**Exercise 1: Creating and Populating the Authors Table**

**Exercise:**

1. Create a table named authors with the following structure:
   * author\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * email: Variable character, Unique.
   * date\_of\_birth: Date.
2. Insert the following records:
   * ('J.K.', 'Rowling', 'jk.rowling@example.com', '1965-07-31')
   * ('George', 'Orwell', 'george.orwell@example.com', '1903-06-25')
   * ('Jane', 'Austen', 'jane.austen@example.com', '1775-12-16')
3. Try to insert a duplicate email ('Herman', 'Melville', 'jane.austen@example.com', '1819-08-01').

**Solution:**

-- Step 1: Create the table

CREATE TABLE authors (

author\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE,

date\_of\_birth DATE,

PRIMARY KEY (author\_id)

);

-- Step 2: Insert the records

INSERT INTO authors (first\_name, last\_name, email, date\_of\_birth)

VALUES

('J.K.', 'Rowling', 'jk.rowling@example.com', '1965-07-31'),

('George', 'Orwell', 'george.orwell@example.com', '1903-06-25'),

('Jane', 'Austen', 'jane.austen@example.com', '1775-12-16');

-- Step 3: Attempt to insert duplicate email (This will fail)

INSERT INTO authors (first\_name, last\_name, email, date\_of\_birth)

VALUES ('Herman', 'Melville', 'jane.austen@example.com', '1819-08-01');

**Exercise 2: Ensuring Positive Prices in the Books Table**

**Exercise:**

1. Create a table named books with the following structure:
   * book\_id: Integer, Auto Increment, Primary Key.
   * title: Variable character, Not Null.
   * author\_id: Integer, Foreign Key referencing authors.author\_id.
   * isbn: Variable character, Unique, Not Null.
   * published\_date: Date.
   * price: Decimal, greater than 0 (Use CHECK constraint).
   * available\_copies: Integer, greater than or equal to 0 (Use CHECK constraint).
2. Insert the following records:
   * ('Harry Potter and the Goblet of Fire', 1, '9780439139601', '2000-07-08', 22.99, 5)
   * ('1984', 2, '9780451524935', '1949-06-08', 15.50, 10)
3. Try to insert a book with a negative price ('Animal Farm', 2, '9780451526342', '1945-08-17', -10.00, 5).

**Solution:**

-- Step 1: Create the table

CREATE TABLE books (

book\_id INT AUTO\_INCREMENT,

title VARCHAR(200) NOT NULL,

author\_id INT,

isbn VARCHAR(13) UNIQUE NOT NULL,

published\_date DATE,

price DECIMAL(10, 2) CHECK (price > 0),

available\_copies INT CHECK (available\_copies >= 0),

PRIMARY KEY (book\_id),

FOREIGN KEY (author\_id) REFERENCES authors(author\_id)

);

-- Step 2: Insert the records

INSERT INTO books (title, author\_id, isbn, published\_date, price, available\_copies)

VALUES

('Harry Potter and the Goblet of Fire', 1, '9780439139601', '2000-07-08', 22.99, 5),

('1984', 2, '9780451524935', '1949-06-08', 15.50, 10);

-- Step 3: Attempt to insert a negative price (This will fail)

INSERT INTO books (title, author\_id, isbn, published\_date, price, available\_copies)

VALUES ('Animal Farm', 2, '9780451526342', '1945-08-17', -10.00, 5);

**Exercise 3: Managing Library Members with Constraints**

**Exercise:**

1. Create a table named members with the following structure:
   * member\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * email: Variable character, Unique, Not Null.
   * phone: Variable character, optional.
   * join\_date: Date, default to current date.
2. Insert the following records:
   * ('Alice', 'Johnson', 'alice.johnson@example.com', '1234567890')
   * ('Bob', 'Smith', 'bob.smith@example.com', '0987654321')
3. Try to insert a member with a NULL email ('Charlie', 'Brown', NULL, '1122334455').

**Solution:**

-- Step 1: Create the table

CREATE TABLE members (

member\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(15),

join\_date DATE DEFAULT CURRENT\_DATE,

PRIMARY KEY (member\_id)

);

-- Step 2: Insert the records

INSERT INTO members (first\_name, last\_name, email, phone)

VALUES

('Alice', 'Johnson', 'alice.johnson@example.com', '1234567890'),

('Bob', 'Smith', 'bob.smith@example.com', '0987654321');

-- Step 3: Attempt to insert with NULL email (This will fail)

INSERT INTO members (first\_name, last\_name, email, phone)

VALUES ('Charlie', 'Brown', NULL, '1122334455');

**Exercise 4: Borrowing Records with Date Constraints**

**Exercise:**

1. Create a table named borrow\_records with the following structure:
   * borrow\_id: Integer, Auto Increment, Primary Key.
   * member\_id: Integer, Foreign Key referencing members.member\_id.
   * book\_id: Integer, Foreign Key referencing books.book\_id.
   * borrow\_date: Date, Not Null.
   * return\_date: Date.
   * due\_date: Date, Not Null.
   * Check constraint to ensure that return\_date is either NULL or greater than or equal to borrow\_date.
2. Insert the following records:
   * member\_id = 1, book\_id = 1, borrow\_date = '2024-09-01', return\_date = '2024-09-15', due\_date = '2024-09-30'
   * member\_id = 2, book\_id = 2, borrow\_date = '2024-09-05', return\_date = NULL, due\_date = '2024-09-25'
3. Try to insert a record with return\_date before borrow\_date member\_id = 1, book\_id = 1, borrow\_date = '2024-09-10', return\_date = '2024-09-05', due\_date = '2024-09-20'.

**Solution:**

-- Step 1: Create the table

CREATE TABLE borrow\_records (

borrow\_id INT AUTO\_INCREMENT,

member\_id INT NOT NULL,

book\_id INT NOT NULL,

borrow\_date DATE NOT NULL,

return\_date DATE,

due\_date DATE NOT NULL,

PRIMARY KEY (borrow\_id),

FOREIGN KEY (member\_id) REFERENCES members(member\_id),

FOREIGN KEY (book\_id) REFERENCES books(book\_id),

CHECK (return\_date IS NULL OR return\_date >= borrow\_date)

);

-- Step 2: Insert the records

INSERT INTO borrow\_records (member\_id, book\_id, borrow\_date, return\_date, due\_date)

VALUES

(1, 1, '2024-09-01', '2024-09-15', '2024-09-30'),

(2, 2, '2024-09-05', NULL, '2024-09-25');

-- Step 3: Attempt to insert return\_date before borrow\_date (This will fail)

INSERT INTO borrow\_records (member\_id, book\_id, borrow\_date, return\_date, due\_date)

VALUES (1, 1, '2024-09-10', '2024-09-05', '2024-09-20');

**Exercise 5: Complex Insert and Constraint Check**

**Exercise:**

1. Insert a new author ('Dan', 'Brown', 'dan.brown@example.com', '1964-06-22').
2. Insert a new book ('The Da Vinci Code', 5, '9780307474278', '2003-03-18', 12.99, 7).
3. Insert a member ('Emma', 'Watson', 'emma.watson@example.com', '5551234567').
4. Create a borrowing record for the newly created member and book, with borrow\_date being today and due\_date being 15 days later.

**Solution:**

-- Step 1: Insert the new author

INSERT INTO authors (first\_name, last\_name, email, date\_of\_birth)

VALUES ('Dan', 'Brown', 'dan.brown@example.com', '1964-06-22');

-- Step 2: Insert the new book

INSERT INTO books (title, author\_id, isbn, published\_date, price, available\_copies)

VALUES ('The Da Vinci Code', 5, '9780307474278', '2003-03-18', 12.99, 7);

-- Step 3: Insert the new member

INSERT INTO members (first\_name, last\_name, email, phone)

VALUES ('Emma', 'Watson', 'emma.watson@example.com', '5551234567');

-- Step 4: Insert the borrowing record

INSERT INTO borrow\_records (member\_id, book\_id, borrow\_date, due\_date)

VALUES (3, 3, CURRENT\_DATE, DATE\_ADD(CURRENT\_DATE, INTERVAL 15 DAY));

**Exercise 6: Managing a Movie Database**

**Exercise:**

1. Create a table named directors with the following structure:
   * director\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * birth\_date: Date, Not Null.
   * country: Variable character, Not Null, Default to 'Unknown'.
2. Create a table named movies with the following structure:
   * movie\_id: Integer, Auto Increment, Primary Key.
   * title: Variable character, Not Null.
   * release\_year: Year, Not Null, Check constraint to ensure year is greater than or equal to 1900.
   * genre: Variable character, Not Null, Default to 'Drama'.
   * director\_id: Integer, Foreign Key referencing directors.director\_id.
   * rating: Decimal(3,1), Not Null, Check constraint to ensure value is between 0 and 10.
   * Unique constraint on (title, release\_year).
3. Insert the following directors:
   * ('Steven', 'Spielberg', '1946-12-18', 'USA')
   * ('Christopher', 'Nolan', '1970-07-30', 'UK')
   * ('Quentin', 'Tarantino', '1963-03-27', 'USA')
   * ('Unknown', 'Director', '1900-01-01', NULL) (will use default value for country)
4. Insert the following movies:
   * ('Inception', 2010, 'Sci-Fi', 2, 8.8)
   * ('Jurassic Park', 1993, 'Adventure', 1, 8.1)
   * ('Pulp Fiction', 1994, 'Crime', 3, 8.9)
5. Try to insert a movie with a year less than 1900 and a rating greater than 10.

**Solution:**

-- Step 1: Create the `directors` table

CREATE TABLE directors (

director\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

birth\_date DATE NOT NULL,

country VARCHAR(100) DEFAULT 'Unknown',

PRIMARY KEY (director\_id)

);

-- Step 2: Create the `movies` table

CREATE TABLE movies (

movie\_id INT AUTO\_INCREMENT,

title VARCHAR(100) NOT NULL,

release\_year YEAR NOT NULL CHECK (release\_year >= 1900),

genre VARCHAR(50) NOT NULL DEFAULT 'Drama',

director\_id INT,

rating DECIMAL(3,1) NOT NULL CHECK (rating BETWEEN 0 AND 10),

UNIQUE (title, release\_year),

PRIMARY KEY (movie\_id),

FOREIGN KEY (director\_id) REFERENCES directors(director\_id)

);

-- Step 3: Insert directors

INSERT INTO directors (first\_name, last\_name, birth\_date, country)

VALUES

('Steven', 'Spielberg', '1946-12-18', 'USA'),

('Christopher', 'Nolan', '1970-07-30', 'UK'),

('Quentin', 'Tarantino', '1963-03-27', 'USA'),

('Unknown', 'Director', '1900-01-01', NULL);

-- Step 4: Insert movies

INSERT INTO movies (title, release\_year, genre, director\_id, rating)

VALUES

('Inception', 2010, 'Sci-Fi', 2, 8.8),

('Jurassic Park', 1993, 'Adventure', 1, 8.1),

('Pulp Fiction', 1994, 'Crime', 3, 8.9);

-- Step 5: Try to insert an invalid movie (This will fail)

INSERT INTO movies (title, release\_year, genre, director\_id, rating)

VALUES ('Invalid Movie', 1800, 'Drama', 1, 15.0);

**Exercise 7: Employee-Department Relationship with Salary Constraints**

**Exercise:**

1. Create a table named departments with the following structure:
   * department\_id: Integer, Auto Increment, Primary Key.
   * department\_name: Variable character, Unique, Not Null.
2. Create a table named employees with the following structure:
   * employee\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * department\_id: Integer, Foreign Key referencing departments.department\_id.
   * hire\_date: Date, Not Null, Default to the current date.
   * salary: Decimal(10,2), Not Null, Check constraint to ensure salary is greater than 2000.
   * Unique constraint on (first\_name, last\_name).
3. Insert the following departments:
   * ('HR')
   * ('Finance')
   * ('IT')
   * ('Sales')
4. Insert the following employees:
   * ('Alice', 'Johnson', 1, '2024-01-10', 5000.00)
   * ('Bob', 'Smith', 2, '2024-03-15', 4500.00)
   * ('Charlie', 'Brown', 3, NULL, 3000.00) (uses default hire\_date)
   * ('David', 'Wilson', 3, NULL, 1800.00) (This should fail due to salary constraint)

**Solution:**

-- Step 1: Create the `departments` table

CREATE TABLE departments (

department\_id INT AUTO\_INCREMENT,

department\_name VARCHAR(100) UNIQUE NOT NULL,

PRIMARY KEY (department\_id)

);

-- Step 2: Create the `employees` table

CREATE TABLE employees (

employee\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

department\_id INT,

hire\_date DATE DEFAULT CURRENT\_DATE,

salary DECIMAL(10, 2) NOT NULL CHECK (salary > 2000),

UNIQUE (first\_name, last\_name),

PRIMARY KEY (employee\_id),

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

-- Step 3: Insert departments

INSERT INTO departments (department\_name)

VALUES ('HR'), ('Finance'), ('IT'), ('Sales');

-- Step 4: Insert employees

INSERT INTO employees (first\_name, last\_name, department\_id, hire\_date, salary)

VALUES

('Alice', 'Johnson', 1, '2024-01-10', 5000.00),

('Bob', 'Smith', 2, '2024-03-15', 4500.00),

('Charlie', 'Brown', 3, NULL, 3000.00);

-- Step 5: Try to insert with invalid salary (This will fail)

INSERT INTO employees (first\_name, last\_name, department\_id, hire\_date, salary)

VALUES ('David', 'Wilson', 3, NULL, 1800.00);

**Exercise 8: Inventory Management System**

**Exercise:**

1. Create a table named suppliers with the following structure:
   * supplier\_id: Integer, Auto Increment, Primary Key.
   * supplier\_name: Variable character, Unique, Not Null.
   * contact\_name: Variable character, Not Null.
   * phone: Variable character, Not Null.
   * email: Variable character, Not Null, Unique.
2. Create a table named products with the following structure:
   * product\_id: Integer, Auto Increment, Primary Key.
   * product\_name: Variable character, Unique, Not Null.
   * supplier\_id: Integer, Foreign Key referencing suppliers.supplier\_id.
   * unit\_price: Decimal(10, 2), Not Null, Check constraint to ensure price is greater than 0.
   * units\_in\_stock: Integer, Not Null, Check constraint to ensure value is greater than or equal to 0.
3. Insert the following suppliers:
   * ('Supplier A', 'John Doe', '123-456-7890', 'contact@supplierA.com')
   * ('Supplier B', 'Jane Doe', '234-567-8901', 'contact@supplierB.com')
4. Insert the following products:
   * ('Product 1', 1, 20.00, 50)
   * ('Product 2', 2, 15.50, 100)
   * ('Product 3', 1, -5.00, 30) (This should fail due to negative price)
   * ('Product 4', 2, 30.00, -10) (This should fail due to negative stock)

**Solution:**

-- Step 1: Create the `suppliers` table

CREATE TABLE suppliers (

supplier\_id INT AUTO\_INCREMENT,

supplier\_name VARCHAR(100) UNIQUE NOT NULL,

contact\_name VARCHAR(100) NOT NULL,

phone VARCHAR(20) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

PRIMARY KEY (supplier\_id)

);

-- Step 2: Create the `products` table

CREATE TABLE products (

product\_id INT AUTO\_INCREMENT,

product\_name VARCHAR(100) UNIQUE NOT NULL,

supplier\_id INT,

unit\_price DECIMAL(10, 2) NOT NULL CHECK (unit\_price > 0),

units\_in\_stock INT NOT NULL CHECK (units\_in\_stock >= 0),

PRIMARY KEY (product\_id),

FOREIGN KEY (supplier\_id) REFERENCES suppliers(supplier\_id)

);

-- Step 3: Insert suppliers

INSERT INTO suppliers (supplier\_name, contact\_name, phone, email)

VALUES

('Supplier A', 'John Doe', '123-456-7890', 'contact@supplierA.com'),

('Supplier B', 'Jane Doe', '234-567-8901', 'contact@supplierB.com');

-- Step 4: Insert products

INSERT INTO products (product\_name, supplier\_id, unit\_price, units\_in\_stock)

VALUES

('Product 1', 1, 20.00, 50),

('Product 2', 2, 15.50, 100);

-- Step 5: Try to insert with negative price (This will fail)

INSERT INTO products (product\_name, supplier\_id, unit\_price, units\_in\_stock)

VALUES ('Product 3', 1, -5.00, 30);

-- Step 6: Try to insert with negative stock (This will fail)

INSERT INTO products (product\_name, supplier\_id, unit\_price, units\_in\_stock)

VALUES ('Product 4', 2, 30.00, -10);

**Exercise 9: E-commerce Order Management System**

**Exercise:**

1. Create a table named customers with the following structure:
   * customer\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * email: Variable character, Not Null, Unique.
   * phone: Variable character, Unique, Not Null.
   * registration\_date: Date, Default to current date.
2. Create a table named products with the following structure:
   * product\_id: Integer, Auto Increment, Primary Key.
   * product\_name: Variable character, Unique, Not Null.
   * price: Decimal(10, 2), Not Null, Check constraint to ensure price is greater than 0.
   * stock\_quantity: Integer, Not Null, Check constraint to ensure quantity is greater than or equal to 0.
   * category: Variable character, Not Null, Default to 'General'.
3. Create a table named orders with the following structure:
   * order\_id: Integer, Auto Increment, Primary Key.
   * customer\_id: Integer, Foreign Key referencing customers.customer\_id.
   * order\_date: Date, Not Null, Default to current date.
   * total\_amount: Decimal(10, 2), Not Null, Default to 0.
4. Create a table named order\_items with the following structure:
   * order\_item\_id: Integer, Auto Increment, Primary Key.
   * order\_id: Integer, Foreign Key referencing orders.order\_id.
   * product\_id: Integer, Foreign Key referencing products.product\_id.
   * quantity: Integer, Not Null, Check constraint to ensure quantity is greater than 0.
   * item\_price: Decimal(10, 2), Not Null, Check constraint to ensure price is greater than 0.
   * Unique constraint on (order\_id, product\_id).
5. Insert the following customers:
   * ('John', 'Doe', 'john.doe@example.com', '1234567890')
   * ('Jane', 'Smith', 'jane.smith@example.com', '0987654321')
6. Insert the following products:
   * ('Laptop', 999.99, 10, 'Electronics')
   * ('Headphones', 49.99, 50, 'Accessories')
   * ('Coffee Mug', 9.99, 100, 'Kitchen')
7. Insert an order for John Doe with the following items:
   * 2 Laptops at $999.99 each.
   * 3 Coffee Mugs at $9.99 each.
8. Update the orders.total\_amount field to reflect the correct total for the above order.
9. Try to insert an order item with a quantity of 0 and a price of -5.

**Solution:**

-- Step 1: Create the `customers` table

CREATE TABLE customers (

customer\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(20) UNIQUE NOT NULL,

registration\_date DATE DEFAULT CURRENT\_DATE,

PRIMARY KEY (customer\_id)

);

-- Step 2: Create the `products` table

CREATE TABLE products (

product\_id INT AUTO\_INCREMENT,

product\_name VARCHAR(100) UNIQUE NOT NULL,

price DECIMAL(10, 2) NOT NULL CHECK (price > 0),

stock\_quantity INT NOT NULL CHECK (stock\_quantity >= 0),

category VARCHAR(50) NOT NULL DEFAULT 'General',

PRIMARY KEY (product\_id)

);

-- Step 3: Create the `orders` table

CREATE TABLE orders (

order\_id INT AUTO\_INCREMENT,

customer\_id INT,

order\_date DATE NOT NULL DEFAULT CURRENT\_DATE,

total\_amount DECIMAL(10, 2) NOT NULL DEFAULT 0,

PRIMARY KEY (order\_id),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

-- Step 4: Create the `order\_items` table

CREATE TABLE order\_items (

order\_item\_id INT AUTO\_INCREMENT,

order\_id INT,

product\_id INT,

quantity INT NOT NULL CHECK (quantity > 0),

item\_price DECIMAL(10, 2) NOT NULL CHECK (item\_price > 0),

UNIQUE (order\_id, product\_id),

PRIMARY KEY (order\_item\_id),

FOREIGN KEY (order\_id) REFERENCES orders(order\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

-- Step 5: Insert customers

INSERT INTO customers (first\_name, last\_name, email, phone)

VALUES

('John', 'Doe', 'john.doe@example.com', '1234567890'),

('Jane', 'Smith', 'jane.smith@example.com', '0987654321');

-- Step 6: Insert products

INSERT INTO products (product\_name, price, stock\_quantity, category)

VALUES

('Laptop', 999.99, 10, 'Electronics'),

('Headphones', 49.99, 50, 'Accessories'),

('Coffee Mug', 9.99, 100, 'Kitchen');

-- Step 7: Insert order for John Doe

INSERT INTO orders (customer\_id)

VALUES (1);

-- Step 8: Insert order items for the order

INSERT INTO order\_items (order\_id, product\_id, quantity, item\_price)

VALUES

(1, 1, 2, 999.99), -- 2 Laptops

(1, 3, 3, 9.99); -- 3 Coffee Mugs

-- Step 9: Update total amount in the order

UPDATE orders

SET total\_amount =

(SELECT SUM(quantity \* item\_price)

FROM order\_items

WHERE order\_id = 1)

WHERE order\_id = 1;

-- Step 10: Try to insert invalid order item (This will fail)

INSERT INTO order\_items (order\_id, product\_id, quantity, item\_price)

VALUES (1, 2, 0, -5); -- Invalid quantity and price

**Exercise 10: University Enrollment System**

**Exercise:**

1. Create a table named students with the following structure:
   * student\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * email: Variable character, Not Null, Unique.
   * birthdate: Date, Not Null.
   * enrollment\_date: Date, Default to current date.
2. Create a table named courses with the following structure:
   * course\_id: Integer, Auto Increment, Primary Key.
   * course\_name: Variable character, Unique, Not Null.
   * credits: Integer, Not Null, Check constraint to ensure credits are between 1 and 5.
3. Create a table named enrollments with the following structure:
   * enrollment\_id: Integer, Auto Increment, Primary Key.
   * student\_id: Integer, Foreign Key referencing students.student\_id.
   * course\_id: Integer, Foreign Key referencing courses.course\_id.
   * enrollment\_date: Date, Not Null, Default to current date.
   * Unique constraint on (student\_id, course\_id).
4. Insert the following students:
   * ('Alice', 'Williams', 'alice.williams@example.com', '2000-05-15')
   * ('Bob', 'Johnson', 'bob.johnson@example.com', '1999-10-20')
   * ('Charlie', 'Davis', 'charlie.davis@example.com', '1998-02-28')
5. Insert the following courses:
   * ('Mathematics', 3)
   * ('Physics', 4)
   * ('Chemistry', 5)
6. Enroll Alice Williams and Bob Johnson in the Mathematics course.
7. Enroll Alice Williams in the Physics course twice. (This should fail due to unique constraint.)

**Solution:**

-- Step 1: Create the `students` table

CREATE TABLE students (

student\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

birthdate DATE NOT NULL,

enrollment\_date DATE DEFAULT CURRENT\_DATE,

PRIMARY KEY (student\_id)

);

-- Step 2: Create the `courses` table

CREATE TABLE courses (

course\_id INT AUTO\_INCREMENT,

course\_name VARCHAR(100) UNIQUE NOT NULL,

credits INT NOT NULL CHECK (credits BETWEEN 1 AND 5),

PRIMARY KEY (course\_id)

);

-- Step 3: Create the `enrollments` table

CREATE TABLE enrollments (

enrollment\_id INT AUTO\_INCREMENT,

student\_id INT,

course\_id INT,

enrollment\_date DATE DEFAULT CURRENT\_DATE,

UNIQUE (student\_id, course\_id),

PRIMARY KEY (enrollment\_id),

FOREIGN KEY (student\_id) REFERENCES students(student\_id),

FOREIGN KEY (course\_id) REFERENCES courses(course\_id)

);

-- Step 4: Insert students

INSERT INTO students (first\_name, last\_name, email, birthdate)

VALUES

('Alice', 'Williams', 'alice.williams@example.com', '2000-05-15'),

('Bob', 'Johnson', 'bob.johnson@example.com', '1999-10-20'),

('Charlie', 'Davis', 'charlie.davis@example.com', '1998-02-28');

-- Step 5: Insert courses

INSERT INTO courses (course\_name, credits)

VALUES

('Mathematics', 3),

('Physics', 4),

('Chemistry', 5);

-- Step 6: Enroll students in courses

INSERT INTO enrollments (student\_id, course\_id)

VALUES

(1, 1), -- Alice in Mathematics

(2, 1); -- Bob in Mathematics

-- Step 7: Try to enroll Alice in Physics twice (This will fail)

INSERT INTO enrollments (student\_id, course\_id)

VALUES

(1, 2), -- First enrollment in Physics

(1, 2); -- Duplicate enrollment in Physics (this will fail)

**Exercise 11: Hospital Management System**

**Exercise:**

1. Create a table named doctors with the following structure:
   * doctor\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * specialization: Variable character, Not Null.
   * phone: Variable character, Unique, Not Null.
   * email: Variable character, Unique, Not Null.
2. Create a table named patients with the following structure:
   * patient\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * dob: Date, Not Null.
   * gender: Enum ('Male', 'Female', 'Other'), Not Null.
   * phone: Variable character, Unique, Not Null.
   * email: Variable character, Unique, Not Null.
3. Create a table named appointments with the following structure:
   * appointment\_id: Integer, Auto Increment, Primary Key.
   * doctor\_id: Integer, Foreign Key referencing doctors(doctor\_id).
   * patient\_id: Integer, Foreign Key referencing patients(patient\_id).
   * appointment\_date: Date, Not Null.
   * appointment\_time: Time, Not Null.
   * status: Enum ('Scheduled', 'Completed', 'Canceled'), Not Null.
   * Unique constraint on (doctor\_id, appointment\_date, appointment\_time).
4. Create a table named prescriptions with the following structure:
   * prescription\_id: Integer, Auto Increment, Primary Key.
   * appointment\_id: Integer, Foreign Key referencing appointments(appointment\_id).
   * medicine\_name: Variable character, Not Null.
   * dosage: Variable character, Not Null.
   * duration: Integer, Not Null Check constraint to ensure duration is greater than 0.
   * Unique constraint on (appointment\_id, medicine\_name).
5. Insert the following doctors:
   * ('John', 'Doe', 'Cardiologist', '1234567890', 'john.doe@hospital.com')
   * ('Emily', 'Clark', 'Dermatologist', '0987654321', 'emily.clark@hospital.com')
6. Insert the following patients:
   * ('Alice', 'Johnson', '1990-04-15', 'Female', '2223334444', 'alice.johnson@gmail.com')
   * ('Bob', 'Smith', '1985-12-30', 'Male', '5556667777', 'bob.smith@yahoo.com')
7. Schedule the following appointments:
   * Alice Johnson with Dr. John Doe on 2024-10-01 at 10:00:00.
   * Bob Smith with Dr. Emily Clark on 2024-10-02 at 14:00:00.
8. Insert a prescription for Alice's appointment with the following details:
   * Medicine: Aspirin, Dosage: 1 tablet twice daily, Duration: 7 days.
9. Try to schedule another appointment for Dr. John Doe with Bob Smith on 2024-10-01 at 10:00:00 (This should fail due to unique constraint).
10. Try to insert a prescription with a duration of -5 (This should fail due to check constraint).

**Solution:**

-- Step 1: Create the `doctors` table

CREATE TABLE doctors (

doctor\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

specialization VARCHAR(100) NOT NULL,

phone VARCHAR(20) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

PRIMARY KEY (doctor\_id)

);

-- Step 2: Create the `patients` table

CREATE TABLE patients (

patient\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

dob DATE NOT NULL,

gender ENUM('Male', 'Female', 'Other') NOT NULL,

phone VARCHAR(20) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

PRIMARY KEY (patient\_id)

);

-- Step 3: Create the `appointments` table

CREATE TABLE appointments (

appointment\_id INT AUTO\_INCREMENT,

doctor\_id INT,

patient\_id INT,

appointment\_date DATE NOT NULL,

appointment\_time TIME NOT NULL,

status ENUM('Scheduled', 'Completed', 'Canceled') NOT NULL,

UNIQUE (doctor\_id, appointment\_date, appointment\_time),

PRIMARY KEY (appointment\_id),

FOREIGN KEY (doctor\_id) REFERENCES doctors(doctor\_id),

FOREIGN KEY (patient\_id) REFERENCES patients(patient\_id)

);

-- Step 4: Create the `prescriptions` table

CREATE TABLE prescriptions (

prescription\_id INT AUTO\_INCREMENT,

appointment\_id INT,

medicine\_name VARCHAR(100) NOT NULL,

dosage VARCHAR(100) NOT NULL,

duration INT NOT NULL CHECK (duration > 0),

UNIQUE (appointment\_id, medicine\_name),

PRIMARY KEY (prescription\_id),

FOREIGN KEY (appointment\_id) REFERENCES appointments(appointment\_id)

);

-- Step 5: Insert doctors

INSERT INTO doctors (first\_name, last\_name, specialization, phone, email)

VALUES

('John', 'Doe', 'Cardiologist', '1234567890', 'john.doe@hospital.com'),

('Emily', 'Clark', 'Dermatologist', '0987654321', 'emily.clark@hospital.com');

-- Step 6: Insert patients

INSERT INTO patients (first\_name, last\_name, dob, gender, phone, email)

VALUES

('Alice', 'Johnson', '1990-04-15', 'Female', '2223334444', 'alice.johnson@gmail.com'),

('Bob', 'Smith', '1985-12-30', 'Male', '5556667777', 'bob.smith@yahoo.com');

-- Step 7: Schedule appointments

INSERT INTO appointments (doctor\_id, patient\_id, appointment\_date, appointment\_time, status)

VALUES

(1, 1, '2024-10-01', '10:00:00', 'Scheduled'), -- Alice with Dr. John Doe

(2, 2, '2024-10-02', '14:00:00', 'Scheduled'); -- Bob with Dr. Emily Clark

-- Step 8: Insert prescription for Alice's appointment

INSERT INTO prescriptions (appointment\_id, medicine\_name, dosage, duration)

VALUES

(1, 'Aspirin', '1 tablet twice daily', 7);

-- Step 9: Try to schedule another appointment for Dr. John Doe at the same time (This will fail)

INSERT INTO appointments (doctor\_id, patient\_id, appointment\_date, appointment\_time, status)

VALUES

(1, 2, '2024-10-01', '10:00:00', 'Scheduled'); -- Conflict with previous appointment

-- Step 10: Try to insert a prescription with invalid duration (This will fail)

INSERT INTO prescriptions (appointment\_id, medicine\_name, dosage, duration)

VALUES

(2, 'Ibuprofen', '1 tablet three times daily', -5); -- Invalid duration

**Exercise 12: E-commerce Platform**

**Exercise:**

1. **Create a table named users:**
   * user\_id: Integer, Auto Increment, Primary Key.
   * username: Variable character, Unique, Not Null.
   * password: Variable character, Not Null.
   * email: Variable character, Unique, Not Null.
   * created\_at: Timestamp, Default to current timestamp.
2. **Create a table named products:**
   * product\_id: Integer, Auto Increment, Primary Key.
   * product\_name: Variable character, Unique, Not Null.
   * description: Text, Not Null.
   * price: Decimal(10, 2), Not Null, Check constraint to ensure price is greater than 0.
   * stock\_quantity: Integer, Not Null, Check constraint to ensure stock is greater than or equal to 0.
3. **Create a table named orders:**
   * order\_id: Integer, Auto Increment, Primary Key.
   * user\_id: Integer, Foreign Key referencing users(user\_id).
   * order\_date: Timestamp, Default to current timestamp.
   * status: Enum('Pending', 'Shipped', 'Delivered', 'Canceled'), Not Null.
4. **Create a table named order\_items:**
   * order\_item\_id: Integer, Auto Increment, Primary Key.
   * order\_id: Integer, Foreign Key referencing orders(order\_id).
   * product\_id: Integer, Foreign Key referencing products(product\_id).
   * quantity: Integer, Not Null, Check constraint to ensure quantity is greater than 0.
   * Unique constraint on (order\_id, product\_id).
5. **Insert the following users:**
   * ('alice123', 'password123', 'alice@example.com')
   * ('bob456', 'password456', 'bob@example.com')
6. **Insert the following products:**
   * ('Laptop', 'High-performance laptop', 999.99, 50)
   * ('Smartphone', 'Latest model smartphone', 699.99, 100)
7. **Create an order for Alice and add items to the order:**
   * Alice places an order with ID 1, which includes:
     + 1 Laptop
     + 2 Smartphones
8. **Try to add the same product to Alice's order again (This should fail due to the unique constraint).**
9. **Try to insert a product with a price of -10 (This should fail due to the check constraint).**
10. **Cancel Alice's order and change the status accordingly.**

**Solution:**

-- Step 1: Create the `users` table

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT,

username VARCHAR(50) UNIQUE NOT NULL,

password VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (user\_id)

);

-- Step 2: Create the `products` table

CREATE TABLE products (

product\_id INT AUTO\_INCREMENT,

product\_name VARCHAR(100) UNIQUE NOT NULL,

description TEXT NOT NULL,

price DECIMAL(10, 2) NOT NULL CHECK (price > 0),

stock\_quantity INT NOT NULL CHECK (stock\_quantity >= 0),

PRIMARY KEY (product\_id)

);

-- Step 3: Create the `orders` table

CREATE TABLE orders (

order\_id INT AUTO\_INCREMENT,

user\_id INT,

order\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status ENUM('Pending', 'Shipped', 'Delivered', 'Canceled') NOT NULL,

PRIMARY KEY (order\_id),

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);

-- Step 4: Create the `order\_items` table

CREATE TABLE order\_items (

order\_item\_id INT AUTO\_INCREMENT,

order\_id INT,

product\_id INT,

quantity INT NOT NULL CHECK (quantity > 0),

UNIQUE (order\_id, product\_id),

PRIMARY KEY (order\_item\_id),

FOREIGN KEY (order\_id) REFERENCES orders(order\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

-- Step 5: Insert users

INSERT INTO users (username, password, email)

VALUES

('alice123', 'password123', 'alice@example.com'),

('bob456', 'password456', 'bob@example.com');

-- Step 6: Insert products

INSERT INTO products (product\_name, description, price, stock\_quantity)

VALUES

('Laptop', 'High-performance laptop', 999.99, 50),

('Smartphone', 'Latest model smartphone', 699.99, 100);

-- Step 7: Create an order for Alice

INSERT INTO orders (user\_id)

VALUES

(1); -- Alice's user\_id

-- Step 8: Add items to Alice's order

INSERT INTO order\_items (order\_id, product\_id, quantity)

VALUES

(1, 1, 1), -- 1 Laptop

(1, 2, 2); -- 2 Smartphones

-- Step 9: Try to add the same product again (This will fail)

INSERT INTO order\_items (order\_id, product\_id, quantity)

VALUES

(1, 2, 1); -- Attempting to add another Smartphone to the same order

-- Step 10: Try to insert a product with a negative price (This will fail)

INSERT INTO products (product\_name, description, price, stock\_quantity)

VALUES

('Invalid Product', 'This product has a negative price', -10.00, 5); -- Invalid price

-- Step 11: Cancel Alice's order

UPDATE orders

SET status = 'Canceled'

WHERE order\_id = 1;

**Exercise 13: Event Management System**

**Exercise:**

1. **Create a table named organizers:**
   * organizer\_id: Integer, Auto Increment, Primary Key.
   * name: Variable character, Not Null.
   * email: Variable character, Unique, Not Null.
   * phone: Variable character, Unique, Not Null.
2. **Create a table named venues:**
   * venue\_id: Integer, Auto Increment, Primary Key.
   * venue\_name: Variable character, Not Null.
   * capacity: Integer, Not Null, Check constraint to ensure capacity is greater than 0.
   * location: Variable character, Not Null.
3. **Create a table named events:**
   * event\_id: Integer, Auto Increment, Primary Key.
   * event\_name: Variable character, Not Null.
   * organizer\_id: Integer, Foreign Key referencing organizers(organizer\_id).
   * venue\_id: Integer, Foreign Key referencing venues(venue\_id).
   * event\_date: Date, Not Null.
   * status: Enum('Scheduled', 'Completed', 'Canceled'), Not Null.
4. **Create a table named tickets:**
   * ticket\_id: Integer, Auto Increment, Primary Key.
   * event\_id: Integer, Foreign Key referencing events(event\_id).
   * ticket\_type: Variable character, Not Null.
   * price: Decimal(10, 2), Not Null, Check constraint to ensure price is greater than 0.
   * available\_quantity: Integer, Not Null, Check constraint to ensure quantity is greater than or equal to 0.
5. **Insert the following organizers:**
   * ('John Events', 'john.events@example.com', '1234567890')
   * ('Anna Concerts', 'anna.concerts@example.com', '0987654321')
6. **Insert the following venues:**
   * ('Stadium A', 5000, 'City Center')
   * ('Convention Hall B', 2000, 'Downtown')
7. **Create an event for John Events in Stadium A:**
   * ('Music Fest', 1, 1, '2024-11-15', 'Scheduled')
8. **Create tickets for the event:**
   * ('VIP', 199.99, 100)
   * ('Regular', 99.99, 200)
9. **Try to insert a ticket with a negative price (This should fail due to the check constraint).**
10. **Change the status of the event to Completed.**

**Solution:**

-- Step 1: Create the `organizers` table

CREATE TABLE organizers (

organizer\_id INT AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(20) UNIQUE NOT NULL,

PRIMARY KEY (organizer\_id)

);

-- Step 2: Create the `venues` table

CREATE TABLE venues (

venue\_id INT AUTO\_INCREMENT,

venue\_name VARCHAR(100) NOT NULL,

capacity INT NOT NULL CHECK (capacity > 0),

location VARCHAR(100) NOT NULL,

PRIMARY KEY (venue\_id)

);

-- Step 3: Create the `events` table

CREATE TABLE events (

event\_id INT AUTO\_INCREMENT,

event\_name VARCHAR(100) NOT NULL,

organizer\_id INT,

venue\_id INT,

event\_date DATE NOT NULL,

status ENUM('Scheduled', 'Completed', 'Canceled') NOT NULL,

PRIMARY KEY (event\_id),

FOREIGN KEY (organizer\_id) REFERENCES organizers(organizer\_id),

FOREIGN KEY (venue\_id) REFERENCES venues(venue\_id)

);

-- Step 4: Create the `tickets` table

CREATE TABLE tickets (

ticket\_id INT AUTO\_INCREMENT,

event\_id INT,

ticket\_type VARCHAR(50) NOT NULL,

price DECIMAL(10, 2) NOT NULL CHECK (price > 0),

available\_quantity INT NOT NULL CHECK (available\_quantity >= 0),

PRIMARY KEY (ticket\_id),

FOREIGN KEY (event\_id) REFERENCES events(event\_id)

);

-- Step 5: Insert organizers

INSERT INTO organizers (name, email, phone)

VALUES

('John Events', 'john.events@example.com', '1234567890'),

('Anna Concerts', 'anna.concerts@example.com', '0987654321');

-- Step 6: Insert venues

INSERT INTO venues (venue\_name, capacity, location)

VALUES

('Stadium A', 5000, 'City Center'),

('Convention Hall B', 2000, 'Downtown');

-- Step 7: Create an event for John Events in Stadium A

INSERT INTO events (event\_name, organizer\_id, venue\_id, event\_date, status)

VALUES

('Music Fest', 1, 1, '2024-11-15', 'Scheduled');

-- Step 8: Create tickets for the event

INSERT INTO tickets (event\_id, ticket\_type, price, available\_quantity)

VALUES

(1, 'VIP', 199.99, 100),

(1, 'Regular', 99.99, 200);

-- Step 9: Try to insert a ticket with a negative price (This will fail)

INSERT INTO tickets (event\_id, ticket\_type, price, available\_quantity)

VALUES

(1, 'Invalid Ticket', -50.00, 10); -- Invalid price

-- Step 10: Change the status of the event to Completed

UPDATE events

SET status = 'Completed'

WHERE event\_id = 1;

**Exercise 14: Library Management System**

**Exercise:**

1. **Create a table named members:**
   * member\_id: Integer, Auto Increment, Primary Key.
   * name: Variable character, Not Null.
   * email: Variable character, Unique, Not Null.
   * phone: Variable character, Unique, Not Null.
   * membership\_date: Date, Not Null.
2. **Create a table named books:**
   * book\_id: Integer, Auto Increment, Primary Key.
   * title: Variable character, Unique, Not Null.
   * author: Variable character, Not Null.
   * isbn: Variable character, Unique, Not Null.
   * copies\_available: Integer, Not Null, Check constraint to ensure greater than or equal to 0.
3. **Create a table named loans:**
   * loan\_id: Integer, Auto Increment, Primary Key.
   * member\_id: Integer, Foreign Key referencing members(member\_id).
   * book\_id: Integer, Foreign Key referencing books(book\_id).
   * loan\_date: Date, Not Null.
   * return\_date: Date, Nullable.
   * Check constraint to ensure return\_date is after loan\_date.
4. **Insert the following members:**
   * ('John Doe', 'john@example.com', '1234567890', '2024-01-15')
   * ('Jane Smith', 'jane@example.com', '0987654321', '2024-02-10')
5. **Insert the following books:**
   * ('The Great Gatsby', 'F. Scott Fitzgerald', '9780743273565', 5)
   * ('To Kill a Mockingbird', 'Harper Lee', '9780061120084', 3)
6. **Record a loan for John Doe:**
   * John borrows "The Great Gatsby" on 2024-09-01.
7. **Try to insert a loan for Jane Smith with a return date that is before the loan date (This should fail due to the check constraint).**
8. **Update the loan for John to set the return date to 2024-09-10.**
9. **Try to insert a book with a negative number of available copies (This should fail due to the check constraint).**

**Solution:**

-- Step 1: Create the `members` table

CREATE TABLE members (

member\_id INT AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(20) UNIQUE NOT NULL,

membership\_date DATE NOT NULL,

PRIMARY KEY (member\_id)

);

-- Step 2: Create the `books` table

CREATE TABLE books (

book\_id INT AUTO\_INCREMENT,

title VARCHAR(100) UNIQUE NOT NULL,

author VARCHAR(100) NOT NULL,

isbn VARCHAR(20) UNIQUE NOT NULL,

copies\_available INT NOT NULL CHECK (copies\_available >= 0),

PRIMARY KEY (book\_id)

);

-- Step 3: Create the `loans` table

CREATE TABLE loans (

loan\_id INT AUTO\_INCREMENT,

member\_id INT,

book\_id INT,

loan\_date DATE NOT NULL,

return\_date DATE,

PRIMARY KEY (loan\_id),

FOREIGN KEY (member\_id) REFERENCES members(member\_id),

FOREIGN KEY (book\_id) REFERENCES books(book\_id),

CHECK (return\_date IS NULL OR return\_date > loan\_date)

);

-- Step 4: Insert members

INSERT INTO members (name, email, phone, membership\_date)

VALUES

('John Doe', 'john@example.com', '1234567890', '2024-01-15'),

('Jane Smith', 'jane@example.com', '0987654321', '2024-02-10');

-- Step 5: Insert books

INSERT INTO books (title, author, isbn, copies\_available)

VALUES

('The Great Gatsby', 'F. Scott Fitzgerald', '9780743273565', 5),

('To Kill a Mockingbird', 'Harper Lee', '9780061120084', 3);

-- Step 6: Record a loan for John Doe

INSERT INTO loans (member\_id, book\_id, loan\_date)

VALUES

(1, 1, '2024-09-01'); -- John borrows "The Great Gatsby"

-- Step 7: Try to insert a loan for Jane Smith with an invalid return date (This will fail)

INSERT INTO loans (member\_id, book\_id, loan\_date, return\_date)

VALUES

(2, 2, '2024-09-05', '2024-09-01'); -- Invalid return date

-- Step 8: Update John's loan to set the return date

UPDATE loans

SET return\_date = '2024-09-10'

WHERE loan\_id = 1;

-- Step 9: Try to insert a book with negative available copies (This will fail)

INSERT INTO books (title, author, isbn, copies\_available)

VALUES

('Invalid Book', 'Unknown Author', '1234567890', -1); -- Invalid copies

**Exercise 15: School Management System**

**Exercise:**

1. **Create a table named students:**
   * student\_id: Integer, Auto Increment, Primary Key.
   * name: Variable character, Not Null.
   * dob: Date, Not Null.
   * email: Variable character, Unique, Not Null.
2. **Create a table named courses:**
   * course\_id: Integer, Auto Increment, Primary Key.
   * course\_name: Variable character, Unique, Not Null.
   * credits: Integer, Not Null, Check constraint to ensure greater than 0.
3. **Create a table named enrollments:**
   * enrollment\_id: Integer, Auto Increment, Primary Key.
   * student\_id: Integer, Foreign Key referencing students(student\_id).
   * course\_id: Integer, Foreign Key referencing courses(course\_id).
   * enrollment\_date: Date, Not Null.
4. **Insert the following students:**
   * ('Michael Johnson', '2000-05-15', 'michael.johnson@example.com')
   * ('Emily Davis', '1999-09-20', 'emily.davis@example.com')
5. **Insert the following courses:**
   * ('Mathematics', 3)
   * ('History', 4)
6. **Enroll Michael in Mathematics:**
   * Enrollment date is 2024-09-01.
7. **Try to enroll Emily in History again (This should fail due to the unique constraint).**
8. **Insert an invalid course with a negative number of credits (This should fail).**

**Solution:**

-- Step 1: Create the `students` table

CREATE TABLE students (

student\_id INT AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

dob DATE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

PRIMARY KEY (student\_id)

);

-- Step 2: Create the `courses` table

CREATE TABLE courses (

course\_id INT AUTO\_INCREMENT,

course\_name VARCHAR(100) UNIQUE NOT NULL,

credits INT NOT NULL CHECK (credits > 0),

PRIMARY KEY (course\_id)

);

-- Step 3: Create the `enrollments` table

CREATE TABLE enrollments (

enrollment\_id INT AUTO\_INCREMENT,

student\_id INT,

course\_id INT,

enrollment\_date DATE NOT NULL,

PRIMARY KEY (enrollment\_id),

FOREIGN KEY (student\_id) REFERENCES students(student\_id),

FOREIGN KEY (course\_id) REFERENCES courses(course\_id),

UNIQUE (student\_id, course\_id) -- Prevents duplicate enrollments

);

-- Step 4: Insert students

INSERT INTO students (name, dob, email)

VALUES

('Michael Johnson', '2000-05-15', 'michael.johnson@example.com'),

('Emily Davis', '1999-09-20', 'emily.davis@example.com');

-- Step 5: Insert courses

INSERT INTO courses (course\_name, credits)

VALUES

('Mathematics', 3),

('History', 4);

-- Step 6: Enroll Michael in Mathematics

INSERT INTO enrollments (student\_id, course\_id, enrollment\_date)

VALUES

(1, 1, '2024-09-01'); -- Michael enrolls in Mathematics

-- Step 7: Try to enroll Emily in History again (This will fail)

INSERT INTO enrollments (student\_id, course\_id, enrollment\_date)

VALUES

(2, 2, '2024-09-02'); -- Attempt to re-enroll Emily in History

-- Step 8: Try to insert an invalid course with negative credits (This will fail)

INSERT INTO courses (course\_name, credits)

VALUES

('Invalid Course', -2); -- Invalid credits

**Exercise 16: Food Delivery System**

**Exercise:**

1. **Create a table named restaurants:**
   * restaurant\_id: Integer, Auto Increment, Primary Key.
   * name: Variable character, Unique, Not Null.
   * address: Variable character, Not Null.
   * rating: Decimal(2, 1), Check constraint to ensure rating is between 0 and 5.
2. **Create a table named menu\_items:**
   * item\_id: Integer, Auto Increment, Primary Key.
   * restaurant\_id: Integer, Foreign Key referencing restaurants(restaurant\_id).
   * item\_name: Variable character, Not Null.
   * price: Decimal(10, 2), Not Null, Check constraint to ensure greater than 0.
3. **Create a table named orders:**
   * order\_id: Integer, Auto Increment, Primary Key.
   * item\_id: Integer, Foreign Key referencing menu\_items(item\_id).
   * order\_date: Date, Not Null.
   * quantity: Integer, Not Null, Check constraint to ensure greater than 0.
4. **Insert the following restaurants:**
   * ('Pizza Place', '123 Pizza Rd, Springfield', 4.5)
   * ('Sushi Spot', '456 Sushi Ave, Springfield', 4.8)
5. **Insert the following menu items:**
   * ('Margherita Pizza', 10.00)
   * ('California Roll', 15.00)
6. **Place an order for Margherita Pizza:**
   * Order date is 2024-09-05 and quantity is 2.
7. **Try to insert a menu item with a negative price (This should fail).**
8. **Update the rating of a restaurant to exceed 5 (This should fail).**

**Solution:**

-- Step 1: Create the `restaurants` table

CREATE TABLE restaurants (

restaurant\_id INT AUTO\_INCREMENT,

name VARCHAR(100) UNIQUE NOT NULL,

address VARCHAR(255) NOT NULL,

rating DECIMAL(2, 1) CHECK (rating >= 0 AND rating <= 5),

PRIMARY KEY (restaurant\_id)

);

-- Step 2: Create the `menu\_items` table

CREATE TABLE menu\_items (

item\_id INT AUTO\_INCREMENT,

restaurant\_id INT,

item\_name VARCHAR(100) NOT NULL,

price DECIMAL(10, 2) NOT NULL CHECK (price > 0),

PRIMARY KEY (item\_id),

FOREIGN KEY (restaurant\_id) REFERENCES restaurants(restaurant\_id)

);

-- Step 3: Create the `orders` table

CREATE TABLE orders (

order\_id INT AUTO\_INCREMENT,

item\_id INT,

order\_date DATE NOT NULL,

quantity INT NOT NULL CHECK (quantity > 0),

PRIMARY KEY (order\_id),

FOREIGN KEY (item\_id) REFERENCES menu\_items(item\_id)

);

-- Step 4: Insert restaurants

INSERT INTO restaurants (name, address, rating)

VALUES

('Pizza Place', '123 Pizza Rd, Springfield', 4.5),

('Sushi Spot', '456 Sushi Ave, Springfield', 4.8);

-- Step 5: Insert menu items

INSERT INTO menu\_items (restaurant\_id, item\_name, price)

VALUES

(1, 'Margherita Pizza', 10.00),

(2, 'California Roll', 15.00);

-- Step 6: Place an order for Margherita Pizza

INSERT INTO orders (item\_id, order\_date, quantity)

VALUES

(1, '2024-09-05', 2); -- Order for 2 Margherita Pizzas

-- Step 7: Try to insert a menu item with negative price (This will fail)

INSERT INTO menu\_items (restaurant\_id, item\_name, price)

VALUES

(1, 'Invalid Item', -5.00); -- Invalid price

-- Step 8: Try to update the rating of a restaurant to exceed 5 (This will fail)

UPDATE restaurants

SET rating = 6.0

WHERE restaurant\_id = 1; -- Invalid rating

**Exercise 17: Event Management System**

**Exercise:**

1. **Create a table named attendees:**
   * attendee\_id: Integer, Auto Increment, Primary Key.
   * name: Variable character, Not Null.
   * email: Variable character, Unique, Not Null.
   * phone: Variable character, Unique, Not Null.
   * registration\_date: Date, Not Null.
2. **Create a table named events:**
   * event\_id: Integer, Auto Increment, Primary Key.
   * event\_name: Variable character, Unique, Not Null.
   * event\_date: Date, Not Null.
   * max\_attendees: Integer, Not Null, Check constraint to ensure greater than 0.
3. **Create a table named registrations:**
   * registration\_id: Integer, Auto Increment, Primary Key.
   * attendee\_id: Integer, Foreign Key referencing attendees(attendee\_id).
   * event\_id: Integer, Foreign Key referencing events(event\_id).
   * registration\_time: Timestamp, Default to current timestamp.
   * Ensure that each attendee can register for the same event only once.
4. **Insert the following attendees:**
   * ('Alice Brown', 'alice.brown@example.com', '3216549870', '2024-08-15')
   * ('Bob Smith', 'bob.smith@example.com', '9876543210', '2024-08-20')
5. **Insert the following events:**
   * ('Tech Conference', '2024-09-30', 100)
   * ('Art Exhibition', '2024-10-15', 50)
6. **Register Alice for the Tech Conference.**
7. **Try to register Bob for the Tech Conference again (This should fail due to the unique constraint).**
8. **Insert an attendee with an invalid email format (This should fail).**

**Solution:**

-- Step 1: Create the `attendees` table

CREATE TABLE attendees (

attendee\_id INT AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(20) UNIQUE NOT NULL,

registration\_date DATE NOT NULL,

PRIMARY KEY (attendee\_id)

);

-- Step 2: Create the `events` table

CREATE TABLE events (

event\_id INT AUTO\_INCREMENT,

event\_name VARCHAR(100) UNIQUE NOT NULL,

event\_date DATE NOT NULL,

max\_attendees INT NOT NULL CHECK (max\_attendees > 0),

PRIMARY KEY (event\_id)

);

-- Step 3: Create the `registrations` table

CREATE TABLE registrations (

registration\_id INT AUTO\_INCREMENT,

attendee\_id INT,

event\_id INT,

registration\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (registration\_id),

FOREIGN KEY (attendee\_id) REFERENCES attendees(attendee\_id),

FOREIGN KEY (event\_id) REFERENCES events(event\_id),

UNIQUE (attendee\_id, event\_id) -- Prevents duplicate registrations

);

-- Step 4: Insert attendees

INSERT INTO attendees (name, email, phone, registration\_date)

VALUES

('Alice Brown', 'alice.brown@example.com', '3216549870', '2024-08-15'),

('Bob Smith', 'bob.smith@example.com', '9876543210', '2024-08-20');

-- Step 5: Insert events

INSERT INTO events (event\_name, event\_date, max\_attendees)

VALUES

('Tech Conference', '2024-09-30', 100),

('Art Exhibition', '2024-10-15', 50);

-- Step 6: Register Alice for the Tech Conference

INSERT INTO registrations (attendee\_id, event\_id)

VALUES

(1, 1); -- Alice registers for the Tech Conference

-- Step 7: Try to register Bob for the Tech Conference again (This will fail)

INSERT INTO registrations (attendee\_id, event\_id)

VALUES

(2, 1); -- Attempt to re-register Bob for the Tech Conference

-- Step 8: Try to insert an attendee with an invalid email (This will fail)

INSERT INTO attendees (name, email, phone, registration\_date)

VALUES

('Invalid Email', 'invalidemail.com', '1234567890', '2024-08-25'); -- Invalid email format

**Exercise 18: Online Course Platform**

**Exercise:**

1. **Create a table named instructors:**
   * instructor\_id: Integer, Auto Increment, Primary Key.
   * name: Variable character, Not Null.
   * email: Variable character, Unique, Not Null.
   * experience\_years: Integer, Not Null, Check constraint to ensure greater than or equal to 0.
2. **Create a table named courses:**
   * course\_id: Integer, Auto Increment, Primary Key.
   * title: Variable character, Unique, Not Null.
   * instructor\_id: Integer, Foreign Key referencing instructors(instructor\_id).
   * description: Text, Not Null.
   * price: Decimal(10, 2), Not Null, Check constraint to ensure greater than 0.
3. **Create a table named enrollments:**
   * enrollment\_id: Integer, Auto Increment, Primary Key.
   * course\_id: Integer, Foreign Key referencing courses(course\_id).
   * student\_id: Integer, Foreign Key referencing students(student\_id).
   * enrollment\_date: Date, Not Null.
   * Ensure that each student can enroll in the same course only once.
4. **Insert the following instructors:**
   * ('Dr. Emily White', 'emily.white@example.com', 5)
   * ('Prof. John Green', 'john.green@example.com', 10)
5. **Insert the following courses:**
   * ('Data Science 101', 1, 'Introduction to Data Science', 199.99)
   * ('Web Development Bootcamp', 2, 'Learn to build websites', 299.99)
6. **Enroll a student (assume student\_id = 1) in Data Science 101.**
7. **Try to enroll the same student in Data Science 101 again (This should fail due to the unique constraint).**
8. **Insert an instructor with negative experience years (This should fail).**

**Solution:**

-- Step 1: Create the `instructors` table

CREATE TABLE instructors (

instructor\_id INT AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

experience\_years INT NOT NULL CHECK (experience\_years >= 0),

PRIMARY KEY (instructor\_id)

);

-- Step 2: Create the `courses` table

CREATE TABLE courses (

course\_id INT AUTO\_INCREMENT,

title VARCHAR(100) UNIQUE NOT NULL,

instructor\_id INT,

description TEXT NOT NULL,

price DECIMAL(10, 2) NOT NULL CHECK (price > 0),

PRIMARY KEY (course\_id),

FOREIGN KEY (instructor\_id) REFERENCES instructors(instructor\_id)

);

-- Step 3: Create the `enrollments` table

CREATE TABLE enrollments (

enrollment\_id INT AUTO\_INCREMENT,

course\_id INT,

student\_id INT,

enrollment\_date DATE NOT NULL,

PRIMARY KEY (enrollment\_id),

FOREIGN KEY (course\_id) REFERENCES courses(course\_id),

FOREIGN KEY (student\_id) REFERENCES students(student\_id),

UNIQUE (course\_id, student\_id) -- Prevents duplicate enrollments

);

-- Step 4: Insert instructors

INSERT INTO instructors (name, email, experience\_years)

VALUES

('Dr. Emily White', 'emily.white@example.com', 5),

('Prof. John Green', 'john.green@example.com', 10);

-- Step 5: Insert courses

INSERT INTO courses (title, instructor\_id, description, price)

VALUES

('Data Science 101', 1, 'Introduction to Data Science', 199.99),

('Web Development Bootcamp', 2, 'Learn to build websites', 299.99);

-- Step 6: Enroll a student in Data Science 101 (assume student\_id = 1)

INSERT INTO enrollments (course\_id, student\_id, enrollment\_date)

VALUES

(1, 1, '2024-09-01'); -- Student enrolls in Data Science 101

-- Step 7: Try to enroll the same student in Data Science 101 again (This will fail)

INSERT INTO enrollments (course\_id, student\_id, enrollment\_date)

VALUES

(1, 1, '2024-09-05'); -- Attempt to re-enroll the same student

-- Step 8: Try to insert an instructor with negative experience years (This will fail)

INSERT INTO instructors (name, email, experience\_years)

VALUES

('Invalid Instructor', 'invalid@example.com', -3); -- Invalid experience

**Exercise 19: Fitness Tracking App**

**Exercise:**

1. **Create a table named users:**
   * user\_id: Integer, Auto Increment, Primary Key.
   * username: Variable character, Unique, Not Null.
   * email: Variable character, Unique, Not Null.
   * password: Variable character, Not Null.
2. **Create a table named activities:**
   * activity\_id: Integer, Auto Increment, Primary Key.
   * activity\_name: Variable character, Unique, Not Null.
   * calories\_burned: Integer, Not Null, Check constraint to ensure greater than 0.
3. **Create a table named user\_activities:**
   * user\_activity\_id: Integer, Auto Increment, Primary Key.
   * user\_id: Integer, Foreign Key referencing users(user\_id).
   * activity\_id: Integer, Foreign Key referencing activities(activity\_id).
   * date: Date, Not Null.
   * Ensure that a user can log the same activity on the same date only once.
4. **Insert the following users:**
   * ('john\_doe', 'john.doe@example.com', 'password123')
   * ('jane\_doe', 'jane.doe@example.com', 'password456')
5. **Insert the following activities:**
   * ('Running', 500)
   * ('Cycling', 300)
6. **Log an activity for John Doe (assume user\_id = 1) for Running on 2024-09-01.**
7. **Try to log the same activity for John Doe on the same date (This should fail).**
8. **Insert an activity with negative calories burned (This should fail).**

**Solution:**

-- Step 1: Create the `users` table

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT,

username VARCHAR(100) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password VARCHAR(100) NOT NULL,

PRIMARY KEY (user\_id)

);

-- Step 2: Create the `activities` table

CREATE TABLE activities (

activity\_id INT AUTO\_INCREMENT,

activity\_name VARCHAR(100) UNIQUE NOT NULL,

calories\_burned INT NOT NULL CHECK (calories\_burned > 0),

PRIMARY KEY (activity\_id)

);

-- Step 3: Create the `user\_activities` table

CREATE TABLE user\_activities (

user\_activity\_id INT AUTO\_INCREMENT,

user\_id INT,

activity\_id INT,

date DATE NOT NULL,

PRIMARY KEY (user\_activity\_id),

FOREIGN KEY (user\_id) REFERENCES users(user\_id),

FOREIGN KEY (activity\_id) REFERENCES activities(activity\_id),

UNIQUE (user\_id, activity\_id, date) -- Prevents duplicate activity logs

);

-- Step 4: Insert users

INSERT INTO users (username, email, password)

VALUES

('john\_doe', 'john.doe@example.com', 'password123'),

('jane\_doe', 'jane.doe@example.com', 'password456');

-- Step 5: Insert activities

INSERT INTO activities (activity\_name, calories\_burned)

VALUES

('Running', 500),

('Cycling', 300);

-- Step 6: Log an activity for John Doe (assume user\_id = 1) for Running on 2024-09-01

INSERT INTO user\_activities (user\_id, activity\_id, date)

VALUES

(1, 1, '2024-09-01'); -- John logs Running

-- Step 7: Try to log the same activity for John Doe on the same date (This will fail)

INSERT INTO user\_activities (user\_id, activity\_id, date)

VALUES

(1, 1, '2024-09-01'); -- Attempt to re-log the same activity

-- Step 8: Try to insert an activity with negative calories burned (This will fail)

INSERT INTO activities (activity\_name, calories\_burned)

VALUES

('Invalid Activity', -100); -- Invalid calories

**Exercise 20: Hotel Reservation System**

**Exercise:**

1. **Create a table named hotels:**
   * hotel\_id: Integer, Auto Increment, Primary Key.
   * hotel\_name: Variable character, Unique, Not Null.
   * location: Variable character, Not Null.
   * rating: Decimal(2, 1), Check constraint to ensure between 0 and 5.
2. **Create a table named rooms:**
   * room\_id: Integer, Auto Increment, Primary Key.
   * hotel\_id: Integer, Foreign Key referencing hotels(hotel\_id).
   * room\_type: Variable character, Not Null.
   * price\_per\_night: Decimal(10, 2), Not Null, Check constraint to ensure greater than 0.
3. **Create a table named reservations:**
   * reservation\_id: Integer, Auto Increment, Primary Key.
   * room\_id: Integer, Foreign Key referencing rooms(room\_id).
   * customer\_name: Variable character, Not Null.
   * check\_in: Date, Not Null.
   * check\_out: Date, Not Null.
   * Ensure that check-out is after check-in.
4. **Insert the following hotels:**
   * ('Grand Hotel', 'New York', 4.5)
   * ('Beach Resort', 'Miami', 4.8)
5. **Insert the following rooms:**
   * (1, 'Deluxe Suite', 300.00)
   * (2, 'Ocean View Room', 250.00)
6. **Make a reservation for a Deluxe Suite at the Grand Hotel from 2024-09-01 to 2024-09-05.**
7. **Try to make a reservation for the same room with overlapping dates (This should fail).**
8. **Insert a room with a negative price per night (This should fail).**

**Solution:**

-- Step 1: Create the `hotels` table

CREATE TABLE hotels (

hotel\_id INT AUTO\_INCREMENT,

hotel\_name VARCHAR(100) UNIQUE NOT NULL,

location VARCHAR(100) NOT NULL,

rating DECIMAL(2, 1) CHECK (rating >= 0 AND rating <= 5),

PRIMARY KEY (hotel\_id)

);

-- Step 2: Create the `rooms` table

CREATE TABLE rooms (

room\_id INT AUTO\_INCREMENT,

hotel\_id INT,

room\_type VARCHAR(100) NOT NULL,

price\_per\_night DECIMAL(10, 2) NOT NULL CHECK (price\_per\_night > 0),

PRIMARY KEY (room\_id),

FOREIGN KEY (hotel\_id) REFERENCES hotels(hotel\_id)

);

-- Step 3: Create the `reservations` table

CREATE TABLE reservations (

reservation\_id INT AUTO\_INCREMENT,

room\_id INT,

customer\_name VARCHAR(100) NOT NULL,

check\_in DATE NOT NULL,

check\_out DATE NOT NULL,

PRIMARY KEY (reservation\_id),

FOREIGN KEY (room\_id) REFERENCES rooms(room\_id),

CHECK (check\_out > check\_in) -- Ensures check-out is after check-in

);

-- Step 4: Insert hotels

INSERT INTO hotels (hotel\_name, location, rating)

VALUES

('Grand Hotel', 'New York', 4.5),

('Beach Resort', 'Miami', 4.8);

-- Step 5: Insert rooms

INSERT INTO rooms (hotel\_id, room\_type, price\_per\_night)

VALUES

(1, 'Deluxe Suite', 300.00),

(2, 'Ocean View Room', 250.00);

-- Step 6: Make a reservation for a Deluxe Suite at the Grand Hotel

INSERT INTO reservations (room\_id, customer\_name, check\_in, check\_out)

VALUES

(1, 'John Doe', '2024-09-01', '2024-09-05'); -- Reservation for John

-- Step 7: Try to make a reservation for the same room with overlapping dates (This will fail)

INSERT INTO reservations (room\_id, customer\_name, check\_in, check\_out)

VALUES

(1, 'Jane Doe', '2024-09-03', '2024-09-06'); -- Overlapping reservation

-- Step 8: Try to insert a room with negative price per night (This will fail)

INSERT INTO rooms (hotel\_id, room\_type, price\_per\_night)

VALUES

(1, 'Invalid Room', -100.00); -- Invalid price

**Exercise 22: E-commerce System**

**Exercise:**

1. **Create a table named customers:**
   * customer\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * email: Variable character, Unique, Not Null.
   * phone: Variable character, Not Null.
   * address: Text.
2. **Create a table named categories:**
   * category\_id: Integer, Auto Increment, Primary Key.
   * category\_name: Variable character, Unique, Not Null.
3. **Create a table named products:**
   * product\_id: Integer, Auto Increment, Primary Key.
   * name: Variable character, Not Null.
   * description: Text.
   * price: Decimal(10, 2), Not Null, Check constraint to ensure greater than 0.
   * stock\_quantity: Integer, Not Null, Check constraint to ensure greater than or equal to 0.
   * category\_id: Integer, Foreign Key referencing categories(category\_id).
4. **Create a table named orders:**
   * order\_id: Integer, Auto Increment, Primary Key.
   * customer\_id: Integer, Foreign Key referencing customers(customer\_id).
   * order\_date: Date, Not Null.
   * total\_amount: Decimal(10, 2), Not Null.
5. **Create a table named order\_items:**
   * order\_item\_id: Integer, Auto Increment, Primary Key.
   * order\_id: Integer, Foreign Key referencing orders(order\_id).
   * product\_id: Integer, Foreign Key referencing products(product\_id).
   * quantity: Integer, Not Null.
6. **Create a table named payments:**
   * payment\_id: Integer, Auto Increment, Primary Key.
   * order\_id: Integer, Foreign Key referencing orders(order\_id).
   * payment\_date: Date, Not Null.
   * amount: Decimal(10, 2), Not Null.
   * payment\_method: Variable character, Not Null.
7. **Create a table named shipping:**
   * shipping\_id: Integer, Auto Increment, Primary Key.
   * order\_id: Integer, Foreign Key referencing orders(order\_id).
   * shipping\_address: Text, Not Null.
   * shipping\_date: Date.
   * delivery\_date: Date.
8. **Create a table named carts:**
   * cart\_id: Integer, Auto Increment, Primary Key.
   * customer\_id: Integer, Foreign Key referencing customers(customer\_id).
9. **Create a table named cart\_items:**
   * cart\_item\_id: Integer, Auto Increment, Primary Key.
   * cart\_id: Integer, Foreign Key referencing carts(cart\_id).
   * product\_id: Integer, Foreign Key referencing products(product\_id).
   * quantity: Integer, Not Null.
10. **Create a table named reviews:**
    * review\_id: Integer, Auto Increment, Primary Key.
    * product\_id: Integer, Foreign Key referencing products(product\_id).
    * customer\_id: Integer, Foreign Key referencing customers(customer\_id).
    * rating: Integer, Check constraint between 1 and 5.
    * comment: Text.

**Solution:**

-- Step 1: Create the `customers` table

CREATE TABLE customers (

customer\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(15) NOT NULL,

address TEXT,

PRIMARY KEY (customer\_id)

);

-- Step 2: Create the `categories` table

CREATE TABLE categories (

category\_id INT AUTO\_INCREMENT,

category\_name VARCHAR(100) UNIQUE NOT NULL,

PRIMARY KEY (category\_id)

);

-- Step 3: Create the `products` table

CREATE TABLE products (

product\_id INT AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

description TEXT,

price DECIMAL(10, 2) NOT NULL CHECK (price > 0),

stock\_quantity INT NOT NULL CHECK (stock\_quantity >= 0),

category\_id INT,

PRIMARY KEY (product\_id),

FOREIGN KEY (category\_id) REFERENCES categories(category\_id)

);

-- Step 4: Create the `orders` table

CREATE TABLE orders (

order\_id INT AUTO\_INCREMENT,

customer\_id INT,

order\_date DATE NOT NULL,

total\_amount DECIMAL(10, 2) NOT NULL,

PRIMARY KEY (order\_id),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

-- Step 5: Create the `order\_items` table

CREATE TABLE order\_items (

order\_item\_id INT AUTO\_INCREMENT,

order\_id INT,

product\_id INT,

quantity INT NOT NULL,

PRIMARY KEY (order\_item\_id),

FOREIGN KEY (order\_id) REFERENCES orders(order\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

-- Step 6: Create the `payments` table

CREATE TABLE payments (

payment\_id INT AUTO\_INCREMENT,

order\_id INT,

payment\_date DATE NOT NULL,

amount DECIMAL(10, 2) NOT NULL,

payment\_method VARCHAR(50) NOT NULL,

PRIMARY KEY (payment\_id),

FOREIGN KEY (order\_id) REFERENCES orders(order\_id)

);

-- Step 7: Create the `shipping` table

CREATE TABLE shipping (

shipping\_id INT AUTO\_INCREMENT,

order\_id INT,

shipping\_address TEXT NOT NULL,

shipping\_date DATE,

delivery\_date DATE,

PRIMARY KEY (shipping\_id),

FOREIGN KEY (order\_id) REFERENCES orders(order\_id)

);

-- Step 8: Create the `carts` table

CREATE TABLE carts (

cart\_id INT AUTO\_INCREMENT,

customer\_id INT,

PRIMARY KEY (cart\_id),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

-- Step 9: Create the `cart\_items` table

CREATE TABLE cart\_items (

cart\_item\_id INT AUTO\_INCREMENT,

cart\_id INT,

product\_id INT,

quantity INT NOT NULL,

PRIMARY KEY (cart\_item\_id),

FOREIGN KEY (cart\_id) REFERENCES carts(cart\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

-- Step 10: Create the `reviews` table

CREATE TABLE reviews (

review\_id INT AUTO\_INCREMENT,

product\_id INT,

customer\_id INT,

rating INT CHECK (rating >= 1 AND rating <= 5),

comment TEXT,

PRIMARY KEY (review\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

**Exercise 23: Hospital Management System**

**Exercise:**

1. **Create a table named patients:**
   * patient\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * dob: Date, Not Null.
   * gender: Enum ('Male', 'Female', 'Other'), Not Null.
   * phone: Variable character, Unique, Not Null.
   * email: Variable character, Unique, Not Null.
   * address: Text.
2. **Create a table named doctors:**
   * doctor\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * specialty: Variable character, Not Null.
   * phone: Variable character, Unique, Not Null.
   * email: Variable character, Unique, Not Null.
3. **Create a table named departments:**
   * department\_id: Integer, Auto Increment, Primary Key.
   * department\_name: Variable character, Unique, Not Null.
4. **Create a table named appointments:**
   * appointment\_id: Integer, Auto Increment, Primary Key.
   * patient\_id: Integer, Foreign Key referencing patients(patient\_id).
   * doctor\_id: Integer, Foreign Key referencing doctors(doctor\_id).
   * appointment\_date: DateTime, Not Null.
   * status: Enum ('Scheduled', 'Completed', 'Cancelled'), Not Null.
5. **Create a table named medical\_records:**
   * record\_id: Integer, Auto Increment, Primary Key.
   * patient\_id: Integer, Foreign Key referencing patients(patient\_id).
   * doctor\_id: Integer, Foreign Key referencing doctors(doctor\_id).
   * diagnosis: Text, Not Null.
   * treatment: Text, Not Null.
   * record\_date: Date, Not Null.
6. **Create a table named medications:**
   * medication\_id: Integer, Auto Increment, Primary Key.
   * name: Variable character, Unique, Not Null.
   * description: Text.
7. **Create a table named prescriptions:**
   * prescription\_id: Integer, Auto Increment, Primary Key.
   * record\_id: Integer, Foreign Key referencing medical\_records(record\_id).
   * medication\_id: Integer, Foreign Key referencing medications(medication\_id).
   * dosage: Variable character, Not Null.
   * frequency: Variable character, Not Null.
8. **Create a table named bills:**
   * bill\_id: Integer, Auto Increment, Primary Key.
   * patient\_id: Integer, Foreign Key referencing patients(patient\_id).
   * amount: Decimal(10, 2), Not Null.
   * payment\_status: Enum ('Paid', 'Pending', 'Cancelled'), Not Null.
   * billing\_date: Date, Not Null.
9. **Create a table named staff:**
   * staff\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * position: Variable character, Not Null.
   * department\_id: Integer, Foreign Key referencing departments(department\_id).
10. **Create a table named insurance:**
    * insurance\_id: Integer, Auto Increment, Primary Key.
    * patient\_id: Integer, Foreign Key referencing patients(patient\_id).
    * provider: Variable character, Not Null.
    * policy\_number: Variable character, Unique, Not Null.
    * coverage\_amount: Decimal(10, 2), Not Null.

**Solution:**

-- Step 1: Create the `patients` table

CREATE TABLE patients (

patient\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

dob DATE NOT NULL,

gender ENUM('Male', 'Female', 'Other') NOT NULL,

phone VARCHAR(15) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

address TEXT,

PRIMARY KEY (patient\_id)

);

-- Step 2: Create the `doctors` table

CREATE TABLE doctors (

doctor\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

specialty VARCHAR(100) NOT NULL,

phone VARCHAR(15) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

PRIMARY KEY (doctor\_id)

);

-- Step 3: Create the `departments` table

CREATE TABLE departments (

department\_id INT AUTO\_INCREMENT,

department\_name VARCHAR(100) UNIQUE NOT NULL,

PRIMARY KEY (department\_id)

);

-- Step 4: Create the `appointments` table

CREATE TABLE appointments (

appointment\_id INT AUTO\_INCREMENT,

patient\_id INT,

doctor\_id INT,

appointment\_date DATETIME NOT NULL,

status ENUM('Scheduled', 'Completed', 'Cancelled') NOT NULL,

PRIMARY KEY (appointment\_id),

FOREIGN KEY (patient\_id) REFERENCES patients(patient\_id),

FOREIGN KEY (doctor\_id) REFERENCES doctors(doctor\_id)

);

-- Step 5: Create the `medical\_records` table

CREATE TABLE medical\_records (

record\_id INT AUTO\_INCREMENT,

patient\_id INT,

doctor\_id INT,

diagnosis TEXT NOT NULL,

treatment TEXT NOT NULL,

record\_date DATE NOT NULL,

PRIMARY KEY (record\_id),

FOREIGN KEY (patient\_id) REFERENCES patients(patient\_id),

FOREIGN KEY (doctor\_id) REFERENCES doctors(doctor\_id)

);

-- Step 6: Create the `medications` table

CREATE TABLE medications (

medication\_id INT AUTO\_INCREMENT,

name VARCHAR(100) UNIQUE NOT NULL,

description TEXT,

PRIMARY KEY (medication\_id)

);

-- Step 7: Create the `prescriptions` table

CREATE TABLE prescriptions (

prescription\_id INT AUTO\_INCREMENT,

record\_id INT,

medication\_id INT,

dosage VARCHAR(50) NOT NULL,

frequency VARCHAR(50) NOT NULL,

PRIMARY KEY (prescription\_id),

FOREIGN KEY (record\_id) REFERENCES medical\_records(record\_id),

FOREIGN KEY (medication\_id) REFERENCES medications(medication\_id)

);

-- Step 8: Create the `bills` table

CREATE TABLE bills (

bill\_id INT AUTO\_INCREMENT,

patient\_id INT,

amount DECIMAL(10, 2) NOT NULL,

payment\_status ENUM('Paid', 'Pending', 'Cancelled') NOT NULL,

billing\_date DATE NOT NULL,

PRIMARY KEY (bill\_id),

FOREIGN KEY (patient\_id) REFERENCES patients(patient\_id)

);

-- Step 9: Create the `staff` table

CREATE TABLE staff (

staff\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

position VARCHAR(100) NOT NULL,

department\_id INT,

PRIMARY KEY (staff\_id),

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

-- Step 10: Create the `insurance` table

CREATE TABLE insurance (

insurance\_id INT AUTO\_INCREMENT,

patient\_id INT,

provider VARCHAR(100) NOT NULL,

policy\_number VARCHAR(50) UNIQUE NOT NULL,

coverage\_amount DECIMAL(10, 2) NOT NULL,

PRIMARY KEY (insurance\_id),

FOREIGN KEY (patient\_id) REFERENCES patients(patient\_id)

);

**Exercise 24: Library Management System**

**Exercise:**

1. **Create a table named authors:**
   * author\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * date\_of\_birth: Date, Not Null.
   * nationality: Variable character, Not Null.
2. **Create a table named publishers:**
   * publisher\_id: Integer, Auto Increment, Primary Key.
   * name: Variable character, Unique, Not Null.
   * address: Text, Not Null.
   * contact\_number: Variable character, Unique, Not Null.
3. **Create a table named categories:**
   * category\_id: Integer, Auto Increment, Primary Key.
   * category\_name: Variable character, Unique, Not Null.
4. **Create a table named books:**
   * book\_id: Integer, Auto Increment, Primary Key.
   * title: Variable character, Not Null.
   * author\_id: Integer, Foreign Key referencing authors(author\_id).
   * publisher\_id: Integer, Foreign Key referencing publishers(publisher\_id).
   * category\_id: Integer, Foreign Key referencing categories(category\_id).
   * isbn: Variable character, Unique, Not Null.
   * publication\_date: Date, Not Null.
   * copies\_available: Integer, Not Null CHECK (copies\_available >= 0).
5. **Create a table named members:**
   * member\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * email: Variable character, Unique, Not Null.
   * phone: Variable character, Unique, Not Null.
   * join\_date: Date, Not Null.
6. **Create a table named loans:**
   * loan\_id: Integer, Auto Increment, Primary Key.
   * book\_id: Integer, Foreign Key referencing books(book\_id).
   * member\_id: Integer, Foreign Key referencing members(member\_id).
   * loan\_date: Date, Not Null.
   * return\_date: Date, Nullable.
7. **Create a table named fines:**
   * fine\_id: Integer, Auto Increment, Primary Key.
   * loan\_id: Integer, Foreign Key referencing loans(loan\_id).
   * amount: Decimal(10, 2), Not Null CHECK (amount > 0).
   * paid: Boolean, Not Null DEFAULT FALSE.
8. **Create a table named reservations:**
   * reservation\_id: Integer, Auto Increment, Primary Key.
   * book\_id: Integer, Foreign Key referencing books(book\_id).
   * member\_id: Integer, Foreign Key referencing members(member\_id).
   * reservation\_date: Date, Not Null.
   * status: Enum ('Active', 'Cancelled', 'Fulfilled'), Not Null.
9. **Create a table named reviews:**
   * review\_id: Integer, Auto Increment, Primary Key.
   * book\_id: Integer, Foreign Key referencing books(book\_id).
   * member\_id: Integer, Foreign Key referencing members(member\_id).
   * rating: Integer CHECK (rating >= 1 AND rating <= 5), Not Null.
   * comment: Text, Nullable.
10. **Create a table named transactions:**
    * transaction\_id: Integer, Auto Increment, Primary Key.
    * loan\_id: Integer, Foreign Key referencing loans(loan\_id).
    * transaction\_date: Date, Not Null.
    * amount: Decimal(10, 2), Not Null CHECK (amount > 0).

**Solution:**

-- Step 1: Create the `authors` table

CREATE TABLE authors (

author\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

date\_of\_birth DATE NOT NULL,

nationality VARCHAR(50) NOT NULL,

PRIMARY KEY (author\_id)

);

-- Step 2: Create the `publishers` table

CREATE TABLE publishers (

publisher\_id INT AUTO\_INCREMENT,

name VARCHAR(100) UNIQUE NOT NULL,

address TEXT NOT NULL,

contact\_number VARCHAR(15) UNIQUE NOT NULL,

PRIMARY KEY (publisher\_id)

);

-- Step 3: Create the `categories` table

CREATE TABLE categories (

category\_id INT AUTO\_INCREMENT,

category\_name VARCHAR(100) UNIQUE NOT NULL,

PRIMARY KEY (category\_id)

);

-- Step 4: Create the `books` table

CREATE TABLE books (

book\_id INT AUTO\_INCREMENT,

title VARCHAR(200) NOT NULL,

author\_id INT,

publisher\_id INT,

category\_id INT,

isbn VARCHAR(20) UNIQUE NOT NULL,

publication\_date DATE NOT NULL,

copies\_available INT NOT NULL CHECK (copies\_available >= 0),

PRIMARY KEY (book\_id),

FOREIGN KEY (author\_id) REFERENCES authors(author\_id),

FOREIGN KEY (publisher\_id) REFERENCES publishers(publisher\_id),

FOREIGN KEY (category\_id) REFERENCES categories(category\_id)

);

-- Step 5: Create the `members` table

CREATE TABLE members (

member\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(15) UNIQUE NOT NULL,

join\_date DATE NOT NULL,

PRIMARY KEY (member\_id)

);

-- Step 6: Create the `loans` table

CREATE TABLE loans (

loan\_id INT AUTO\_INCREMENT,

book\_id INT,

member\_id INT,

loan\_date DATE NOT NULL,

return\_date DATE,

PRIMARY KEY (loan\_id),

FOREIGN KEY (book\_id) REFERENCES books(book\_id),

FOREIGN KEY (member\_id) REFERENCES members(member\_id)

);

-- Step 7: Create the `fines` table

CREATE TABLE fines (

fine\_id INT AUTO\_INCREMENT,

loan\_id INT,

amount DECIMAL(10, 2) NOT NULL CHECK (amount > 0),

paid BOOLEAN NOT NULL DEFAULT FALSE,

PRIMARY KEY (fine\_id),

FOREIGN KEY (loan\_id) REFERENCES loans(loan\_id)

);

-- Step 8: Create the `reservations` table

CREATE TABLE reservations (

reservation\_id INT AUTO\_INCREMENT,

book\_id INT,

member\_id INT,

reservation\_date DATE NOT NULL,

status ENUM('Active', 'Cancelled', 'Fulfilled') NOT NULL,

PRIMARY KEY (reservation\_id),

FOREIGN KEY (book\_id) REFERENCES books(book\_id),

FOREIGN KEY (member\_id) REFERENCES members(member\_id)

);

-- Step 9: Create the `reviews` table

CREATE TABLE reviews (

review\_id INT AUTO\_INCREMENT,

book\_id INT,

member\_id INT,

rating INT CHECK (rating >= 1 AND rating <= 5) NOT NULL,

comment TEXT,

PRIMARY KEY (review\_id),

FOREIGN KEY (book\_id) REFERENCES books(book\_id),

FOREIGN KEY (member\_id) REFERENCES members(member\_id)

);

-- Step 10: Create the `transactions` table

CREATE TABLE transactions (

transaction\_id INT AUTO\_INCREMENT,

loan\_id INT,

transaction\_date DATE NOT NULL,

amount DECIMAL(10, 2) NOT NULL CHECK (amount > 0),

PRIMARY KEY (transaction\_id),

FOREIGN KEY (loan\_id) REFERENCES loans(loan\_id)

);

**Exercise 25: University Management System**

**Exercise:**

1. **Create a table named departments:**
   * department\_id: Integer, Auto Increment, Primary Key.
   * department\_name: Variable character, Unique, Not Null.
   * location: Variable character, Not Null.
2. **Create a table named courses:**
   * course\_id: Integer, Auto Increment, Primary Key.
   * course\_name: Variable character, Unique, Not Null.
   * department\_id: Integer, Foreign Key referencing departments(department\_id).
   * credits: Integer, Not Null CHECK (credits > 0).
3. **Create a table named students:**
   * student\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * email: Variable character, Unique, Not Null.
   * date\_of\_birth: Date, Not Null.
   * enrollment\_date: Date, Not Null.
4. **Create a table named professors:**
   * professor\_id: Integer, Auto Increment, Primary Key.
   * first\_name: Variable character, Not Null.
   * last\_name: Variable character, Not Null.
   * department\_id: Integer, Foreign Key referencing departments(department\_id).
   * hire\_date: Date, Not Null.
5. **Create a table named enrollments:**
   * enrollment\_id: Integer, Auto Increment, Primary Key.
   * student\_id: Integer, Foreign Key referencing students(student\_id).
   * course\_id: Integer, Foreign Key referencing courses(course\_id).
   * semester: Variable character, Not Null.
   * year: Integer, Not Null CHECK (year > 2000).
6. **Create a table named grades:**
   * grade\_id: Integer, Auto Increment, Primary Key.
   * enrollment\_id: Integer, Foreign Key referencing enrollments(enrollment\_id).
   * grade: Char(1) CHECK (grade IN ('A', 'B', 'C', 'D', 'F')), Not Null.
7. **Create a table named assignments:**
   * assignment\_id: Integer, Auto Increment, Primary Key.
   * course\_id: Integer, Foreign Key referencing courses(course\_id).
   * title: Variable character, Not Null.
   * due\_date: Date, Not Null.
8. **Create a table named submissions:**
   * submission\_id: Integer, Auto Increment, Primary Key.
   * assignment\_id: Integer, Foreign Key referencing assignments(assignment\_id).
   * student\_id: Integer, Foreign Key referencing students(student\_id).
   * submission\_date: Date, Not Null.
   * score: Decimal(5, 2) CHECK (score >= 0 AND score <= 100), Nullable.
9. **Create a table named classrooms:**
   * classroom\_id: Integer, Auto Increment, Primary Key.
   * room\_number: Variable character, Unique, Not Null.
   * capacity: Integer, Not Null CHECK (capacity > 0).
10. **Create a table named class\_schedule:**
    * schedule\_id: Integer, Auto Increment, Primary Key.
    * course\_id: Integer, Foreign Key referencing courses(course\_id).
    * classroom\_id: Integer, Foreign Key referencing classrooms(classroom\_id).
    * day\_of\_week: Enum ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'), Not Null.
    * start\_time: Time, Not Null.
    * end\_time: Time, Not Null CHECK (end\_time > start\_time).

**Solution:**

-- Step 1: Create the `departments` table

CREATE TABLE departments (

department\_id INT AUTO\_INCREMENT,

department\_name VARCHAR(100) UNIQUE NOT NULL,

location VARCHAR(100) NOT NULL,

PRIMARY KEY (department\_id)

);

-- Step 2: Create the `courses` table

CREATE TABLE courses (

course\_id INT AUTO\_INCREMENT,

course\_name VARCHAR(100) UNIQUE NOT NULL,

department\_id INT,

credits INT NOT NULL CHECK (credits > 0),

PRIMARY KEY (course\_id),

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

-- Step 3: Create the `students` table

CREATE TABLE students (

student\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

date\_of\_birth DATE NOT NULL,

enrollment\_date DATE NOT NULL,

PRIMARY KEY (student\_id)

);

-- Step 4: Create the `professors` table

CREATE TABLE professors (

professor\_id INT AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

department\_id INT,

hire\_date DATE NOT NULL,

PRIMARY KEY (professor\_id),

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

-- Step 5: Create the `enrollments` table

CREATE TABLE enrollments (

enrollment\_id INT AUTO\_INCREMENT,

student\_id INT,

course\_id INT,

semester VARCHAR(10) NOT NULL,

year INT NOT NULL CHECK (year > 2000),

PRIMARY KEY (enrollment\_id),

FOREIGN KEY (student\_id) REFERENCES students(student\_id),

FOREIGN KEY (course\_id) REFERENCES courses(course\_id)

);

-- Step 6: Create the `grades` table

CREATE TABLE grades (

grade\_id INT AUTO\_INCREMENT,

enrollment\_id INT,

grade CHAR(1) CHECK (grade IN ('A', 'B', 'C', 'D', 'F')) NOT NULL,

PRIMARY KEY (grade\_id),

FOREIGN KEY (enrollment\_id) REFERENCES enrollments(enrollment\_id)

);

-- Step 7: Create the `assignments` table

CREATE TABLE assignments (

assignment\_id INT AUTO\_INCREMENT,

course\_id INT,

title VARCHAR(100) NOT NULL,

due\_date DATE NOT NULL,

PRIMARY KEY (assignment\_id),

FOREIGN KEY (course\_id) REFERENCES courses(course\_id)

);

-- Step 8: Create the `submissions` table

CREATE TABLE submissions (

submission\_id INT AUTO\_INCREMENT,

assignment\_id INT,

student\_id INT,

submission\_date DATE NOT NULL,

score DECIMAL(5, 2) CHECK (score >= 0 AND score <= 100),

PRIMARY KEY (submission\_id),

FOREIGN KEY (assignment\_id) REFERENCES assignments(assignment\_id),

FOREIGN KEY (student\_id) REFERENCES students(student\_id)

);

-- Step 9: Create the `classrooms` table

CREATE TABLE classrooms (

classroom\_id INT AUTO\_INCREMENT,

room\_number VARCHAR(10) UNIQUE NOT NULL,

capacity INT NOT NULL CHECK (capacity > 0),

PRIMARY KEY (classroom\_id)

);

-- Step 10: Create the `class\_schedule` table

CREATE TABLE class\_schedule (

schedule\_id INT AUTO\_INCREMENT,

course\_id INT,

classroom\_id INT,

day\_of\_week ENUM('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday') NOT NULL,

start\_time TIME NOT NULL,

end\_time TIME NOT NULL CHECK (end\_time > start\_time),

PRIMARY KEY (schedule\_id),

FOREIGN KEY (course\_id) REFERENCES courses(course\_id),

FOREIGN KEY (classroom\_id) REFERENCES classrooms(classroom\_id)

);