Database Testing for Online Bookstore Project

Database testing is essential to ensure the integrity, performance, and reliability of the database design. Below is a structured approach for testing the **Online Bookstore Database**, covering validation tests, constraint tests, relationship checks, and performance tests.

---

1. Testing Objectives

1. Verify the correctness of relationships (e.g., cascading, restricting).
2. Validate data integrity constraints (e.g., primary keys, foreign keys, unique constraints).
3. Test cascading actions for `ON DELETE` and `ON UPDATE`.
4. Check for performance issues with queries and indexes.

---

2. Validation Tests

2.1 Primary Key Constraints

- **Test Case**: Attempt to insert duplicate `author_id` in the `authors` table.

```sql
INSERT INTO authors (author_id, name, email)
VALUES (1, 'Duplicate Author', 'duplicate@example.com');
-- Expected: Error due to duplicate primary key.
```

2.2 Unique Constraints

- **Test Case**: Attempt to insert duplicate email in the `customers` table.

```sql
INSERT INTO customers (name, email, phone)
VALUES ('John Doe', 'alice@example.com', '1234567890');
-- Expected: Error due to unique constraint violation on email.
```

2.3 NOT NULL Constraints

- **Test Case**: Try inserting a book without a title.

```sql
INSERT INTO books (price, stock_quantity, author_id)
VALUES (19.99, 10, 1);
-- Expected: Error due to NOT NULL constraint on title.
```

2.4 Check Constraints

- **Test Case**: Try inserting a negative quantity in `order_details`.

```sql
INSERT INTO order_details (order_id, book_id, quantity)
VALUES (1, 1, -5);
-- Expected: Error due to check constraint on quantity.
```

---

3. Relationship Testing

3.1 Cascading Updates

- **Test Case**: Update `author_id` in the `authors` table and check if it cascades to the `books` table.

```sql
UPDATE authors
SET author_id = 3
WHERE author_id = 1;

SELECT * FROM books WHERE author_id = 3;
-- Expected: All books previously associated with author_id 1 should now have author_id 3.
```

3.2 Cascading Deletions

- **Test Case**: Delete an `order` and verify cascading deletion in `order_details`.

```sql
DELETE FROM orders
WHERE order_id = 1;

SELECT * FROM order_details WHERE order_id = 1;
-- Expected: All order details for order_id 1 should be deleted.
```

3.3 Restricting Deletions

- **Test Case**: Attempt to delete a `customer` who has existing orders.

```sql
DELETE FROM customers
WHERE customer_id = 1;
-- Expected: Error due to restricting rule on customer_id in the orders table.
```

3.4 Prevent Deleting Books in Orders

- **Test Case**: Attempt to delete a book that is part of an order.

```sql
DELETE FROM books
WHERE book_id = 1;
-- Expected: Error due to restricting rule on book_id in order_details.
```

---

4. Performance Testing

4.1 Index Efficiency

- **Test Case**: Measure query performance with and without an index on `email` in the `customers` table.

```sql
EXPLAIN SELECT * FROM customers WHERE email = 'alice@example.com';
-- Expected: Query using an index should perform significantly faster than without it.
```

4.2 Query Execution Time

- **Test Case**: Analyze execution time for retrieving all orders with customer and book details.

```sql
SELECT
    orders.order_id,
    customers.name AS customer_name,
    books.title AS book_title,
    order_details.quantity
FROM
    orders
JOIN
    customers ON orders.customer_id = customers.customer_id
JOIN
```

```
    order_details ON orders.order_id = order_details.order_id
JOIN
    books ON order_details.book_id = books.book_id;
-- Expected: Query should execute within acceptable time limits for large
datasets.
```

4.3 Large Data Load

- **Test Case**: Populate tables with a large dataset and test retrieval performance.

```
-- Bulk insert customers
INSERT INTO customers (name, email, phone)
SELECT CONCAT('Customer ', id), CONCAT('customer', id, '@example.com'),
'1234567890'
FROM generate_series(1, 100000) AS id;

-- Query customers
SELECT * FROM customers WHERE email = 'customer50000@example.com';
-- Expected: Query should perform efficiently even with 100,000 records.
```

---

5. Data Integrity Tests

5.1 Validate Total Amount Calculation

- **Test Case**: Verify the `total_amount` in `orders` matches the sum of `line_total` in `order_details`.

```
SELECT
    orders.order_id,
    orders.total_amount,
    SUM(order_details.line_total) AS calculated_total
FROM
    orders
JOIN
    order_details ON orders.order_id = order_details.order_id
GROUP BY
    orders.order_id;
-- Expected: `total_amount` and `calculated_total` should match for all
orders.
```

5.2 Inventory Validation

- **Test Case**: Check if any book's stock quantity goes negative after multiple orders.

```
SELECT
    books.title,
    books.stock_quantity,
    SUM(order_details.quantity) AS total_ordered
FROM
    books
LEFT JOIN
    order_details ON books.book_id = order_details.book_id
GROUP BY
    books.book_id
HAVING
    books.stock_quantity < SUM(order_details.quantity);
-- Expected: No records should be returned.
```

---

6. Edge Case Testing

6.1 Empty Relationships

- **Test Case**: Query books with no associated author.

```
SELECT *
FROM books
WHERE author_id IS NULL;
-- Expected: Return books where author_id is NULL.
```

6.2 Orders Without Details

- **Test Case**: Query orders with no associated details.

```
SELECT *
FROM orders
WHERE NOT EXISTS (
    SELECT 1
    FROM order_details
    WHERE order_details.order_id = orders.order_id
);
-- Expected: Return orders with no entries in order_details.
```

Next Steps

1. Execute these test cases step by step in your database.
2. Document the results and resolve any unexpected errors.
3. Optimize the schema or queries based on the test results.