SQLModel Relationships and ORM Guide

This document explains how to connect tables in SQLModel using ORM relationships. It covers 1-1, 1-M, and M-M relationships in depth.

1■■ ONE-TO-MANY (1-M)

Example:
One user can write many blog posts.

User ■■■< BlogPost

Code:

```
class User(SQLModel, table=True):
id: UUID = Field(default_factory=uuid4, primary_key=True)
username: str = Field(index=True)
blog_posts: List["BlogPost"] = Relationship(back_populates="author")

class BlogPost(SQLModel, table=True):
id: UUID = Field(default_factory=uuid4, primary_key=True)
title: str
content: str
user_id: UUID = Field(foreign_key="user.id")
author: "User" = Relationship(back_populates="blog_posts")
```

Explanation:
- Foreign key in BlogPost (`user_id`) connects to User (`id`).
- Relationship() is defined in both classes using back_populates.
- The relationship key does not create a DB column; it maps Python-level class links.

2■■ ONE-TO-ONE (1-1)

Example:
One user has one profile.

User ■■■ Profile

Code:

```
class User(SQLModel, table=True):
id: UUID = Field(default_factory=uuid4, primary_key=True)
username: str
profile: Optional["Profile"] = Relationship(back_populates="user")

class Profile(SQLModel, table=True):
id: UUID = Field(default_factory=uuid4, primary_key=True)
bio: str
user_id: UUID = Field(foreign_key="user.id", unique=True)
user: "User" = Relationship(back_populates="profile")
```

Explanation:

- Profile table has a unique foreign key, ensuring only one profile per user.
- Both sides connect using Relationship() + back_populates().

3■■ MANY-TO-MANY (M-M)

Example:
A blog post can have many tags, and a tag can belong to many posts.

BlogPost >■■< Tag

Code:

```
class PostTagLink(SQLModel, table=True):
post_id: UUID = Field(foreign_key="blogpost.id", primary_key=True)
tag_id: UUID = Field(foreign_key="tag.id", primary_key=True)

class BlogPost(SQLModel, table=True):
id: UUID = Field(default_factory=uuid4, primary_key=True)
title: str
tags: List["Tag"] = Relationship(back_populates="posts", link_model=PostTagLink)

class Tag(SQLModel, table=True):
id: UUID = Field(default_factory=uuid4, primary_key=True)
name: str
posts: List["BlogPost"] = Relationship(back_populates="tags", link_model=PostTagLink)
```

Explanation:
- M-M uses a linking (association) table — PostTagLink.
- Relationship uses `link_model` to define the middle table.
- SQLModel automatically handles the join table logic.

4■■ ORM RELATIONSHIP SUMMARY

- PK–FK = Real database connection.
- Relationship() = Python-side mapping (no DB column).
- back_populates = Defines two-way synchronization.
- Once mapped, you can access related data directly:
user.blog_posts → all posts by a user
post.author → user object of a post

5■■ ORM MINDSET

Think of ORM relationships like this:
- Foreign key = the bridge (database level)
- Relationship = the doorway (Python level)
- back_populates = two-way key keeping both in sync

After defining models:
1. Configure Alembic
2. Generate migration script
3. Apply migration to create tables
4. Use ORM for CRUD without writing raw SQL