1. Relationships Mapping

Entity Framework Core (EF Core) supports different types of relationships between entities:

- **One-to-One**
  Each entity is related to exactly one other entity.
  Example: *User* ↔ *Profile*.
- **One-to-Many**
  One entity can be related to many others.
  Example: *Category* → *Products*.
- **Many-to-Many**
  Multiple entities can be related to multiple others.
  Example: *Students* ↔ *Courses*.
    - EF Core 5+ supports skip navigations (no need for explicit join table).

Key tools:

- Navigation Properties (virtual ICollection<T>).
- Data Annotations ([ForeignKey], [InverseProperty]).
- Fluent API (HasOne, WithMany, WithMany).

---

2. Soft Delete

Soft delete = **logically deleting** a record instead of physically removing it.

- The record stays in the database with a flag (e.g., IsDeleted = true).
- Query filters are applied to hide deleted records from normal queries.

Advantages:

- Data recovery possible.
- Keeps history for auditing.

Disadvantages:

- Table size grows over time.
- Queries need filtering (HasQueryFilter).

---

3. Delete Behaviors

When deleting an entity that has relationships, EF Core defines how related entities are handled:

- **Cascade**
  Deletes related dependent entities automatically.
  Example: Deleting a Category deletes its Products.
- **Restrict**
  Prevents deletion if dependents exist.
  Example: Cannot delete a Category that still has Products.
- **NoAction**
  No action taken by EF Core; database handles behavior (depends on FK constraint).
- **SetNull** *(extra option)*
  Sets foreign key to NULL when the parent is deleted.

Default: Cascade for required relationships, ClientSetNull for optional.

---

4. Change Tracking

EF Core tracks entity states to know what to update in the database.

- **AsTracking (default)**
  - EF Core monitors changes.
  - Suitable for update and delete operations.
  - Uses more memory.
- **AsNoTracking**
  - EF Core does not track changes.
  - Read-only performance is much faster.
  - Best for reporting and queries without updates.

---

5. Round Trips to Database

A **round trip** = a request from EF Core to the database and back.

- **Problems with too many round trips**:
    - Slows down performance.
    - Increases network load.
- **How to reduce**:
    - Use **eager loading** (Include) instead of lazy loading multiple queries.
    - Use **bulk operations** where possible.
    - Apply **projection** (select only needed columns).
    - Use **batching** (SaveChanges groups multiple updates).