Python Database Interaction – Case Studies

Case Study 1: Student Management System

Difficulty: Beginner

Problem Statement:

You are building a **Student Management System** to maintain student records for a school. This system allows adding, searching, updating, and deleting student information stored in a **SQL database**.

Requirements:

1. **Database Table: `students`**
    - `id` (Primary Key, Auto Increment)
    - `name` (VARCHAR)
    - `age` (INT)
    - `grade` (VARCHAR)
    - `email` (VARCHAR, unique)
2. **Functionalities:**
    - Add a new student record.
    - Update student details (age, grade, email).
    - Delete a student record by ID.
    - Search student by name or grade.
    - Display all students.
3. **Exception Handling Considerations:**
    - Handle **database connection errors**.
    - Handle **duplicate email insertion**.
    - Handle **invalid queries** gracefully.

Learning Points:

- Connect to MySQL using Python (`mysql.connector` or `SQLAlchemy`).
- Execute **CRUD operations**.
- Use **parameterized queries** to prevent SQL injection.
- Handle **database and query exceptions**.

Scenario Example:

- Add students Alice, Bob, Charlie.
- Update Bob's grade from "10th" → "11th".
- Delete student Charlie.
- Search for all students in grade "10th".

Case Study 2: Employee Attendance Tracker

Difficulty: Beginner → Medium

Problem Statement:

You are creating an **Employee Attendance Tracker** for a company. Employees can check in and check out, and the system tracks total hours worked. All data is stored in a **SQL database**.

Requirements:

1. **Database Table: `attendance`**
   - `attendance_id` (Primary Key)
   - `employee_id` (INT)
   - `check_in` (DATETIME)
   - `check_out` (DATETIME)
   - `total_hours` (FLOAT)
2. **Database Table: `employees`**
   - `employee_id` (Primary Key)
   - `name` (VARCHAR)
   - `department` (VARCHAR)

3. **Functionalities:**
   - Add a new employee.
   - Check-in: record current timestamp for an employee.
   - Check-out: record current timestamp and calculate total hours.
   - Generate attendance report for a specific employee or date range.
   - List employees with incomplete attendance (no check-out).
4. **Exception Handling Considerations:**
   - Handle **missing employee ID**.
   - Handle **duplicate check-ins**.
   - Handle **database connection or query errors**.

Learning Points:

- Use **Python datetime** with database.
- Join tables (`employees` + `attendance`) for reports.
- Handle errors in **data insertion and updates**.
- Query data for **report generation**.

Scenario Example:

- Employee Alice checks in at 9:00 AM and checks out at 5:00 PM.
- System calculates 8 hours and stores in `total_hours`.
- Generate a report showing Alice's attendance and total hours for the week.

Case Study 3: Online Store Product Catalog

Difficulty: Medium

Problem Statement:

You are designing a **Product Catalog System** for an online store. Products, categories, and inventory are stored in a **SQL database**. The system allows adding, updating, searching, and reporting on products.

Requirements:

1. **Database Table: `categories`**
   - `category_id` (Primary Key)
   - `category_name` (VARCHAR)
2. **Database Table: `products`**
   - `product_id` (Primary Key)
   - `name` (VARCHAR)
   - `category_id` (Foreign Key)
   - `price` (FLOAT)
   - `stock_quantity` (INT)
3. **Functionalities:**
   - Add new products and categories.
   - Update product details (price, stock).
   - Delete a product by ID.
   - Search products by name, category, or price range.
   - Generate report for low stock products (stock < threshold).
   - Display all products with their category names.
4. **Exception Handling Considerations:**
   - Handle **invalid category ID** when adding product.
   - Handle **foreign key constraint errors**.
   - Handle **database connection failures**.
   - Handle **duplicate product names** gracefully.

Learning Points:

- Understand **one-to-many relationships** (categories → products).
- Use **joins** to display category names with products.
- Perform **CRUD operations** on multiple tables.

- Handle **foreign key and data integrity exceptions**.

Scenario Example:

- Add category “Electronics”.
- Add products: Laptop ($1000), Mouse ($20) in Electronics.
- Update Laptop stock from 10 → 15.
- Search for all products under $50.
- Generate report showing products with stock < 5.

Case Study 4: Student Course Enrollment System

Difficulty: Beginner → Medium

Problem Statement:
A university needs a system to track student course enrollments. Each student can enroll in multiple courses, and the system should record which students are in which courses. All data is stored in a SQL database.

Requirements:

1. **Database Table: students**
   - student_id (Primary Key)
   - name (VARCHAR)
   - email (VARCHAR)
2. **Database Table: courses**
   - course_id (Primary Key)
   - course_name (VARCHAR)
   - credits (INT)
3. **Database Table: enrollments**
   - enrollment_id (Primary Key)
   - student_id (Foreign Key)
   - course_id (Foreign Key)
   - enrollment_date (DATE)

4. **Functionalities:**
   - Add a new student.
   - Add a new course.
   - Enroll a student in a course.
   - List all courses a student is enrolled in.
   - Generate course-wise report of enrolled students.
5. **Exception Handling Considerations:**
   - Handle duplicate enrollment (student cannot enroll twice in the same course).
   - Handle invalid student or course IDs.
   - Handle missing data during enrollment.

Learning Points:

- Use relationships (many-to-many) in SQL.
- Perform `JOIN` queries across multiple tables.
- Handle uniqueness constraints.
- Manage insertion and validation of relational data.

Scenario Example:

- Student **John** enrolls in **Python Programming** and **Data Structures**.
- Enrollment dates are recorded.
- When generating a course-wise report, both courses show John in the enrolled student list.

Case Study 5: Online Order Management System

Difficulty: Beginner → Medium

Problem Statement:
An e-commerce company needs an order management system to track products, customers, and orders. All data is stored in a SQL database.

Requirements:

1. **Database Table: customers**
   - customer_id (Primary Key)
   - name (VARCHAR)

    - email (VARCHAR)
2. **Database Table: products**
    - product_id (Primary Key)
    - product_name (VARCHAR)
    - price (FLOAT)
    - stock (INT)
3. **Database Table: orders**
    - order_id (Primary Key)
    - customer_id (Foreign Key)
    - order_date (DATE)
    - total_amount (FLOAT)
4. **Database Table: order_items**
    - item_id (Primary Key)
    - order_id (Foreign Key)
    - product_id (Foreign Key)
    - quantity (INT)
    - item_price (FLOAT)
5. **Functionalities:**
    - Add new customers and products.
    - Place an order (reduce stock, calculate total).
    - Generate invoice for an order.
    - View order history by customer.
    - Generate sales report by date range.
6. **Exception Handling Considerations:**
    - Handle insufficient stock.
    - Handle invalid product or customer IDs.
    - Handle transaction rollback if part of the order fails.

Learning Points:

- Use multiple related tables with foreign keys.
- Implement transactions in SQL.
- Aggregate sales using `SUM()` and `GROUP BY`.
- Connect Python with SQL for order placement.

Scenario Example:

- Customer **Bob** orders 2 × *Laptop* and 1 × *Mouse*.
- Stock decreases accordingly.
- System generates invoice with total = (2 × Laptop price + 1 × Mouse price).
- Sales report for the day includes Bob’s purchase.