Week 5 Tutorial Notes

▼ Agenda

Q1 → simple programming language task

Q3 → setof integer

Q4 → harder programming language task

Q7,8,9,10 → beers db

Q12 → SQL and PLpgSQL functions

Pre-Tute

- Assignment 1
- No Quiz
- Help sessions - not running 😢

SQL Views

```
CREATE VIEW view_name AS
SELECT
   *
FROM
   table_name
WHERE
   (conditions)
;
```

SQL Functions

```
CREATE OR REPLACE FUNCTION func_name(paramType1, paramType2, ...) RETU
AS $$
    -- single SQL query
$$ LANGUAGE SQL;
```

Key characteristics

- A single SQL statement with no RETURN → just outputs the result of the query.
- No named parameters → uses positional parameter notation: $1, $2, $3, ...
- Can only return what an SQL query can return (atomic value, single row, table)

PLpgSQL Functions

```
CREATE OR REPLACE FUNCTION func_name(param1 Type1, param2 Type2, ...) R
AS $$
DECLARE
    -- declare variables here
    var1  varType;
    var2  varType  := defaultValue;
    ...
BEGIN
    -- programming logic goes here
END;
$$ LANGUAGE PLpgSQL;
```

Key characteristics:

- Allows actual programming logic
- Parameters are named
- Requires RETURN

    ▼ Normal RETURN

```
-- To print all suppliers who supplies all parts of a given colour (returning
CREATE OR REPLACE SuppliesAllParts(_colour text) RETURNS text
AS $$
DECLARE
   _sid    text;
   _ret    text := '';
BEGIN
   -- loop through tuples of the sql query
   for _sid in
      select
         S.sid
      from
         Suppliers S
      where not exists (
         (select P.pid from Parts P where P.colour = _colour)
         except
         (select C.pid from Catalog C where C.sid = S.sid)
      )
   loop
      _ret := _ret || _sid || e'\n';
   end loop;

   return _ret;
END;
$$ LANGUAGE plpgsql;
```

▼ RETURN setof

```
-- If we wanted to do the same thing for a function returning setof text
CREATE OR REPLACE SuppliesAllParts(_colour text) RETURNS setof text
AS $$
DECLARE
   _sid     text;
BEGIN
   for _sid in
```

```
        select
            S.sid
        from
            Suppliers S
        where not exists (
            (select P.pid from Parts P where P.colour = _colour)
            except
            (select C.pid from Catalog C where C.sid = S.sid)
        )
    loop
        return next _sid;
    end loop;

    return;
END;
$$ LANGUAGE plpgsql;
```

- PERFORM vs SELECT
  - ▼ PERFORM: For queries that you want to perform but don't want to store

```
CREATE OR REPLACE SuppliesPart(partial_name text) RETURNS text
AS $$
DECLARE
    _pid    integer;
BEGIN
    perform
        *
    from
        Parts
    where
        pname ILIKE '%'||partial_name||'%' -- Case-insensitive partial string
    ;
    -- typically you do this for when you want to check valid input.
    -- when you do a perform or select query, it updates a keyword called
    -- outputted some tuples and false if no tuples
```

```
    if not found then
        return 'No part matches';
    end if;

    ...
END;
$$ LANGUAGE plpgsql;
```

▼ SELECT: For when you want to store variables

```
CREATE OR REPLACE SuppliesPart(partial_name text) RETURNS text
AS $$
DECLARE
    _pid    integer;
BEGIN
    -- Note that below we assume the output of the query will be
    -- a single value, not multiple tuples (multiple matches)
    select
        pid -- which column of data
    into
        _pid -- which variable we're storing into
    from
        Parts
    where
        pname ILIKE '%'||partial_name||'%'
    ;
    if not found then
        return 'No part matches';
    end if;
    -- now we can use _pid as a variable later in the code.

    ...
END;
$$ LANGUAGE plpgsql;
```