Implementing User Defined Integrity with Triggers

Implementing user-defined integrity with triggers in a database can be a powerful way to enforce business rules and maintain data integrity automatically. Here's a guide on how to implement this in a SQL-based database system:

1. Understanding Triggers

A trigger is a database object that is automatically executed or fired when certain events occur. Triggers can be used to enforce rules, validate data, and maintain complex business logic at the database level.

2. Basic Syntax of a Trigger

The basic syntax for creating a trigger in SQL is:

```
CREATE TRIGGER trigger_name
{ BEFORE | AFTER } { INSERT | UPDATE | DELETE }
ON table_name
FOR EACH ROW
BEGIN
  -- Trigger logic here
END;
```

3. Implementing User-Defined Integrity

Let's consider a scenario where we need to ensure that the value of a salary column in an employees table does not exceed a certain limit. We can use a trigger to enforce this rule.

Step-by-Step Implementation

1. Create the Employees Table:

```
CREATE TABLE employees (
  employee_id INT PRIMARY KEY,
  name VARCHAR(50),
  salary DECIMAL(10, 2)
);
```

2. Define the Trigger:

```
CREATE TRIGGER check_salary_limit
BEFORE INSERT OR UPDATE ON employees
```

```
FOR EACH ROW
BEGIN
  IF NEW.salary > 10000 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Salary cannot exceed 10000';
  END IF;
END;
```

3. Explanation:

- BEFORE INSERT OR UPDATE: The trigger will fire before an insert or update operation.
- NEW.salary > 10000: This condition checks if the new salary value exceeds the limit.
- SIGNAL SQLSTATE '45000': This statement raises an error with a specific SQLSTATE code and message.

4. Testing the Trigger:

```
-- This will succeed
INSERT INTO employees (employee_id, name, salary) VALUES (1, 'John Doe', 9000);

-- This will fail
INSERT INTO employees (employee_id, name, salary) VALUES (2, 'Jane Doe', 11000);
```

4. Updating Existing Records

To update existing records while ensuring the integrity check is applied, the same trigger logic will enforce the rule.

```
-- This will succeed
UPDATE employees SET salary = 9500 WHERE employee_id = 1;

-- This will fail
UPDATE employees SET salary = 10500 WHERE employee_id = 1;
```

5. Managing Complex Rules

For more complex integrity rules, you can extend the logic within the trigger. For instance, you can check multiple conditions or enforce relationships between different tables.

Example: Checking Foreign Key Integrity

1. Create the Departments Table:

```
CREATE TABLE departments (
  department_id INT PRIMARY KEY,
  department_name VARCHAR(50)
);
```

2. Modify the Employees Table:

```
ALTER TABLE employees ADD department_id INT;

-- Add a foreign key constraint
ALTER TABLE employees ADD CONSTRAINT fk_department
FOREIGN KEY (department_id) REFERENCES departments(department_id);
```

3. Define the Trigger for Foreign Key Check:

```
CREATE TRIGGER check_department_exists
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
  DECLARE dept_count INT;

  -- Check if the department exists
  SELECT COUNT(*) INTO dept_count
  FROM departments
  WHERE department_id = NEW.department_id;

  IF dept_count = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Department does not exist';
  END IF;
END;
```