COMP 3005 A - Fall 2025
Database Management Systems
Final Project Report
Raymond Liu 101264487
Afaq Virk 101338854

Object-Relational Mapping (ORM) Usage

We used SQLAlchemy's Declarative ORM to map Python classes to relational tables and to express relationships and constraints directly in code. A single shared base class is defined with `declarative_base()`, and each entity declares its table name, columns, constraints, and relationships. This allows us to perform type-safe CRUD operations, joins, and cascades using Pythonic APIs instead of writing raw SQL for common operations. Relationships (e.g., one-to-many between `User` and `Metric`) are modeled with `relationship(...)` along with `back_populates` for bidirectional navigation, and we use options such as `cascade="all, delete-orphan"` to enforce lifecycle rules.

The application layer interacts with the database through a SQLAlchemy `Session`. Typical patterns include querying with `session.query(...).filter_by(...).order_by(...)`; creating domain objects, calling `session.add(...)` and `session.commit()`; performing relational joins for cross-entity views; and deleting records with transactional rollback on error. This approach keeps business logic in Python while the ORM translates it into efficient SQL.

Listing 1: Example SQLAlchemy model mapping

```python
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class User(Base):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True)
    email = Column(String(255), nullable=False, unique=True)
    first_name = Column(String(100), nullable=False)
    last_name = Column(String(100), nullable=False)
    role = Column(Integer, ForeignKey('role.id'), nullable=False)

    role_obj = relationship("Role", back_populates="users")
    metrics = relationship("Metric", back_populates="user",
                           cascade="all,␣delete-orphan")
```

Listing 2: Typical ORM query/CRUD usage

```python
# Recent metrics for a member
recent = (session.query(Metric)
          .filter_by(user_id=user.id)
```

```
         .order_by(Metric.logged_date.desc())
         .limit(5)
         .all())

# Create and commit a new metric
new_metric = Metric(user_id=user.id, metric_type=metric_type_id, value=Decimal("
   72.5"))
session.add(new_metric)
session.commit()

# Join across relationships to see upcoming sessions
upcoming = (session.query(Enrollment)
           .join(Session)
           .join(Schedule)
           .filter(Enrollment.member_id == user.id, Schedule.date >= date.today()
              )
           .order_by(Schedule.date, Schedule.start_time)
           .all())
```

Major mapped entities

- Role, User
- Service, Bill, Item
- MetricType, Metric, Goal
- Room, EquipmentStatus, Equipment
- ScheduleType, Schedule, Session, Enrollment