MATS UNIVERSITY

NAAC GRADE A+ ACCREDITED UNIVERSITY

# MATS CENTRE FOR DISTANCE & ONLINE EDUCATION

## Database Management System

**Diploma in Computer Application (DCA)**
**Semester - 1**

SELF LEARNING MATERIAL

MATS UNIVERSITY
www.matsuniversity.ac.in

NAAC GRADE A+ ACCREDITED UNIVERSITY

# Diploma in Computer Application (DCA)

## Database Management System

## DCA104

**COURSE DEVELOPMENT EXPERT COMMITTEE**

Prof. (Dr.) K. P. Yadav, Vice Chancellor, MATS University, Raipur, Chhattisgarh

Prof. (Dr.) Omprakash Chandrakar, Professor and Head, School of Information Technology, MATS University, Raipur, Chhattisgarh

Prof. (Dr.) Sanjay Kumar, Professor and Dean, Pt. Ravishankar Shukla University, Raipur, Chhattisgarh

Prof. (Dr.) Jatinder kumar R. Saini, Professor and Director, Symbiosis Institute of Computer Studies and Research, Pune

Dr. Ronak Panchal, Senior Data Scientist, Cognizant, Mumbai

Mr. Saurabh Chandrakar, Senior Software Engineer, Oracle Corporation, Hyderabad

**COURSE COORDINATOR**

Prof. (Dr.) Omprakash Chandrakar, Professor and Head, School of Information Technology, MATS University, Raipur, Chhattisgarh

**COURSE PREPARATION**

Prof. (Dr.) Omprakash Chandrakar, Professor and Head and Mrs. Shraddha Doye, Assistant Professor, School of Information Technology, MATS University, Raipur, Chhattisgarh

**March, 2025**

**ISBN: 978-81-986955-2-9**



Printed & Published on behalf of MATS University, Village-Gullu, Aarang, Raipur by Mr. Meghanadhudu Katabathuni, Facilities & Operations, MATS University, Raipur (C.G.)



Print date: The Digital Press, Krishna Complex, Raipur – 492001 (Chhattisgarh)

# Acknowledgement

The material (pictures and passages) we have used is purely for educational purposes. Every effort has been made to trace the copyright holders of material reproduced in this book. Should any infringement have occurred, the publishers and editors apologize and will be pleased to make the necessary corrections in future editions of this book.

# COURSE INTRODUCTION

Database Management Systems (DBMS) are essential for organizing, storing, and managing data efficiently. This course provides a comprehensive understanding of database concepts, data modeling, relational models, and database operations. Students will gain theoretical knowledge and practical skills in designing databases, managing tables, and performing data manipulation tasks. The course aims to equip learners with the foundational principles needed for effective database administration and development.

**Module 1: Introduction to Database Management System**

A Database Management System (DBMS) is a crucial technology that allows for efficient data storage, retrieval, and management. This Module introduces the fundamental concepts of databases, types of database systems, and their applications across various industries. Students will understand the role of DBMS in modern data-driven environments.

**Module 2: Data Modeling and Database Design**

Data modeling is a vital step in designing structured and efficient databases. This Module covers Entity-Relationship (ER) modeling, normalization techniques, and schema design principles. Understanding data modeling ensures proper database structuring, reducing redundancy and enhancing data integrity.

**Module 3: Relational Model**

The relational model is the foundation of most modern databases, defining how data is organized and accessed using tables. This Module explores key relational concepts such as primary keys, foreign keys, integrity constraints, and relational algebra. Students will learn how relational databases facilitate data consistency and efficient query execution.

**Module 4: Managing Database and Table**

Effective database management involves creating, modifying, and maintaining databases and tables. This Module covers SQL commands for database creation, table structures, indexing, and constraints. Understanding these techniques is essential for efficient database organization and maintenance.

**Module 5: Data Manipulation**

Data manipulation allows users to retrieve, insert, update, and delete records within a database. This Module focuses on SQL commands such as SELECT, INSERT, UPDATE, and DELETE, as well as advanced data retrieval techniques like joins and subqueries. Mastering data manipulation ensures efficient handling and analysis of stored information**.**

# MODULE 1
# INTRODUCTION TO DATABASE MANAGEMENT SYSTEM

## 1.0 LEARNING OUTCOMES

- Understand the fundamental concepts and purpose of a Database Management System (DBMS).
- Learn about data abstraction, instances, schemas, and data models.
- Understand the different database languages (DDL and DML).
- Learn about database architecture (Two-tier and Three-tier).
- Identify the roles of database users and administrators.
- Get an introduction to Data Mining, Data Warehousing, Big Data, and Data Analytics.

# Unit 1.1: Introduction to Database

## 1.1.1 Introduction and Purpose of Database

Integrated relationship jammed the data as it never been handled before. Manipulation. This introduced a completely new way to manage data allowing thousands of with hierarchical and network models leading to the relational model, which is still the dominant paradigm today. Shorthand for the idea of one table storing information in the form of rows and columns, allowing for versatile querying and data rise of database systems. The first database management systems (DBMS) were developed in the 1960s, were just simple text files that had little organizing or retrieval features. With the quantity of data growing, requirements from flat files became limiting and the data, with a never-ending growing need for better data management. Data storage had a humble beginning back when computers were in their infancy flat files, which book to complex ERP (Enterprise Resource Planning) systems. Database history is the history of how people find and use a big deal of information that can be accessed, updated or deleted quickly. Their purpose being to be the digital backbone for hundreds of applications - from a basic address definition can only begin to half justice to their degree of influence on contemporary civilization. In short, very typically a database is an efficient mechanism of storing nonetheless, this basic the heart of organizations, allowing them to flourish in a data-centered world. And report data, giving great insights about business performance and trends. Databases are work from the same information. In addition, database helps to analyze throughout the organization. They also facilitate data sharing and collaboration, allowing different departments to businesses as they enable organizations to manage their data effectively. However, in capable hands, databases allow for the organization of data in a consistent and accurate way make data-driven decisions, streamline operations, and improve customer service. Databases are crucial to databases have in modern-day organizations. They play a crucial role in storing and managing vital business data, allowing companies to There is no overstating the degree of importance made them well-suited for applications to analyze complicated networks — for instance social networks and recommendation systems edges—also known as

relationships between nodes to represent connections between entities. The same has them. For example, graph databases like Neo4j and Amazon Neptune store data as nodes and databases: Examples of object-oriented databases include Objected and Versant. Applications that require complex relationships and data structures are well-matched for stuff like web applications, big data analytics, etc. Object Monod, Cassandra, Reds) are designed to accommodate unstructured or semi-structured data (e.g., documents, key-value pairs, graphs). They scale broadly and offer flexibility, which is what makes them perfect for fit. Nasal databases (e.g., Microsoft SQL Server). For applications that "need" structured data and complex SQL queries, they remain a natural based on their benefits and drawbacks. The most widely used type of database is relational databases, which use tables to organize information and SQL (Structured Query Language) to retrieve and modify information (Myself, Oracle, But no databases are identical and can be broadly organized into a few categories records, tax administration, and public services to make sure that the government functions in transparency and in an efficient manner. Information systems, libraries, and online learning platforms that provide quality education services. Databases in the government sector are used in citizen used in telecommunications (for call data records CDRs, network management and customer billing), contributing to the reliability and efficiency of telecommunication services. In education, databases help in managing student their operations and improve customer experience. They are and integrity of transactions are top priority. Databases play a critical role in many sectors including retail, where they are employed for inventory management, customer relationship management (CRM), and e-commerce, helping retailers optimize providers to deliver personalized care and improve patient outcomes. Examples include transaction processing, fraud detection, and risk management in the financial sector where security have a plethora of applications across multiple sectors and domains. Examples: In the health domain, databases provide storage for patient records, medical images, and clinical data, allowing healthcare Databases.

The role of databases in modern organizations extends beyond data storage and retrieval. They are integral to business intelligence (BI) and data warehousing, enabling organizations to analyze large

volumes of data and gain valuable insights. Data warehouses, such as Amazon Redshirt and Snowflake, are designed to store and analyze historical data, providing a comprehensive view of business performance. BI tools, such as Tableau and Power BI, connect to databases and data warehouses, allowing users to create interactive dashboards and reports. Databases also play a crucial role in application development, serving as the backend for web applications, mobile apps, and enterprise systems. Developers use database management systems (DBMS) to create and manage databases, ensuring data integrity and performance. The choice of DBMS depends on the specific requirements of the application, such as scalability, performance, and data consistency. Furthermore, databases are essential for data security and compliance. Organizations must protect their sensitive data from unauthorized access, ensuring compliance with regulations such as GDPR, HIPAA, and CCPA. Database security measures include access control, encryption, and auditing. Access control ensures that only authorized users can access specific data. Encryption protects data from unauthorized access, even if it is intercepted. Auditing tracks user activity, providing a record of who accessed what data and when. Databases also support data backup and recovery, ensuring that data can be restored in the event of a system failure or data loss. Regular backups and disaster recovery plans are essential for maintaining business continuity. The evolution of database technology continues to shape the landscape of data management. Cloud databases, such as Amazon RDS, Azure SQL Database, and Google Cloud SQL, offer scalability, flexibility, and cost-effectiveness, enabling organizations to manage their data in the cloud. Database as a service providers handle database administration tasks, such as provisioning, patching, and backups, allowing organizations to focus on their core business. In-memory databases, such as Reds and SAP HANA, store data in RAM, providing extremely fast data access. They are well-suited for applications that require real-time data processing and analysis. New database technologies, such as block chain databases, are emerging, offering decentralized and immutable data storage. Block chain databases can enhance data security and transparency, making them suitable for applications such as supply chain management and digital identity. In conclusion, databases are fundamental to modern

organizations, serving as the cornerstone of data management and enabling a wide range of applications. From storing and managing critical business information to supporting data analysis and application development, databases play a vital role in driving innovation and improving operational efficiency. The diverse types of databases, including relational, Nasal, object-oriented, and graph databases, cater to different data requirements and application needs. The applications of databases span across various industries, impacting healthcare, finance, retail, and telecommunications, education, and government sectors. As database technology continues to evolve, cloud databases, in-memory databases, and block chain databases are shaping the future of data management, offering scalability, performance, and security. Organizations must embrace these advancements to remain competitive in the data-driven world, ensuring they can effectively manage, analyze, and protect their valuable data assets.

### 1.1.2 View of Data – Abstraction, Instances, Schemas, and Models

Abstraction helps demystify the complexities of data management and offers a user-friendly environment to work in while developing data-centric applications. Features and relationships those are not always important for any user or application. This the data. This is important because real global data can be highly complicated, with lots of schemas, and data models that are important for designing and implementing a separate database system. At its heart, data abstraction is the hiding of unnecessary details from users in order to provide a simplified view of with data at various granularities. This Module discusses the fundamental aspects of this concept data abstraction, instances, of information in databases and other information systems, data management concepts and processes, and what, if any, is accessible as a view of data to the end-user. It helps abstract the complexity of the data, allowing users and applications to interact The term "View of Data" refers to the perspective configured for a specific purpose so the user only sees the data they need to see, rather than being overwhelmed via complexity in the entire database. Applications or business functionalities. And they can be describes a portion of the database. It presents user specific view for concerned with this level. The view level, the most abstract level of the database

the structure of the data, without caring about how the data is stored physically. Database designers and application developers are most what data is stored in the database and the relationships among those data. Instead, it describes administrators and system programmers. The logical level is the next level of abstraction and with file organization, indexing, and data compression. This level is mainly relevant to database physically stored in the storage devices. This involves dealing varying levels of detail. The physical level is the lowest level of abstraction, representing how the data abstraction happens at different levels with. Data that this database would hold, for example a specific list of books, authors, and borrowers at a specific point in time. And borrowers and the relations between these. The a database for a library. E.g., the schema would describe tables for books, authors, constantly changing as records are inserted, updated, or deleted. Here is an example of and relationships exist, and any constraints. The schema remains relatively constant, whereas the instance is whole database. It defines the structure of the data, including which types of data snapshot of the database, displaying the actual values of the data elements. A schema is, however, a design for the as a database state) is the data that the database holds at a specific point in time. It is a instance and the schema. Overall, an instance (often referred to The two main concepts that are barely thinking about abstraction are the database structure helps in improving understanding and communication which is what the ER model provides using diagrams. links between entities, e.g. “writes” between authors and books. A visual representation of name for

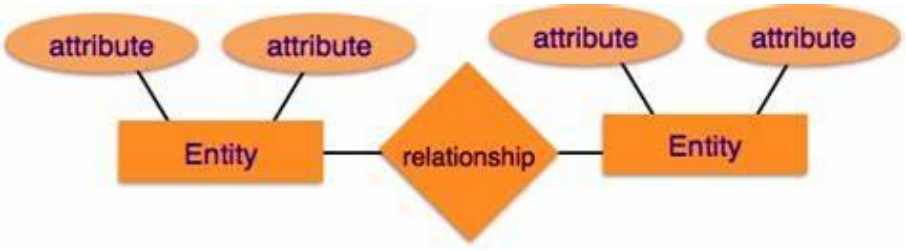



*Figure 1.1.1: Data Model*
*[Source: https://th.bing.com/]*

books, or borrower ID in the case of borrowers. Relationships are Borrower. Examples are title and author proposed in 1976 as a means of expressing more complex data relationships than the common hierarchical or network models of the day.

Entities are real-world objects or concepts. Books. Book, Author, and are Multiple Types of Data Models with their perspectives. The entity-relationship (ER) model was first database and communicate with database designers, application developers, and users. Features of Data Models There show a synopsis of the data as we focused on the application domain entities, attributes, and relationships. Data models are used to help design the is very close to the concept of data models. These this concept.

it is not only flexible but also simpler than earlier models (such as hierarchical models or network models). Relationships. The relational model has grown in popularity since parent-child relationship with the records, meaning there's a hierarchy. Another older data model, the network model, depicts data in terms of a graph, in which records can have multiple parent-child structure. The record has a versatile and functional model for accessing and modifying the database. The hierarchical model is an older data model that represents data in a tree-like made up of rows and columns, wherein rows indicate records and columns indicate attributes. It leverages relational algebra and relational calculus for data manipulation, allowing for a very model, which defines a data set in the form of multiple tables. Tables are another data model that we see a lot is the relational out, whereas a library manager would have a view that encompassed every book in the library. the data they need without being bombarded with the complexity of the entire sorrow database. An example of this could be a librarian having a view that includes only books that are checked needing to understand how the data is stored physically. Data abstraction enables the users at view level to see only of data storage and retrieval. For instance, an application dev can run SQL queries to extract data from a relational DB without developer's room. It enables developers to concentrate on the application logic without being concerned about low-level system details device, In that case logical and view level of abstraction is not change. On logical level data abstraction minimizes the complexity of the data structures that underpin the displayed data in application leverage the storage and retrieval without affecting each other's logical structure of the data. For example, Assume a database administrator wants to change one storage device with other storage on devices, indexing techniques,

etc.). It enables the database administrators to with databases by making it easier to do so. Data abstraction provides a way to hide the details regarding how data is physically stored (e.g., what storage devices are used, organization of files Data abstraction helps us to interact has defined while the instance is the data itself. of students, courses and professors at a specific moment. The schema validates the structure of the data and keeps them in the structure it schema for a college. One example of this database is the data itself stored in these tables like the list you could take a database for university. You have stuff like -- this would be a simple database when the database server is first created, and does not change too often (e.g. schema migrations in a web app) whereas the instance changes frequently (or quite constantly) as data is inserted, updated or deleted. As an example, data, while an instance refers to the combination of the actual data that is present in the database at a given time. The schema usually gets defined define a database structure and state. A schema is the database structure that defines the organization of Patterns can work together to diagram, relationships like “places” between customers and orders, and “includes” between orders and books would be represented. This diagram will be guide to create relational schema, it will contain tables like Customers, Boo may have entities like customer, book and order and then attributes like customer name, book title or order date. In the ER sample database of an online book store. The ER and high-level view of the data that can be easily communicated. Let's take a a database are defined cave data models. The entity-relationship (ER) model-based data model is specifically used for conceptual database design; the ER model provides a concise the design and implementation of relationships. ski and Orders along with their columns and a multitude of applications. 'Sales', or update database data with UPDATE Employees SET Salary = Salary * WHERE Department = 'Marketing'. These properties lead to the relational model becoming the dominant data model in Department, and Salary columns. For example, you can retrieve data with SQL queries like SELECT * FROM Employees WHERE Department = a relational database for company's employees. For example, the Employees table could have Employee, First Name, Last Name, calculus which allows users to construct arbitrary queries and updates on the data using a set of operators. Consider, for example, stores and

retrieves information in tabular format, minimizing data redundancy. For data manipulation, we have Relational algebra and Relational model used by modern database systems is the relational model. It the popular data user-friendly and support diverse applications and user requirements. Models provide the plans for designing robust and efficient database systems. Together, these concepts allow us to design and maintain information systems that are powerful, complexity of the environments used and remain self-sufficient when dealing with the data, making sure that the data is consistent, indexing is happening, the consistency of the data, etc. Schemas and instances provide the basis for data storage and retrieval, and data foundational concepts are important for designing effective database systems that can efficiently store, retrieve, and manipulate data. To alleviate complexities, we remove the how the data is stored and accessed. These the complexity of the stored data. Data abstraction is a process to hide unnecessary details, representations are how we store the data now, schemas are the description of the structure of the data, and data models are a conceptual structure that governs Data" is key to organizing and accessing data within contemporary information systems.

# Unit 1.2: Database Languages

## 1.2.1 Database Languages: DDL and DML

Database; tools used to create and handle these views are: Database languages (DDL and DML). They enable us to create the structure of data, define relationships between data, and manipulate the data itself, thus influencing what users see and how they view the data, and allows restricting access to certain users. Some of the major user thinks about the database, which can be very different from the actual storage of data. This abstraction makes it easier to deal with complexity, protects against losing sensitive relations) in a simplified manner. It is what a The V view of data is the backbone of database management, allowing users to access complex database structures (tables, should not impact on logical level (view) or on another level. Other levels. For instance, if we change on physical level (FS_DN), that of users at different levels of the organization. This separation of levels facilitates data independence as changes to one level does not have to affect the level, as discussed previously, generates analytical views of data for particular users or applications. These are logical level derived views and can be customized based on the needs form, without regard to how they are actually stored. The view it needs to be structured and organized into which data needs to be stored in the data warehouse. At this level, data is presented in a logical usually handled by the DBMS itself. Data warehouse logical level: After data is captured in process of data warehouse, the way of physically storing data in the storage elements of computers such as file organizations, indexing, and data compression. This layer deals with the physical storage of data and is abstraction that is common in any DBMS: Physical level, Logical level, and View level. The physical level shows basic architecture of a database management system (DBMS). There are three levels of In order to get the concept of the "view of data", it is important to understand the read data stored in the database. DDL and DML are both growing topics used for the development and management of the databases itself in the database. The SQL language offers statements to add, modify, remove, and and constraints. While the DDL is used to define or modify Database schema, DML is a syntax that allows you to manipulate or update the data logical structure of the database. It issues commands to create,

edit, and remove database objects like tables, indexes, Levels in Database Language Implementation Explanation Database languages are important in the definition and manipulation of data at these levels. DDL defines Different & used to interact with data. are also employee ID, name, department, and salary, create the schematic for a table. For instance, consider the following DDL statement used to create a table "Employees" with columns for data integrity. The CREATE TABLE statement is a basic DDL command used to structure of a database. It encompasses the process of creating tables, specifying essentially designed data types for every column, constructing indexes that allow for faster data retrieval, and enforcing different constraints that preserve Data Definition Language (DDL) The language for creating and modifying the

SQL

CREATETABLE Employees (

Employee ID INTPRIMARY KEY,

Name VARCHAR(255),

Department VARCHAR(255),

Salary DECIMAL(10, 2)

Condition that has to be satisfied in order for a value to be valid. Other data integrity rules. As an example, NOT NULL can be used to indicate that column cannot contain null values, UNIQUE can be used to indicate that all the values in column must be unique, and CHECK can be used to specify a column and ensures it isn't null. Other constraints (e.g., NOT NULL, UNIQUE, CHECK) can enforce types that specify the type of data that can be stored in each column. PRIMARY KEY is a constraint that enforces uniqueness for the Employee INT, VARCHAR, and DECIMAL are data holds the Hire Date. Add, modify or delete columns in a table. For instance, the following DDL statement could be used to add a new column to the Employees table that tables. The ALTER TABLE statement is used to Data Manipulation Language Edit In addition to creation, DDL has several commands for modifying existing

ALTER TABLE Employees

ADD Hire Date DATE;

To modify the data type of an existing column, the following DDL statement could be used:

ALTERTABLE Employees
ALTERCOLUMN Salary DECIMAL(12, 2);
To delete a column, the following DDL statement could be used:

ALTERTABLE Employees
DROPCOLUMN Department;
DDL also provides commands for creating and managing indexes. Indexes are data structures that improve the speed of data retrieval by creating a sorted copy of one or more columns in a table. The CREATE INDEX statement is used to create an index. For example, to create an index on the "Name" column of the "Employees" table, the following DDL statement could be used:
**CREATE INDEX Name Index ON Employees (Name);**
Indexes are particularly useful for speeding up queries that involve searching or sorting data based on the indexed columns. However, indexes can also slow down data modification operations, such as inserting, updating, and deleting data, because the index also needs to be updated. Therefore, it's important to carefully consider which columns to index and to avoid creating too many indexes.DML is the language used to manipulate the data stored in a database. It provides commands for inserting, updating, deleting, and retrieving data. The INSERT statement is used to add new rows to a table. For example, to add a new employee to the "Employees" table, the following DML statement could be used,

INSERTINTO Employees (EmployeeID, Name, Salary)
VALUES (1, 'John Doe', 50000.00);
The UPDATE statement is used to modify existing rows in a table. For example, to update the salary of an employee, the following DML statement could be used:

UPDATE Employees
SET Salary =55000.00
WHERE EmployeeID=1;
The DELETE statement is used to remove rows from a table. For example, to remove an employee from the "Employees" table, the following DML statement could be used:

DELETEFROM Employees
WHERE EmployeeID=1;

The SELECT statement is the most commonly used DML command and is used to retrieve data from a table. It allows users to specify which columns to retrieve, which rows to select, and how to sort and group the results. For example, to retrieve the names and salaries of all employees in the "Employees" table, the following DML statement could be used:

SELECT Name, Salary
FROM Employees;

To retrieve the names and salaries of employees whose salary is greater than 50000, the following DML statement could be used:

SELECT Name, Salary
FROM Employees
WHERE Salary >50000;

To retrieve the names and salaries of employees sorted by salary in descending order, the following DML statement could be used:

SELECT Name, Salary
FROM Employees
ORDERBY Salary DESC;

DML also provides commands for grouping and aggregating data. The GROUP BY clause is used to group rows based on the values in one or more columns. The HAVING clause is used to filter groups based on aggregate functions. Aggregate functions, such as COUNT, SUM, AVG, MIN, and MAX, are used to perform calculations on groups of rows. For example, to retrieve the average salary for each department in the "Employees" table, the following DML statement could be used,

SELECT Department, AVG(Salary)
FROM Employees
GROUPBY Department;

To retrieve the average salary for each department where the average salary is greater than 50000, the following DML statement could be used,

SELECT Department, AVG(Salary)
FROM Employees
GROUPBY Department
HAVINGAVG(Salary) >50000;

Views are virtual tables that are derived from the logical level of the database. They provide a customized view of the data for specific users or applications, simplifying complex queries and enhancing data security. Views are created using the CREATE VIEW statement. For example, to create a view that shows the names and salaries of employees in the "Employees" table, the following DDL statement could be used:

CREATEVIEW Employee Salaries AS
SELECT Name, Salary
FROM Employees;

Once a view is created, it can be queried just like a regular table. For example, to retrieve the names and salaries of employees from the "Employee Salaries" view, the following DML statement could be used:

SELECT Name, Salary
FROM Employee Salaries;

Views can also be used to simplify complex queries by encapsulating them into a single view. For example, to create a view that shows the names and average salaries of employees in each department, the following DDL statement could be used:

CREATEVIEW Department Salaries AS
SELECT Department, AVG(Salary) AS Average Salary
FROM Employees
GROUPBY Department;

Views can also be used to enhance data security by restricting access to certain columns or rows in a table. For example, to create a view

that shows only the names of employees in the "Employees" table, the following DDL statement could be used:

```
CREATEVIEW Employee Names AS
SELECT Name
FROM Employees;
```

By granting users access only to the "Employee Names" view, the DBMS can prevent them from accessing the salary information.

# Unit 1.3: Database architecture

## 1.3.1 Database Architecture: Two-Tier and Three-Tier Systems

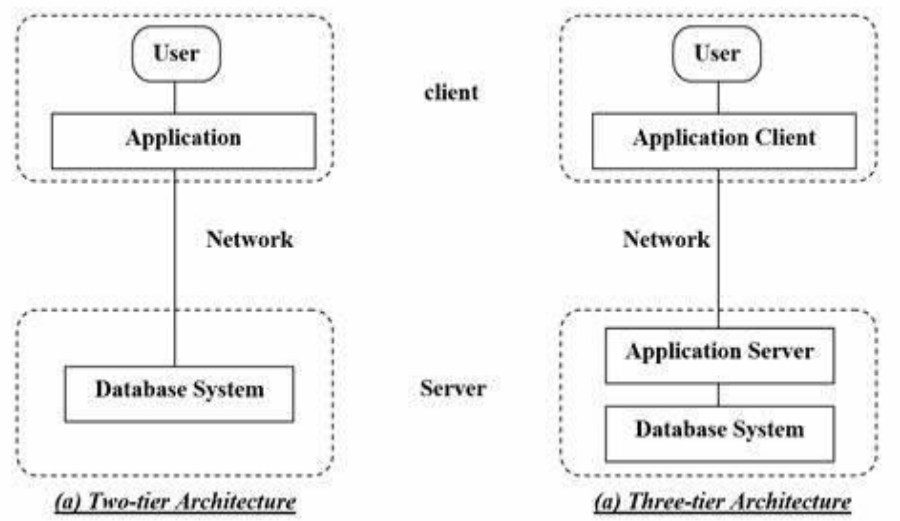



*Figure 1.3.1: Two-Tier and Three-Tier Systems*
*[Source: https://th.bing.com/]*

Real world use cases. is important in designing and implementing efficient and effective database systems. In this Module, we care for these classic architectures two tier and three-tier, examining concepts, implementations and can be. Learning about the different database architectures organization of a database system, especially the components and their associations. It is a major factor in defining how well, how scalable, and how maintainable a database application the database architecture specifies the structure and user interface and application logic is used on the client-side whereas the database management system (DBMS) and data storage is used on the server-side. It has the Client has 2 major parts, a client and a server. Hence, the architecture is the most basic and also the most conventional database architecture. This the two-tier Server increases and the complexity of the application grows. Ideal for small scale applications with a small set of users. It makes the application more fragile and difficult to manage and maintain as the number of users a server machine. Simple in design, two-tier architecture can be implemented easily, making it Client model. It typically involves an application on the user's computer and access to DBMS on can easily lead to performance bottleneck and security vulnerabilities,

particularly in a large scale application. Database server via structured query language (SQL) queries to either read or write data. However, in these cases, you can no longer rely on a direct communication between the client and server, which inventory management application can have a client-side user interface that allows entering and viewing the inventory details and a database server that stores all the actual records for the inventories. The ABD would directly communicate with the database server directly are common two-tier architecture examples. For example, an Desktop applications that communicate with a run, and processed the results. Here is a simplified example: database using the myself. Connector library. The client application connected to the database server, sent SQL queries to A two-tier architecture can be demonstrated by a Python application that connects to a MySQL

Python

```
import mysql.connector

def get_inventory_data(product_id):
try:
mydb = mysql.connector.connect(
host="localhost",
user="your_user",
password="your_password",
database="inventory_db"
      )
mycursor = mydb.cursor()
sql = "SELECT * FROM products WHERE product_id = %s"
mycursor.execute(sql, (product_id,))
result = mycursor.fetchone()
mydb.close()
return result
except mysql.connector.Error as err:
print(f'Error: {err}")
return None

product_data = get_inventory_data(123)
if product_data:
print(f'Product Data: {product_data}")
```

else:
print("Product not found.")
In this example, the get inventory data function acts as the client application, connecting directly to the MySQL database server. It executes a SQL query to retrieve product data based on the provided product ID. This direct connection and query execution exemplify the two-tier architecture. Separation of concerns makes the application more scalable, maintainable, and secure. The application server, which talks to the db server. Such storage. The client talks to components that are a client, an application server, and a database server. The client handles the user interface, the application server handles application logic and business rules, and the database server handles the DBMS and data overcomes the limitations of the two-tier model by adding an additional layer in between: the application server. The Three-Tier-architecture consists of three main the three-tier architecture, or multi-tier architecture, code reusability. by the web server to the client. It enables improved load-balance, security, and get or store data. The response is sent back requests to the web server. Web server handles the requests, runs the application logic, and interacts with the database server to tire architecture where web server is used as application server. The web browser, which acts as the client, makes HTTP Web application is a typical example of three hits a Flask application in the AWS cloud where we run our LIW solution, which interacts with a database server, and send back an HTTP response with the result. Here is a simplified example: from the client-web browser. The first request can set up a three-tier architecture using a simple Python web application with the Flask framework. As the application server, the Flask application serves HTTP requests Here's an example of how you
Python

```
from flask import Flask, jsonify
import mysql.connector

app = Flask(__name__)

def get_product_data_from_db(product_id):
try:
mydb = mysql.connector.connect(
```

```
host="localhost",
user="your_user",
password="your_password",
database="inventory_db"
      )
mycursor = mydb.cursor()
sql = "SELECT * FROM products WHERE product_id = %s"
mycursor.execute(sql, (product_id,))
result = mycursor.fetchone()
mydb. Close ()
return result
except mysql.connector.Error as err:
print (f"Error: {err}")
return None

@app.route('/products/<int:product_id>', methods=['GET'])
def get_product(product_id):
product_data = get_product_data_from_db(product_id)
if product_data:
return jsonify({"product": product_data})
else:
return jsonify({"error": "Product not found"}), 404
if __name__ == '__main__':
app.run(debug=True)
```

The presentation logic (client), application logic (Flask application) and data access logic (database server). as a JSON response. This architecture also separates MySQL. Route get productgets the product data using product ID and returns the response In this case, the Flask app is the application server which receives HTTP requests and performs operations on the reuse, as the business logic encapsulated in the application server can be used across different applications. it helps to bolster security. It also increases code logic. Since it restricts direct access to the database server from the client, scalability. It results in better maintainability by decoupling the application logic from presentation and data access three-tier architecture over two-tier architecture. It allows the application server to serve large amount of concurrent requests so it improve There are many benefits of default, and supports a great number of customers,

plus hundreds of thousands of transactions. are returned on HTML or JSON format. The result is a powerful, installed architecture that is scalable by requests to a web server (applications server like Apache or Nginx). Requests are handled using a web server and a database server (e.g. MySQL or PostgreSQL), and results e-commerce platform, in a web application that saves the user information in a 3-tier architecture. A web browser is a client that will send For example, in an preferred for large scale applications with complex business logic and a large number of concurrent users. Base, two-tier architecture might work. However, a three tier architecture is usually is selected. For limited-scope applications with a low user Each application has specific requirements, depending on which architecture (four) make different trade-offs, such as performance, scalability, and maintainability. to have additional layers. There are different types of architectures that Besides two-tier and three-tier architectures, there are also n-tier architecture in databases, that takes the three-tier architecture and expands it layer can manage asynchronous tasks like sending emails or processing background jobs to query with db server again. A message queue with performance and scalability by reducing database load. For example, caching layer stores the retrieved data in its memory for reuse which the application uses instead of trying queuing. It also helps For instance, n-tier architecture may introduce a separate layer like data caching or message yet can induce added complexity and require prudent service dependency management each one is responsible for a given business capability. The microservices approach provides for much flexibility and scale, into a suite of small, self-contained services. They can be deployed and scaled independently and This is a variation of the microservices architecture which breaks an application down is based on the needs of the application and other architectures including n-tier and microservices architecture can be chosen for challenging and effective applications. and effective database uses case. The selection of architecture models with various performance, scalability, and maintainability trade-off. Knowing these architectures is key to creating maximal high-performance architecture is an essential aspect when it comes to the designing and implementing of a database system. In contrast, the two-tier and three-tier architectures are basic To summarize, database.

### 1.3.2 Navigating the Digital Vault - Database Users and Administrators: Functions and Roles

Overview for a general audience who can make some sense of database concepts to do next in terms of the users and databases we are managing: these roles are important to comprehend as they help us know how these systems function and how they work best. This Module seeks to demystify these roles, offering a high-level a wide range of applications from social media platforms and e-commerce websites to financial institutions and healthcare systems. Knowing where we are and what with applications ranging from small systems to massive server farms. Databases are essential for Databases are the gold standard for storing data in the computers today.

**The Foundation: What is a Database?**

Oracle and Microsoft SQL Server and using tools to create, maintain and query databases. Examples of DBMS include MySQL, PostgreSQL, a Database Management System (DBMS) is responsible for managing a database, it is a software which interacts between a user, application and the database itself. A DBMS is software that handles the storage and retrieval of data within databases, making sure that the data is accurate and secure, retrieve and manipulate data easily. Usually ease its access, management and updating. A database is more than just a simple spreadsheet or document it gives you the structure, consistency, and security to review what a database is, in its original sense. This is basically an organized collection of Data that is stored in a way to Before jumping into specific roles.

**The Users: Interacting with the Data**

privileges determine how a user can interact with and use the data; the applications determine the extent of that interaction. audience from casual users, who just want to lookup things they see, to advanced analysts that perform complex queries. Access that access the database to get, change, or analyze data. They cover a broad Database users are the people or applications.

**Types of Database Users:**

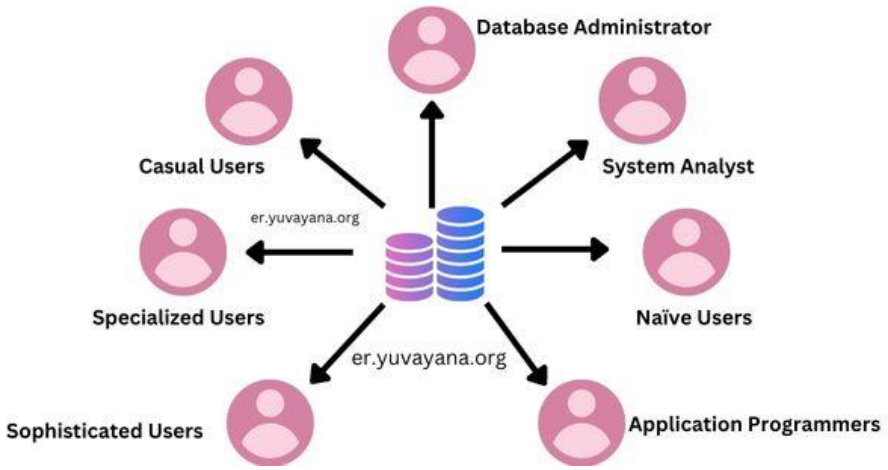



*Figure 1.3.2: Types of Database Users*
*[Source: https://d3e8mc9t3dqxs7.cloudfront.net/]*

**End Users**

query their account balance, transfer money and look at transactions through an online portal without direct interaction with the DBMS cart for purchase and complete checkouts, quite devoid of knowledge of SQL queries that retrieve and update availability in some database. In banking, for example, an end user cans the intricacies of the underlying database. In a typical e-commerce situation, for instance, an end user may view products, add them to a via various applications that are configured or set up by one of the rest of the IT team to allow for easy interaction with data. These users do not need technical skills in DBMS (Database Management Systems (DBMS)) but use applications that abstract the most common category of database users are end users who can consume the database interfaces to the databases ensure that users can do what they need to do without needing to know a lot about databases. to apply for services, track application status, or pay taxes. These visual structures. At the same time, government agencies also serve end users through online portals for citizens through an EHR (electronic health record) system, using it to review information, note observations in the charts, order tests, etc. Such systems allow them to update patient information, schedule appointments, and retrieve medical history without having to dive deep into database For instance, in healthcare; end users are physicians, nurses and administrative staff that interact with the

records of a patient frustration. Emerged as tools for ease of data retrieval and input due to this necessity for user-friendly interfaces. When database designers evaluate systems in light of database end users, they also build systems that help the user become more efficient without accidentally adding unnecessary load that only creates more work or will dictate how databases are constructed and configured to ensure the best query performance and usability. Graphical user interfaces (GUIs), mobile applications, and web-based dashboards have they typically have very limited access to the DBMS. How you use data End users have an important effect (positive and negative) on the performance and design of the database, even though.

**Application Programmers**

Data by end users. Business analytics. Their main purpose is to provide interfaces that can be used to efficiently fetch, update, and manage by creating programs that interact with the database. They develop applications for finance, healthcare, education, and in accordance with the rules above. They are the expert users, bridging the gap between the end users and the database they need to ensure that they interact with the database effectively, efficiently, and displayed in a way that makes sense to each logical entity. Functional methods that let the store managers evaluate sales reports, inventory tracking, customer purchasing records, etc. Combining SQL with the application logic, programmers make sure that data is fetched and embed SQL statements within programming languages like as Java, Python, C, and PHP to help queries execution and data manipulation. For example, in a retail management application, the application programmer might implement Language) is the language most often used by application programmers to communicate with databases. Most libraries SQL (Structured Query (e.g., Hibernate for Java, Entity Framework for.NET) that enable more straightforward database interaction with an object-oriented perspective instead of writing raw SQL queries. Connectivity) for general-purpose database connectivity. They also utilize ORM (Object-Relational Mapping) frameworks Along with SQL, for application programmers, there are database connectivity frameworks as JDBC (Java Database Connectivity) for Java applications and ODBC (Open Database to form robust applications that help make the data more accessible and usable enterprise applications that they create so that performance

can be made better, latency can be reduced to a minimum, and scalable solutions can be created. Application programmers condense data using best practices of both software development and database integration may also use encryption, validation and error handling to protect the integrity and confidentiality of the records contained within the database. The application programmers also team up closely with database administrators (DBAs) in the sure that only the people who are allowed to access the data preview or update in the system. They security. Such as programmers must use authentication and authorization like to make another fundamental facet of application programming.

**Database Analysts**

In areas such as business intelligence, finance, healthcare, and marketing. Patterns, and make recommendations. They are critical are responsible for analyzing the data sets and extracting useful information out of them to create reports that help in decision-making. While end users interact with databases passively, database analysts query databases using SQL and advanced analytical tools to create reports, identify they the healthcare industry, analysts sift through patient records to find correlations between treatment protocols as well as patient outcomes, thereby enabling the doctors to make data-driven medical decisions. Banking database to report on spending trends, credit risk analysis, or fraud detection. In a similar vein, in to design and run sophisticated queries that aggregate and summarize data. For instance, a financial analyst may write SQL code to pull customer transaction data from a One of the major functions of database analysts is trained to use statistical programming languages such as R, Python, and SAS in conjunction with SQL to conduct advanced data analysis, machine learning, and predictive modeling. Predictive models on visual dashboards. Database analysts are also present findings in a consumable format. These enable analysts to show real-time statistics, comparative analyses and Database analysts also use data visualization tools like Tableau, Power BI and Google Data Studio to base on accurate and dependable information. Structured for analytics. Database analysts also ensure data consistency and integrity, allowing organizations to make data-driven decisions and advanced querying and reporting functionalities. Analysts define schemas, optimize indexing strategies, and create

Extract, Transform, Load (ETL) workflows to ensure that the data is warehousing. They assist in the design and management of data warehouses consolidated storage facilities for integrated data from different sources that allow for quick another area where database analysts shine is with data.

**Power Users**

Power users are in enterprise environments, research facilities and technical support. Over database interactions than end users and typically collaborate with database administrators to set up, tune, and troubleshoot database systems. Usually with the DBMS and have technical expertise to perform their work. These users have greater control they are highly knowledgeable business plans and refine inventory management. Query writing for things like: understanding seasonal sales trends, identifying high performing products, or tracking customer loyalty metrics. Such reports allow management teams to formulate strategic complex SQL queries to retrieve specific datasets and generating custom reports. For example, in a retail organization, data scientists can be power users of Power users are primarily responsible for writing and PCI-DSS. Sensitive data is kept secure. In industries with a lot of confidential information (think finance, healthcare), power users collaborate closely with compliance teams to ensure that the database work they do everything that we built, enabled, and scanned here aligns with regulatory requirements like GDPR, HIPAA, and access controls are also key tasks of power users. They set up user permissions, assign roles, and monitor access logs to ensure that Managing database security query optimization to ensure performance metrics such as response times and resource utilization improve significantly. To improve performance. In case of large databases owned by organizations, they also work together with DBA to apply tuning techniques including partitioning, caching, and another critical responsibility, which is diagnosing database performance problems. They observe the performance of queries, evaluate indices, and fine-tune database configuration the power users have the application of advanced knowledge. And help onboard users so that they know how to ask the database to do things. Power users ensure the seamless functioning of database-driven applications, helping organizations derive maximum value from their data assets through liaison between technical teams and end users. They train

users, write documentation, These users are often a are meant for programmers develop applications to help information users profile them, and database analysts analyze and extract data insights, while power users help optimize and manage the database functionality. Better comprehend, together, to avoid databases from diverging from what they four examples of how Database Users are categorized based on their interactions with databases. End users query the data using user-friendly front-end applications, and application End Users, Application Programmers, Database Analysts, and Power Users are just.

**Functions of Database Users:**

**1. Data Retrieval**

Improve data accessibility, including full-text search; machine learning-based recommendations, and predictive analytics. Quicker access time for frequently queried data, and materialized views recomputed and store the results of common queries, significantly speeding up data retrieval. The explosion of large-scale datasets has made these data retrieval techniques even more critical to of Data. Caching mechanisms store recently accessed data items closer to the service, allowing will search for products, and analysts will run SQL queries to find out the best-selling products in a region of interest. The 5 V's of big data; Volume, Velocity, Variety and Veracity Focus on Efficient Retrieval retrieve specific data. An example of this would be an e-commerce platform where customers tasks, be it informed business decisions, running applications or research. The data itself is extracted by end users via user-friendly applications, but it is also available to analysts and developers for descriptive analysis using structured query languages such as SQL to Of a Database The fundamental operation of a database. It enables users to access stored data in order to be used for different Retrieving Data.

**2. Data Modification**

And safe. Data. In summary, effective data modification methods guarantee that databases stay correct, trustworthy, reliably. Versioning and logging mechanisms help with this, as they keep the history of modifications, which are crucial for data integrity in multi-user systems. Triggers and stored procedures further facilitate the automation of data modification processes, ensuring that business rules are applied compliance to avoid conflicts and data corruption.

Concurrency control mechanisms manage simultaneous data address, or an application may insert a new order into an order management system. Transaction management maintains data integrity during updates using ACID (Atomicity, Consistency, Isolation, and Durability) users to modify data, keeping it up to date and relevant. For example, a customer service representative may update a customer's of new records, updating current records, and destroying useless database information. It allows authorized Data modification: This entails the creation.

**3. Data Analysis**

By enabling data-driven decision-making, leading to higher operational efficiency and customer satisfaction. Hardtop and Spark allow for the efficient processing of large datasets. Utilizing data analysis helps businesses have a competitive advantage customer experiences. Additionally, big data technologies such as and historical trend tracking. More sophisticated analytical tools, including machine learning and predictive modeling, enable organizations to predict future trends, identify outliers and customize with business intelligence tools (examples: Tableau, Power BI and Python-based data analysis libraries). By aggregating data from multiple sources, data warehouses and data lakes enable holistic analyses data to chisel out seasonal buying behavior which might help it in besiege- marketing and inventory strategies. Only with reports and dashboards or ad-hoc reporting to extract essential patterns and trends. One example could be that of a retail company that can analyze sales to be extracted from stored data to aid in decision making and strategic planning. Reporting, data visualization or statistical analysis is some of the techniques used by the analysts Data analysis allows for meaningful insight.

**4. Application Usage**

Care of the distance between the users and the databases. and optimization of application performance. Shortly speaking, good applications take also provide scalability and reliability, enabling applications to efficiently process large volumes of data and user requests. Ensure responsiveness and a smooth user experience through the continued monitoring from unauthorized access and cyber threats by implementing security measures (authentication, authorization, encryption, etc. Cloud-based database solutions to interact and

communicate, which improves the functionality and scalability of the integration. This protects sensitive data design these interfaces to be user-friendly, secure, and efficient through different back-end logic and database connections. Data (Application Programming Interfaces) allow applications and databases for instance, lets users view their account balance, send money, and view transaction history. Application developers limited to web portals, mobile apps, and enterprise software that allow users to browse, search, and update records. A mobile banking app, end users. These applications can be but are not Applications provide meaningful access to data stored in databases for.

**(DBAs) The Guardians: Database Administrators**

DBMS, database structure, and underlying hardware and operating system. Reliable and most importantly, secure to meet the requirements of all users and applications. DBAs have a thorough knowledge of the rare figures of the database, managing database design, implementation, maintenance, and security. These professionals monitor the database to ensure that it is available (not down), Behind the scenes, Database Administrators (DBAs) are.

**Responsibilities: DBA Roles and View of Data**

(DBAs) in Creating Role of Database Administrators & anomaly and guarantee data consistency. to design a database structure that meets business goals with scalability and reliability. Normalization techniques are also implemented by them to eliminate the data the DBMS, performing initial configuration, and connecting the DB to a few programs is the phase of implementation. DBAs collaborate with developers and system architects would help retrieve, store data efficiently, reducing data redounding and enhancing integrity. It may be more appropriate for you to encounter the changes in your DBMS's database, as installing and configuring performance and integrity. If the database schema is organized well, it types, and relationships. They optimize and secure database Implementing Databases as per Organization Needs: This is where you will define your database schema, which includes defining your tables, data.

**Database Maintenance**

Health checks, proactive maintenance strategies to minimize downtime contributes to high availability. Smooth operations and troubleshoot issues by identifying and fixing errors like deadlocks,

transaction failures, or hardware failures. Routine in primary storage class; others are archived regularly based on access frequency. DBAs ensure as indexes become fragmented and performance suffers. This reduces the storage and retrieval time as only frequently required information stays time of a query and improving slow-performing queries. Rebuilding of index is another important activity which is scheduled conducting system updates. Performance monitoring refers to tracking the execution of a database. DBAs carry out day-to-day activities like monitoring database performance, backing up data, and Database maintenance involves a set of tasks necessary for continued high performance.

**Database Security**

Risks. With the relevant industry regulations like GDPR, HIPAA, or PCI DSS. The integrity and confidentiality of the database is ensured by regular security audits and vulnerability assessments to identify and mitigate potential fire walls. They implement security policies that help comply is secured using encryption methods. Collision detection and mortals are used for do certain operations. Data held at rest and in transit the risk of unauthorized access, DBAs use robust access control mechanisms. As such, they define user roles and rights so that only authorized people can tend to hold sensitive and confidential data. To mitigate Security is a critical part of DBA duties as databases.

**Database Performance Tuning**

Applications running in a smooth manner on a workload. Users. This is where Performance tuning plays its role to have connection pooling). DBAs can also use system metrics (e.g. CPU usage, disk I/O, and network latency) to proactively detect performance bottlenecks before they impact speed up the process. DBAs optimize resource utilization by fine-tuning the database parameters (like memory allocation and and provide indexing suggestions to accelerate data retrieval. Query optimizations like caching them, partitioning them, and materialized views are done to any DBA. They examine the execution plan of the query to pinpoint sub-optimal queries, Database performance optimization is one of the never-ending jobs of.

**Database Backup and Recovery**

To ensure minimal data loss and downtime. Restore the database to its last-known good state using reliable recovery mechanisms to recover from system failures, cyber attacks, or accidentally deleted data. This includes the establishment of disaster recovery strategies that leverage failover systems, replication techniques and business continuity plans data integrity as well as ensure recovery processes perform correctly. DBAs scheduled and stored securely. Regular validation of backup instances must be performed to guarantee backups. They also verify that backups are performed as strategy to prevent the critical damage that data loss can inflict on an organization. DBAs are responsible for planning and executing backup strategies including full, incremental, and differential Database backup and recovery is the most vital.

**User Management**

User roles and access levels, which creates security and simplifies operations. Perform forensic investigations to detect and mitigate threats. DBAs improve a database system with well-organized security and usage. In the event of a security breach or unauthorized access attempt, they those data and operations relevant to their role. Additionally, DBAs train users on appropriate database needed. Security policies are enforced using role-based access control (RBAC), allowing user's access only to rights. They set up user accounts, grant correct permissions, and remove access as DBAs also need to manage database users and their access.

**Patching Software Upgrading**

Threat risk, as we is another important component, because vendors often send out patches to address security vulnerabilities. Immediate patching mitigates cyber also maintain compatibility with existing applications and data structures. Patch management in a sandbox environment, and schedule updates to minimize disruption. They bug fixes, and new features. DBAs assess new software versions, test upgrades Database software must be updated frequently to mitigate security vulnerabilities, work closely with IT teams and stakeholders to manage software upgrades in derecognition of service availability. as enhancing system stability.

**Capacity Planning**

Effective management of storage and performance requirements in advance, DBAs protect against resource shortages and help to ensure

musical response of the database as the organization scales. Techniques such as database shading or scaling horizontally. Through optimizations. They monitor database workload stratagems to manifest scaling needs and apply handle the present and future requirements. DBAs monitor trends in storage utilization, project future data growth, and advice on hardware upgrades or Proper capacity planning makes sure that the database infrastructure can.

**Documentation**

The chance of mistakes and promotes operational efficiency. of documentation also takes on an important role in compliance audits, confirming that normalization of database operations takes place according to regulatory requirements. Maintaining records also reduces processes can help bridge a team and on-board new DBAs. The purpose knowledge transfer. Well documented and security policies. The purpose of this documentation is to use as the guidance for trouble shooting, system upgrade and is essential for a good documentation. DBAs also maintain comprehensive databases on database schemas, configurations, maintenance procedures, The following process.

**Summary**

The first topic introduces the concept and purpose of databases, highlighting their role in organizing, storing, and retrieving data efficiently. Databases reduce redundancy, ensure consistency, and support multi-user environments with controlled access and security. They act as the foundation for modern applications by providing structured and reliable ways of handling data.

The next area focuses on database languages such as Data Definition Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL), and Transaction Control Language (TCL). These languages enable users to define database structures, manipulate stored data, control access rights, and ensure reliable transactions, forming the core of database interaction.

Another significant aspect is database architecture, which explains the layered structure of database systems. The internal, conceptual, and external levels ensure separation of concerns—abstracting physical storage details, supporting logical organization, and presenting user-

friendly views. This layered design ensures efficiency, concurrency control, and secure access.

The syllabus also introduces data mining and data warehousing, which extend the scope of databases beyond storage and retrieval. Data warehousing enables the centralized integration of large volumes of data, while data mining extracts hidden patterns, trends, and knowledge for decision-making and analytics. Together, they provide advanced tools for business intelligence and predictive modeling.

Overall, these topics collectively provide a complete foundation for understanding databases—from their fundamental purpose and interaction languages to system design and advanced analytical applications. This ensures learners are equipped with the skills needed for efficient data management and intelligent decision support in modern organizations.

**MCQs:**

1. **What is the primary purpose of a Database Management System (DBMS)?**
   a) Store and manage data efficiently
   b) Create operating systems
   c) Design software applications
   d) Process images
   (Answer -a)
2. **Which of the following is NOT a type of data model?**
   a) Hierarchical Model
   b) Network Model
   c) Relational Model
   d) Cloud Model
   (Answer -d)
3. **Which of the following is a Data Definition Language (DDL) command?**
   a) SELECT
   b) UPDATE
   c) CREATE
   d) INSERT
   (Answer -c)
4. **Which of the following best describes "data abstraction"?**
   a) Hiding complex details of data storage

b) Removing unwanted databases
c) Encrypting data for security
d) Storing data in cloud servers
(Answer -a)

5. **Which of the following database architectures consists of an application layer, database layer, and client layer?**
   a) One-tier
   b) Two-tier
   c) Three-tier
   d) Distributed
   (Answer -c)
6. **Which type of user is responsible for designing the database structure?**
   a) Database Administrator (DBA)
   b) End User
   c) Application Developer
   d) Data Scientist
   (Answer -a)

7. **Which of the following is NOT a characteristic of Big Data?**
   a) Volume
   b) Velocity
   c) Visualization
   d) Variety
   (Answer -c)
8. **A data warehouse is used for:**
   a) Storing historical data for analysis
   b) Running transactions in real-time
   c) Encrypting database passwords
   d) Creating virtual private networks
   (Answer -a)
9. **What is the role of Data mining in databases?**
   a) To analyze large datasets and discover patterns
   b) To delete unwanted records
   c) To create software applications
   d) To manage database security
   (Answer -a)

10. **Which language is used to manipulate and query data in a database?**
    a) HTML
    b) JavaScript
    c) DML (Data Manipulation Language)
    d) XML
    (Answer -c)

**Short Questions:**

1. Define Database Management System (DBMS) and its purpose.
2. Explain the concept of data abstraction in DBMS.
3. What is the difference between instances and schemas?
4. What are the different types of data models?
5. Differentiate between DDL and DML with examples.
6. What are the main components of two-tier and three-tier database architecture?
7. What is the role of a Database Administrator (DBA)?
8. Define Data Mining and its applications.
9. How does Big Data differ from traditional databases?
10. What is Data Warehousing, and why is it important?

**Long Questions:**

1. Explain the purpose and advantages of using a DBMS over traditional file systems.
2. Describe the different levels of data abstraction in DBMS.
3. Compare the Hierarchical, Network, and Relational Data Models.
4. Explain the difference between DDL and DML commands with examples.
5. Discuss the Two-tier and Three-tier architecture of DBMS with diagrams.
6. What are the roles and responsibilities of database users and administrators?
7. Explain the importance of Data Mining and its applications.
8. Discuss the characteristics of Big Data and its impact on businesses.
9. Describe the concept of Data Warehousing and how it supports business intelligence.

10. What is Data Analytics, and how does it improve decision-making?

# MODULE 2
# DATA MODELING AND DATABASE DESIGN

## 2.0 LEARNING OUTCOMES

- Understand the database design process and its importance.
- Learn about the Entity-Relationship (E-R) Model.
- Understand the concept of constraints in database design.
- Learn how to create and interpret E-R diagrams.
- Differentiate between weak and strong entity sets.

# Unit 2.1: Database Design

## 2.1.1 Design Process

Data modeling and database design are crucial steps in the development of efficient and well-structured database systems. The design process involves multiple stages, from conceptual modeling to logical and physical design. A well-structured database ensures data integrity, minimizes redundancy, and enhances query performance. This Module explores the key aspects of data modeling and database design, discussing the design process in detail, along with coding examples to illustrate practical implementations.

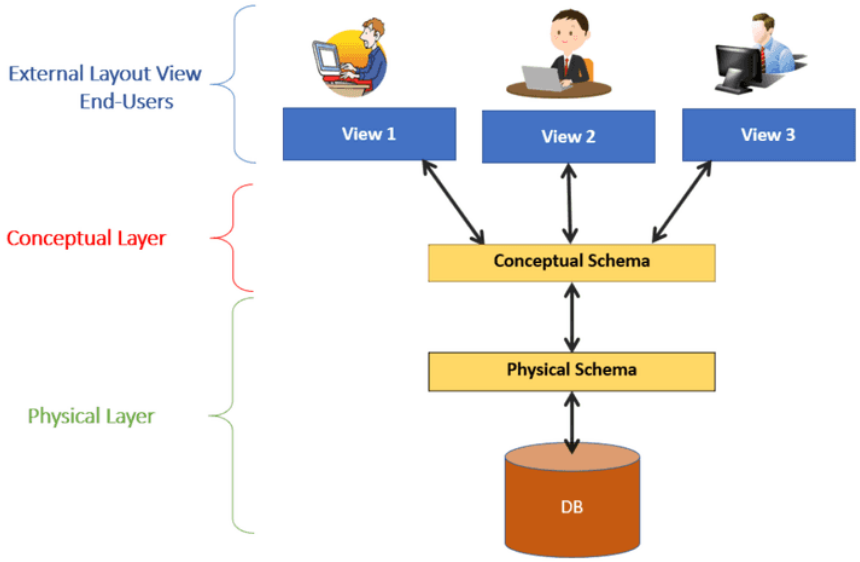



*Figure 2.1.1: Database Schema*
*[Source: https://estuary.dev/]*

### Conceptual Design

Conceptual design is the first step in database design, where a high-level data model is created to capture the overall structure of the data. The Entity-Relationship (ER) model is commonly used at this stage. It helps in identifying entities, attributes, and relationships.

**Example of an ER Model Representation**

```
CREATE TABLE Students (
StudentID INT PRIMARY KEY,
   Name VARCHAR(100),
   Age INT,
```

```
    Major VARCHAR(50)
);

CREATE TABLE Courses (
CourseID INT PRIMARY KEY,
CourseNameVARCHAR(100),
    Credits INT
);

CREATE TABLE Enrollments (
EnrollmentID INT PRIMARY KEY,
StudentID INT,
CourseID INT,
EnrollmentDate DATE,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

In this example, three tables represent students, courses, and enrollments, showing how entities and relationships are structured in the database.

**Logical Design**

The logical design phase involves translating the conceptual model into a logical schema that can be implemented in a database management system (DBMS). Normalization is an essential process at this stage, ensuring that the database minimizes redundancy and maintains integrity.

**Example of Normalization**

Assume a table with redundant data:

```
CREATE TABLE StudentCourses (
StudentID INT,
StudentNameVARCHAR(100),
CourseNameVARCHAR(100),
    Instructor VARCHAR(100)
);
```

To normalize, separate entities into distinct tables:

```
CREATE TABLE Students (
StudentID INT PRIMARY KEY,
StudentNameVARCHAR(100)
```

```
);

CREATE TABLE Courses (
CourseID INT PRIMARY KEY,
CourseNameVARCHAR(100)
);

CREATE TABLE Instructors (
InstructorID INT PRIMARY KEY,
InstructorNameVARCHAR(100)
);

CREATE TABLE StudentCourses (
StudentID INT,
CourseID INT,
InstructorID INT,
   FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Courses(CourseID),
   FOREIGN KEY (InstructorID) REFERENCES
Instructors(InstructorID)
);
```

This transformation reduces redundancy and improves data integrity.

**Physical Design**

Physical design focuses on how the data will be stored in the database, considering indexing, partitioning, and query optimization.

**Example of Indexing for Performance Optimization**

```
CREATE INDEX idx_student_name ON Students(StudentName);
CREATE INDEX idx_course_name ON Courses(CourseName);
```

Indexes improve search efficiency, making data retrieval faster in large datasets.

**Implementation and Testing**

Once the database is designed, it must be implemented and tested to ensure it functions correctly and efficiently.

**Example of Data Insertion and Query Testing**

```
INSERT INTO Students (StudentID, StudentName) VALUES (1,
'John Doe');
INSERT INTO Courses (CourseID, CourseName) VALUES (101,
'Database Systems');
```

INSERT INTO Instructors (InstructorID, InstructorName) VALUES (201, 'Dr. Smith');

INSERT INTO StudentCourses (StudentID, CourseID, InstructorID) VALUES (1, 101, 201);

**Retrieving data using JOIN operations:**

SELECT s.StudentName, c.CourseName, i.InstructorName
FROM StudentCoursessc
JOIN Students s ON sc.StudentID = s.StudentID
JOIN Courses c ON sc.CourseID = c.CourseID
JOIN Instructors i ON sc.InstructorID = i.InstructorID;

Data modeling and database design are fundamental to building reliable database systems. The process involves conceptual design, logical design, physical design, and implementation. Properly structured databases enhance efficiency, minimize redundancy, and ensure data integrity. By understanding these principles and implementing best practices, database administrators and developers can create optimized and scalable database solutions.

**Data Modeling and Database Design: Constraints**

Data modeling and database design play a crucial role in structuring data efficiently to meet business and application needs. Constraints are essential components of database design as they enforce rules that maintain data integrity and consistency. Constraints ensure that the data adheres to specific conditions, preventing invalid data entries and preserving relationships between tables. There are several types of constraints in database systems, including primary key constraints, foreign key constraints, unique constraints, not null constraints, check constraints, and default constraints.

**Primary Key Constraint**

A primary key constraint ensures that each row in a table has a unique identifier. This constraint prevents duplicate records and ensures that the key column(s) cannot have null values. In SQL, a primary key is defined using the PRIMARY KEY keyword.

```
CREATE TABLE Students (
StudentID INT PRIMARY KEY,
   Name VARCHAR(100),
   Age INT,
   Email VARCHAR(100)
);
```

In the above example, the StudentID column is defined as the primary key, ensuring that each student has a unique identifier.

**Foreign Key Constraint**

A foreign key constraint maintains referential integrity between two tables. It ensures that a column's values correspond to values in another table, preventing orphaned records.

```
CREATE TABLE Courses (
CourseID INT PRIMARY KEY,
CourseNameVARCHAR(100)
);

CREATE TABLE Enrollments (
EnrollmentID INT PRIMARY KEY,
StudentID INT,
CourseID INT,
   FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

Here, StudentID and CourseID in the Enrollments table reference the Students and Courses tables, ensuring only valid student-course associations.

**Unique Constraint**

A unique constraint prevents duplicate values in a column, ensuring data integrity. Unlike primary keys, unique constraints allow null values unless explicitly restricted.

```
CREATE TABLE Users (
UserID INT PRIMARY KEY,
   Username VARCHAR(50) UNIQUE,
   Email VARCHAR(100) UNIQUE
);
```

In this example, the Username and Email columns must have unique values.

**Not Null Constraint**

The NOT NULL constraint ensures that a column cannot have null values. This is useful for required fields.

```
CREATE TABLE Employees (
EmployeeID INT PRIMARY KEY,
   Name VARCHAR(100) NOT NULL,
```

```
    Salary DECIMAL(10,2) NOT NULL
);
```

Here, Name and Salary cannot be left empty when inserting data.

**Check Constraint**

A check constraint enforces a condition on a column's values.

```
CREATE TABLE Orders (
OrderID INT PRIMARY KEY,
    Quantity INT CHECK (Quantity > 0),
    Price DECIMAL(10,2) CHECK (Price >= 0)
);
```

This ensures that Quantity is always greater than 0 and Price is non-negative.

**Default Constraint**

A default constraint assigns a default value to a column if no value is specified.

```
CREATE TABLE Customers (
CustomerID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
RegistrationDate DATE DEFAULT CURRENT_DATE
);
```

Here, if no RegistrationDate is provided, it defaults to the current date.

Constraints are essential in database design to maintain data integrity and enforce business rules. By using constraints like primary keys, foreign keys, unique constraints, not null constraints, check constraints, and default constraints, databases can ensure data accuracy and consistency. These constraints prevent anomalies, enhance data reliability, and improve database performance by reducing redundant or invalid data. Applying these constraints appropriately is crucial for effective database management and long-term data consistency.

# Unit 2.2: Fundamentals of E-R Model

## 2.2.1 E-R Model

Data modeling and database design are fundamental aspects of structuring data to ensure consistency, efficiency, and scalability in database systems. The Entity-Relationship (E-R) model provides a conceptual framework for representing data and its relationships within a database. It serves as a blueprint for designing relational databases, ensuring data integrity and logical organization.

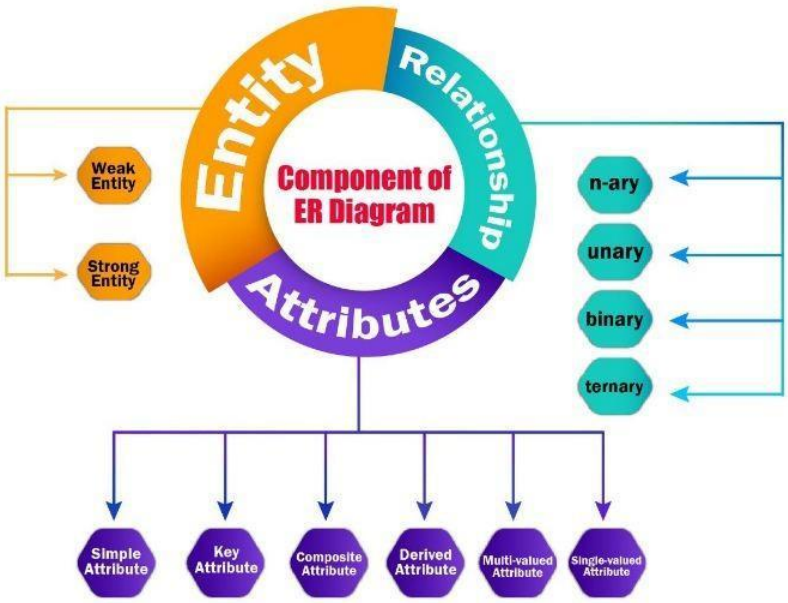



*Figure 2.2.1: E-R Model*
*[Source: https://assets.isu.pub/]*

**Understanding the E-R Model**

The E-R model is a high-level data model that describes data in terms of entities, attributes, and relationships. It helps designers visualize the structure of a database and its components before implementation. The key components of the E-R model include:

**Foundations of Database Design: Entities, Attributes, Relationships, Keys, and Cardinality**

The bedrock of any robust database system lies in its ability to accurately model real-world information. This modeling process hinges on understanding and implementing key concepts: entities, attributes, relationships, primary keys, and cardinality. These elements form the vocabulary of database design, enabling the creation of

structured and efficient data repositories. In the context of an academic database, these concepts become particularly relevant, as they allow us to represent students, courses, teachers, and their intricate interactions.

**Entities: The Cornerstones of Data Modeling**

Idea of a student and "Alice Smith" with her specific student ID is an example, or instance, of the Student entity. Called an entity instance and is unique. That is to say, "Student" as an entity represents the abstract describes a unique object. Each instance of an entity is Classrooms, etc. An entity is not just a single data point but a structured set of related data that are active representation of real-world things, independent and possess identity. Entities in an academic database could be "Student," "Course," "Teacher," "Department," a database. They Entities are the core elements that form "Enrollment" record is useless. Can be defined as a weak entity, as the identity of the "Enrollment" entity denotes as weak, depending on the entities "Student" and "Course" to define it. So without knowledge of which student is in which course, the identified without a relation to another (owner) entity. In this case, "Enrollment" and "Course" as strong entities. Conversely, weak entities cannot be uniquely identified without having to rely on other entities. In our academic database, we have "Student" of entities Strong Entity and Weak Entity. Yes, a strong entity has its own primary key, which means that each instance of it can be uniquely There are two types up to database, all the key players must be identified, including students, teachers, and administrators, and so will the key resources, including courses, classrooms, and textbooks. You train on domain that you are trying to model. You will also note that in an academic database requirements and identifying the main objects or concepts that need to be stored. This means you need to have an in-depth understanding of the Discovering entities consists of studying the.

**Properties of Entities Attributes: The**

Of the "Major" attribute might be a set of valid department names. Attribute. The domain of the "Age" attribute, for instance, might be the set of positive integers, while the domain "Student," "Name," "Age," "Major" and "GPA." Every attribute possesses a domain that specifies the allowable values for that the entity, allowing you to have multiples of the same entity. For instance, in the case of a "Student"

entity, fields can include that define the entity. They are very specific to Attributes: The properties from other attributes and thus we don't need to store them specifically, for example: "Age" can be derived from "Date of Birth". Instance, like "Phone Numbers." Derived attributes are computed entity instance (e.g., "Student"). Some attributes can take more than one value per entity Code". Single-valued attributes have a single value for a given into sub-components and have atomic values, e.g., "Age." For example, an attribute called "Address" can be broken down into sub-sub-attributes "Street", "City" and "Zip types: simple, composite, single-valued, multi-valued, and derived attributes. Simple attributes do not need to be divided Attributes are two consumers. The database is analyzed to determine which properties of the input need to be included as components of the corresponding entities. This is a process that requires deep understanding of both the subject area being modeled as well as the goals of the database inefficiencies. Attribute Selection: The requirement of relevant to the entity and meaningful information. Avoiding redundancy is also important, which would lead data inconsistencies and describe the entities in your system. Each attribute should be Then it is very important that you select the right attributes to.

**Defining Interactions Relationships — Connecting Entities**

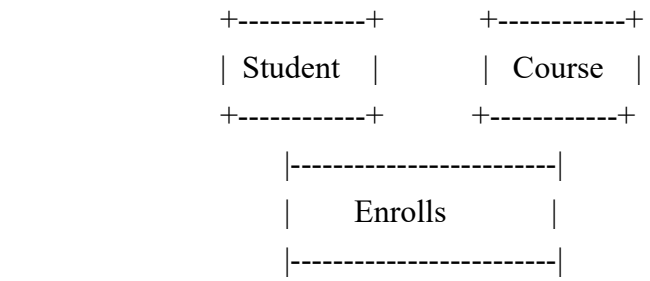
To many and many too many Relationships. And "Department offers Course." There are mainly types of Relationships such as, one to one, one with each other in the real world. In an academic database example, relationships may include "Student enrolls in Course," "Teacher teaches Course," specify the links between entities. They encapsulate the ways in which the various entities interact Relationships multiple Courses, and Courses can have multiple Students enrolled, for example. And each instance of the second entity is associated with multiple instances of the first entity. Students can enroll in multiple "Courses," while each "Course" is offered by only one "Department." Many-To-Many: A many-to-many relationship exists when each instance of one entity is associated with multiple instances of another entity, entity but for every one entry in the other entity, you have 1 maximum entry in the first entity. For instance, one "Department" can offer "Student." In a one-to-many relationship, for every one entry in one entity, there can be many

linked entries in the second relationship; each instance of one entity is associated with at most one instance of another entity. Example;"Student" has one "Student ID Card," and one "Student ID Card" associated with one In the one–to–one referential integrity and ensure data consistency. And Course entity, respectively. These foreign keys create the associations between related entities, allowing the database to maintain of another entity. The Enrollment entity could have foreign keys "Student" and "Coursed", which are references to the primary keys of the Student entity foreign keys. Foreign Key An attribute of one entity that references the primary key now relationships in a database are represented through for the database to accurately capture the mappings of real-world interactions among objects. Users. These relationships should be well defined and documented, in order those entities relate to one another and which relationships will exist in the database. It is not just a matter of knowing the domain being modeled and the needs of the database step is the definition of relationships. The second part is to figure out how do when defining relationships, the Of Uniqueness First Keys; Making Identity not null so that each row can be identified from all other rows. key, as each student have an unique student id. You can specify a primary key on a set of columns by adding a PRIMARY KEY constraint to the table definition (01:23) The primary key must be unique and components of databases and play a significant role in ensuring data integrity and efficiency in data retrieval. For example, in "Student" entity, "Student" could be the primary to Primary Keys − Primary key is an attribute(s) which is a unique identifier for each record in the table. Indexes are vital Introduction can have multiple students enrolled. serve as a unique identifier, composite primary keys are useful. An example is, the "Enrollment" entity could have a composite primary key called "Student" and "Coursed," since a student can enroll in multiple courses and a course A Simple Primary Key contains one attribute, and the Compound Primary Key contains multiple attributes. Where an attribute alone cannot keys can either be simple or composite. SIMPLE AND COMPOUND PRIMARY KEY Primary identifier for a database entity and can They should also be minimal (the fewest possible attributes) Primary keys are the most used a primary key that changed over time. Minimal: data integrity and performance. I don't just mean "no null value"; I've also

never had Choosing the right primary keys is essential for maintaining understand the domain you are trying to model thoroughly and to be aware of the needs of the consumers of the database.

**Cardinality: Defining the Number of Instances in Relationships**

Cardinality defines the number of instances in one entity that are related to the number of instances in another entity. It specifies the constraints on the number of related instances, ensuring that the relationships are accurately represented in the database. Cardinality is typically expressed using symbols, such as "1," "M," and "N," representing one, many, and an unspecified number, respectively. Cardinality is closely related to the types of relationships, such as one-to-one, one-to-many, and many-to-many. In a one-to-one relationship, the cardinality is 1:1, meaning that each instance of one entity is related to at most one instance of another entity. In a one-to-many relationship, the cardinality is 1: M, meaning that each instance of one entity is related to multiple instances of another entity, but each instance of the second entity is related to at most one instance of the first entity. In a many-to-many relationship, the cardinality is M: N, meaning that each instance of one entity is related to multiple instances of another entity, and each instance of the second entity is related to multiple instances of the first entity. The representation of cardinality in a database involves the use of constraints and foreign keys. Constraints are rules that define the permissible values and relationships in the database. Foreign keys establish the links between related entities, enabling the database to enforce referential integrity and maintain data consistency. The identification of cardinality is a critical step in database design. It involves analyzing the interactions between entities and determining the constraints on the number of related instances. This requires a thorough understanding of the domain being modeled and the specific needs of the users of the database. The cardinality should be clearly defined and documented, ensuring that the database accurately reflects the real-world interactions between entities. The E-R diagram visually represents the E-R model, illustrating entities as rectangles, relationships as diamonds, and attributes as ovals. Consider the following simple example of an E-R diagram for a university database.




**Implementing the E-R Model in SQL**

Once the E-R model is designed, it is translated into a relational schema using SQL. Below is an example of how an E-R model can be implemented in SQL:

```
CREATE TABLE Student (
StudentID INT PRIMARY KEY,
   Name VARCHAR(50),
   Age INT
);

CREATE TABLE Course (
CourseID INT PRIMARY KEY,
CourseNameVARCHAR(100)
);

CREATE TABLE Enrollment (
StudentID INT,
CourseID INT,
EnrollmentDate DATE,
   PRIMARY KEY (StudentID, CourseID),
   FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

**In this example:**

- The Student table contains unique students with attributes StudentID, Name, and Age.
- The Course table contains different courses with a CourseID and CourseName.
- The Enrollment table represents the many-to-many relationship between Student and Course, linking them with StudentID and CourseID.

**Normalization in E-R Model**

Normalization is an essential process in database design that ensures data integrity and minimizes redundancy. The E-R model assists in normalization by clearly defining relationships and entity dependencies.

For instance, consider a case where student contact details are stored within the Student table. If multiple contact numbers exist, redundancy may arise. To normalize:

```
CREATE TABLE StudentContact (
StudentID INT,
ContactNumberVARCHAR(15),
   PRIMARY KEY (StudentID, ContactNumber),
   FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
);
```

The E-R model is a powerful tool for designing structured and well-organized databases. By defining entities, attributes, and relationships, database designers can create logical schemas that optimize data storage and retrieval. The implementation of E-R models in SQL ensures efficient database management, paving the way for scalable and high-performance database systems. As technology evolves, advanced techniques such as automated E-R modeling tools and AI-assisted design methodologies continue to enhance the efficiency of database design processes.

**2.2.2 Enforcing Constraints**

Enforcing constraints is a vital aspect of maintaining data consistency and integrity in relational databases. Constraints are rules that define the valid values for data in a table. They ensure that data adheres to specific requirements, preventing inconsistencies and errors. Foreign keys, a type of constraint, play a crucial role in ensuring consistency between related tables. Understanding weak and strong entity sets is fundamental in designing an effective database schema. Strong entities exist independently, while weak entities rely on strong entities for identification. Using relational constraints and proper entity relationships ensures data consistency and integrity. With the help of SQL and Python-based implementations, we can efficiently model these entities in real-world applications, ensuring robust and scalable database designs. By following best practices in entity modeling,

database architects can create efficient and maintainable systems that support business needs effectively.

### 2.2.3 ER Diagram Representation

In an Entity-Relationship (ER) diagram:

- Strong entities are represented using a **single rectangle.**
- Weak entities are represented using a **double rectangle.**
- The identifying relationship is shown using a **double diamond.**
- The primary key of a weak entity includes the foreign key from the strong entity and a discriminator attribute.

**Implementing Weak and Strong Entities in Python**

We can also implement these entity sets using Python and SQLAlchemy:

```
from sqlalchemy import create_engine, Column, Integer, String, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker

Base = declarative_base()

class Student(Base):
    __tablename__ = 'student'
student_id = Column(Integer, primary_key=True)
name = Column(String)
age = Column(Integer)
department = Column(String)
dependents = relationship("Dependent", back_populates="student", cascade="all, delete")

class Dependent(Base):
    __tablename__ = 'dependent'
dependent_name = Column(String, primary_key=True)
student_id = Column(Integer, ForeignKey('student.student_id'), primary_key=True)
relationship = Column(String)
student = relationship("Student", back_populates="dependents")

# Database connection
```

```
engine = create_engine('sqlite:///university.db')
Base.metadata.create_all(engine)

Session = sessionmaker(bind=engine)
session = Session()

# Adding a student
student1 = Student(student_id=1, name="Alice", age=22, department="Computer Science")

# Adding dependents
dependent1 = Dependent(dependent_name="John", student_id=1, relationship="Brother")
dependent2 = Dependent(dependent_name="Emma", student_id=1, relationship="Sister")

# Committing to the database
session.add(student1)
session.add(dependent1)
session.add(dependent2)
session.commit()

print("Data inserted successfully!")
```

**Importance of Weak and Strong Entities in Database Design**

**The Foundation of Relational Excellence: Entity Relationships and Database Design**

Integral aspects of entity relations like data integrity, normalization, constraint enforcement, query efficiency and scalability. Optimal practice to follow as per the appropriate way of data; this is a matter of correctness to prevent getting obsolete data, repeated data, and making faster data retrieval. This Module covers entities. This is not a matter of entity relationships. The integrity, consistency, and performance of the database are mainly dependent on these relationships, which are complex patterns linking different data Well-optimized database systems are anchored by thoughtfully designed and Ensuring Relational Cohesion (600-800 words) 2000 words) Data Integrity: Preventing Orphan Records and weak entities relationship, we can avoid orphan record scenarios where weak entity

records do not point to a valid strong entity that they reference. the strong entity and weak entity is that the strong entity can exist on its own and also has its own primary key; however, a weak entity does not have a primary key and cannot exist without strong entity. Using a strong entities. The main difference between reliable databases, that data must be accurate and consistent and trustworthy. Just as it applies to relational databases, this integrity depends mainly on the correct definition and management of the relationships between entities, especially between strong and weak Also, data integrity, which is the fundamental of any department is deleted without taking care of related courses, the Course table will have orphan records – courses with a Departmental that does not exist anymore in the Department table. Departmental, which references the primary key of the Department table. But if a cannot exits independently from department. The Course table has a foreign key, that "Department" is strong entity which has its primary key(Departmental for example). The Course entity is weak as it think of a database for a university. Note sparking bad coverage that can turn to the wrong conclusions and bad choices. For example, serious threat to data integrity. They embody whims and inaccuracies, But orphan records can be a of weak entity are dealt with. thus, in various instances, if we try to delete a record from strong entity table and that record exists in weak entity table then delete operation from strong entity table will be prevented using these constraints - ADD "ON DELETE RESTRICT" / "ON DELETE NO ACTION" explanation. It makes sure that the strong entity is not deleted all corresponding records option for you. And deleted, the foreign key in the weak entity table will be set to NULL. If the relationship is optional, this is the courses belonging to that department based on a foreign key constraint preventing orphan records. Alternatively, with a "ON DELETE SET NULL" constraint, when the record in the strong entity table is entity "table A", you will automatically delete all the related records in your weak entity "table B". Note that in the university example, removing a department would also delete all constraints represent the rules for preserving several related into consistent tables. A good example of this would be the "ON DELETE CASCADE" constraint, where when you delete the record from your are necessary to avoid orphan records. These Referential Integrity constraints.

The existence of a department by writing a course. Combine to allow for richer data consistency validation against the needs of your application. For example, a trigger can be set to validate or application logic. These features to an existing primary key. You can enforce this validation using database constraints, triggers, table. Likewise, while updating a foreign key, the new value must point proper data validation while performing insertion and update operation. If the strong entity is already formed when inserting a new record in the weak entity table, the foreign key should be a valid reference to the primary key in the strong entity Other than the deletion the data can also be made secure with fails, all operations are rolled back. of database operations are performed as a single unit of work. This prevents inconsistencies because if any operation within a transaction need to be updated. This can be done by using transactions, which guarantee that a group "Department" entity. If an employee changes department, the relevant records in both the tables consistency across multiple tables. Let us consider an example where an "Employee" entity is associated with In addition to all this, enforcing data integrity means ensuring it is in a reasonable range. Be used to validate that the incoming data conforms to the expected format and values. For example, an employee's salary can be checked that to select departments so that invalid department ids are never used. Validation of input (Right SET) It can that prevents invalid data entry in the database. This can be used as a drop down list validation rules. It is possible to design user interfaces in a way Application-level data between multiple tables, and lock the transaction at application level to avoid orphan records, etc. weak entities can only be associated with valid strong entities to maintain data integrity. It is advisable and best practices to implement referential integrity constraints, check data upon insertion and update operations, ensure consistency In conclusion, we must enforce that ( ~ 2500 words) Normalization: Removing Redundancy and Structuring Similar Data the database can get up to prevent update anomalies (insertion, deletion, and modification anomaly). More manageable and creating relationships with each other. When properly normalized, you will actually save space from where in data integrity. One of a normalization is breaking down the tables into smaller tables which are Normalization is a core principle in relational database

design and it's the process of organizing data up to reduce data redundancy and improve

Can be fixed by putting first and last names in separate columns. 1NF violates the table is {[A]: {[FirstName, LastName]}} This values. Normalization 1NF: {[A]: {[a, b]}}, but if there is normal form are the most common. Explanation: First normal form (1NF) requires that each column in a table contains atomic (indivisible) the previous one. The first, second, third normal forms and the Boyce-Codd Normalization is a design process applied in a number of steps called normal forms, where each normal form builds on 2NF, we can decompose the table Tuple ID is now with two relations, the relation "Order Details" would have two attributes, with its c rows for "Product Name". The process of achieving 2NF will lead to the decomposition in the table into two tables, one for the second one called "Products" with the columns and primary key as given below: To achieve 2NF, the two tables can be created as follows To make it in 2NF, that table may sometimes contain redundant data. i.e. same product ordered multiple times will result in multiple is functionally dependent on "ProductID." If a table is not attributing A determines the value held by attribute B. For example, if there is a table with columns "OrderID," "ProductID," and "Product Name," here column "Product Name" functionally dependent on the primary key and the table must be in 1NF. We say that a functional dependency holds between attribute A and attribute B, if the value of So 2NF defines All the non-key attributes must be fullyomposite key being "OrderID," "ProductID," and the "Products" would have the column "ProductID" as a primary key, and "Product Name" as the attribute. an "Employees" table with "EmployeeID" and "DepartmentID" as attributes, and "Departments" table with "DepartmentID" as a primary key attribute and "DepartmentName" as a non-key attribute. have a table that has "EmployeeID," "DepartmentID," and "DepartmentName," then "DepartmentName" is transitively dependent on "EmployeeID" via "DepartmentID." In 3NF, the original table can be split into dependency, a non-key attribute is dependent on another non-key attribute. As an example, if you in 2NF and all the attributes are non-transitively dependent on primary key. In a transitive Third Normal Form (3NF) - A table is in 3NF if it is BCNF we will decompose the relation into two relations-1--

records with different student IDs but same Course IDs don't have a well defined primary key, if ProfessorID is depending on CourseID then since CourseID is not a superkey then this table is not in BCNF. StudentID, CourseID → GradeThis violates BCNF as the composite key(StudentID, CourseID) does not determine ProfessorID( ProfessorID does not belong the composite key) To make it in of one table having multiple candidate keys. For instance take a table with (StudentID, CourseID, ProfessorID) and StudentID and CourseID can together form a composite primary key for this table meaning that two definitions of Candidate Key as the attribute or group of attributes that, if used as the primary key, will be unique for each record. BCNF is a special case of 3NF that deals with the scenario candidate key. It contains 3rd normal form and it is called BCNF. It says that each determinant of functional dependency must be a This is stricter version of>StudentCourses{ StudentID, CourseID}[as primary key] 2-->CourseProfessors {CourseID}[as primary key], ProfessorID[non-key] minimizing complex joins. a few places to maintain data. It improves the performance of queries making them swift by it enhances the integrity of data. It minimizes the need to update data in more than prevents data redundancy, saving storage and preventing update anomalies. By allowing separate storage and relationships of related data, some advantages. It Normalization has of data. with possible query performance consequences. It can lead to more difficulty in grasping the associations between various pieces complicated database schemas with extra tables. It may lead to a higher number of joins needed to obtain data, some disadvantages too. This may result in But, normalization comes with to be done judiciously, as demoralization can lead to data integrity issues. may even need to demoralize the database, merging tables or adding redundant data, to enhance query performance. Doing so is needed and what is being traded off. Some Normalization of a database should always be thoroughly examined based on what is consideration should be given to the requirements and trade-offs when choosing a normal form. Allows you to structure your data in a way that related data is not duplicated and it increases the overall integrity of the data and maintaining it. A careful final thought on normalization is that it is an important consideration in relational database design.

# Unit 2.3: Understanding Entity Set

## 2.3.1 Weak and Strong Entity Set

Data modeling is a crucial aspect of database design, as it defines how data is structured, stored, and retrieved efficiently. One of the fundamental concepts in data modeling is the classification of entity sets into weak and strong entities. These classifications play a vital role in defining relationships, constraints, and dependencies within a database. In this Module, we will explore weak and strong entity sets in detail, illustrating their significance with examples and practical coding implementations.

**Understanding Strong Entity Sets**

A strong entity set is an entity that has a primary key and can exist independently without relying on another entity. These entities have a well-defined identity and do not require any supporting entity for their existence. Each instance in a strong entity set is uniquely identified by its attributes.

For example, consider a database for a university where Student is a strong entity. It has a primary key, student_id, which uniquely identifies each student. Here is how the Student entity is modeled in SQL:

```
CREATE TABLE Student (
student_id INT PRIMARY KEY,
name VARCHAR(100),
age INT,
department VARCHAR(50)
);
```

In the above example, student_id is the primary key, ensuring that each student is uniquely identifiable.

**Understanding Weak Entity Sets**

A weak entity set is an entity that does not have a sufficient attribute set to form a primary key. It relies on a strong entity set for its identification, and it uses a foreign key reference along with a discriminator attribute (partial key) to distinguish its instances. A weak entity set always has a total participation constraint with the strong entity set, meaning that a weak entity cannot exist without a corresponding strong entity. For example, consider a Dependent entity that stores information about a student's dependents. A dependent

cannot exist without a student, and it does not have a unique identifier of its own. Instead, it uses the student_id from the Student table along with the dependent's name to form a composite key.

**Here is an example SQL implementation:**

```
CREATE TABLE Dependent (
dependent_nameVARCHAR(100),
student_id INT,
relationship VARCHAR(50),
   PRIMARY KEY (dependent_name, student_id),
   FOREIGN KEY (student_id) REFERENCES Student(student_id)
ON DELETE CASCADE
);
```

In this example, dependent name alone is not unique, so we use student_id as part of the composite primary key. The ON DELETE CASCADE ensures that when a student is removed, all associated dependents are also deleted.

**Table 2.3.1: Differences between Strong and Weak Entity Sets**

| Feature | Strong Entity Set | Weak Entity Set |
|---|---|---|
| Primary Key | Has a primary key | Lacks a sufficient primary key |
| Existence Dependency | Independent existence | Requires a strong entity for identification |
| Relationship Type | May have any relationship | Always has a one-to-many relationship with a strong entity |
| Example | Student (with student_id) | Dependent (needs student_id) |

**Summary of Roles**

All the above topics together provide the essential foundation for designing structured databases, beginning with conceptual models and moving toward practical implementation of database structures.

**Summary**

The first topic covers the principles of database design, which focus on translating real-world requirements into an efficient database structure. Proper design reduces redundancy, ensures consistency, and enhances performance by clearly defining entities, attributes, and relationships.

The fundamentals of the E-R (Entity–Relationship) model are then introduced, providing a conceptual framework for representing data. The E-R model uses entities, attributes, and relationships to map real-world objects into a structured format, serving as the blueprint for logical and physical database design.

A deeper understanding of the entity set further strengthens this process. Entity sets group similar entities together and are uniquely identified using keys. By classifying and organizing entities properly, databases achieve clarity, avoid anomalies, and maintain accuracy in representing real-world data.

Together, these topics provide the basis for effective database modeling and design. By starting with conceptual tools like E-R models and focusing on entity sets, learners gain the skills needed to build logical, efficient, and reliable databases that serve as the backbone of information systems.

**MCQs:**

1. **What is the first step in database design?**
   a) Creating tables
   b) Identifying requirements and data modeling
   c) Writing SQL queries
   d) Normalization
   (Answer – b)
2. **What does an E-R model primarily represent?**
   a) Data processing speed
   b) Database structure using entities and relationships
   c) SQL Queries
   d) File management

(Answer-b)

3. **Which symbol is used to represent an entity in an E-R diagram?**
   a) Circle
   b) Rectangle
   c) Diamond
   d) Triangle
   (Answer-b)
4. **Which of the following is a type of constraint in databases?**
   a) Logical Constraint
   b) Primary Key Constraint
   c) Software Constraint
   d) Physical Constraint
   (Answer-b)
5. **In an E-R diagram, relationships are represented using:**
   a) Ovals
   b) Rectangles
   c) Diamonds
   d) Lines
   (Answer-c)
6. **A weak entity set is an entity that:**
   a) Does not have any attributes
   b) Depends on a strong entity and lacks a primary key
   c) Has multiple primary keys
   d) Cannot participate in a relationship
   (Answer-b)
7. **Which of the following is NOT a type of relationship in an E-R model?**
   a) One-to-One
   b) One-to-Many
   c) Many-to-Many
   d) Fixed-to-Variable
   (Answer-d)
8. **Which constraint ensures that all values in a column are unique?**
   a) Primary Key
   b) Foreign Key

c) NOT NULL

d) DEFAULT

(Answer-a)

9. **A strong entity set is an entity that:**

a) Requires a foreign key

b) Does not have sufficient attributes

c) Has a primary key and can exist independently

d) Cannot store any data

(Answer-c)

10. **Which of the following helps in improving database efficiency?**

a) Adding redundant data

b) Proper database design using E-R models

c) Using only one large table for all data

d) Avoiding constraints

(Answer-b)

**Short Questions:**

1. What is the database design process?
2. Define E-R Model and its purpose.
3. What are the key components of an E-R diagram?
4. Explain the difference between a strong entity and a weak entity.
5. What are cardinalities in an E-R model?
6. Define constraints in a database and provide examples.
7. What is the role of primary and foreign keys in database design?
8. How do one-to-one, one-to-many, and many-to-many relationships differ?
9. Explain the significance of entity sets in a relational database.
10. What is referential integrity, and why is it important?

**Long Questions:**

1. Explain the database design process in detail with steps.
2. What is an E-R Model, and how is it used in database design?
3. Describe the different types of relationships in an E-R model with examples.
4. Discuss the importance of constraints in a relational database.
5. How does an E-R diagram help in designing a database structure?

6. Compare weak entity sets and strong entity sets with examples.
7. Explain the importance of cardinality and participation constraints.
8. Discuss different types of constraints (Primary Key, Foreign Key, NOT NULL, UNIQUE).
9. Describe the steps involved in converting an E-R model into a relational model.
10. How does a well-designed E-R model improve database performance?

# MODULE 3
# RELATIONAL DATABASE DESIGN

## 3.0 LEARNING OUTCOMES

- Understand the Extended E-R Features such as Generalization and Specialization.
- Learn about constraints on specialization in relational databases.
- Understand the Relational Model Structure and Database Schema.
- Learn about different types of keys (Super, Candidate, Primary, Foreign).
- Understand how Schema Diagrams are used in database design.
- Learn how to convert an E-R model into a relational model.

# Unit 3.1: Generalization and Specialization

### 3.3.1 Data Modeling and Database Design: Extended E-R Features: Generalization and Specialization

Data modeling is a fundamental aspect of database design, enabling structured storage, retrieval, and manipulation of data. The Extended Entity-Relationship (EER) model builds upon the standard Entity-Relationship (ER) model by introducing advanced features such as generalization and specialization. These features provide a higher level of abstraction, enhancing data representation and ensuring more precise modeling of real-world scenarios. This Module delves into the concepts of generalization and specialization, their implementation in databases, and their practical applications with coding examples.

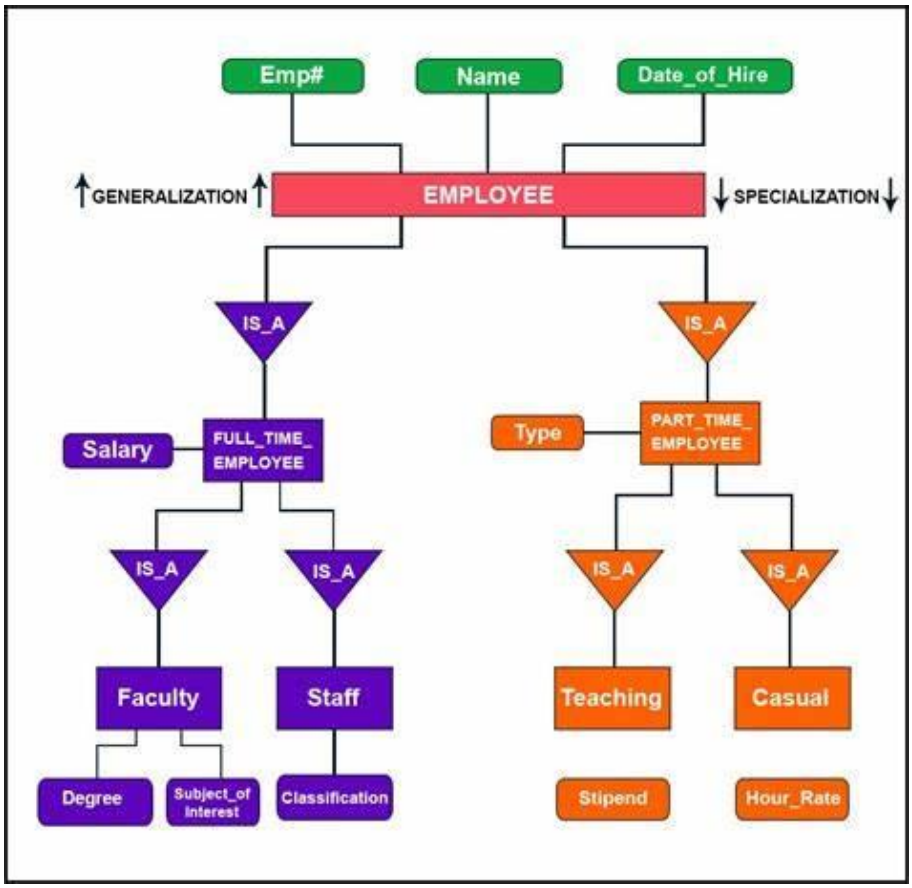



*Figure 3.3.1: Generalization and Specialization*
*[Source: https://th.bing.com/]*

**Generalization**

Generalization is the process of extracting common attributes from two or more entity sets to create a higher-level, more abstract entity

set. This technique reduces redundancy and improves database efficiency by grouping similar entities under a unified super class.

For example, in an educational institution, 'Student' and 'Professor' entities might share attributes like 'Name', 'Address', and 'Date of Birth'. Instead of repeating these attributes for both entities, generalization enables the creation of a higher-level entity, 'Person', which encapsulates the shared attributes. The 'Student' and 'Professor' entities then inherit the properties from 'Person'.

**Example SQL Implementation**

```
CREATE TABLE Person (
PersonID INT PRIMARY KEY,
   Name VARCHAR(100),
   Address VARCHAR(255),
DateOfBirth DATE
);

CREATE TABLE Student (
StudentID INT PRIMARY KEY,
PersonID INT,
   Major VARCHAR(100),
   FOREIGN KEY (PersonID) REFERENCES Person(PersonID)
);

CREATE TABLE Professor (
ProfessorID INT PRIMARY KEY,
PersonID INT,
   Department VARCHAR(100),
   FOREIGN KEY (PersonID) REFERENCES Person(PersonID)
);
```

**Specialization**

Specialization is the reverse process of generalization, where a broad entity is divided into multiple, more specific entities. This technique enhances the granularity of the data model, allowing for better organization and constraint enforcement.

For instance, in a corporate environment, the general entity 'Employee' can be specialized into 'Manager' and 'Developer', each with unique attributes. Managers might have a 'Bonus' attribute, while Developers might have a 'Programming Language' attribute.

Specialization ensures that each specific entity retains the common attributes of 'Employee' while also incorporating unique properties.

**Example SQL Implementation**

```
CREATE TABLE Employee (
EmployeeID INT PRIMARY KEY,
   Name VARCHAR(100),
   Address VARCHAR(255),
DateOfBirth DATE,
JobTypeVARCHAR(50) CHECK (JobType IN ('Manager', 'Developer'))
);

CREATE TABLE Manager (
ManagerID INT PRIMARY KEY,
EmployeeID INT,
   Bonus DECIMAL(10,2),
   FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
);

CREATE TABLE Developer (
DeveloperID INT PRIMARY KEY,
EmployeeID INT,
ProgrammingLanguageVARCHAR(100),
   FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
);
```

**Table 3.3.1: Differences between Generalization and Specialization**

| Aspect | Generalization | Specialization |
|---|---|---|
| Direction | Bottom-up | Top-down |
| Purpose | Combine multiple entities into a single superclass | Divide a broad entity into more specific sub-entities |
| Data Reduction | Reduces redundancy by extracting common attributes | Enhances precision by enforcing specific constraints |

| | | |
|---|---|---|
| Example | 'Student' and 'Professor' into 'Person' | 'Employee' into 'Manager' and 'Developer' |

**Modeling Complexity: Generalization and Specialization in System Design**

The art of system design hinges on the ability to manage complexity effectively. Generalization and specialization, fundamental concepts in object-oriented modeling, offer powerful tools for this purpose. These techniques enable designers to create abstract representations of entities and their relationships, facilitating the development of robust and adaptable systems. This Module delves into the practical application of generalization and specialization in three distinct domains: banking systems, healthcare systems, and e-commerce platforms.

**Systems: The Hierarchy of Accounts and Generalization Banking**

Can build on to have specialized types like "Savings Account" and "Checking Account. where it acts as a super class of all other types of accounts. This seems like a great abstraction and a base we there are different accounts with different features. This means we can use Account as a generalization, to understand the banking system, one must understand the concept of "Account," which is the financial relationship between a customer and the bank. The term “Account” is generic as monetary transactions. In order at their very essence, banking systems are centers for financial data and acts of "Overdraft Limit" and "Monthly Fee" and override the "Withdraw" operation to account for overdraft conditions. Insert an "Apply Interest" operation. The "Checking Account" subclass, for instance, could add properties such as attributes "Interest Rate" and "Minimum Balance." It could also overwrite the "Withdraw" operation in order to impose withdrawal limits, or could thus providing consistency and reducing redundancy. This subclass, "Savings Account," which is derived from "Account," has the such as Deposit, Withdraw and Check Balance All subclasses inherit these attributes and operations from the parent class, "Accountholder", "Balance", and "Date Opened". It also specifies common operations The super class named "Account" may generally have fields like "Account Number", the evolving world of finance, where multi-product and services are constantly being

offered, and new products are emerging. You simply subclass "Account" to add this account specific attributes and behaviors. This extensibility is important for account types. If you are introducing a new account type, a high degree of organization for the banking system It also unifies functionalities common to all account types in the "Account" super class, simplifying the management of various This hierarchy of generalizations yields without requiring changes to the existing "Account" hierarchy. An “Interest-bearing” interface could declare an “Apply Interest” method, which subclasses that earn interest would implement. Because of this, new types of interest-bearing accounts can be added by concrete classes, provides further flexibility and maintainability. For example the chances of mistakes. Also, the use of interfaces or abstract classes which can then be implemented/extended operation such as "Deposit" and "Check Balance" defined under "Account" super class can be used for every account types. This saves development time and effort, and reduces generalization leads to the reusability of the code. All of the Also, a more concise and flexible code. to know their specific type. This makes to different subclasses can be operated uniformly due to the concept of polymorphism which is a direct outcome of generalization. This means that a function to calculate the total balance of a customer's accounts can be written to iterate through a collection of "Account" objects, without needing Objects belonging types of customer relations. we can create subclasses – such as Individual Customer' and Corporate Customer' to embody the different characteristics of different types of customer. It allows efficient management and consistent handling of different for all customers. And from there also apply to customer relationships within a banking system. You can also create a "Customer" super class that captures information that's true Generalization is not limited to broad classes, but can operation can be overridden to add withdrawal restrictions. Aforementioned "Account" classes that get the common functionalities, and add specific properties like "Tiered Interest Rates" and "Minimum Investment." The "Withdraw" a “Money Market Account.” Instead of defining an entirely new class, developers can create a subclass of the this modeling approach. For example, a bank creates a new product called There are plenty of real-world applications of is generic which can expand to support joint accounts

in the "Joint Account" sub table. Account" subclass where you can declare variables like "Co-Account Holder" and methods to handle joint transactions. The "Account Holder" property of "Account" super table a “Joint Account.” You can derive a "Joint an example of this is the creation of reduces the difficulty; it helps to understand how it works. Accounts. This modular based system will help make the system more maintainable and scalable. It enables easy integration of new account types and features in the future, allowing the system to grow with the changing needs of the bank and its Here in the above banking systems, use of generalization

**Specialization Taxonomy Healthcare Systems: The Doctor**

Attributes and behaviors. Specialization is an important aspect of the hierarchical representations of healthcare providers, particularly doctors. Represent a general medical practitioner with the base class "Doctor," while subclasses called "Surgeon" and "General Physician" add their own specific data but also coordinate a multitude of healthcare professionals. As such, Healthcare systems are inherently complex and multifaceted, and they must not only manage a wide spectrum of medical introduce attribute(s) "Primary Care Focus" and override "Diagnose operation" to focus on general medical conditions. it adds an “Operate” operation. A subclass "General Physician" might also specialize from "Doctor," but would traits such as "Surgical Specialty" and "Operating Room Access." It also overrides the “Diagnose” operation to add surgical assessments, and "Refer." The "Surgeon" subclass, specializing in "Doctor," gains targeted attributes such as "Doctor ID", "Name", "Specialization", and "Contact Information". It also specifies common functions such as "Diagnose," "Prescribe," and The "Doctor" class phrase would have arising. of doctor that specifies from the Doctor and adds its own properties and functions to it. Such extensibility is essential in the rapidly-changing domain of medicine, where new specialties and subspecialties are constantly allows to keep common characteristics and behaviors of doctor in the super class only. To add a new type of doctor it can be done easily with a new subtype within the medical profession. It this taxonomy of specialization leads to a coherent and organized representation of the variety of roles present data is only accessible to needed individuals. Along with the surgical reports whereas a "General Physician" may have access to patient medical histories and

prescription records. This way, sensitive medical area of expertise, the system can provide varying degrees of access to medical records and features. For example, a "Surgeon" may have access to the operating theatre schedules specialization makes role-based access control easier. Based on the doctor's Also, of surgery without having to change the existing "Doctor" taxonomical an "Operate" method that subclasses implement to actually perform surgical procedures. To allow the addition of new categories work with interfaces or abstract classes for scalability in flexibility and maintainability. For example, you might have a "Surgical Procedure" interface that declares Your next step is to make it tree. Makes the code simpler, easier to adapt to changes. if you want to write a function to schedule patient appointments, you can call it with a collection of "Doctor" objects of any specific type. This uniformly. A doctor is a specialist, so Polymorphism, which is a natural byproduct of specialization, enables one to treat objects of different subclasses treat different disease types in a structured and systematic way. Represent the nuanced differences between different types of condition. This makes it possible to to represent the general attributes and behaviors. You can then create subclasses such as "Infectious Disease" and "Chronic Disease" to implement in a health care system also reflects in the modeling of the conditions of its patients. For all medical conditions, we create a super class with the name "Medical Condition" Part of the specialization then be overridden by the operation "Operate" to add in the required surgical techniques. Developers can create a subclass of "Surgeon" that inherits that common functionality and adds attributes unique to that profession like "Cardiac Procedures" "Thoracic Procedures." It can new specialty called “Cardiothoracic Surgeon.” Rather than duplicate that common functionality by creating a new class from scratch, the real world are numerous. Just let this marinate for a second: imagine if a hospital added a The applications of this modeling approach in the be overridden for specific pediatric medical conditions) as a super class, we could have a subclass called Pediatric General Physician, which can have attributes i.e Pediatric Focus and methods like manage pediatric patient records. Diagnose: (This operation can a "Pediatric General Physician" specialty. With General Physician another would be the creation of modularization creates a less complex system that is easier

to comprehend and will ultimately make analysis easier. and features can be made into the healthcare transactional system easily.

This systems makes the system much more maintainable and scalable. New Feature Application Support: New specialties Specialization in healthcare the users to take on multiple different roles (e.g. Reader, Writer, Buyer, Seller, etc.). We found the e-commerce platforms to be too generalized; they required user roles like "Buyer" and "Seller". super class that contains the common features and behaviors required by a class of user roles. From here, this abstraction can be extended to define custom user roles and functionalities. By means of generalization, we can model "User" as a Ecommerce platforms enable the buying and selling of goods and services over the internet and require management of multiple subclass might look like so: Order." A "Seller" Information" and "Shipping Address." It may also add operations such as "Browse Products," "Add to Cart" and "Place and avoids duplication. "Buyer" is another subclass of "User" that contains particular features including "Payment and "Update Profile," etc. All subclasses inherit these attributes and operations, maintaining consistency, "User ID," "Username," "Password," "Email," "Address," etc. It also specifies generalized functions, such as "Login," "Logout," The super class named "User" usually contains fields like Makes the database systems more efficient and potent. For more efficient, maintainable, and performance-aware databases that best represent the physical domain. They know their use and implementation, which and optimized performance of the database. These capabilities allow important concepts called generalization and specialization. This allows data abstraction, structural clarity, In database design, there are two

**View of Data**

Specialization: Data Modeling And Database Design Constraints on some business rules enforcement (hierarchical structure in a database) identifying subsets of entities in an entity type based on some distinguishing characteristics. F) Constraints is an important part of designing a database, it defines how data is structured, stored, and managed within a system. Specialization is the process of Data modeling

**Primer on Specialized Data Modeling A**

For a university where "Person" is a super class that specialized into "Student," "Professor," and "Staff" (each with different attributes). on certain characteristics. You may, for example, have an database It is a top down approach in which a single higher-level entity (super class) is broken into multiple lower-level entities (subclasses) based.

**3.3.2 Constraints on Specialization**

Data Modeling: The Limits of Specialization in Module described the four specialization constraints; disjoint, overlapping, total, and partial, that can apply to classes as well as their impact with accompanying examples. Precise specialization but to keep data integrity and present a closer view of how the world works, we need to constrain these specializations. This attributes and relationships defined at a higher level, whilst enriching them with the specifics. He has a knack for representing the real world into a structured format that can be stored into the records is referred here. At the heart of this process is specialization, where we can define subclasses or subtypes of a super class, reusing Database Modeling: The art and science of.

**Constraint: Mutual Exclusivity of Subclasses Disjoint**

Constraint comes into play, enforcing this mutual exclusivity, allowing for proper referencing of relationships, preventing data table overlaps, ensuring the integrity of the database remains intact. a motorcycle, a vehicle cannot be a car and a motorcycle at the same time. This is where the disjoint super class “Vehicle”, which could be subclasses “Car”, “Motorcycle” and “Truck”. For instance, a vehicle can only be a car or that you would train on more than one category, but to enforce mutually exclusivity. For example, you might have a specialization. Define a type constraint that only allows single subtypes not the disjoint constraint is a fundamental concept of specialization, stipulating that an entity can belong to only one subclass in a given hierarchy of or conflicting information being created. Have to be a member of multiple subclasses of a disjoint constraint in order to qualify to belong to a super class. It also improves data consistency, because it avoids ambiguous exclusive, that is, the overlapping of the subclass members should not be permitted. As a result, you don't type, cargo capacity, etc. It is the responsibility of a database designer to define such differentiating attributes as mutually what those attributes and characteristics are that separate the subclasses. In our "Vehicle" example, subclasses can

be differentiated by number of wheels, engine Using the disjoint constraint involves have some careful consideration regarding constraints have use cases. Would make more sense than the adjacent one. Disjoint and overlapping some individuals might be both students and employees who will violate the disjoint condition in these situations, the overlapping constraint that we will cover later and "Employee." end-user allows for some overlap in subclasses. For instance, a super class of "Person" might be divided into "Student" On the other hand, disjoint constraint might be too strict, because in practical applications it may be that the knowledge of the.

**Constraints: Multiple Memberships of a Subclass Overlapping**

Contexts while retaining their identity and the overlapping constraints are added on top of that. Editor, and reviewer all at once, or some combination thereof. This allows for the duplicity of structures to be represented, where an entity may serve multiple roles in different "Author", "Editor" and "Reviewer". One person can be an author, not mutually exclusive and an object can have characteristics of more than one subtype, applying this constraint is necessary. Having a hierarchy of classes makes life easier, e.g. a super class "Person" can be a super class to restricts an entity from belonging simultaneously to more than one subclass within a specialization hierarchy, the overlapping constraint permits this type of multiple membership. When the subclasses represent categories that are unlike the disjoint constraint, which if an entity must belong only one subclass. More complex and nuanced entities. In addition, it enhances data accuracy by retrieving the lost information that would be thrown away instances for all classes; hence the designer of the database must make sure that these differentiating attributes are not mutually exclusive. From a database perspective the overlapping constraint increases the expressiveness of the model by representing example of a "Person" class, subclasses may be distinguished by what sort of publications they write, their role in the publications process, or their areas of expertise. Subclass membership must be able to be the same in some properties a class possesses in an abstract way. For the In order to apply the overlapping constraint, one needs to identify well what the features and being modeled. And relationships. Whether to use disjoint or overlapping constraint depends on the specific application needs and characteristics of the data the queries more complex. This

introduces a challenge of maintaining data consistency as it requires dealing with overlapping attributes the other hand, the overlapping constraint could complicate retrieving and manipulating data. However, when querying the database, you need to verify multiple subclass memberships for any single entity, making on.

**Membership Total Specialization: Required Subclass**

Ambiguous data. Shape can be "not a circle, nor a square, nor a triangle". This completeness is ensured by the total specialization constraint, as to not create unclassified or "Shape" and derived classes called "Circle", "Square", and "Triangle". No that cannot be assigned to a subclass. For example, you could have a base class called belong to at least one subclass. However, this constraint is especially valuable when the subclasses together capture the entire range of meaningful subtypes of the super class there's no instance of the super class Total specialization (also referred to as specialization completeness) is a constraint on a particular specialization hierarchy that mandates every instance of the super class to strict manner that excludes any other possibility of the super class. When the total specialization constraint is in hand, the database remain relationally orphaned. By specifying the subclass with all its differentiating properties in specialization must ensure a complete sub classing of the super class class. The database designer needs to guarantee that the subclasses comprehensively capture all possible sub classifications of the super class such that none of the super-class instances Total data consistency because it avoids the creation of partial, obscure or misleading information. is no need to check for unclassified instances of the super class. It also improves retrieval and manipulation is so much simpler since there specialization will depend on the specific application requirements and the nature of the data being modeled in each case. this case, we will use more appropriate partial specialization which we will see later. The decision between total and partial classify under any of these categories, including books or food items. In might get specialized into "Electronics," "Clothing," "Furniture," etc. Certain products that do not in the real world super class object can exist without being in any subclasses. So, a super class of "Product" But with this case total specialization might not be ideal.

**With Optional Subclass Membership Data Lineage: Forward Engineering**

By the partial specialization constraint, which permits the inclusion of instances that are not well categorized into any of the specified subclasses. Personnel. This flexibility is addressed and "Salesperson." Others may not be managers or salespeople, like administrative staff or technical support there could be instances that cannot be classified into any of the defined subclasses. For instance, think of a super class called "Employee" that can be specialized into subclasses "Manager" require this. This restriction is necessary when there are subclasses that do not account for all possible subtypes of the super class, and While total specialization involves having instances of a super class that fall only within each associated subclass, partial specialization does not true nature. Which in turn adds expressiveness to database designs. It also helps keep data accurate, because it avoids a situation in which an entity gets such a subclass enforced upon it, even if it does not reflect its can allow such subclasses without necessarily covering every possible variation of the class in question. Partial specialization provides an additional degree of implicit bundles, of the database must take into account the fact that the subtypes should not be exhausted: creating new instances can, and should, leave some objects unclassified (objects not admitting any type to which they can belong). This is possible when you it covers. The design Partial specialization is bit of a tricky thing here because it doesn't really tell you all the subclasses of the data being modeled. to deal with unclassified instances, as well as their attributes, with caution. The decision between complete vs partial specialization depends on the particular needs of the application and the characteristics there is an instance of the superclass that has not been classified yet and this can result in expensive queries. This may also lead to complications relating to data consistency since they might need through relationships Due to the fact that you query the DB class by class, you need to check if Create complex queries, Ensure data integrity Specialization Constraints Modeling Real-World Scenarios Using a Combination of Disjoint constraint guaranteeing that a customer is either an Individual or a Corporation, but not both. And "Salesperson" with the total specialization constraint that every employee is either a manager or a salesperson.

For example, the "Customer" subclass can be further specialized into "Individual Customer" and "Corporate Customer," with the constraint permitting individuals to hold the roles of both an employee and a customer. An "Employee" subclass may be specialized into "Manager" the data. As an example, a super class called "Person" could be specialized into "Employee" and "Customer" classes, with the overlapped In practice, database models may involve more complex combinations of specialization constraints to capture a richer representation of in terms of database design, implementation, and maintenance, which need to be carefully considered along with constraints and their effects making sure that the data many be correct and reliable. But it also adds complexities constraints on specialization, one user can interact with a very unhappy end-of-line feeder and avoid the mess that occurs when multiple users try to define a similar schematic of a specialty feeder in a confused manner. The process also improves on data integrity and consistency, By imposing such.

**Practical Examples and Applications**

To further illustrate the concepts discussed in this Module, let's consider a few practical examples and applications of specialization constraints in database modeling. University Database: A university database might have a "Person" super class, specialized into "Student," "Faculty," and "Staff." The overlapping constraint would allow individuals to be both students and employees (e.g., teaching assistants). The "Student" subclass might be further specialized into "Undergraduate Student" and "Graduate Student," with the disjoint constraint ensuring that a student is either an undergraduate or a graduate, but not both. The "Faculty" subclass might be specialized into "Professor," "Associate Professor," and "Assistant Professor," with the total specialization constraint requiring every faculty member to be one of these ranks. E-commerce Database: An e-commerce database might have a "Product" super class, specialized into "Electronics," "Clothing," and "Furniture." The partial specialization constraint would allow for the existence of products that do not fit into any of these categories.

**Implementing Specialization Constraints in SQL**

```
CREATE TABLE Person (
PersonID INT PRIMARY KEY,
```

```
    Name VARCHAR(100),
    DOB DATE
);

CREATE TABLE Student (
PersonID INT PRIMARY KEY,
EnrollmentNoVARCHAR(50),
    FOREIGN KEY (PersonID) REFERENCES Person(PersonID)
);

CREATE TABLE Professor (
PersonID INT PRIMARY KEY,
EmployeeIDVARCHAR(50),
    FOREIGN KEY (PersonID) REFERENCES Person(PersonID)
);

CREATE TABLE Staff (
PersonID INT PRIMARY KEY,
    Department VARCHAR(100),
    FOREIGN KEY (PersonID) REFERENCES Person(PersonID)
);
```

In the above example, specialization constraints are enforced using foreign keys, ensuring that only valid references exist between the super class (Person) and its subclasses (Student, Professor, Staff).

**Enforcing Specialization Constraints Programmatically**

In programming languages like Java, we can implement specialization using inheritance. The following example demonstrates specialization constraints using Java classes.

```
class Person {
int personID;
    String name;
    String dob;

public Person(int id, String name, String dob) {
this.personID = id;
        this.name = name;
this.dob = dob;
    }
```

```
}

class Student extends Person {
   String enrollmentNo;

public Student(int id, String name, String dob, String enrollmentNo) {
super(id, name, dob);
this.enrollmentNo = enrollmentNo;
   }
}

class Professor extends Person {
   String employeeID;

public Professor(int id, String name, String dob, String employeeID) {
super(id, name, dob);
this.employeeID = employeeID;
   }
}

class Staff extends Person {
   String department;

public Staff(int id, String name, String dob, String department) {
super(id, name, dob);
this.department = department;
   }
}
```

Constraints on specialization play a vital role in ensuring data integrity and consistency in databases. By implementing these constraints using SQL and programming constructs, we can enforce business rules effectively. Proper database design using specialization constraints improves maintainability and prevents redundancy, ensuring efficient data management.

# Unit 3.2: Relational Model

### 3.2.1 Relational Model Structure

Data modeling is a crucial step in the database design process that ensures data is structured and stored efficiently. The relational model provides a structured framework that organizes data into tables, or relations, which consist of rows (tulles) and columns (attributes). This model is based on mathematical set theory and provides a logical representation of data, making it easier to manage and retrieve information. A well-designed relational database minimizes redundancy, maintains data integrity, and improves performance. In the relational model, data is structured into relations, where each relation represents a real-world entity. Each relation is defined by a schema, which outlines the attributes and their data types. A key aspect of relational modeling is normalization, a process that reduces redundancy and improves data consistency by organizing data into multiple related tables. Relationships between tables are established using primary and foreign keys, ensuring referential integrity.

Consider the following example of a relational database schema for a university system:

```
CREATE TABLE Students (
StudentID INT PRIMARY KEY,
   Name VARCHAR(100),
   Age INT,
   Major VARCHAR(50)
);

CREATE TABLE Courses (
CourseID INT PRIMARY KEY,
CourseNameVARCHAR(100),
   Credits INT
);

CREATE TABLE Enrollments (
EnrollmentID INT PRIMARY KEY,
StudentID INT,
CourseID INT,
   Semester VARCHAR(20),
```

```
    Grade CHAR(2),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

As a primary key, while the Courses table will use the CourseID as a primary key, and Enrollments will create foreign keys to map students to courses. Courses table will hold course information, and the Enrollments table will create a many-to-many relationship between students and courses. The Students table will use the StudentID In the previous example, the Students table will hold student information.

**Data Abstraction**

Data abstraction; clear and small model of data. The relational model has three levels of Data abstraction simplifies complex data structures so that we can have a Level, The View Level The Tripartite Architecture of Databases; The Physical Level, The Logical it drives home the point that knowing and understanding these levels is fundamental for anyone who works with designing, developing, or administering a database, providing a comprehensive framework that helps you understand how data is managed through a database system. Examines the three layers in database architecture: Physical, Logical and View. And and divide the knotty problems of data management into different levels of abstraction. The post and preservation. For this, databases follow a layered architecture, which means that you can separate Modern databases are complex systems that store and organize large sets of data for convenient access.

**Physical Level: The Foundation of Data Storage The**

Optimization is essential for performance maximization, integrity checking, storage saving and other features. Indexing methods, and storage allocation methods. Physical Level disks, solid state drives, or cloud-based storage. This level deals with the physical arrangement of data, like file formats, level of abstraction is Physical Level or internal level. It addresses the fundamental issues surrounding data storage at the physical level, where data is stored on storage devices like hard.

**Data Storage Techniques:**

use a balanced tree structure to enable efficient searching, insertion, and deletion of records. to manage when collisions happen. One such indexing technique is B-tree indexes, which index regimes and

random access and sequential access regimes. The world of digits: Hashed files: where the data are stored by a hash function function that works as an index and allows random access to the data, but adds a level of complexity in a linear sequence, which is ideal for batch processing but not for random access. Indexed sequential file systems are more sophisticated than sequential file systems because they include both in different types of structures and each has its own advantages and disadvantages. Sequential files, for instance, keep records Physical Level — relates to how data is stored in disk files can use sequential files. files or B-tree indexes may be more useful for a database that needs many random accesses to specific records. On the other hand, a database that does mostly batch processing performance requirements. As an example, hashed by the application for the database. The choice of appropriate storage structures depends on various factors, including access patterns, data volume, and the type of storage technique varies.

**Indexing Techniques**

System (DBMS) can access individual records swiftly from the indexed values. Storage going to maintaining the index data structure. The indexes are usually created over one or multiple columns of the table so the Database Management retrieval operations on a database table. It does this with the price of additional writes and data can be retrieved. An index is a data structure that enhances the speed of data Indexing is a key operation in the Physical Level as it determines the speed at which the while they provide speed for equality searches, those are just as limited in supporting ordering queries as B-trees are for non-equalities. Commonly used for primary and secondary indexes, allowing for efficient search capabilities. On the flip side of the coin, hash indexes employ a hash function to direct index values to certain storage locations, so physical location of the records. B-tree indexes are of the database table Non-clustering secondary indexes maintain a separate index structure, with pointers to the Example of clustered (primary) indexing it sets the records on the physical order possible to reduce the query execution time considerably and improve database performance significantly. Could use hash indexes. By creating meaningful indexes, it is of range queries may want B-tree indexes. On the other hand, a data base performing mostly equality searches the insert performance slightly (the performance penalty is

compensated with better read performance, no issues there!). As a case in point, a database that performs a lot this allows for lightning-fast read operations, but will impact.

**Allocation Strategies Storage**

Providing fast sequential and random access. Down sequential access. Indexed allocation improves on both contiguous and linked allocation by using an index to keep track of pointers for non-contiguous blocks, file is allocated a contiguous block of storage space, allowing for fast sequential access but raising the risk of external fragmentation. Linked allocation is another method where storage is allocated in non-contiguous blocks that are linked in the form of pointers, reducing the extent of external fragmentation but slowing allocation strategies that control how much storage space is allocated to a database file or an index. Contiguous allocation means that each there are a few storage storage organizations is essential to optimizing storage consumption and avoiding I/O overhead. a database that does sequential access most of the time may choose to go with contiguous allocation. Effective deleting entries, which could potentially be supported with the use of linked or indexed allocation. On the flip side, resources available will help determine your storage allocation method An example of this is a database that is constantly inserting or Different workloads and storage.

**Compression and Encryption Diversity of Data: Data**

RSA. Transforming that into unreadable format. They encrypt data using encryption algorithms such as AES and Huffman coding and Lempel-Ziv. Whereas data encryption helps prevent unauthorized access to sensitive data by space, which significantly minimizes storage and improves I/O performance. Some examples of listlessly compression algorithms are as data compression and data encryption are used. Subsequently, Compression is the process of taking up less At the Physical Level, techniques such to prevent unauthorized usage of data and adhere to regulatory data encryption. At the Physical Level, the application of relevant security elements is imperative high volume of text data. In contrast, a database capturing sensitive financial or personal data may need and sensitivity of data. For instance, data compression may be used by a database that contains a Compression and encryption is generally used depending on the needs of the database application obligations.

**Hardware Considerations**

Bandwidth: Bandwidth will impact on how fast data can move from the database server to client applications, which will affect the overall responsiveness of the database system. Mirroring and redundancy in case of storage collapse, so data is not lost if there is hardware failure. Database network input and output (I/O) tremendously compared to a hard disk drive (HDD) and thus, enhancing the overall performance of database operations. Redundant array of independent disks (RAID) provide data Physical Level, which are type of storage devices used, speed of input/output operations performed, and networking bandwidth. The introduction of solid-state drives or SSDs increases the hardware factors also affect the the foundation for improving the performance and availability of database system. Performance and scalability needs. The high efficiency of hardware resource allocation is database system to achieve optimum performance and reliability. Hardware configurations depend on factors like budget, It is significant to select the hardware components correctly for the.

**Logical Level: Specifying Data Structure and Interrelations**

Mechanism. Data modeling. This layer abstracts away the physical details of the storage, allowing users to work directly on data while hiding the storage level provides a high-level description of the data stored in a database. It involves the relationship of various data elements, data integrity rules, and The logical level also called the conceptual.

**Data Models**

Graph models, to support a wide range of data structures and access patterns. Data and behavior into a single logical unit called an object. First, we generally see two distinct database types in use today relational databases and NoSQL databases. NoSQL databases use different data models, such as document-oriented, key-value, and general presentation of the data as entities and relationships using diagrams. An object-oriented data model encapsulates columns. In contrast to the table of contents and entity relationship (ER) model, which encompasses the Logical Level, where each data model has its own concepts and notations. One example is the relational data model that defines data as a set of tables with rows and We use different data models like these represent user data at the will be used to store the logical structure. the choice can be a NoSQL database. The data

model must be SM because it database needs to record complex relationships between data elements it may be useful to employ the relational or ER model. On the other hand, if a database contains mainly semi-structured or unstructured data, then database systems. As an example, if a There are different data models used in.

**Definition Language (DDL): Data**

Is utilized for table deletion. Columns. The DROP TABLE statement defines the columns in the new table along with their data types. The ALTER TABLE statement is generally used to modify an existing table and adding or deleting specified here. For instance, in SQL, a new table can be created using the CREATE TABLE statement, which commands to create, update and delete database objects like tables, indexes and views. Data types, constrains, and relationships between different data elements are The Logical Level The Logical Level is specified with a Data Definition Language (DDL), defining of the application commands also plays an important role in maintaining the integrity and consistency of the database schema. Application data designing a database schema to support the data management needs and manage logical structures in the database. Proper and optimal use of DDL The DBMS executes DDL statements to create.

**Data Relationships and Constraints:**

The Logical Level defines the relationships between different data elements, ensuring data integrity and consistency. Relationships are typically represented using foreign keys, which establish links between tables. Constraints, such as primary key constraints, foreign key constraints, and check constraints, are used to enforce data integrity rules. Primary key constraints ensure that each record in a table is uniquely identified. Foreign key constraints ensure that relationships between tables are maintained. Check constraints enforce specific conditions on the values of data elements. The definition of appropriate relationships and constraints is crucial for maintaining data integrity and ensuring the consistency of the database. For example, a foreign key constraint can ensure that a customer's order is associated with a valid customer record. Check constraints can ensure that the values of data elements fall within a specific range or satisfy certain conditions. The enforcement of data integrity rules is essential for preventing data inconsistencies and

ensuring the accuracy of the database. By separating these levels, the relational model ensures that changes at one level do not impact the others, making database management more flexible and scalable.

**Instances and Schemas**

An **instance** refers to the actual content stored in a database at a specific point in time, while a **schema** defines the structure and constraints of the database. A schema acts as a blueprint, dictating the organization of data. For example, in the university system above, the schema specifies the tables, attributes, and relationships, while the instance consists of the actual student records, courses, and enrollments stored in the database at a given moment.

INSERT INTO Students (StudentID, Name, Age, Major) VALUES (1, 'Alice Johnson', 20, 'Computer Science');
INSERT INTO Students (StudentID, Name, Age, Major) VALUES (2, 'Bob Smith', 22, 'Mathematics');

INSERT INTO Courses (CourseID, CourseName, Credits) VALUES (101, 'Database Systems', 4);
INSERT INTO Courses (CourseID, CourseName, Credits) VALUES (102, 'Operating Systems', 3);

INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID, Semester, Grade) VALUES (1, 1, 101, 'Fall 2024', 'A');
INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID, Semester, Grade) VALUES (2, 2, 102, 'Fall 2024', 'B');

Here, the schema remains constant, but the instance changes as new students enroll in courses.

**Data Models**

Data models define how data is structured, stored, and manipulated within a database system. The relational model is one of the most widely used data models, but others include hierarchical, network, and object-oriented models. The relational model's simplicity, scalability, and strong theoretical foundation make it a preferred choice for most applications. A relational database management system (RDBMS) such as MySQL, PostgreSQL, or SQL Server implements the relational model, enabling efficient data management and retrieval. SQL (Structured Query Language) is used to interact with relational

databases, performing operations such as data insertion, retrieval, and modification.

```
SELECT Students.Name, Courses.CourseName, Enrollments.Grade
FROM Students
JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
JOIN Courses ON Enrollments.CourseID = Courses.CourseID
WHERE Enrollments.Semester = 'Fall 2024';
```

This query retrieves student names, enrolled courses, and grades for the fall 2024 semester by joining the relevant tables.

Understanding the relational model structure is essential for designing efficient and scalable databases. By applying data modeling techniques, utilizing schema definitions, and enforcing data integrity through relationships and constraints, organizations can build robust database systems. The relational model's abstraction levels, schema-instance distinction, and powerful query capabilities make it a dominant paradigm in database design. Through practical examples and SQL implementation, developers and database administrators can leverage the relational model to manage complex datasets effectively.

### 3.3.2 Database Schema

Database schema is a fundamental component of database design that defines the structure and organization of data within a database. It serves as a blueprint for how data is stored, accessed, and managed, ensuring consistency and efficiency. A database schema encompasses tables, relationships, constraints, indexes, and other structural elements that dictate how information is organized.

**Understanding Database Schema**

A database schema represents the logical configuration of a database. It defines tables, columns, data types, and relationships between tables, ensuring a structured approach to data storage. A well-designed schema improves database performance, scalability, and maintainability.

**Types of Database Schema**

1. **Physical Schema**: Defines how data is physically stored on storage devices, including indexing, partitioning, and clustering strategies.
2. **Logical Schema**: Represents the logical structure of data, including tables, relationships, views, and constraints.

3. **Conceptual Schema**: Offers a high-level view of data organization, abstracting technical details.

**Importance of Database Schema**

A properly designed database schema ensures data integrity, eliminates redundancy, and facilitates efficient data retrieval. Without a well-structured schema, databases can become inefficient and difficult to manage.

**Creating a Database Schema with SQL**

Below is an example of an SQL script to create a simple database schema for an e-commerce application:

```
CREATE DATABASE ECommerceDB;
USE ECommerceDB;

CREATE TABLE Customers (
CustomerID INT PRIMARY KEY AUTO_INCREMENT,
   Name VARCHAR(100) NOT NULL,
   Email VARCHAR(100) UNIQUE NOT NULL,
   Phone VARCHAR(15),
   Address TEXT
);

CREATE TABLE Products (
ProductID INT PRIMARY KEY AUTO_INCREMENT,
   ProductName VARCHAR(100) NOT NULL,
   Category VARCHAR(50),
   Price DECIMAL(10,2) NOT NULL,
   Stock INT NOT NULL
);

CREATE TABLE Orders (
OrderID INT PRIMARY KEY AUTO_INCREMENT,
CustomerID INT,
OrderDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
TotalAmountDECIMAL(10,2) NOT NULL,
   FOREIGN KEY (CustomerID) REFERENCES
Customers(CustomerID)
);
```

```
CREATE TABLE OrderDetails (
OrderDetailID INT PRIMARY KEY AUTO_INCREMENT,
OrderID INT,
ProductID INT,
   Quantity INT NOT NULL,
   Price DECIMAL(10,2) NOT NULL,
   FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

**Normalization and Schema Optimization**

Database normalization is the process of structuring a database to reduce redundancy and improve integrity. The key normal forms are:

- **1st Normal Form (1NF)**: Eliminates duplicate columns from tables and creates separate tables for related data.
- **2nd Normal Form (2NF)**: Ensures that all non-key attributes are fully functionally dependent on the primary key.
- **3rd Normal Form (3NF)**: Removes transitive dependencies.

**Schema Design Best Practices**

1. **Use meaningful table and column names to** improvereadability and maintainability.
2. **Normalize the database** to eliminate redundancy and ensure consistency.
3. **Index frequently accessed columns** to optimize query performance.
4. **Use foreign keys and constraints** to enforce referential integrity.
5. **Plan for scalability** by considering partitioning and clustering strategies.

Database schema design plays a crucial role in the effectiveness of a database system. A well-structured schema ensures data consistency, enhances performance, and simplifies database management. By following best practices and normalization techniques, database designers can create efficient and scalable database structures that meet business requirements. This Module has provided a comprehensive overview of database schema, its types, importance, and best practices. By understanding the principles of database design, developers can build robust and efficient database systems for various applications.

# Unit 3.3: Concept of Keys in Database System

## 3.3.1 Keys in Database Systems

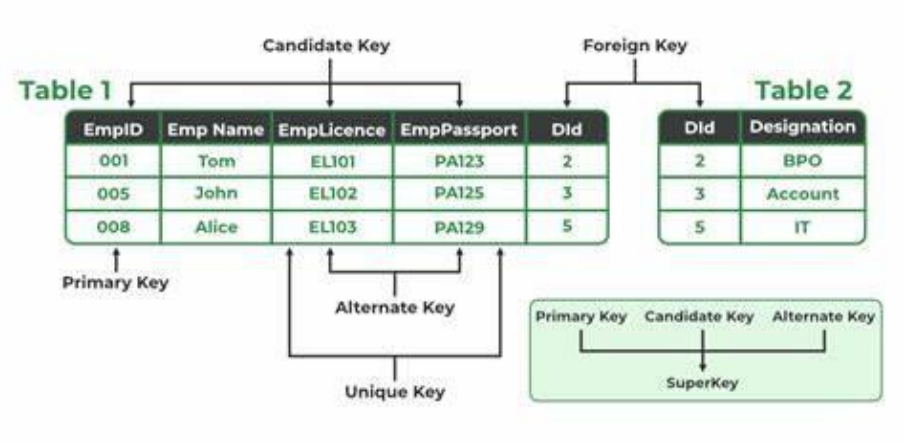



*Figure 3.3.1: Keys in Database System*
*[Source: https://th.bing.com/]*

**1 Super Key**

A super key is a set of one or more attributes that uniquely identify a record in a table. It is a superset of a candidate key and ensures that no two rows have the same combination of values for the attributes included in the super key. Super keys may include extra attributes that are not necessary for uniqueness but still qualify as unique identifiers.

**Example in SQL:**

```
CREATE TABLE Students (
StudentID INT,
   Email VARCHAR(255),
   Name VARCHAR(255),
   Course VARCHAR(100),
   PRIMARY KEY (StudentID, Email) -- This combination acts as a Super Key
);
```

In the above example, both StudentID and Email together form a super key since they uniquely identify a student record.

2. **Candidate Key**

A candidate key is a minimal super key, meaning it contains only the necessary attributes required for unique identification. If an attribute is removed from a candidate key, it no longer remains a key.

**Example in SQL:**

```
CREATE TABLE Employees (
EmployeeID INT UNIQUE,
NationalIDVARCHAR(20) UNIQUE,
   Name VARCHAR(255),
   Department VARCHAR(100)
);
```

Here, both EmployeeID and NationalID are candidate keys because either of them can uniquely identify an employee.

**3. Primary Key**

A primary key is a specific candidate key chosen to uniquely identify each record in a table. It must be unique and not null.

**Example in SQL:**

```
CREATE TABLE Orders (
OrderID INT PRIMARY KEY,
CustomerID INT,
OrderDate DATE
);
```

OrderID serves as the primary key, ensuring uniqueness across all order records.

**4. Foreign Key**

A foreign key is an attribute in one table that establishes a link to the primary key of another table, enforcing referential integrity.

**Example in SQL:**

```
CREATE TABLE Customers (
CustomerID INT PRIMARY KEY,
   Name VARCHAR(255),
   Email VARCHAR(255)
);

CREATE TABLE Orders (
OrderID INT PRIMARY KEY,
CustomerID INT,
OrderDate DATE,
   FOREIGN KEY (CustomerID) REFERENCES
Customers(CustomerID)
);
```

In this example, CustomerID in the Orders table acts as a foreign key referencing the CustomerID in the Customers table.

Understanding keys is fundamental to database design. Super keys define uniqueness, candidate keys refine them, primary keys enforce uniqueness, and foreign keys ensure referential integrity. Implementing these properly ensures database efficiency, consistency, and data integrity.

**3.Schema Diagram**

Data modeling is a fundamental process in database design that involves defining and structuring data elements and their relationships. The primary goal of data modeling is to create an efficient, scalable, and maintainable database structure that meets business and application needs. It provides a conceptual representation of data and ensures consistency, integrity, and optimization. A schema diagram is a visual representation of this model, illustrating tables, attributes, relationships, and constraints.

**Data Abstraction**

Data abstraction refers to the process of simplifying complex data structures by defining different levels of abstraction. It helps in managing data by providing a systematic way to interact with it without needing to understand the underlying complexities. The three levels of data abstraction are:

1. **Physical Level:** The lowest level that describes how data is actually stored on storage media, such as disks and memory.
2. **Logical Level:** The intermediate level that defines what data is stored in the database and the relationships among them.
3. **View Level:** The highest level that provides different ways to interact with the data, customized for different users.

**Instances and Schemas**

A database schema is the overall design and structure of the database, whereas an instance is a snapshot of data stored in the database at a particular moment in time. The schema remains unchanged, whereas instances change over time as data is inserted, updated, or deleted.

**Example Schema Diagram (SQL-Based)**

```
CREATE TABLE Customers (
CustomerID INT PRIMARY KEY,
   Name VARCHAR(100),
   Email VARCHAR(255),
```

```
    Phone VARCHAR(20)
);

CREATE TABLE Orders (
OrderID INT PRIMARY KEY,
CustomerID INT,
OrderDate DATE,
TotalAmountDECIMAL(10,2),
    FOREIGN KEY (CustomerID) REFERENCES
Customers(CustomerID)
);
```

This schema defines two tables, Customers and Orders, with a one-to-many relationship, where each customer can place multiple orders.

**Data Models**

A data model defines how data is organized and manipulated in a database. There are several types of data models:

The evolution of database management systems (DBMS) has been marked by the development of various data models, each designed to address specific requirements and challenges in data organization and retrieval. These models provide the conceptual framework for structuring data, defining relationships, and enforcing constraints, ultimately influencing the efficiency and effectiveness of database applications. This Module delves into the four fundamental database models: Hierarchical, Network, Relational, and Object-Oriented, examining their structures, strengths, weaknesses, and historical significance.

**1. Hierarchical Model: The Tree-Structured Legacy**

it intuitive for some sorts of application. can have multiple children records, however a child record can have only one parent. This rigid framework reflects the hierarchy of companies and files systems, which makes and potentially multiple children. This model serves a one-to-many relationship, where in such relation necessary parent 1- Hierarchical Model The Hierarchical model is one of the oldest database models, arranging records in a tree-like structure in which each record has a single parent (unless it is the root) an Organizational schema, with the CEO at the root and employees below them at level 1, level 2, etc. hierarchical data structures and is particularly useful in the context of applications where data is efficiently captured in

hierarchical relationships. A Hierarchical model is a great fit to represent of these two structures using pointers is what makes the data in the node called fields. The model is capable of handling highest-level node is referred to as the root, and all other nodes are descendants of the root. The combination the primary object is a tree consisting of nodes, where a node represents a record, and branches represent the relationship between records. The In the Hierarchical model data reads with this setup. to navigate from root to the desired node in the tree. Applications that need to access data lower in the hierarchy frequently could see inefficient student is in multiple courses, then there has to be a full record for a student for each course, which causes data redundancy. Also, this navigation of the model is challenging, as one might need often needed resulting in inefficiency and possible inconsistency For instance, if a real-world data. Data Redundancy: To model such relationships, data redundancy is limitations. Its inflexible structure makes it impossible to represent many to many relationships common to Nevertheless, the Hierarchical model has certain used Maintain field type consistency, etc. may not be a desired behavior. A database allows constraints to be placed on the data to enforce data integrity, i.e., each child record must have a parent record, the data type of the field is important to consider the parent-child relationships because if you change something at one spot of the tree you might break something at a totally different and unrelated tree path. For example, deleting a parent record will also delete its children records, this Hierarchical model, data manipulation involves following the predefined paths in the tree when adding, deleting, or updating records it In the properties. Hierarchical model has generally been replaced. However, remnants can still be found in some domains like file systems and XML data structures with hierarchical mainframe environments for transaction processing and data management. Nevertheless, since more flexible and powerful data models became available, the Information Management System (IMS), one of the first DBMS, had a Hierarchical model. IMS was prevalent in quite relevant. IBM's Sure enough, the Hierarchical model is historically The creation of advanced data models that overcome such challenges. Decline as an approach. Its legacy, though, has opened the door for While the Hierarchical model is simple and efficient for representing a one-to-

many relationship, it is less suited to handling a many-to-many relationship and complex navigation, which ultimately led to its.

**Model: Understanding Complex Relationships Network**

Which allows more complex relationships between data to be represented. Model an extension of the hierarchical model; it overcomes the limitations of its predecessor by allowing a record to have multiple parent records, representing many-to-many relationships. In this model, the data model is built up as a graph, with records as nodes and relationships as links, Network to more than one project. Member) therefore it may have multiple parent records, unlike the Hierarchical model. The schema allows for flexibility, representing complex relationships, such as students taking more than one course or employee's assigned record per owner and one or more records per member. A member record may belong to multiple sets (is a record relationships. A set contains one and groups). Sets of records show one-to-many The central element is the network itself (formed by records which may not be the desired behavior in all cases. larger network of information. As an example, when an owner record gets deleted, all member records get deleted automatically, can be more than one parent record makes navigating more difficult. As such, when you want to create, delete or edit records you need to think about the set relationships as one small piece of a traversing the network using predefined paths just as in the Hierarchical model. But the fact that there In the arrows from the network model, the manipulation of data occurs through department. The instruction created are membership in the set and how to be inside. That is, a student only gets to register for courses that are part of their has an owner record, and that field data types are consistent. Also it is possible to define some constraint to be applied in it, Constraints are used to maintain data integrity in the Network model, for example by ensuring that every member record also its obsolescence. its impact is still felt in some contexts, such as network databases and graph databases, which employ graph structures to model relationships between data. Made the use of Network models less common. However, However, this flexibility comes at a cost, as the Model is complex and the emergence of the Relational model has

**3. Relational Model: The Power of Tables**

The Relational model, introduced by E.F. Codd in 1970, revolutionized database management by representing data in tables with rows and columns. This model provides a simple and intuitive way to organize data, define relationships, and enforce constraints, making it the dominant database model in modern applications. The core component of the Relational model is the relation, which is represented as a table. Each table consists of rows and columns. Rows represent records, and columns represent attributes. The Relational model uses primary keys to uniquely identify records and foreign keys to establish relationships between tables. This approach eliminates data redundancy and ensures data integrity. Data manipulation in the Relational model is performed using Structured Query Language (SQL), a powerful and flexible language for querying and manipulating data. SQL allows users to perform various operations, such as selecting, inserting, updating, and deleting data. The Relational model's simplicity and the availability of SQL have made it accessible to a wide range of users, from database administrators to application developers. Data integrity in the Relational model is maintained through constraints, such as primary key constraints, foreign key constraints, and check constraints. Primary key constraints ensure that each record is uniquely identified. Foreign key constraints ensure that relationships between tables are valid. Check constraints ensure that data values meet specific criteria. The Relational model also supports referential integrity, which ensures that relationships between tables are consistent. The Relational model's historical significance is immense. IBM's System R, one of the earliest relational DBMS, demonstrated the feasibility and power of the Relational model. Oracle, Microsoft SQL Server, and MySQL are examples of widely used relational DBMS. The Relational model's simplicity, flexibility, and the availability of SQL have made it the dominant database model in modern applications, ranging from enterprise systems to web applications. The Relational model's ability to represent data in a simple and intuitive way, its support for SQL, and its emphasis on data integrity have made it the foundation of modern database management. Its influence is pervasive, and it continues to evolve to meet the changing needs of database applications.

**4. Object-Oriented Model: Bridging the Gap**

The Object-Oriented model, an attempt to bridge the gap between object-oriented programming (OOP) and database management, integrates OOP principles into database design. This model represents data as objects, which encapsulate data and behavior, providing a more natural and intuitive way to model real-world entities. The core component of the Object-Oriented model is the object, which consists of attributes (data) and methods (behavior). Objects are organized into classes, which define the structure and behavior of objects. The Object-Oriented model supports inheritance, encapsulation, and polymorphism, which are fundamental principles of OOP. These principles enable the creation of complex and reusable data structures. Data manipulation in the Object-Oriented model involves invoking methods on objects. This approach provides a more natural and intuitive way to interact with data, as it mirrors the way objects interact in OOP. The Object-Oriented model also supports object-oriented query languages, which provide a more expressive and powerful way to query and manipulate data. Data integrity in the Object-Oriented model is maintained through encapsulation and constraints. Encapsulation ensures that data is accessed and modified only through methods, providing a level of control over data access. Constraints, such as primary key constraints and foreign key constraints, ensure that data values meet specific criteria and that relationships between objects are valid.

**Example of a Relational Data Model**

```
CREATE TABLE Products (
ProductID INT PRIMARY KEY,
   ProductName VARCHAR(255),
   Price DECIMAL(10,2)
);

CREATE TABLE OrderDetails (
OrderDetailID INT PRIMARY KEY,
OrderID INT,
ProductID INT,
   Quantity INT,
   FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

In this relational model, the Products table stores product details, and Order Details links orders to products, implementing a many-to-many relationship. Understanding the view of data through abstraction, schema diagrams, and data models is crucial in database design. Proper structuring ensures efficient data retrieval, integrity, and scalability, making databases more robust and reliable.

### 3.3.2 Conversion of E-R to Relational Model

Data modeling and database design are crucial aspects of database management systems (DBMS). The process involves structuring and organizing data systematically to facilitate efficient storage, retrieval, and modification. One of the fundamental techniques in database design is the Entity-Relationship (E-R) model, which visually represents entities, their attributes, and relationships. Converting an E-R model into a relational model is a vital step in implementing a functional database.

**Understanding the E-R Model**

The Entity-Relationship model is a conceptual framework that represents the structure of data. It comprises entities, attributes, and relationships. Entities are objects or concepts that store data, while attributes define the properties of entities. Relationships describe the associations between entities.

For example, consider a university database where Student, Course, and Professor are entities. A student enrolls in a course, and a professor teaches a course. These relationships help define the logical connections in the system.

```
CREATE TABLE Student (
StudentID INT PRIMARY KEY,
   Name VARCHAR(100),
   Age INT,
   Major VARCHAR(50)
);

CREATE TABLE Course (
CourseID INT PRIMARY KEY,
CourseNameVARCHAR(100),
   Credits INT
);
```

```
CREATE TABLE Professor (
ProfessorID INT PRIMARY KEY,
   Name VARCHAR(100),
   Department VARCHAR(50)
);
```

**Step 1: Mapping Entities**

Each entity in the E-R diagram is mapped to a table in the relational model. The primary key of the entity becomes the primary key of the table.

For example:

```
CREATE TABLE Student (
StudentID INT PRIMARY KEY,
   Name VARCHAR(100),
   Age INT,
   Major VARCHAR(50)
);
```

**Step 2: Mapping Relationships**

Relationships between entities are converted into foreign keys or separate tables, depending on the cardinality of the relationship.

For a Many-to-Many relationship like Student and Course, a junction table is created:

```
CREATE TABLE Enrollment (
StudentID INT,
CourseID INT,
EnrollmentDate DATE,
   PRIMARY KEY (StudentID, CourseID),
   FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

**Step 3: Mapping Attributes**

Attributes are represented as column

## Summary of Roles

All the above topics together build a strong foundation for relational database design by addressing abstraction, structural representation, and the mechanisms for uniquely identifying and managing data.

**Summary**

The first topic introduces generalization and specialization, which are key concepts in refining database design. Generalization abstracts common features from multiple entities into a higher-level entity, while specialization defines subgroups of entities with distinct characteristics. These techniques enhance flexibility and accuracy in modeling real-world scenarios.

The relational model is then presented as the most widely used data model in database systems. It organizes data into tables (relations) consisting of rows (tuples) and columns (attributes). With its simple structure, mathematical foundation, and strong query capabilities, the relational model ensures data consistency, logical clarity, and ease of use.

The concept of keys plays a vital role in relational databases. Keys such as primary, candidate, foreign, and composite keys uniquely identify records, enforce relationships between tables, and maintain data integrity. Proper use of keys ensures that data remains consistent, unambiguous, and reliable across the system.

Together, these topics provide the theoretical and practical basis for relational database design. By applying abstraction techniques, understanding the relational structure, and effectively using keys, learners gain the ability to design robust and efficient databases that meet organizational needs.

**MCQs:**

1. **Generalization in a database is the process of:**
   a) Combining multiple entities into a higher-level entity
   b) Splitting one entity into multiple sub-entities
   c) Creating foreign keys
   d) Deleting redundant data
   (Answer-a)
2. **Specialization in an E-R model refers to:**
   a) Merging two entities into one
   b) Creating sub-entities from a higher-level entity
   c) Removing attributes from a table
   d) Encrypting a database
   (Answer-b)

3. **A Super Key is:**
   a) A key that uniquely identifies a tuple but may have extra attributes
   b) A key used for indexing
   c) A key with duplicate values
   d) A key used only for foreign relations
   (Answer-a)
4. **Which of the following is a Candidate Key?**
   a) A key that can be used as a Primary Key
   b) A key that contains duplicate values
   c) A foreign key
   d) A key that cannot be unique
   (Answer-a)
5. **The Primary Key in a relational database:**
   a) Uniquely identifies each record
   b) Can have NULL values
   c) Is always a foreign key
   d) Must contain duplicate values
   (Answer-a)
6. **A Foreign Key is used to:**
   a) Uniquely identify a record in a table
   b) Enforce referential integrity between two tables
   c) Store encrypted data
   d) Improve query performance
   (Answer-b)
7. **Which diagram is used to represent the structure of a relational database?**
   a) Flowchart
   b) Schema Diagram
   c) E-R Diagram
   d) UML Diagram
   (Answer-b)
8. **What does E-R to Relational Model Conversion involve?**
   a) Mapping entities and relationships to tables
   b) Writing SQL queries
   c) Creating indexes for tables
   d) Deleting duplicate records
   (Answer-a)

9. **Which of the following constraints ensures referential integrity in a database?**
   a) Primary Key
   b) Foreign Key
   c) NOT NULL
   d) CHECK
   (Answer-b)
10. **The Relational Model consists of:**
   a) Tables with rows and columns
   b) Images and videos
   c) Hierarchical data storage
   d) Graph-based relationships
   (Answer-a)

**Short Questions:**

1. What is the difference between Generalization and Specialization?
2. Define Super Key, Candidate Key, and Primary Key.
3. Explain the Relational Model Structure in databases.
4. What are the different types of keys in a relational database?
5. How does a Foreign Key maintain referential integrity?
6. What are the constraints on Specialization in E-R models?
7. Explain how an E-R model is converted into a relational model.
8. What is the role of a Schema Diagram in database design?
9. Define Database Schema and its types.
10. What is the importance of constraints in relational databases?

**Long Questions:**

1. Explain Generalization and Specialization in the E-R model with examples.
2. Discuss the role of constraints on Specialization in database design.
3. What is a Relational Model? Explain its structure with examples.
4. Describe the different types of keys and their importance in a relational database.
5. Explain the concept of Foreign Keys and how they enforce referential integrity.

6. What is a Schema Diagram, and how does it help in database design?
7. Describe the process of converting an E-R model into a relational model.
8. Explain the importance of normalization in relational databases.
9. Compare and contrast Primary Key and Foreign Key.
10. How does a well-designed relational model improve database efficiency?

# MODULE 4
# MANAGING DATABASE AND TABLE

## 4.0 LEARNING OUTCOMES

- Learn how to create, select, and drop databases in SQL.
- Understand how to create, rename, alter, truncate, and drop tables.
- Learn about different data types in a relational database.
- Understand how to insert, update, and delete records from a table.
- Learn about constraints such as Primary Key, Foreign Key, UNIQUE, NOT NULL, DEFAULT, and CHECK constraints.

**Data Modeling and Database Design**

Data modeling is a fundamental process in database design that involves structuring and organizing data to facilitate efficient storage, retrieval, and manipulation. This process ensures that databases are designed to support business requirements, data integrity, and scalability. There are several approaches to data modeling, including conceptual, logical, and physical models. Each stage refines the representation of data from abstract concepts to implementable structures. Conceptual models focus on high-level entity relationships and do not include technical details. Logical models define data attributes, relationships, and constraints, preparing the data for implementation. Physical models translate these structures into database-specific schemas, incorporating storage considerations and indexing strategies. Effective data modeling enables organizations to maintain data consistency, optimize performance, and streamline application development.

### 4.4.1 Select, Create, and Drop Database

The SELECT statement is a fundamental SQL command used to retrieve data from a database. It allows users to specify columns, apply conditions, aggregate results, and join tables. Efficient selection of data is critical for database performance, as it affects query execution time and resource utilization.

**Example:**

```
SELECT * FROM employees;
SELECT name, salary FROM employees WHERE department = 'HR';
SELECT department, AVG(salary) FROM employees GROUP BY department;
```

Indexes and query optimization techniques enhance the efficiency of the SELECT statement. Proper indexing ensures that searches and retrievals are performed quickly, reducing the overall database load.

**Create Database**

Creating a database is the first step in database design. The CREATE DATABASE command initializes a new database in a relational database management system (RDBMS). It defines the storage structure, character set, and collation settings.

**Example:**

```
CREATE DATABASE CompanyDB;
USE CompanyDB;
```

After creating a database, tables and relationships must be defined. Proper planning ensures that the schema supports data consistency and business rules.

**Drop Database**

Dropping a database permanently deletes all its data and structures. This operation should be executed with caution, as it cannot be undone.

**Example:**

```
DROP DATABASE CompanyDB;
```

Before dropping a database, it is advisable to take a backup to prevent accidental data loss. Database administrators should also ensure that no active transactions depend on the database being dropped.

Understanding data modeling and database design is essential for developing efficient, scalable, and maintainable database systems.

The SELECT, CREATE DATABASE, and DROP DATABASE commands are fundamental operations that enable database interaction, management, and maintenance. Proper implementation of these concepts ensures data integrity, security, and optimal performance.

### 4.4.2 Create, Rename, Alter, Truncate, and Drop Table

Data modeling and database design are fundamental aspects of database management systems (DBMS). These processes define how data is structured, stored, and manipulated efficiently. Effective database design ensures data integrity, reduces redundancy, and optimizes performance. Structured Query Language (SQL) provides various commands for managing tables, including creating, renaming, altering, truncating, and dropping tables. This Module delves into these operations with detailed explanations and examples to provide a deep understanding of database management.

**Creating a Table**

Creating a table is the first step in designing a database. The CREATE TABLE statement in SQL allows the definition of a table structure, including columns, data types, constraints, and relationships.

**Syntax:**

```
CREATE TABLE table_name (
column1 datatype constraints,
column2 datatype constraints,
   ...
columnN datatype constraints
);
```

**Example:**

```
CREATE TABLE Students (
StudentID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   Age INT,
   Email VARCHAR(100) UNIQUE
);
```

This example defines a Students table with four attributes. The StudentID column serves as the primary key, ensuring uniqueness. The Email field is constrained to store unique values, preventing duplicate entries.

**Renaming a Table**

The RENAME statement is used to change the name of an existing table. This is useful when restructuring the database or standardizing naming conventions.

**Syntax:**

ALTER TABLE old_table_name RENAME TO new_table_name;

**Example:**

ALTER TABLE Students RENAME TO UniversityStudents;

This command renames the Students table to UniversityStudents, ensuring better clarity and consistency in the database.

**Altering a Table**

The ALTER TABLE statement modifies an existing table by adding, modifying, or dropping columns. This is essential when evolving a database to accommodate new requirements.

**Syntax (Adding a Column):**

ALTER TABLE table_name ADD column_name datatype constraints;

**Example:**

ALTER TABLE Students ADD DateOfBirth DATE;

This statement adds a DateOfBirth column to the Students table.

**Syntax (Modifying a Column):**

ALTER TABLE table_name MODIFY column_namenew_datatype constraints;

**Example:**

ALTER TABLE Students MODIFY Age SMALLINT;

This modifies the Age column's data type from INT to SMALLINT for optimized storage.

**Syntax (Dropping a Column):**

ALTER TABLE table_name DROP COLUMN column_name;

**Example:**

ALTER TABLE Students DROP COLUMN Email;

This command removes the Email column from the Students table.

**Truncating a Table**

The TRUNCATE statement is used to delete all records from a table without removing the structure. This is faster than DELETE as it bypasses logging mechanisms.

**Syntax:**

TRUNCATE TABLE table_name;

**Example:**

TRUNCATE TABLE Students;

This deletes all records from the Students table while retaining the table structure.

**Dropping a Table**

The DROP TABLE statement permanently deletes a table and all associated data.

**Syntax:**

DROP TABLE table_name;

**Example:**

DROP TABLE Students;

This removes the Students table from the database.

Understanding data modeling and database design is crucial for efficient data management. SQL provides powerful commands to create, rename, alter, truncate, and drop tables, allowing developers to structure and modify databases effectively. Proper database design ensures performance optimization and data integrity, supporting robust application development. The ability to manipulate tables dynamically ensures that a database remains scalable and adaptable to evolving business needs.

**Data Modeling and Database Design**

Data modeling and database design are foundational aspects of database systems that involve structuring and organizing data efficiently to ensure optimal performance, integrity, and usability. These processes define how data is stored, retrieved, and managed in a database. Data modeling involves creating abstract representations of real-world entities and their relationships, while database design focuses on the practical implementation of these models using database management systems (DBMS). A well-structured database design ensures consistency, minimizes redundancy, and enhances data retrieval speed. Various data models, such as hierarchical, network, relational, and object-oriented, dictate how data is organized and related. The relational model, which uses tables, is the most widely used due to its flexibility and efficiency in handling structured data.

# Unit 4.2: Data Types in DBMS

## 4.2.1 Data Types: Bit, Boolean, Char, Archer, Date, Date time, Decimal

Different database management systems provide a variety of data types to store and manipulate data effectively. Choosing the appropriate data type is crucial for optimizing storage, ensuring data integrity, and improving query performance. The following are some commonly used data types in databases:

1. **BIT Data Type**The BIT data type is used to store binary values (0 or 1). It is commonly used for Boolean operations and binary flags.
2. CREATE TABLE UserPreferences (
3. id INT PRIMARY KEY,
4. receive_notifications BIT
5. );
6. **BOOLEAN Data Type** BOOLEAN is a data type that holds true or false values. Some database systems implement it using the BIT type.
7. CREATE TABLE Employees (
8. id INT PRIMARY KEY,
9. is_active BOOLEAN
10. );
11. **CHAR Data Type**The CHAR data type stores fixed-length character strings. It is suitable for fields with uniform length, such as country codes.
12. CREATE TABLE Countries (
13. code CHAR(3) PRIMARY KEY,
14. name VARCHAR(50)
15. );
16. **VARCHAR Data Type**The VARCHAR data type stores variable-length character strings, making it more flexible than CHAR.
17. CREATE TABLE Users (
18. id INT PRIMARY KEY,
19. username VARCHAR(50)
20. );

21. **DATE Data Type**The DATE data type is used to store calendar dates without time components.
22. CREATE TABLE Events (
23. event_id INT PRIMARY KEY,
24. event_date DATE
25. );
26. **DATETIME Data Type**The DATETIME data type stores date and time information.
27. CREATE TABLE Appointments (
28. appointment_id INT PRIMARY KEY,
29. appointment_time DATETIME
30. );
31. **DECIMAL Data Type**The DECIMAL data type stores precise numeric values, which is essential for financial calculations.
32. CREATE TABLE Transactions (
33. transaction_id INT PRIMARY KEY,
34. amount DECIMAL(10,2)
35. );

Each of these data types serves a specific purpose and is chosen based on the requirements of the database system. Understanding these data types and their practical applications helps in designing efficient and robust database systems that cater to the needs of different applications.

# Unit 4.3: Manipulation of data in database

## 4.3.1 Insert, Update, and Delete Records

Managing databases and tables is a fundamental aspect of database administration, ensuring that data is accurately stored, modified, and removed as needed. The three essential operations INSERT, UPDATE, and DELETE—are crucial for maintaining data integrity and consistency within a relational database. These operations allow users to add new records, modify existing entries, and remove unnecessary or outdated data. Effective management of databases and tables using these operations ensures smooth data handling, enhances performance, and maintains accuracy in various applications, including business, finance, healthcare, and education. The INSERT operation is used to add new records into a table, ensuring that fresh data is properly stored for retrieval and processing. In relational databases, each table consists of multiple columns, and when inserting data, it is necessary to provide values for the relevant fields. The INSERT INTO statement is used to add data, specifying the table name and the values to be inserted. For example, in an employee database, adding a new employee record involves inserting details such as employee ID, name, department, and salary. Proper data insertion ensures that records are available for queries, reporting, and further processing. Additionally, constraints such as NOT NULL, UNIQUE, and PRIMARY KEY help maintain data integrity by enforcing rules on inserted values. If an attempt is made to insert a record that violates these constraints, the database rejects the entry, ensuring data consistency. Updating records is equally important in database management, as information often changes over time. The UPDATE statement is used to modify existing data within a table, allowing records to be corrected, modified, or refreshed. This operation is essential for keeping information up to date, such as changing an employee's salary, updating a customer's contact details, or modifying a product's price. The UPDATE command is used along with the SET keyword to specify new values for one or more fields, and a WHERE clause is used to target specific records that need modification. Without a WHERE clause, the update operation may affect all records in a table, leading to unintended changes. For example, in a student database, updating a student's grade for a

specific course ensures that the database reflects the latest academic performance. To maintain data integrity, constraints and triggers may be applied to enforce business rules, ensuring that updates follow predefined guidelines.

The DELETE operation is used to remove records from a table when they are no longer needed. Deleting unnecessary or outdated records helps maintain an optimized database by reducing storage space and improving query performance. The DELETE FROM statement is used with a WHERE clause to specify which records should be removed. If a WHERE clause is omitted, all records in the table will be deleted, which can lead to unintended data loss. For example, in an online shopping database, deleting an order that has been canceled ensures that only valid transactions remain in the system. In cases where data recovery may be required, soft deletion techniques can be used, where a record is marked as inactive instead of being physically removed from the database. This approach allows data to be restored if needed, preserving historical information. Efficient management of INSERT, UPDATE, and DELETE operations is crucial for maintaining a well-organized database. Indexing plays a significant role in optimizing these operations, allowing the database to quickly locate and modify records. Additionally, transaction control mechanisms such as COMMIT and ROLLBACK ensure data consistency by allowing changes to be saved or undone in case of errors. The ACID (Atomicity, Consistency, Isolation, Durability) properties of a database management system (DBMS) ensure that all operations are processed reliably. For instance, in a banking system, when transferring money between accounts, the database ensures that either both the debit and credit operations are completed successfully or neither occurs, preventing data inconsistencies. Security is another key aspect of managing databases and tables. Permissions and access controls restrict unauthorized users from inserting, updating, or deleting records, ensuring that only authorized personnel can modify critical data. Backup strategies further safeguard data by providing recovery options in case of accidental deletion or corruption. Regular database maintenance, including indexing, optimizing queries, and archiving old records, ensures that databases remain efficient and scalable. In conclusion, INSERT, UPDATE, and DELETE operations are vital components of managing databases and tables, enabling users

to handle data effectively. Proper execution of these operations ensures data accuracy, consistency, and security, supporting various applications in different industries. By leveraging constraints, indexing, transaction management, and security measures, databases can be efficiently maintained, providing reliable and high-performance data management solutions.

**Summary**

The first topic introduces fundamental SQL commands, which form the backbone of interacting with relational databases. Commands such as `CREATE`, `SELECT`, `INSERT`, `UPDATE`, and `DELETE` allow users to define structures, query data, and perform essential database operations. Mastering these commands is crucial for efficient data handling.

Datatypes in DBMS are then explored, which define the nature of values that can be stored in a database. Proper use of datatypes like integer, varchar, date, or boolean ensures accurate storage, efficient processing, and logical representation of real-world data within a system.

The manipulation of data in databases further builds on SQL by focusing on operations that insert, update, delete, and retrieve data. These operations enable dynamic interaction with stored information and allow users to maintain data according to organizational requirements.

Integrity constraints are also introduced, which safeguard the correctness and consistency of data. Constraints such as primary keys, foreign keys, unique, not null, and check conditions prevent invalid entries, maintain relationships between tables, and enforce business rules at the database level.

Together, these topics provide the practical foundation for managing relational databases and tables. By combining SQL commands, proper datatype usage, data manipulation techniques, and integrity constraints, learners gain the ability to design and maintain secure, accurate, and reliable database systems.

**MCQs:**

1. **Which SQL command is used to create a new database?**
   a) MAKE DATABASE
   b) CREATE DATABASE
   c) NEW DATABASE
   d) ADD DATABASE
   (Answer-b)
2. **Which command is used to delete an entire database permanently?**
   a) DROP DATABASE
   b) DELETE DATABASE
   c) REMOVE DATABASE
   d) TRUNCATE DATABASE

(Answer-a)

3. **Which SQL command is used to remove all records from a table but keep the structure?**
   a) DELETE
   b) DROP
   c) TRUNCATE
   d) ALTER
   (Answer-c)
4. **Which of the following is a valid SQL data type?**
   a) STRING
   b) TEXT
   c) CHAR
   d) NUMERIC
   (Answer-c)
5. **Which command is used to change the structure of an existing table?**
   a) MODIFY TABLE
   b) CHANGE TABLE
   c) ALTER TABLE
   d) EDIT TABLE
   (Answer-c)
6. **What does the NOT NULL constraint do?**
   a) Ensures that a column does not contain duplicate values
   b) Prevents a column from having NULL values
   c) Sets a default value for the column
   d) Creates a new table
   (Answer-b)
7. **Which of the following statements about PRIMARY KEY is true?**
   a) A table can have multiple primary keys
   b) A primary key column can contain duplicate values
   c) A primary key ensures uniqueness and cannot be NULL
   d) A primary key can be removed using DELETE
   (Answer-c)
8. **Which SQL command is used to modify existing records in a table?**
   a) MODIFY
   b) CHANGE

c) UPDATE
d) ALTER
(Answer-c)

9. **What does the CHECK constraint do?**
   a) Ensures values in a column meet a specific condition
   b) Automatically fills a column with a default value
   c) Allows NULL values in a column
   d) Creates a new table
   (Answer-a)
10. **Which command is used to remove a table completely, including its structure?**
    a) DROP TABLE
    b) DELETE TABLE
    c) REMOVE TABLE
    d) TRUNCATE TABLE
    (Answer-a)

**Short Questions:**

1. What is the purpose of the CREATE DATABASE command?
2. How does the DROP DATABASE command work?
3. What is the difference between DELETE, DROP, and TRUNCATE?
4. What are the different data types available in SQL?
5. Explain the difference between CHAR and VARCHAR.
6. How does the ALTER TABLE command work?
7. What is the function of NOT NULL and UNIQUE constraints?
8. How can we update records in a table using SQL?
9. What is the purpose of the CHECK constraint?
10. How does the DEFAULT constraint work in SQL?

**Long Questions:**

1. Explain the process of creating and deleting a database in SQL.
2. Discuss the different SQL commands used to manage tables.
3. What are SQL data types, and how are they used in table creation?
4. Explain the differences between DELETE, DROP, and TRUNCATE with examples.
5. How does the ALTER TABLE command modify table structures?

6. Describe the different types of constraints used in database design.
7. Explain how the PRIMARY KEY and FOREIGN KEY constraints enforce data integrity.
8. What is the purpose of the CHECK constraint, and how is it implemented?
9. Write SQL queries to insert, update, and delete records from a table.
10. Discuss the importance of constraints in database security and integrity.

# MODULE 5
# DATA MANIPULATION

## 5.0 LEARNING OUTCOMES

- Learn how to use the SELECT, ORDER BY, WHERE, and SELECT DISTINCT commands.
- Understand different operators like AND, OR, IN, BETWEEN, LIKE, LIMIT, and IS NULL.
- Learn how to apply numeric, string, and date functions in SQL.
- Understand joins and their types: INNER JOIN, LEFT JOIN, RIGHT JOIN, SELF JOIN.
- Learn about aggregate functions like GROUP BY, HAVING, MIN(), MAX(), AVG(), SUM(), COUNT().
- Understand the concept of sub queries and their usage.

### 5.1.1 Select, Order By, Where, and Select Distinct

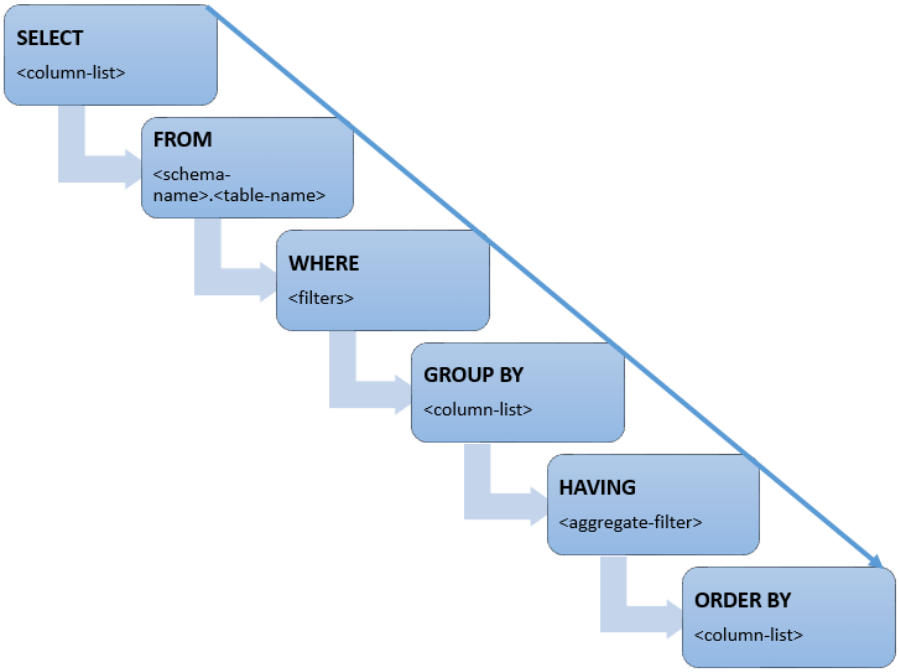



*Figure 5.1.1: Statement order of Queries*
*[Source: https://www.sqlrelease.com/]*

The world of data is vast and intricate, and the ability to extract meaningful information from databases is paramount in today's data-driven landscape. Structured Query Language (SQL) serves as the lingua franca for interacting with relational databases, allowing users to manipulate and retrieve data efficiently. This Module delves into the core SQL commands: Select, Order By, Where, and Select Distinct, which form the building blocks for data retrieval. Mastering these commands is essential for anyone seeking to navigate and leverage the power of relational databases. The SELECT statement is the cornerstone of data retrieval in SQL. It allows users to specify the columns they wish to retrieve from a table. The basic syntax of the Select statement is as follows: SELECT column1, column2, FROM table name;. Here, column1, column2, and so on, represent the names of the columns to be retrieved, and table name specifies the table from which the data will be fetched. For example, consider a table named "Employees" with columns "EmployeeID," "FirstName,"

"LastName," and "Department." To retrieve the "FirstName" and "LastName" of all employees, the following SQL query can be used: SELECT FirstName, LastName FROM Employees;. This query will return a result set containing the first and last names of all employees in the "Employees" table. Furthermore, the SELECT statement allows users to retrieve all columns from a table using the asterisk (*) wildcard. The syntax for this is: SELECT * FROM table name;. This retrieves every single column present in the table. For instance, SELECT * FROM Employees; would return all columns (Employee, First Name, LastName, Department) for all employees. While convenient, using SELECT * can be inefficient when dealing with large tables, as it retrieves unnecessary data. It's generally recommended to explicitly specify the columns needed to improve query performance. The ORDER BY clause is used to sort the result set of a SELECT statement in ascending or descending order based on one or more columns.[1] The syntax for ORDER BY is: SELECT column1, column2, ... FROM table name ORDER BY column name [ASC|DESC];. Here, column name specifies the column to sort by, and ASC (ascending) or DESC (descending) specifies the sorting order. If no sorting order is specified, the default is ascending. For instance, to retrieve all employees sorted by their "LastName" in ascending order, the following query can be used: SELECT FROM Employees ORDER BY LastName ASC;. To sort by "LastName" in descending order, the query would be: SELECT * FROM Employees ORDER BY LastName DESC.

Additionally, the ORDER BY clause can be used to sort by multiple columns. The sorting order is determined by the order in which the columns are specified. For example, to retrieve employees sorted first by "Department" in ascending order and then by "LastName" in ascending order within each department, the following query can be used: Select From Employees ORDER BY Department ASC, LastName ASC;. This query will first sort the employees by their department, and then within each department, it will sort them by their last name. The Where clause is used to filter the result set of a select statement based on specified conditions. It allows users to retrieve only the rows that meet certain criteria. The syntax for the Where clause is: SELECT column1, column2, . FROM table name Where

condition;. Here, condition represents the criteria that must be met for a row to be included in the result set. For example, to retrieve all employees from the "Sales" department, the following query can be used: Select From Employees Where Department = 'Sales';. This query will return only the rows where the "Department" column has the value "Sales." The WHERE clause can use various comparison operators, such as =, !=(not equal), >, <, >=, and <=. For instance, to retrieve all employees with an "Employee" greater than 100, the following query can be used: Select From Employees Where EmployeeID>100;. The WHERE clause can also use logical operators, such as AND, OR, and NOT, to combine multiple conditions. For example, to retrieve all employees from the "Sales" department with an "EmployeeID" greater than 100, the following query can be used: Select * From Employees Where Department = 'Sales' AND EmployeeID>100;. To retrieve all employees from either the "Sales" or "Marketing" department, the query would be: SELECT * FROM Employees Where Department = 'Sales' OR Department = ‘Marketing’. Furthermore, the Where clause can use the LIKE operator for pattern matching. The LIKE operator uses wildcard characters, such as % (any sequence of characters) and _ (any single character). For example, to retrieve all employees whose "LastName" starts with "S," the following query can be used: Select * From Employees WHERE LastName LIKE 'S%';. To retrieve all employees whose "LastName" contains "son," the query would be: SELECT * FROM Employees WHERE LastName LIKE '%son%';.TheWHERE clause can also use the IN operator to specify a list of values. For example, to retrieve all employees from the "Sales," "Marketing," or "HR" department, the following query can be used: Select From Employees where Department IN ('Sales', 'Marketing', 'HR'); The Select Distinct statement is used to retrieve only the unique values from a column. It eliminates duplicate rows from the result set. The syntax for Select Distinct Is: Select Distinct column1, column2, ... FROM table name;. For example, to retrieve the unique departments from the "Employees" table, the following query can be used: Select Distinct Department FROM Employees;. This query will return a list of all unique departments in the table, without any duplicates. The SELECT DISTINCT statement can also be used with multiple columns. In this case, it returns the unique combinations of values

from the specified columns. For example, to retrieve the unique combinations of "Department" and "Location" from the "Employees" table, the following query can be used: SELECT DISTINCT Department, Location FROM Employees;. This query will return a list of all unique combinations of department and location, without any duplicates. Now, let's explore some more complex scenarios and examples to solidify our understanding of these foundational SQL commands.

**Example 1: Retrieving Employees with Specific Criteria**

Consider a table named "Products" with columns "ProductID," "ProductName," "Category," and "Price." To retrieve all products from the "Electronics" category with a price greater than $500, the following query can be used:

SQL

SELECT*

FROM Products

WHERE Category ='Electronics'AND Price >500;

This query will return all columns for products that meet both conditions: they belong to the "Electronics" category and their price is greater than $500.

Example 2: Sorting Products by Price and Category

To retrieve all products sorted first by "Category" in ascending order and then by "Price" in descending order within each category, the following query can be used:

SQL

SELECT*

FROM Products

ORDERBY Category ASC, Price DESC;

This query will first sort the products by their category, and then within each category, it will sort them by their price in descending order.

Example 3: Retrieving Unique Categories

To retrieve the unique categories from the "Products" table, the following query can be used:

SQL

SELECTDISTINCT Category

FROM Products;

This query will return a list of all unique categories in the table, without any duplicates.

Example 4: Using LIKE for Pattern Matching

To retrieve all products whose "ProductName" starts with "Laptop," the following query can be used:

SQL

SELECT*

FROM Products
WHERE ProductName LIKE'Laptop%';
This query will return all columns for products whose name starts with "Laptop."
Example 5: Using IN for Multiple Values
To retrieve all products from the "Electronics," "Clothing," or "Books" category, the following query can be used:
SQL
SELECT*
FROM Products
WHERE Category IN ('Electronics', 'Clothing', 'Books');
This query will return all columns for products that belong to any of the specified categories.
Example 6: Combining WHERE, ORDER BY, and SELECT DISTINCT
To retrieve the unique categories of products with a price greater than $100, sorted in ascending order, the following query can be used:
SQL
SELECTDISTINCT Category
FROM Products
WHERE Price >100
ORDERBY Category ASC;
This query demonstrates how to combine multiple SQL commands to achieve a specific result.
Example 7: Retrieving Employees with Specific Name Patterns
Considering our "Employees" table, if we want to retrieve all employees whose first name contains the letter "a", we would use:
SQL
SELECT*
FROM Employees
WHERE FirstName LIKE'%a%';

**5.1.2 Operators: And, or, In, Between, Like, Limit, Is Null**

In the realm of data manipulation and analysis, the ability to filter and extract specific subsets of data is paramount. Operators play a crucial role in defining the criteria for these extractions, enabling us to pinpoint precise information within vast datasets. This Module delves into the practical application of several key operators: And, Or, In, Between, Like, Limit, and IS Null. We will explore how these

operators function, their syntax, and their real-world applications, particularly within the context of data analysis using Python and the pandas library.

**Logical Operators: AND and OR**

Logical operators are fundamental tools for combining and evaluating multiple conditions. The AND operator requires all specified conditions to be true, while the OR operator requires at least one condition to be true.

**AND Operator**

The AND operator is used to filter data that meets multiple criteria simultaneously. Consider a scenario where we have a dataset of customer information, including age and city. We want to extract records of customers who are both older than 25 and reside in New York.

Python

```
import pandas as pd

# Sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
'Age': [25, 30, 22, 28, 35],
'City': ['New York', 'London', 'Paris', 'New York', 'Rome']}
df = pd.DataFrame(data)

# Filter data using AND operator
filtered_df_and = df[(df['Age'] >25) & (df['City'] == 'New York')]
print("Filtered DataFrame (AND):\n", filtered_df_and)
```

In this example, (df['Age'] > 25) and (df['City'] == 'New York') are the conditions, and the & symbol represents the AND operator. The resulting filtered_df_andDataFrame will only contain rows where both conditions are satisfied.

**OR Operator**

The OR operator is used to filter data that meets at least one of the specified criteria. If we want to extract records of customers who are either older than 30 or reside in Paris, we can use the OR operator.

Python

```
# Filter data using OR operator
filtered_df_or = df[(df['Age'] >30) | (df['City'] == 'Paris')]
print("\nFilteredDataFrame (OR):\n", filtered_df_or)
```

Here, (df['Age'] > 30) and (df['City'] == 'Paris') are the conditions, and the | symbol represents the OR operator. The resulting filtered_df_orDataFrame will contain rows where either condition is true.

Relational Operators: IN, BETWEEN, LIKE, IS NULL

Relational operators allow us to compare values and filter data based on specific relationships.

**IN Operator**

The IN operator is used to filter data that matches any value in a specified list. For example, if we want to extract records of customers who reside in either New York or London, we can use the IN operator.

Python

```
# Filter data using IN operator
filtered_df_in = df[df['City'].isin(['New York', 'London'])]
print("\nFilteredDataFrame (IN):\n", filtered_df_in)
```

The isin() method is used to check if the values in the 'City' column are present in the provided list.

**BETWEEN Operator**

The BETWEEN operator is used to filter data that falls within a specified range. Consider a scenario where we have a dataset with a 'Salary' column, and we want to extract records of employees whose salaries are between $50,000 and $100,000.

Python

```
# Assuming 'Salary' is another column in your DataFrame
data['Salary'] = [60000, 80000, 45000, 90000, 110000]
df = pd.DataFrame(data)

filtered_df_between = df[(df['Salary'] >= 50000) & (df['Salary'] <= 100000)]
print("\nFilteredDataFrame (BETWEEN):\n", filtered_df_between)
```

In this example, we use the AND operator along with greater than or equal to (>=) and less than or equal to (<=) to define the range.

**LIKE Operator**

The LIKE operator is used for pattern matching in string columns. It allows us to filter data based on specific text patterns. In Python, we can achieve similar functionality using the str.startswith(), str.endswith(), and str.contains() methods. For example, if we want to

extract records of customers whose names start with 'A', we can use str.startswith().

Python

```
# Filter data using LIKE operator (startswith)
filtered_df_like = df[df['Name'].str.startswith('A')]
print("\nFilteredDataFrame (LIKE):\n", filtered_df_like)

# Filter data using LIKE operator (contains)
filtered_df_contains = df[df['Name'].str.contains('li')]
print("\nFilteredDataFrame (CONTAINS):\n", filtered_df_contains)
```

The str.startswith() method checks if the strings in the 'Name' column start with the specified prefix. The str.contains() method checks if the strings contain the specified substring.

**IS NULL Operator**

The IS NULL operator is used to filter data where a specific column contains null values. Null values represent missing or undefined data. If we have a dataset with a 'Phone' column, and we want to extract records where the phone number is missing, we can use the IS NULL operator.

Python

```
# Assuming 'Phone' is another column in your DataFrame
data['Phone'] = ['123-456-7890', None, '987-654-3210', None, '111-222-3333']
df = pd.DataFrame(data)

filtered_df_isnull = df[df['Phone'].isnull()]
print("\nFilteredDataFrame (IS NULL):\n", filtered_df_isnull)
```

The isnull() method returns a boolean Series indicating whether each value in the 'Phone' column is null.

**LIMIT Operator**

The LIMIT operator is used to restrict the number of rows returned by a query. While pandas doesn't have a direct equivalent to the SQL LIMIT operator, we can achieve the same result using slicing.

Python

```
# Limit the number of rows using slicing
limited_df = df[:3]
print("\nLimitedDataFrame:\n", limited_df)
```

This example extracts the first three rows of the DataFrame.

Advanced Filtering Techniques

Combining Multiple Operators

We can combine multiple operators to create complex filtering conditions. For example, we can extract records of customers who are older than 25, reside in New York, and have a salary between $50,000 and $100,000.

Python

```
combined_df = df[
   (df['Age'] >25) &
   (df['City'] == 'New York') &
   (df['Salary'] >= 50000) &
   (df['Salary'] <= 100000)
]
print("\nCombined Filtered DataFrame:\n", combined_df)
```

Using Query Method

Pandas provides a query() method that allows us to filter data using a string-based query. This method can be more concise and readable for complex filtering conditions.

Python

```
queried_df = df.query("Age > 25 and City == 'New York' and Salary >= 50000 and Salary <= 100000")
print("\nQueriedDataFrame:\n", queried_df)
```

Practical Applications

Data Cleaning and Preprocessing

Operators are essential for data cleaning and preprocessing. We can use them to identify and remove or replace missing values, filter out outliers, and correct inconsistencies in the data.

Python

```
# Remove rows with missing phone numbers
cleaned_df = df.dropna(subset=['Phone'])
print("\nCleanedDataFrame:\n", cleaned_df)
```

Exploratory Data Analysis (EDA)

Operators are used extensively in EDA to explore and analyze data. We can use them to create subsets of data for specific groups or conditions, calculate summary statistics, and visualize data distributions.

Python

```
# Calculate average salary for customers in New York
```

```
new_york_salary = df[df['City'] == 'New York']['Salary'].mean()
print("\nAverage Salary in New York:", new_york_salary)
```

Reporting and Visualization

Operators are used to generate reports and visualizations based on specific criteria. We can use them to create tables and charts that highlight key findings and insights.

Python

```
# Create a bar chart of age distribution for customers in
```

### 5.1.3 Numeric, String, and Date Functions

In the realm of programming and data analysis, the ability to manipulate data effectively is paramount. This Module delves into the fundamental operations involving numeric, string, and date data types, providing a comprehensive overview of essential functions and techniques. Mastering these functions is crucial for processing, transforming, and analyzing data, enabling developers and analysts to extract meaningful insights and build robust applications.

**Numeric Functions: The Foundation of Mathematical Operations**

Numeric functions form the bedrock of mathematical computations, enabling programmers to perform a wide range of operations on numerical data. These functions are essential for tasks such as data analysis, scientific computing, and financial modeling.

1. Basic Arithmetic Operations

The most fundamental numeric operations include addition, subtraction, multiplication, and division. These operations are essential for performing basic calculations and are supported by most programming languages.

Python

```
# Addition
result_addition = 10 + 5
print(f"10 + 5 = {result_addition}")

# Subtraction
result_subtraction = 10 - 5
print(f"10 - 5 = {result_subtraction}")

# Multiplication
result_multiplication = 10 * 5
print(f"10 * 5 = {result_multiplication}")
```

```
# Division
result_division = 10 / 5
print(f"10 / 5 = {result_division}")

# Integer Division
result_integer_division = 10 // 3
print(f"10 // 3 = {result_integer_division}")

# Modulus (Remainder)
result_modulus = 10 % 3
print(f"10 % 3 = {result_modulus}")

# Exponentiation
result_exponentiation = 2 ** 3
print(f"2 ** 3 = {result_exponentiation}")
```

These operations can be combined to perform more complex calculations, following the standard order of operations (PEMDAS/BODMAS).

2 Mathematical Functions

Beyond basic arithmetic, programming languages provide a rich set of mathematical functions for performing advanced calculations.

2.1 Absolute Value

The absolute value of a number is its distance from zero, regardless of its sign.

Python

```
# Absolute value
absolute_value = abs(-10)
print(f"Absolute value of -10: {absolute_value}")
```

2.2 Rounding Functions

Rounding functions are used to approximate numerical values to a specified number of decimal places.

Python

```
# Rounding
rounded_value = round(3.14159, 2)
print(f"Rounded value of 3.14159 to 2 decimal places: {rounded_value}")
```

2.3 Trigonometric Functions

Trigonometric functions, such as sine, cosine, and tangent, are essential for working with angles and geometric shapes.

Python

```
import math

# Sine
sine_value = math.sin(math.radians(30))  # Convert degrees to radians
print(f"Sine of 30 degrees: {sine_value}")

# Cosine
cosine_value = math.cos(math.radians(60))
print(f"Cosine of 60 degrees: {cosine_value}")

# Tangent
tangent_value = math.tan(math.radians(45))
print(f"Tangent of 45 degrees: {tangent_value}")
```

1.2.4 Logarithmic and Exponential Functions

Logarithmic and exponential functions are used in various scientific and engineering applications.

Python

```
import math

# Natural logarithm
log_value = math.log(10)
print(f"Natural logarithm of 10: {log_value}")

# Base-10 logarithm
log10_value = math.log10(100)
print(f"Base-10 logarithm of 100: {log10_value}")

# Exponential function
exp_value = math.exp(2)
print(f"Exponential of 2: {exp_value}")
```

2.4 Square Root and Power Functions

These functions are used to calculate square roots and powers of numbers.

Python

```
import math
```

```
# Square root
sqrt_value = math.sqrt(25)
print(f"Square root of 25: {sqrt_value}")

# Power
power_value = math.pow(2, 3)
print(f"2 raised to the power of 3: {power_value}")
```

3 Random Number Generation

Random number generation is crucial for simulations, games, and cryptographic applications.

Python

```
import random

# Generate a random integer between 1 and 10
random_integer = random.randint(1, 10)
print(f"Random integer between 1 and 10: {random_integer}")

# Generate a random float between 0 and 1
random_float = random.random()
print(f"Random float between 0 and 1: {random_float}")

# Generate a random choice from a list
choices = ['apple', 'banana', 'cherry']
random_choice = random.choice(choices)
print(f"Random choice from the list: {random_choice}")
```

Section 2: String Functions: Manipulating Text Data

String functions are essential for working with text data, enabling programmers to perform operations such as searching, replacing, and formatting strings.

Basic String Operations

Basic string operations include concatenation, slicing, and indexing.

Python

```
# Concatenation
string1 = "Hello"
string2 = "World"
concatenated_string = string1 + " " + string2
print(f"Concatenated string: {concatenated_string}")
```

```
# Slicing
text = "Python Programming"
sliced_text = text[0:6]
print(f"Sliced text: {sliced_text}")

# Indexing
first_char = text[0]
print(f"First character: {first_char}")
```

String Manipulation Functions

String manipulation functions are used to transform and modify strings.

Case Conversion

Case conversion functions are used to change the case of characters in a string.

Python

```
# Uppercase
uppercase_text = "hello world".upper()
print(f"Uppercase text: {uppercase_text}")

# Lowercase
lowercase_text = "HELLO WORLD".lower()
print(f"Lowercase text: {lowercase_text}")

# Title case
titlecase_text = "hello world".title()
print(f"Title case text: {titlecase_text}")
```

Searching and Replacing

Searching and replacing functions are used to find and replace substrings within a string.

Python

```
# Finding a substring
index = "Python Programming".find("Programming")
print(f"Index of 'Programming': {index}")

# Replacing a substring
replaced_text = "Python Programming".replace("Python", "Java")
print(f"Replaced text: {replaced_text}")
```

Splitting and Joining

Splitting and joining functions are used to divide and combine strings.

Python

```
# Splitting a string
words = "apple,banana,cherry".split(",")
print(f"Split words: {words}")

# Joining a list of strings
joined_string = "-".join(words)
print(f"Joined string: {joined_string}")
```

Trimming and Padding

Trimming and padding functions are used to remove whitespace and add characters to strings.

Python

```
# Trimming whitespace
trimmed_text = "  helloworld  ".strip()
print(f"Trimmed text: '{trimmed_text}'")

# Padding a string
padded_text = "hello".ljust(10, "*")
print(f"Padded text: '{padded_text}'")
```

String Formatting

String formatting is used to create formatted strings with placeholders for variables.

Python

```
# Using f-strings
name = "Alice"
age = 30
formatted_string = f"My name is {name} and I am {age} years old."
print(formatted_string)

# Using the format() method
formatted_string_2 = "My name is {} and I am {} years old.".format(name, age)
print(formatted_string_2)
```

Section 3: Date Functions: Working with Time Data

Date functions are essential for working with time data, enabling programmers to perform operations such as calculating time differences, formatting dates, and extracting date components.

Basic Date Operations

Basic date operations include creating date objects and extracting date components.

Python

```
import datetime

# Creating a date object
date_object = datetime.date, 1
```

### 5.2.1 Joins: Inner Join, Left Join, Right Join, Self Join

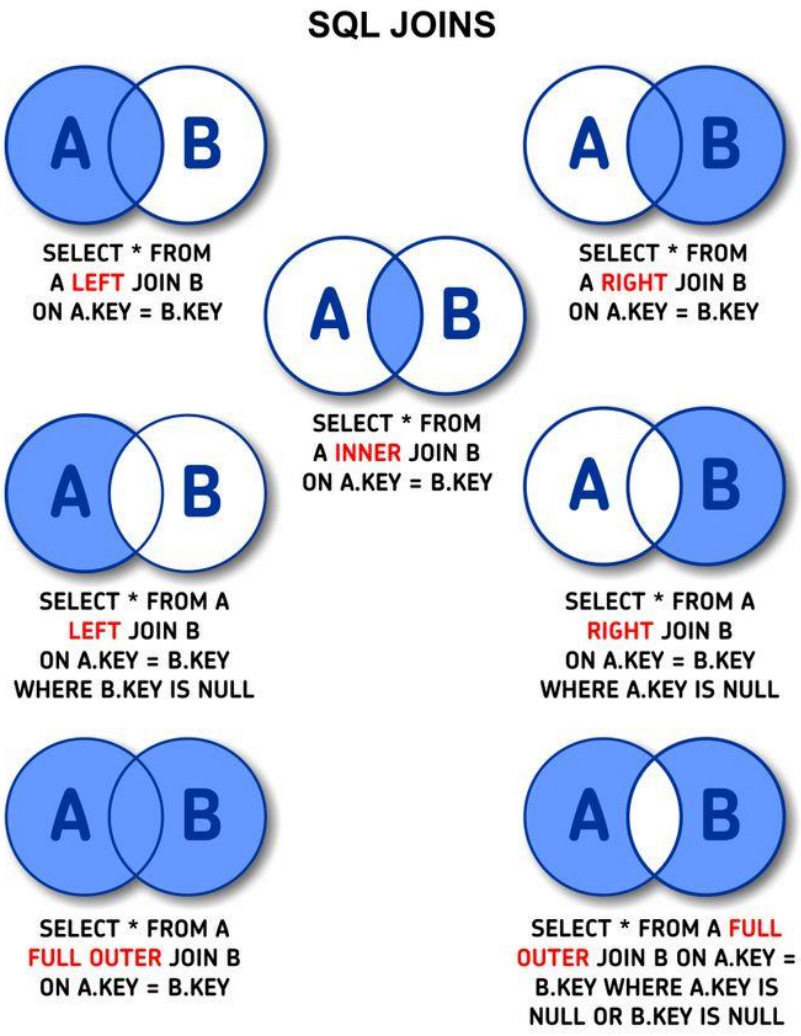



*Figure 5.2.12: SQL JOINS*
*[Source: https://i.pinimg.com/]*

Joins are essential in data manipulation as they allow users to retrieve and combine data from multiple tables based on related columns. Using different types of joins, databases efficiently link records to provide meaningful insights. The primary types of joins include INNER JOIN, LEFT JOIN, RIGHT JOIN, and SELF JOIN, each serving a specific purpose in data retrieval.

INNER JOIN is the most commonly used join type. It retrieves only the matching records between two tables based on a common column. Any record that does not have a corresponding match in both tables is

excluded. For example, in a database containing Orders and Customers tables, an INNER JOIN can be used to retrieve only the orders where a valid customer exists.

LEFT JOIN (or LEFT OUTER JOIN) returns all records from the left table and only the matching records from the right table. If no match is found in the right table, NULL values are displayed for those columns. This is useful when retrieving all customers and their orders, ensuring that customers without orders are still included in the result.

RIGHT JOIN (or RIGHT OUTER JOIN) works oppositely to LEFT JOIN. It retrieves all records from the right table and only the matching records from the left table. If no match is found in the left table, NULL values appear for its columns. This is beneficial when ensuring all orders are listed, even if some are not associated with a registered customer. SELF JOIN is a special join where a table is joined with itself. It is useful for hierarchical data, such as an employee table where each employee has a manager, linking the Employee_ID and Manager_ID columns within the same table.

**Mastering SQL Joins: Uniting Data from Multiple Tables**

In the realm of database management, relational databases are the cornerstone of organized data storage. These databases are structured around the concept of tables, where each table represents a distinct entity or concept. The power of relational databases lies in their ability to establish relationships between these tables, enabling the retrieval and analysis of combined data. This is where SQL joins come into play. Joins are fundamental SQL operations that allow you to combine rows from two or more tables based on a related column between them. This capability is essential for extracting meaningful insights from complex datasets, enabling you to answer questions that span multiple tables.

**Understanding the Need for Joins**

Imagine a scenario where you have two tables: Customers and Orders. The Customers table contains information about your customers, such as their names,[1]addresses, and contact details. The Orders table contains information about the orders placed by these customers, such as the order ID, order date, and the customer ID. To answer questions like "Which customer placed order 101?" or "What orders were

placed by Alice?", you need to combine data from both tables. This is where joins become indispensable. Without joins, you would have to manually search through both tables, which is inefficient and error-prone. Joins automate this process, allowing you to retrieve the combined data with a single SQL query.

**Types of SQL Joins**

SQL offers several types of joins, each serving a specific purpose. The most common types are:

- INNER JOIN: Returns only the rows that have matching values in both tables.
- LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table[2] and the matching rows from the right table. If there is no match,[3]the result will contain NULL values for the columns from the right table.
- RIGHT JOIN (or RIGHT OUTER JOIN): Returns all rows from the right table and the matching rows from the left table. If there is no match,[4] the result will contain NULL values for the columns from the left table.
- FULL OUTER JOIN: Returns all rows when there is a match in either left or right table.
- SELF JOIN: Joins a table with itself, using different aliases for the table.

INNER JOIN: Retrieving Matching Rows

The INNER JOIN is the most basic type of join. It returns only the rows that have matching values in both tables. The syntax for an INNER JOIN is as follows:

SQL

```
SELECT columns
FROM table1
INNERJOIN table2
ON table1.column_name = table2.column_name;
```

Here, table1 and table2 are the tables you want to join, and column_name is the column that is common to both tables. The ON clause specifies the join condition, which determines how the rows from the two tables are matched.

Example: INNER JOIN with Customers and Orders

Let's illustrate the INNER JOIN with our Customers and Orders tables.

Python

```
import pandas as pd

customers = pd.DataFrame({
'customer_id': [1, 2, 3],
'name': ['Alice', 'Bob', 'Charlie'],
'city': ['New York', 'London', 'Paris']
})

orders = pd.DataFrame({
'order_id': [101, 102, 103, 104],
'customer_id': [1, 1, 2, 3],
'order_date': ['-01-15', '2023-02-20', '2023-03-10', '2023-04-05']
})

inner_join_result = pd.merge(customers, orders, on='customer_id')
print("INNER JOIN:\n", inner_join_result)
```

This code will produce the following output:

```
INNER JOIN:
customer_id     name      city  order_idorder_date
0           1    Alice  New York       101  2023-01-15
1           1    Alice  New York       102  2023-02-20
2           2      Bob    London       103  2023-03-10
3           3  Charlie     Paris       104  2023-04-05
```

The INNER JOIN returns only the rows where the customer_id column has matching values in both tables.

LEFT JOIN: Retrieving All Rows from the Left Table

The LEFT JOIN returns all rows from the left table and the matching rows from the right table. If there is no match, the result will contain NULL values for the columns from the right table. The syntax for a LEFT JOIN is as follows:

SQL

```
SELECT columns
FROM table1
LEFTJOIN table2
ON table1.column_name = table2.column_name;
```

Here, table1 is the left table, and table2 is the right table.

Example: LEFT JOIN with Customers and Orders

Let's add a new customer to the Customers table who has not placed any orders.

Python

```
customers = pd.DataFrame({
'customer_id': [1, 2, 3, 4],
'name': ['Alice', 'Bob', 'Charlie', 'David'],
'city': ['New York', 'London', 'Paris', 'Tokyo']
})

left_join_result = pd.merge(customers, orders, on='customer_id', how='left')
print("\nLEFT JOIN:\n", left_join_result)
```

This code will produce the following output:

```
LEFT JOIN:
customer_id     name      city  order_idorder_date
0           1    Alice  New York     101.0  2023-01-15
1           1    Alice  New York     102.0  2023-02-20
2           2      Bob    London     103.0  2023-03-10
3           3  Charlie     Paris     104.0  2023-04-05
4           4    David    Tokyo       NaNNaN
```

The LEFT JOIN returns all rows from the Customers table, including the row for David, who has not placed any orders. The order_id and order_date columns for David contain NULL values.

RIGHT JOIN: Retrieving All Rows from the Right TableTheRIGHT JOIN is similar to the LEFT JOIN, but it returns all rows from the right table and the matching rows from the left table. If there[5] is no match, the result will contain NULL values for the columns from the left table. The syntax for a RIGHT JOIN is as follows:

SQL

```
SELECT columns
FROM table1
RIGHTJOIN table2
ON table1.column_name = table2.column_name;
```

Here, table1 is the left table, and table2 is the right table.

Example: RIGHT JOIN with Customers and Orders

Let's add a new order to the Orders table that does not have a corresponding customer in the Customers table.

Python

```
orders = pd.DataFrame({
'order_id': [101, 102, 103, 104, 105],
'customer_id': [1, 1, 2, 3, 5],
'order_date': ['2023-01-15', '2023-02-20', '2023-03-10', '2023-04-05',
'2023-05-10']
})

right_join_result = pd.merge(customers, orders, on='customer_id',
how='right')
print("\nRIGHT JOIN:\n", right_join_result)
```

This code will produce the following output:

```
RIGHT JOIN:
customer_id     name      city  order_idorder_date
0           1    Alice  New York       101  2023-01-15
1           1    Alice  New York       102  2023-02-20
2           2      Bob    London       103  2023-03-10
3           3  Charlie     Paris       104  2023-04-05
4           5      NaNNaN105  2023-05-10
```

The RIGHT JOIN returns all rows from the Orders table, including the row for order 1

# Unit 5.3: Mastering Aggregate Functions

## 5.3.1 Mastering Aggregate Functions: Unveiling Insights from Data

In the realm of relational databases, aggregate functions are indispensable tools for extracting meaningful insights from data. They enable us to summarize and analyze large datasets, revealing trends, patterns, and statistical measures that would otherwise remain hidden. This Module delves into the power of aggregate functions in SQL, focusing on `GROUP BY`, `HAVING`, `MIN()`, `MAX()`, `AVG()`, `SUM()`, and `COUNT()`, providing a comprehensive guide to their usage and applications.

**1. Introduction to Aggregate Functions**

Aggregate functions operate on a set of values, returning a single summary value. These functions are fundamental in data analysis, allowing us to calculate statistics such as minimum, maximum, average, sum, and count. Understanding and effectively utilizing aggregate functions is crucial for any database professional seeking to extract actionable information from their data.

**2. The `GROUP BY` Clause: Categorizing Data**

The `GROUP BY` clause is a cornerstone of aggregate function usage. It allows us to partition a dataset into groups based on one or more columns. Once grouped, aggregate functions can be applied to each group, producing summary results for each category.

Syntax:

```sql
SELECT column1, column2, aggregate_function(column3)
FROM table_name
GROUP BY column1, column2;
```

Example:

Consider a table named `Orders` with columns `CustomerID`, `ProductID`, and `Quantity`. To find the total quantity of each product ordered by each customer, we can use the following query:

```sql
```

```
SELECT CustomerID, ProductID, SUM(Quantity) AS TotalQuantity
FROM Orders
GROUP BY CustomerID, ProductID;
```

Explanation:

This query groups the `Orders` table by `CustomerID` and `ProductID`, calculating the sum of `Quantity` for each unique combination. The result set will contain the `CustomerID`, `ProductID`, and the corresponding `TotalQuantity` for each group.

3. The `HAVING` Clause: Filtering Grouped Data

The `HAVING` clause is used to filter the results of a `GROUP BY` query based on aggregate function values. It is similar to the `WHERE` clause, but operates on groups rather than individual rows.

Syntax:

```sql
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING aggregate_function(column2) condition;
```

Example:

Continuing with the `Orders` table, to find customers who have ordered a total quantity of more than 100 for any product, we can use the following query:

```sql
SELECT CustomerID, SUM(Quantity) AS TotalQuantity
FROM Orders
GROUP BY CustomerID
HAVING SUM(Quantity) > 100;
```

Explanation:

This query groups the `Orders` table by `CustomerID`, calculates the sum of `Quantity` for each customer, and then filters the results to include only customers whose total quantity exceeds 100.

4. The `MIN()` Function: Finding the Minimum Value

The `MIN()` function returns the minimum value from a set of values. It is commonly used to find the smallest value in a column or group.*Syntax:

```sql
SELECT MIN(column_name)
FROM table_name;
```

Example:

Consider a table named `Products` with columns `ProductID`, `ProductName`, and `Price`. To find the lowest price of all products, we can use the following query:

```sql
SELECT MIN(Price) AS LowestPrice
FROM Products;
```

Explanation:

This query returns the minimum value from the `Price` column, representing the lowest price of all products.

5. The `MAX()` Function: Finding the Maximum Value

The `MAX()` function returns the maximum value from a set of values. It is commonly used to find the largest value in a column or group.

Syntax:

```sql
SELECT MAX(column_name)
FROM table_name;
```

Example:

Using the `Products` table, to find the highest price of all products, we can use the following query:

```sql
SELECT MAX(Price) AS HighestPrice
FROM Products;
```

Explanation:

This query returns the maximum value from the `Price` column, representing the highest price of all products.

6. The `AVG()` Function: Calculating the Average Value

The `AVG()` function calculates the average value of a set of values. It is commonly used to find the mean of a column or group.

Syntax:

```sql
SELECT AVG(column_name)
FROM table_name;
```

Example:

Using the `Products` table, to find the average price of all products, we can use the following query:

```sql
SELECT AVG(Price) AS AveragePrice
FROM Products;
```

Explanation:
This query returns the average value from the `Price` column, representing the average price of all products.
7. The `SUM()` Function: Calculating the Sum of Values
The `SUM()` function calculates the sum of a set of values. It is commonly used to find the total of a column or group.
Syntax:
```sql
SELECT SUM(column_name)
FROM table_name;
```
Example:
Using the `Orders` table, to find the total quantity of all products ordered, we can use the following query:
```sql
SELECT SUM(Quantity) AS TotalQuantity
```

FROM Orders;
```

Explanation:
This query returns the sum of all values in the `Quantity` column, representing the total quantity of all products ordered.
8. The `COUNT()` Function: Counting Rows
The `COUNT()` function counts the number of rows in a table or group. It can be used to count all rows or only rows that meet specific criteria.
Syntax
```sql
SELECT COUNT(column_name)
FROM table_name;
SELECT COUNT(*)
FROM table_name;
```
Example:
Using the `Customers` table, to find the total number of customers, we can use the following query:
```sql
SELECT COUNT(*) AS TotalCustomers
FROM Customers;
```
To find the number of customers with a specific city, we can use the following query:

```sql
SELECT COUNT(*) AS CustomersInCity
FROM Customers
WHERE City = 'New York';
```
Explanation:
The first query returns the total number of rows in the `Customers` table, representing the total number of customers. The second query returns the number of rows where the `City` column is 'New York', representing the number of customers in New York.
9. Combining Aggregate Functions and `GROUP BY`
```

The true power of aggregate functions is realized when combined with the `GROUP BY` clause. This allows for the calculation of summary statistics for each group within a dataset.

Example:

Using the `Orders` table, to find the total quantity of each product ordered by each customer, and then filter the results to include only customers who have ordered a total quantity of more than 100 for any product, we can use the following query:

```sql
SELECT CustomerID, ProductID, SUM(Quantity) AS TotalQuantity
FROM Orders
GROUP BY CustomerID, ProductID
HAVING SUM(Quantity) > 100;
```

Explanation:

This query groups the `Orders` table by `CustomerID` and `ProductID`, calculates the sum of `Quantity` for each group, and then filters the results to include only customers who have ordered a total quantity of more than 100 for any product.

10. Nested Aggregate Functions

In some cases, it may be necessary to use nested aggregate functions to perform more complex calculations.

Example:

Using the `Orders` table, to find the average total quantity ordered by each customer, we can use the following query:

```sql
SELECT AVG(TotalQuantity) AS AverageTotalQuantity
FROM (
    SELECT CustomerID, SUM(Quantity) AS TotalQuantity
    FROM Orders
    GROUP BY CustomerID
) AS CustomerTotals;
```

Explanation:

This query first calculates the total quantity ordered by each customer using a subquery. Then, it calculates the average of these total quantities using the `AVG()` function.

11. Advanced `GROUP BY` and `HAVING` Usage

Advanced usage of `GROUP BY` and `HAVING` can involve grouping by multiple columns, using complex conditions in the `HAVING` clause, and combining aggregate functions with other SQL features.

Example:

Using the `Sales` table, to find the average sales amount for each product category in each region, and then filter the results to include only categories where the average sales amount is greater than $1000, we can use the following query:

```sql
SELECT Region, ProductCategory, AVG(SalesAmount) AS AverageSales
FROM Sales
GROUP BY Region, ProductCategory
HAVING AVG(SalesAmount) > 1000;
```

Explanation:

This query groups the `Sales` table by `Region` and `Product Category`, calculates the average sales amount for each group.

**5.3.2 Mastering Sub queries**

Sub queries, a powerful feature of SQL and data analysis tools, enable the construction of intricate queries by embedding one query within another. This capability allows for complex data retrieval and manipulation, enhancing the flexibility and expressiveness of data analysis. In this Module, we will delve into the intricacies of sub queries, exploring their various forms, applications, and best practices. We will also illustrate these concepts using Python and the Pandas library, bridging the gap between theoretical understanding and practical implementation.

**Understanding Sub queries**

A sub query, also known as an inner query or a nested query, is a query embedded within another query. The outer query, known as the main query, utilizes the result of the sub query to refine its own result set. Sub queries can appear in various parts of a query, including the

SELECT, FROM, WHERE, And HAVING clauses. They provide a means to break down complex data retrieval tasks into smaller, more manageable steps, enhancing the readability and maintainability of queries.

**Types of Sub queries**

Subqueries can be categorized into several types based on their behavior and the context in which they are used.

1. Scalar Subqueries: A scalar subquery returns a single value. It is typically used in the SELECT or WHERE clause to compare a value with the result of the subquery.
2. Row Subqueries: A row subquery returns a single row. It is used to compare a row of values with the result of the subquery.
3. Column Subqueries: A column sub query returns a single column of values. It is used in the IN, ANY, ALL, or EXISTS operators.
4. Table Subqueries: A table subquery returns a table of values. It is used in the FROM clause to treat the result of the subquery as a table.

Scalar Subqueries: Retrieving Single Values

Scalar subqueries are the simplest form of subqueries, returning a single value that can be used in comparisons or calculations. They are particularly useful for retrieving aggregate values, such as the maximum, minimum, or average of a column.

Example 1: Retrieving the Maximum Sales

Python

```
import pandas as pd

data = {'Year': [2022, 2022, 2023, 2023, 2024, 2024],
'Quarter': ['Q1', 'Q2', 'Q1', 'Q2', 'Q1', 'Q2'],
'Region': ['North', 'South', 'North', 'South', 'North', 'South'],
'Sales': [1000, 1200, 1500, 1800, 1200, 1500]}
df = pd.DataFrame(data)

max_sales = df['Sales'].max()
print("Maximum Sales:", max_sales)
```

In this example, the max() function acts as a scalar subquery, returning the maximum sales value from the DataFrame.

Example 2: Retrieving Sales Greater Than Average

Python

```
import pandas as pd
```

```
data = {'Year': [2022, 2022, 2023, 2023, 2024, 2024],
'Quarter': ['Q1', 'Q2', 'Q1', 'Q2', 'Q1', 'Q2'],
'Region': ['North', 'South', 'North', 'South', 'North', 'South'],
'Sales': [1000, 1200, 1500, 1800, 1200, 1500]}
df = pd.DataFrame(data)

avg_sales = df['Sales'].mean()
sales_above_avg = df[df['Sales'] >avg_sales]
print("Sales Above Average:\n", sales_above_avg)
```

Here, the mean() function acts as a scalar subquery, calculating the average sales, which is then used to filter the DataFrame.

Row Subqueries: Comparing Rows of Values

Row subqueries return a single row of values and are used to compare multiple columns simultaneously. They are particularly useful for finding rows that match a specific set of criteria.

Example 3: Finding the Row with Maximum Sales

Python

```
import pandas as pd

data = {'Year': [2022, 2022, 2023, 2023, 2024, 2024],
'Quarter': ['Q1', 'Q2', 'Q1', 'Q2', 'Q1', 'Q2'],
'Region': ['North', 'South', 'North', 'South', 'North', 'South'],
'Sales': [1000, 1200, 1500, 1800, 1200, 1500]}
df = pd.DataFrame(data)

max_sales_row = df[df['Sales'] == df['Sales'].max()]
print("Row with Maximum Sales:\n", max_sales_row)
```

In this example, the max() function acts as a scalar subquery, and the resulting max value is used to filter the dataframe.

Column Subqueries: Using IN, ANY, ALL, and EXISTS

Column subqueries return a single column of values and are used with operators like IN, ANY, ALL, and EXISTS.

1. IN Operator: The IN operator checks if a value exists in a set of values returned by the subquery.
2. ANY Operator: The ANY operator checks if a value satisfies the condition with any value returned by the subquery.

3. ALL Operator: The ALL operator checks if a value satisfies the condition with all values returned by the subquery.
4. EXISTS Operator: The EXISTS operator checks if a subquery returns any rows.

Example 4: Finding Sales in Specific Quarters

Python

```
import pandas as pd

data = {'Year': [2022, 2022, 2023, 2023, 2024, 2024],
'Quarter': ['Q1', 'Q2', 'Q1', 'Q2', 'Q1', 'Q2'],
'Region': ['North', 'South', 'North', 'South', 'North', 'South'],
'Sales': [1000, 1200, 1500, 1800, 1200, 1500]}
df = pd.DataFrame(data)

quarters = ['Q1', 'Q2']
sales_in_quarters = df[df['Quarter'].isin(quarters)]
print("Sales in Quarters Q1 and Q2:\n", sales_in_quarters)
```

In this example, the isin() function acts as an IN subquery equivalent, checking if the Quarter column values are in the specified list.

Example 5: Finding Sales Greater Than ANY Sales in 2022

Python

```
import pandas as pd

data = {'Year': [2022, 2022, 2023, 2023, 2024, 2024],
'Quarter': ['Q1', 'Q2', 'Q1', 'Q2', 'Q1', 'Q2'],
'Region': ['North', 'South', 'North', 'South', 'North', 'South'],
'Sales': [1000, 1200, 1500, 1800, 1200, 1500]}
df = pd.DataFrame(data)

sales_2022 = df[df['Year'] == 2022]['Sales']
sales_greater_any = df[df['Sales'] >sales_2022.min()]
print("Sales Greater Than ANY Sales in 2022:\n", sales_greater_any)
```

Here, the min() function acts as an ANY subquery equivalent, finding the minimum sales in 2022, and then filtering.

Example 6: Finding Sales Greater Than ALL Sales in 2022

Python

```
import pandas as pd
```

data = {'Year': [2022, 2022, 2023, 2023, 2024, 2024],
'Quarter': ['Q1', 'Q2', 'Q1', 'Q2', 'Q1', 'Q2'],
'Region': ['North', 'South', 'North', 'South', 'North', 'South'].

**Summary**

Explains how **aggregate functions** and **subqueries** help in analyzing data effectively. Aggregate functions like MIN(), MAX(), AVG(), SUM(), and COUNT() summarize data, often used with GROUP BY to categorize and HAVING to filter grouped results. They allow us to find totals, averages, or counts across datasets. Subqueries (nested queries) make complex analysis easier by embedding one query inside another. They can return a single value (scalar), a row, a column (used with IN, ANY, ALL, EXISTS), or even a table. Together, aggregate functions and subqueries provide powerful tools to extract meaningful insights, whether in SQL or replicated using Python libraries like Pandas.

**MCQs:**

1. **Which SQL statement is used to retrieve data from a database?**
   a) FETCH
   b) GET
   c) SELECT
   d) RETRIEVE
   (Answer-c)
2. **Which SQL clause is used to sort records in ascending or descending order?**
   a) SORT
   b) ORDER BY
   c) ARRANGE
   d) GROUP BY
   (Answer-b)
3. **Which SQL operator is used to filter results based on a range of values?**
   a) IN
   b) BETWEEN
   c) LIKE
   d) OR
   (Answer-b)

4. **Which function is used to find the highest value in a column?**
   a) COUNT()
   b) MAX()
   c) SUM()
   d) AVG()
   (Answer-b)
5. **What type of JOIN returns only matching records from both tables?**
   a) LEFT JOIN
   b) RIGHT JOIN
   c) INNER JOIN
   d) FULL JOIN
   (Answer-c)
6. **Which SQL function is used to count the number of records in a table?**
   a) COUNT()
   b) TOTAL()
   c) NUMBER()
   d) RECORDS()
   (Answer-a)
7. **What does the WHERE clause do in SQL?**
   a) Sorts data
   b) Filters records based on a condition
   c) Deletes records
   d) Modifies table structure
   (Answere-b)
8. **Which SQL operator is used to search for a pattern in a column?**
   a) LIKE
   b) IN
   c) IS NULL
   d) AND
   (Answer-a)
9. **A subquery is:**
   a) A query inside another query
   b) A duplicate query

c) A function call
d) A SQL join
(Answer-9)

10. **Which clause is used to filter records after grouping them?**
    a) GROUP BY
    b) WHERE
    c) HAVING
    d) ORDER BY
    (Answer-c)

**Short Questions:**

1. What is the purpose of the SELECT statement in SQL?
2. Explain the ORDER BY clause and how it works.
3. What is the difference between WHERE and HAVING clauses?
4. How do you filter records using BETWEEN and IN operators?
5. Define numeric functions in SQL with examples.
6. What are string functions? Give examples.
7. How do joins work in SQL? Explain different types.
8. What are aggregate functions, and how are they used?
9. Explain the difference between INNER JOIN and LEFT JOIN.
10. What is a subquery, and when is it used?

**Long Questions:**

1. Explain the SELECT statement with multiple examples.
2. Discuss the different SQL operators and their uses.
3. How do numeric, string, and date functions work in SQL? Provide examples.
4. Explain different types of joins with real-world examples.
5. How do aggregate functions work? Explain GROUP BY, HAVING, MIN(), MAX(), AVG(), SUM(), COUNT().
6. What is the difference between WHERE and HAVING clauses?
7. Explain ORDER BY and LIMIT in SQL.
8. Discuss subqueries and how they can be used to filter data.
9. Write SQL queries to demonstrate different JOIN operations.
10. Explain how data manipulation queries improve database performance.

**Glossary**

- ACID Properties: Set of principles (Atomicity, Consistency, Isolation, Durability) ensuring reliable transactions in a database.
- Application Layer: The middle tier in 3-tier architecture where application logic runs (e.g., Java, Python).
- Backup & Recovery: Mechanisms to restore data in case of failure or corruption.
- Big Data: Large and complex datasets characterized by Volume, Velocity, Variety, and Veracity.
- Clustered Storage: Storage method that groups related data physically close to improve performance.
- Concurrency Control: Ensures correct execution of transactions by multiple users at the same time.
- CREATE (DDL): SQL command to create a new object such as a table.
- Data Abstraction: Hiding complex storage details while presenting a user-friendly view of data.
- Data Definition Language (DDL): SQL language subset used to define or alter database schema (e.g., CREATE, ALTER).
- Data Independence: Ability to change storage structure without affecting applications.
- Data Manipulation Language (DML): SQL commands used to manage data in tables (e.g., SELECT, INSERT).
- Data Mining: Process of discovering patterns and knowledge from large datasets.
- Data Model: Framework for organizing data and defining relationships (e.g., Relational, ER, Object-Based).
- Data Warehouse: A central repository that stores historical data for analytical purposes.
- Database Administrator (DBA): Person responsible for database installation, security, tuning, and backups.
- Database Designer: Professional who defines the structure and schema of a database.
- Database Management System (DBMS): Software used to store, retrieve, and manage data in databases.

- Entity: An object or thing in the real world that is distinguishable from other objects (e.g., Student, Course).
- Entity-Relationship (ER) Model: Conceptual model used for database design showing entities, attributes, and relationships.
- Execution Plan: Optimized sequence of operations for executing a database query.
- Foreign Key: A key in one table that refers to the primary key in another table to establish relationships.
- FROM (SQL): Clause used to specify the table in a SQL query.
- GRANT (DCL): SQL command to give privileges to a user or role.
- GROUP BY: SQL clause used to group rows with the same values in specified columns.
- HAVING: Clause used to filter groups created by GROUP BY.
- Index: Database object that speeds up data retrieval operations.
- Instance: The current state or snapshot of the data in a database at a specific moment.
- INSERT (DML): SQL command used to add new records to a table.
- JOIN: SQL operation used to combine rows from two or more tables based on a related column.
- Logical Level: Middle level of data abstraction showing what data is stored and how it relates.
- LIKE: SQL operator used for pattern matching.
- MongoDB: A NoSQL database that stores data in JSON-like documents.
- MySQL: Popular open-source relational database management system.
- Normalization: Process of organizing data to reduce redundancy and improve integrity.
- NoSQL: Type of database that handles semi-structured or unstructured data (e.g., MongoDB, Couchbase).

- Object-Based Model: A data model that represents real-world entities as objects with attributes and methods.

- ORDER BY: SQL clause used to sort query results in ascending or descending order.
- Physical Level: The lowest level of data abstraction detailing how data is physically stored.
- Primary Key: A field (or combination of fields) that uniquely identifies each record in a table.
- PostgreSQL: Open-source object-relational database system.
- Query: A command used to retrieve or manipulate data from a database.
- Query Optimization: Process to enhance SQL query performance using indexes, execution plans, and rewriting.
- Relational Model: Data model that organizes data into tables with rows and columns.
- REVOKE (DCL): Removes access privileges from users or roles.
- ROLLBACK (TCL): Reverses changes made in a transaction since the last COMMIT.
- Schema: The overall design or blueprint of a database, including tables, attributes, and constraints.
- SELECT (DML): SQL command to retrieve data from a table.
- Semi-Structured Model: Data model used for loosely organized data like JSON or XML.
- System Analyst: User who connects system requirements with database implementation.
- TCL (Transaction Control Language): SQL subset to control transactions (e.g., COMMIT, ROLLBACK).
- TRUNCATE (DDL): Removes all records from a table while preserving its structure.
- Transaction: A sequence of database operations treated as a single logical unit.
- UPDATE (DML): SQL command to modify existing records in a table.
- User Roles: Defined set of permissions assigned to users to control database access.
- View Level: Highest level of abstraction, showing user-specific perspectives of data.

- WHERE Clause: SQL condition to filter query results.
- Weak Entity: An entity that cannot exist without a related strong entity and does not have a primary key.

## References

### Introduction to Database Management System (Module 1)


1. Elmasri, R., & Navathe, S. B. (2016). Fundamentals of Database Systems (7th ed.). Pearson.
2. Ramakrishnan, R., & Gehrke, J. (2020). Database Management Systems (3rd ed.). McGraw-Hill.
3. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database System Concepts (7th ed.). McGraw-Hill.
4. Date, C. J. (2019). An Introduction to Database Systems (8th ed.). Pearson.
5. Coronel, C., & Morris, S. (2018). Database Systems: Design, Implementation, & Management (13th ed.). Cengage Learning.


### Data Modeling and Database Design (Module 2)


1. Teorey, T. J., Lightstone, S. S., Nadeau, T., & Jagadish, H. V. (2011). Database Modeling and Design (5th ed.). Morgan Kaufmann.
2. Connolly, T. M., & Begg, C. E. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management (6th ed.). Pearson.
3. Hoffer, J. A., Ramesh, V., & Topi, H. (2016). Modern Database Management (12th ed.). Pearson.
4. Chen, P. P. (1976). The Entity-Relationship Model—Toward a Unified View of Data. ACM Transactions on Database Systems, 1(1), 9-36.
5. Bagui, S., & Earp, R. (2011). Database Design Using Entity-Relationship Diagrams (2nd ed.). CRC Press.


### Relational Database Design (Module 3)


1. Garcia-Molina, H., Ullman, J. D., & Widom, J. (2008). Database Systems: The Complete Book (2nd ed.). Pearson.
2. Date, C. J. (2015). SQL and Relational Theory: How to Write Accurate SQL Code (3rd ed.). O'Reilly Media.
3. Fagin, R., Vardi, M. Y., & Ullman, J. D. (1983). The Theory of Data Dependencies – A Survey. Mathematics of Information Processing, 19, 19-71.
4. Kent, W. (1983). A Simple Guide to Five Normal Forms in Relational Database Theory. Communications of the ACM, 26(2), 120-125.
5. Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, 13(6), 377-387.


### Managing Database and Table (Module 4)

1. Beaulieu, A. (2020). Learning SQL: Generate, Manipulate, and Retrieve Data (3rd ed.). O'Reilly Media.
2. Faroult, S., & Robson, P. (2006). The Art of SQL. O'Reilly Media.
3. Groff, J. R., Weinberg, P. N., & Oppel, A. J. (2009). SQL: The Complete Reference (3rd ed.). McGraw-Hill.
4. Taylor, A. G. (2018). SQL For Dummies (9th ed.). For Dummies.
5. Price, J. (2019). Oracle Database 19c: The Complete Reference. McGraw-Hill.

**Data Manipulation (Module 5)**

1. Celko, J. (2014). SQL for Smarties: Advanced SQL Programming (5th ed.). Morgan Kaufmann.
2. Viescas, J. L. (2018). SQL Queries for Mere Mortals: A Hands-On Guide to Data Manipulation in SQL (4th ed.). Addison-Wesley.
3. Molinaro, A. (2020). SQL Cookbook: Query Solutions and Techniques for All SQL Users (2nd ed.). O'Reilly Media.
4. Richards, B. (2018). Learning SQL: Master SQL Fundamentals (3rd ed.). O'Reilly Media.
5. Forta, B. (2018). Sams Teach Yourself SQL in 10 Minutes (5th ed.). Sams Publishing.

# MATS UNIVERSITY

## MATS CENTRE FOR DISTANCE AND ONLINE EDUCATION


**UNIVERSITY CAMPUS:** Aarang Kharora Highway, Aarang, Raipur, CG, 493 441

**RAIPUR CAMPUS:** MATS Tower, Pandri, Raipur, CG, 492 002

**T :** 0771 4078994, 95, 96, 98 **Toll Free ODL MODE :** 81520 79999, 81520 29999

**Website:** www.matsodl.com