```
-- Function to increment a number
CREATE OR REPLACE FUNCTION inc(n INT) RETURNS INT AS
$BODY$
BEGIN
	RETURN n+1;
END;
$BODY$
LANGUAGE plpgsql;

-- Function to add two numbers.
CREATE OR REPLACE FUNCTION sumOfTwo(n1 REAL, n2 REAL) RETURNS REAL AS
$BODY$
BEGIN
	RETURN n1+n2;
END;
$BODY$
LANGUAGE plpgsql;

--Function to find the largest and smallest among three numbers
--				Using OUT we can receive more than one variable from function.
CREATE OR REPLACE FUNCTION high_low(
n1 NUMERIC, n2 NUMERIC, n3 NUMERIC,
OUT high NUMERIC, OUT low NUMERIC
)
AS
$BODY$
BEGIN
	high = GREATEST(n1, n2, n3);
	low = LEAST(n1, n2, n3);
END;
$BODY$
LANGUAGE plpgsql;

--Function to find sum of n natural numbers
CREATE OR REPLACE FUNCTION sumOfNatNum(n INT) RETURNS INT AS
$BODY$
DECLARE
	cnt INT = 1;
	s INT = 0;
BEGIN
	LOOP
	EXIT WHEN cnt > n;
		s = s + cnt;
		cnt = cnt + 1;
	END LOOP;
	RETURN s;
END;
$BODY$
LANGUAGE plpgsql;

--FUNCTION to find the nth term in fibonacci series
CREATE OR REPLACE FUNCTION fib(n INT) RETURNS INT AS
$BODY$
DECLARE
	prev INT = 0;
```

```
        curr INT = 1;
        next INT;
        cnt INT = 0;
BEGIN
        LOOP
        EXIT WHEN cnt=n-1;
                next = prev + curr;
                prev = curr;
                curr = next;
                cnt = cnt + 1;
        END LOOP;
        RETURN prev;
END;
$BODY$
LANGUAGE plpgsql;

--Function which returns terms in fibonacci series upto N
CREATE OR REPLACE FUNCTION fibo(n INT) RETURNS INT[] AS
$BODY$
DECLARE
        prev INT = 0;
        curr INT = 1;
        next INT;
        cnt INT = 0;
        fib INT[];
BEGIN
        LOOP
        EXIT WHEN cnt=n;
                fib[cnt] = prev;
                next = prev + curr;
                prev = curr;
                curr = next;
                cnt = cnt + 1;
        END LOOP;
        RETURN fib;
END;
$BODY$
LANGUAGE plpgsql;


--Fuction to check whether a number is prime or not
CREATE OR REPLACE FUNCTION isPrime(n INT) RETURNS BOOL AS
$BODY$
DECLARE
        cnt INT = 2;
        f BOOL = TRUE;
BEGIN
        IF (n = 1) THEN
                RAISE INFO '% is neither prime nor composite', n;
                RETURN NULL;
        ELSE
        LOOP
                EXIT WHEN cnt >n/2;
                IF(n%cnt = 0) THEN
                        f = FALSE;
                EXIT;
            END IF;
        cnt = cnt+1;
        END LOOP;
        RETURN f;
        END IF;
END;
$BODY$
LANGUAGE plpgsql;

--Function to divide two numbers
CREATE OR REPLACE FUNCTION divide(n1 NUMERIC, n2 NUMERIC) RETURNS NUMERIC AS
$BODY$
DECLARE
        res NUMERIC = 0;
BEGIN
        --IF n2 = 0 THEN RAISE EXCEPTION 'Division by zero' USING HINT='The denominator cannot
be zero';
        --ELSE res = n1/n2;
        --END IF;
        res = n1/n2;
        RETURN res;
END;
$BODY$
LANGUAGE plpgsql;

--Function to divide two numbers with exception handling
CREATE OR REPLACE FUNCTION divide_exc_hand(n1 NUMERIC, n2 NUMERIC) RETURNS
NUMERIC AS
$BODY$
DECLARE
        res NUMERIC = 0;
BEGIN
        res = n1/n2;
        RETURN res;
        EXCEPTION
        WHEN division_by_zero THEN
        RAISE NOTICE 'caught division_by_zero';
        RETURN res;
        WHEN others THEN
        RAISE NOTICE 'Some Exception has occured';
        RETURN res;
END;
$BODY$
LANGUAGE plpgsql;

SELECT sumOfTwo(4, 5);
SELECT high_low(3, 6, 2);
SELECT sumOfNatNum(10);
SELECT fib(10);
SELECT fibo(10);
SELECT isprime(17);
SELECT divide(5,2);
SELECT divide_exc_hand(5, 2);
```

```
/*
STUDENT DATABASE
class_(class_id, class_name, division, st_cnt)
student(st_id, st_fname, st_lname, class_id)
teacher(tr_id, tr_fname, tr_lname, subject)
stud_class(st_id, class_id, tr_id)
*/

CREATE TABLE class_(
	class_id SERIAL PRIMARY KEY,
	class_name VARCHAR(30) NOT NULL,
	division VARCHAR(2) NOT NULL DEFAULT 'A',
	st_cnt INT NOT NULL CHECK(st_cnt >= 0),
	UNIQUE(class_name, division)
);

INSERT INTO class_ (class_name, division, st_cnt) VALUES
('Ten', 'A', 3),
('Ten', 'B', 4),
('Ten', 'C', 5),
('Ten', 'D', 6),
('Ten', 'E', 7),
('Ten', 'F', 8);

CREATE TABLE student(
	st_id SERIAL PRIMARY KEY,
	st_fname VARCHAR(30) NOT NULL,
	st_lname VARCHAR(30) NOT NULL,
	class_id INT REFERENCES class_(class_id)
);

INSERT INTO student (st_fname, st_lname, class_id) VALUES
('Abijith', 'R', 1),
('Abirami', 'S Pillai', 1),
('Adish', 'A P', 1),
('Aiswarya', 'A S', 2),
('Aiswarya', 'Ramdas', 2),
('Anagha', 'Sree', 2),
('Anagha', 'V',    2),
('Anamika', 'Sudheesh', 3),
('Ansaf', 'Muhammed P T', 3),
('Anushamol', 'S', 3),
('Arjun', 'T B', 3),
('Aromal', 'V Ashokan', 3),
('Ashwati', 'Ashok Kumar', 4),
('Ashwin', 'Das', 4),
('Ashwin', 'Titus', 4),
('Ashwin', 'Pradeep', 4),
('Athira', 'S Pillai', 4),
('BalaSubhramani', '', 4),
('Bhagyalekshmi', 'Jaya Prakash', 5),
('Deepu', 'Krishnan U', 5),
('Delna', 'K Bijo', 5),
('Devi', 'R Jayan', 5),
('Fathima', 'Heera F', 5),
('Fathimath', 'Shifana E S', 5),
('Gokul', 'S Krishnan', 5),
('Jishma', 'Shareena', 6),
('Jithu', 'P', 6),
('Kavya', 'T Kunjumon', 6),
('Manikantan', 'K', 6),
('Midhun', 'K R', 6),
('Karupaswamy', 'M', 6),
('Mohamed', 'Saif Muthanikkatt', 6),
('Muhsina', 'Binth Abdulla C', 6);


CREATE TABLE teacher(
	tr_id SERIAL PRIMARY KEY,
	tr_fname VARCHAR(30) NOT NULL,
	tr_lname VARCHAR(30) NOT NULL,
	subject VARCHAR(30) NOT NULL
);

INSERT INTO teacher (tr_fname, tr_lname, subject) VALUES
('Lina', 'Max', 'English'),
('Jaya', 'Prakash', 'Maths'),
('Naveen', '', 'CS'),
('Manu', 'M L', 'Science');
/*
If we are going to do such mapping (more than one subject for a teacher), then it would be better
to create a new table subject with sub_id and subject and then put sub_id in the teacher table.
INSERT INTO teacher (tr_fname, tr_lname, subject) VALUES
('Lina', 'Max', 'Maths'),
('Jaya', 'Prakash', 'English'),
('Naveen', '', 'Science'),
('Manu', 'M L', 'CS');
*/

CREATE TABLE stud_class(
	st_id INT REFERENCES student(st_id),
	class_id INT REFERENCES class_(class_id),
	tr_id INT REFERENCES teacher(tr_id),
	UNIQUE(st_id, class_id, tr_id)
);


--To find students belonging to a particular class using Join and Nested Query
SELECT st_id, CONCAT(class_name, ' - ', division) AS "class", CONCAT(st_fname, ' ', st_lname) AS
"Name"
FROM student NATURAL JOIN class_
WHERE class_id IN (SELECT class_id FROM class_ WHERE class_name LIKE 'Ten' AND division
LIKE 'C');

--To find teachers who teach Science
SELECT *FROM teacher WHERE subject LIKE 'Science';
```

```
--To find st_id of students belonging to a particualar class.
SELECT st_id
FROM student NATURAL JOIN class_
WHERE class_id IN (SELECT class_id FROM class_ WHERE class_name LIKE 'Ten' AND division
LIKE 'C');

--To find tr_id based on the teacher's name and subject they are teaching
SELECT tr_id FROM teacher
WHERE tr_fname LIKE 'Manu' AND tr_lname LIKE 'M L'
AND subject LIKE 'Science';


--Function to map teacher to entire students in a class_    using cursor.
CREATE OR REPLACE FUNCTION mapTrToClass(t_fname_ VARCHAR, tr_lname_ VARCHAR,
class_name_ VARCHAR, division_ VARCHAR)
RETURNS VOID AS
$BODY$
DECLARE
    --s_id INT;
    c_id INT;
    t_id INT;
    rec RECORD;
    curStu CURSOR FOR SELECT s.st_id
            FROM student AS s NATURAL JOIN class_ AS c
            WHERE c.class_id
            IN (SELECT class_id FROM class_ AS c1 WHERE c1.class_name LIKE class_name_
AND c1.division LIKE division_);
BEGIN
    SELECT class_id INTO c_id FROM class_ AS c1 WHERE c1.class_name LIKE class_name_
AND c1.division LIKE division_;
    SELECT tr_id INTO t_id FROM teacher WHERE tr_fname LIKE t_fname_ AND tr_lname LIKE
tr_lname_;
    FOR rec in curStu
    LOOP
        --RAISE NOTICE '%', rec.st_id;
        INSERT INTO stud_class VALUES(rec.st_id, c_id, t_id);
    END LOOP;
    EXCEPTION
    WHEN unique_violation THEN
    RAISE NOTICE 'An unique constraint violation has occured';
    WHEN others THEN
    RAISE NOTICE 'An exception has occured';
RAISE NOTICE 'c_id: %       t_id: %', c_id, t_id;
END;
$BODY$
LANGUAGE plpgsql;

SELECT mapTrToClass('Lina', 'Max', 'Ten', 'C');

--POSTGRESQL TRIGGERS
--To create log details of every student's entry date.
CREATE TABLE audit(
    st_id INT NOT NULL,
    entry_date DATE NOT NULL
);

CREATE OR REPLACE FUNCTION auditLogFunc()
RETURNS TRIGGER AS $audit_entry_trigger$
BEGIN
        INSERT INTO audit(st_id, entry_date) VALUES (new.st_id, DATE(CURRENT_TIMESTAMP));
        RETURN NEW;
END;
$audit_entry_trigger$
LANGUAGE plpgsql;

CREATE TRIGGER audit_entry_trigger AFTER INSERT ON student
FOR EACH ROW EXECUTE PROCEDURE auditLogFunc();
```