

## **What is a database management system?**

A database management system (DBMS) is software that allows users to create, organize, store, retrieve, manipulate, and manage data efficiently. It provides an interface for interacting with databases and enables users to perform various operations on the data.

A DBMS serves as an intermediary between the users and the database. It ensures data integrity, security, and consistency while handling multiple concurrent user requests. Some common features of a DBMS include:

**Data definition:** The DBMS allows users to define the structure of the database, specifying data types, relationships between entities, and constraints.

**Data manipulation:** It enables users to insert, update, delete, and retrieve data from the database using structured query languages (SQL) or other programming interfaces.

**Data storage and organization:** The DBMS manages the physical storage of data on disks or other storage media, determining how data is stored, indexed, and accessed for optimal performance.

**Data security:** DBMS provides mechanisms to control access to the database, enforcing user authentication, authorization, and data encryption to protect sensitive information.

**Data concurrency and recovery:** It handles concurrent access to the database by multiple users, ensuring that transactions are processed reliably and in a controlled manner. It also provides mechanisms for data backup and recovery in case of system failures or errors.

**Data integrity and consistency:** The DBMS enforces data integrity constraints, such as primary key and foreign key constraints, to maintain the consistency and accuracy of the stored data.

**Data scalability and performance optimization:** A DBMS allows for scaling the database as data volume increases, and it includes features to optimize query execution and enhance system performance.

Examples of popular DBMSs include Oracle Database, Microsoft SQL Server, MySQL, PostgreSQL, MongoDB, and SQLite. Each DBMS may have its own strengths, limitations, and specialized features, catering to different application requirements.

### **Basics of MySQL:**

**1. Creating a database-** To create a database use the following command:

```
CREATE DATABASE database_name;
```

## 2. Constructing tables under the database:

Use the CREATE TABLE statement to define the structure of your table. Specify the table name and define the columns along with their data types and any additional constraints.

```
CREATE TABLE employees (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    email VARCHAR(100),  
    hire_date DATE  
);
```

In the above example, we created a table with columns for id, first\_name, last\_name, email, and hire\_date. The id column is defined as the primary key with the AUTO\_INCREMENT attribute, which automatically generates a unique value for each new row.

**3. Specify constraints (optional):** You can add constraints to enforce data integrity. For example, to specify a column as the primary key, use the PRIMARY KEY constraint. You can also define NOT NULL constraints to ensure that certain columns always have a value.

## 4. Inserting data in the tables:

There are two ways to insert data into a table. Let's say we are familiar with the serial of the attributes that we have added in the table. In that case we can use:

```
INSERT INTO employees VALUES ('val1','val2'. ....)
```

However, in reality, we do not always remember the exact order at which we put the attributes. So using this command is easier:

```
INSERT INTO employees(id, first_name,last_name,email,hire_date) VALUES ('001','Stephen',  
'Davis', 'sth@gmail.com', '20/1/12')
```

*Note: You can also choose the attributes which you want to populate, so just select those attributes that you want to insert into the table!*

### **5. Deleting an attribute from the table which meets a certain condition:**

```
DELETE FROM employees
```

```
WHERE id= 001
```

**Deleting an attribute(email for this example) from the table:**

```
ALTER TABLE employees
```

```
DROP COLUMN email;
```

### **6. Query Design:**

**The basic query design is:**

```
SELECT attribute_name  
FROM table_name  
WHERE condition (if there is some condition)
```

The query above is manipulated as per need.

1.Let's say we want to see all the employees' IDs.

```
SELECT id  
FROM employees
```

2. If we want to view all rows we have to use the \* command which is:

```
SELECT *  
FROM employees
```

*This query fetches the whole table employees.*

3. If we want to view the last\_name of the employee with ID=001, this command is used:

```
SELECT last_name  
FROM employees
```

WHERE id= 001

Now, we also have to remember what the data type of attribute is, if it was inserted as text, then we have to write the above command like this:

```
SELECT last_name
FROM employees
WHERE id= '001'
```

#### 4. Multiple Conditions:

In case of multiple conditions, we have to understand the query first (i.e. does it require that both conditions are met or only one of them?)

- a. Query: Find the employee ID of those employees whose last name is Rahman and were hired on 5/5/16. *(In this query, we have to fulfill both conditions so we should use AND)*

```
SELECT id
FROM employees
WHERE last_name= 'Rahman' AND hire_date= '05/05/16'
```

- b. Query: Find the employee ID of those employees whose last name is Rahman or Ahmed *(In this query, we have to fulfill only one condition so we should use OR)*

```
SELECT id
FROM employees
WHERE last_name= 'Rahman' OR last_name= 'Ahmed'
```

#### 5. Some common functions:

To retrieve data sometimes you need to use some functions as per the query.  
For example, we want to count the number of employees, the command will be:

```
SELECT COUNT(id)
FROM employees
```

There are other functions we can use as well such as AVG, SUM, MAX, MIN, etc.

#### 6. Operator Meaning

- = Equal to

- > Greater than
- >= Greater than or equal to
- < Less than
- <= Less than or equal to
- <> Not equal to
- != Not equal to

Example: Find the employee id of those whose last name is not Rahman

```
SELECT id
FROM employees
WHERE last_name != 'Rahman'
```

## 7. Ordering:

To order the data in ascending order or descending order, we use ASC/DESC functions.

Example: Find the employee id and sort them in descending order.

```
SELECT id
FROM employees
ORDER BY id DESC
```

*Note: By default, mySQL sorts data in ascending order so you do not necessarily need to add the ASC function.*

## 7. Grouping:

If we want to group the data we want to retrieve, we have to use “GROUP BY” in the SQL command.

Example: Count the number of employees and group them as per their hire date.

```
SELECT COUNT(id)
FROM employees
GROUP BY hire_date
```

## 8. Having:

The HAVING clause is used in SQL to filter the results of a query based on conditions involving aggregated data. It complements the WHERE clause, which filters individual rows, by allowing you to filter groups of rows based on the results of aggregate functions and grouped data.

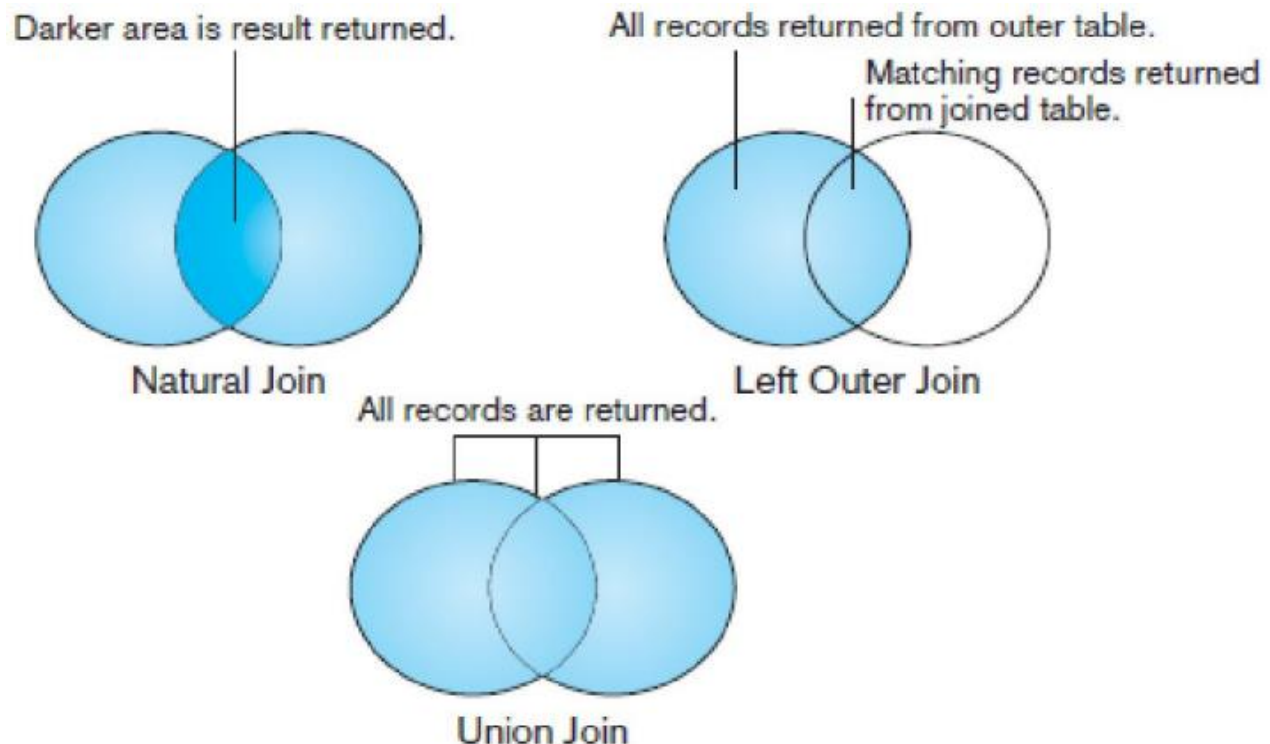
Example: Find only states with more than one customer.

```
SELECT CustomerState, COUNT (CustomerState)
```

```
FROM Customer_T
GROUP BY CustomerState
HAVING COUNT (CustomerState) > 1;
```

## 9. Join:

- Join A relational operation that causes two tables with a common domain to be combined into a single table or view.
- Equi-join A join in which the joining condition is based on equality between values in the common columns. Common columns appear (redundantly) in the result table.
- Natural join A join that is the same as an equi-join except that one of the duplicate columns is eliminated in the result table.
- Outer join A join in which rows that do not have matching values in common columns are nevertheless included in the result table.



Example: What are the customer IDs and names of all customers, along with the order IDs for all the orders they have placed?

```
SELECT C.CustomerID, O.CustomerID, CustomerName,
OrderID
FROM Customer_T AS C INNER JOIN Order_T AS O
ON
Customer_T.CustomerID = Order_T.CustomerID
ORDER BY OrderID;
```

Example: For each customer who has placed an order, what is the customer's ID, name, and order number?

```
SELECT C.CustomerID, CustomerName, OrderID
FROM Customer_T AS C NATURAL JOIN Order_T AS O
ON
C.CustomerID = O.CustomerID;
```

Example: List customer name, identification number, and order number for all customers listed in the Customer table. Include the customer identification number and name even if there is no order available for that customer.

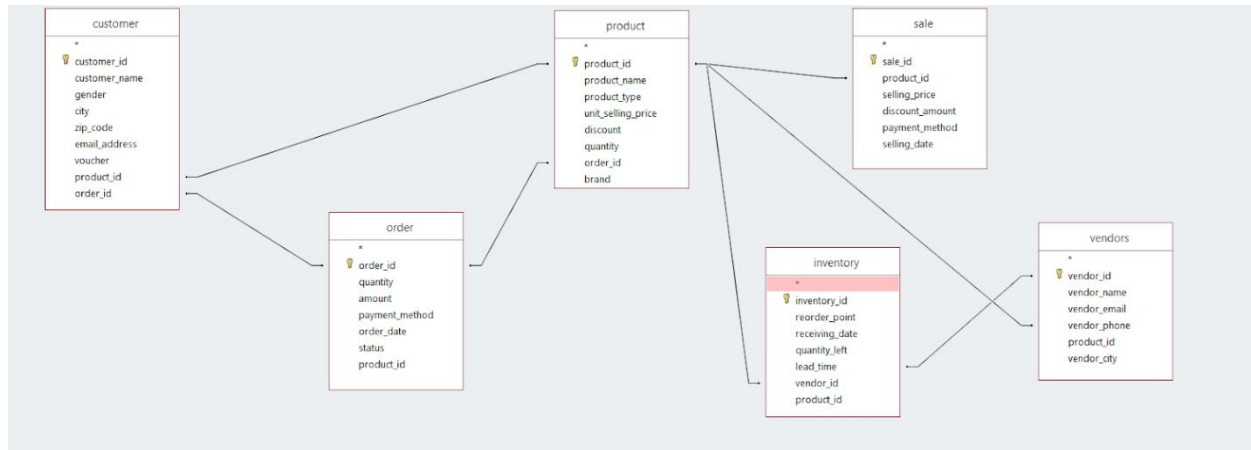
```
SELECT C.CustomerID, CustomerName, OrderID
FROM Customer_T AS C LEFT OUTER JOIN Order_T AS O
WHERE C.CustomerID = O. CustomerID;
```

### **Entity Relationship Diagram (ERD):**

ERDs are valuable tools for understanding, designing, and communicating the structure of a database system. They aid in requirements analysis, data modeling, database design, and optimization, ensuring data integrity and facilitating collaboration among stakeholders.

1. **Identify entities:** Start by identifying the main entities in your system or database. Entities are objects, concepts, or things that you want to store information about. For example, in a university database, entities could include students, courses, and instructors.
2. **Determine relationships:** Determine the relationships between the entities you identified. Relationships represent how the entities are related to each other. For example, a student can enroll in multiple courses, and a course can have multiple students. This is a many-to-many relationship, which would require a linking table.
3. **Define attributes:** Define the attributes for each entity. Attributes are the properties or characteristics of an entity. For example, a student entity might have attributes like student ID, name, and date of birth. Add these attributes to their respective entities.
4. **Identify primary keys:** Determine the primary key for each entity. A primary key is a unique identifier for each record in a table. It can be a single attribute or a combination of attributes. For example, in a student entity, the student ID could be the primary key.
5. **Add relationships:** Add the relationships between entities. You can use different notation symbols to represent relationships, such as crow's foot notation. For example, you can draw a crow's foot line between the student and course entities to indicate a many-to-many relationship.
6. **Specify cardinality and participation:** Determine the cardinality and participation constraints for each relationship. Cardinality defines the number of occurrences of one entity that can be associated with the number of occurrences of another entity. Participation indicates whether an entity is mandatory (total participation) or optional (partial participation) in a relationship. Represent these constraints using appropriate symbols or labels.
7. **Refine the ERD:** Review and refine your ERD to ensure it accurately represents the relationships, attributes, and constraints of your system. Make sure it follows good design principles and accurately captures the required information.
8. **Normalize the ERD:** If you're designing a relational database, you may want to apply normalization techniques to ensure data integrity and eliminate redundancy. This involves breaking down larger entities into smaller ones and establishing relationships between them.

Example: Taking an e-commerce website, we have to construct the ERD first so that we can build the database.

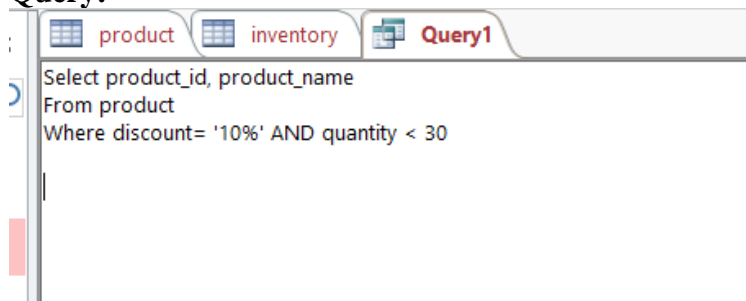


Now, taking this ERD, the database was designed.

## SQL queries and output:

1. Find the product ID and product name of those products, which have a discount of 10% and have less than 30 quantities.

**Query:**



**Output:**

product_id	product_name
1007	Huawei smart watch gt 2
0	

2. Find the average price of products that are accessories.



### Query:

product	inventory	Query1
Select AVG(unit_selling_price)		
From product		
Where product_type='accessories'		

### Output:

product	inventory	Tasnia1	Tasnia2
Expr1000			
\$1,500.00			

### 3. Find the max price of smartphones.

### Query:

product	inventory	Query1
Select MAX(unit_selling_price)		
From product		
Where product_type= 'smartphone'		

### Output:

product	inventory	Tasnia1	Tasnia2	Tasnia3
Expr1000				
\$55,499.00				

### 4. What is the reorder point of the inventory with product ID “1001” ?

**Query:**

product	inventory	Query1
Select reorder_point		
From inventory		
Where product_id = 1001		

**Output:**

product	inventory	Query1
reorder_point		
2		
*	p	

**5. What is the ratio of max/min price of products?**

**Query:**

product	inventory	Query1
Select MAX(unit_selling_price)/MIN(unit_selling_price)		
From product		

**Output:**



7. Show the total revenue earned by selling smartphones and the smartphone name as well.

Query:

product	inventory	Query1
Select unit_selling_price*quantity, product_name		
From product		
Where product_type= 'smartphone'		

Output:

product	inventory	Query1
Expr1000	product_name	
\$569,970.00	Realme C55	
\$319,600.00	Symphony 180	
\$599,950.00	Symphony Z55	
\$1,664,970.00	Samsung galaxy	
\$399,980.00	Vivo Y22	
\$319,990.00	Vivo V25e	
\$250,000.00	Samsung galaxy	
*		

8. Find the difference between the MAX and Min price of smartphones.

Query:

```

Query1
SELECT Max(unit_selling_price) - MIN(unit_selling_price)
FROM product
WHERE product_type = 'smartphone'

```

**Output:**

Expr1000	
\$47,509.00	

9. Show the customer ID and customer name of those who are from Rajshahi city and are female.

**Query:**

```

customer Query1
SELECT customer_id , customer_name
From customer
WHERE city = "Rajshahi" and gender = "Female"

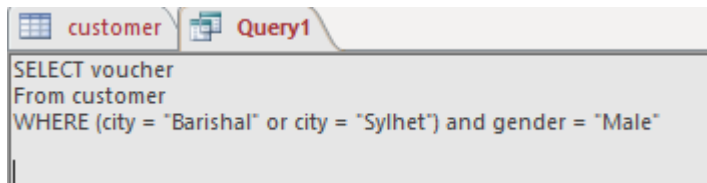
```

**Output:**

customer_id	customer_name
4028	Anushka
*	0

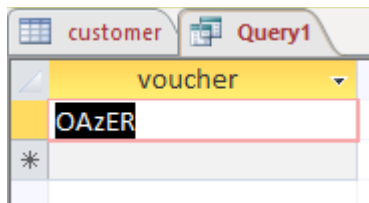
10. Show the voucher of male customers who are from Barishal city or Sylhet city.

**Query:**



```
customer Query1
SELECT voucher
From customer
WHERE (city = "Barishal" or city = "Sylhet") and gender = "Male"
```

## Output:



voucher
OAzER

## NORMALIZATION:

Normalization in database management is a process of organizing data in a database in order to eliminate redundancy and improve data integrity. It involves breaking down a database into multiple tables and establishing relationships between them.

The main objectives of normalization are:

1. Eliminating data redundancy: Redundancy refers to storing the same data in multiple places within a database. This can lead to inconsistencies and anomalies when updating or deleting data. By normalizing the database, redundant data is eliminated, and each piece of data is stored in only one place, reducing the chances of inconsistency.
2. Ensuring data integrity: Normalization helps maintain the accuracy and consistency of data by enforcing integrity constraints. These constraints define rules that must be followed for data to be considered valid. By organizing data into separate tables and establishing relationships, normalization allows for the enforcement of integrity constraints such as primary keys, foreign keys, and referential integrity.
3. Improving query performance: Normalization can also improve the performance of database queries. By breaking down data into smaller, more manageable tables, queries can be more efficient and targeted. This reduces the amount of data that needs to be processed and retrieved, resulting in faster query execution.

Normalization is typically achieved through a series of normalization forms, known as normal forms. The most commonly used normal forms are:

1. First Normal Form (1NF): In 1NF, data is organized into tables, and each column contains atomic values (indivisible and non-repeating).
2. Second Normal Form (2NF): In 2NF, the database is in 1NF, and all non-key attributes are dependent on the entire primary key, eliminating partial dependencies.

3. Third Normal Form (3NF): In 3NF, the database is in 2NF, and no non-key attribute is transitively dependent on the primary key. This eliminates transitive dependencies.

There are additional normal forms beyond 3NF, such as Boyce-Codd Normal Form (BCNF) and Fourth Normal Form (4NF), which address more complex dependencies and further reduce redundancy. The choice of which normal form to apply depends on the specific requirements and complexity of the database design.

By normalizing a database, you can achieve a more efficient and reliable data structure that minimizes redundancy, enforces integrity, and facilitates effective data management.

Example of Normalization:

Step 1.1: Removing Repeating Groups.

<u>OrderID</u>	<u>Order Date</u>	<u>Customer ID</u>	<u>Customer Name</u>	<u>Customer Address</u>	<u>ProductID</u>	<u>Product Description</u>	<u>Product Finish</u>	<u>Product StandardPrice</u>	<u>Ordered Quantity</u>
1006	10/24/2010	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
					5	Writer's Desk	Cherry	325.00	2
					4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2010	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
					4	Entertainment Center	Natural Maple	650.00	3

<u>OrderID</u>	<u>Order Date</u>	<u>Customer ID</u>	<u>Customer Name</u>	<u>Customer Address</u>	<u>ProductID</u>	<u>Product Description</u>	<u>Product Finish</u>	<u>Product StandardPrice</u>	<u>Ordered Quantity</u>
1006	10/24/2010	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2010	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2010	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2010	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
1007	10/25/2010	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

Step 1.2: Selection of Primary Key.

- Primary key should determine all the attributes. Below we have the functional dependencies.
- OrderID → OrderDate, CustomerID, CustomerName, C[REDACTED]s
- CustomerID → CustomerName, CustomerAddress
- ProductID → ProductDescription, ProductFinish, ProductStandardPrice

- OrderID, ProductID → OrderedQuantity
- Thus OrderID and ProductID would be the primary key

OrderID	Order Date	Customer ID	Customer Name	Customer Address	ProductID	Product Description	Product Finish	Product StandardPrice	Ordered Quantity
1006	10/24/2010	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2010	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2010	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2010	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
1007	10/25/2010	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

### Step 2: Convert to Second Normal Form.

- **Second normal form (2NF)** A relation in first normal form in which every nonkey attribute is fully functionally dependent on the primary key.
- **Partial functional dependency** A functional dependency in which one or more nonkey attributes are functionally dependent on part (but not all) of the primary key. •

#### **Normalization steps**

- Create a new relation for each primary key attribute (or combination of attributes) that is a determinant in a partial dependency. That attribute is the primary key in the new relation.
- Move the nonkey attributes that are dependent on this primary key attribute (or attributes) from the old relation to the new relation.

<u>OrderID</u>	OrderDate	CustomerID	CustomerName	CustomerAddress	CUSTOMER ORDER
<u>ProductID</u>	ProductDescription	ProductFinish	ProductStandardPrice	PRODUCT	
<u>OrderID</u>	<u>ProductID</u>	Ordered Quantity	ORDERLINE		

- A relation that is in first normal form will be in second normal form if any one of the following conditions applies:
  1. The primary key consists of only one attribute. By definition, there cannot be a partial dependency in such a relation.
  2. No nonkey attributes exist in the relation (thus all of the attributes in the relation are components of the primary key). There are no functional dependencies in such a relation.
  3. Every nonkey attribute is functionally dependent on the full set of primary key attributes (e.g., the attribute OrderedQuantity in the ORDER LINE relation).

### Step 3: Convert to third Normal Form.

- **Third normal form (3NF)** A relation that is in second normal form and has no transitive dependencies.
- **Transitive dependency** A functional dependency between the primary key and one or more nonkey attributes that are dependent on the primary key via another nonkey attribute.



- Steps of removing Transitive dependencies:

1. For each nonkey attribute (or set of attributes) that is a determinant in a relation, create a new relation. That attribute (or set of attributes) becomes the primary key of the new relation.

2. Move all of the attributes that are functionally dependent on the primary key of the new relation from the old to the new relation. 3. Leave the attribute that serves as a primary key in the new relation in the old relation to serve as a foreign key that allows you to associate the two relations.

