Multi Choice Type Question A company has employees and each employee belongs to a department. A department can have multiple employees. Which type of relationship is described in this scenario? Marks : 1 Negative Marks : 0 image Answer here One-to-One (1-1) One-to-Many (1-n) Many-to-Many (n-n) None of these options

The scenario describes:

* **Each employee belongs to a department** → an employee is associated with **one** department.
* **A department can have multiple employees** → one department has **many** employees.

This is a **One-to-Many (1-n)** relationship.

**Answer:**  
**One-to-Many (1-n)**

**You said:**

Multi Choice Type Question A university has students and courses. Each student can enroll in multiple courses, and each course can have multiple students. Which type of relationship is described in this scenario? Marks : 1 Negative Marks : 0 image Answer here One-to-One (1-1) One-to-Many (1-n) Many-to-Many (n-n) None of these options

In this scenario:

* **Each student can enroll in multiple courses** → a student relates to **many courses**.
* **Each course can have multiple students** → a course relates to **many students**.

This describes a **Many-to-Many (n-n)** relationship.

**Answer:**  
**Many-to-Many (n-n)**

**You said:**

Multi Choice Type Question You have a database table named "Employees" with columns "Id", "Name", and "DepartmentId". You want to retrieve all employees who belong to the "Sales" department. The "Sales" department has a department ID of 1. Which of the following LINQ queries can be used? Marks : 1 Negative Marks : 0 image Answer here var employees = dbContext.Employees.Find(e => e.DepartmentId == 1); var employees = dbContext.Employees.Select(e => e.DepartmentId == 1); var employees = dbContext.Employees.Where(e => e.DepartmentId == 1); var employees = dbContext.Employees.All(e => e.DepartmentId == 1);

To retrieve all employees whose DepartmentId is 1, the correct LINQ method is **Where**, which filters a collection based on a condition.

* Find is used in some contexts for primary key lookups, not for filtering multiple records.
* Select is used for projecting data, not filtering.
* All checks if **all elements** satisfy a condition, not for retrieving filtered results.

**Correct answer:**

var employees = dbContext.Employees.Where(e => e.DepartmentId == 1);

**You said:**

Multi Choice Type Question You have a database table named "Books" with columns "Id", "Title", and "AuthorId". You want to retrieve all books along with their corresponding authors. Which of the following LINQ queries can be used? Marks : 1 Negative Marks : 0 image Answer here var books = dbContext.Books.Select(b => b.AuthorId).ToList(); var books = dbContext.Books.FindAll(b => b.AuthorId).ToList(); var books = dbContext.Books.Include(b => b.Author).ToList(); var books = dbContext.Books.All(b => b.AuthorId).ToList();

To retrieve all books **along with their corresponding authors**, you need to **eagerly load** the related Author entity. In Entity Framework, this is done using Include.

* Select(b => b.AuthorId) only retrieves the AuthorId, not the full author details.
* FindAll is not valid in EF Core for this scenario.
* All checks a condition for all records, not for fetching related data.

**Correct answer:**

var books = dbContext.Books.Include(b => b.Author).ToList();

**You said:**

Multi Choice Type Question You have a database table named "Products" with columns "Id", "Name", and "Price". You want to retrieve all products with a price greater than $100. Which of the following LINQ queries can be used? Marks : 1 Negative Marks : 0 image Answer here var products = dbContext.Products.Where(p => p.Price > 100); var products = dbContext.Products.Find(p => p.Price > 100); var products = dbContext.Products.Select(p => p.Price > 100); var products = dbContext.Products.All(p => p.Price > 100);

To retrieve all products with a price greater than 100, the correct LINQ method is **Where**, which filters records based on a condition.

* Find is used for primary key lookups, not filtering multiple records.
* Select(p => p.Price > 100) would return a sequence of booleans, not the filtered products.
* All(p => p.Price > 100) checks if **all products** satisfy the condition, not filter them.

**Correct answer:**

var products = dbContext.Products.Where(p => p.Price > 100);

**You said:**

Multi Choice Type Question A blog has authors and posts. Each author can write multiple posts, and each post belongs to a single author. Which type of relationship is described in this scenario? Marks : 1 Negative Marks : 0 image Answer here One-to-One (1-1) One-to-Many (1-n) Many-to-Many (n-n) None of these options

In this scenario:

* **Each author can write multiple posts** → one author relates to **many posts**.
* **Each post belongs to a single author** → each post relates to **one author**.

This describes a **One-to-Many (1-n)** relationship.

**Answer:**  
**One-to-Many (1-n)**

**You said:**

Multi Choice Type Question You have a database table named "Orders" with columns "Id", "CustomerId", and "TotalAmount". You want to retrieve all orders placed by a specific customer with a customer ID of 100. Which of the following LINQ queries can be used? Marks : 1 Negative Marks : 0 image Answer here var orders = dbContext.Orders.Select(o => o.CustomerId == 100); var orders = dbContext.Orders.Find(o => o.CustomerId == 100); var orders = dbContext.Orders.Where(o => o.CustomerId == 100); var orders = dbContext.Orders.All(o => o.CustomerId == 100);

To retrieve all orders for a specific customer, you should **filter** the records using Where.

* Select(o => o.CustomerId == 100) would return a sequence of booleans, not orders.
* Find is for primary key lookups, not filtering multiple records.
* All checks if **all orders** satisfy the condition, not for retrieving filtered orders.

**Correct answer:**

var orders = dbContext.Orders.Where(o => o.CustomerId == 100);

**You said:**

Multi Choice Type Question You have a database table named "Students" with columns "Id", "Name", "Age", and "Grade". You want to retrieve all students who are in the 9th grade and are above 15 years old. Which of the following LINQ queries can be used? Marks : 1 Negative Marks : 0 image Answer here var students = dbContext.Students.Where(s => s.Grade == 9 && s.Age > 15).ToList(); var students = dbContext.Students.FindAll(s => s.Grade == 9 && s.Age > 15).ToList(); var students = dbContext.Students.Select(s => s.Grade == 9 && s.Age > 15).ToList(); var students = dbContext.Students.All(s => s.Grade == 9 && s.Age > 15).ToList();

To retrieve all students in the 9th grade **and** older than 15, you need to **filter** the records using Where.

* FindAll is not used in EF Core.
* Select would return booleans, not the filtered students.
* All checks if **all students** satisfy the condition, not filter them.

**Correct answer:**

var students = dbContext.Students.Where(s => s.Grade == 9 && s.Age > 15).ToList();

**You said:**

Multi Choice Type Question You have a database table named "Employees" with columns "Id", "Name", "Salary", and "DepartmentId". You want to retrieve the names of all employees who earn a salary higher than $50,000. Which of the following LINQ queries can be used? Marks : 1 Negative Marks : 0 image Answer here var employees = dbContext.Employees.Select(e => e.Name).Where(e => e.Salary > 50000).ToList(); var employees = dbContext.Employees.FindAll(e => e.Salary > 50000).Select(e => e.Name).ToList(); var employees = dbContext.Employees.Where(e => e.Salary > 50000).Select(e => e.Name).ToList(); var employees = dbContext.Employees.All(e => e.Salary > 50000).Select(e => e.Name).ToList();

To get the **names of employees** earning more than $50,000, you need to first **filter** with Where and then **project** the Name using Select.

* The first option applies Select before Where, which will fail because after Select(e => e.Name) you no longer have access to Salary.
* FindAll is not used in EF Core.
* All checks if **all employees** satisfy the condition, not filter them.

**Correct answer:**

var employees = dbContext.Employees.Where(e => e.Salary > 50000).Select(e => e.Name).ToList();

**You said:**

Multi Choice Type Question You have a database table named "Students" with columns "Id", "Name", and "Age". Which of the following LINQ queries can be used to retrieve all students from the database? Marks : 1 Negative Marks : 0 image Answer here var students = dbContext.Students.All(); var students = dbContext.Students.FindAll(); var students = dbContext.Students.ToList(); var students = dbContext.Students.SelectAll();

To retrieve **all students** from the database in EF Core, you use **ToList()**.

* All() checks a condition for all elements, not for retrieving data.
* FindAll() is not available in EF Core.
* SelectAll() does not exist in LINQ/EF Core.

**Correct answer:**

var students = dbContext.Students.ToList();

**You said:**

Multi Choice Type Question In a Products and Categories relationship, each product belongs to one category, and each category can have multiple products. How would you write a LINQ query to retrieve all products in the "Electronics" category? Marks : 1 Negative Marks : 0 image Answer here var products = dbContext.Products.Where(p => p.Category.Name == "Electronics").ToList(); var products = dbContext.Products.FindAll(p => p.Category.Name == "Electronics"); var products = dbContext.Products.Select(p => p.Category.Name == "Electronics").ToList(); var products = dbContext.Products.All(p => p.Category.Name == "Electronics").ToList();

To retrieve all products in the "Electronics" category, you need to **filter** products based on their related Category's Name using Where.

* FindAll is not available in EF Core.
* Select would return booleans, not the filtered products.
* All checks if **all products** satisfy the condition, not for filtering.

**Correct answer:**

var products = dbContext.Products.Where(p => p.Category.Name == "Electronics").ToList();

**You said:**

Multi Choice Type Question You have two tables, Authors and Books, where each author can write multiple books, and each book is written by one author. Which LINQ query would you use to get all books written by a specific author with authorId of 5? Marks : 1 Negative Marks : 0 image Answer here var books = dbContext.Books.Where(b => b.AuthorId == 5).ToList(); var books = dbContext.Books.Select(b => b.AuthorId == 5).ToList(); var books = dbContext.Books.FindAll(b => b.AuthorId == 5); var books = dbContext.Books.All(b => b.AuthorId == 5);

To get all books written by the author with AuthorId = 5, you need to **filter** the Books table using Where.

* Select(b => b.AuthorId == 5) would return booleans, not the books.
* FindAll is not used in EF Core.
* All checks if **all books** satisfy the condition, not for retrieving filtered records.

**Correct answer:**

var books = dbContext.Books.Where(b => b.AuthorId == 5).ToList();

**You said:**

Multi Choice Type Question In Entity Framework Core, when using LINQ to query data involving a one-to-many relationship, what does the Include method do? Marks : 1 Negative Marks : 0 image Answer here It filters the data based on a specific condition. It includes related entities in the query results. It specifies the projection of the data to be returned. It executes raw SQL queries directly on the database.

In Entity Framework Core, the **Include** method is used to **eagerly load related entities** in a query, especially in one-to-many or many-to-many relationships.

* It does **not** filter data (Where does that).
* It does **not** project data (Select does that).
* It does **not** execute raw SQL.

**Correct answer:**  
**It includes related entities in the query results.**

**You said:**

Multi Choice Type Question In a many-to-many relationship using Entity Framework Core, how is the join table typically represented, and how does LINQ query it? Marks : 1 Negative Marks : 0 image Answer here The join table is represented as a separate entity class with LINQ queries involving direct manipulation of this class. The join table is automatically created and managed by EF Core, and LINQ queries use navigation properties of the related entities. The join table is represented as a dictionary in LINQ queries for direct key-value access. The join table is not represented explicitly, and LINQ queries use only primary keys for data retrieval.

In EF Core (5.0 and later), for a **many-to-many relationship**:

* The **join table** is automatically created and managed by EF Core if you don’t define it explicitly.
* LINQ queries typically use the **navigation properties** of the related entities to access related data, without directly querying the join table.

**Correct answer:**  
**The join table is automatically created and managed by EF Core, and LINQ queries use navigation properties of the related entities.**

**You said:**

Multi Choice Type Question In a Customers and Orders relationship, each customer can have multiple orders, but each order belongs to only one customer. How would you write a LINQ query to retrieve all orders for a specific customer with customerId of 10? Marks : 1 Negative Marks : 0 image Answer here var orders = dbContext.Orders.Where(o => o.CustomerId == 10).ToList(); var orders = dbContext.Orders.Select(o => o.CustomerId == 10).ToList(); var orders = dbContext.Orders.FindAll(o => o.CustomerId == 10); var orders = dbContext.Orders.All(o => o.CustomerId == 10);

To retrieve all orders for the customer with CustomerId = 10, you need to **filter** the Orders table using Where.

* Select(o => o.CustomerId == 10) returns booleans, not orders.
* FindAll is not used in EF Core.
* All checks if **all orders** satisfy the condition, not filter them.

**Correct answer:**

var orders = dbContext.Orders.Where(o => o.CustomerId == 10).ToList();

Top of Form

Bottom of Form