SQL Practice Challenges - Chapter 2

Challenge 1: Create a Students Table

Create a table for storing student information with appropriate data types and constraints.

```
-- TODO: Create a table named 'students' with columns:
-- id (primary key, auto increment)
-- name (required, max 100 characters)
-- email (unique, max 150 characters)
-- age (integer)
-- gpa (decimal with 2 decimal places)
-- enrollment_date (date, default current date)
```

Expected Result: A properly structured students table.

Challenge 2: Insert Student Data

Add multiple students to the table you created.

```
-- TODO: Insert the following students:
-- Name: 'Alice Johnson', Email: 'alice@university.edu', Age: 20, GPA: 3.75
-- Name: 'Bob Smith', Email: 'bob@university.edu', Age: 22, GPA: 3.25
-- Name: 'Carol Davis', Email: 'carol@university.edu', Age: 19, GPA: 3.90
```

Expected Result: Three student records in the database.

Challenge 3: Update Student Information

Modify existing student data based on conditions.

```
-- TODO: Update Bob Smith's GPA to 3.50
-- TODO: Update all students' age by adding 1 year
```

Expected Result: Updated student records with new GPA and ages.

Challenge 4: Create and Populate a Courses Table

Create a courses table and add sample data.

```
-- TODO: Create 'courses' table with:
-- id (primary key)
-- course_name (required, max 100 characters)
-- credits (integer, default 3)
-- instructor (max 100 characters)

-- TODO: Insert these courses:
-- 'Introduction to SQL', 3 credits, 'Dr. Smith'
-- 'Database Design', 4 credits, 'Dr. Johnson'
-- 'Data Analysis', 3 credits, 'Prof. Davis'
```

Challenge 5: Data Cleanup

Remove unwanted data from tables.

```
-- TODO: Delete students with GPA less than 3.0
-- TODO: Delete courses with less than 3 credits
```

Expected Result: Only high-performing students and substantial courses remain.

Challenge 6: Table Modifications

Alter existing table structure.

```
-- TODO: Add a 'major' column to the students table (VARCHAR 50)
-- TODO: Update Alice's major to 'Computer Science'
-- TODO: Update Bob's major to 'Mathematics'
```

Solutions

Challenge 1 Solution:

```
CREATE TABLE students (
    id INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE,
    age INTEGER,
    gpa DECIMAL(3,2),
    enrollment_date DATE DEFAULT CURRENT_DATE
);
```

Challenge 2 Solution:

```
INSERT INTO students (name, email, age, gpa)
VALUES
    ('Alice Johnson', 'alice@university.edu', 20, 3.75),
    ('Bob Smith', 'bob@university.edu', 22, 3.25),
    ('Carol Davis', 'carol@university.edu', 19, 3.90);
```

Challenge 3 Solution:

```
-- Update Bob's GPA
UPDATE students
SET gpa = 3.50
WHERE name = 'Bob Smith';

-- Add 1 year to all students
UPDATE students
SET age = age + 1;
```

Challenge 4 Solution:

```
-- Create courses table
CREATE TABLE courses (
    id INTEGER PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL,
    credits INTEGER DEFAULT 3,
    instructor VARCHAR(100)
);

-- Insert courses
INSERT INTO courses (id, course_name, credits, instructor)
VALUES
    (1, 'Introduction to SQL', 3, 'Dr. Smith'),
    (2, 'Database Design', 4, 'Dr. Johnson'),
    (3, 'Data Analysis', 3, 'Prof. Davis');
```

Challenge 5 Solution:

```
-- Delete low GPA students
DELETE FROM students
WHERE gpa < 3.0;

-- Delete courses with low credits
DELETE FROM courses
WHERE credits < 3;
```

Challenge 6 Solution:

```
-- Add major column
ALTER TABLE students
ADD COLUMN major VARCHAR(50);

-- Update majors
```

```
UPDATE students
SET major = 'Computer Science'
WHERE name = 'Alice Johnson';

UPDATE students
SET major = 'Mathematics'
WHERE name = 'Bob Smith';
```

Bonus Challenges

Bonus 1: Create a Complete School Database

```
-- Create related tables with foreign keys
CREATE TABLE departments (
    id INTEGER PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    head_of_department VARCHAR(100)
);

CREATE TABLE students (
    id INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE,
    department_id INTEGER,
    FOREIGN KEY (department_id) REFERENCES departments(id)
);

-- Insert sample data
INSERT INTO departments (id, name, head_of_department)
VALUES
    (1, 'Computer Science', 'Dr. Johnson'),
    (2, 'Mathematics', 'Dr. Smith');

INSERT INTO students (name, email, department_id)
VALUES
    ('Alice Johnson', 'alice@university.edu', 1),
    ('Bob Smith', 'bob@university.edu', 2);
```

Bonus 2: Advanced Data Operations

```
-- Find students in each department
SELECT d.name as department, COUNT(s.id) as student_count
FROM departments d
LEFT JOIN students s ON d.id = s.department_id
GROUP BY d.id, d.name;

-- Update multiple tables in transaction
BEGIN TRANSACTION;
UPDATE students SET email = 'newemail@university.edu' WHERE id = 1;
UPDATE departments SET head_of_department = 'Dr. Wilson' WHERE id = 1;
COMMIT;
```

Bonus 3: Data Validation and Cleanup

```
-- Add constraints to existing table
ALTER TABLE students
ADD CONSTRAINT check_gpa CHECK (gpa >= 0.0 AND gpa <= 4.0);

ALTER TABLE students
ADD CONSTRAINT check_age CHECK (age >= 16 AND age <= 100);

-- Clean up invalid data before adding constraints
UPDATE students SET gpa = 4.0 WHERE gpa > 4.0;
DELETE FROM students WHERE age < 16 OR age > 100;

-- Create view for high-performing students
CREATE VIEW honor_students AS
SELECT name, email, gpa
FROM students
WHERE gpa >= 3.5
ORDER BY gpa DESC;
```

Bonus 4: Backup and Recovery

```
-- Create backup table
CREATE TABLE students_backup AS
SELECT * FROM students;

-- Restore from backup if needed
DELETE FROM students;
INSERT INTO students SELECT * FROM students_backup;

-- Export data for external backup
SELECT * FROM students
INTO OUTFILE '/backup/students.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Common Patterns

Pattern 1: Bulk Operations

```
-- Update multiple records efficiently
UPDATE products
SET price = CASE
    WHEN category = 'Electronics' THEN price * 0.9
    WHEN category = 'Clothing' THEN price * 0.8
    ELSE price * 0.95
END;

-- Insert from another table
INSERT INTO archived_orders (order_id, customer_id, order_date)
SELECT id, customer_id, order_date
```

```
FROM orders
WHERE order_date < '2023-01-01';
```

Pattern 2: Data Verification

```
-- Check for orphaned records
SELECT s.* FROM students s
LEFT JOIN departments d ON s.department_id = d.id
WHERE d.id IS NULL;

-- Find duplicate emails
SELECT email, COUNT(*)
FROM students
GROUP BY email
HAVING COUNT(*) > 1;

-- Validate data integrity
SELECT * FROM students
WHERE gpa < 0 OR gpa > 4.0 OR age < 16;
```

Pattern 3: Table Maintenance

```
-- Add indexes for better performance
CREATE INDEX idx_student_email ON students(email);
CREATE INDEX idx_student_department ON students(department_id);

-- Analyze table statistics
ANALYZE TABLE students;

-- Optimize table storage
OPTIMIZE TABLE students;

-- Show table information
DESCRIBE students;
SHOW CREATE TABLE students;
```