```
/* Trigger 1 */

CREATE OR REPLACE FUNCTION check_ausgeschrieben() RETURNS TRIGGER AS $$
DECLARE
      jahr DOUBLE PRECISION;
      praktikum_row RECORD;
BEGIN
      SELECT CAST(a.jahr AS DOUBLE PRECISION) INTO jahr
      FROM Austauschprogramm a
      WHERE new.land = a.land AND new.jahr = a.jahr AND new.name = a.name;

      SELECT * INTO praktikum_row
      FROM Praktikum p
      WHERE p.prnr = new.prnr AND p.abtnr = new.abtnr AND p.fanr = new.fanr;

      IF (EXTRACT(YEAR FROM praktikum_row.von) < jahr OR EXTRACT(YEAR FROM praktikum_row.bis) < jahr) THEN
            RAISE EXCEPTION 'Das "von"-Datum und das "bis"-Datum duerfen nicht vor dem Jahr des
Austauschprogramms liegen!';
      END IF;

      RETURN new;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER t_before_ausgeschrieben BEFORE INSERT ON ausgeschrieben
FOR EACH ROW EXECUTE PROCEDURE check_ausgeschrieben();

/* Trigger 2 */

CREATE OR REPLACE FUNCTION check_zustaendig() RETURNS TRIGGER AS $$
DECLARE
BEGIN
      IF EXISTS(SELECT * FROM Kommission WHERE koordinator = new.betreuer) THEN
            RAISE EXCEPTION 'Ein Koordinator darf nicht Betreuer einer Firma sein!';
      END IF;

      IF EXISTS(
            SELECT svnr
            FROM ausgeschrieben aus NATURAL JOIN Austauschprogramm ap
            JOIN Kommission k ON ap.kommission = k.id JOIN Mitarbeiter m ON k.id = m.arbeitet
            WHERE fanr = new.firma AND svnr = new.betreuer) THEN

            RETURN NEW;

      END IF;

      RAISE EXCEPTION 'Der Betreuer arbeitet nicht fuer ein Austauschprogramm, zu welchem die Firma ein
Praktikum ausgeschrieben hat!';
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER t_before_zustaendig BEFORE INSERT ON zustaendig
FOR EACH ROW EXECUTE PROCEDURE check_zustaendig();

/* Function */

CREATE OR REPLACE FUNCTION f_calc_salary(soznr VARCHAR(40), datum DATE) RETURNS NUMERIC(7,2) AS $$
DECLARE
      mitarbeiter_r RECORD;
      betr_studs_c INTEGER;
      bon NUMERIC(7,2);
      sum NUMERIC(7,2);
BEGIN
      IF NOT EXISTS(SELECT * FROM Mitarbeiter WHERE svnr = soznr) THEN
            RAISE EXCEPTION 'Mitarbeiter mit der SVNR % nicht vorhanden!', soznr;
      END IF;

      SELECT * INTO mitarbeiter_r FROM Mitarbeiter WHERE svnr = soznr;

      IF (mitarbeiter_r.beschaeftigt_seit > datum) THEN
```

```
            RAISE EXCEPTION 'Mitarbeiter mit der SVNR % war zu dem uebergebenem Zeitpunkt noch nicht
angestellt!', soznr;
      END IF;

      sum := 0.0;

      IF(mitarbeiter_r.gehaltsklasse = 'C1') THEN sum := 3000; END IF;
      IF(mitarbeiter_r.gehaltsklasse = 'C2') THEN sum := 2500; END IF;
      IF(mitarbeiter_r.gehaltsklasse = 'M1') THEN sum := 2000; END IF;
      IF(mitarbeiter_r.gehaltsklasse = 'M2') THEN sum := 1500; END IF;
      IF(mitarbeiter_r.gehaltsklasse = 'M3') THEN sum := 1000; END IF;

      SELECT count(matched_student) INTO betr_studs_c
      FROM betreut NATURAL JOIN Praktikum
      WHERE betreuer = mitarbeiter_r.svnr AND datum <= bis AND datum >= von;

      SELECT bonus INTO bon FROM Betreuer WHERE svnr = soznr;

      IF (bon IS NOT NULL) THEN
            sum := sum + (betr_studs_c * bon);
      END IF;

      RETURN sum;
END;
$$ LANGUAGE plpgsql;

/* Procedure */

CREATE OR REPLACE FUNCTION p_update_students() RETURNS VOID AS $$
DECLARE
      students_c REFCURSOR;
      student_r RECORD;
BEGIN
      OPEN students_c FOR SELECT * FROM Student;
      FETCH students_c INTO student_r;

      WHILE found LOOP
            IF NOT EXISTS(SELECT * FROM ShortlistedStudent WHERE svnr = student_r.svnr) THEN
                  INSERT INTO ShortlistedStudent(svnr) VALUES (student_r.svnr);
                  RAISE NOTICE 'Updated Student with SVNR % to ShortlistedStudent', student_r.svnr;
            ELSE
                  IF NOT EXISTS(SELECT * FROM MatchedStudent WHERE svnr = student_r.svnr) THEN
                        INSERT INTO MatchedStudent(svnr) VALUES (student_r.svnr);
                        RAISE NOTICE 'Updated ShortlistedStudent with SVNR % to MatchedStudent',
student_r.svnr;
                  END IF;
            END IF;

            FETCH students_c INTO student_r;
            END LOOP;
      CLOSE students_c;
END;
$$ LANGUAGE plpgsql;
```