Lesson 10: ORM & Non-SQL Overview

Duration: 20 minutes
Deliverable: `lesson10_comparison.md`

🎯 Learning Objectives

By the end of this lesson, you will be able to:

- Understand what Object-Relational Mapping (ORM) is
- Compare SQL to ORM approaches
- Recognise when to use each approach
- Understand NoSQL database types
- Know the differences between SQL and NoSQL
- Make informed decisions about database choices

📚 What is ORM?

Object-Relational Mapping (ORM) is a technique that lets you interact with your database using objects in your programming language, rather than writing SQL.

Without ORM (Raw SQL):

```
cursor.execute("SELECT * FROM characters WHERE species = ?", ("Human",))
characters = cursor.fetchall()
```

With ORM (SQLAlchemy):

```
characters = session.query(Character).filter_by(species="Human").all()
```

🛠️ Part 1: ORM Introduction (10 minutes)

What Problems Does ORM Solve?

1. **Reduces boilerplate code:** Less repetitive database connection code
2. **Type safety:** Objects have defined properties
3. **Abstraction:** Don't need to know SQL to use databases
4. **Portability:** Easier to switch databases (SQLite → PostgreSQL → MySQL)
5. **Object-oriented:** Work with Python objects, not query results

Popular Python ORMs

ORM	Description	Best For
SQLAlchemy	Most powerful and flexible	Complex applications, enterprise
Django ORM	Built into Django framework	Django web apps
Peewee	Lightweight and simple	Small projects, prototypes
Tortoise ORM	Async support	Async Python applications

Step 1: Create Your Comparison File

1. Create: `lesson10_comparison.md`
2. We'll fill it in as we learn!

Quick SQLAlchemy Example

Note: SQLAlchemy is pre-installed in your Codespace!

```python
# lesson10_orm_demo.py
from sqlalchemy import create_engine, Column, Integer, String, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, relationship

# Create base class for models
Base = declarative_base()

# Define Character model (like a table)
class Character(Base):
    __tablename__ = 'characters'

    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    species = Column(String)
    height = Column(Integer)
    homeworld_id = Column(Integer, ForeignKey('planets.id'))

    # Relationship to Planet
    homeworld = relationship("Planet", back_populates="characters")

    def __repr__(self):
        return f"<Character(name='{self.name}', species='{self.species}')>"

# Define Planet model
class Planet(Base):
    __tablename__ = 'planets'

    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    climate = Column(String)
    population = Column(Integer)

    # Relationship to Character
    characters = relationship("Character", back_populates="homeworld")

    def __repr__(self):
        return f"<Planet(name='{self.name}')>"

# Connect to database
engine = create_engine('sqlite:///database/starwars.db')
Session = sessionmaker(bind=engine)
session = Session()

# Query examples
print("=== ORM Query Examples ===\n")

# 1. Get all characters
print("1. All characters:")
characters = session.query(Character).all()
for char in characters[:3]:  # Show first 3
    print(f"  - {char.name} ({char.species})")

# 2. Filter by species
print("\n2. Human characters:")
```

```
humans = session.query(Character).filter_by(species='Human').all()
for human in humans[:3]:
    print(f"  - {human.name}")

# 3. Order and limit
print("\n3. Three tallest characters:")
tallest = session.query(Character)\
    .filter(Character.height.isnot(None))\
    .order_by(Character.height.desc())\
    .limit(3)\
    .all()
for char in tallest:
    print(f"  - {char.name}: {char.height}cm")

# 4. Join (automatic with relationships!)
print("\n4. Characters with their homeworlds:")
chars_with_planets = session.query(Character)\
    .join(Planet)\
    .filter(Planet.name == 'Tatooine')\
    .all()
for char in chars_with_planets:
    print(f"  - {char.name} from {char.homeworld.name}")

# 5. Aggregate functions
from sqlalchemy import func
print("\n5. Statistics:")
avg_height = session.query(func.avg(Character.height)).scalar()
char_count = session.query(func.count(Character.id)).scalar()
print(f"  - Average height: {avg_height:.1f}cm")
print(f"  - Total characters: {char_count}")

session.close()
```

Save this as `lesson10_orm_demo.py` and run it:

```
python lesson10_orm_demo.py
```

📊 Part 2: SQL vs ORM Comparison (8 minutes)

Side-by-Side Comparison

Example 1: Simple SELECT

SQL:

```
SELECT * FROM characters WHERE species = 'Human';
```

Python + SQL:

```
cursor.execute("SELECT * FROM characters WHERE species = ?", ("Human",))
results = cursor.fetchall()
```

ORM (SQLAlchemy):

```
characters = session.query(Character).filter_by(species="Human").all()
```

Example 2: JOIN Query

SQL:

```
SELECT c.name, p.name
FROM characters c
JOIN planets p ON c.homeworld_id = p.id
WHERE p.climate = 'arid';
```

Python + SQL:

```
cursor.execute("""
    SELECT c.name, p.name
    FROM characters c
    JOIN planets p ON c.homeworld_id = p.id
    WHERE p.climate = ?
""", ("arid",))
results = cursor.fetchall()
```

ORM (SQLAlchemy):

```
characters = session.query(Character)\
    .join(Planet)\
    .filter(Planet.climate == "arid")\
    .all()
# Access planet through relationship:
for char in characters:
    print(f"{char.name} from {char.homeworld.name}")
```

Example 3: INSERT

SQL:

```
INSERT INTO characters (name, species, height)
VALUES ('New Character', 'Human', 175);
```

Python + SQL:

```
cursor.execute("""
    INSERT INTO characters (name, species, height)
    VALUES (?, ?, ?)
""", ("New Character", "Human", 175))
conn.commit()
```

ORM (SQLAlchemy):

```
new_char = Character(name="New Character", species="Human", height=175)
session.add(new_char)
session.commit()
```

Advantages and Disadvantages

ORM Advantages ✅

1. **Less code:** More concise for common operations
2. **Object-oriented:** Work with Python objects
3. **Database agnostic:** Switch databases easily
4. **IDE support:** Autocomplete for model properties
5. **Relationships:** Easy to navigate between related tables
6. **Migrations:** Built-in schema versioning (in most ORMs)

ORM Disadvantages ❌

1. **Performance:** Can generate inefficient SQL
2. **Learning curve:** Must learn ORM syntax
3. **Complex queries:** Hard to express some SQL operations
4. **Debugging:** Generated SQL can be hard to inspect
5. **Overhead:** Extra abstraction layer

6. **Less control:** Can't optimise queries as precisely

SQL Advantages ✅

1. **Performance:** Write optimised queries
2. **Universal:** Works everywhere
3. **Powerful:** Express any query
4. **Transparency:** See exactly what's executed
5. **Simple:** Straightforward for basic operations

SQL Disadvantages ❌

1. **More code:** Repetitive connection/cursor handling
2. **String-based:** No autocomplete or type checking
3. **Database-specific:** Syntax varies between databases
4. **Manual work:** Must handle all relationships yourself

🌐 Part 3: NoSQL Databases Overview (8 minutes)

NoSQL (Not Only SQL) databases store data in formats other than tables.

Types of NoSQL Databases

1. Document Databases (e.g., MongoDB)

Store data as JSON-like documents.

Example - Star Wars Character in MongoDB:

```
{
    "_id": "507f1f77bcf86cd799439011",
    "name": "Luke Skywalker",
    "species": "Human",
    "height": 172,
    "homeworld": {
        "name": "Tatooine",
        "climate": "arid",
        "population": 200000
    },
    "vehicles": [
        {
            "name": "X-wing",
            "class": "Starfighter"
        },
        {
            "name": "Snowspeeder",
            "class": "Airspeeder"
        }
    ]
}
```

Advantages:

- Flexible schema (can have different fields)
- Nested data (no JOINs needed!)
- Fast for read-heavy workloads
- Scales horizontally

Disadvantages:

- Data duplication
- Complex queries less efficient
- Less mature transaction support

2. Key-Value Stores (e.g., Redis)

Store data as simple key-value pairs.

Example:

```
character:1 → "Luke Skywalker|Human|Tatooine|172"
character:2 → "Leia Organa|Human|Alderaan|150"
species:count → "5"
```

Use Cases:

- Caching
- Session storage
- Real-time analytics
- Configuration data

3. Column-Family Stores (e.g., Cassandra)

Store data in columns rather than rows.

Use Cases:

- Time-series data
- IoT sensor data
- Large-scale analytics

4. Graph Databases (e.g., Neo4j)

Store data as nodes and relationships.

Example - Star Wars as Graph:

```
(Luke)-[:FLIES]->(X-wing)
(Luke)-[:FROM]->(Tatooine)
(Luke)-[:SIBLING]->(Leia)
(Leia)-[:FROM]->(Alderaan)
```

Use Cases:

- Social networks
- Recommendation engines
- Network analysis
- Relationship-heavy data

🔄 Part 4: When to Use What?

Use SQL (Relational) When:

✅ Data has clear structure and relationships
✅ Need complex queries with JOINs
✅ Require ACID transactions (banking, inventory)
✅ Data integrity is critical
✅ Reporting and analytics are important

Examples: Banking systems, ERP, e-commerce, HR systems

Use NoSQL When:

✅ Schema changes frequently
✅ Need horizontal scaling
✅ Handling huge volumes of data
✅ Real-time data processing
✅ Unstructured or semi-structured data

Examples: Social media, IoT, content management, caching, real-time analytics

Hybrid Approach

Many modern applications use **both**:

- SQL for critical transactional data
- NoSQL for caching, sessions, logs
- Example: E-commerce site uses PostgreSQL for orders, Redis for cart, MongoDB for product catalogue

📝 Part 5: Complete Your Comparison Document

Now, fill in your `lesson10_comparison.md` file:

````
# SQL, ORM, and NoSQL Comparison

**Student Name:** [Your Name]
**Date:** [Today's Date]

---

<div class="page-break"></div>

## 1. SQL (Structured Query Language)

### What is it?

[Your explanation here]

### Advantages

- [List advantages]

### Disadvantages

- [List disadvantages]

### When to use

- [List use cases]

### Example Query

```sql
[Your example]
```
````

2. ORM (Object-Relational Mapping)

What is it?

[Your explanation]

Advantages

- [List advantages]

Disadvantages

- [List disadvantages]

When to use

- [List use cases]

Example Code (SQLAlchemy)

```
[Your example]
```

3. NoSQL (Document Database)

What is it?

[Your explanation]

Example Document Structure

```
{
    [Your example]
}
```

Advantages

- [List advantages]

Disadvantages

- [List disadvantages]

When to use

- [List use cases]

4. My Recommendation

For the Star Wars database project, I would choose: **[SQL/ORM/NoSQL]** because:

[Your reasoning here - 2-3 sentences]

5. Real-World Example

Describe a real application and which database approach would be best:

Application: [e.g., Social media app, online shop, etc.]

Best Choice: [SQL/NoSQL/Hybrid]

Reasoning: [Your explanation]

Reflection Questions

1. **What surprised you most about ORMs?** [Your answer]
2. **Can you think of a situation where NoSQL would be better than SQL?** [Your answer]
3. **For a school project, which would you choose and why?** [Your answer]

✅ Checkpoint: What You've Learnt

Before finishing, make sure you can:

- ✅ Explain what ORM is
- ✅ Compare SQL and ORM approaches
- ✅ List advantages and disadvantages of each
- ✅ Describe different types of NoSQL databases
- ✅ Know when to use SQL vs NoSQL
- ✅ Make informed database technology choices

🎯 Discussion Questions (Optional)

1. **When would you choose SQL over NoSQL?**

   Consider: data structure, relationships, query complexity, scalability

2. **What are the trade-offs of using ORM?**

   Think about: ease of use vs performance, abstraction vs control

3. **How would you structure our Star Wars database in MongoDB?**

   Consider: nested documents, data duplication, query patterns

---

💾 Save Your Work with Git

```bash
git status
git add lessons/lesson10_comparison.md
git add lessons/lesson10_orm_demo.py  # If you created this
git commit -m "Completed Lesson 10: ORM and NoSQL database comparison"
git push
```

📖 Key Concepts Summary

Technology	Data Model	Query Method	Best For
SQL	Tables with rows/ columns	SQL language	Structured data, complex queries
ORM	Objects (abstracts SQL)	Object methods	Application development, productivity
NoSQL (Document)	JSON-like documents	Database-specific	Flexible schema, scalability
NoSQL (Key-Value)	Key-value pairs	Simple lookups	Caching, sessions
NoSQL (Graph)	Nodes and edges	Graph traversal	Relationships, social networks

🌟 Technology Trends

Current Industry Usage:

Most Popular SQL Databases:

1. PostgreSQL - Open source, powerful
2. MySQL - Open source, widely used
3. SQLite - Embedded, mobile apps
4. Microsoft SQL Server - Enterprise
5. Oracle - Enterprise

Most Popular NoSQL Databases:

1. MongoDB - Document store
2. Redis - Key-value cache
3. Cassandra - Column store
4. Neo4j - Graph database
5. DynamoDB (AWS) - Managed NoSQL

Future Trends:

- **NewSQL:** Combining SQL benefits with NoSQL scalability
- **Multi-model:** Databases supporting multiple data models
- **Cloud-native:** Databases designed for cloud deployment
- **Serverless:** Pay-per-query database services

📚 Further Learning Resources

For SQL:

- PostgreSQL Tutorial (https://www.postgresqltutorial.com/)
- SQLBolt (https://sqlbolt.com/)
- Mode SQL Tutorial (https://mode.com/sql-tutorial/)

For ORMs:

- SQLAlchemy Docs (https://docs.sqlalchemy.org/)
- Django ORM Guide (https://docs.djangoproject.com/en/stable/topics/db/)

For NoSQL:

- MongoDB University (https://university.mongodb.com/)
- Redis University (https://university.redis.com/)
- Neo4j GraphAcademy (https://graphacademy.neo4j.com/)

🎉 Congratulations!

You've completed all 10 lessons of the SQL Basics course! You now have a solid foundation in:

✅ Database fundamentals
✅ SQL query language
✅ Database design and relationships
✅ Python database integration
✅ Understanding of ORM and NoSQL alternatives

What's Next?

1. **Practice:** Build your own database project
2. **Explore:** Try PostgreSQL or MongoDB
3. **Build:** Create a full application with database backend
4. **Learn more:** Study advanced SQL topics (indexes, views, stored procedures)

📊 Course Complete!

You've learnt:

- Created databases and tables
- Written hundreds of SQL queries
- Designed multi-table relationships
- Used JOINs to combine data
- Performed CRUD operations
- Integrated Python with SQLite
- Compared different database technologies

Keep practising and building! 🚀

Need Help?

- Review previous lessons
- Check the resources folder
- Ask your instructor
- Join SQL communities online
- Keep experimenting!

May the Force be with you in your database journey! ⭐