Mindmap to Learn SQLModel

1. **Introduction to SQLModel**
   * **Overview**: SQLModel is a Python library for interacting with SQL databases using Python type hints and Pydantic models. It is built on top of SQLAlchemy and Pydantic, providing a simple and intuitive way to work with databases.
   * **Installation**: Install SQLModel using pip: pip install sqlmodel.
2. **Getting Started**
   * **Creating a Model**: Define a model by inheriting from SQLModel.
     + Example: from sqlmodel import SQLModel, Field; class Hero(SQLModel, table=True): id: int = Field(default=None, primary\_key=True); name: str.
   * **Connecting to a Database**: Use SQLAlchemy's create\_engine to connect to a database.
     + Example: from sqlmodel import create\_engine; engine = create\_engine("sqlite:///database.db").
3. **CRUD Operations**
   * **Create**: Insert data into the database.
     + Example: hero = Hero(name="Spider-Man"); session.add(hero); session.commit().
   * **Read**: Query data from the database.
     + Example: heroes = session.query(Hero).all().
   * **Update**: Modify existing data.
     + Example: hero = session.query(Hero).filter(Hero.name == "Spider-Man").first(); hero.name = "Spidey"; session.commit().
   * **Delete**: Remove data from the database.
     + Example: session.delete(hero); session.commit().
4. **Session Management**
   * **Creating a Session**: Use Session from SQLModel to interact with the database.
     + Example: from sqlmodel import Session; session = Session(engine).
   * **Context Manager**: Use sessions with context managers to ensure proper resource management.
     + Example: with Session(engine) as session: session.add(hero); session.commit().
5. **Relationships**
   * **One-to-Many**: Define one-to-many relationships between models.
     + Example: class Team(SQLModel, table=True): id: int = Field(default=None, primary\_key=True); name: str; heroes: List[Hero] = Relationship(back\_populates="team").
   * **Many-to-One**: Define many-to-one relationships between models.
     + Example: class Hero(SQLModel, table=True): id: int = Field(default=None, primary\_key=True); name: str; team\_id: int = Field(default=None, foreign\_key="team.id"); team: Team = Relationship(back\_populates="heroes").
   * **Many-to-Many**: Define many-to-many relationships using an association table.
     + Example: class HeroTeamLink(SQLModel, table=True): hero\_id: int = Field(foreign\_key="hero.id"); team\_id: int = Field(foreign\_key="team.id").
6. **Field Types and Constraints**
   * **Field Types**: Define the type and constraints of each field.
     + Example: id: int = Field(default=None, primary\_key=True); name: str = Field(index=True).
   * **Validators**: Use Pydantic validators to enforce custom constraints.
     + Example: @validator('name'); def name\_must\_not\_be\_empty(cls, v): if not v: raise ValueError('Name must not be empty'); return v.
7. **Database Configuration**
   * **Database URL**: Specify the database URL for different database backends (SQLite, PostgreSQL, MySQL).
     + Example: DATABASE\_URL = "sqlite:///database.db".
   * **Engine Configuration**: Configure the SQLAlchemy engine with parameters like echo and pool size.
     + Example: engine = create\_engine(DATABASE\_URL, echo=True).
8. **Migrations**
   * **Alembic Integration**: Use Alembic for database migrations.
     + Example: alembic init alembic.
   * **Generating Migrations**: Create migration scripts to reflect model changes.
     + Example: alembic revision --autogenerate -m "Initial migration".
   * **Applying Migrations**: Apply the generated migrations to the database.
     + Example: alembic upgrade head.
9. **Advanced Queries**
   * **Filtering**: Use filter to narrow down query results.
     + Example: session.query(Hero).filter(Hero.name == "Spidey").
   * **Joins**: Perform joins between related tables.
     + Example: session.query(Hero, Team).join(Team).filter(Team.name == "Avengers").
   * **Aggregations**: Perform aggregate functions like count, sum, avg.
     + Example: session.query(func.count(Hero.id)).scalar().
10. **Async Support**
    * **Async SQLModel**: Use SQLModel with asynchronous database drivers like asyncpg.
      + Example: from sqlmodel.ext.asyncio.session import AsyncSession; async with AsyncSession(engine) as session: await session.execute(statement).
    * **Async Queries**: Perform async queries and operations.
      + Example: await session.execute(select(Hero)).
11. **Testing**
    * **Unit Tests**: Write unit tests for your SQLModel models and queries.
      + Example: def test\_create\_hero(): hero = Hero(name="Batman"); session.add(hero); session.commit(); assert hero.id is not None.
    * **Test Databases**: Use SQLite in-memory databases for fast testing.
      + Example: engine = create\_engine("sqlite:///:memory:").
12. **Performance Optimization**
    * **Indexing**: Add indexes to frequently queried fields.
      + Example: name: str = Field(index=True).
    * **Query Optimization**: Profile and optimize complex queries.
      + Use tools like EXPLAIN in SQL to understand query performance.
13. **Deployment**
    * **Containerization**: Use Docker to containerize your SQLModel application.
      + Example: Create a Dockerfile and build the image.
    * **Cloud Databases**: Connect to cloud-managed databases like AWS RDS or Google Cloud SQL.
      + Example: DATABASE\_URL = "postgresql://user:password@host/db".
14. **Best Practices**
    * **Code Organization**: Organize models, CRUD operations, and session management into separate modules.
      + Example: models.py, crud.py, database.py.
    * **Type Hints**: Use type hints extensively for better code readability and maintainability.
      + Example: def get\_hero\_by\_id(session: Session, hero\_id: int) -> Optional[Hero]:.
    * **Documentation**: Document your models and functions for better collaboration and maintenance.
      + Example: Use docstrings and comments.