# Database Design (Normalised)

Below is a **logical model** based on the requirements, designed to be in **3rd Normal Form (3NF)**.

## Core Tables

### 1. User (CustomUser)

Extends AbstractUser.

#### Fields

- id (PK)
- username
- email
- password
- role (Reader / Editor / Journalist)

Groups & permissions handled via Django's Group and Permission tables.

### 2. Publisher

#### Fields

- id (PK)
- name
- description

#### Relationships

- One publisher → many editors
- One publisher → many journalists
- One publisher → many articles

## 3. Article

### Fields

- id (PK)
- title
- content
- created_at
- is_approved (Boolean)
- publisher_id (FK, nullable)
- journalist_id (FK, nullable)

### Rules

- An article belongs to **either**:
    - a publisher **or**
    - an independent journalist
- Never both at the same time (business logic validation).

## 4. Newsletter

### Fields

- id (PK)
- title
- content
- created_at
- publisher_id (FK, nullable)
- journalist_id (FK, nullable)

# Relationship Tables (Normalisation)

## 5. PublisherEditors

(Many-to-many)

### Fields

- id (PK)
- publisher_id (FK)
- user_id (FK → Editor)

## 6. PublisherJournalists

(Many-to-many)

### Fields

- id (PK)
- publisher_id (FK)
- user_id (FK → Journalist)

# Subscription Tables

## 7. PublisherSubscription

### Fields

- id (PK)
- reader_id (FK → User)
- publisher_id (FK)

### 8. JournalistSubscription

#### Fields

- id (PK)
- reader_id (FK → User)
- journalist_id (FK)

## Why This Is Normalized

- No repeating groups
- No redundant data
- Clear separation of concerns
- Flexible subscriptions (publisher OR journalist)
- Scales well for API usage

## Key Business Rules (Important for Marks)

- Reader users:
    - Have subscriptions
    - Do **not** own articles or newsletters
- Journalist users:
    - Own articles/newsletters
    - Have **no subscription data**
- Editor users:
    - Linked to publishers
    - Cannot create articles

These are enforced via:

- model validation
- permissions
- role-based logic