**SE IV AIDS/AIML DBMS ALARD UNIVERSITY**

****

# SCHOOL OF ENGINEERING AND TECHNOLOGY

## DEPARTMENT OF AIML/AIDS ENGINEERING

**=================================================================================**

**Case Study: 01**

**Name of Student: Roll Number: Class:**

**TITLE: MySQL Database AIM**:

1. Study of MySQL Open-source software. Discuss the characteristics like efficiency, scalability, performance and transactional properties.
2. Install and configure client and server of MySQL. (Show all commands and necessary steps for installation and configuration)
3. Study of SQLite: What is SQLite? Uses of SQLite. Building and installing SQLite

**OBJECTIVE:** Understand the fundamental concepts of database management. These concepts include aspects of database design, database languages, and database-system implementation. To learn the SQL database system. To provide a strong formal foundation in database concepts, recent technologies and best industry practices.

## SOFTWARE REQUIREMENTS:

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Name of Software and Hardware** | **Latest Version** |
| 1 | SQL Server 2022 | 16.0.4105.2 |
| 2 | CPU, RAM, Hard drive, Network adapter | 4 cores, 2500 MHz, 8 GB,  300 GB, RAID  recommended, 1 Gbit |

**SE IV AIDS/AIML DBMS ALARD UNIVERSITY THEORY:**

### Study of MySQL Open-source software. Discuss the characteristics like efficiency, scalability,

**performance and transactional properties.**

MySQL is a relational database management system (RDBMS) based on the SQL (Structured Query Language) queries. It is one of the most popular languages for accessing and managing the records in the table. MySQL is open-source and free software under the GNU license. Oracle Company supports it.

### Easy to use

MySQL is easy to use. We have to get only the basic knowledge of SQL. We can build and interact with MySQL by using only a few simple SQL statements.

### It is secure

MySQL consists of a solid data security layer that protects sensitive data from intruders. Also, passwords are encrypted in MySQL.

### Client/ Server Architecture

MySQL follows the working of a client/server architecture. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they can query data, save changes, etc.

### Free to download

MySQL is free to use so that we can download it from MySQL official website without any cost.

### It is scalable

MySQL supports multi-threading that makes it easily scalable. It can handle almost any amount of data, up to as much as 50 million rows or more. The default file size limit is about 4 GB. However, we can increase this number to a theoretical limit of 8 TB of data.

### Speed

MySQL is considered one of the very fast database languages, backed by a large number of the benchmark test.

### High Flexibility

MySQL supports a large number of embedded applications, which makes MySQL very flexible. Compatible on many operating systems

## SE IV AIDS/AIML DBMS ALARD UNIVERSITY

MySQL is compatible to run on many operating systems, like Novell NetWare, Windows\* Linux\*, many varieties of UNIX\* (such as Sun\* Solaris\*, AIX, and DEC\* UNIX), OS/2, FreeBSD\*, and others. MySQL also provides a facility that the clients can run on the same computer as the server or on another computer (communication via a local network or the Internet).

### Allows roll-back

MySQL allows transactions to be rolled back, commit, and crash recovery.

### Memory efficiency

Its efficiency is high because it has a very low memory leakage problem.

### High Performance

MySQL is faster, more reliable, and cheaper because of its unique storage engine architecture. It provides very high-performance results in comparison to other databases without losing an essential functionality of the software. It has fast loading utilities because of the different cache memory.

### High Productivity

MySQL uses Triggers, Stored procedures, and views that allow the developer to give higher productivity. Platform Independent

It can download, install, and execute on most of the available operating systems.

### Partitioning

This feature improves the performance and provides fast management of the large database.

### GUI Support

MySQL provides a unified visual database graphical user interface tool named "MySQL Workbench" to work with database architects, developers, and Database Administrators. MySQL Workbench provides SQL development, data modeling, data migration, and comprehensive administration tools for server configuration, user administration, backup, and many more. MySQL has a fully GUI supports from MySQL Server version 5.6 and higher.

### Dual Password Support

MySQL version 8.0 provides support for dual passwords: one is the current password, and another is a secondary password, which allows us to transition to the new password.

## SE IV AIDS/AIML DBMS ALARD UNIVERSITY

### Install and configure client and server of MySQL. (Show all commands and necessary steps for installation and configuration)

**Step 1:** After downloading the setup, unzip it anywhere and double click the MSI **installer .exe file.** It will give the following screen:

**Step 2:** In the next wizard, choose the **Setup Type**. There are several types available, and you need to choose the appropriate option to install MySQL product and [features.](https://www.javatpoint.com/mysql-features) Here, we are going to select the **Full** option and click on the Next button. This option will install the following things: MySQL Server, MySQL Shell, MySQL Router, [MySQL Workbench,](https://www.javatpoint.com/mysql-workbench) MySQL Connectors, documentation, samples and examples, and many more.

**Step 3:** Once we click on the Next button, it may give information about some features that may fail to install on your system due to a lack of requirements. We can resolve them by clicking on the **Execute** button that will install all requirements automatically or can skip them. Now, click on the Next button.

**Step 4:** In the next wizard, we will see a dialog box that asks for our confirmation of a few products not getting installed. Here, we have to click on the **Yes** button. After clicking on the Yes button, we will see the list of the products which are going to be installed. So, if we need all products, click on the Execute button.

**Step 5:** Once we click on the Execute button, it will download and install all the products. After completing the installation, click on the Next button.

**Step 6:** In the next wizard, we need to configure the MySQL Server and Router. Here, I am not going to configure the Router because there is no need to use it with MySQL. We are going to show you how to configure the server only. Now, click on the Next button.

**Step 7:** As soon as you will click on the Next button, you can see the screen below. Here, we have to configure the MySQL Server. Now, choose the Standalone MySQL Server/Classic MySQL Replication option and click on Next. Here, you can also choose the InnoDB Cluster based on your needs.

**Step 8:** In the next screen, the system will ask you to choose the Config Type and other connectivity options. Here, we are going to select the **Config Type** as 'Development Machine' and Connectivity as **TCP/IP,** and **Port Number** is 3306, then click on Next.

## SE IV AIDS/AIML DBMS ALARD UNIVERSITY

**Step 9:** Now, select the Authentication Method and click on Next. Here, I am going to select the first option.

**Step 10:** The next screen will ask you to mention the MySQL Root Password. After filling the password details, click on the Next button.

**Step 11:** The next screen will ask you to configure the Windows Service to start the server. Keep the default setup and click on the Next button.

**Step 12:** In the next wizard, the system will ask you to apply the Server Configuration. If you agree with this configuration, click on the Execute button.

**Step 13:** Once the configuration has completed, you will get the screen below. Now, click on the **Finish**

button to continue.

**Step 14:** In the next screen, you can see that the Product Configuration is completed. Keep the default setting and click on the Next-> Finish button to complete the MySQL package installation.

**Step 15:** In the next wizard, we can choose to configure the Router. So, click on Next->Finish and then click the Next button.

**Step 16:** In the next wizard, we will see the Connect to Server option. Here, we have to mention the root password, which we had set in the previous steps. In this screen, it is also required to check about the connection is successful or not by clicking on the Check button. If the connection is successful, click on the Execute button. Now, the configuration is complete, click on Next.

**Step 17:** In the next wizard, select the applied configurations and click on the Execute button.

**Step 18:** After completing the above step, we will get the following screen. Here, click on the Finish button.

**Step 19:** Now, the MySQL installation is complete.

## SE IV AIDS/AIML DBMS ALARD UNIVERSITY

### Study of SQLite: What is SQLite? Uses of SQLite. Building and installing SQLite THEORY:

* SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a database, which is zero-configured, which means like other databases you do not needto configure it in your system.
* SQLite engine is not a standalone process like other databases, you can linkit statically or dynamically as per your requirement with your application.
* SQLite accesses its storage files directly.

**Why Use SQLite?**

* SQLite is an open-source software. The software does not require any license afterinstallation.
* SQLite is serverless as it doesn't need a different server process or system tooperate.
* SQLite facilitates you to work on multiple databases on the same sessionsimultaneously, thus making it flexible.
* SQLite is a cross-platform DBMS that can run on all platforms, including macOS,Windows, etc.
* SQLite doesn't require any configuration. It needs no setup or administration.

**How to Install SQLite on Windows?**

Follow these steps:

**Step1:** Go to the official [SQLite website](https://www.sqlite.org/download.html) and download precompiled binaries from theWindows section.

**Step2:** Download the file (sqlite-tools-win32-x86-3270200.zip) as shown in the figure belowand extract these files in a folder of your choice.

**Step3:** Create a folder named sqlite in the C directory and copy the sqlite3.exe file in thesqlite folder.

**Step4:** Now, open the Command Prompt and navigate to the C:\sqlite folder. Type sqlite3 andpress enter.

**SE IV AIDS/AIML DBMS ALARD UNIVERSITY**

**SQLite Commands**

1. **DDL (Data Definition Language)** is used to create and modify database objects such as tables, indices, and views.

**Some examples of DDL statements in SQLite are:**

CREATE TABLE: creates a new table in the database ALTER TABLE: modifies an existing table in the database DROP TABLE: deletes a table from the database CREATE INDEX: creates a new index on a table

DROP INDEX: deletes an index from a table

1. **DML (Data Modification Language)** is used to modify the data stored in the database.

**Some examples of DML statements in SQLite are:**

INSERT INTO: inserts a new row into a table

UPDATE: updates the data in one or more rows of a table DELETE FROM: deletes one or more rows from a table

1. **DQL (Data Query Language)** is used to retrieve data from the database.

**Some examples of DQL statements in SQLite are:**

SELECT: retrieves data from one or more tables in the database

JOIN: retrieves data from multiple tables based on a common field GROUP BY: groups the results of a query by one or more fields HAVING: filters the results of a query based on a condition

**SE IV AIDS/AIML DBMS ALARD UNIVERSITY**

## CONCLUSION:

In conclusion, MySQL is a robust open-source database system with strong characteristics in efficiency, scalability, performance, and transactional properties. Its features make it a popular choice for various applications ranging from small projects to large-scale enterprise systems. In conclusion, the steps provided above guide you through the installation and configuration of MySQL server and client on a Linux system. By following these steps, you can set up MySQL, secure the installation, create a new user and database, and test the connection successfully. This ensures that you have a functional MySQL environment ready for use on your system. In conclusion, SQLite is a lightweight, self-contained, serverless, public domain database engine that is widely used due to its simplicity, reliability, and efficiency. It is commonly employed in embedded systems, mobile applications, small to medium-sized websites, and as a local storage solution. Building and installing SQLite involves downloading the source code, configuring the build, compiling it, and installing the binaries on the system, allowing users to utilize SQLite for their data storage needs effectively

## QUESTIONS:

1. How does MySQL handle concurrency control in a multi-user environment, and what mechanisms does it employ to ensure data consistency and integrity during transactions?
2. Can you discuss the role of indexing in MySQL databases and how it impacts query performance, especially in scenarios with large datasets and complex queries?
3. What are the commands and steps needed to install and configure both MySQL client and server on a system?
4. Could you please provide a comprehensive guide with all the commands and steps required to install and configure MySQL client and server on a system?
5. What is SQLite, what are its common uses, and how can one build and install SQLite?
6. How is it typically utilized, and what are the steps involved in building and installing SQLite?
7. How is SQLite commonly used?
8. What are the steps for building and installing SQLite?
9. How can I install and configure MySQL client and server, including all necessary commands and steps?
10. How does MySQL handle data replication and what strategies can be employed to ensure high availability and fault tolerance in distributed database systems using MySQL?

**SE IV AIDS/AIML DBMS ALARD UNIVERSITY**

****

# SCHOOL OF ENGINEERING AND TECHNOLOGY

## DEPARTMENT OF AIML/AIDS ENGINEERING

**=================================================================================**

**Case Study: 02**

**Name of Student: Roll Number: Class:**

**TITLE: Student / Timetable / Reservation / any data Management System**

**AIM**: Design any database with at least 3 entities and relationships between them. Draw suitable ER/EER diagram for the system

**OBJECTIVE:** To give systematic database design approaches covering conceptual design, logical design and an overview of physical design.

**SOFTWARE REQUIREMENTS:**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Name of Software and Hardware** | **Latest Version** |
| 1 | SQL Server 2022 | 16.0.4105.2 |
| 2 | CPU, RAM, Hard drive, Network adapter | 4 cores, 2500 MHz, 8 GB,  300 GB, RAID  recommended, 1 Gbit |

**SE IV AIDS/AIML DBMS ALARD UNIVERSITY**

**THEORY:**

### [What is an ER Diagram?](https://www.simplilearn.com/tutorials/sql-tutorial/er-diagram-in-dbms#what_is_an_er_diagram)

* An Entity Relationship Diagram is a diagram that represents relationships among entities in a database. It is commonly known as an ER Diagram. An ER Diagram in [DBMS](https://www.simplilearn.com/what-is-database-management-article) plays a crucial role in designing the database.
* An Entity Relationship Diagram (ER Diagram) pictorially explains the relationship between entities to be stored in a database. Fundamentally, the ER Diagram is a structural design of the database. It acts as a framework created with specialized symbols for the purpose of defining the relationship between the database entities. ER diagram is created based on three principal components: entities, attributes, and relationships.
* The following diagram showcases two entities - Student and Course, and their relationship. The rela- tionship described between student and course is many-to-many, as a course can be opted by several students, and a student can opt for more than one course. Student entity possesses attributes - Stu\_Id, Stu\_Name & Stu\_Age. The course entity has attributes such as Cou\_ID & Cou\_Name.

### What is an ER Model?

An Entity-Relationship Model represents the structure of the [database](https://www.simplilearn.com/tutorials/sql-tutorial/create-mysql-database) with the help of a diagram. ER Modelling is a systematic process to design a database as it would require you to analyze all data require- ments before implementing your database.

### History of ER models

Peter Chen proposed ER Diagrams in 1971 to create a uniform convention that can be used as a concep- tual modeling tool. Many models were presented and discussed, but none were suitable. The data structure diagrams offered by Charles Bachman also inspired his model.

### Why Use ER Diagrams in DBMS?

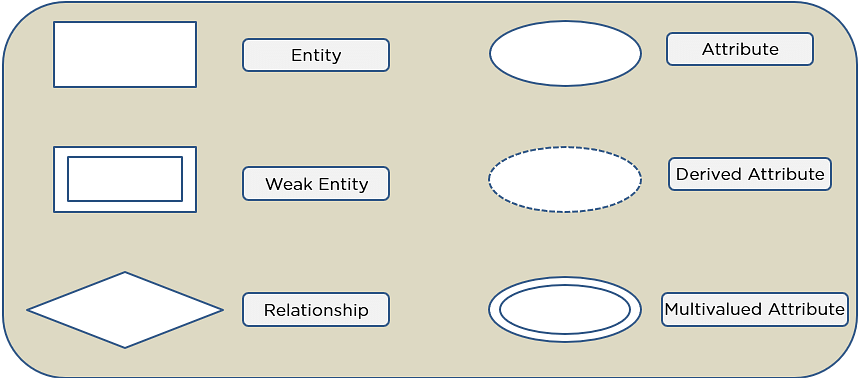
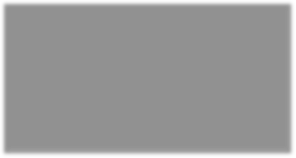
* ER Diagram helps you conceptualize the database and lets you know which fields need to be embedded for a particular entity
* ER Diagram gives a better understanding of the information to be stored in a database

13

* It reduces complexity and allows database designers to build databases quickly
* It helps to describe elements using Entity-Relationship models
* It allows users to get a preview of the logical structure of the database

### Symbols Used in ER Diagrams

* **Rectangles:** This Entity Relationship Diagram symbol represents entity types
* **Ellipses:** This symbol represents attributes
* **Diamonds:** This symbol represents relationship types
* **Lines:** It links attributes to entity types and entity types with other relationship types
* **Primary key:** Here, it underlines the attributes
* **Double Ellipses:** Represents multi-valued attributes



### Components of ER Diagram

You base an ER Diagram on three basic concepts:

### Entities

* 1. Weak Entity
  2. Strong Entity

### Attributes

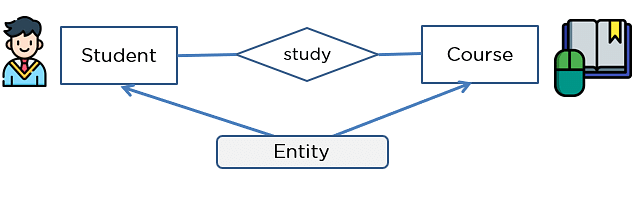
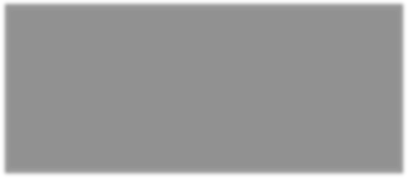
* 1. Key Attribute
  2. Composite Attribute
  3. Multivalued Attribute
  4. Derived Attribute

### Relationships

* 1. One-to-One Relationships
  2. One-to-Many Relationships
  3. Many-to-One Relationships
  4. Many-to-Many Relationships

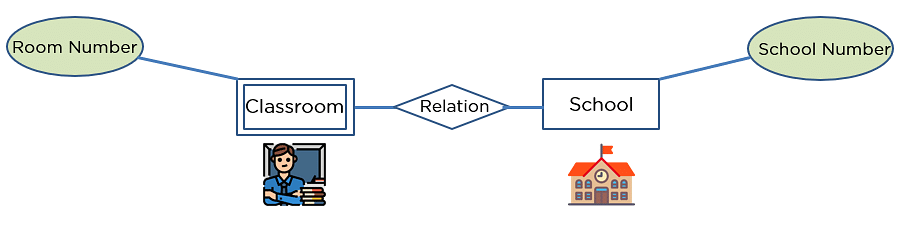
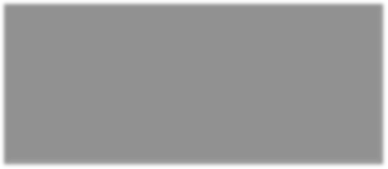
### Entities

* + An entity can be either a living or non-living component.
  + It showcases an entity as a rectangle in an ER diagram.
  + For example, in a student study course, both the student and the course are entities.



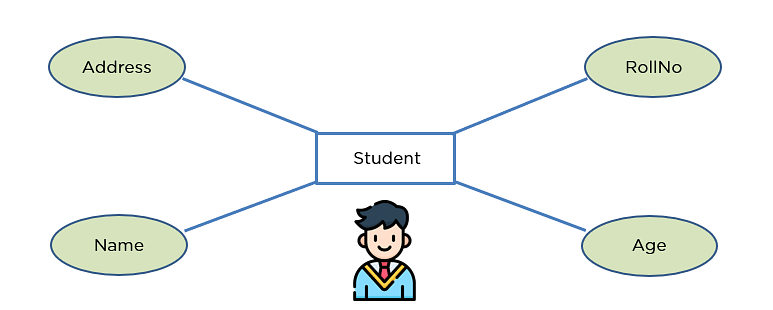
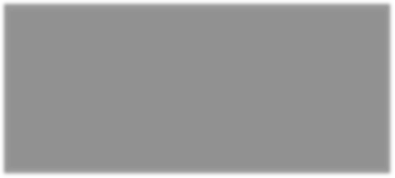
### a. Weak Entity

* + An entity that makes reliance over another entity is called a weak entity
  + You showcase the weak entity as a double rectangle in ER Diagram.
  + In the example below, school is a strong entity because it has a primary key attribute - school number. Unlike school, the classroom is a weak entity because it does not have any primary key and the room number here acts only as a discriminator.



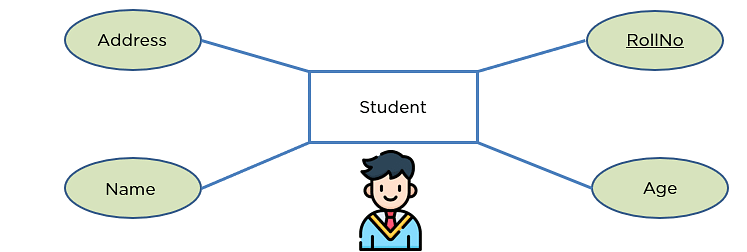
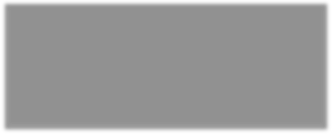
### Attribute

* + An attribute exhibits the properties of an entity.
  + You can illustrate an attribute with an oval shape in an ER diagram.



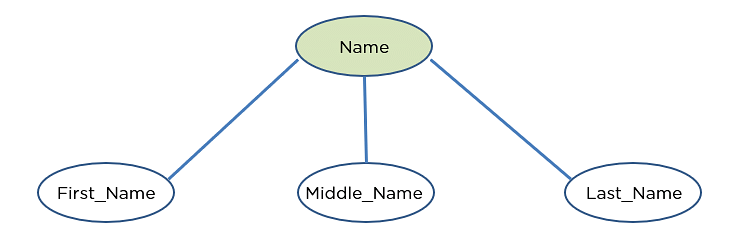
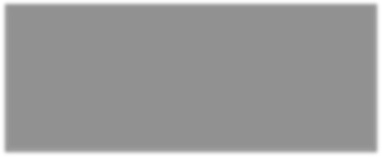
### Key Attribute

* + Key attribute uniquely identifies an entity from an entity set.
  + It underlines the text of a key attribute.
  + For example: For a student entity, the roll number can uniquely identify a student from a set of stu- dents.



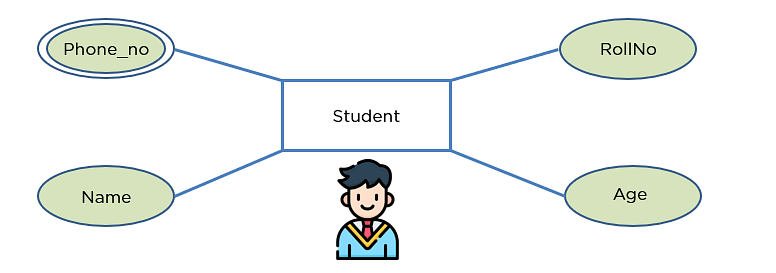
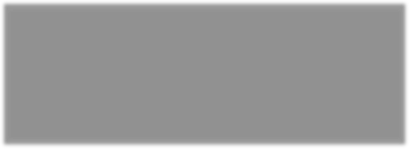
### Composite Attribute

* + An attribute that is composed of several other attributes is known as a composite attribute.
  + An oval showcases the composite attribute, and the composite attribute oval is further connected with other ovals.



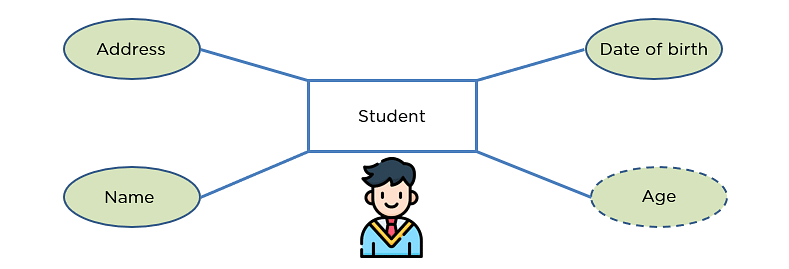
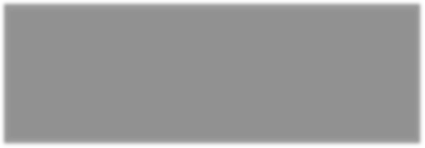
### Multivalued Attribute

* + Some attributes can possess over one value, those attributes are called multivalued attributes.
  + The double oval shape is used to represent a multivalued attribute.



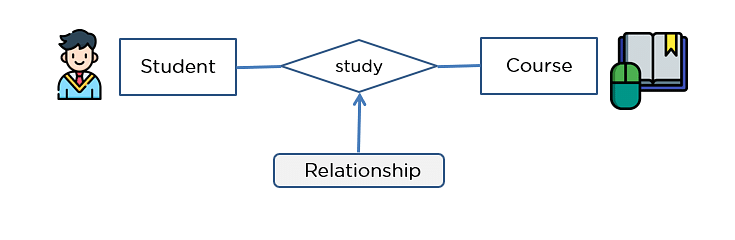
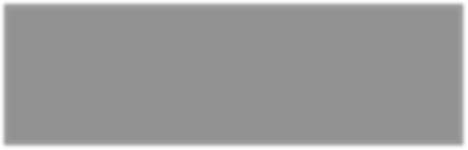
### Derived Attribute

* + An attribute that can be derived from other attributes of the entity is known as a derived attribute.
  + In the ER diagram, the dashed oval represents the derived attribute.



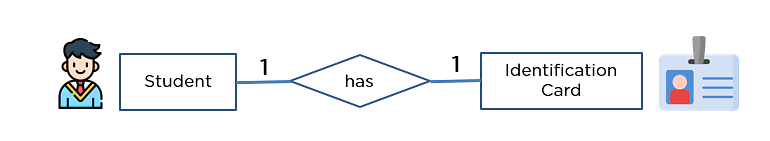
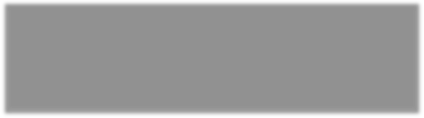
### Relationship

* + The diamond shape showcases a relationship in the ER diagram.
  + It depicts the relationship between two entities.
  + In the example below, both the student and the course are entities, and study is the relationship be- tween them.



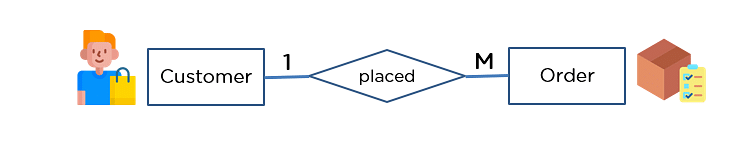
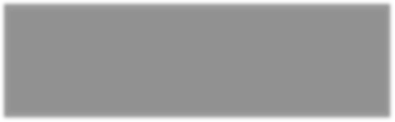
### One-to-One Relationship

* + When a single element of an entity is associated with a single element of another entity, it is called a one-to-one relationship.
  + For example, a student has only one identification card and an identification card is given to one per- son.



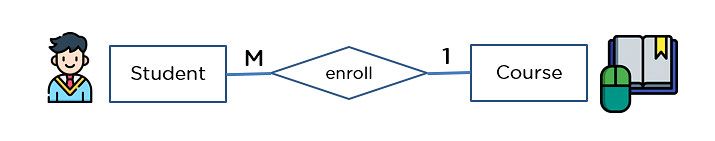
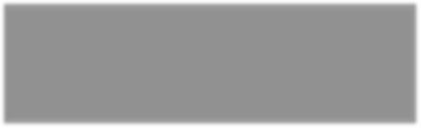
### One-to-Many Relationship

* + When a single element of an entity is associated with more than one element of another entity, it is called a one-to-many relationship
  + For example, a customer can place many orders, but an order cannot be placed by many customers.



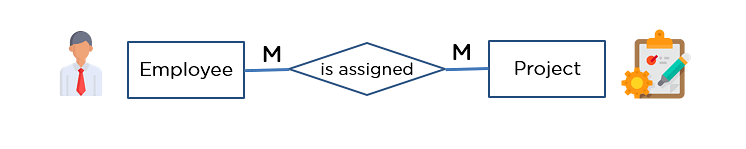
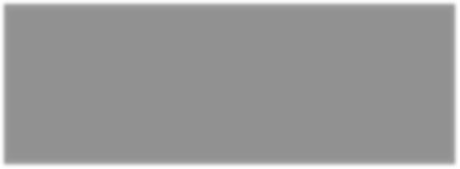
### Many-to-One Relationship

* + When more than one element of an entity is related to a single element of another entity, then it is called a many-to-one relationship.
  + For example, students have to opt for a single course, but a course can have many students.



### Many-to-Many Relationship

* + When more than one element of an entity is associated with more than one element of another entity, this is called a many-to-many relationship.
  + For example, you can assign an employee to many projects and a project can have many employees.



### How to Draw an ER Diagram?

Below are some important points to draw ER diagram:

* First, identify all the Entities. Embed all the entities in a rectangle and label them properly.
* Identify relationships between entities and connect them using a diamond in the middle, illustrating the relationship. Do not connect relationships with each other.
* Connect attributes for entities and label them properly.
* Eradicate any redundant entities or relationships.
* Make sure your ER Diagram supports all the data provided to design the database.
* Effectively use colors to highlight key areas in your diagrams.

## CONCLUSION:

ER Diagram in DBMS is widely used to describe the conceptual design of databases. It helps both users and database developers to preview the structure of the database before implementing the database

## QUESTIONS:

1. Define what an entity is in the context of a database and provide examples.
2. Explain the concept of relationships in a database. What are the different types of relationships that can exist between entities?
3. Describe the components of an ER diagram and their significance.
4. Discuss the difference between an entity set and an entity instance.
5. What is cardinality in the context of ER diagrams? How is it represented in the diagram?
6. Explain the terms "weak entity" and "strong entity" in the context of ER modeling.
7. Describe the process of converting an ER diagram into a relational schema?
8. What is an attribute in an ER diagram? Provide examples of different types of attributes?
9. Discuss the importance of normalization in database design and its relation to ER modeling?



# SCHOOL OF ENGINEERING AND TECHNOLOGY

## DEPARTMENT OF AIML/AIDS ENGINEERING

**=================================================================================**

**Case Study: 03**

**Name of Student:**

**Roll Number: Class:**

**TITLE: Employee database system AIM**:

1. Create Table with primary key and foreign key constraints. a. Alter table with add n modify b. Drop table
2. Perform following SQL queries on the database created in assignment 1.
   * Implementation of relational operators in SQL
   * Boolean operators and pattern matching
   * Arithmetic operations and built in functions
   * Group functions
   * Processing Date and Time functions
   * Complex queries and set operators
3. Execute DDL/DML statements which demonstrate the use of views. Update the base table using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables**.**
4. Write and execute PL/SQL stored procedure and function to perform a suitable task on the database. Demonstrate its use.
5. Write and execute suitable database triggers. Consider row level and statement level triggers

**OBJECTIVE:**

To provide a strong formal foundation in database concepts, recent technologies and best industry practices. To give systematic database design approaches covering conceptual design, logical design and an overview of physical design. To give systematic database design approaches covering conceptual design, logical design and an overview of physical design. To learn and understand various Database Architectures and its use for application development.

## SOFTWARE REQUIREMENTS:

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Name of Software and Hardware** | **Latest Version** |
| 1 | SQL Server 2022 | 16.0.4105.2 |
| 2 | CPU, RAM, Hard drive, Network adapter | 4 cores, 2500 MHz, 8 GB,  300 GB, RAID  recommended, 1 Gbit |

**THEORY:**

### Primary Key

A primary key is a column or a set of columns in a table that uniquely identifies each row in the table. It must contain unique values and cannot have NULL values. The primary key constraint ensures data integrity by enforcing uniqueness and providing a way to identify each record uniquely. In most database systems, a primary key is automatically indexed, which helps in faster retrieval of data.

**Primary Key Syntax**

In SQL, when creating a table, you can define a primary key using the PRIMARY KEY constraint. Here's the syntax for creating a table with a primary key:

CREATE TABLE Students ( StudentID INT PRIMARY KEY, Name VARCHAR(50),

Age INT

);

### Foreign Key

A foreign key is a column or a set of columns in a table that establishes a link between data in two tables. It creates a relationship between two tables by referencing the primary key of another table. The foreign key constraint ensures referential integrity by enforcing that the values in the foreign key column must exist in the referenced table's primary key column or be NULL. This constraint helps maintain consistency and data integrity across related tables in a database.

When creating a table with a foreign key constraint that references another table, you can use the FOREIGN KEY constraint. Here's an example of defining a foreign key in SQL:

CREATE TABLE Orders ( OrderID INT PRIMARY KEY,

CustomerID INT, OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID)

);

In this example, the Orders table has a foreign key constraint on the CustomerID column, which references the CustomerID column in the Customers table. This establishes a relationship between the Orders and Customers tables based on the CustomerID column.

### \*\*Creating Table with Primary Key and Foreign Key Constraints\*\*

To create a table with primary key and foreign key constraints, you can use the following SQL syntax:

CREATE TABLE Orders ( OrderID INT PRIMARY KEY,

CustomerID INT, OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID)

);

In this example, the `Orders` table has a primary key constraint on the `OrderID` column and a foreign key constraint on the `CustomerID` column referencing the `CustomerID` column in the `Customers` table.

### \*\*Altering Table with ADD and MODIFY\*\*

To alter a table by adding a new column or modifying an existing column, you can use the following SQL syntax:

-- Add a new column ALTER TABLE Orders

ADD TotalAmount DECIMAL (10, 2);

-- Modify an existing column ALTER TABLE Orders

MODIFY OrderDate DATETIME;

In this example, we add a new column `TotalAmount` to the `Orders` table and modify the data type of the

`OrderDate` column.

### \*\*Dropping Table\*\*

To drop a table from the database, you can use the following SQL syntax:

DROP TABLE Orders;

This statement will delete the `Orders` table and all its data from the database.

## THEORY:

In a database management system (DBMS) like SQL, operators are used to perform operations on data. Here are some common SQL operators along with examples and syntax:

### Arithmetic Operators:

* + Addition (+): Adds two values.

Example: SELECT column1 + column2 FROM table\_name;

* + Subtraction (-): Subtracts one value from another. Example: SELECT column1 - column2 FROM table\_name;
  + Multiplication (\*): Multiplies two values.

Example: SELECT column1 \* column2 FROM table\_name;

* + Division (/): Divides one value by another.

Example: SELECT column1 / column2 FROM table\_name;

### Comparison Operators:

* + Equal to (=): Compares if two values are equal.

Example: SELECT \* FROM table\_name WHERE column1 = value;

* + Not equal to (!= or <>): Compares if two values are not equal. Example: SELECT \* FROM table\_name WHERE column1 != value;
  + Greater than (>): Compares if one value is greater than another. Example: SELECT \* FROM table\_name WHERE column1 > value;
  + Less than (<): Compares if one value is less than another.

Example: SELECT \* FROM table\_name WHERE column1 < value;

### Logical Operators:

* + AND: Returns true if both conditions are true.

Example: SELECT \* FROM table\_name WHERE condition1 AND condition2;

* + OR: Returns true if either condition is true.

Example: SELECT \* FROM table\_name WHERE condition1 OR condition2;

* + NOT: Returns true if the condition is not true.

Example: SELECT \* FROM table\_name WHERE NOT condition;

### Group Function:

In a database management system (DBMS) like SQL, group functions (also known as aggregate functions) are used to perform operations on a set of values and return a single value as a result. These functions are often used with the GROUP BY clause to group rows that have the same values into summary rows. Here are some common group functions in SQL:

1. COUNT(): This function is used to count the number of rows in a result set or the number of non-null values in a specific column.

Example: SELECT COUNT (column\_name) FROM table\_name;

1. SUM(): This function calculates the sum of values in a specific column. Example: SELECT SUM (column\_name) FROM table\_name;
2. AVG(): This function calculates the average of values in a specific column. Example: SELECT AVG (column\_name) FROM table\_name;
3. MAX(): This function returns the maximum value in a specific column. Example: SELECT MAX (column\_name) FROM table\_name;
4. MIN(): This function returns the minimum value in a specific column. Example: SELECT MIN (column\_name) FROM table\_name;

## THEORY:

* DDL (Data Definition Language) and DML (Data Manipulation Language) statements can be used to create and manipulate views in a database.
* Views are virtual tables that display data from one or more tables based on a query.
* Here are examples of DDL and DML statements to demonstrate the use of views:

### \*\*Creating a view from a single table: \*\*

-- Create a view to display specific columns from a table CREATE VIEW employee\_view AS

SELECT emp\_id, emp\_name, emp\_salary FROM employees;

### \*\*Creating a view from multiple tables: \*\*

-- Create a view to display data from multiple tables CREATE VIEW employee\_department\_view AS SELECT e.emp\_id, e.emp\_name, d.department\_name FROM employees e

JOIN departments d ON e.department\_id = d.department\_id;

### \*\*Updating the base table using its corresponding view: \*\*

-- Update the base table through a view UPDATE employee\_view

SET emp\_salary = 60000 WHERE emp\_id = 101;

### \*\*Restrictions on updatable views: \*\*

Views can be updatable if they meet certain criteria:

* + The view must reference only one base table.
  + The view must not contain any aggregate functions (e.g., SUM, AVG).
  + The view must not contain GROUP BY or HAVING clauses.

### \*\*Advantages of using views: \*\*

* + Simplify complex queries: Views can encapsulate complex SQL queries, making them easier to use and understand.
  + Security: Views can restrict access to certain columns or rows, providing an additional layer of security.
  + Data abstraction: Views hide the complexity of the underlying database schema, allowing users to interact with a simplified representation of the data.
  + Performance optimization: Views can precompute joins or aggregations, improving query performance.

### \*\*Types of views: \*\*

* + Simple views: Views based on a single table.
  + Complex views: Views based on multiple tables or using subqueries.
  + Materialized views: Views that store the result set physically, allowing for faster data retrieval but requiring maintenance to keep them up to date.

### \*\*Creating updatable views: \*\*

* + To create an updatable view, ensure that the view meets the criteria mentioned earlier, such as referencing only one base table and not containing certain functions or clauses.
  + Updatable views allow you to insert, update, and delete data through the view, which will be reflected in the base table.

### \*\*View limitations: \*\*

* + Views cannot have an ORDER BY clause unless the view is created with the TOP clause in SQL Server.
  + Views cannot have a DISTINCT clause.
  + Views cannot have output parameters.

Views are a powerful feature in databases that enable users to customize their data access and presentation. By understanding how to create and use views effectively, you can enhance your database querying capabilities and improve data management practices. If you have any more specific questions or need further information, feel free to ask!

## THEORY:

In PL/SQL (Procedural Language/Structured Query Language), you can create stored procedures and functions to perform specific tasks in a database. Stored procedures are blocks of code that can be called to execute a series of SQL statements, while functions return a single value.

Here is an example of creating and executing a simple PL/SQL stored procedure and function to demonstrate their use:

### \*\*Creating a stored procedure: \*\*

**-- Create a stored procedure to update employee salary by a certain percentage**

CREATE OR REPLACE PROCEDURE update\_salary(p\_emp\_id IN NUMBER, p\_percentage IN NUMBER) IS

v\_new\_salary NUMBER; BEGIN

SELECT emp\_salary \* (1 + p\_percentage/100) INTO v\_new\_salary FROM employees

WHERE emp\_id = p\_emp\_id;

UPDATE employees

SET emp\_salary = v\_new\_salary WHERE emp\_id = p\_emp\_id;

COMMIT; END;

### \*\*Executing the stored procedure: \*\*

-- Call the stored procedure to update the salary of employee with ID 101 by 10% BEGIN

update\_salary(101, 10); END;

### \*\*Creating a function: \*\*

-- Create a function to calculate the total salary for a department

CREATE OR REPLACE FUNCTION get\_total\_department\_salary (p\_department\_id IN NUMBER)

RETURN NUMBER IS

v\_total\_salary NUMBER: = 0; BEGIN

SELECT SUM (emp\_salary) INTO v\_total\_salary FROM employees

WHERE department\_id = p\_department\_id;

RETURN v\_total\_salary; END;

### \*\*Executing the function: \*\*

-- Call the function to get the total salary for department with ID 1 DECLARE

v\_department\_salary NUMBER; BEGIN

v\_department\_salary := get\_total\_department\_salary(1);

DBMS\_OUTPUT.PUT\_LINE ('Total salary for department 1: ' || v\_department\_salary); END;

/

In this example, the stored procedure `update salary` updates an employee's salary based on a percentage increase, and the function `get\_total\_department\_salary` calculates the total salary for a specific department. These PL/SQL constructs can be used to perform various tasks efficiently within a database.

## THEORY:

Triggers in oracle are blocks of PL/SQL code which oracle engine can execute automatically based on some action or event.

These events can be:

### DDL statements (CREATE, ALTER, DROP, TRUNCATE) DML statements (INSERT, SELECT, UPDATE, DELETE)

* Database operation like connecting or disconnecting to oracle (LOGON, LOGOFF, SHUTDOWN)
* Triggers are automatically and repeatedly called upon by oracle engine on satisfying certain condition.
* Triggers can be activated or deactivated depending on the requirements.

### PL/SQL: Uses of Triggers

* Here we have mentioned a few use cases where using triggers proves very helpful:
* Maintaining complex constraints which is either impossible or very difficult via normal constraint (like primary, foreign, unique etc.) applying technique.
* Recording the changes made on the table.
* Automatically generating primary key values.
* Prevent invalid transactions to occur.
* Granting authorization and providing security to database.
* Enforcing referential integrity.

### PL/SQL: Parts of a Trigger

**Whenever a trigger is created, it contains the following three sequential parts:**

1. **Triggering Event or Statement:** The statements due to which a trigger occurs is called triggering event or statement. Such statements can be DDL statements, DML statements or any database operation, executing which gives rise to a trigger.
2. **Trigger Restriction:** The condition or any limitation applied on the trigger is called trigger restriction. Thus, if such a condition is TRUE then trigger occurs otherwise it does not occur.
3. **Trigger Action:** The body containing the executable statements that is to be executed when trigger occurs that is with the execution of Triggering statement and upon evaluation of Trigger restriction as True is called Trigger Action.

### Level Triggers

There are 2 different types of level triggers, they are:

## ROW LEVEL TRIGGERS

* + It fires for every record that got affected with the execution of DML statements like INSERT, UPDATE, DELETE etc. It always uses a FOR EACH ROW clause in a triggering statement.
  + Row-level triggers are fired for each row that is affected by the triggering statement (e.g., INSERT, UPDATE, DELETE).
  + These triggers can access the values of the columns in the row being processed using the: NEW.column\_name and: OLD.column\_name syntax.
  + Row-level triggers are useful when you need to perform actions based on individual rows being modified.

### Row-Level Trigger Example:

CREATE OR REPLACE TRIGGER trg\_row\_level BEFORE INSERT ON your\_table

FOR EACH ROW BEGIN

-- Trigger logic for row-level trigger

-- You can access the values using :NEW.column\_name END;

## STATEMENT LEVEL TRIGGERS

* + It fires once for each statement that is executed.
  + Statement-level triggers are fired once for each triggering SQL statement, regardless of the number of rows affected.
  + These triggers do not have access to individual row values like row-level triggers but can perform actions based on the entire statement.
  + Statement-level triggers are beneficial when you need to execute logic that applies to the whole set of affected rows.

### Statement-Level Trigger Example:

CREATE OR REPLACE TRIGGER trg\_statement\_level AFTER DELETE ON your\_table

BEGIN

-- Trigger logic for statement-level trigger

-- You can perform actions based on the entire statement END;

### Event Triggers

There are 3 different types of event triggers, they are:

## DDL EVENT TRIGGER

It fires with the execution of every DDL statement (CREATE, ALTER, DROP, TRUNCATE).

## DML EVENT TRIGGER

It fires with the execution of every DML statement (INSERT, UPDATE, DELETE).

## DATABASE EVENT TRIGGER

It fires with the execution of every database operation which can be LOGON, LOGOFF, SHUTDOWN, SERVERERROR etc.

### Timing Triggers

There are 2 different types of timing triggers, they are:

## BEFORE TRIGGER

It fires before executing DML statement.

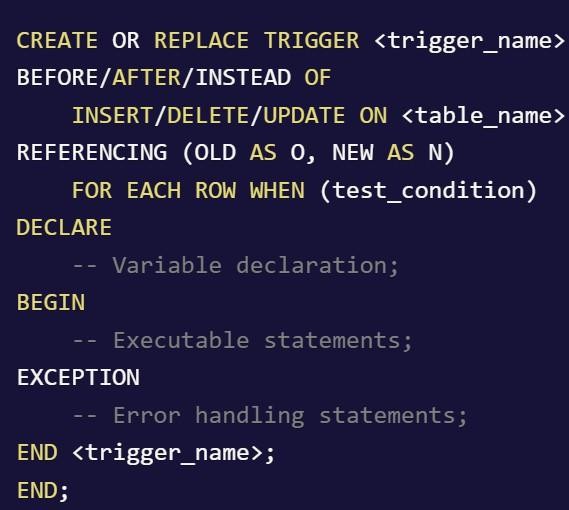
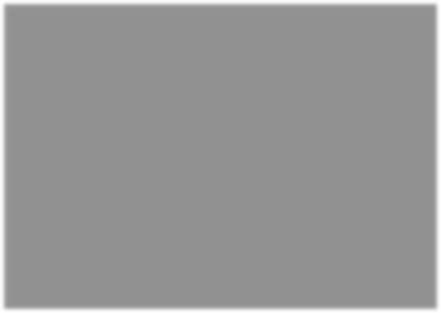
Triggering statement may or may not executed depending upon the before condition block.

## AFTER TRIGGER

It fires after executing DML statement.

**Syntax for creating Triggers**

**Following is the syntax for creating a trigger:**



**Were,**

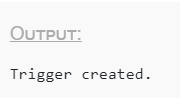
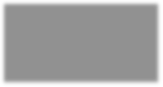
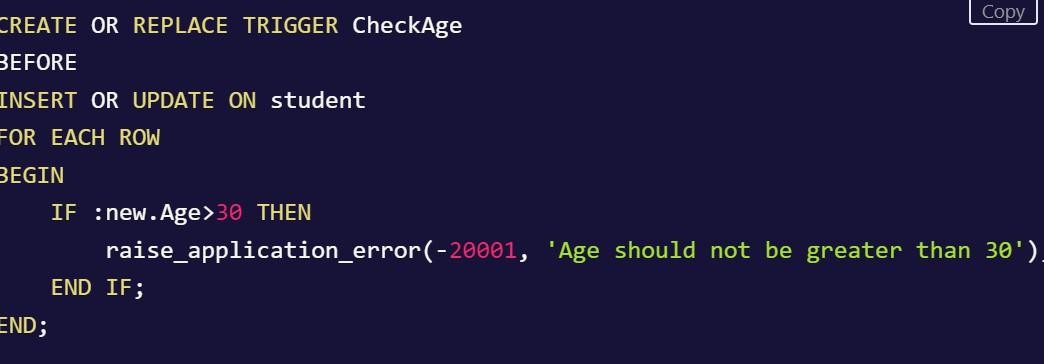
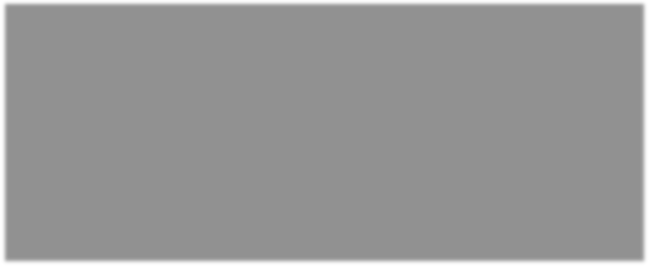
* **CREATE OR REPLACE TRIGGER** is a keyword used to create a trigger and <trigger\_name> is user-defined where a trigger can be given a name.
* **BEFORE/AFTER/INSTEAD OF** specify the timing of the trigger's occurrences. INSTEAD OF is used when a view is created.
* **INSERT/UPDATE/DELETE** specify the DML statement.

### <table\_name> specify the name of the table on which DML statement is to be applied.

* **REFERENCING** is a keyword used to provide reference to old and new values for DML statements.
* **FOR EACH ROW** is the clause used to specify row level tigger.
* **WHEN** is a clause used to specify condition to be applied and is only applicable for row-level trigger.
* **DECLARE, BEGIN, EXCEPTION, END** are the different sections of PL/SQL code block containing variable declaration, executable statements, error handling statements and marking end of PL/SQL block respectively where DECLARE and EXCEPTION part are optional.

**Time for an Example!**

**Below we have a simple program to demonstrate the use of Triggers in PL/SQL code block.**



## CONCLUSION:

In conclusion, when designing a database, it is essential to understand and implement primary key and foreign key constraints to ensure data integrity and establish relationships between tables. Hence, we conclude that all Relational Operators in DBMS are executed with all SQL Queries with the help of all DDL and DML commands. Hence successfully done multiple tables done by VIEW. So, in this Assignments we need to executed by all PLSQL procedure functions. Hence Database triggers automate tasks, ensure data integrity, and track changes in a database. Careful design and testing are essential for optimal performance.

## QUESTIONS:

1. Explain the concept of primary key and foreign key constraints in a database table.
2. Demonstrate how to create a table with primary key and foreign key constraints.
3. Discuss the significance of primary key and foreign key constraints in maintaining data integrity.
4. Explain the process of creating views using Data Definition Language (DDL) statements in SQL.
5. Discuss the types of arithmetic operations and built-in functions available in SQL for data manipulation.
6. Explain the purpose of group functions in SQL and provide examples of when they are used in queries.
7. How can date and time functions be utilized in SQL for processing date-related data?
8. Explain the purpose of PL/SQL stored procedures and functions in database programming.
9. Differentiate between a stored procedure and a function in PL/SQL, highlighting their key characteristics.
10. Describe the usage of complex queries in SQL and the role of set operators in combining query results.
11. How many types of triggers exist in PL SQL?
12. Which type of trigger uses the old and new qualifiers?
13. What is the difference between trigger new and trigger old?
14. What is a row-level trigger in PL SQL?
15. What is a trigger? Explain with an example?
16. What are the after triggers?



# SCHOOL OF ENGINEERING AND TECHNOLOGY

## DEPARTMENT OF AIML/AIDS ENGINEERING

**=================================================================================**

**Case Study: 04**

**Name of Student:**

**Roll Number: Class:**

**TITLE: Employee Database design AIM**:

1. Design and implement a database (for assignment no 1) using DDL statements and apply normalization on them.
2. Design and implement a database using joins operations.

## OBJECTIVE:

To give systematic database design approaches covering conceptual design, logical design and an overview of physical design.

## SOFTWARE REQUIREMENTS:

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Name of Software and Hardware** | **Latest Version** |
| 1 | SQL Server 2022 | 16.0.4105.2 |
| 2 | CPU, RAM, Hard drive, Network adapter | 4 cores, 2500 MHz, 8 GB,  300 GB, RAID  recommended, 1 Gbit |

**THEORY:**

In database design, normalization is the process of organizing data in a database efficiently. The main objective of normalization is to reduce data redundancy and improve data integrity. There are different normal forms that help in achieving this goal:

1. **First Normal Form (1NF):** Ensures that each column in a table contains atomic values, and there are no repeating groups of columns.
2. **Second Normal Form (2NF):** In addition to being in 1NF, all non-key attributes are fully functional dependent on the primary key. This means that no partial dependencies exist.
3. **Third Normal Form (3NF):** In addition to being in 2NF, there should be no transitive dependencies. This means that non-key attributes should not depend on other non-key attributes.
4. **Boyce-Codd Normal Form (BCNF):** A stronger version of 3NF where every determinant is a candidate key.
5. **Fourth Normal Form (4NF):** Deals with multi-valued dependencies.
6. **Fifth Normal Form (5NF):** Deals with join dependencies.

By applying normalization techniques, you can ensure that your database schema is well-structured, efficient, and free from anomalies such as insertion, update, and deletion anomalies. Each normal form serves a specific purpose in eliminating different types of redundancies and dependencies in the database design.

When implementing a database using Data Definition Language (DDL) statements, you can apply normalization principles by structuring your tables appropriately. Here is an example of how you can design tables and apply normalization using DDL statements:

### Let's consider an example where we have two entities: Students and Courses.

1. **First Normal Form (1NF):**

Ensure that each table has a primary key.

Avoid storing multiple values in a single column. CREATE TABLE Students (

student\_id INT PRIMARY KEY, student\_name VARCHAR (50), date\_of\_birth DATE

);

CREATE TABLE Courses ( course\_id INT PRIMARY KEY, course\_name VARCHAR(50), instructor VARCHAR(50)

);

### Second Normal Form (2NF):

Ensure that all non-key attributes are fully functionally dependent on the primary key. CREATE TABLE Enrollments (

student\_id INT, course\_id INT, enrollment\_date DATE,

PRIMARY KEY (student\_id, course\_id),

FOREIGN KEY (student\_id) REFERENCES Students(student\_id), FOREIGN KEY (course\_id) REFERENCES Courses(course\_id)

);

### Third Normal Form (3NF):

Ensure that there are no transitive dependencies. CREATE TABLE Instructors (

instructor\_id INT PRIMARY KEY, instructor\_name VARCHAR(50)

);

ALTER TABLE Courses

ADD COLUMN instructor\_id INT, ADD CONSTRAINT fk\_instructor

FOREIGN KEY (instructor\_id) REFERENCES Instructors(instructor\_id);

### Boyce-Codd Normal Form (BCNF):

Ensure that every determinant is a candidate key.

CREATE TABLE Departments ( department\_id INT PRIMARY KEY, department\_name VARCHAR(50)

);

CREATE TABLE Courses ( course\_id INT PRIMARY KEY, course\_name VARCHAR(50), department\_id INT,

FOREIGN KEY (department\_id) REFERENCES Departments(department\_id)

);

### Fourth Normal Form (4NF):

Address multi-valued dependencies.

CREATE TABLE StudentAddresses ( student\_id INT,

address\_type VARCHAR(10), address VARCHAR(100),

PRIMARY KEY (student\_id, address\_type),

FOREIGN KEY (student\_id) REFERENCES Students(student\_id)

);

### Fifth Normal Form (5NF):

Handle join dependencies.

CREATE TABLE CourseInstructors ( course\_id INT,

instructor\_id INT,

PRIMARY KEY (course\_id, instructor\_id),

FOREIGN KEY (course\_id) REFERENCES Courses(course\_id), FOREIGN KEY (instructor\_id) REFERENCES Instructors(instructor\_id)

);

By incorporating these additional tables and relationships in your database schema using DDL statements, you can further normalize your database design to eliminate various types of redundancies and dependencies. This structured approach will help in maintaining data integrity and efficiency within your database system.

### Different Types of SQL Joins

**SQL joins** allow you to combine data from multiple tables based on related columns. There are different types of joins, each serving a specific purpose in querying data. Understanding how each join works is key to optimizing database queries for various scenarios.

In this [SQL Server Tutorial,](https://www.scholarhat.com/tutorial/sqlserver)we will explore different types of SQL Joins and how they work.

### What is SQL Joins

SQL Joins are used to fetch/retrieve data from two or more data tables, based on a join condition. A join condition is a relationship among some columns in the data tables that take part in SQL join. Database tables are related to each other with SQL keys. We use this key relationship in SQL Joins.Also, refer to the article SQL Joins with C# LINQ.

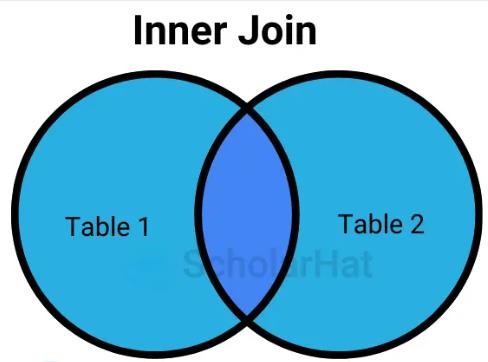
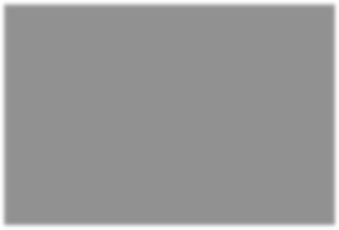
### Types of SQL Joins

**In SQL, there are several types of joins:**

1. Inner Join / Simple Join
2. Left Outer Join / Left Join
3. Right Outer Join / Right Join
4. Full Outer Join
5. Cross Join
6. Self-Join

## INNER JOIN

The inner join in SQL is used to select all rows or columns that match in both tables or as long as the SQL condition is valid.

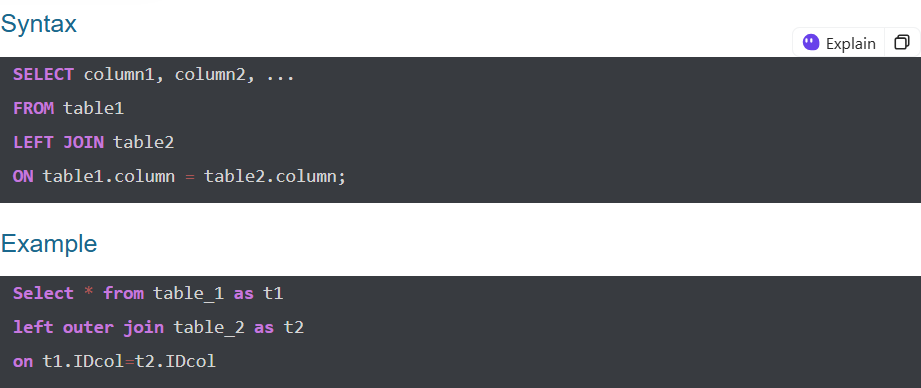


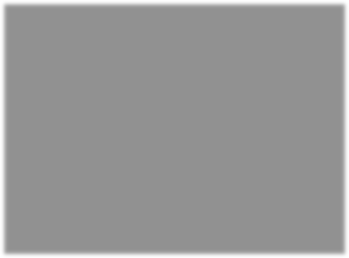
This query joins all columns from table\_1 and table\_2 where the ID col column values match, effectively combining related information from both tables.



1. **LEFT OUTER JOIN / LEFT JOIN**

The LEFT JOIN in SQL retrieves all data from the left table (table1) and the rows or columns that match from the right table (table2). If neither table contains any matched rows or columns, it returns NULL.

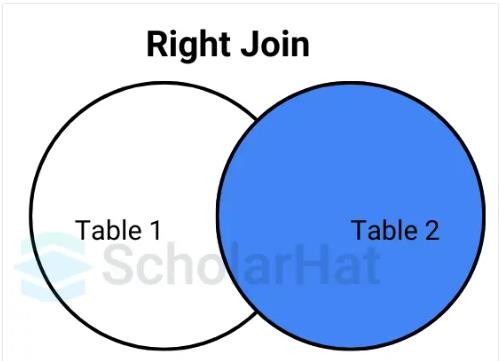
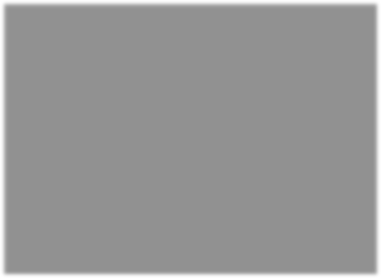


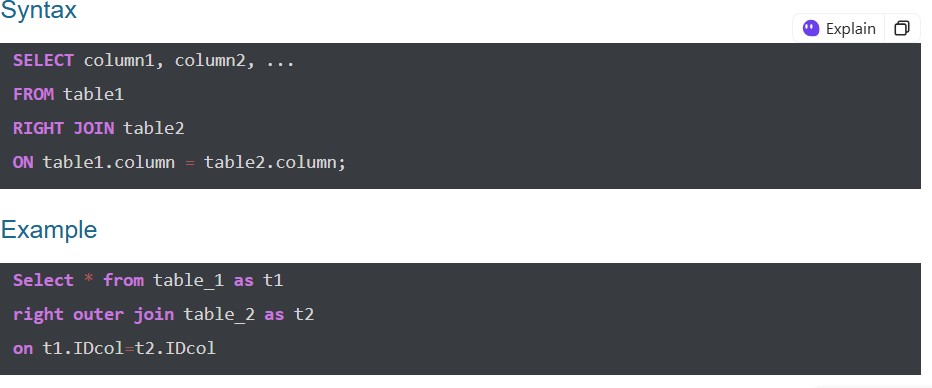


This query combines all columns from table\_1 and any matching columns from table\_2 based on ID col, retaining all rows from table\_1 even if no match exists in table\_2.

## RIGHT JOIN / RIGHT OUTER JOIN

The RIGHT JOIN retrieves all data from the right table (table 2) as well as the matching rows or columns from the left table (table 1). If neither table contains any matched rows or columns, it returns NULL.

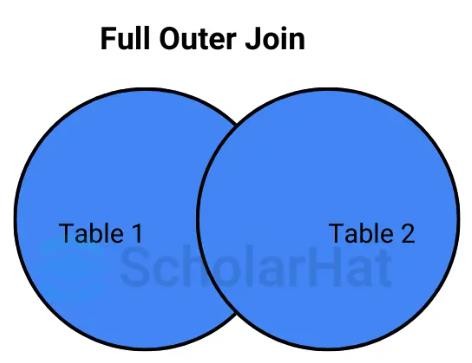
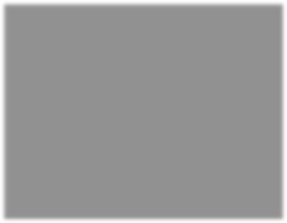


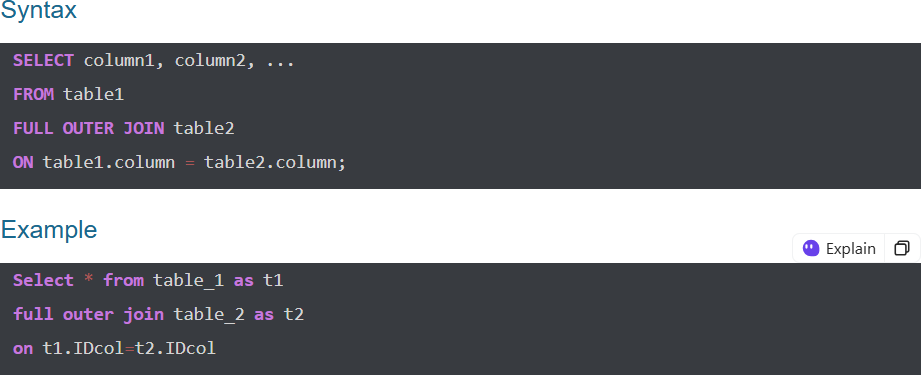


This query merges all columns from table\_1 & table\_2 where ID col values match, ensuring that all rows from table\_2 are included even if no matches exist in table\_1.

## FULL OUTER JOIN

It is a result set that combines both LEFT JOIN & RIGHT JOIN. The connected tables return all records from both tables and place NULL if no matches are found in the table. It is also known as a [FULL OUTER](https://www.scholarhat.com/tutorial/sqlserver/full-outer-join-in-sql) [JOIN.](https://www.scholarhat.com/tutorial/sqlserver/full-outer-join-in-sql)





This query includes all rows and columns from table\_1 and table\_2, matching them based on IDcol wherever feasible and filling in missed matches with null values to guarantee that everything is included.

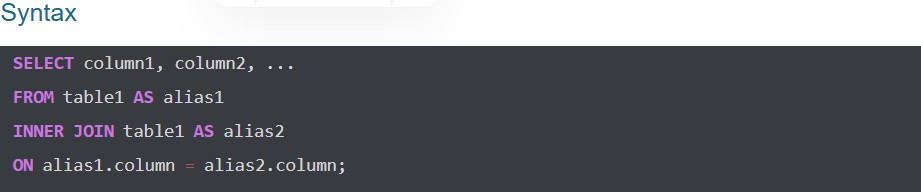
## CROSS JOIN

CARTESIAN JOIN, which returns the Cartesian product of two or more connected tables, is another name for it. The CROSS JOIN creates a table that merges each row from the first table with each row from the second table. There is no need to provide any conditions in CROSS JOIN.



## SELF JOIN

It is a SELF JOIN that was used to build a table by combining two tables. It names at least one table temporarily in an SQL statement.



## CONCLUSION:

In conclusion, when designing and implementing a database using Data Definition Language (DDL) statements and applying normalization techniques, it is essential to follow a structured approach to ensure data integrity and efficiency. By organizing data into tables and defining relationships between them, you can reduce data redundancy and eliminate anomalies in your database design.

## QUESTIONS:

1. What are the different types of database normalization?
2. How can I create a relationship between two tables in a database?
3. What is BCNF?
4. Write a Syntax of DDL statements?
5. Define the process of designing and implementing a database using (DDL) statements.



# SCHOOL OF ENGINEERING AND TECHNOLOGY

## DEPARTMENT OF AIML/AIDS ENGINEERING

**===========================================================================**

**Case Study: 05**

**Name of Student:**

**Roll Number: Class:**

**TITLE: Banking Transaction**

**AIM**: Design and implement a database using Banking Transaction.

## OBJECTIVE:

Apply ACID properties for transaction management and concurrency control.

## SOFTWARE REQUIREMENTS:

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Name of Software and Hardware** | **Latest Version** |
| 1 | SQL Server 2022 | 16.0.4105.2 |
| 2 | CPU, RAM, Hard drive, Network adapter | 4 cores, 2500 MHz, 8 GB,  300 GB, RAID  recommended, 1 Gbit |

**THEORY:**

### Transaction in DBMS

Transactions in Database Management Systems (DBMS) are sets of operations performed to modify data, including insertion, updates, or deletions. These transactions have various states indicating their progress and required actions. They ensure data consistency even during system failures, demonstrating a key advantage of DBMS. Transactions, executed repeatedly, like ATM withdrawals, generate multiple instances, each maintaining database integrity through specific properties.

We can define a transaction as a group of tasks in DBMS. Here a single task refers to a minimum processing unit, and we cannot divide it further. Now let us take the example of a certain simple transaction. Suppose any worker transfers Rs 1000 from X’s account to Y’s account. This given small and simple transaction involves various low-level tasks.

## EXAMPLE:

**X’s Account** Open\_Account(X) Old\_Bank\_Balance = X.balance

New\_Bank\_Balance = Old\_Bank\_Balance – 1000 A.balance = New\_Bank\_Balance Close\_Bank\_Account(X)

**Y’s Account** Open\_Account(Y) Old\_Bank\_Balance = Y.balance

New\_Bank\_Balance = Old\_Bank\_Balance + 1000 B.balance = New\_Bank\_Balance Close\_Bank\_Account(Y)

## ACID PROPERTIES

The transaction refers to a small unit of any given program that consists of various low-level tasks. Every transaction in DBMS must maintain ACID – A (Atomicity), C (Consistency), I (Isolation), D (Durability). One must maintain ACID so as to ensure completeness, accuracy, and integrity of data.

### Atomicity

The property of atomicity states that we must treat any given transaction as an atomic unit. It means that either all or none of its operations need to be executed. One must ensure that there is no state in the

database in which a transaction happens to be left partially completed. One must either define the states before or after the execution/failure/abortion of the transaction.

### Consistency

The property of consistency states that the database must always remain in a consistent state after any transaction. Thus, a transaction must never have any damaging effect on the data and information that resides in the database. In case, before the execution of a transaction, the database happens to be in a consistent state, then it has to remain consistent even after the transaction gets executed.

### Durability

The property of durability states that any given database must be durable enough to all of its latest updates, and it must happen even if the system suddenly restarts or fails. The database would hold the modified data in case a transaction updates and commits some chunk of information in the database. In case a transaction commits and yet the system fails before we write the data on the disk, then the information would be actually updated after the system springs back into action.

### Isolation

The property of isolation states that when multiple transactions are being simultaneously executed and in parallel in a database system, then the carrying out and execution of the transaction would occur as if it is the only transaction that exists in the system. None of the transactions would affect any other transaction’s existence.

### Serializability

Whenever the operating system executes multiple transactions in a multiprogramming environment, then there is always a possibility that instructions of one transaction interleave with some other transaction.

### Schedule

A schedule refers to a chronological execution sequence of a given transaction. Any schedule can have multiple transactions in it, and each comprises a number of tasks/instructions.

### Serial Schedule

It refers to that schedule in which the transactions are aligned in a way that just one of the transactions is executed first. Whenever the cycle completion of the first transaction occurs, the execution of the next transaction also occurs. Thus, the transactions are ordered here one after another. Such a type of schedule is

known as a serial schedule since the execution of the transactions occurs serially.

Serial schedules are like a benchmark in multi-transaction environments. We cannot change the execution sequence of instructions in a transaction, but we can execute the instructions of two transactions in a random fashion. Such an execution does not cause any harm if the two transactions are independent (mutually) and work on different data segments. But in case both of these transactions work on the same data segment, then its results may vary. These ever-varying results may bring the database to a state that is inconsistent.

If we want to resolve this issue, we allow execution of the transaction schedule parallelly in case the transactions are serializable or if we have an equivalence relation among them.

### Equivalence Schedules

The equivalence schedules can be of the given types −

### Result Equivalence

When the execution of two schedules produces the same result, they are considered to be equivalent. This type of equivalence is not considered to be very significant in general since they may yield a similar result for some of the values and different ones for others.

### View Equivalence

Two given schedules are considered to be in view equivalence when the transactions present in both of these schedules perform a similar kind of action in a similar manner.

Now, for example,

In case T reads the data initially in S1, then it will also read the data initially in S2.

In case T reads the value that is written by J in S1, then it will also read the value that is written by J in S2. If the final write on a data value in S1 is performed by T, then it will also perform the final write on a data value in S2.

Conflict Equivalence

The two given schedules would be conflicting in case they have these given properties − They both access a similar data item.

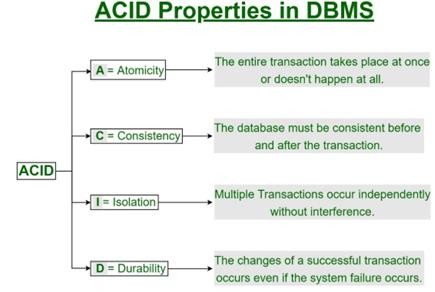
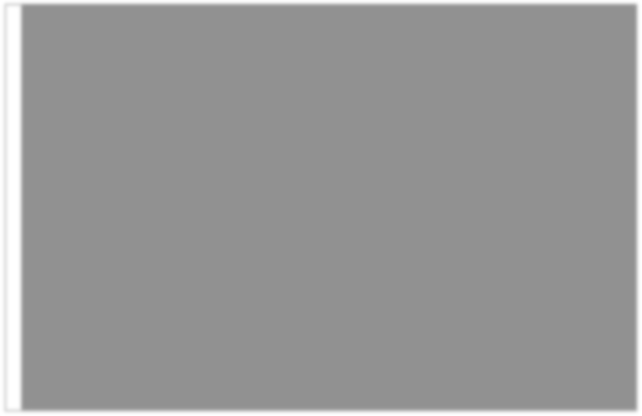
They both belong to different transactions. At least one of these is the “write” operation.

The two schedules that have multiple transactions and conflicting operations are conflict equivalent, if –

Both of these schedules consist of a similar set of Transactions.

We maintain the order of conflicting operation pairs in both schedules.

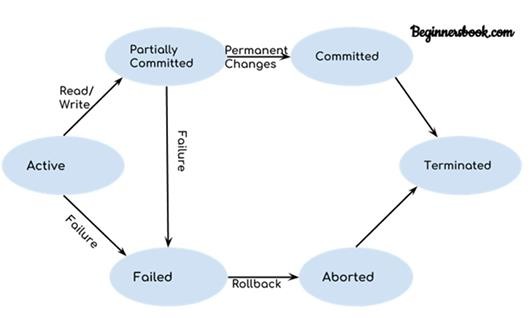
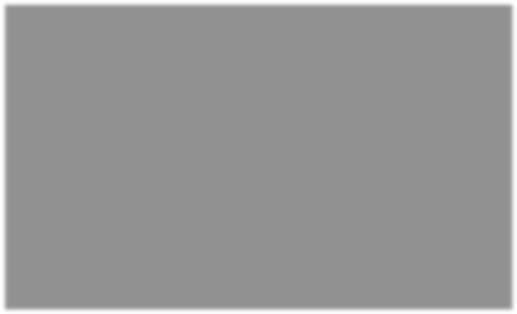
Note − The view equivalent schedules happen to be view serializable, while the conflict equivalent schedules happen to be conflict serializable. Also, all the conflict serializable schedules happen to be viewed as serializable as well.



### DBMS Transaction States

In this guide, we will discuss the states of a transaction in DBMS. A transaction in DBMS can be in one of the following states.

### DBMS Transaction States Diagram



Let’s discuss these states one by one. Active State As we have discussed in the DBMS transaction introduction that a transaction is a sequence of operations.

If a transaction is in execution, then it is said to be in active state. It doesn’t matter which step is in execution, until unless the transaction is executing, it remains in active state. Failed State If a transaction is executing and a failure occurs, either a hardware failure or a software failure then the transaction goes into failed state from the active state.

Partially Committed State As we can see in the above diagram that a transaction goes into “partially committed” state from the active state when there are read and write operations present in the transaction. A transaction contains number of read and write operations.

Once the whole transaction is successfully executed, the transaction goes into partially committed state where we have all the read and write operations performed on the main memory (local memory) instead of the actual database.

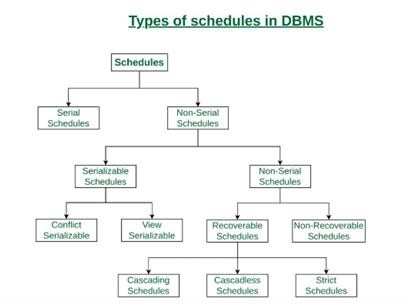
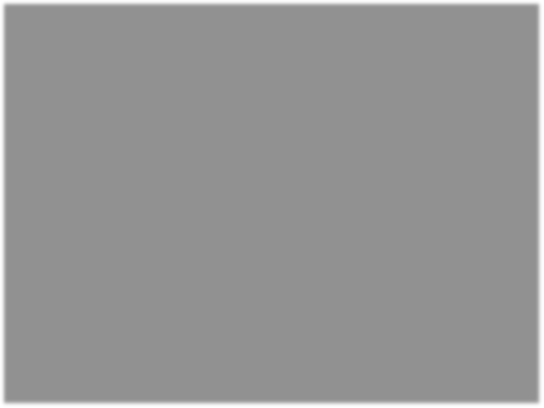
The reason why we have this state is because a transaction can fail during execution so if we are making the changes in the actual database instead of local memory, database may be left in an inconsistent state in case of any failure.

This state helps us to roll back the changes made to the database in case of a failure during execution. Committed State If a transaction completes the execution successfully then all the changes made in the

local memory during partially committed state are permanently stored in the database.

You can also see in the above diagram that a transaction goes from partially committed state to committed state when everything is successful. Aborted State As we have seen above, if a transaction fails during execution, then the transaction goes into a failed state. The changes made into the local memory (or buffer) are rolled back to the previous consistent state and the transaction goes into aborted state from the failed state. Refer the diagram to see the interaction between failed and aborted state.

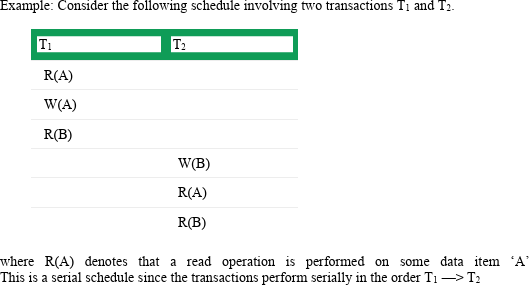
### Types of Schedules in DBMS



1. **Serial Schedules:**

Schedules in which the transactions are executed non-interleaved, i.e., a serial schedule is one in which no transaction starts until a running transaction has ended are called serial schedules. i.e., In Serial schedule, a transaction is executed completely before starting the execution of another transaction. In other words, you can say that in serial schedule, a transaction does not start execution until the currently running transaction finished execution. This type of execution of transaction is also known as noninterlaced execution. The example we have seen above is the serial schedule

.



### Non-Serial Schedule:

This is a type of Scheduling where the operations of multiple transactions are interleaved. This might lead to a rise in the concurrency problem. The transactions are executed in a nonserial manner, keeping the end result correct and same as the serial schedule. Unlike the serial schedule where one transaction must wait for another to complete all its operation, in the non-serial schedule, the other transaction proceeds without waiting for the previous transaction to complete. This sort of schedule does not provide any benefit of the concurrent transaction. It can be of two types namely, Serializable and Non-Serializable Schedule.

The Non-Serial Schedule can be divided further into Serializable and Non-Serializable.

### Serializable:

This is used to maintain the consistency of the database. It is mainly used in the non-serial scheduling to verify whether the scheduling will lead to any inconsistency or not. On the other hand, a serial schedule does not need the serializability because it follows a transaction only when the previous transaction is complete. The non-serial schedule is said to be in a serializable schedule only when it is equivalent to the serial schedules, for an n number of transactions. Since concurrency is allowed in this case thus, multiple transactions can execute concurrently.

These are of two types:

### Conflict Serializable:

A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non- conflicting operations.

### View Serializable:

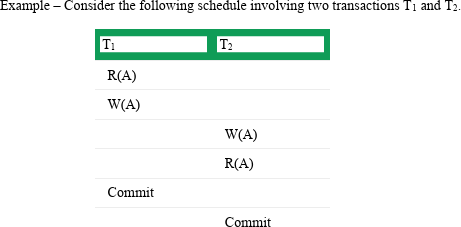
A Schedule is called view serializable if it is view equal to a serial schedule (no overlapping transactions). A conflict schedule is a view serializable but if the serializability contains blind writes, then the view serializable does not conflict serializable.

### Non-Serializable:

The non-serializable schedule is divided into two types, Recoverable and Non-recoverable Schedule.

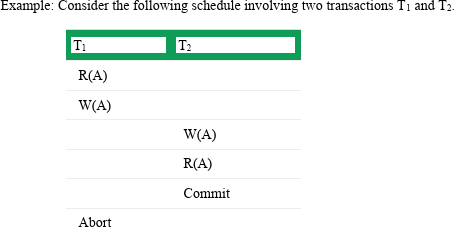
### Recoverable Schedule:

Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules. In other words, if some transaction Tj is reading value updated or written by some other transaction Ti, then the commit of Tj must occur after the commit of Ti.

This is a recoverable schedule since

T1 commits before T2, that makes the value read by T2 correct.

### Non-Recoverable Schedule:

Example: Consider the following schedule involving two transactions T1 and T2.

## CONCLUSION:

Transactions are the backbone of reliable data management in DBMS, ensuring atomicity, consistency, isolation, and durability (ACID) across operations like inserts, updates, and deletions.

## QUESTIONS:

1. Define a transaction. What are its key characteristics?
2. Explain the ACID properties with a real-world example (e.g., banking).
3. What is the difference between a transaction and a single SQL query?
4. Why is atomicity critical for financial systems?
5. Name the states of a transaction and describe the "Partially Committed" state?
6. Explain two-phase locking (2PL). Why is it called "two-phase"?
7. What happens if two transactions try to update the same data simultaneously?
8. Describe the write-ahead logging (WAL) protocol and its role in recovery?



# SCHOOL OF ENGINEERING AND TECHNOLOGY

## DEPARTMENT OF AIML/AIDS ENGINEERING

**===========================================================================**

**Case Study: 06**

**Name of Student:**

**Roll Number: Class:**

**TITLE: Banking Transaction**

**AIM**: Explain the tools in database Realm DB, ORM Lite, Couchbase Lite

## OBJECTIVE:

Analyze various database architectures and technologies.

## SOFTWARE REQUIREMENTS:

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Name of Software and Hardware** | **Latest Version** |
| 1 | SQL Server 2022 | 16.0.4105.2 |
| 2 | CPU, RAM, Hard drive, Network adapter | 4 cores, 2500 MHz, 8 GB,  300 GB, RAID  recommended, 1 Gbit |

**THEORY:**

### Realm DB

A modern, object-oriented database designed for mobile apps (iOS, Android, cross-platform) and IoT devices. It replaces SQLite with a faster, more intuitive engine.

### Key Features

* 1. **Object-Oriented Model**: Stores data as live objects, bypassing ORM mapping.
  2. **Real-Time Sync**: Built-in synchronization for offline-first apps (via Realm Sync).
  3. **Performance:** Optimized for mobile with lazy-loading and zero-copy architecture.
  4. **Cross-Platform**: Supports Swift, Kotlin, Java, JavaScript, .NET, and Flutter.
  5. **Encryption:** AES-256 encryption for data at rest.

### Use Cases

Mobile apps requiring complex data relationships (e.g., fitness trackers, social networks). Apps needing real-time collaboration or offline sync (e.g., note-taking tools).

### Limitations

Larger app size (~5–10 MB added). No direct SQL query support.

### ORM Lite

A lightweight Object-Relational Mapping (ORM) library for Java/Android, simplifying SQLite and JDBC interactions without heavyweight frameworks.

### Key Features

* 1. **Flexible Backends**: Supports SQLite, H2, MySQL, and others.
  2. **Minimal Overhead**: Small library size (<500 KB).
  3. **Raw SQL Access**: Allows direct SQL execution alongside ORM.
  4. **Simplified DAOs**: Auto-generates CRUD operations via annotations (@DatabaseTable, @DatabaseField).

### Use Cases

Android apps needing basic SQLite abstraction (e.g., todo lists, local caches). Projects requiring SQL flexibility without complex setup.

### Limitations

Manual schema migrations. No built-in sync or encryption.

### Couchbase Lite

A NoSQL embedded database for mobile and edge devices, part of the Couchbase ecosystem with sync capabilities.

### Key Features

* 1. **Document Model**: JSON-based storage with schemaless design.
  2. **Sync Gateway**: Bidirectional sync with Couchbase Server or cloud.
  3. **Query Engine**: N1QL-like queries and full-text search.
  4. **Conflict Resolution**: Automatic and custom strategies for data conflicts.
  5. **Multiplatform:** iOS, Android, Java, .NET, C.

### Use Cases

Offline-first apps with cloud sync (e.g., field service tools, retail POS). Apps handling unstructured or semi-structured data (e.g., content catalogs).

### Limitations

Steeper learning curve for NoSQL newcomers. Server setup required for sync.

## CONCLUSION:

These three database tools address distinct scenarios in mobile and embedded development:

## QUESTIONS:

1. What problem does Realm DB solve that SQLite cannot?
2. Explain the term "zero-copy architecture" in Realm DB.
3. How does ORM Lite simplify SQLite usage in Android?
4. What is the primary data model of Couchbase Lite?
5. Name one advantage of using Realm DB over traditional ORMs.
6. Compare the synchronization mechanisms of Realm DB and Couchbase Lite.
7. Why would you choose ORM Lite over Realm DB for a simple Android app?
8. How do the conflict resolution strategies differ between Realm DB and Couchbase Lite?
9. Which tool is better for unstructured data: Realm DB or Couchbase Lite? Justify.
10. Contrast the query languages used in these three tools.