Database Structure and Design

Understanding Keys, Relationships, and Normalization in Relational Databases

Learning Objectives

Understand primary and foreign keys and their role in data integrity.

Explore Entity-Relationship Diagrams (ERD) to visualize database structures.

Learn how database normalization improves database efficiency.

What is Data Integrity?

Definition: Data integrity refers to the accuracy, consistency, and reliability of data throughout its lifecycle.

Importance: Maintains trust in the data, ensuring its reliability for users and applications.

Primary Keys

Definition: A primary key is a column or a set of columns that uniquely identifies each record in a table.

Example: A **StudentID** in the Students table or an **OrderID** in the Orders table.

Properties:

- Must be unique.
- Cannot be NULL.

Foreign Keys

Definition: A foreign key is a column that creates a relationship between two tables by referencing the primary key of another table.

Example: An **OrderID** in the OrderDetails table, which links to the Orders table.

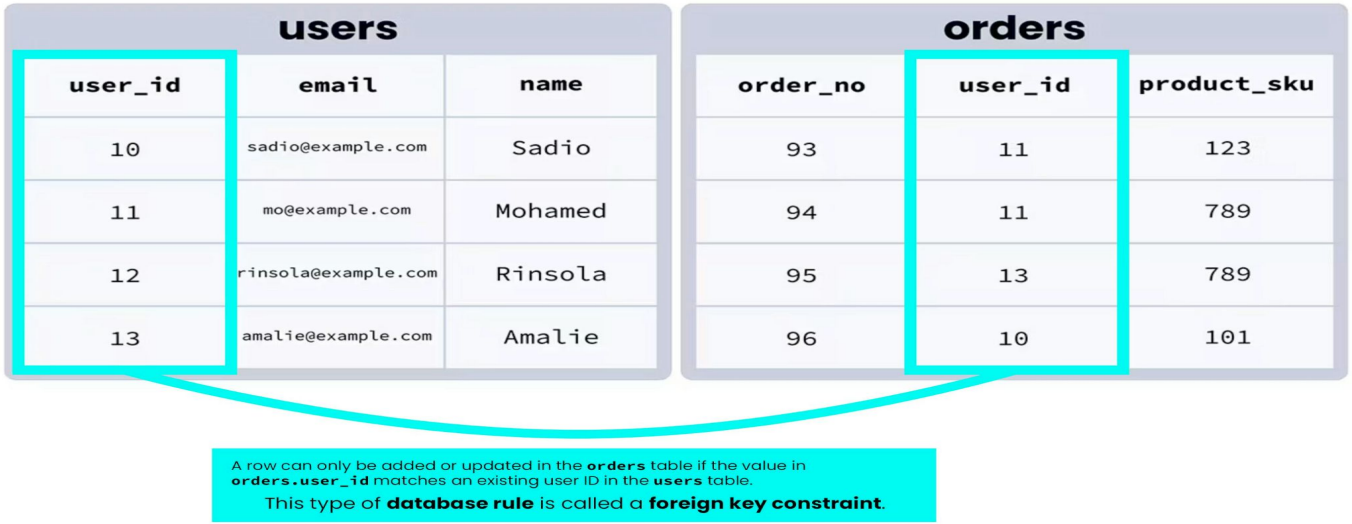
Purpose: Enforces referential integrity by ensuring that data in one table matches data in another.

Primary and Foreign Keys in Action

Primary Key: Ensures each record in the table is unique.

Foreign Key: Ensures that relationships between tables are valid.

Example: The **CustomerID** in the Orders table links to the **CustomerID** in the Customers table.



Introduction to ERD

Definition: An ERD is a visual representation of the entities (tables) in a database and how they relate to each other.

Purpose: Helps designers visualize the structure of a database before creating it.

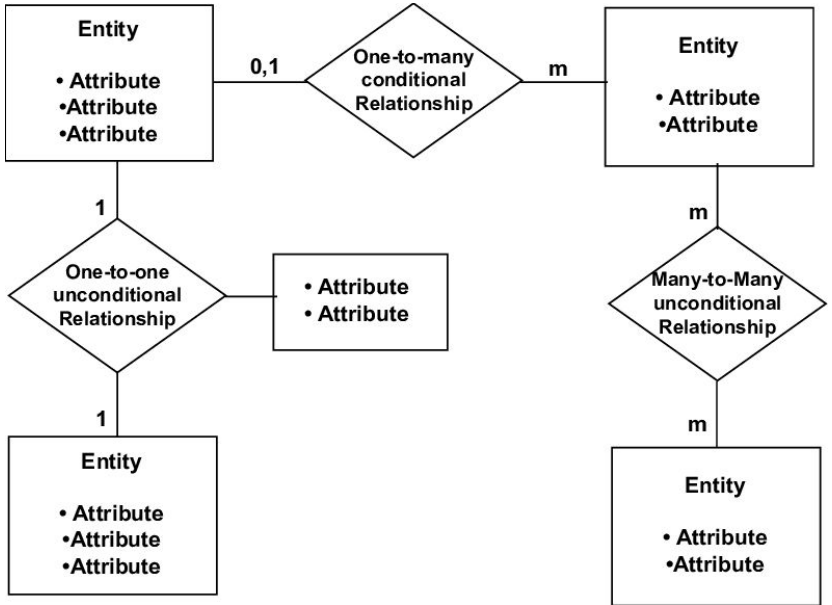
Components of ERD

Entities: Represent objects (e.g., Customers, Orders, Products).

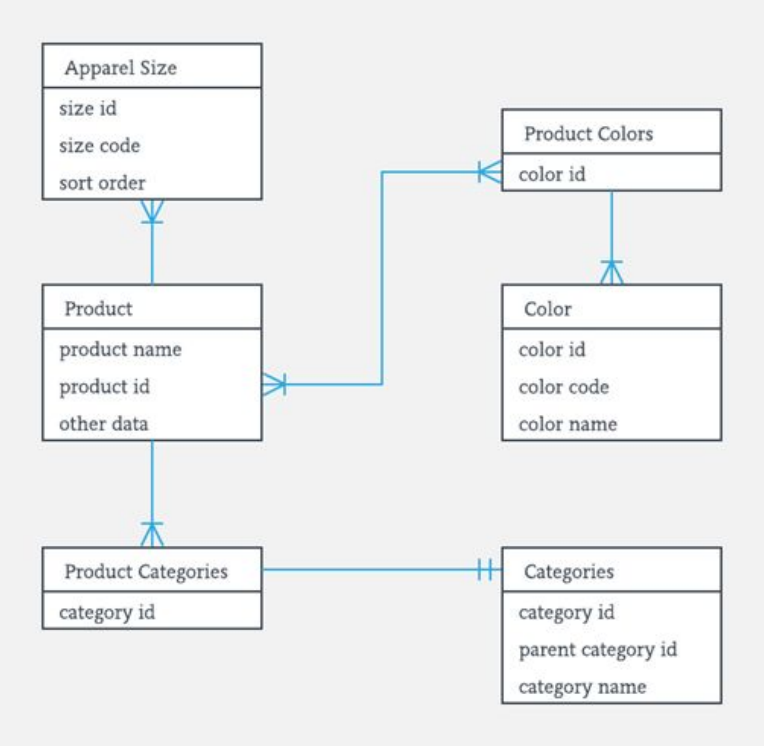
Attributes: Define properties of entities (e.g., CustomerName, OrderDate).

Relationships: Show how entities are related (e.g., a customer places an order).

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



ERD Example


Apparel Size
size id
size code
sort order
Product Colors
color id
Product
product name
product id
other data
Color
color id
color code
color name
Product Categories
category id
Categories
category id
parent category id
category name

Database Normalization

Definition: Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity.

Purpose: Prevents anomalies in data insertion, deletion, and updates by breaking down large tables into smaller, related ones.

First Normal Form (1NF)

Definition: Ensures that each column contains atomic (indivisible) values and each record is unique.

Example: No repeating groups or arrays in a table.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

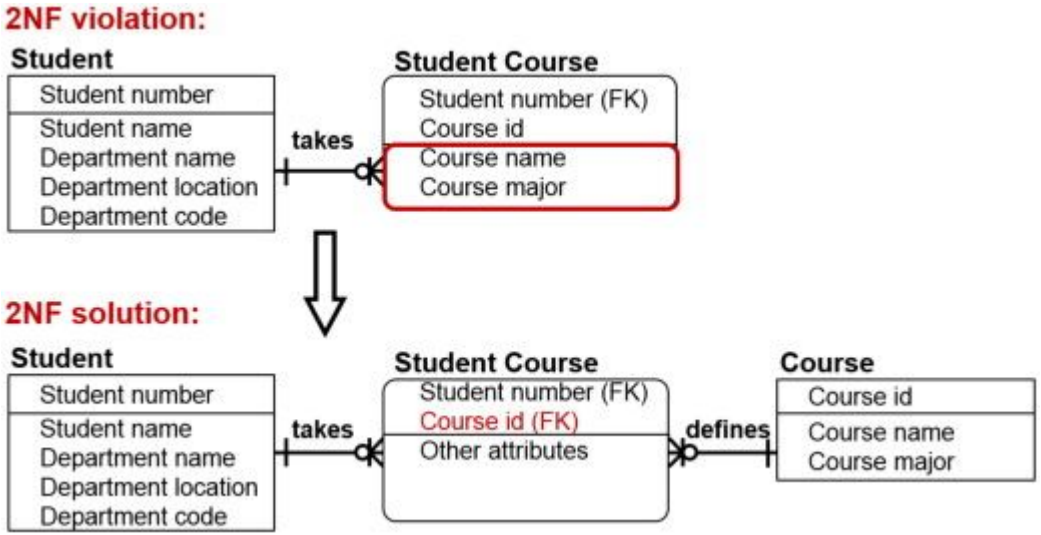
STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

Second Normal Form (2NF)

Definition: Ensures that all non-key attributes are fully dependent on the primary key.

Example: If a table has a composite primary key, each non-key attribute must be related to the entire key, not just part of it.



Third Normal Form (3NF)

Definition: Ensures that all attributes are only dependent on the primary key, and not on other non-key attributes.

Example: Removes transitive dependencies.

Transforming to 3NF

- Move the attributes involved in transitive dependency to another relation

Order_ID	Order_Date	Cust_ID	Prod_Name	Cust_Address
1006	10/24/2004	2	Value Furniture	Plano, TX
1007	10/25/2004	6	Furniture Gallery	Boulder, CO
1008	11/1/2004	2	Value Furniture	Plano, TX

Order

Order_ID	Order_Date	Cust_ID
1006	10/24/2004	2
1007	10/25/2004	6
1008	11/1/2004	2

Customer

Cust_ID	Prod_Name	Cust_Address
2	Value Furniture	Plano, TX
6	Furniture Gallery	Boulder, CO

Example Scenario:

We have a company that tracks employee data. Initially, we might store the following information in a single table:

EmployeeID	EmployeeName	Department	Manager
1	Alice	Sales	John Smith
2	Bob	Sales	John Smith
3	Charlie	HR	Jane Doe
4	Dave	HR	Jane Doe

EmployeeID is the primary key (a unique identifier for each employee).

EmployeeName is the name of the employee.

Department is the department the employee works in.

Manager is the manager of the department.

Problem with 3NF:

Notice that the **Manager** is dependent on the **Department**, not directly on the **EmployeeID**. Every time you have employees in the **Sales** department, the **Manager** is always "John Smith". This violates the rule of 3NF, where all non-key attributes (like Manager) should depend only on the **primary key** (EmployeeID), not on other non-key attributes (Department).

Solution to Make It 3NF:

To fix this and meet the requirements of 3NF, we need to break the table into two tables:

1. **Employee Table**: Information only related to employees.
2. **Department Table**: Information about departments and their managers.

New Tables:

Employee Table:

EmployeeID	EmployeeName	DepartmentID
1	Alice	1
2	Bob	1
3	Charlie	2
4	Dave	2

- Now, we have removed the **Manager** column, and we are using a **DepartmentID** to reference

New Tables:

Employee Table:

EmployeeID	EmployeeName	DepartmentID
1	Alice	1
2	Bob	1
3	Charlie	2
4	Dave	2

- Now, we have removed the **Manager** column, and we are using a **DepartmentID** to reference departments.

Department Table:

DepartmentID	DepartmentName	Manager
1	Sales	John Smith
2	HR	Jane Doe

- The **Department** and **Manager** information is now stored separately in this table. It's no longer repeated for each employee.

Now We Have:

- **Employee Table**: Each employee is linked to their department by the **DepartmentID**, and the manager information is handled separately.
- **Department Table**: Each department is linked to its manager.

This way:

- **EmployeeName** depends on **EmployeeID**.
- **DepartmentID** depends on **EmployeeID**.
- **Manager** depends on **DepartmentID**, not on **EmployeeID** anymore, making the table adhere to **3NF**.

Benefits:

- No redundancy: The manager's name is stored once for each department.
- Easier updates: If a department changes its manager, you only need to update it in one place, not for every employee in that department.

Benefits of Normalization

Reduces Redundancy: Eliminates duplicate data.

Improves Data Integrity: Ensures that the data is logically stored.

Better Data Management: Prevents insertion, update, and deletion anomalies.

Conclusion

Summary: Primary and foreign keys ensure data integrity. ERDs help visualize database relationships. Normalization reduces redundancy and enhances database efficiency.

Next Steps: Explore more advanced database design concepts such as indexing and query optimization.