1 Administrivia

Announcements

Assignment

Read 11.1–4.

From Last Time

Triggers.

Outline

1. PL/pgSQL.

2. Lab.

Coming Up

Physical data organization.
2 PL/pgSQL

2.1 Structure of a PL/pgSQL Procedure

PL/pgSQL is block structured:

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

Example:

```sql
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;
  END;
  RAISE NOTICE ''Quantity here is %'',quantity; -- Quantity here is 50
END;
' LANGUAGE 'plpgsql';
```

2.2 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(INTEGER);

CREATE FUNCTION Test(INTEGER) RETURNS INTEGER AS 'DECLARE

    StudRec  Student%ROWTYPE;
    TransRec Transcript%ROWTYPE;
    Count    INTEGER := 0;
    Id       ALIAS FOR $1;

BEGIN

    -- SELECT result must be a single tuple.
    SELECT INTO StudRec *
    FROM   Student S
    WHERE  S.Id = Id;

    -- Aggregate functions appear not to work. Manually iterate over
    -- entire result set to compute count.

    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);
```

2.3 Features Specific to Trigger Procedures

1. Must be a function with no parameters and a return type of OPAQUE.
2. Special variables automatically created in the top-level block:

(a) \texttt{NEW} — new tuple value on UPDATE/INSERT row level triggers.

(b) \texttt{OLD} — old tuple value on UPDATE/DELETE row level triggers.

3. Must either return \texttt{NULL} (or execute \texttt{RAISE EXCEPTION}) or a tuple matching the structure of the relation the trigger was called on.

(a) \texttt{BEFORE} triggers return \texttt{NULL} to signal that the operation for this tuple should be skipped, \texttt{RAISE EXCEPTION} to abort the transaction, return a modified result, or do nothing (return \texttt{NEW} unchanged) to allow the intended result.

(b) \texttt{AFTER} triggers can return \texttt{NULL} with no effect.

4. Example:

```sql
CREATE FUNCTION RaiseCheck () RETURNS OPAQUE 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();
```

3 Lab

Refer to handout.