Certainly! Let's dive into each aspect of Database Design with two code examples for Entity-Relationship Diagrams (ERD), Normalization, and Denormalization.

1. Entity-Relationship Diagrams (ERD):

Entity-Relationship Diagrams are visual representations of the relationships among entities in a database. Entities are represented as rectangles, and relationships are represented as lines connecting the entities.

Example 1: ERD using Crow's Foot Notation

```plaintext
+--------------------+       +----------------------+
|     Customers      |       |       Orders         |
+--------------------+       +----------------------+
| PK CustomerID      |       | PK OrderID           |
| Name               |       | OrderDate            |
| Email              |       | FK CustomerID        |
| Phone              |       +----------------------+
+--------------------+       | FK ProductID         |
                             +----------------------+
                             | Quantity             |
                             +----------------------+
```

In this example, we have two entities: Customers and Orders. Customers have a primary key (PK) of CustomerID, and Orders have a PK of OrderID. There is a foreign key (FK) relationship between Orders and Customers using CustomerID.

Example 2: ERD using Chen's Notation

```plaintext
+--------------------+
|     Employees      |
+--------------------+
| PK EmployeeID      |
| FirstName          |
| LastName           |
| HireDate           |
| Position           |
+--------------------+
      |
      | 1      1
      |
```

```
+--------------------+
|     Departments    |
+--------------------+
| PK DepartmentID    |
| DepartmentName     |
| Location           |
+--------------------+
```

In this example, we have two entities: Employees and Departments. Employees have a PK of EmployeeID, and Departments have a PK of DepartmentID. There is a one-to-many relationship between Employees and Departments.

2. Normalization:

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing large tables into smaller ones and defining relationships between them.

Example 1: First Normal Form (1NF) - Removing Repeating Groups

```sql
-- Before normalization
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Courses VARCHAR(255)  -- Comma-separated list of courses
);

-- After normalization (1NF)
CREATE TABLE Student (
    StudentID INT PRIMARY KEY
);

CREATE TABLE StudentCourses (
    StudentID INT,
    Course VARCHAR(255),
    PRIMARY KEY (StudentID, Course),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
);
```

In this example, the original table had a repeating group (Courses). After normalization, we have two tables: Student and StudentCourses.

Example 2: Second Normal Form (2NF) - Removing Partial Dependencies

```sql
-- Before normalization
CREATE TABLE OrderDetails (
    OrderID INT PRIMARY KEY,
    ProductID INT,
    ProductName VARCHAR(255),
    Quantity INT
);

-- After normalization (2NF)
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY
);

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(255)
);

CREATE TABLE OrderItems (
    OrderID INT,
    ProductID INT,
    Quantity INT,
    PRIMARY KEY (OrderID, ProductID),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

In this example, the original table had a partial dependency of ProductName on the OrderID. After normalization, we have three tables: Orders, Products, and OrderItems.

3. Denormalization:

Denormalization is the process of combining tables back into a single table to improve query performance, especially for read-heavy operations.

Example 1: Combining Tables

```sql
-- Normalized Tables
CREATE TABLE Customers (
```

```
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255),
    Phone VARCHAR(20)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

-- Denormalized Table
CREATE TABLE DenormalizedOrders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    Name VARCHAR(255),
    Email VARCHAR(255),
    Phone VARCHAR(20),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

In this example, we denormalize the Orders and Customers tables into a single DenormalizedOrders table for improved query performance.

Example 2: Adding Redundant Data for Performance

```sql
-- Normalized Table
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(255),
    CategoryID INT,
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);

CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY,
    CategoryName VARCHAR(255)
);
```

```
-- Denormalized Table with Redundant Data
CREATE TABLE DenormalizedProducts (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(255),
    CategoryID INT,
    CategoryName VARCHAR(255),
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);
```

In this example, we denormalize the Products table by including the CategoryName directly in the DenormalizedProducts table to avoid joining with the Categories table during queries.

Certainly! Let's continue with more examples for Entity-Relationship Diagrams (ERD), Normalization, and Denormalization.

1. Entity-Relationship Diagrams (ERD):

Example 3: Extended ERD with Cardinality

```plaintext
+---------------------+       +----------------------+
|      Students       |       |       Courses        |
+---------------------+       +----------------------+
| PK StudentID        |       | PK CourseID          |
| Name                |       | CourseName           |
| DateOfBirth         |       +----------------------+
| Major               |       | Credits              |
+---------------------+       | InstructorID (FK)    |
| AdviserID (FK)      |       +----------------------+
+---------------------+              |
                                     |
                          +----------------------+
                          |     Instructors      |
                          +----------------------+
                          | PK InstructorID      |
                          | InstructorName       |
                          +----------------------+
```

In this example, we have three entities: Students, Courses, and Instructors. Students have an AdviserID foreign key, and Courses have an InstructorID foreign key, indicating relationships.

Example 4: ERD for a Blogging System

```plaintext
+--------------------+        +-----------------------+
|       Users        |        |       BlogPosts       |
+--------------------+        +-----------------------+
| PK UserID          |        | PK PostID             |
| Username           |        | Title                 |
| Email              |        | Content               |
| Password           |        | DatePublished         |
+--------------------+        | FK UserID (Author)    |
|                    |        +-----------------------+
|                    |             |
|                    |        +-----------------------+
|                    |        |      Comments         |
|                    |        +-----------------------+
|                    |        | PK CommentID          |
|                    |        | CommentText           |
|                    |        | DateCommented         |
|                    |        | FK UserID (Commenter) |
|                    |        | FK PostID (Post)      |
|                    |        +-----------------------+
```

In this example, we have entities for Users, BlogPosts, and Comments, depicting a simple blogging system with relationships between users, blog posts, and comments.

2. Normalization:

Example 3: Third Normal Form (3NF) - Removing Transitive Dependencies

```sql
-- Before normalization
CREATE TABLE EmployeeSkills (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(255),
    Department VARCHAR(255),
    Skill VARCHAR(255),
    SkillDescription VARCHAR(255)
);

-- After normalization (3NF)
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(255),
```

```
    Department VARCHAR(255)
);

CREATE TABLE Skills (
    SkillID INT PRIMARY KEY,
    Skill VARCHAR(255),
    SkillDescription VARCHAR(255)
);

CREATE TABLE EmployeeSkills (
    EmployeeID INT,
    SkillID INT,
    PRIMARY KEY (EmployeeID, SkillID),
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),
    FOREIGN KEY (SkillID) REFERENCES Skills(SkillID)
);
```

In this example, we have removed the transitive dependency by creating separate tables for Employees and Skills and creating a junction table EmployeeSkills.

Example 4: Boyce-Codd Normal Form (BCNF) - Removing Non-Prime Attributes

```sql
-- Before normalization
CREATE TABLE ProjectTasks (
    ProjectID INT,
    TaskID INT,
    EmployeeID INT,
    EmployeeName VARCHAR(255),
    TaskDescription VARCHAR(255),
    PRIMARY KEY (ProjectID, TaskID),
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);

-- After normalization (BCNF)
CREATE TABLE Projects (
    ProjectID INT PRIMARY KEY
);

CREATE TABLE Tasks (
    TaskID INT PRIMARY KEY,
    TaskDescription VARCHAR(255)
);
```

```
CREATE TABLE ProjectTasks (
    ProjectID INT,
    TaskID INT,
    EmployeeID INT,
    PRIMARY KEY (ProjectID, TaskID),
    FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID),
    FOREIGN KEY (TaskID) REFERENCES Tasks(TaskID),
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
```

In this example, we have achieved BCNF by separating Projects and Tasks into their own tables and maintaining the ProjectTasks relationship.

3. Denormalization:

Example 3: Materialized Views for Performance

```sql
-- Original Tables
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255),
    Phone VARCHAR(20)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

-- Materialized View for Denormalization
CREATE MATERIALIZED VIEW DenormalizedOrders AS
SELECT
    Orders.OrderID,
    Orders.OrderDate,
    Customers.CustomerID,
    Customers.Name,
    Customers.Email,
    Customers.Phone
```

```
FROM
    Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

In this example, we use a materialized view to denormalize the Orders and Customers tables for better query performance.

Example 4: Storing Aggregated Data

```sql
-- Original Tables
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY,
    ProductID INT,
    SaleDate DATE,
    Quantity INT,
    Amount DECIMAL(10, 2),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

-- Denormalized Table with Aggregated Data
CREATE TABLE DenormalizedSalesSummary (
    ProductID INT PRIMARY KEY,
    TotalQuantity INT,
    TotalAmount DECIMAL(10, 2),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

In this example, we denormalize the Sales table into a DenormalizedSalesSummary table, aggregating total quantity and total amount for each product to simplify reporting and analysis.