

Introduction to PL/pgSQL and Triggers

Tom Kelliher, CS 317

1 PL/pgSQL

1.1 Structure of a PL/pgSQL Procedure

PL/pgSQL is block structured:

```
[ Label ]
[ DECLARE
    declarations ]
BEGIN
    statements;
END;
```

Example:

```
DROP FUNCTION somefunc();

-- The double $ delimits the beginning/end of a function's code.
CREATE FUNCTION somefunc() RETURNS INTEGER AS $$
DECLARE
    quantity INTEGER := 30;
BEGIN
    RAISE NOTICE 'Quantity here is %',quantity; -- Quantity here is 30
    quantity := 50;
    --
    -- Create a sub-block
    --
    DECLARE
        quantity INTEGER := 80;
    BEGIN
        RAISE NOTICE 'Quantity here is %',quantity; -- Quantity here is 80
    END;
```

```

    RAISE NOTICE 'Quantity here is %',quantity; -- Quantity here is 50
    RETURN quantity;
END;
$$ LANGUAGE 'plpgsql';

-- Run the function.
SELECT somefunc();

```

1.2 Variables

1. Has all SQL types.
2. Possible to create tuple variables using %ROWTYPE attribute. Example showing this and also control structures:

```

DROP FUNCTION Test(sid INTEGER);

CREATE FUNCTION Test(sid INTEGER) RETURNS INTEGER AS $$
DECLARE

    StudRec  Student%ROWTYPE;
    TransRec Transcript%ROWTYPE;
    Count    INTEGER := 0;

BEGIN

    -- SELECT result must be a single tuple.

    SELECT INTO StudRec *
    FROM Student S
    WHERE S.Id = sid;

    FOR TransRec IN SELECT * FROM Transcript T
    WHERE StudRec.Id = T.StuId
    LOOP
        Count := Count + 1;
    END LOOP;

    IF Count < 10 THEN
        RAISE NOTICE '% has taken too few courses', StudRec.Name;
    END IF;

    RETURN count;
END;
$$ LANGUAGE 'plpgsql';

```

```
SELECT Test(666666666);
```

1.3 Features Specific to Trigger Procedures

1. Must be a function with no parameters and a return type of **TRIGGER**.
2. Special variables automatically created in the top-level block:
 - (a) **NEW** — new tuple value on UPDATE/INSERT row level triggers.
 - (b) **OLD** — old tuple value on UPDATE/DELETE row level triggers.
3. Must either return **NULL** (or execute **RAISE EXCEPTION**) or a tuple matching the structure of the relation the trigger was called on.
 - (a) **BEFORE** triggers return **NULL** to signal that the operation for this tuple should be skipped, **RAISE EXCEPTION** to abort the transaction, return a modified result, or do nothing (return **NEW** unchanged) to allow the intended result.
 - (b) **AFTER** triggers can return **NULL** with no effect.
4. Example:

```
CREATE FUNCTION RaiseCheck () RETURNS TRIGGER AS $$
BEGIN
    IF NEW.Salary > 1.05 * OLD.Salary THEN
        -- Excessive salary raise - limit it.
        RAISE NOTICE '% given an excessive raise', OLD.EmpName;
        NEW.Salary := 1.05 * OLD.Salary;
    END IF;
    -- Salary increase OK, proceed with transaction.
    RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER LimitRaises BEFORE UPDATE ON Employee
FOR EACH ROW EXECUTE PROCEDURE RaiseCheck();
```