# Advanced Management of Data
## Exercise 3 Topic 2:
# Extensions of SQL

# Control statements
# IF, THEN, ELSE

- the `hello(`*`name`*`)` function doesn't return anything, if the *`name`* is `NULL`
- Task: rewrite the function and check the input parameter
  - if it is `NULL` then return the same as the `hello()` function without parameter does

# Control statements
# IF, THEN, ELSE

```
CREATE OR REPLACE FUNCTION hello(name VARCHAR) RETURNS VARCHAR AS $$
  BEGIN
    IF name IS NULL THEN  -- also possible: IF name ISNULL THEN
      RETURN hello();
    ELSE
      RETURN format('Hello, %s!', name);
    END IF;
  END;
$$ LANGUAGE plpgsql;
```

- this is much better, but empty strings still look ugly
- Task: rewrite your function to use “Anonymous” as name for empty strings

# Control statements
# IF, THEN, ELSE

```
CREATE OR REPLACE FUNCTION hello(name VARCHAR) RETURNS VARCHAR AS $$
  BEGIN
    IF name IS NULL THEN
      RETURN hello();
    ELSEIF name = '' THEN
      RETURN hello('Anonymous');
    ELSE
      RETURN format('Hello, %s!', name);
    END IF;
  END;
$$ LANGUAGE plpgsql;
```

- Task: now, also check for “Bob” and use “Robert” instead and change “Bill” to “William”, too

# Control statements
# IF, THEN, ELSE

```
CREATE OR REPLACE FUNCTION hello(name VARCHAR) RETURNS VARCHAR AS $$
  BEGIN
    IF name IS NULL THEN
      name = 'World';
    ELSEIF name = '' THEN
      name = 'Anonymous';
    ELSEIF name = 'Bob' THEN
      name = 'Robert';
    ELSEIF name = 'Bill' THEN
      name = 'William';
    END IF;
    RETURN format('Hello, %s!', name);
  END;
$$ LANGUAGE plpgsql;
```

- Task: rewrite it and try to use the CASE control structure instead

# Control statements
# CASE, WHEN, THEN, ELSE

```
CREATE OR REPLACE FUNCTION hello(name VARCHAR) RETURNS VARCHAR AS $$
  BEGIN
    CASE                          -- using CASE name WHEN ... WHEN ... is not possible here
      WHEN name IS NULL THEN      -- because you can't check for NULL with this construct
        name = 'World';
      WHEN name = '' THEN
        name = 'Anonymous';
      WHEN name = 'Bob' THEN
        name = 'Robert';
      WHEN name = 'Bill' THEN
        name = 'William';
      ELSE                        -- without this ELSE only the mentioned cases would work and all other cases would result in an error
        NULL;
    END CASE;
    RETURN format('Hello, %s!', name);
  END;
$$ LANGUAGE plpgsql;
```

# Control statements
# CASE, WHEN, THEN, ELSE

- use `CASE` just in case you want to test something for different values and don't want to check for `NULL`
- else it is more complicated than `IF`
- Task: write a new rating function that
  - takes an integer as input parameter
  - and returns a rating string
    - "poor" for 1, 2 and 3
    - "average" for 4, 5 and 6
    - "good" for 7, 8 and 9
    - "excellent" for 10
    - "out of range" for anything else
  - and uses the `CASE` construct

# Control statements
# CASE, WHEN, THEN, ELSE

```
CREATE OR REPLACE FUNCTION rating(score INTEGER) RETURNS VARCHAR AS $$
  BEGIN
    CASE score
      WHEN 1, 2, 3 THEN
        return 'poor';
      WHEN 4, 5, 6 THEN
        return 'average';
      WHEN 7, 8, 9 THEN
        return 'good';
      WHEN 10 THEN
        return 'excellent';
      ELSE
        return 'out of range';
    END CASE;
  END;
$$ LANGUAGE plpgsql;
```

- Task: write a new function that executes this rating function with input values in range from 0 to 20 and use the `FOR` control construct

# Control statements
# FOR

```
CREATE OR REPLACE FUNCTION rating_test() RETURNS SETOF VARCHAR AS $$
  BEGIN
    FOR score IN 0..20 LOOP -- score is defined as local INTEGER
      RETURN NEXT rating(score); -- NEXT can return multiple rows
    END LOOP;
    RETURN; -- this indicates that there are no more rows
  END;      -- but as we have already reached the END this is not needed
$$ LANGUAGE plpgsql;
```

- Task: now write another function that returns random `INTEGER` values in a range of 1 to 100 and exits when the number 42 is returned
  - each number should be returned at its own row
  - hint: the built-in function `random()` returns `DOUBLE PRECISION` values in a range of [0.0 .. 1.0)

# Control statements
# LOOP

```
CREATE OR REPLACE FUNCTION random42() RETURNS SETOF INTEGER AS $$
  DECLARE
    rnd INTEGER;
  BEGIN
    LOOP
      rnd = ceil(random() * 100)::INTEGER; -- INTEGER casts are rounding
      RETURN NEXT rnd;
      EXIT WHEN rnd = 42;  -- the same as: IF rnd = 42 THEN EXIT; END IF;
    END LOOP;              -- also possible: IF rnd = 42 THEN RETURN; END IF;
  END;
$$ LANGUAGE plpgsql;
```

- Task: let's write another function, that
  - counts the number of random numbers (number of rows) returned by our last function and
  - if this count doesn't equal to 42, increases another counter that indicates how many tries it took to get 42 random numbers and
  - returns this counter

# Control statements
# WHILE

```
CREATE OR REPLACE FUNCTION count42() RETURNS INTEGER AS $$
  DECLARE
    c INTEGER = 1;
  BEGIN
    WHILE (SELECT COUNT(*) FROM random42()) != 42 LOOP
      c = c + 1; -- there is no c++ or ++c
    END LOOP;
    RETURN c;
  END;
$$ LANGUAGE plpgsql;
```