2. Movie-Actor Database
Movies (m_name varchar (25), release_year integer, budget money)
Actor (a_name char (30), role char (30), charges money, a_address varchar(30))
Producer (producer_id integer, name char (30), p_address varchar (30))
The relationships are as follows:
Each actor has acted in one or more movies. Each producer has produced many movies and
each movie
can be produced by more than one producers. Each movie has one or more actors acting in it,
in different
roles.
1. Write a stored function to accept movie name as input and display the details of actors for
that
movie and sort it by their charges in descending order. (Accept movie name as input
parameter).
Raise an exception for an invalid movie name.
2. Write a stored function to accept actor / actress name as input and display the names of
moviesin
which that actor has acted in. (Accept actor name as input parameter). Raise an exception for
an
invalid actor name.
3. Write a stored function to accept producer name as input and display the count of movies
he/she
has produced. (Accept producer name as input parameter). Raise an exception for an invalid
producer name.

Certainly! Here are the stored functions for the given requirements:

1. Write a stored function to accept the movie name as input and display the details of actors for that movie and sort it by their charges in descending order. (Accept movie name as an input parameter). Raise an exception for an invalid movie name.

```plpgsql
CREATE OR REPLACE FUNCTION get_actors_by_movie(movie_name VARCHAR(25))
RETURNS TABLE(a_name CHAR(30), role CHAR(30), charges MONEY) AS $$
BEGIN
   RETURN QUERY
   SELECT a.a_name, a.role, a.charges
   FROM Actor a
   JOIN Movies_Actors ma ON a.a_name = ma.a_name
   JOIN Movies m ON ma.m_name = m.m_name
   WHERE m.m_name = movie_name
   ORDER BY a.charges DESC;

   IF NOT FOUND THEN
      RAISE EXCEPTION 'Invalid movie name: %', movie_name;
   END IF;
END;
$$ LANGUAGE plpgsql;
```

2. Write a stored function to accept actor/actress name as input and display the names of movies in which that actor has acted in. (Accept actor name as an input parameter). Raise an exception for an invalid actor name.

```plpgsql
CREATE OR REPLACE FUNCTION get_movies_by_actor(actor_name CHAR(30))
RETURNS TABLE(m_name VARCHAR(25), release_year INTEGER, budget MONEY) AS $$
BEGIN
  RETURN QUERY
  SELECT m.m_name, m.release_year, m.budget
  FROM Movies m
  JOIN Movies_Actors ma ON m.m_name = ma.m_name
  JOIN Actor a ON ma.a_name = a.a_name
  WHERE a.a_name = actor_name;

  IF NOT FOUND THEN
    RAISE EXCEPTION 'Invalid actor name: %', actor_name;
  END IF;
END;
$$ LANGUAGE plpgsql;
```

3. Write a stored function to accept producer name as input and display the count of movies he/she has produced. (Accept producer name as an input parameter). Raise an exception for an invalid producer name.

```plpgsql
CREATE OR REPLACE FUNCTION get_movie_count_by_producer(producer_name
CHAR(30))
RETURNS INTEGER AS $$
DECLARE
  movie_count INTEGER;
BEGIN
  SELECT COUNT(*)
  INTO movie_count
  FROM Movies_Producers mp
  JOIN Producer p ON mp.producer_id = p.producer_id
  WHERE p.name = producer_name;

  IF movie_count = 0 THEN
    RAISE EXCEPTION 'Invalid producer name: %', producer_name;
  END IF;

  RETURN movie_count;
END;
$$ LANGUAGE plpgsql;
```

These stored functions should help you retrieve the desired information from your Movie-Actor database. Ensure that your database tables and relationships are set up correctly before running these functions.

1. Bank Database
Branch (bid integer, br_name char (30), br_city char (10))
Customer (cno integer, cname char (20), caddr char (35), city char(20))
Loan_application (lno integer, l_amt_require money, l_amt_approved Money, l_date
date)
The relationships are as follows:
Branch, customer, loan_application are related with ternary relationship.
Ternary (bid integer, cno integer, lno integer).
1. Write a stored function to accept customer name as input and display the loan details of that customer. (Accept customer name as input parameter). Raise an exception for an invalid customer
name.
2. Write a stored function to display details of customers of particular branch. (Accept branch name
as input parameter) Display appropriate error message if branch name is invalid.
3. Write a stored function to display numbers of loan approved after a particular date. (Acceptloan
date as input parameter) Display appropriate message if loan date does not exist.

Certainly! Here are the stored functions for the given requirements:

1. Write a stored function to accept the customer name as input and display the loan details of that customer. (Accept customer name as an input parameter). Raise an exception for an invalid customer name.

```plpgsql
CREATE OR REPLACE FUNCTION get_loan_details_by_customer(customer_name
CHAR(20))
RETURNS TABLE(lno INT, l_amt_require MONEY, l_amt_approved MONEY, l_date DATE) AS
$$
BEGIN
   RETURN QUERY
   SELECT la.lno, la.l_amt_require, la.l_amt_approved, la.l_date
   FROM Loan_application la
   JOIN Ternary t ON la.lno = t.lno
   JOIN Customer c ON t.cno = c.cno
   WHERE c.cname = customer_name;

   IF NOT FOUND THEN
      RAISE EXCEPTION 'Invalid customer name: %', customer_name;
   END IF;
END;
$$ LANGUAGE plpgsql;
```

`` ``` ``

2. Write a stored function to display details of customers of a particular branch. (Accept branch name as an input parameter). Display an appropriate error message if the branch name is invalid.

```plpgsql
CREATE OR REPLACE FUNCTION get_customers_by_branch(branch_name CHAR(30))
RETURNS TABLE(cno INT, cname CHAR(20), caddr CHAR(35), city CHAR(20)) AS $$
BEGIN
    RETURN QUERY
    SELECT c.cno, c.cname, c.caddr, c.city
    FROM Customer c
    JOIN Ternary t ON c.cno = t.cno
    JOIN Branch b ON t.bid = b.bid
    WHERE b.br_name = branch_name;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Invalid branch name: %', branch_name;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

3. Write a stored function to display the numbers of loans approved after a particular date. (Accept loan date as an input parameter). Display an appropriate message if the loan date does not exist.

```plpgsql
CREATE OR REPLACE FUNCTION get_approved_loans_after_date(loan_date DATE)
RETURNS TABLE(lno INT, l_amt_approved MONEY, l_date DATE) AS $$
BEGIN
    RETURN QUERY
    SELECT l.lno, l.l_amt_approved, l.l_date
    FROM Loan_application l
    WHERE l.l_date > loan_date;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'No loans approved after the given date: %', loan_date;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

These stored functions should help you retrieve the desired information from your Bank database. Ensure that your database tables and relationships are set up correctly before running these functions.
2.Railway Reservation System Database
TRAIN: (train_no int, train_name varchar(20), depart_time time , arrival_time time, source_stn varchar (20),dest_stn varchar (20), no_of_res_bogies int ,bogie_capacity int)

PASSENGER : (passenger_id int, passenger_name varchar(20), address varchar(30), age int
,gender
char)
Relationships:
Train _Passenger: M-M relationship named ticket with descriptive attributes as follows
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: The status of a berth can be 'W' (waiting) or 'C' (confirmed).
1. Write a stored function to print the details of train wise confirmed bookings on date
“ ” (Accept date as input parameter).Raise an error in case of invalid date.
2. Write a stored function to accept date and passenger name and display no of berths reserved
and
ticket amount paid by him. Raise exception if passenger name is invalid.
3. Write a stored function to display the ticket details of a train. (Accept train name as input
parameter).Raise an exception in case of invalid train name.

Here are the stored functions for the given requirements:

1. Write a stored function to print the details of train-wise confirmed bookings on date “ ” (Accept
date as an input parameter). Raise an error in case of an invalid date.

```plpgsql
CREATE OR REPLACE FUNCTION get_confirmed_bookings_by_date(reservation_date DATE)
RETURNS TABLE(train_name VARCHAR(20), passenger_name VARCHAR(20), no_of_berths
INT, ticket_amount DECIMAL(7,2)) AS $$
BEGIN
   RETURN QUERY
   SELECT t.train_name, p.passenger_name, t.no_of_berths, t.ticket_amt
   FROM TICKET t
   JOIN TRAIN_PASSENGER tp ON t.train_no = tp.train_no AND t.passenger_id =
tp.passenger_id
   JOIN PASSENGER p ON tp.passenger_id = p.passenger_id
   WHERE t.tdate = reservation_date AND t.status = 'C';

   IF NOT FOUND THEN
      RAISE EXCEPTION 'No confirmed bookings found for the given date: %',
reservation_date;
   END IF;
END;
$$ LANGUAGE plpgsql;
```

2. Write a stored function to accept date and passenger name and display the number of berths
reserved and ticket amount paid by him. Raise an exception if the passenger name is invalid.

```plpgsql
CREATE OR REPLACE FUNCTION get_berths_and_amount_by_passenger(reservation_date
DATE, passenger_name VARCHAR(20))
RETURNS TABLE(no_of_berths INT, ticket_amount DECIMAL(7,2)) AS $$
```

```
BEGIN
    RETURN QUERY
    SELECT t.no_of_berths, t.ticket_amt
    FROM TICKET t
    JOIN PASSENGER p ON t.passenger_id = p.passenger_id
    WHERE t.tdate = reservation_date AND p.passenger_name = passenger_name;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Invalid passenger name: %', passenger_name;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

3. Write a stored function to display the ticket details of a train. (Accept train name as an input parameter). Raise an exception in case of an invalid train name.

```plpgsql
CREATE OR REPLACE FUNCTION get_ticket_details_by_train(train_name VARCHAR(20))
RETURNS TABLE(train_no INT, passenger_name VARCHAR(20), ticket_no INT, no_of_berths
INT, ticket_amount DECIMAL(7,2), status CHAR) AS $$
BEGIN
    RETURN QUERY
    SELECT t.train_no, p.passenger_name, t.ticket_no, t.no_of_berths, t.ticket_amt, t.status
    FROM TICKET t
    JOIN TRAIN_PASSENGER tp ON t.train_no = tp.train_no AND t.passenger_id =
tp.passenger_id
    JOIN PASSENGER p ON tp.passenger_id = p.passenger_id
    WHERE t.train_name = train_name;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Invalid train name: %', train_name;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

1) Bus Driver Database
BUS (bus_no int , 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)
The relationships are as follows:
BUS_ROUTE: M-1
BUS_DRIVER: M-M with descriptive attributes Date of duty allotted and Shift – it can be 1 (Morning)
or 2 (Evening).
Constraints:
License_no is unique. 2. Bus capacity is not null
1. Write a stored function to accept the bus_no and date and print its allotted drivers. Raise an

exception in case of invalid bus number.
2. Write a stored function to display the all Dates on which a driver has driven any bus.(Accept driver name as input parameter).Raise an exception in case of invalid driver name.
3. Write a stored function to display the details of the buses that run on route_no = “ ”.
(accept route_no as input parameter). Raise an error in case of invalid driver name.

Here are the stored functions for the given requirements:

1. Write a stored function to accept the bus_no and date and print its allotted drivers. Raise an exception in case of an invalid bus number.

```plpgsql
CREATE OR REPLACE FUNCTION get_allotted_drivers(bus_number INT, duty_date DATE)
RETURNS TABLE(driver_name CHAR(20), license_no INT, shift INT) AS $$
BEGIN
   RETURN QUERY
   SELECT d.driver_name, d.license_no, bd.shift
   FROM BUS_DRIVER bd
   JOIN DRIVER d ON bd.driver_no = d.driver_no
   WHERE bd.bus_no = bus_number AND bd.duty_date = duty_date;

   IF NOT FOUND THEN
      RAISE EXCEPTION 'Invalid bus number: %', bus_number;
   END IF;
END;
$$ LANGUAGE plpgsql;
```

2. Write a stored function to display all Dates on which a driver has driven any bus. (Accept driver name as an input parameter). Raise an exception in case of an invalid driver name.

```plpgsql
CREATE OR REPLACE FUNCTION get_all_dates_for_driver(driver_name CHAR(20))
RETURNS TABLE(duty_date DATE) AS $$
BEGIN
   RETURN QUERY
   SELECT DISTINCT bd.duty_date
   FROM BUS_DRIVER bd
   JOIN DRIVER d ON bd.driver_no = d.driver_no
   WHERE d.driver_name = driver_name;

   IF NOT FOUND THEN
      RAISE EXCEPTION 'Invalid driver name: %', driver_name;
   END IF;
END;
$$ LANGUAGE plpgsql;
```

3. Write a stored function to display the details of the buses that run on route_no = “ ”. (accept route_no as an input parameter). Raise an error in case of an invalid route number.

```plpgsql
CREATE OR REPLACE FUNCTION get_buses_on_route(route_number INT)
RETURNS TABLE(bus_no INT, capacity INT, depot_name VARCHAR(20)) AS $$
BEGIN
    RETURN QUERY
    SELECT b.bus_no, b.capacity, b.depot_name
    FROM BUS_ROUTE br
    JOIN BUS b ON br.bus_no = b.bus_no
    WHERE br.route_no = route_number;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Invalid route number: %', route_number;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

2. Student Teacher Database
Student (sno integer, s_name char(30), s_class char(10), s_addr Char(50))
Teacher (tno integer, t_name char (20), qualification char (15),experience
integer)
The relationship is as follows:
Student-Teacher: M-M with descriptive attribute Subject.
1. Write a stored function to count the number of the teachers teaching to a student named
“ ”. (Accept student name as input parameter). Raise an exception if student name does
not exist.
2. Write a stored function to count the number of the students who are studying subject named
“ ” (Accept subject name as input parameter). Display error message if subject name is
not valid.
3. Write a stored function to display teacher details who have qualification as “ ” (Accept
teacher’s qualification as input parameter). Raise an exception for invalid qualification.

Here are the stored functions for the given requirements:

1. Write a stored function to count the number of teachers teaching to a student named “ ”. (Accept student name as an input parameter). Raise an exception if the student name does not exist.

```plpgsql
CREATE OR REPLACE FUNCTION count_teachers_for_student(student_name VARCHAR)
RETURNS INTEGER AS $$
DECLARE
    teacher_count INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO teacher_count
    FROM Teacher t
    JOIN Student_Teacher st ON t.tno = st.tno
    JOIN Student s ON st.sno = s.sno
    WHERE s.s_name = student_name;

    IF teacher_count = 0 THEN
        RAISE EXCEPTION 'Student % not found.', student_name;
    END IF;

    RETURN teacher_count;
END;
$$ LANGUAGE plpgsql;
```

2. Write a stored function to count the number of students who are studying a subject named “ ” (Accept subject name as an input parameter). Display an error message if the subject name is not valid.

```plpgsql
CREATE OR REPLACE FUNCTION count_students_for_subject(subject_name VARCHAR)
RETURNS INTEGER AS $$
DECLARE
    student_count INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO student_count
    FROM Student_Teacher st
    WHERE st.subject = subject_name;

    IF student_count = 0 THEN
        RAISE EXCEPTION 'Subject % not found.', subject_name;
    END IF;

    RETURN student_count;
END;
$$ LANGUAGE plpgsql;
```

3. Write a stored function to display teacher details who have qualification as “ ” (Accept teacher’s qualification as an input parameter). Raise an exception for an invalid qualification.

```plpgsql
CREATE OR REPLACE FUNCTION get_teachers_by_qualification(teacher_qualification
VARCHAR)
RETURNS TABLE(t_name CHAR(20), qualification CHAR(15), experience INTEGER) AS $$
BEGIN
    RETURN QUERY
    SELECT t.t_name, t.qualification, t.experience
    FROM Teacher t
    WHERE t.qualification = teacher_qualification;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Invalid qualification: %', teacher_qualification;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

These stored functions should help you achieve the desired functionality in your Student-Teacher database.