**1) Explain what is ADO.NET entity framework?**

ADO.NET entity framework is an ORM (Object Relational Mapping) framework developed by Microsoft. It is an extension of ADO.NET that provides an automated mechanism to access and store data in the database. With the help of ADO.NET, database can be accessed without much required programming or code.

**2) Mention what is the key advantage of using Entity Framework or EF?**

The main advantage of using Entity Framework or EF is that it generates code automatically for the Model (Middle Layer), Mapping code and Data Access Layer. It reduces a lot of time during the development process.

**3) Mention in what all scenarios Entity Framework can be applicable?**

Entity Framework can be applicable in three scenarios

If you have existing database already or you want to build your database first than other parts of the application

If your prime focus is your domain classes and then create the database from your domain classes

If you want to design your database schema on the visual designer and create the classes and database

**4) Explain how you can load related entities in EF (Entity Framework)?**

You can load related entities or data in EF in three ways

Eager Loading

Lazy Loading

Explicit Loading

**Eager Loading**  
Eager Loading helps you to load all your needed entities at once; i.e., all your child entities will be loaded at single database call. **This can be achieved, using the Include method**, which returns the related entities as a part of the query and a large amount of data is loaded at once.

For example, you have a User table and a UserDetails table (related entity to User table), then you will write the code given below. Here, we are loading the user with the Id equal to userId along with the user details.

1. User usr = dbContext.Users.Include(a => a.UserDetails).FirstOrDefault(a => a.UserId == userId);

If you have multiple level of child entities, then you can load, using the query given below.

1. User usr = dbContext.Users.Include(a => a.UserDetails.Select(ud => ud.Address)).FirstOrDefault(a => a.UserId == userId);

**Lazy Loading(By default Lazy Loading is on)**  
**It is the default behaviour of an Entity Framework**, where a child entity is loaded only when it is accessed for the first time. It simply delays the loading of the related data, until you ask for it.

For example, when we run the query given below, UserDetails table will not be loaded along with the User table.

1. User usr = dbContext.Users.FirstOrDefault(a => a.UserId == userId);

It will only be loaded when you explicitly call for it, as shown below. UserDeatils ud = usr.UserDetails; // UserDetails are loaded here

**Explicit Loading**  
There are options to disable Lazy Loading in an Entity Framework. **After turning Lazy Loading off, you can still load the entities by explicitly calling the Load method for the related entities**. There are two ways to use Load method Reference (to load single navigation property) and Collection (to load collections), as shown below.

1. User usr = dbContext.Users.FirstOrDefault(a => a.UserId == userId);
2. dbContext.Entry(usr).Reference(usr => usr.UserDetails).Load();

When to use what

1. Use Eager Loading when the relations are not too much. Thus, Eager Loading is a good practice to reduce further queries on the Server.
2. Use Eager Loading when you are sure that you will be using related entities with the main entity everywhere.
3. Use Lazy Loading when you are using one-to-many collections.
4. Use Lazy Loading when you are sure that you are not using related entities instantly.
5. When you have turned off Lazy Loading, use Explicit loading when you are not sure whether or not you will be using an entity beforehand.

**5) What is Navigation Property?**  
It is a way to represent a foreign key relationship in the database or define the relationship between the two entities. A navigation property points to related tables in your database model. For instance, if you have a customer table that relates to the orders table, the navigation property points from the customers table to the orders table. While the primary and foreign keys are physical properties of the table, the navigation properties are a logical part of a data model.  
  
**Example**Client - Client ID, Name.  
  
Project - Project ID, Client ID.  
  
It means the project is linked with the client on the basis of Client ID.  
  
Relationship: Project HAS-a client.  
  
If we model this relationship, it would look like as shown below:

1. class Client
2. {
3. **public** **string** Name {**get**; **set**}
4. }
6. cass Project
7. {
8. **public** Client objClient {**get**; **set**}
9. }

It means we can do this in the following manner:  
  
Project.objClient.Name;  
  
Now, if we want to achieve this structure in the database, we need to define our model in the way shown below:

1. class Client
2. {
3. **public** int Id {**get**; **set** ;}
4. **public** **string Name** {**get**; **set** ;}
5. }
7. class Project
8. {
9. public **int** Id {**get**; **set** ;}
10. public Client client {**get**; **set** ;} // Project HAS-a client. Name of Field will be
11. // [PropertyName]\_[PropertyIDName] “client\_Id”
12. }

**Note**Association between two entities defines three things: type of entity, type of association, and referential integrity.   
  
**6) How Navigation property works**When we apply navigation property in our code, it means we are asking EF to automatically perform a join between the two tables.  
  
**Example**Code accesses project info of the client name ASD, as shown below:

1. Context.Projects.Where(p ->p.client.Name == ‘ASD’);

Now, SQL translation of the code, given above is as follows:

1. **Select** Projects .Id, Projects .client\_Id
2. **From** Projects **inner** join Clients
3. **On** Projects.client\_Id = Clients.Id
4. **Where** Clients.**Name** = ‘ASD’

**7) How and when navigation properties load**  
**By default, navigation properties are null, they are not loaded by default**.  For loading navigation property, we use “include” method of IQuearable and this type of loading is called Eager loading.  
  
Eager loading: It is a process by which a query for one type of entity loads the related entities as a part of query and it is achieved by “include” method of IQuerable.   
  
**8) How Navigation properties loading works,**

1. class Client
2. {
3. **public** **int** ID {get; **set** ;}
4. **public** string **Name** {get; **set** ;}
5. }
7. Class Project
8. {
9. **public** **int** ID {get; **set** ;}
10. **public** Client Clients {get; **set** ;} // Project HAS-a client. **Name** **of** Field will be
11. [PropertyName]\_[PropertyIDName] “Clients\_ID”
12. }

**Clients**

|  |  |
| --- | --- |
| ID | Name |
| 1 | CMD |
| 2 | NSH |

**Projects**

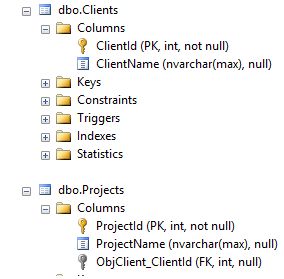
|  |  |
| --- | --- |
| ID | Clients\_ID |
| 100 | 1 |
| 101 | 1 |

1. Get all the projects but do not get the linked clients.
   1. Context.Projects.ToArray()

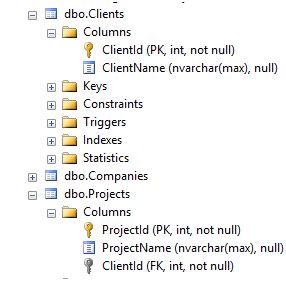
   4. <ArrayOfProject>
   5. <Project>
   6. <ID>100</ID>
   7. <Clients>null</Clients>
   8. </Project>
   9. <Project>
   10. <ID>101</ID>
   11. <Clients>null</Clients>
   12. </Project>
   13. </ArrayOfProjects>
2. Filter, using where clause, which works perfectly and filters the records, but the linked property clients are not loaded.
   1. Context.Project.Where(p =>p.Clients.Name==”CMD”) .ToArray()
   3. <ArrayOfProject>
   4. <Project>
   5. <ID>100</ID>
   6. <Clients>**null**</Clients>
   7. </Project>
   8. </ArrayOfProjects>
3. Include statement is able to fetch or load the linked item of an entity.
   1. Context.Project.include(p =>p.Clients).ToArray();
   3. <ArrayOfProject>
   4. <Project>
   5. <ID>100 </ID>
   6. <Client>
   7. <ID>1</ID>
   8. <Name> CMD</Name>
   9. </Client>
   10. </Project>
   11. <Project>
   12. <ID>101 </ID>
   13. <Client>
   14. <ID>1</ID>
   15. <Name> CMD</Name>
   16. </Client>
   17. </Project>
   18. <ArrayOfProject>

**9) How Entity Framework detect Navigation properties**

1. **Complex type**When we define complex type inside class, which is attached with the context for Entity framework, consider this complex type as a foreign key inside this class. Hence, it creates Foreign key by name [PropertyName]\_[PropertyIDName] (Name by which property is defined and the name of Primary key is of complex type.)  
     
   **Example**
   1. class Client
   2. {
   3. **public** **int** ClientId {**get**; **set**;}
   4. **public** **string** ClientName {**get**; **set**;}
   6. }
   8. Class Project
   9. {
   10. **public** **int** ProjectId {**get**; **set**;}
   11. **public** **string** ProjectName {**get**; **set**;}
   13. **public** Client ObjClient {**get**; **set**;}
   14. }

  
  
In the above example, we have two classes: Client and Project.  
  
Now, the client is defined inside the Project class. Hence, these two classes are attached with the context by dbset.Thus, entity frame considers it as a Foreign key.  
  
It will create a Foreign key inside the Project table by the following naming convention [PropertyName]\_[PropertyIDName], which means, it will look like “ObjClient\_ClientId”

1. **By Naming conventions**  
   When we define the complex type property inside a class and also add the property with the same name as a Primary key of complex type, the Entity Framework considers this and defines a property as a Foreign key of the table with the same name.  
     
   **Example**
   1. class Client
   2. {
   3. **public** **int** ClientId {**get**; **set**;}
   4. **public** **string** ClientName {**get**; **set**;}
   5. }
   6. class Project
   7. {
   8. **public** **int** ProjectId {**get**; **set**;}
   9. **public** **string** ProjectName {**get**; **set**;}
   11. **public** **int** ClientId {**get**; **set**;} // Entity framework treat it as foreign key, and create same
   12. // name property as Foreign key into table.
   13. **public** Client ObjClient {**get**; **set**;}
   14. }

  
  
In the above example in Project table, the Entity framework creates Foreign key by the name “ClientId”.

**10) Ways to define Foreign key in Entity Framework**

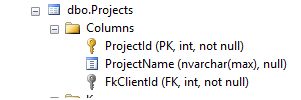
1. By adding complex type as a property,
   1. class Client
   2. {
   3. public intClientId {**get**; **set**;}
   4. public **string** ClientName {**get**; **set**;}
   6. }
   8. class Project
   9. {
   10. public intProjectId {**get**; **set**;}
   11. public **string** ProjectName {**get**; **set**;}
   13. public Client ObjClient {**get**; **set**;}
   14. }

Only add property of complex type, and it will automatically create “ObjClient\_ClientId” foreign key into project table.

1. By adding complex type as a property and adding the property by the same name and typing it as Primary key of a complex type,
   1. class Client
   2. {
   3. public **int** ClientId {**get**; **set**;}
   4. public **string** ClientName {**get**; **set**;}
   6. }
   8. class Project
   9. {
   10. public **int** ProjectId {**get**; **set**;}
   11. public **string** ProjectName {**get**; **set**;}
   13. public **int** ClientId {**get**; **set**;} // Entity framework treat it as foreign key, and create same
   14. // name property as Foreign key into table.
   15. public Client ObjClient {**get**; **set**;}
   16. }
2. By using data annotation Foreign key attribute,  
     
   Data annotation Foreign key attribute can be used in the two ways -- one with the navigation property defined inside the class or with a Foreign key, defined inside the class.  
     
   **Example**Foreign key attribute is used over a navigation property.
   1. class Project
   2. {
   3. **public** **int** ProjectId {**get**;**set**;}
   4. **public** **string** ProjectName{**get**; **set**;}
   6. [ForeignKey("FkClientId")]
   7. **public** Client client {**get**; **set**;} // Navigation property
   8. }

https://csharpcorner-mindcrackerinc.netdna-ssl.com/article/navigation-property-with-code-first-navigation-property-in-ef/Images/error%202.jpgHence, at this stage, it will give an error, because here we have specified a Foreign key. Hence, EF will not create a default foreign key by [PropertyName]\_[PropertyIdName]. Hence, EF tries to find Foreign key, specified by name “FkClientId”. It will give an error for not finding Foreign key by the FkClientId inside the Project class. Hence, we need to define Foreign key property.

* 1. class Project
  2. {
  3. **public** **int** ProjectId {**get**;**set**;}
  4. **public** **string** ProjectName{**get**; **set**;}
  5. **public** **int** FkClientId {**get**; **set**;} // It will become foreign key.
  6. [ForeignKey("FkClientId")]
  7. **public** Client client {**get**; **set**;} // Navigation property
  8. }

  
  
**Example**  
Foreign key attribute is used over the property with which you want to make Foreign key.

* 1. class Project
  2. {
  3. **public** **int** ProjectId {**get**;**set**;}
  4. **public** **string** ProjectName{**get**; **set**;}
  5. [ForeignKey ("Client")]
  6. **public** **int** FkClient {**get**; **set**;}
  7. }

https://csharpcorner-mindcrackerinc.netdna-ssl.com/article/navigation-property-with-code-first-navigation-property-in-ef/Images/Error.jpg  
On this stage, the code given above will give an error, because in the code given above, we are saying tha, FkClient is a Foreign key of Client table, but EF will not create this foreign key due to the absence of the navigation property of Client, it means at least one navigation property of client should be there adding its key as a foreign key into the project.

* 1. class Project
  2. {
  3. **public** **int** ProjectId {**get**;**set**;}
  4. **public** **string** ProjectName{**get**; **set**;}
  5. [ForeignKey ("Client")]
  6. **public** **int** FkClient {**get**; **set**;}
  7. **public** Client Client{**get**;**set**;} // Navigation Property.
  8. }

1. By using data annotation Foreign key attribute, we can give different name to Foreign key property.

**Note**The first part of the expression defines the navigation property on the current entity, the second part of the expression defines the reverse navigation property.  
  
**11) What is Reverse Navigation property?**

Here, we will discuss reverse navigation property.  
  
Two navigation properties that we have defined on each end of a relationship are in fact different ends of the same relationship.  
  
It has been able to do this because there has been only one possible match.   
  
Type of Navigation properties,

1. Optional relationship (Null able foreign key and multiplicity, 0...1)(One to one relationship )(Zero or one).
2. Required relationship (One to many relationship).
3. Many relationship (Many to Many relationship).

**One-to-One Relationship**  
Optional relationship (Null able foreign key and multiplicity, 0...1)(One to one relationship )(Zero or one).  
  
Primary key value table contains one record and its related table contains one or zero records related to this table. This relationship is called a one to one relationship.  
  
Suppose, take an example of “Client” and “ClientAddress”, which has a one to one relationship (zero or one),this means *the client can have one or zero addresses.*  
Hence, if we create a class of the client having ID and Name; create Class ClientAddress contains ID , Address and City.

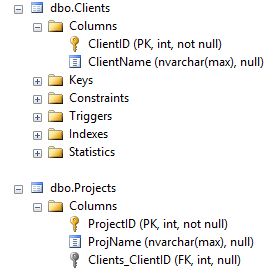
For setting up one to one relationships between the Client and ClientAddress, we need to define the navigation property of the client into ClientAddress and specify a Foreign Key of client into ClientAddress and then define the reverse navigation property of ClientAddress into the client. In this way, EF handles One to One relationships.

1. class Client
2. {
3. **public** **int** ClientID{**get**;**set**;};
4. **public** String Name{**get**;**set**;};
5. }
6. clientAddress
7. {
9. **public** **int** ClientAddressId {**get**;**set**;}
10. **public** String Address {**get**;**set**;}
11. **public** **string** City {**get**;**set**;}
12. [ForeignKey ("Client")]
13. **public** **int** FkClient {**get**; **set**;}
14. **public** Client Client{**get**;**set**;} // Navigation Property.
15. }

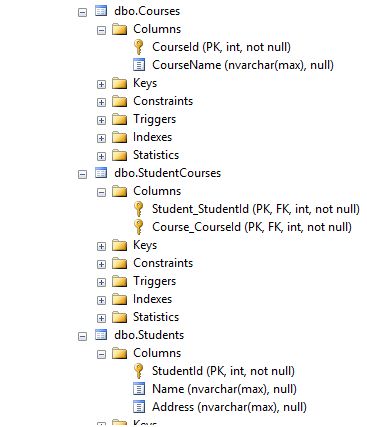
**One-to-many relationships**  
Primary key value table contains one record and its related table contains one, zero or many records related to it. This relationship is called a One-to-Many relationship.  
  
**Example**Suppose we have “Client” and “Project” entity.   
  
*One client can have multiple projects, but one project can only link with one project.*  
  
**Note**Reverse navigation property of the expression is defined as a collection. In this case, there is a one-to-many relationship.

1. class Client
2. {
3. **public** **int** ClientID{**get**;**set**;}
4. **public** String ClientName{**get**;**set**;}
5. // Second Part of expression define reverse navigation property
6. **public** **virtual** ICollection<Project> Projects { **get**; **set**; }
7. }


11. class Project
12. {
13. **public** **int** ProjectID{**get**;**set**;}
14. **public** String ProjName {**get**;**set**;}
15. // First Part of expression define navigation property
16. **public** Client Clients {**get**;**set**}
17. }

  
  
Hence, in the code given above, you can see the client type contains a collection of Project type. Project type contains client type entity.   
  
These are the two ends of the expression of the navigation properties and it shows that the client can have a multiple project due to which, the project is defined in the collection inside the client.  
  
On the other hand, project only links with the single client, due to which it contains the client type entity.   
  
**Many-to-Many relationship**  
One table records are able to relate with any numbers (or zero records) of records of the second, the second table records are able to relate with any number of records of the first table.   
  
*Many-to-many relationship require a third table, which is called linking or mapping the table.  
  
Example: relationship between “Student” and “Course” entity,*   
  
One student can be enrolled for multiple courses and one course can be taught to many students.   
  
Hence, in this case, a third table is required, which will contain Pk of both entities.

1. class Student
2. {
3. **public** **int** StudentId {**get**;**set**;}
4. **public** String Name {**get**;**set**;}
5. **public** String Address {**get**; **set**; }
6. **public** **virtual** ICollection<Course > Courses { **get**; **set**; }
8. }
10. class Course
11. {
12. **public** **int** CourseId {**get**; **set**;}
13. **public** String CourseName {**get**;**set**;}
14. **public** **virtual** ICollection<Student>Students { **get**; **set**; }
15. }



**12) How can you enhance the performance of Entity Framework?**

To enhance the performance of Entity Framework, you have to follow the following steps

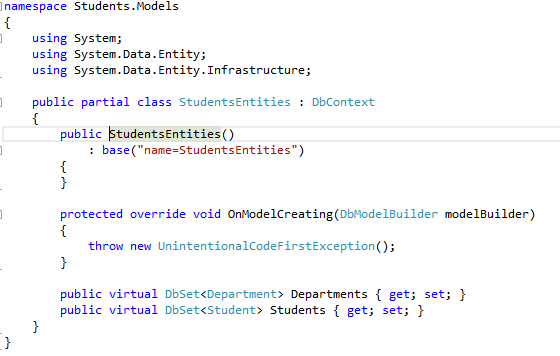
* Try to avoid to put all the DB objects into one single entity model
* Disable change tracking for entity if not needed
* Reduce response time for the first request by using pre-generating Views
* If not required try to avoid fetching all the fields
* For data manipulation select appropriate collection
* Wherever needed use compiled query
* Avoid using Views and Contains
* While binding data to grid or paging, retrieve only required no of records
* Debug and Optimize LINQ query

**13) Explain why T4 entity is important in Entity Framework?**

T4 entity is important in Entity framework as it is the heart of entity framework code generation. **It reads the EDMX XML file and generate C# behind code.**

**14) Where do we use Virtual classes in Entity Framework DbContext Models?**

We use Virtual classes in Entity Framework in context class where we define DBSet of corresponding table. As we can see easily in below code sample for Students and Departments:



**15) What’s new in Entity Framework 6?**

* Customizing Code First Conventions.
* Logging of database commands.
* Stored Procedure Mapping.
* Asynchronous Queries and Save support.

**16) How we can use a property for an Entity that is not mapped to database in Code First Approach?**

In Code First Approach, all properties are normally mapped to database. But in certain scenarios, if we don’t want a specific property to be mapped to database, Entity Framework provides us with NotMapped attribute for annotation.

Let’s take a practical example, we have an entity i.e. Product and we want to have a ProductCode that will not be mapped to database. So, we will do the following to achieve in Code First Approach.

public class Product

{

[Key]

public int UniqueProductIdentifier{ get; set; }

public string ProductName { get; set; }

[NotMapped]

public string ProductCode

{

get{

return UniqueProductIdentifier + "-" + ProductName

}

}

public decimal Price { get; set; }

....

....

}

**17) If we have not followed proper conventions in Code First approach, then how we can mark a field/property as primary key?**

An important point to remember about EF Code First approach; we have to follow proper convention for a primary key, means Entity Framework expects ID combined with Entity Name. In our case, above referred Code First Example, we have two entities (i.e. Category and Product) and for both of these entities, we used proper convention for primary key i.e. CategoryId and ProductId.

But in case we used something different i.e. “UniqueCategoryIdentifier” or “UniqueProductIdentifier”, then we have to tell that this is our primary key column and this can be done using a simple attribute i.e. “Key” as follows:

public class Category

{

[Key]

public int UniqueCategoryIdentifier{ get; set; }

public string CategoryName{ get; set; }

public ICollection<Product> Products { get; set; }

}

public class Product

{

[Key]

public int UniqueProductIdentifier{ get; set; }

public string ProductName { get; set; }

public decimal Price { get; set; }

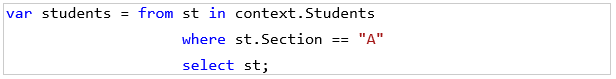
....

....

}

If we don’t follow the convention as well as not using the “Key” attribute, then will end up with following error.

**18) What is the query syntax we use to query an ADO.NET Entity Data Model?**

We can use LINQ to Query ADO.Net Entity Framework. For Example:

**19) Is LINQ a feature of Entity Framework?**

Yes, above is the example to get student record from Section A.

**20) Can you describe the feature of split entity in Entity Framework?**

Entity splitting gives us the ability to take an entity in our model and split this entity into multiple database tables. When we query an entity, Entity Framework will create a query that Automatically joins the related physical tables for us.

**21) How to handle transactions in Entity Framework 6**

Transaction being a single unit of work that are either successful or failed has really important for an application that is developed using Entity Framework.

**1. DbContext.Database.BeginTransaction:** Below we can find source code to understand that how different operations are combined within the same transaction and further are all committed or rollbacked. This method also allows us to specify the isolation level for the transaction.

using (EFModelFirst\_DemoContainer context = new EFModelFirst\_DemoContainer())

{

using (var transaction = context.Database.BeginTransaction())

try

{

Category c = new Category();

c.CategoryName = "Mobile";

context.Categories.Add(c);

context.SaveChanges();

Product p = new Product();

p.CategoryId = c.CategoryId;

p.ProductName = "HTC";

p.Price = "15000";

context.Products.Add(p);

context.SaveChanges();

transaction.Commit();

}

catch (Exception ex)

{

transaction.Rollback();

}

}

**2. DbContext.Database.UseTransaction:** It allows DbContext to use a transaction that was stated outside of the Entity Framework. It means using this API we can use any existing transaction with Entity Framework.

It is used when sometimes we must use an existing transaction that is started outside of the Entity Framework.

public EFModelFirst\_DemoContainer() : base("name=EFModelFirst\_DemoContainer")

{}

(SqlConnection con = new SqlConnection("connectionString"))

{

con.Open();

using (var transaction = con.BeginTransaction())

{

using (var transaction1 = context.Database.BeginTransaction()) {

try

{

Category c = new Category();

c.CategoryName = "Mobile";

context.Categories.Add(c);

context.SaveChanges();

Product p = new Product();

p.CategoryId = c.CategoryId;

p.ProductName = "HTC";

p.Price = "15000";

context.Products.Add(p);

context.SaveChanges();

transaction.Commit();

}

catch (Exception ex)

{

transaction.Rollback();

}

} }

**22) What are three Schema Definition Language in EDM?**

**SSDL** (Storage Schema Definition Language)

It stores the schema of database which we created

**CSDL** (Conceptual Schema Definition Language)

It stores the schema of cs file which generated for each table of database. This files defines the properties created for that table and all the relationship.

**ML** (Mapping Language) or C-S M Mapping

It stores the mapping of SSDL and CSDL.

**23) What is ObjectContext class in Entity Framework?**

ObjectContext is the class which helps us to perform Insert, Update, Delete and create in Entity Framework.

This class is available in System.Data.Entity namespace.

**24) Which namespace we use in Code first approach to use [key] attribute**

System.ComponentModel.DataAnnotation

**25) Which namespace we use in Code first approach to use [Foreign key] attribute**

System.ComponentModel.DataAnnotation.Schema

**26) How to disable the lazy loading framework?**

You can disable the lazy loading by

context.ContextOptions.LazyLoadingEnabled = false;

**27) What are the types of property in Entity Framework?**

We have three types of property in entity framework.

* Scalar property
* Navigation Property
* Complex Property

**28) What are scalar and navigation properties in Entity Framework?**

**Scalar properties** are the property which is directly links with the database.

e.g. Department name, Department Id are Scalar property.

**Navigation Properties** are property which is association with the other entity property.

e.g. Employees is Navigation property.

**29) What is client wins and store wins mode in Entity Framework concurrency?**

Client and Store wins are actions which we need to take when concurrency happens. In store Wins, data are loaded into entity objects and in Client wins data will store from client side to database.

**30) What are the components of Entity Framework Architecture?**

Below are the components of Entity Framework –

* Entity Data Model (EDM)
* LINQ to Entities
* Entity SQL
* Object Service
* Entity Client Data Provider
* ADO.Net Data Provider.

**31) What is Compiled Query?**

To avoide parsing overhead can be removed by compiling this linq query. To compile a Linq query we use CompiledQuery class and this class if present in System.Data.Linq

**//Without CompiledQuery**

static void Main() {

using(SampleDataContext dbContext = new SampleDataContext())

{

//Every time this linq query is executed this has to be parse and the transaction sql //statement will have to be dynamically generated on the fly so this parsing overhead can //be removed by compiling this linq query

Student stu = (from s in dbContext.Students

Where s.Id == 1

Select s).Single();

Console.WriteLine(stu.FirstName + “ “ + stu.LastName);

}

}

**//By using CompiledQuery**

static void Main() {

var compileStuQuery = CompiledQuery.Compile(

(SampleDataContext dataContext, int Stuid) =>

(from s in dataContext.Students

Where s.Id == Stuid

Select s).Single()

);

using(SampleDataContext dbContext = new SampleDataContext())

{

Student stu = compileStuQuery(dbContext, 1);

Console.WriteLine(stu.FirstName + “ “ + stu.LastName);

}

}

ADO.NET Entity Framework Version History

ADO.NET Entity Framework Version

Supported Framework & IDE

Features Detail

6.0

.NET Framework 4.5.1 and Visual Studio 2013

1. **Async Query and Save**
2. Code-Based Configuration
3. **Dependency Resolution**
4. Interception/SQL logging
5. Improved Connection Management
6. Improved Transaction Support

5.0

.NET Framework 4.5 and Visual Studio 2012

1. Enum Support in Code First and EF Designer
2. Spatial Data Types in Code First and EF Designer
3. **Table-Valued Functions**
4. Multiple Diagrams per Model

4.3

.NET Framework 4.0 and Visual Studio 2010

1. **Code First Migrations**
2. **Automatic Migrations**

4.2

.NET Framework 4.0 and Visual Studio 2010

1. The EF 4.2 release included the bug fixes to EF 4.1

4.1

.NET Framework 4.0 and Visual Studio 2010

1. Code First development
2. Introduced DbContext API
3. Data Annotations and Fluent API Validation

4.0

.NET Framework 4.0 and Visual Studio 2010

1. Model-first development
2. POCO support
3. Lazy Loading
4. T4 Code Generation

3.5

.NET Framework 3.5 SP1 and Visual Studio 2008 SP1

1. This release provided basic O/RM support using the Database first development