Interactive - 36 - stored procedures

We have been using store procedures for a number of examples so far. Triggers in PostgreSQL call stored procedures. In PostgreSQL all stored procedures are called "functions".

PostgreSQL allows for stored procedures in multiple languages. The default language is PL/pgSQL and this us defined using the `LANGUAGE` specifier. I usually put this at the end but it can be done at the top also.

The section from $$ to $$ is the body that is saved and run when the function is called.

There are a slew of options on how a function is run.

```
CREATE or REPLACE FUNCTION function_name ( parameter_list varchar )
RETURNS varchar
AS $$
DECLARE
    data text;
BEGIN

    -- do something

    RETURN data;
END;
$$ LANGUAGE plpgsql;
```

The easy way to call a function is with a "select". For example:

```
select function_name ( 'a' );
```

Stored procedures have some advantages. The code is run inside or near to the database. This makes the cost of moving data back and forth very low. Stored procedures are a well developed and optimized technology and tend to be fast.

With PL/pgSQL as the language the data types exactly match with the database and the handling of values like NULL is clear and easy.

We can check to see if a select fails to return data and for nulls in a fashion that is built into the language.

```
CREATE or REPLACE FUNCTION function_name ( parameter_list varchar )
RETURNS varchar
AS $$
DECLARE
    data text;
BEGIN

    SELECT 'PASS'
```

```
        INTO data
        FROM ct_config
        WHERE config_id = 1;
    IF not found THEN
        data = 'FAIL';
    END IF;

    RETURN data;
END;
$$ LANGUAGE plpgsql;
```

In PostgreSQL a stored procedure creates a transaction at the "BEGIN" and ends it at the "END". This means that if you "RAISE" an error in the middle it will result in rolling back the entire transaction.

Stored procedures can return rows of data. Note the fact that the loop returns a single row and then restart's for each additional row.

For example:

```
-- using table from hw13_4.sql us_state

CREATE OR REPLACE FUNCTION getStateFipsCode()
RETURNS SETOF us_state
AS $$
DECLARE
    r us_state%rowtype;
BEGIN
    FOR r IN
        SELECT * FROM us_state
        WHERE gdp_growth > 1.0
    LOOP

        -- can do some processing here

        RETURN NEXT r; -- return current row of SELECT
    END LOOP;
    RETURN;
END
$$
LANGUAGE 'plpgsql' ;

SELECT *
    FROM getStateFipsCode()
;
```