◇ Big Picture First (Say This Line)

“We implemented the ER diagram using Entity Framework Core with a Code-First approach.”

Meaning:

We **designed C# models**
EF Core **automatically created tables**
Relationships came from **foreign keys & navigation properties**

◇ Step 1: Convert ER Diagram → C# Models

Each **entity in ER diagram = one C# class**

Example: User Entity

You explain like this 👇

“The User table is implemented as a User class. The primary key is defined using attributes, and navigation properties represent relationships.”

Conceptually:

`UserId` → Primary Key
Other fields → Columns
Collections → One-to-Many relationships

◇ Step 2: Primary Keys & Foreign Keys

How PKs are handled

EF Core uses:

```
[Key]
public int UserId { get; set; }
```

How FKs are handled

Example logic you explain (no need to show code unless asked):

"Foreign keys like user_id and doc_id are added as properties, and EF Core automatically maps relationships."

So:

```
user_id → UserId
doc_id → DocumentId
```

🔷 Step 3: Relationships Implementation (Very Important)

1️⃣ One User → Many Documents

Explain like this:

"In the User model, we define a collection of Documents, and in the Document model, we define a foreign key and a navigation property back to User."

EF understands:

- One user
- Many documents
- Automatically enforces relationship

2️⃣ One Document → Many Versions

Your line 👇

“Each document can have multiple versions, so we created a one-to-many relationship between Document and DocumentVersion.”

Handled using:

- `DocumentId` as FK
- `ICollection<DocumentVersion>`

3️⃣ Inline Comments Relationships

Explain confidently:

“Inline comments are linked to the document, the exact version, and the user using foreign keys. This allows precise commenting on specific text ranges.”

This matches your ER diagram exactly.

🔷 Step 4: DbContext (Heart of EF)

Say this clearly:

“DbContext acts as a bridge between our application and the database.”

What you did:

- Created `ApplicationDbContext`
- Added:

```
DbSet<User>
DbSet<Document>
DbSet<DocumentVersion>
DbSet<InlineComment>
DbSet<ActivityLog>
DbSet<PasswordResetToken>
```

Explain:

“Each DbSet represents a table in the database.”

◇ Step 5: Fluent API (If Asked WHY)

You can say:

“Where annotations were not enough, we used Fluent API to define relationships and constraints clearly.”

Examples (in words):

- Cascade delete rules
- Required relationships
- Foreign key behavior

◇ Step 6: Migrations (Very Viva-Friendly)

This line is GOLD 🥇:

“We used EF Core migrations to convert our models into database tables in a controlled and versioned way.”

Flow:

- Create models
- Run `Add-Migration`
- Run `Update-Database`
- EF creates:
- Tables
- Keys
- Relationships

🔷 Step 7: Data Flow at Runtime

Explain this flow slowly 👇

User sends request (upload / comment / edit)
Controller receives request
Service layer processes logic
EF Core translates LINQ → SQL
SQL Server stores or retrieves data
Response sent back

🔷 Step 8: Why EF Core Was the Right Choice

Say this in conclusion:

Reduces manual SQL
Prevents SQL injection
Keeps code clean and maintainable
Easy schema changes using migrations
Strong mapping with ER diagram

🧠 One-Minute Summary (Memorize This)

"We implemented the ER diagram using EF Core Code-First approach. Each entity became a C# model, relationships were defined using foreign keys and navigation properties, DbContext connected the application to SQL Server, and migrations were used to generate and update the database schema."