

Database Management System

IT 261

Bachelor of Information Technology

Prepared by Dr. Khaled Eskaf

Lecture 3





Course Outlines

Week	Topic	Objective	MySQL Practical Activities
1	Introduction to Database Concepts	Understand basic database concepts.	Install MySQL, explore the interface, create a basic database.
2	Relational Data Model	Learn about the relational data model.	Create relational tables in MySQL.
3	Relational Integrity Rules	Understand relational integrity constraints.	Implement primary and foreign key constraints in MySQL.
4	Relational Algebra and Basic SQL	Explore relational algebra and basic SQL commands.	Use basic SQL queries in MySQL.
5	Advanced SQL Queries	Delve into complex SQL operations.	Execute advanced SQL commands in MySQL.
6	Introduction to Database Normalization	Begin learning database normalization principles.	Apply the first normal form (1NF) in MySQL.
7	Advanced Database Normalization	Explore advanced normalization techniques.	Normalize a database in MySQL to 2NF and 3NF.
8	Data Modeling and ER Diagrams	Understand data modeling and ER diagrams.	Create an ER diagram and implement it in MySQL.
9	Database Design for Real-World Apps	Apply database design to real-world scenarios.	Develop a database schema based on a real-world example.
10	Project and Review	Apply all learned concepts in a comprehensive project.	Complete a project in MySQL incorporating all learned skills.

Revision

CHAPTER 2



Week	Topic	Objective	MySQL Practical Activities
	Introduction to Database		Install MySQL, explore the
1	Concepts	Understand basic database concepts.	interface, create a basic database.
2	Relational Data Model	Learn about the relational data model.	Create relational tables in MySQL.
	Relational		

The Entity-Relationship Model

Components of ER Diagram

You base an ER Diagram on three basic concepts:

- Entities
- Attributes
 - Key Attribute
 - Composite Attribute
 - Multivalued Attribute
 - Derived Attribute
- Relationships
 - One-to-One Relationships
 - One-to-Many Relationships
 - Many-to-One Relationships
 - Many-to-Many Relationships

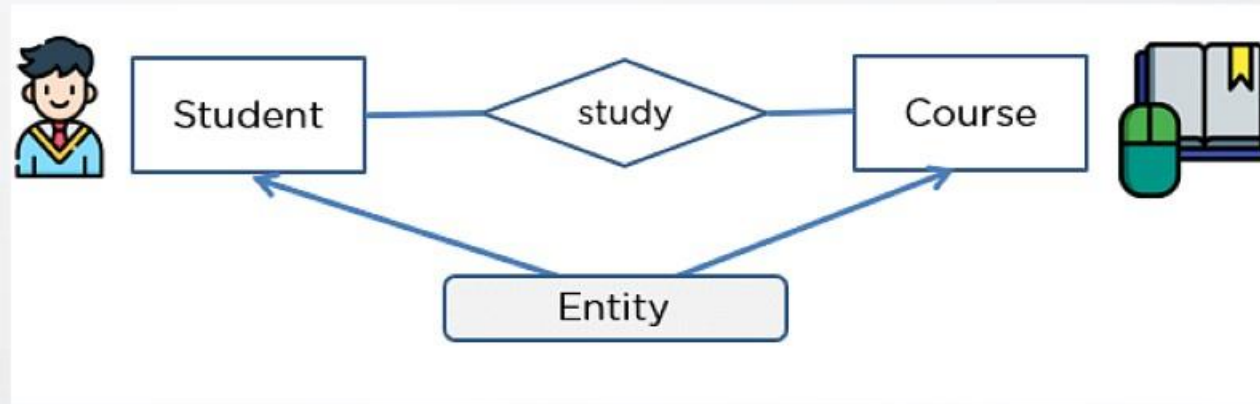
Component	Symbol
Entity	
Relationship	
Attribute	
Multivalued-Attribute	
Key Attribute	
Composite Attribute	
Weak Entity	
Weak Entity Relationship	

Entities

An entity can be either a living or non-living component.

It showcases an entity as a rectangle in an ER diagram.

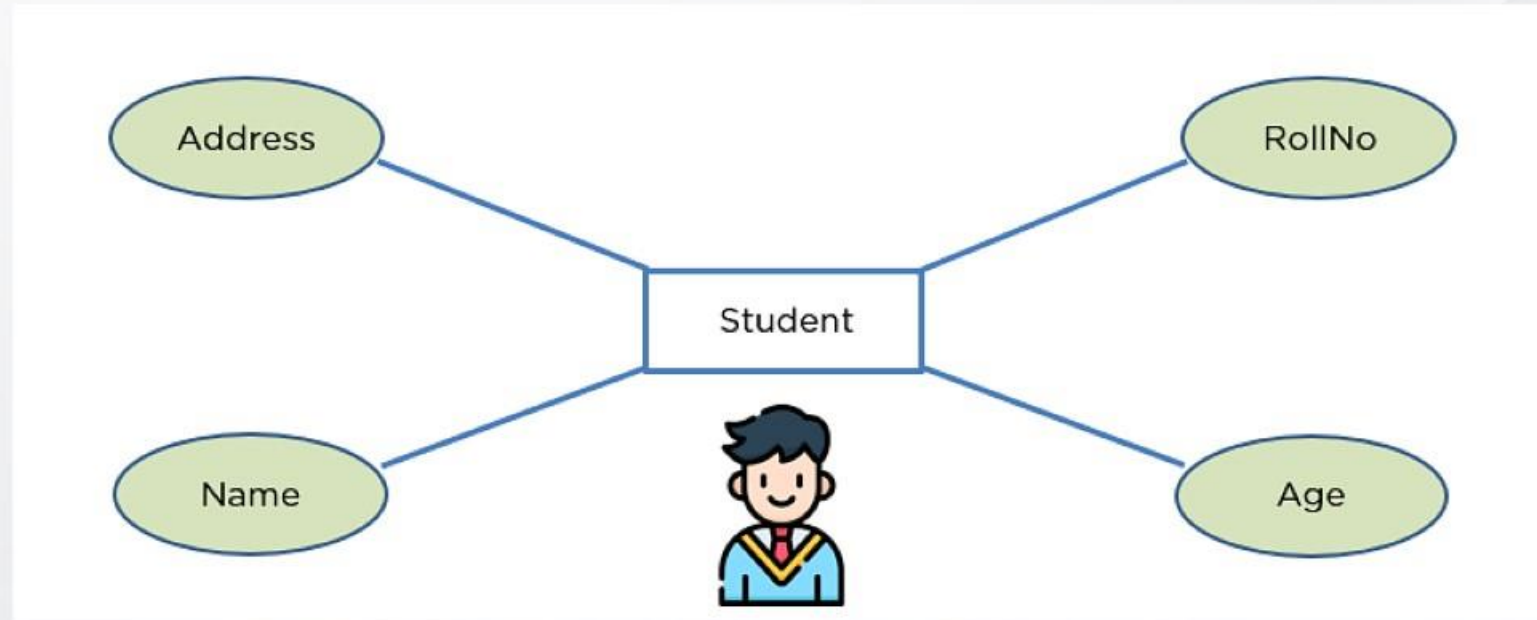
For example, in a student study course, both the student and the course are entities.



Attribute

An attribute exhibits the properties of an entity.

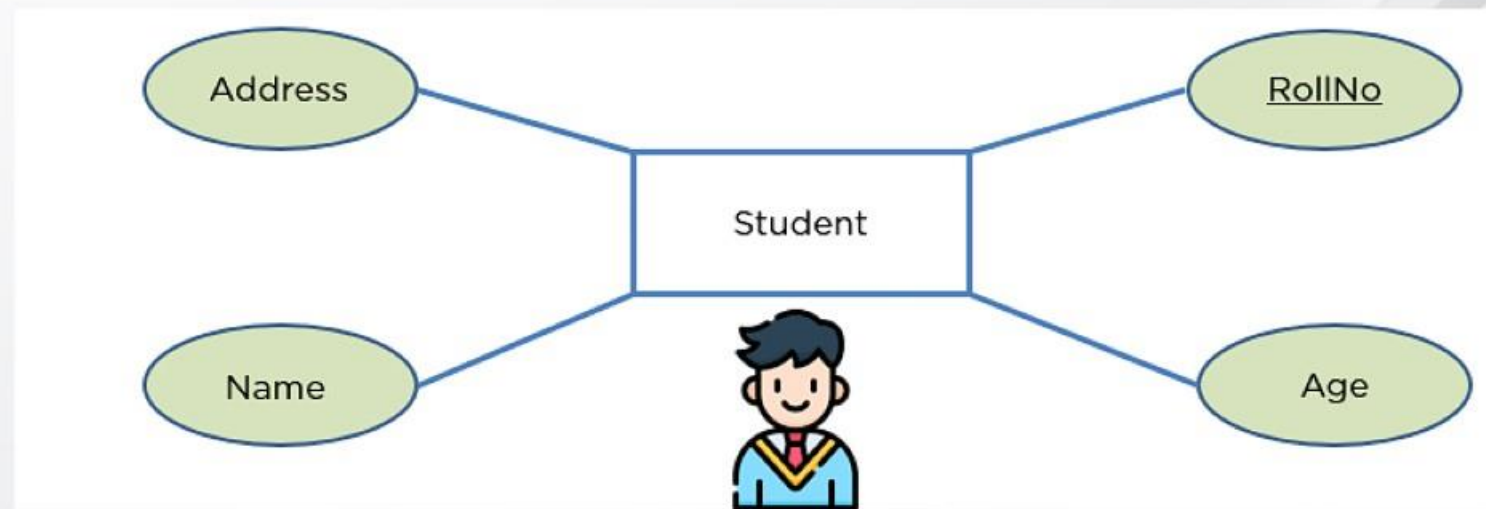
You can illustrate an attribute with an oval shape in an ER diagram.



Key Attribute

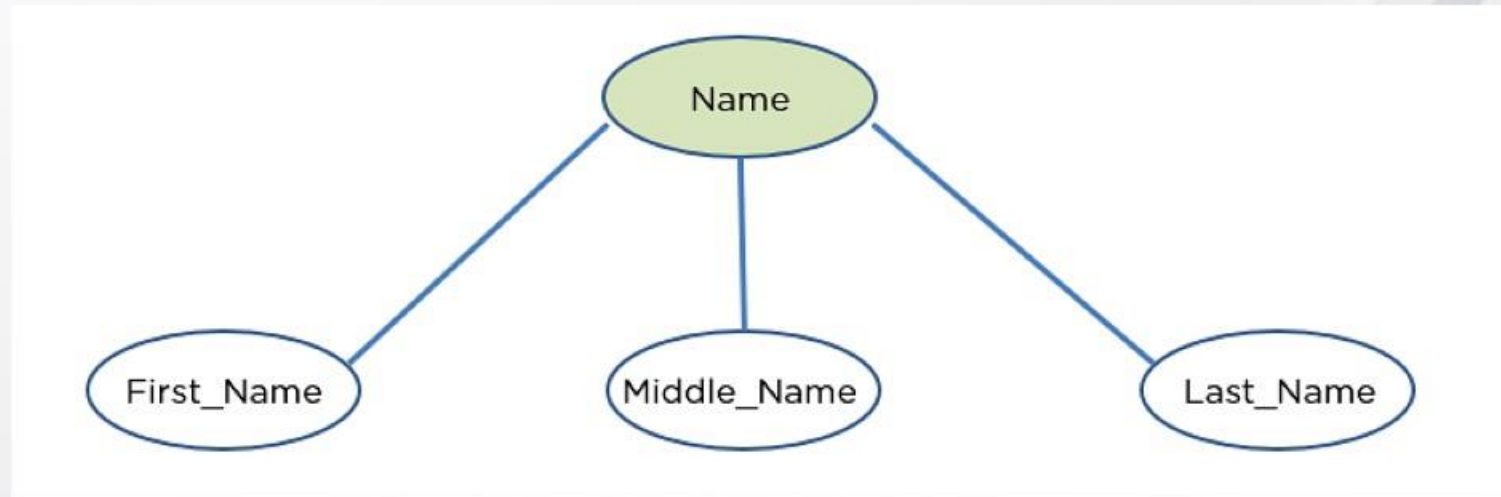
Key attribute uniquely identifies an entity from an entity set.
It underlines the text of a key attribute.

For example: For a student entity, the roll number can uniquely identify a student from a set of students.



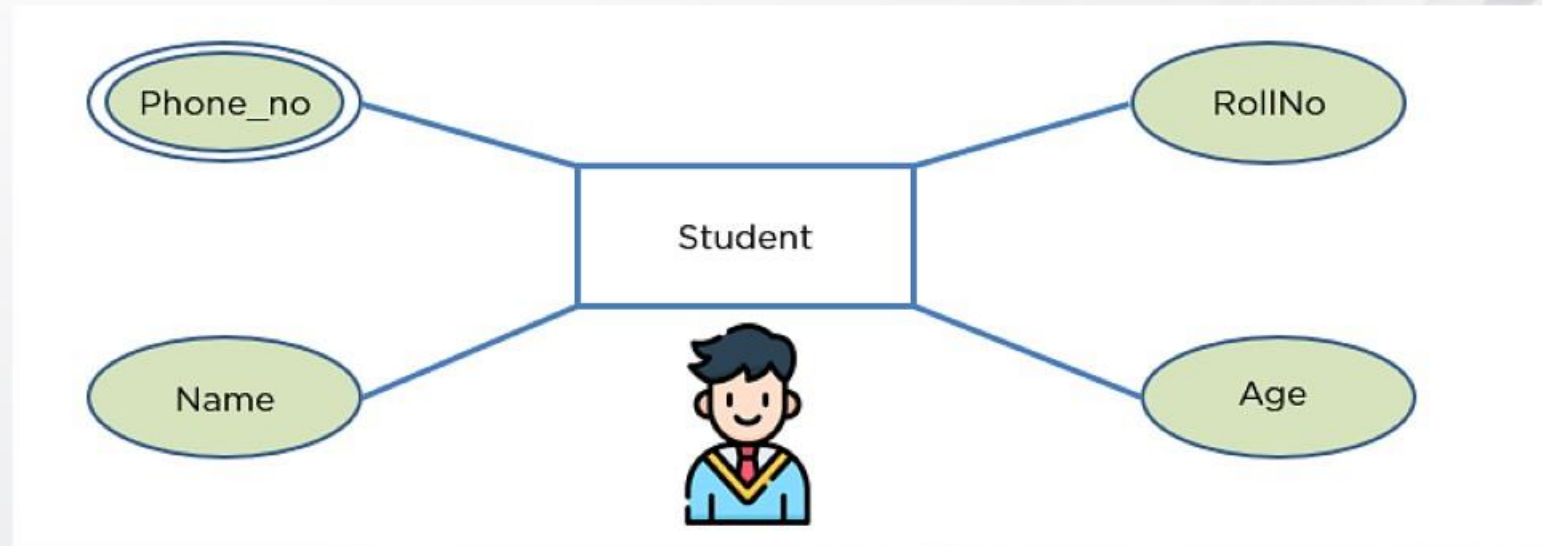
Composite Attribute

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.



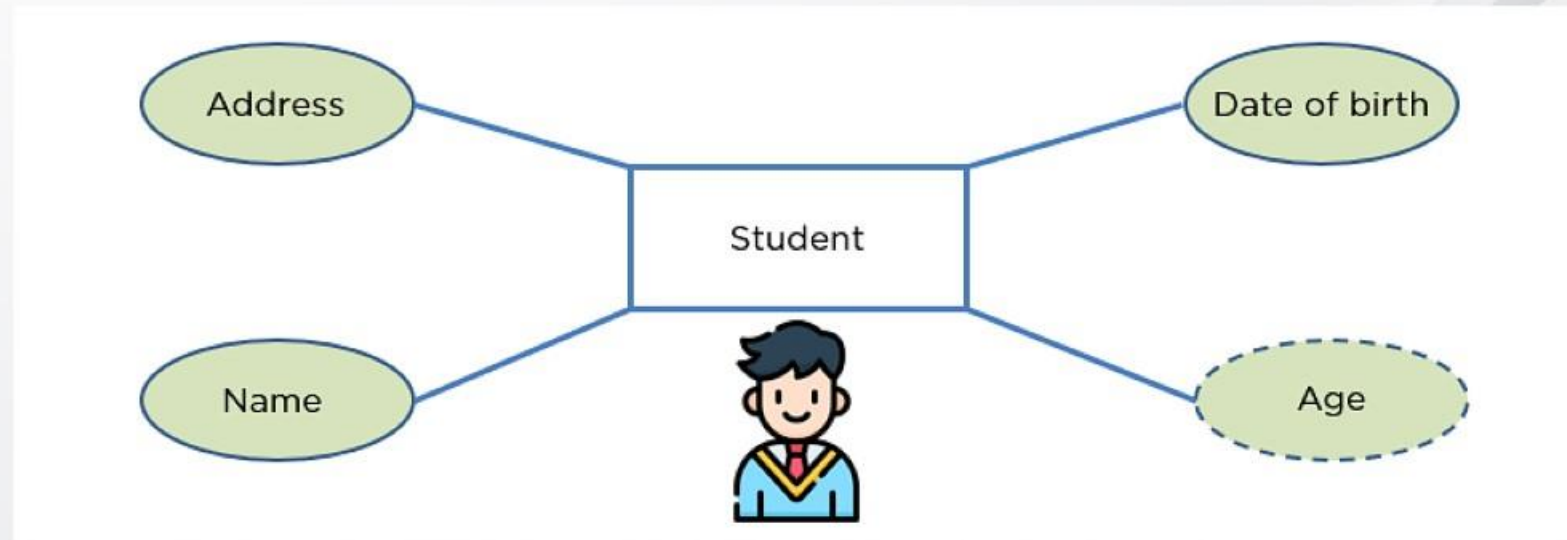
Multivalued Attribute

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

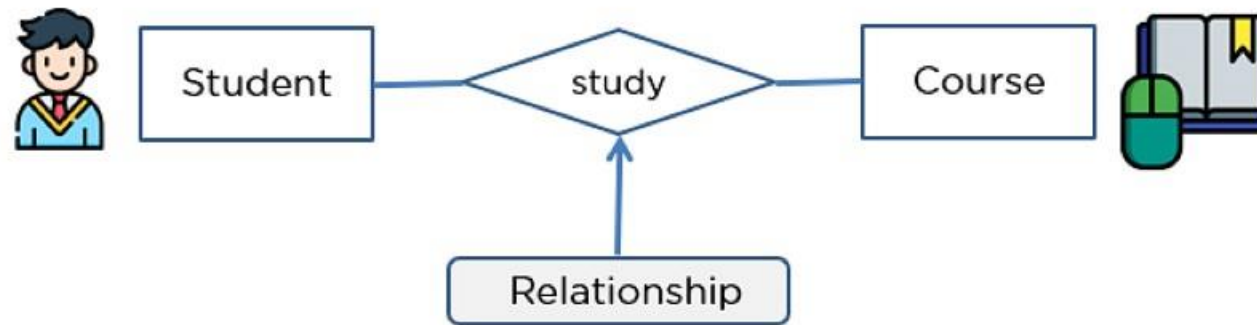
An attribute that can be derived from other attributes of the entity is known as a derived attribute. In the ER diagram, the dashed oval represents the derived attribute.



Relationship

The diamond shape showcases a relationship in the ER diagram.
It depicts the relationship between two entities.

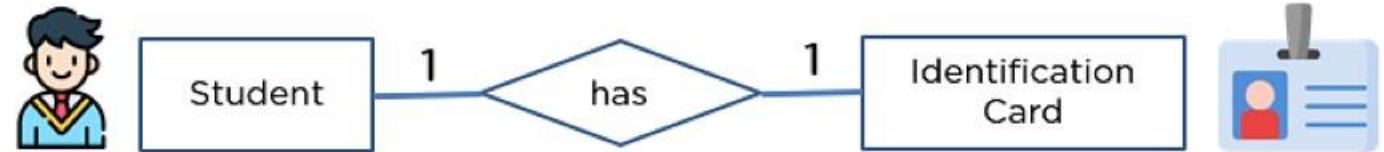
In the example below, both the student and the course are entities, and study is the relationship between them.



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 person.



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

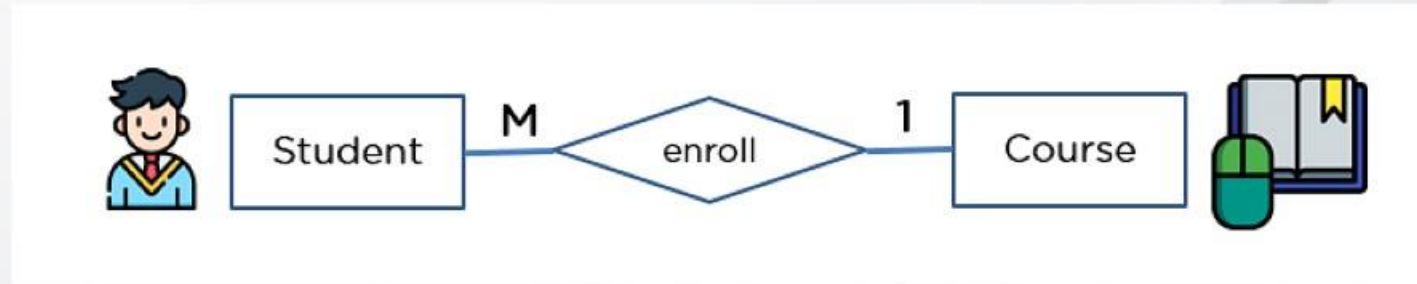
For example, a customer can place many orders, but an order cannot be placed by many customers.



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 Create an ER Diagram

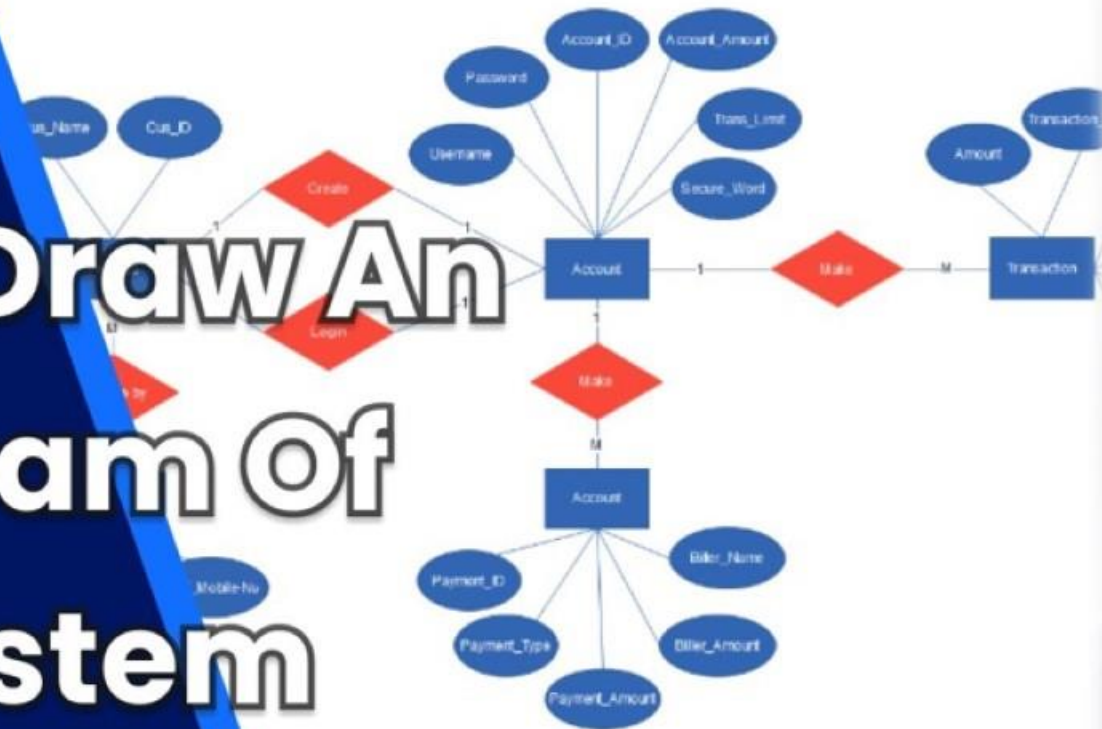
Before you design a relational database, it makes sense to create an ER diagram. You may have software that is equipped to handle this process. If not, Put pen (or pencil) to paper, or find a software program that can handle the charting requirements.

To create an ER diagram by hand:

1. Create a rectangular box for each **entity** or concept relevant to your model.
2. Draw lines to connect related entities to model the **relationships**. Label the relationships using verbs inside diamond shapes.
3. Identify the relevant **attributes** for each entity, beginning with the most important attributes, and enter them in ovals in the diagram. Later, you can make the attribute lists more detailed.

Creating an Entity-Relationship (ER) Diagram for a simple bank system involving customers and their bank accounts can be a great way to understand how ER diagrams work.

How To Draw An ER Diagram Of Bank System



ER Diagram for Bank Database:

In an ER diagram for a Banking System, we will have to keep in my mind the following points while making the diagram:

- There are going to be a number of entities, namely customers, banks, bank accounts, and transactions.
- The various entities in our diagram will have many different attributes like the Customer has **First Name** and **Last Name**.
- There are going to be varying relationships between the various entities like the Bank account, and the Customer will have a relationship called owned by, where we show that the account is being held by the Customer.

Let's start with making our ER diagram:

Deciding Entities

We first decide the entities that we are going to have in our diagram :

- Bank**
- Bank Account**
- Transaction**
- Customer**

- Bank
- Bank Account
- Transaction
- Customer

Bank Account

Customer

Bank

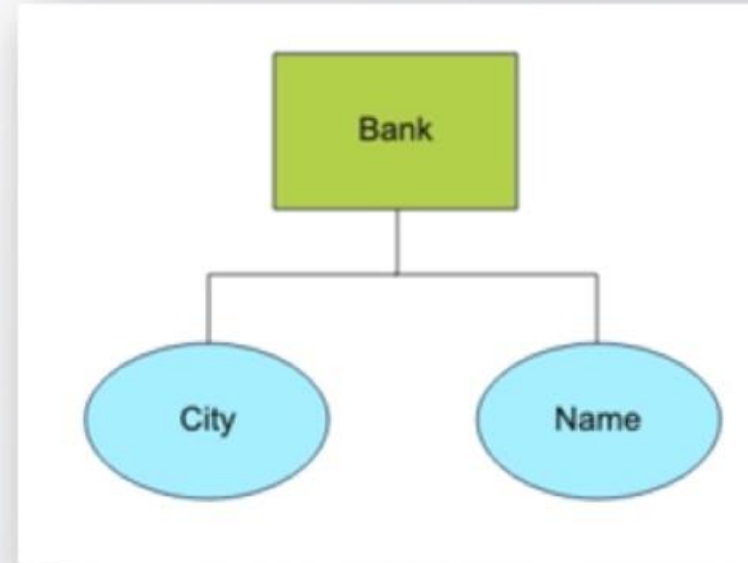
Transaction

Deciding Attributes :

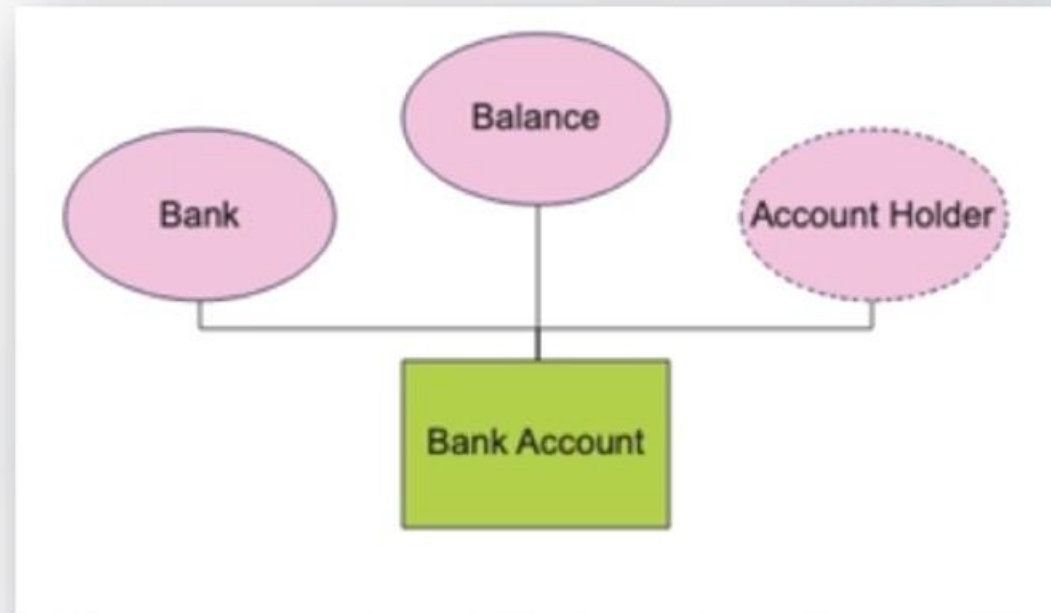
We then decide what attributes are associated with these entities:

- Bank: **Name, City**
- Bank Account: **Balance, Account holder, Bank**
- Customer: **First Name, Last Name, Phone Number**
- Transaction: **Deposit Account, Withdrawal Account, Timestamp, Currency, Amount**

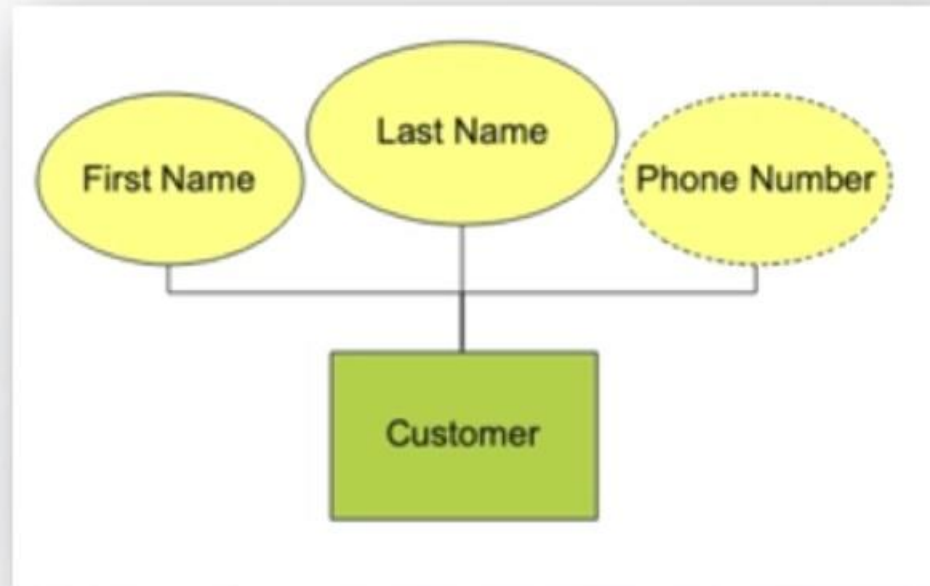
Bank: **Name, City**



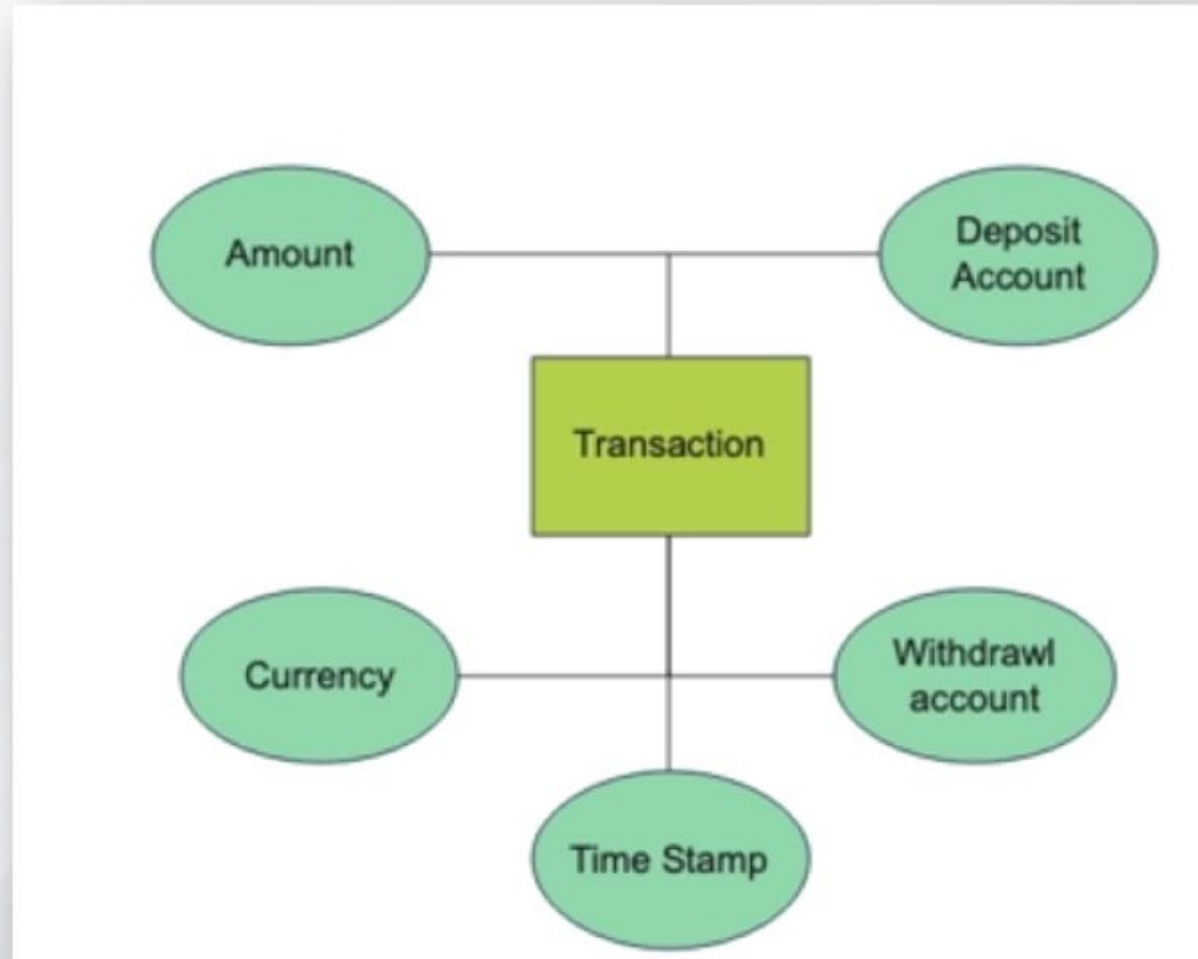
Bank Account: **Balance**, **Account holder**, **Bank**



Customer: **First Name, Last Name, Phone Number**



- Transaction: **Deposit Account, Withdrawal Account, Timestamp, Currency, Amount**

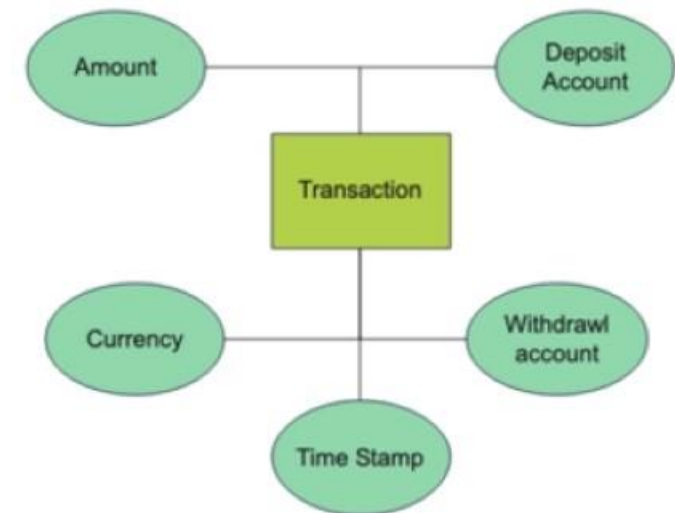
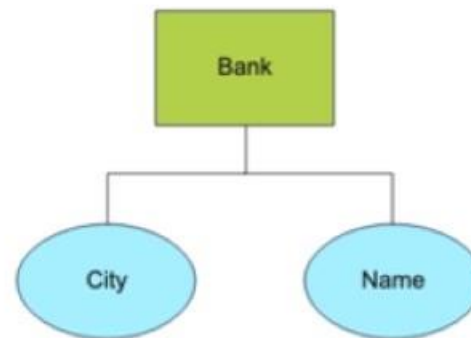
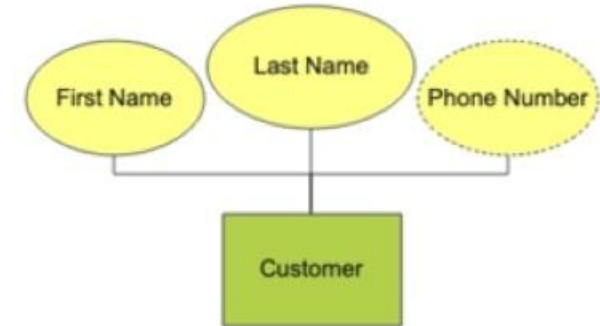
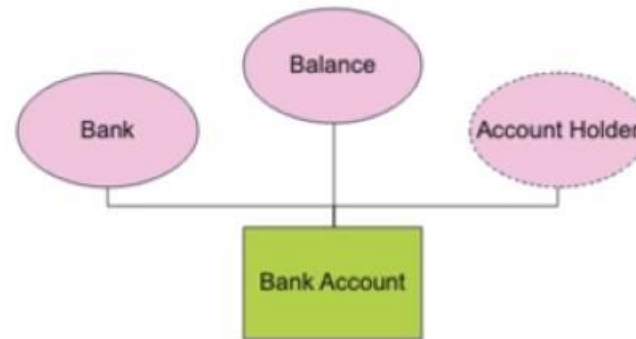


•Bank: **Name, City**

•Bank Account: **Balance, Account holder, Bank**

•Customer: **First Name, Last Name, Phone Number**

•Transaction: **Deposit Account, Withdrawal Account, Timestamp, Currency, Amount**

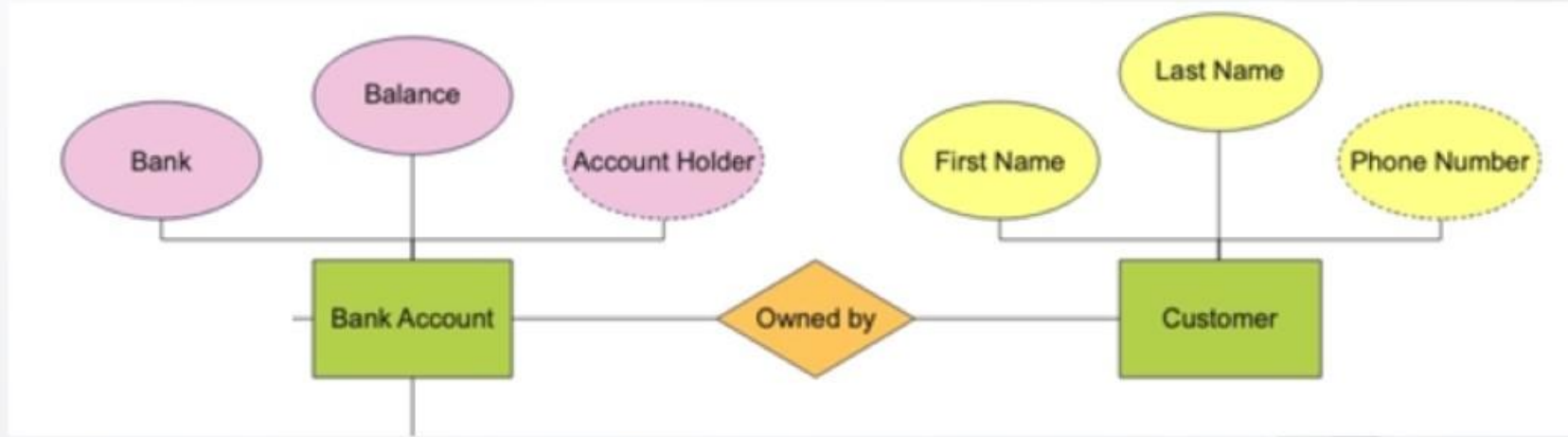


Deciding Relationships :

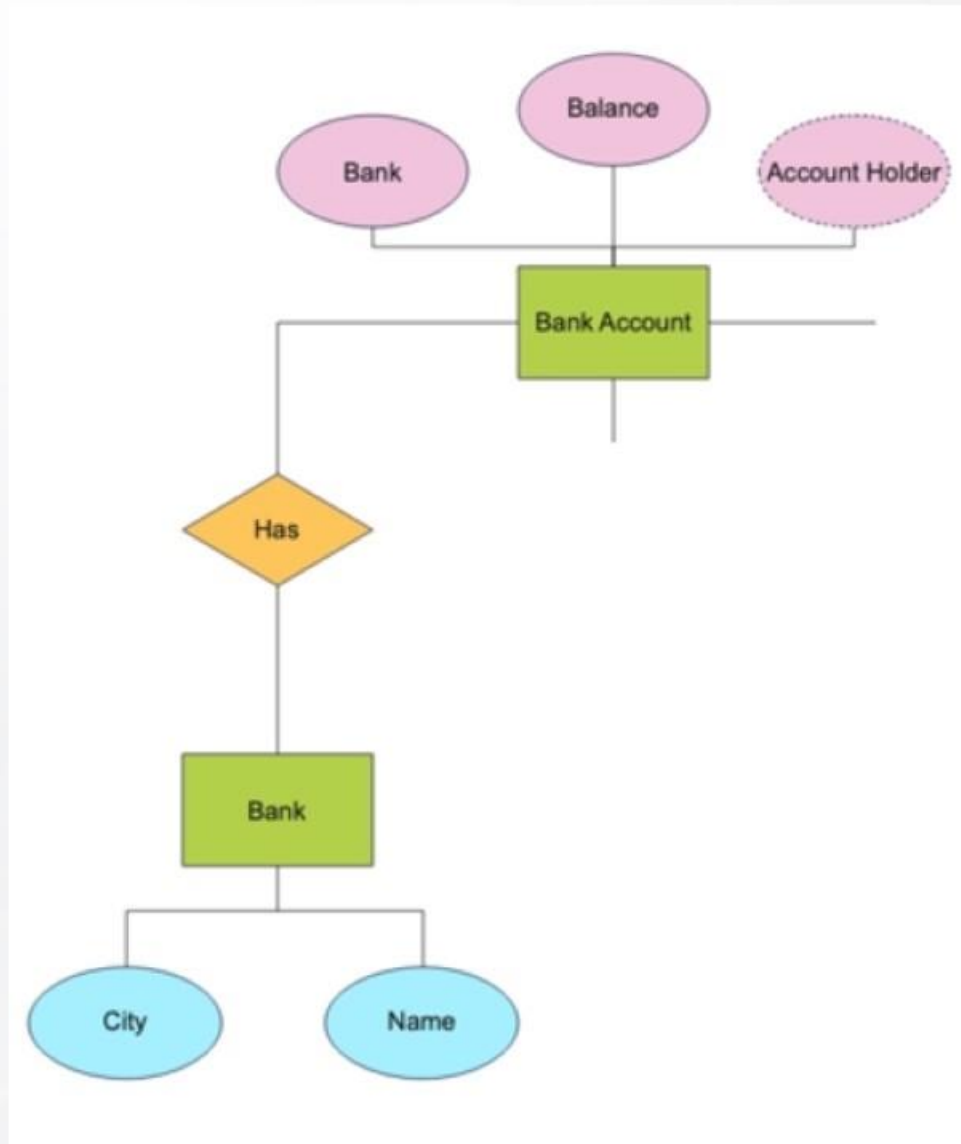
We then decide the relationships between the various entities:

- A Bank Account is owned by a Customer.
- A Bank has a Bank Account.
- A Transaction is linked to the Bank Account.
- The relationship between Bank Account and Customer is many to one because a Customer can have many Bank Accounts.
- The relationship between Bank and Bank Account is one too many as a Bank has many bank accounts.
- The Transaction to bank account relationship is many to one because multiple transitions can be linked to one bank account.

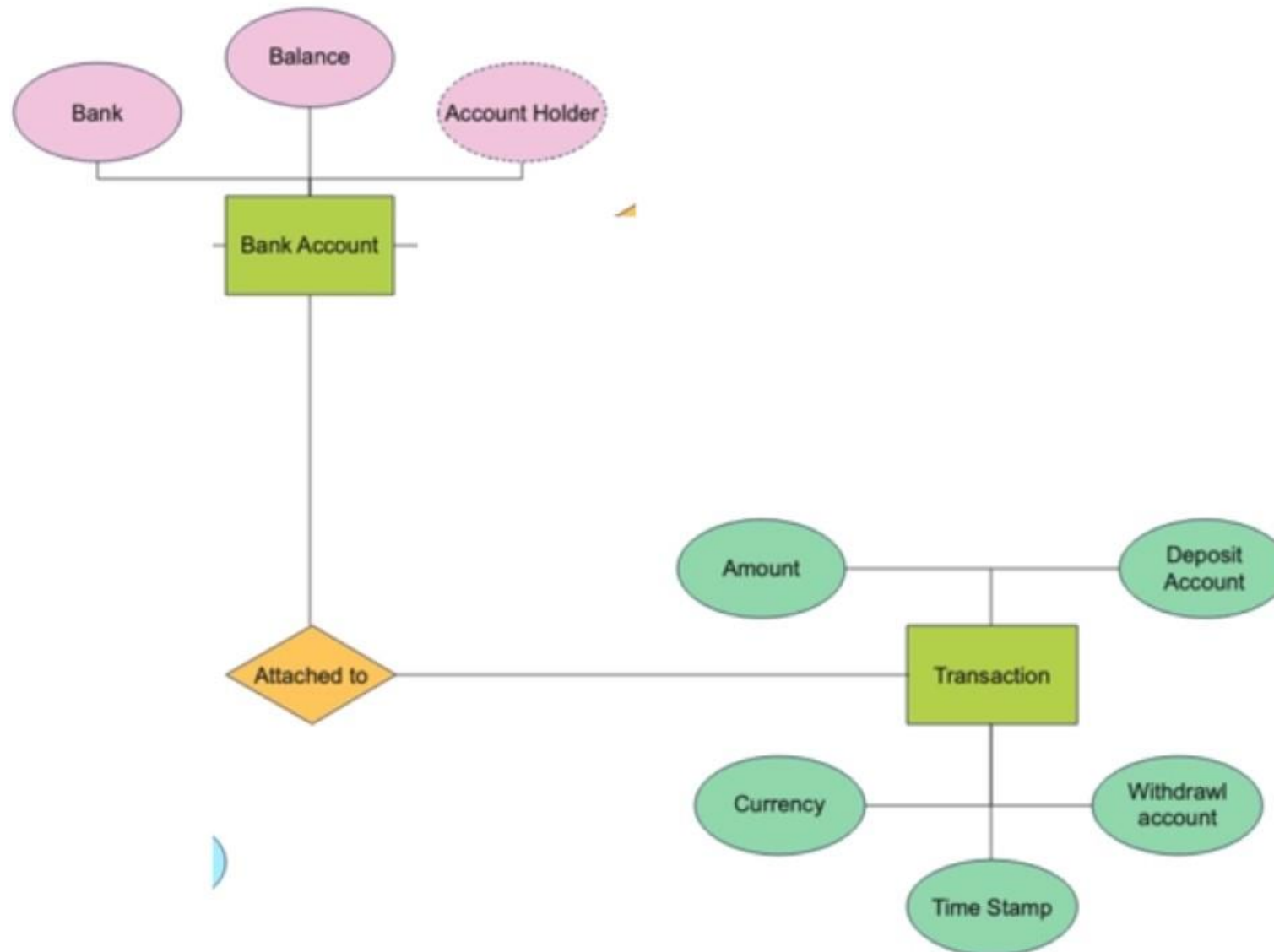
A Bank Account is owned by a Customer.



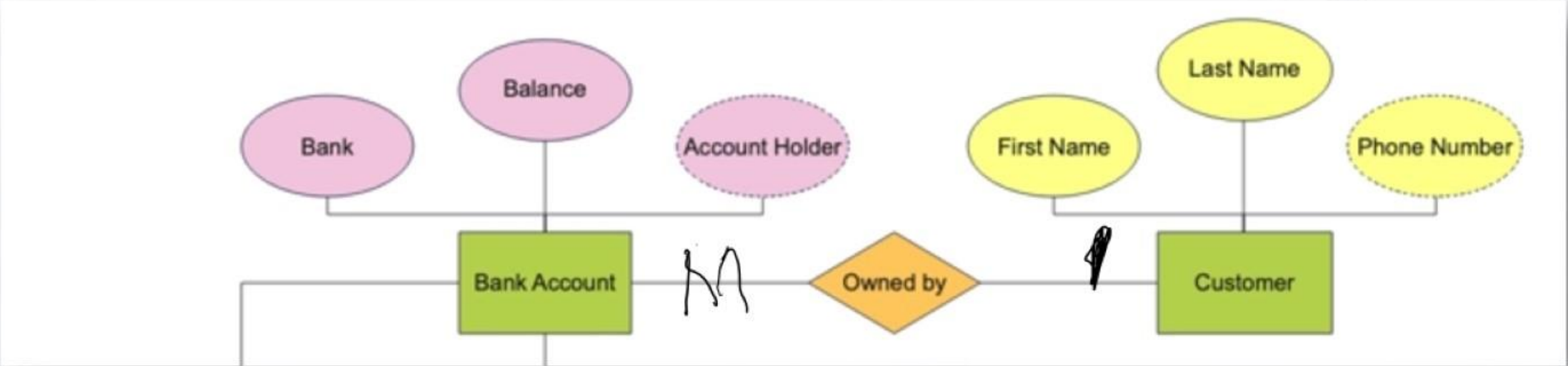
A Bank has a Bank Account.



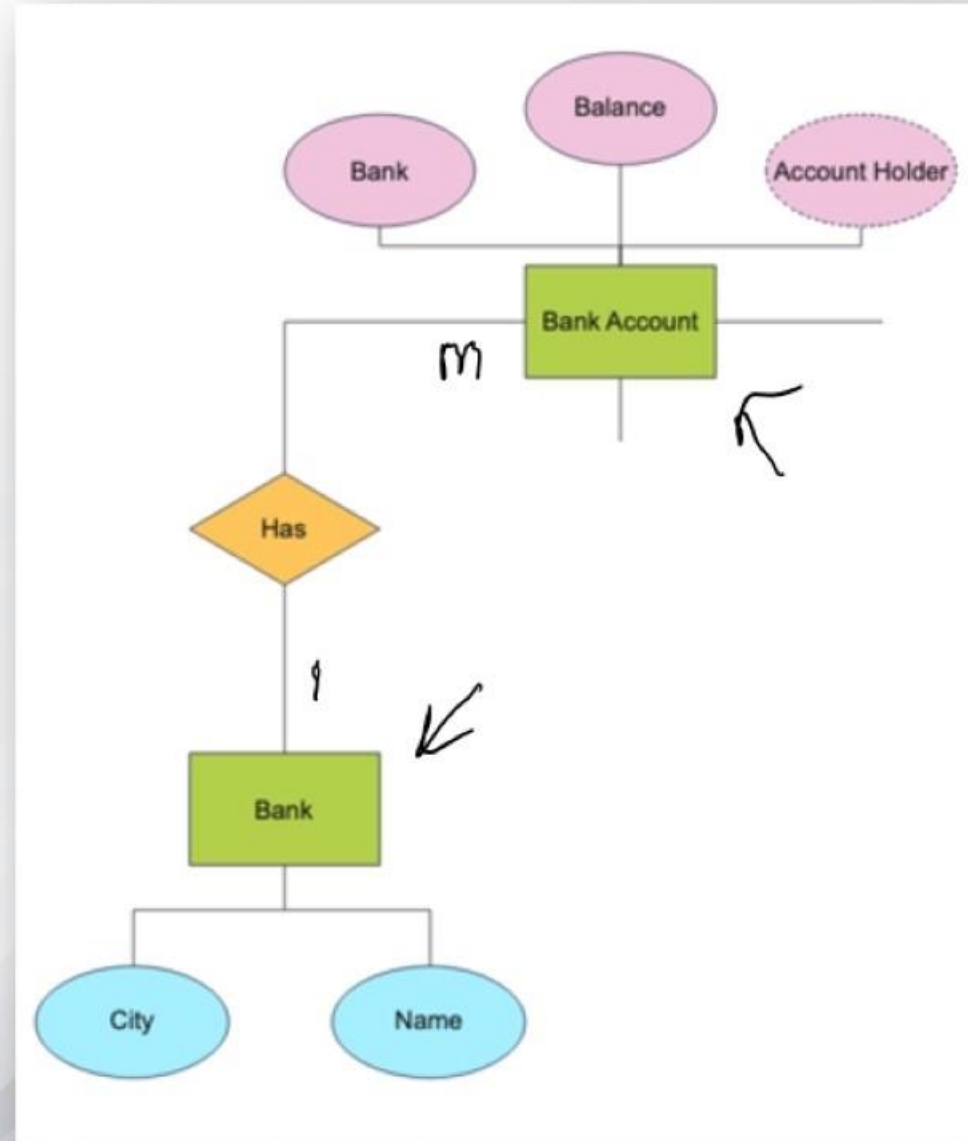
A Transaction is linked to the Bank Account



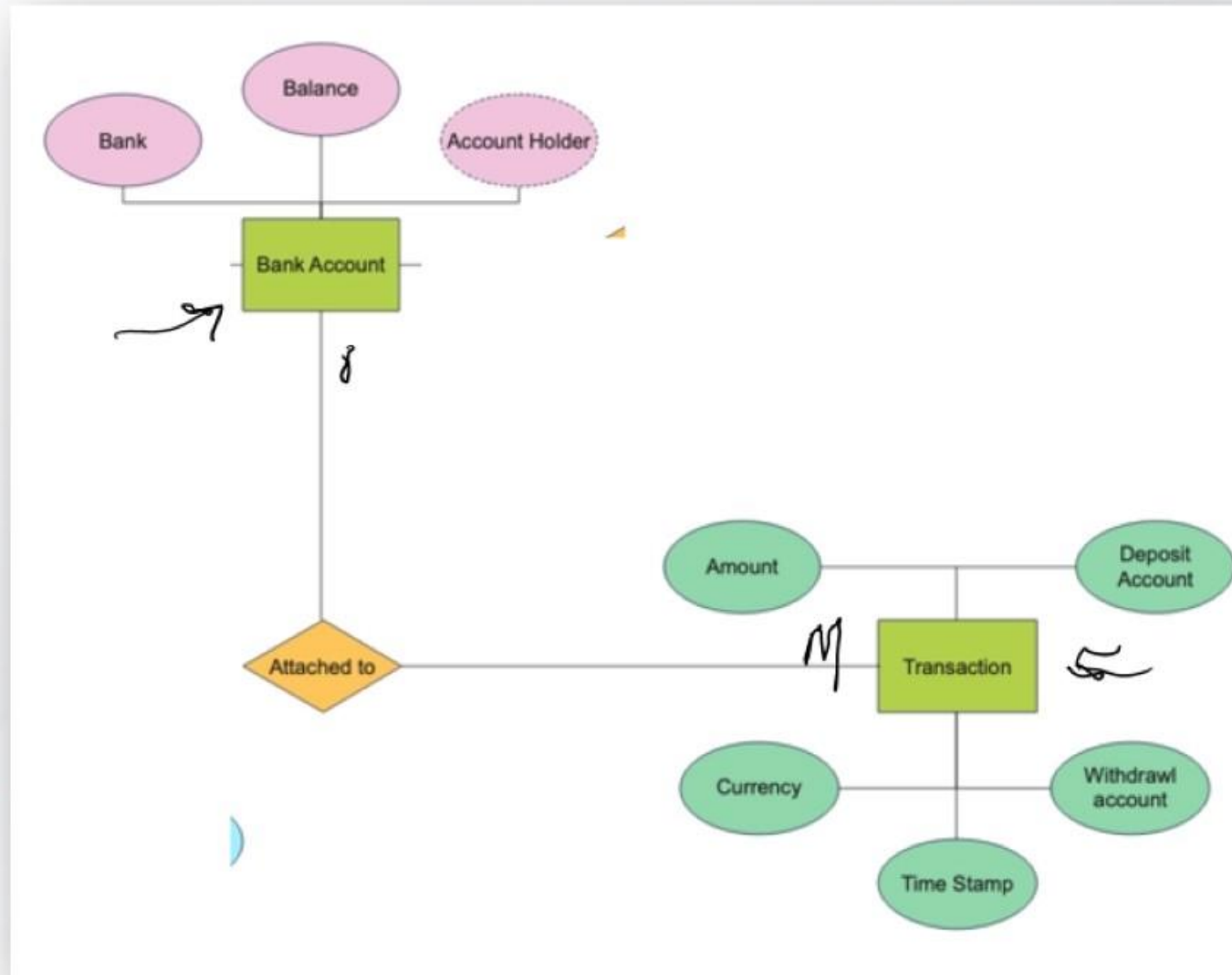
The relationship between Bank Account and Customer is many to one because a Customer can have many Bank Accounts.

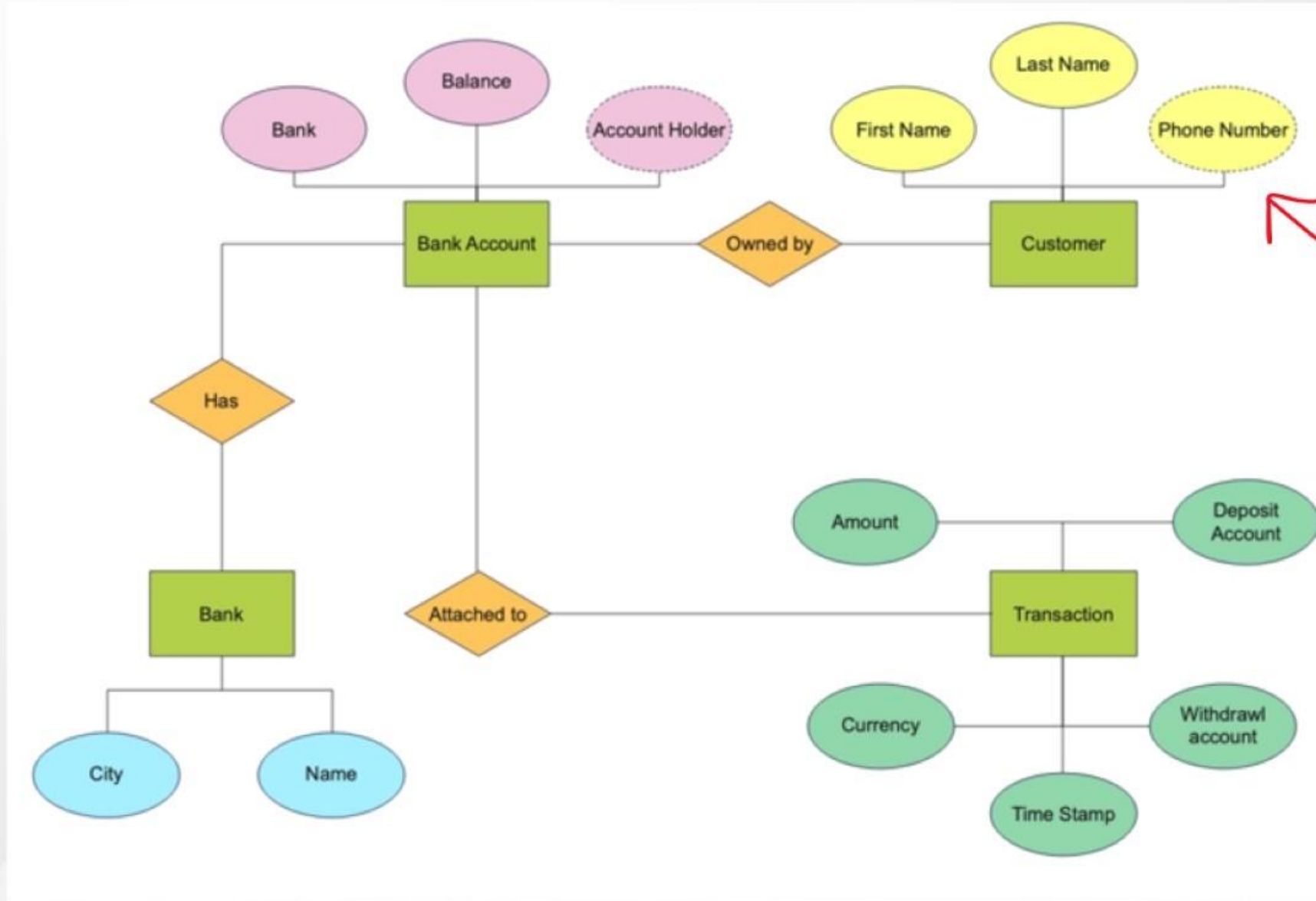


The relationship between Bank and Bank Account is one too many as a Bank has many bank accounts.



The Transaction to bank account relationship is many to one because multiple transactions can be linked to one bank account.





note

يمكن اعتبار رقم الهاتف
سمة مشتقة إذا كان يتم
اشتقاقه أو استنتاجه من سمة
أخرى مثل 'جهة اتصال
الرسائل القصيرة' (SMS
Contact). في هذه الحالة،
يتم استخراج رقم الهاتف
بناءً على بيانات أخرى، مما
يجعله سمة مشتقة.

Week 3 Outlines

2	Data Model	model.	Create relational tables in MySQL.
3	Relational Integrity Rules	Understand relational integrity constraints.	Implement primary and foreign key constraints in MySQL.
	Relational		

Relational integrity rules



Relational integrity rules are fundamental in relational database design, ensuring data remains accurate and consistent. They include primary keys to uniquely identify records, foreign keys to maintain referential integrity (ensuring relationships between tables are preserved), and unique constraints to prevent duplicate entries. These rules help in maintaining the reliability of data relationships across the database, crucial for beginners to understand as they form the backbone of database structure and integrity.

Relational integrity rules include:

- **Primary Keys:** Uniquely identify each record in a table.
- **Foreign Keys:** Establish and enforce relationships between tables.
- **Unique Constraints:** Prevent duplicate entries in specific columns.
- **Check Constraints:** Ensure that column values meet a specified condition.
- **Not Null Constraints:** Ensure that a column cannot have a null value.

These rules ensure data accuracy and consistency within relational databases.

Table named **Students**, where each student has a unique **student ID**, a name, and an age. The primary key for this table would be the student ID because it uniquely identifies each record. Here's a basic example:

•**Table: Students**

•**Columns:** StudentID (Primary Key), Name, Age

•**Rows:**

- Stu_Id: 1, Stu_Name: ali ahmed, Stu_Age:20
- Stu_Id : 2, Stu_Name: hassan ahmed, Stu_Age:2

<u>Stu_Id</u>	Stu_Name	Stu_Age
101	Steve	23
102	John	24
103	Robert	28
104	Steve	29
105	Carl	29

Primary Key

Unique values

Integrity Constraints: Entity Integrity

Definition: A primary key attribute can NOT be null (i.e. MUST have data)

SALESPERSON

salesperson_id	first_name	last_name	salary	territory_id
100	John	Stewart	48,000	1
140	Allen	Beeton	52,000	2
110	Chris	Lucero	43,000	1
190	Lorenzo	Davis	55,000	OK
150	Susan	Martin	50,000	3

SALESPERSON

salesperson_id	first_name	last_name	salary	territory_id
100	John	Stewart	48,000	1
X	Allen	Beeton	52,000	2
110	Chris	Lucero	43,000	1
190	Lorenzo	Davis	55,000	2
150	Susan	Martin	50,000	3

Note: Foreign keys can only be NULL with an optional one on the parent side of the relationship

Foreign keys establish a **relationship** between **two tables** in a database, ensuring referential integrity. For example, consider two tables: **Students** and **Courses**. Each student enrolls in a course, creating a relationship between students and courses.

•**Students Table:**

•Columns: **StudentID (Primary Key)**, StudentName

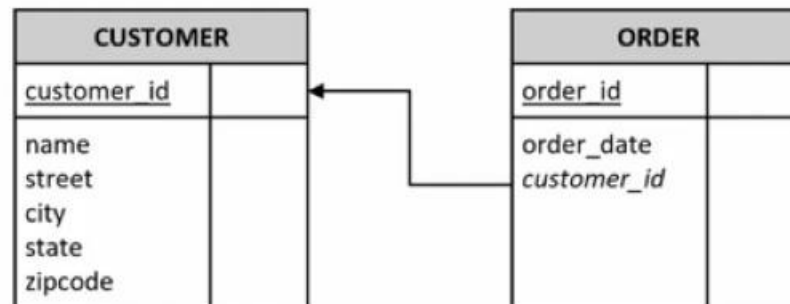
•**Courses Table:**

•Columns: CourseID (Primary Key), CourseName, **StudentID (Foreign Key)**

The **StudentID** in the **Courses** table is a foreign key that references the **StudentID** in the **Students** table. This setup ensures that each course entry is associated with a valid student, maintaining the integrity of the database.

Integrity Constraints: Referential Integrity

Definition: Any foreign key value **MUST** match a primary key value in the relation of the one side (if optional the foreign key can be null)



Unique constraints ensure that all values in a column are different from each other, preventing duplicate entries in a database table. For instance, in a table named **Students**, a unique constraint on the **email** column guarantees that no two students can have the same email address. Here's a simplified example of how this might be applied:

- **Table Name:** Students
- **Columns:** StudentID, Name, Email
- **Unique Constraint:** Applied to the **Email** column.

By setting a unique constraint on the **Email** column, the database ensures that every student has a distinct email address, thus maintaining the integrity and uniqueness of student contact information.

Chapter

03

This is illegal because both the Name and Email columns contain the same values for two rows.

Name	Email	Age
Amy Smith	amy.s@email.com	24
Jim Anderson	jim.a@email.com	21
Jim Anderson	jim.a@email.com	32

This is correct.

Name	Email	Age
Amy Smith	amy.s@email.com	24
Jim Anderson	jim.a@email.com	21
Jim Anderson	jim@email.com	32

Check constraints in a database ensure that all values in a column meet a specific condition. For example, if you have a table named "Employees" and you want to make sure that the age of each employee is greater than 18, you would add a check constraint on the "Age" column.

Example Table: Employees

- Columns: EmployeeID, Name, Age

```
CREATE TABLE tablename(  
    Column1 datatype,  
    Column2 datatype'  
    CHECK (column1 > 0)  
);
```

Not Null Constraints Example:

Imagine we have a table named **Students**, where each student must have a unique **student_id** and a name. To ensure every student record has a name, we apply a Not Null constraint on the **name** column.

•**Table Name:** Students

•**Columns:** student_id (INT), name (VARCHAR)

```
SELECT * FROM ConstraintDemo1
```

100 %

Results Messages

	ID	Name
1	1	Ali
2	2	NULL

A domain constraint

A domain constraint specifies the permissible values for a given attribute in a database table. For example, consider a table named "Students" with an attribute "Grade". If the "Grade" attribute only accepts values from A to F, then the domain constraint for "Grade" would be set to only these letters. This ensures data integrity by restricting the grade values entered into the database to only those that are valid within the specified domain.

Integrity Constraints: Domain Constraint

Definition: All allowable values for an attribute

Table	Attribute	Description	Domain
CUSTOMER	customer_id	Set of all possible customer IDs	integer: size 11
CUSTOMER	name	Set of all possible customer names	varchar: size 255
CUSTOMER	state	Set of all possible states	char: size 2
ORDER	order_date	Set of all possible order dates	datetime: format yyyy-mm-dd hh:mm:ss
ORDER_LINE	quantity	Set of all possible quantities	integer: size 8
PRODUCT	price	Set of all possible prices	double: size 10, 2 decimals
...	...	...	...



Do the following for the relations below:

1. Identify the attributes that would enforce entity integrity constraints.
2. Identify the referential integrity constraints that must exist.
3. Identify the domain constraints for each attribute of each relation.

DEPARTMENT	
<u>department_id</u>	
name	

EMPLOYEE	
<u>employee_id</u>	
first_name	
last_name	
salary	
department_id	

CERTIFICATE	
<u>employee_id</u>	
<u>course_id</u>	
completed	

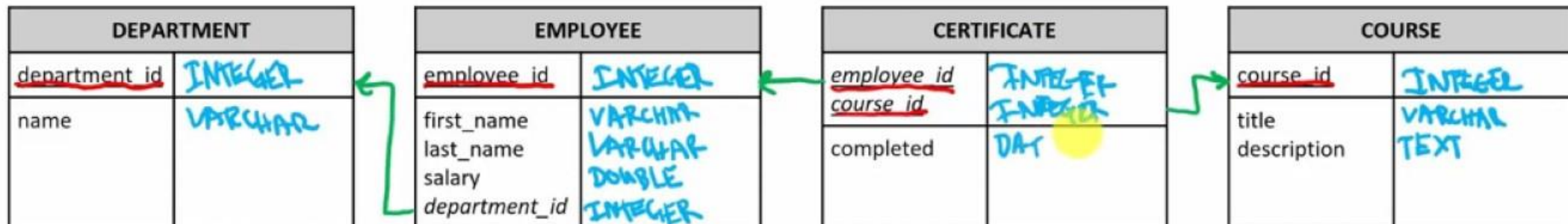
COURSE	
<u>course_id</u>	
title	
description	

Chapter

03

Do the following for the relations below:

1. Identify the attributes that would enforce entity integrity constraints.
2. Identify the referential integrity constraints that must exist.
3. Identify the domain constraints for each attribute of each relation.





To create a database in MySQL Workbench for students, courses, and teachers, follow these steps:

1.Open MySQL Workbench and connect to your MySQL server.

2.Create a New Schema: Click on the "+" icon next to "Schemas" to create a new database. Name it **school_db** and apply the changes.

3.Create Tables: Use the toolbar to create new tables for **Students**, **Courses**, and **Teachers**.

- For **Students**, add fields like **student_id**, **name**, **email**.
- For **Courses**, include **course_id**, **course_name**, **teacher_id**.
- For **Teachers**, fields might be **teacher_id**, **name**.

4.Define Primary Keys for each table (e.g., **student_id**, **course_id**, **teacher_id**) and set them to auto-increment.

5.Establish Relationships: Use foreign keys to link **Courses** to **Teachers** by **teacher_id**.

6.Insert Data: Use **INSERT INTO** statements to populate each table with sample data.

1.Create a Database (Schema):

```
CREATE DATABASE school_db; USE school_db;
```

2.Create a Table for Students:

```
CREATE TABLE Students ( student_id INT AUTO_INCREMENT, name VARCHAR(100), email VARCHAR(100), PRIMARY KEY(student_id) );
```

3. Create a Table for Courses:

```
CREATE TABLE Courses ( course_id INT AUTO_INCREMENT, course_name VARCHAR(100), teacher_id INT, PRIMARY KEY(course_id), FOREIGN KEY(teacher_id) REFERENCES Teachers(teacher_id) );
```

4.Create a Table for Teachers:

```
CREATE TABLE Teachers ( teacher_id INT AUTO_INCREMENT, name VARCHAR(100), PRIMARY KEY(teacher_id) );
```



Thank You