault tolerance and higher availability.

High Performance: NoSQL databases are often designed to be highly performant, with fast read and write operations and low latency.

Applications of NoSQL Databases

NoSQL databases offer a flexible, scalable, and high-performance alternative to traditional relational databases. They are well-suited for a variety of use cases, from web applications and big data to real-time data processing and IoT. By understanding the key features of NoSQL databases and their use cases, developers can choose the right database for their application and optimize its performance. NoSQL databases have the following applications:

Web Applications: NoSQL databases are well-suited for web applications that require high scalability and fast read and write operations. Examples include social media platforms, e-commerce sites, and content management systems.

Big Data: NoSQL databases are often used in big data applications, where large volumes of unstructured or semi-structured data need to be processed and analyzed. Examples include data warehousing, data analytics, and machine learning.

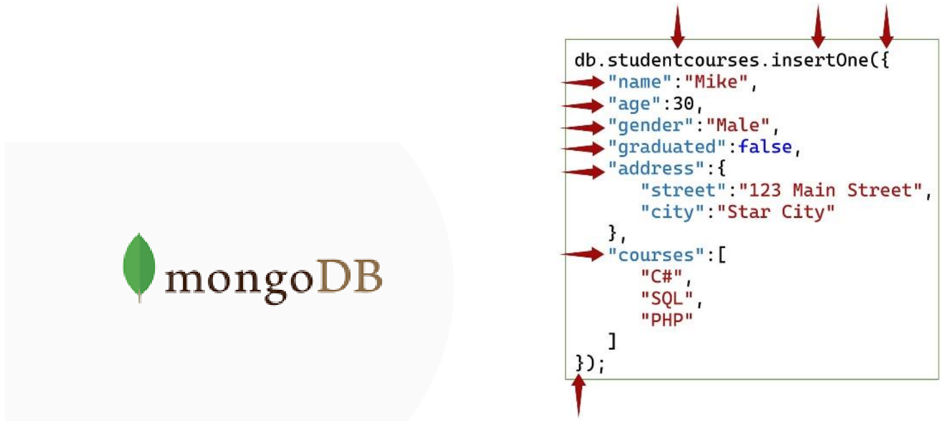
Real-time Data Processing: NoSQL databases are often used in applications that require real-time data processing, such as stock trading, online gaming, and social media analytics.

Internet of Things (IoT): NoSQL databases are used in IoT applications, where large volumes of data are generated from sensors and devices, and need to be processed and analyzed in real-time.

Example of NoSQL Code

NoSQL code using the MongoDB database, which uses a document-based data model. This code connects to the MongoDB server using the MongoDB driver for Node.js, inserts a document into the "mycollection" collection with a name, age, and city field, and then retrieves all documents in the collection with a city field equal to "New York" using the find() method.

```
// Connect to the MongoDB server

const MongoClient = require('mongodb').MongoClient;

const uri =
"mongodb+srv://<username>:<password>@cluster0.mongodb.net/test?retryWrites=true&w=majority";

const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });

client.connect(err => {

  const collection = client.db("mydatabase").collection("mycollection");



  // Insert a document into the collection

  collection.insertOne({ name: "John", age: 30, city: "New York" }, function(err, res) {

    if (err) throw err;

    console.log("1 document inserted");

    client.close();

  });
  // Find documents in the collection

  collection.find({ city: "New York" }).toArray(function(err, result) {

    if (err) throw err;

    console.log(result);

    client.close();

  });

});
```



```
db.studentcourses.insertOne({
   "name":"Mike",
   "age":30,
   "gender":"Male",
   "graduated":false,
   "address":{
      "street":"123 Main Street",
      "city":"Star City"
   },
   "courses":[
      "C#",
      "SQL",
      "PHP"
   ]
});
```

Figure 8.2 Example of MongoDB