1)Student-Teacher Database
Consider the following Entities and their Relationships for Student-Teacher database.
Student (s_no int, s_name varchar (20), s_class varchar (10), s_addr varchar (30))
Teacher (t_no int, t_name varchar (20), qualification varchar (15), experience int)
Relationship between Student and Teacher is many to many with descriptive attribute subject.
Constraints: Primary Key,
s_class should not be null.
Create trigger for the following:
1. Write a trigger before insert the record of Student. If the sno is less than or equal to zero give the
message “Invalid Number”.
2. Write a trigger before update a student’s s_class from student table. Display appropriatemessage.
3. Write a trigger before inserting into a teacher table to check experience. Experience should be
minimum 2 year. Display appropriate message.

Certainly! Here are the triggers for the specified requirements:

1. **Trigger to Check Invalid Student Number:**

```plpgsql
CREATE OR REPLACE FUNCTION check_student_number()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.s_no <= 0 THEN
        RAISE EXCEPTION 'Invalid Number';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_student
BEFORE INSERT ON Student
FOR EACH ROW
EXECUTE FUNCTION check_student_number();
```

2. **Trigger to Display Appropriate Message before Updating Student's Class:**

```plpgsql
CREATE OR REPLACE FUNCTION display_update_message()
RETURNS TRIGGER AS $$
BEGIN
    RAISE NOTICE 'Updating student class from % to %', OLD.s_class, NEW.s_class;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_update_student_class
BEFORE UPDATE ON Student
FOR EACH ROW
EXECUTE FUNCTION display_update_message();
```

3. **Trigger to Check Minimum Experience for Teacher:**

```plpgsql
CREATE OR REPLACE FUNCTION check_teacher_experience()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.experience < 2 THEN
      RAISE EXCEPTION 'Teacher must have a minimum experience of 2 years';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_teacher
BEFORE INSERT ON Teacher
FOR EACH ROW
EXECUTE FUNCTION check_teacher_experience();
```

These triggers will enforce the specified constraints and display appropriate messages based on the conditions specified in your requirements. Adjust them according to your specific needs and database schema.

1)Project-Employee Database Consider the following Entities and their
Relationships for Project-Employee database.
Project (pno integer, pname char (30), ptype char (20), duration integer)
Employee (eno integer, ename char (20), qualification char (15), joining_date date)
Relationship between Project and Employee is many to many with descriptive attribute start_date
date, no_of_hours_worked integer.
Constraints: Primary Key,
pname should not be null.
Create trigger for the following:
1. Write a trigger before inserting into an employee table to check current date should be always greater than joining date. Display appropriate message.
2. Write a trigger before inserting into a project table to check duration should be always greater than zero. Display appropriate message.
3. Write a trigger before deleting an employee record from employee table. Raise a notice and display the message “Employee record is being deleted”.

Certainly! Below are the triggers for the specified requirements:

1. **Trigger to Check Current Date against Joining Date for Employee:**

```plpgsql
CREATE OR REPLACE FUNCTION check_employee_joining_date()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.joining_date >= CURRENT_DATE THEN
      RAISE EXCEPTION 'Joining date should be less than the current date';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_employee
BEFORE INSERT ON Employee
FOR EACH ROW
EXECUTE FUNCTION check_employee_joining_date();
```

2. **Trigger to Check Duration for Project:**

```plpgsql
CREATE OR REPLACE FUNCTION check_project_duration()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.duration <= 0 THEN
      RAISE EXCEPTION 'Duration should be greater than zero';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_project
BEFORE INSERT ON Project
FOR EACH ROW
EXECUTE FUNCTION check_project_duration();
```

3. **Trigger before Deleting an Employee Record:**

```plpgsql
CREATE OR REPLACE FUNCTION before_delete_employee_record()
RETURNS TRIGGER AS $$
BEGIN
   RAISE NOTICE 'Employee record is being deleted';
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_delete_employee
```

```
BEFORE DELETE ON Employee
FOR EACH ROW
EXECUTE FUNCTION before_delete_employee_record();
```

These triggers will enforce the specified constraints and display appropriate messages based on the conditions specified in your requirements. Adjust them according to your specific needs and database schema.

1) Railway Reservation Database
Consider the following Entities and their Relationships for Railway Reservation database.
Train (tno int, tname varchar (20), depart_time time, arrival_time time, source_stn char (10), dest_stn char (10), no_of_res_bogies int ,bogie_capacity int)
Passenger (passenger_id int, passenger_name varchar (20), address varchar (30), age int, gender
char)
Relationship between Train and Passenger is many to many with descriptive attribute ticket.
Ticket (train_no int, passenger_id int, ticket_no int,bogie_no int, no_of_berths int, tdate date, ticket_amt decimal (7,2),status char)
Constraints: Primary Key,
Status of a berth can be 'W' (waiting) or 'C' (confirmed)
Create trigger for the following:
1. Write a trigger to restrict the bogie capacity of any train to 30.
2. Write a trigger after insert on passenger to display message “Age above 5 will be charged full fare” if age of passenger is more than 5.
3. Write a trigger to restrict no. of berths of ticket booking to 20

Certainly! Below are the triggers for the specified requirements:

1. **Trigger to Restrict Bogie Capacity of Any Train to 30:**

```plpgsql
CREATE OR REPLACE FUNCTION restrict_bogie_capacity()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.bogie_capacity > 30 THEN
      RAISE EXCEPTION 'Bogie capacity cannot exceed 30';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_train
BEFORE INSERT ON Train
FOR EACH ROW
EXECUTE FUNCTION restrict_bogie_capacity();
```

2. **Trigger After Insert on Passenger to Display Message for Age:**

```plpgsql
CREATE OR REPLACE FUNCTION display_age_message()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.age > 5 THEN
      RAISE NOTICE 'Age above 5 will be charged full fare';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_insert_passenger
AFTER INSERT ON Passenger
FOR EACH ROW
EXECUTE FUNCTION display_age_message();
```

3. **Trigger to Restrict Number of Berths in Ticket Booking to 20:**

```plpgsql
CREATE OR REPLACE FUNCTION restrict_berths_limit()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.no_of_berths > 20 THEN
      RAISE EXCEPTION 'Number of berths cannot exceed 20';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_ticket
BEFORE INSERT ON Ticket
FOR EACH ROW
EXECUTE FUNCTION restrict_berths_limit();
```

These triggers will enforce the specified constraints and display appropriate messages based on the conditions specified in your requirements. Adjust them according to your specific needs and database schema.

2) Bus Transport Database
Consider the following Entities and their Relationships for Bus Transport database.
Bus (bus_no int , b_capacity int , depot_name varchar(20))
Route (route_no int, source char (20), destination char (20), no_of_stations int)
Driver (driver_no int ,driver_name char(20), license_no int, address char(20), d_age int , salary float)

Relationship between Bus and Route is many to one and relationship between Bus and Driver is
many to many with descriptive attributes date_of_duty_allotted and shift.
Constraints: Primary Key
license_no is unique, b_capacity should not be null,
shift can be 1 (Morning) or 2(Evening).
Create trigger for the following:
1. Write a trigger after insert or update the record of driver if the age is between 18 and 50 give the
message “valid entry” otherwise give appropriate message.
2. Write a trigger after delete the record of bus having capacity < 10. Display the appropriate message.
3. Write a trigger which will prevent deleting drivers living in .

Certainly! Below are the triggers for the specified requirements:

1. **Trigger After Insert or Update Driver with Age Check:**

```plpgsql
CREATE OR REPLACE FUNCTION check_driver_age()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.d_age BETWEEN 18 AND 50 THEN
      RAISE NOTICE 'Valid entry';
   ELSE
      RAISE EXCEPTION 'Driver age should be between 18 and 50';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_insert_or_update_driver
AFTER INSERT OR UPDATE ON Driver
FOR EACH ROW
EXECUTE FUNCTION check_driver_age();
```

2. **Trigger After Delete Bus with Capacity Check:**

```plpgsql
CREATE OR REPLACE FUNCTION check_bus_capacity()
RETURNS TRIGGER AS $$
BEGIN
   IF OLD.b_capacity < 10 THEN
      RAISE EXCEPTION 'Cannot delete bus with capacity less than 10';
   ELSE
      RAISE NOTICE 'Bus record deleted';
   END IF;
   RETURN OLD;
```

```
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_delete_bus
AFTER DELETE ON Bus
FOR EACH ROW
EXECUTE FUNCTION check_bus_capacity();
```

3. **Trigger to Prevent Deleting Drivers Living In:**

```plpgsql
CREATE OR REPLACE FUNCTION prevent_deleting_drivers_living_in()
RETURNS TRIGGER AS $$
DECLARE
   driver_address VARCHAR(20);
BEGIN
   SELECT address INTO driver_address FROM Driver WHERE driver_no = OLD.driver_no;

   IF position('.' IN driver_address) > 0 THEN
      RAISE EXCEPTION 'Cannot delete drivers living in .';
   END IF;

   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_delete_driver
BEFORE DELETE ON Driver
FOR EACH ROW
EXECUTE FUNCTION prevent_deleting_drivers_living_in();
```

These triggers will enforce the specified constraints and display appropriate messages based on the conditions specified in your requirements. Adjust them according to your specific needs and database schema.

Consider the following Entities and their Relationships for Student-Competition database.
Student (sreg_no int ,s_name varchar(20), s_class char(10))
Competition (c_no int ,c_name varchar(20), c_type char(10))
Relationship between Student and Competition is many to many with descriptive attributes rank and year.
Constraints: Primary Key,
c_type should not be null,
c_type can be ‘sport’ or ‘academic’.
Create trigger for the following:
1. Write a trigger that restricts insertion of rank value greater than 3. (Raise user defined exception
and give appropriate massage)
2. Write a trigger on relationship table .If the year entered is greater than current year, itshould display message “Year is Invalid”.

3. Write a trigger on relationship table .If the year entered is greater than current year, it should be
changed current year.

Certainly! Below are the triggers for the specified requirements:

1. **Trigger to Restrict Insertion of Rank Value Greater Than 3:**

```plpgsql
CREATE OR REPLACE FUNCTION check_rank_value()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.rank > 3 THEN
      RAISE EXCEPTION 'Invalid rank value. Rank should not be greater than 3.';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_relationship
BEFORE INSERT ON Relationship
FOR EACH ROW
EXECUTE FUNCTION check_rank_value();
```

2. **Trigger to Check and Display Message for Invalid Year:**

```plpgsql
CREATE OR REPLACE FUNCTION check_invalid_year()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.year > EXTRACT(YEAR FROM CURRENT_DATE) THEN
      RAISE EXCEPTION 'Year is Invalid';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_relationship_year_check
BEFORE INSERT ON Relationship
FOR EACH ROW
EXECUTE FUNCTION check_invalid_year();
```

3. **Trigger to Change Year to Current Year if Entered Year is Greater:**

```plpgsql
CREATE OR REPLACE FUNCTION adjust_year_to_current()
RETURNS TRIGGER AS $$
```

```
BEGIN
   IF NEW.year > EXTRACT(YEAR FROM CURRENT_DATE) THEN
      NEW.year := EXTRACT(YEAR FROM CURRENT_DATE);
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_relationship_adjust_year
BEFORE INSERT ON Relationship
FOR EACH ROW
EXECUTE FUNCTION adjust_year_to_current();
```

These triggers will enforce the specified constraints and display appropriate messages based on the conditions specified in your requirements. Adjust them according to your specific needs and database schema.

2)Bank Database

Consider the following Entities and their Relationships for Bank database.
Branch (br_id integer, br_name char (30), br_city char (10))
Customer (cno integer, c_name char (20), caddr char (35), city char (20))
Loan_application (lno integer, l_amt_required money, l_amt_approved money, l_date date)
Relationship between Branch, Customer and Loan_application is Ternary.
Ternary (br_id integer, cno integer, lno integer)
Constraints: Primary Key,
l_amt_required should be greater than zero.
Create trigger for the following:
1. Write a trigger which will execute when you update customer number from customer. Display message “You can’t change existing customer number”.
2. Write a trigger to validate the loan amount approved. It must be less than the loan amount required.
3. Write a trigger before insert record of customer. If the customer number is less than or equal to
zero and customer name is null then give the appropriate message.

Certainly! Below are the triggers for the specified requirements:

1. **Trigger to Prevent Changing Existing Customer Number:**

```plpgsql
CREATE OR REPLACE FUNCTION prevent_change_existing_customer_number()
RETURNS TRIGGER AS $$
BEGIN
   IF OLD.cno <> NEW.cno THEN
      RAISE EXCEPTION 'You can''t change existing customer number';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_update_customer_number
BEFORE UPDATE ON Customer
FOR EACH ROW
EXECUTE FUNCTION prevent_change_existing_customer_number();
```

2. **Trigger to Validate Loan Amount Approved:**

```plpgsql
CREATE OR REPLACE FUNCTION validate_loan_amount()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.l_amt_approved >= NEW.l_amt_required THEN
      RAISE EXCEPTION 'Loan amount approved must be less than the loan amount required';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_loan_application
BEFORE INSERT ON Loan_application
FOR EACH ROW
EXECUTE FUNCTION validate_loan_amount();
```

3. **Trigger Before Inserting Record of Customer:**

```plpgsql
CREATE OR REPLACE FUNCTION check_customer_insert()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.cno <= 0 OR NEW.c_name IS NULL THEN
      RAISE EXCEPTION 'Customer number should be greater than zero and customer name
cannot be null';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_customer
BEFORE INSERT ON Customer
FOR EACH ROW
EXECUTE FUNCTION check_customer_insert();
```

These triggers will enforce the specified constraints and display appropriate messages based on the conditions specified in your requirements. Adjust them according to your specific needs and database schema.