**SQL Schema**

sql

-- Create Table for Magazine Information

CREATE TABLE magazine\_info (

id SERIAL PRIMARY KEY,

title VARCHAR(255) NOT NULL,

author VARCHAR(255),

publication\_date DATE,

category VARCHAR(100),

-- Add other fields as needed

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Create Table for Magazine Content

CREATE TABLE magazine\_content (

id SERIAL PRIMARY KEY,

magazine\_id INTEGER REFERENCES magazine\_info(id) ON DELETE CASCADE,

content TEXT,

vector\_representation VECTOR(300), -- Assumes a vector length of 300; adjust as needed

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Indexes for performance optimization

-- Index for quick keyword-based searches

CREATE INDEX idx\_magazine\_title ON magazine\_info(title);

CREATE INDEX idx\_magazine\_author ON magazine\_info(author);

CREATE INDEX idx\_magazine\_category ON magazine\_info(category);

-- Index for faster vector search (requires vector indexing support from the database)

-- For PostgreSQL with pgvector, you would need to create an index like this:

CREATE INDEX idx\_vector\_representation ON magazine\_content USING ivfflat(vector\_representation);

**ORM Models**

If you are using an ORM like Sequelize (for Node.js) or SQLAlchemy (for Python), here are the corresponding models:

**Sequelize (Node.js)**

javascript

const { DataTypes } = require('sequelize');

const sequelize = require('./db'); // Assuming you have a db.js for Sequelize initialization

// Magazine Info Model

const MagazineInfo = sequelize.define('MagazineInfo', {

title: {

type: DataTypes.STRING,

allowNull: false,

},

author: {

type: DataTypes.STRING,

},

publication\_date: {

type: DataTypes.DATE,

},

category: {

type: DataTypes.STRING,

},

created\_at: {

type: DataTypes.DATE,

defaultValue: DataTypes.NOW,

},

updated\_at: {

type: DataTypes.DATE,

defaultValue: DataTypes.NOW,

},

}, {

tableName: 'magazine\_info',

timestamps: false,

});

// Magazine Content Model

const MagazineContent = sequelize.define('MagazineContent', {

magazine\_id: {

type: DataTypes.INTEGER,

references: {

model: MagazineInfo,

key: 'id',

},

onDelete: 'CASCADE',

},

content: {

type: DataTypes.TEXT,

},

vector\_representation: {

type: DataTypes.ARRAY(DataTypes.FLOAT), // Use this if using PostgreSQL and pgvector

},

created\_at: {

type: DataTypes.DATE,

defaultValue: DataTypes.NOW,

},

updated\_at: {

type: DataTypes.DATE,

defaultValue: DataTypes.NOW,

},

}, {

tableName: 'magazine\_content',

timestamps: false,

});

// Establish relationships

MagazineInfo.hasMany(MagazineContent, { foreignKey: 'magazine\_id' });

MagazineContent.belongsTo(MagazineInfo, { foreignKey: 'magazine\_id' });

module.exports = { MagazineInfo, MagazineContent };

**SQLAlchemy (Python)**

python

from sqlalchemy import create\_engine, Column, Integer, String, Text, Date, Float, ForeignKey, TIMESTAMP

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import relationship, sessionmaker

from sqlalchemy.sql import func

Base = declarative\_base()

# Magazine Info Model

class MagazineInfo(Base):

\_\_tablename\_\_ = 'magazine\_info'

id = Column(Integer, primary\_key=True, autoincrement=True)

title = Column(String(255), nullable=False)

author = Column(String(255))

publication\_date = Column(Date)

category = Column(String(100))

created\_at = Column(TIMESTAMP, default=func.now())

updated\_at = Column(TIMESTAMP, default=func.now(), onupdate=func.now())

# Relationship to MagazineContent

contents = relationship('MagazineContent', back\_populates='magazine')

# Magazine Content Model

class MagazineContent(Base):

\_\_tablename\_\_ = 'magazine\_content'

id = Column(Integer, primary\_key=True, autoincrement=True)

magazine\_id = Column(Integer, ForeignKey('magazine\_info.id', ondelete='CASCADE'))

content = Column(Text)

vector\_representation = Column(Float) # Adjust as needed, or use an appropriate type for vector data

created\_at = Column(TIMESTAMP, default=func.now())

updated\_at = Column(TIMESTAMP, default=func.now(), onupdate=func.now())

# Relationship to MagazineInfo

magazine = relationship('MagazineInfo', back\_populates='contents')

# Database connection setup

DATABASE\_URL = "postgresql://user:password@localhost/mydatabase"

engine = create\_engine(DATABASE\_URL)

Base.metadata.create\_all(engine)

Session = sessionmaker(bind=engine)

session = Session()

**Notes**

1. **Vector Representation:**
   * For PostgreSQL, using the pgvector extension requires additional configuration for handling vector data.
   * Ensure the VECTOR type or equivalent is supported by your chosen database, or use a suitable workaround for vector storage.
2. **Indexes:**
   * The vector indexing approach may vary depending on the database system and its support for vector operations. The SQL provided uses ivfflat indexing for PostgreSQL as an example.
3. **ORM Models:**
   * The provided ORM models assume usage of SQLAlchemy for Python and Sequelize for Node.js. Adjust according to your specific setup and ORM capabilities.