COMP 543: Tools & Models for Data Science
Imperative SQL

Chris Jermaine & Risa Myers

Rice University

RICE

? Why is this useful?

- Why useful?
    - Encapsulation — make it easy for the programmer
    - Safety — protect the database from the programmer
    - Performance — fewer end-to-end trips
    - Reuse
    - Can process records sequentially
    - Can respond to events

- Common form of imperative code: stored procedure and / or function
    - Code stored in the DB
    - Can be invoked from
        - The command line
        - An external program
        - Another stored procedure / function
        - A trigger

```
CREATE FUNCTION myFunc(/* list params */)
    RETURNS <datatype> AS
$$
DECLARE
/* declarations here */
BEGIN
/* code here */
END;
$$
LANGUAGE plpgsql ;
```

First Stored Procedure Example

```
PEAK (NAME, ELEV, DIFF, MAP, REGION)
```

? Write a stored procedure to get the peak count in a region
  - But if no region given...
  - Return the count overall

First Stored Procedure Example

```
PEAK (NAME, ELEV, DIFF, MAP, REGION)

CREATE OR REPLACE FUNCTION getNumPeaks(whichRegion CHARACTER VARYING)
  RETURNS INTEGER AS
$$
DECLARE
        /* declarations here */
BEGIN
        /* code here */
END;
$$
LANGUAGE plpgsql;
```

First Stored Procedure Example

```
CREATE OR REPLACE FUNCTION
   getNumPeaks(whichRegion character varying)
   RETURNS integer AS
$$
DECLARE
  queryString TEXT;
  numPeaks INTEGER;
BEGIN
  numPeaks = 0;
  -- build the query
  queryString = 'SELECT COUNT(*) FROM peak ' ||
      COALESCE (' WHERE Region=' ||
          QUOTE_LITERAL(whichRegion) ,'');
  -- run the query
  EXECUTE queryString INTO numPeaks;
  RETURN numPeaks;
END;
$$
LANGUAGE plpgsql;
```

- All local variables need to be declared
- Code is bounded by **$$**
- Language is specified at the end
- ? What's the deal with `QUOTE_LITERAL`?
- ? What's `COALESCE`?
- ? `EXECUTE`: common, powerful, dangerous! Why?

Then to call

```sql
SELECT * FROM getNumPeaks('Corocoran_to_Whitney');
```

and

```sql
SELECT * FROM getNumPeaks(NULL);
```

- Like before, but now we'll use a cursor

```
CREATE OR REPLACE FUNCTION
    getNumPeaks(whichRegion CHARACTER VARYING) RETURNS INTEGER AS
$$
DECLARE
  myCursor REFCURSOR;
  queryString TEXT;
  numPeaks INTEGER;
BEGIN
  numPeaks = 0;
  -- build the query
  queryString = 'SELECT COUNT(*) FROM peak ' || COALESCE (' WHERE Region=' ||
      QUOTE_LITERAL(whichRegion) ,'');
  -- run the query
  OPEN myCursor FOR EXECUTE queryString;
  FETCH  myCursor INTO numPeaks;
  CLOSE myCursor;
  RETURN numPeaks;
END;
$$
LANGUAGE plpgsql;
```

```
BEGIN
  numPeaks = 0;
  -- build the query
  queryString = 'SELECT COUNT(*) FROM peak ' || COALESCE (' WHERE Region=' ||
      QUOTE_LITERAL(whichRegion) ,'');
  -- run the query
  OPEN myCursor FOR EXECUTE queryString;
  FETCH  myCursor INTO numPeaks;
  CLOSE myCursor;
  RETURN numPeaks;
END;
```

- What's new here: a "cursor"
    - Standard abstraction for dealing with record sets
    - Essentially an iterator

Steps for Using a Cursor

1. Declare
2. Open
3. Loop
   1. Fetch
   2. Check for done
   3. Repeat from 3.1
4. Close

Cursor Options / Variations

- `FETCH / MOVE` **options**
  - `NEXT / PRIOR`
  - `FIRST / LAST`
  - `ABSOLUTE / RELATIVE` **n**
  - `FORWARD / BACKWARD` **n**
  - ...
- Bound vs. Unbound

  myCursor `CURSOR FOR SELECT * FROM` **peak;**
- Explicit vs. Implicit

  `FOR` **rowData** `IN SELECT ...   LOOP ...   END LOOP`

  `SELECT INTO...`

- It's primitive
- Use `RAISE NOTICE '%', <var>`
- e.g. `RAISE NOTICE '%',` queryString
- Start simple
- Consider writing data to an external (temporary table)

Questions?