3. Core Relational Concepts (Tables, Keys, Normalization)

> Think of this section as your “SQL DNA class.” We’re going to cover what makes data relational, why primary and foreign keys matter, and how normalization shapes your data — all the stuff that makes your queries reliable and your pipelines sane.

3.1 What is a Table?

In SQL, a **table** is a **relation**, which is a fancy math word for a set of tuples (rows). Important things to remember:

1. **Rows are unordered** — SQL only guarantees order if you use `ORDER BY`.
2. **Columns have types** — the schema defines data type, nullability, default values, etc.
3. **Tables are sets of rows** — duplicates are possible unless restricted by constraints.

Example Table — Customers

```
CREATE TABLE dbo.Customers (
    CustomerID INT IDENTITY PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    Email NVARCHAR(100) UNIQUE,
    CreatedAt DATETIME2 DEFAULT SYSUTCDATETIME()
);
```

- `CustomerID` → Primary key (unique row identifier)
- `Email` → Unique constraint (no duplicate emails)
- `CreatedAt` → Default value
- `NVARCHAR` → Unicode strings

3.2 Primary Key (PK)

A **primary key** uniquely identifies a row in a table.

Rules & Tips:

- Cannot be `NULL`.
- Can be single column or composite (multiple columns).
- Often used in indexes to speed up joins.

Example — Single PK

```sql
CREATE TABLE dbo.Products (
    ProductID INT IDENTITY PRIMARY KEY,
    ProductName NVARCHAR(100) NOT NULL,
    Price DECIMAL(10,2) NOT NULL
);
```

Example — Composite PK

```sql
CREATE TABLE dbo.OrderItems (
    OrderID INT NOT NULL,
    ProductID INT NOT NULL,
    Quantity INT NOT NULL,
    UnitPrice DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (OrderID, ProductID)
);
```

- Composite PK = combination of columns must be unique.
- Useful in **junction tables** (many-to-many relationships).

3.3 Foreign Key (FK)

A **foreign key** enforces referential integrity. It ensures a column or set of columns **references a valid primary key in another table**.

Example — Orders referencing Customers

```sql
CREATE TABLE dbo.Orders (
    OrderID INT IDENTITY PRIMARY KEY,
    CustomerID INT NOT NULL,
    OrderDate DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME(),
    CONSTRAINT FK_Orders_Customers FOREIGN KEY (CustomerID)
        REFERENCES dbo.Customers(CustomerID)
);
```

- `CustomerID` in `Orders` **must exist** in `Customers`.
- Helps prevent “orphan” rows.

Cascading Options

- `ON DELETE CASCADE` → delete child rows if parent is deleted
- `ON UPDATE CASCADE` → update child if parent key changes

```sql
ALTER TABLE dbo.Orders
ADD CONSTRAINT FK_Orders_Customers
FOREIGN KEY (CustomerID)
REFERENCES dbo.Customers(CustomerID)
ON DELETE CASCADE;
```

3.4 Data Relationships (ER perspective)

- **One-to-Many (1:N)** — One customer → many orders.
- **Many-to-Many (M:N)** — Many products → many orders → junction table (`OrderItems`).
- **One-to-One (1:1)** — Rare; e.g., User → UserProfile.

Visual example:

```
Customers (1) ─< Orders (N) ─< OrderItems (N) >─ Products (1)
```

3.5 Normalization — Why we care

Normalization is the **art of reducing redundancy and anomalies**.

1NF (First Normal Form)

- Each column is atomic (no lists or arrays).
- Each row is unique.

```sql
-- ❌ Bad (violates 1NF)
CREATE TABLE dbo.BadCustomerOrders (
    CustomerID INT,
    Orders NVARCHAR(MAX) -- list of order IDs
);
```

```sql
-- ☑ Good
CREATE TABLE dbo.Orders (
    OrderID INT IDENTITY PRIMARY KEY,
    CustomerID INT NOT NULL
);
```

2NF (Second Normal Form)

- Table is in 1NF.

- Every non-PK column depends **on the whole PK** (applies to composite keys).

```
-- OrderItems table example
PRIMARY KEY (OrderID, ProductID)
-- Quantity depends on both OrderID and ProductID, not just one.
```

3NF (Third Normal Form)

- Table is in 2NF.
- No transitive dependencies: non-PK columns do not depend on other non-PK columns.

```
-- ❌ Bad
CREATE TABLE dbo.Employees (
    EmployeeID INT PRIMARY KEY,
    DepartmentID INT,
    DepartmentName NVARCHAR(50) -- violates 3NF
);
```

```
-- ☑ Good
CREATE TABLE dbo.Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName NVARCHAR(50) NOT NULL
);

CREATE TABLE dbo.Employees (
    EmployeeID INT PRIMARY KEY,
    DepartmentID INT FOREIGN KEY REFERENCES dbo.Departments(DepartmentID)
);
```

3.6 Practical Tips for Data Engineering

1. **Always define PKs** — makes joins predictable.
2. **Use FKs if you can** — catches data issues early.
3. **Normalize for OLTP, denormalize for analytics** — star schema, fact/dimension tables.
4. **Naming conventions matter** — e.g., `tbl_Orders` vs `Orders`, `FK_Orders_Customers`.
5. **Indexes go hand-in-hand with PK/FK** — speed joins and filters.

3.7 Hands-On Exercise

1. Create `Customers`, `Products`, `Orders`, and `OrderItems` tables.
2. Insert sample data:

```sql
INSERT INTO dbo.Customers (FirstName, LastName, Email) VALUES
('Alice', 'Smith', 'alice@example.com'),
('Bob', 'Jones', 'bob@example.com');

INSERT INTO dbo.Products (ProductName, Price) VALUES
('Widget', 19.99),
('Gadget', 29.99);

INSERT INTO dbo.Orders (CustomerID) VALUES (1), (2);

INSERT INTO dbo.OrderItems (OrderID, ProductID, Quantity, UnitPrice) VALUES
(1, 1, 2, 19.99),
(1, 2, 1, 29.99),
(2, 2, 3, 29.99);
```

3. Query total spending per customer:

```sql
SELECT c.FirstName, c.LastName, SUM(oi.Quantity * oi.UnitPrice) AS TotalSpent
FROM dbo.Customers c
JOIN dbo.Orders o ON o.CustomerID = c.CustomerID
JOIN dbo.OrderItems oi ON oi.OrderID = o.OrderID
GROUP BY c.FirstName, c.LastName;
```

☑ Expected result: Alice and Bob’s total spendings calculated correctly.