Week 4: Models - Introduction to databases

L4.1 & L4.2: *Persistent Storage & Mechanisms for Persistent Storage and Relational Databases*

- Persistent storage is a mechanism for storing data on a permanent basis, usually on a hard drive or other storage device.
- **Web Applications**: Persistent storage refers to the ability to store and retrieve data that persists across different user sessions or browser refreshes.
- **Model in MVC**: Persistent storage refers to the mechanism used to store and retrieve data within the model component.

Models

- Models are used to store and retrieve data within the model component.
- THe model represents the data and it's business logic.

We can store data in memory like this:

```
class User:
    def __init__(self, name, email, password):
        self.name = name
        self.email = email
        self.password = password
```

OR

```
names = ["John", "Jane", "Jack", ...]
emails = ["johdoe@gmail.com", "jane.123@gmail.com", "jack.ma@gmail.com", ...]
passwords = ["123456", "654321", "108938", ...]
# users will be a list of tuples
users = list(zip(names, emails, passwords))
```

But this data will be lost when the application is closed or restarted.

Here are some common approaches to achieve persistent storage:

1. RDBMS

RDBMS, such as **MySQL**, **PostgreSQL**, **Oracle**, or **SQL Server**, provide persistent storage for structured data.

- Models in the MVC architecture can map to database tables, with each table representing an entity or

object.
- The model interacts with the database using queries and transactions to perform CRUD (Create, Read, Update, Delete) operations on the data.
- Object-Relational Mapping (ORM) frameworks like Hibernate, Django ORM, or Sequelize can be used to simplify database interactions and mapping.

Example: E-commerce application:

- We can create multiple relations such as `User`, `Product`, `Order` etc...
- We can perform CRUD operations on these relations using SQL queries.

2. NoSQL

NoSQL databases, such as **MongoDB**, **Cassandra**, or **Redis**, provide flexible and schema-less storage for unstructured or semi-structured data.

- Models can be designed to work with the data model provided by the NoSQL database.
- The model interacts with the NoSQL database using appropriate APIs or query languages, such as MongoDB Query Language (MQL) for MongoDB.

Example: E-commerce application:

- We can create multiple collections such as `User`, `Product`, `Order` etc...

3. File System

- Data can be stored in files, such as **JSON** or **XML**, and read or written by the model component.
- The model can interact with the file system to store and retrieve data.
- The model needs to handle file I/O operations, including reading, writing, and parsing the data.

Example: E-commerce application:

- We can create multiple files such as `User.json`, `Product.json`, `Order.json` etc...

4. SpreadSheets

- Data can be stored in multiple inter-linked sheets within single spreadsheet.
- Data is organized into rows and columns.
- It can serve as a persistent storage option for small-scale applications or when dealing with tabular data.

Example: E-commerce application:

- We can create multiple sheets such as `Product`, `Review`, `Order` etc...

L4.3: *Relations and ER Diagrams*

Relationship types

One to One

- A single instance of an entity is associated with a single instance of another entity.
- Example: A person has a single passport.

One to Many / Many to One

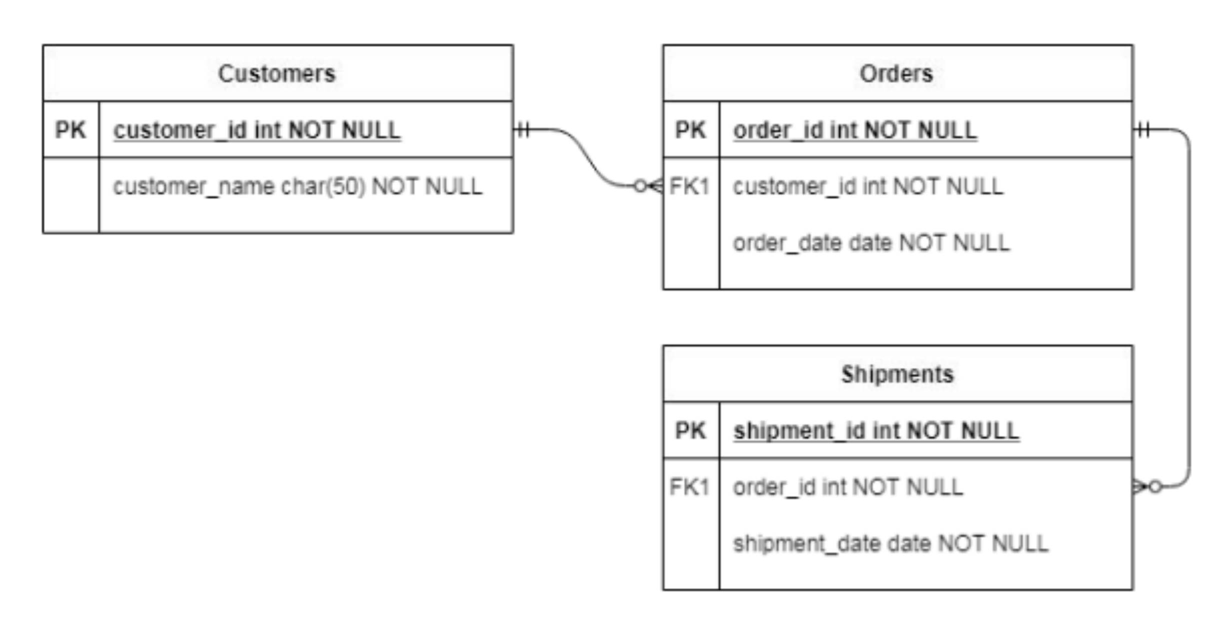
- A single instance of an entity is associated with multiple instances of another entity.
- Example: A person can have multiple credit cards.

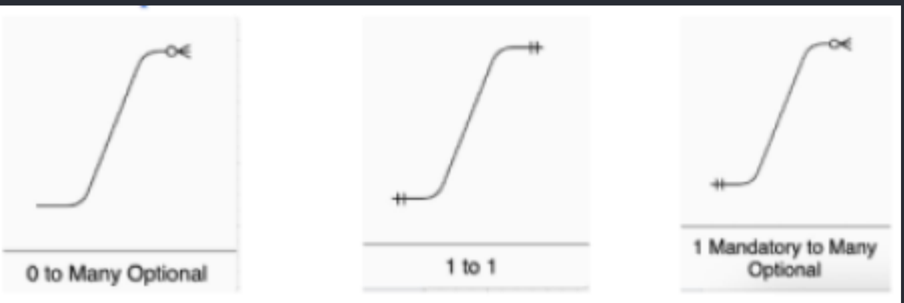
Many to Many

- Multiple instances of an entity are associated with multiple instances of another entity.
- Example: A student can register for many courses and many students can register for a single course.

ER Diagrams

- **Entity-relationship diagrams** (ERDs) are a type of graphical notation used to represent the relationships between entities in a database.
- In Models, ERDs can be used to represent the relationship in different models of an application.





An ERD typically consists of three typs of elements:

Entities:

- These represent the different types of data that are stored in the database.
- Example: Customer and order are entities in an e-commerce application.

Attributes:

- These represent the properties of the entities.
- Example: Customer name, email and phone number are attributes of the customer entity.

Relationships:

- These represents the connections between entities.
- Example: A customer entity can have a relationship with orders entity, indicating that customer placed the order.

L4.4: *SQL*

History of Query Language

- IBM developed *Structured English Query Language (SEQUEL)* in early 1970s.
- Then renamed it to *Structured Query Language (SQL)*.
- Then in 1986 it is formalized as a standard by ANSI and ISO.

Data Definition Language (DDL)

- Specification notation for defining the database schema
- DDL compiler generates a set of table templates stored in a *data dictionary*.
- Data dictionary contains metadata
  - Database schema
  - Integrity constraints
    - Primary Key (ID uniquely identifies instructors)
  - Authorization
    - Who can access what

Domain Types in SQL (DataTypes)

- `char(n)` - Fixed length character string of length `n`.
- `varchar(n)` - Variable length character string of maximum length `n`.
- `int` - Integer (32 bits)
- `smallint(n)` - Smaller integer with maximum `n` digits.

- `numeric(p, d)` - Fixed point number with `p` digits, `d` of which are after the decimal point.
  - Example: `numeric(5, 2)` can store `-999.99` to `999.99`.
- `float(n)` - Floating point number with `n` digits of precision.

Create Table Construct

1. Start with the `CREATE TABLE` keyword.
2. Specify the name of the table.
3. List the columsn in the table, along with their data types.
4. Optionally, specify constraints for the columns.
5. End the statement with a semicolon.

Example

```
CREATE TABLE customers (
  customer_id INT NOT NULL AUTO_INCREMENT,
  first_name VARCHAR(20),
  last_name VARCHAR(10),
  email VARCHAR(50),
  PRIMARY KEY (customer_id)
);
```

This statement creates a table named `customers` with the following columns:

- `customer_id` is an integer that is the primary key of the table.
- `first_name` and `last_name` are strings that store the customer's first and last name.
- `email` is a string that stores the customer's email address.

The `NOT NULL` constraint on the `customer_id` column ensures that this column cannot be null. The `PRIMARY KEY` constraint on the `customer_id` column ensures that this column is unique.

```
CREATE TABLE orders (
  order_id INT AUTO_INCREMENT,
  consumer_id INT NOT NULL,
  order_date DATETIME,
  total_price DECIMAL(10,2),
  PRIMARY KEY (order_id),
  FOREIGN KEY (consumer_id) REFERENCES customers(customer_id)
);
```

This statement creates a table called `orders` with the following columns:

- `order_id` is an integer that is the primary key of the table.
- `consumer_id` is an integer that references the `customer_id` column in the `customers` table.
- `order_date` is a date and time that stores the date and time of the order.
- `total_price` is a decimal number that stores the total price of the order.

`PRIMARY KEY` declaration on an attribute automatically ensures `NOT NULL`.

Check rest of the SQL queries here 🔗 (Pg no. 1111)

Lab Assignment Hints

1. Read the problem statement carefully, understand the requirements.
2. Install required packages using `pip install flask matplotlib`.
3. Use `matplotlib.use("Agg")` to prevent the error: `RuntimeError: main thread is not in main loop`
4. Check for inputs properly, does user entered only numeric value, because `int()` will throw error if user enters string, use `try-except` to handle this.
5. Check Jinja docs 🔗 to know how to use `for` loop in Jinja.

For Flask, you can check this tutorial by CS50.