Database Compatibility Notes - SQLite3 & MySQL

Schema Compatibility

✅ Compatible Features Used:

1. **Data Types:**
   - `VARCHAR(n)` - Both support (SQLite ignores size, MySQL enforces)
   - `INTEGER` - Compatible
   - `FLOAT` - Compatible
   - `TEXT` - Compatible
   - `DATETIME` - Compatible (use this instead of TIMESTAMP)
2. **Constraints:**
   - `PRIMARY KEY` - Compatible
   - `FOREIGN KEY` - Compatible (enable in SQLite: `PRAGMA foreign_keys = ON;`)
   - `UNIQUE` - Compatible
   - `CHECK` - Compatible (SQLite 3.3.0+, MySQL 8.0+)
   - `NOT NULL` - Compatible
   - `DEFAULT` - Compatible
3. **Functions:**
   - `CURRENT_TIMESTAMP` - Compatible for both

⚠️ Differences to Handle:

1. Auto Increment:

```sql
-- SQLite3:
id INTEGER PRIMARY KEY AUTOINCREMENT

-- MySQL:
id INTEGER PRIMARY KEY AUTO_INCREMENT
```

Solution: Schema uses AUTOINCREMENT (SQLite). For MySQL, find/replace with AUTO_INCREMENT.

2. Updated At Triggers:

SQLite doesn't support `ON UPDATE CURRENT_TIMESTAMP`, but MySQL does.

For SQLite, create trigger:

```sql
CREATE TRIGGER update_brands_timestamp
AFTER UPDATE ON brands
BEGIN
  UPDATE brands SET updated_at = CURRENT_TIMESTAMP WHERE id = NEW.id;
END;
```

For MySQL:

```sql
ALTER TABLE brands MODIFY updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT
```

3. Case Sensitivity:

- SQLite: Case-insensitive by default for ASCII
- MySQL: Depends on collation (usually case-insensitive)
- Use `COLLATE NOCASE` in SQLite if needed

4. Foreign Keys:

```sql
-- SQLite requires enabling foreign keys per connection:
PRAGMA foreign_keys = ON;

-- MySQL has them enabled by default
```

5. Boolean Values:

- SQLite: Use INTEGER (0/1)
- MySQL: Has BOOLEAN type (stored as TINYINT)
- Schema avoids this - not needed

📋 Migration Path: SQLite → MySQL

📦 Migration Path: SQLite → MySQL

1. **Export data from SQLite:**

```
bash

sqlite3 diet.db .dump > diet.sql
```

2. **Modify schema:**

```
bash

sed 's/AUTOINCREMENT/AUTO_INCREMENT/g' diet.sql > diet_mysql.sql
```

3. **Add ON UPDATE triggers to MySQL version:**

```
sql

ALTER TABLE brands MODIFY updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_
ALTER TABLE brand_data MODIFY updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRE
-- etc. for all tables
```

4. **Import to MySQL:**

```
bash

mysql -u user -p database < diet_mysql.sql
```

🔧 Application Code Considerations:

Connection Setup:

SQLite3:

```
javascript

import sqlite3 from 'sqlite3';
const db = new sqlite3.Database('./db/diet.db');
db.run('PRAGMA foreign_keys = ON');
```

MySQL:

```
javascript
```

```
import mysql from 'mysql2/promise';
const pool = mysql.createPool({
    host: 'localhost',
    user: 'diet_user',
    password: 'password',
    database: 'diet_db',
    waitForConnections: true,
    connectionLimit: 10
});
```

Query Differences:

Parameter Placeholders:

- SQLite: Uses `?` or `$name`
- MySQL: Uses `?`
- **Use `?` for compatibility**

LIMIT/OFFSET: Both support the same syntax - compatible ✅

Date Functions:

- SQLite: `datetime('now')`
- MySQL: `NOW()`
- Use `CURRENT_TIMESTAMP` in queries for compatibility ✅

📝 Testing Checklist:

- ☐ Test foreign key constraints work (SQLite needs PRAGMA)
- ☐ Test CHECK constraints work (requires modern versions)
- ☐ Test UNIQUE constraints on composite keys
- ☐ Test CASCADE deletes work properly
- ☐ Test DEFAULT CURRENT_TIMESTAMP works
- ☐ Test VARCHAR size limits (only MySQL enforces)
- ☐ Test date/time insertion and retrieval
- ☐ Test NULL vs NOT NULL constraints

🎯 Recommendations:

1. **Start with SQLite3** for development (simpler, file-based)
2. **Test on MySQL** periodically to catch incompatibilities early

3. **Use ORM** (like Knex.js or Sequelize) to abstract differences
4. **Document migration scripts** for moving SQLite → MySQL
5. **Keep schema.sql** in version control with comments about differences