SCHARP

at FRED HUTCH

PostgreSQL Bulk Commands

and examples of how to think in bulk

By Lloyd Albin

SCHARP at FRED HUTCH

Principals

- If you use the principals described in the following slides, you will be able to make your SQL more efficient and thereby execute faster allowing you to be able to handle larger volumes of data.
- There are examples at the end of the presentation where I took SQL and PL/PGSQL and refactored it using these principals to speed up the code by a factor of 100 or more times faster.

SCHARP at FRED HUTCH

What we are going to cover

- Commands
  - *SELECT*
  - *TRUNCATE*
  - *DELETE*
  - *CREATE TABLE AS*
  - *COPY*
  - *INSERT INTO*
  - *INSERT (Multi Row)*
  - *UPDATE using INSERT*
  - *UPDATE FROM*
  - *UPDATE/INSERT CTE*
  - *INSERT / DO UPDATE*
- Examples
  - *Update comparison and why updating everything can be faster than a partial update.*
  - *A good example of how to refactor your functions for speed.*

SCHARP at FRED HUTCH

SELECT

This one should be obvious

SELECT

- Everyone should know SELECT allows us to retrieve multiple rows via a single command.
- What people may not realize is that you can do this with many other commands also, such as INSERT, UPDATE, DELETE AND COPY.

```
SELECT * FROM table1;
```

TRUNCATE

Bulk Delete the Entire Table

TRUNCATE

- While truncate is very fast, it also puts an exclusive lock on the table until your transaction is finished.
- This means that any other reads of this table will be delayed until your transaction is finished.
- Truncate also removes all the contents of the table and all indexes, thereby removing any bloat that can happen when deleting and then re-entering the same records.

```
TRUNCATE TABLE table_a;
```

SCHARP at FRED HUTCH

DELETE

Bulk Delete part of the Table

DELETE (from Query)

- While delete is a slower process, it allows other transactions to read the deleted rows until the transaction finishes.
- Most people use “field1 = x”, the problem with this is that x is a fixed value.
- You may use a sub-query to find all the results you wish to use as keys to perform the delete, allowing you to replace the “field1 = x” with a list of values.

```
DELETE FROM table1;

DELETE FROM table1 WHERE field1 = x;

DELETE FROM table1 WHERE field1 NOT IN
(SELECT field1 FROM table1 WHERE field2 = x);

DELETE FROM table1 WHERE field1 IN
(SELECT field1 FROM table1 WHERE field2 = x);
```

SCHARP at FRED HUTCH

CREATE TABLE AS

Create the Table and Bulk Insert
the Data as a Single Command

SCHARP
at FRED HUTCH

CREATE TABLE AS (with Bulk Insert)

- The SELECT may be as complex as you want.
- The new table will automatically use the field types defined via any table(s) used in the query.
- After the CREATE TABLE AS, you can then create any index's, primary key's, etc.

```
CREATE TABLE table_a AS
SELECT field_name, … FROM table_b;
```

SCHARP at FRED HUTCH

COPY

Bulk Inserting from a FILE or STDIN
Bulk Exporting to a FILE or STDOUT

COPY (Bulk Insert)

- When you are coping FROM a file, this requires you to be a superuser, although you can wrap this inside a function where the function is owned and runs under the superuser's privileges.
- When coping FROM STDIN, you can be any user.
- This is the best method when needing to write hundreds of lines at a time.

```
COPY table_name (field_name, …) FROM 'filename';
COPY table_name (field_name, …) FROM STDIN;
COPY table_name (field_name, …) FROM STDIN (DELIMITER '|', HEADER TRUE);
COPY table_name (field_name, …) FROM STDIN (FORMAT CSV, HEADER TRUE);
COPY table_name (field_name, …) FROM PROGRAM 'gunzip  filename.gz';
```

SCHARP at FRED HUTCH

COPY (Bulk Export)

- When you are coping TO a file, this requires you to be a superuser, although you can wrap this inside a function where the function is owned and runs under the superuser's privileges.
- When coping TO STDOUT, you can be any user.
- This is the best method when needing to write hundreds of lines at a time.

```
COPY table_name (field_name, …) TO 'filename';
COPY table_name (field_name, …) TO STDOUT;
COPY table_name (field_name, …) TO STDOUT (DELIMITER '|', HEADER TRUE);
COPY table_name (field_name, …) TO STDOUT (FORMAT CSV, HEADER TRUE);
COPY table_name (field_name, …) TO PROGRAM 'gzip > filename.gz';
```

```
COPY (SELECT * FROM table1) TO 'filename';
COPY (SELECT * FROM table1) TO STDOUT;
COPY (SELECT * FROM table1) TO STDOUT (DELIMITER '|', HEADER TRUE);
COPY (SELECT * FROM table1) TO STDOUT (FORMAT CSV, HEADER TRUE);
COPY (SELECT * FROM table1) TO PROGRAM 'gzip > filename.gz';
```

SCHARP at FRED HUTCH

COPY (via psql)

- We can use the copy command via psql to import or export a CSV file.
- The postgres dump files use the COPY command to load the data back into the database either using psql or pg_restore.

```
psql -h hostname -d database -c 'COPY schema.table TO
STDOUT (FORMAT CSV, HEADER TRUE)' > file.csv

psql -h hostname -d database -c 'COPY schema.table FROM
STDIN (FORMAT CSV, HEADER TRUE)' < file.csv
```

SCHARP at FRED HUTCH

COPY (via Driver)

- The copy command is available via some *but not all* drivers and some are *into PostgreSQL only*.
- .NET Drivers
  - NPGSQL
    - BeginBinaryImport
    - BeginBinaryExport
    - BeginTextImport
    - BeginTextExport
    - BeginRawBinaryCopy
    - BeginRawBinaryCopy
  - Devart dotConnect for PostgreSQL
    - PgSqlLoader
- Java Drivers
  - JDBC
    - copyIn
    - copyOut
- `R Drivers`
  - `RPostgreSQL`
    - postgresqlCopyIn
    - postgresqlCopyInDataframe
- `Perl`
  - `DBD::Pg`
    - pg_putcopydata
    - pg_getcopydata
- `Python`
  - `Psycopg2`
    - copy_to
    - copy_from
    - copy_expert
- `Ruby`
  - `PG::Connection`
    - put_copy_data
    - get_copy_data
- `Go`
  - `lib/pq`
    - CopyIn

SCHARP at FRED HUTCH

COPY (via Other Programs)

There are some other 3rd party tools that help you do this also.

- Pgloader
  - http://pgloader.io/
- pg_bulkload
  - http://ossc-db.github.io/pg_bulkload/index.html

SCHARP at FRED HUTCH

INSERT INTO

Bulk Inserting from another Table/Query

INSERT INTO (Bulk Insert)

- This command will insert all results from simple to complex queries into an existing table.
- One great example of this is inserting columns of data from a normalized table into 3rd normal form tables or into key value table.
- For example if you have 78 fields that all need to get written to one key value table, you can do this with 78 INSERT INTO's instead of 78 * the number of rows using normal INSERT statements. For a 1,000 rows this would be 78,000 normal INSERT's making it a lot faster to perform just 78 INSERT INTO's.

```
INSERT INTO table_a
SELECT field_name, … FROM table_b;
```

SCHARP at FRED HUTCH

INSERT (Multi Row)

Bulk Inserting

INSERT (Bulk Insert)

- The multi row insert is a big time saver over single row inserts.
- For JAVA you can set reWriteBatchedInserts=true to enable JAVA to convert single row inserts into multi row inserts. This option has been available since 9.4.1208 of the JDBC driver.
- https://jdbc.postgresql.org/documentation/94/connect.html

```
-- Single Row Insert Example
INSERT INTO pg_mug (id, item) VALUES (1, 'coffee');
INSERT INTO pg_mug (id, item) VALUES (2, 'sugar');
INSERT INTO pg_mug (id, item) VALUES (3, 'cream');

-- Multi Row Insert Example
INSERT INTO pg_mug (id, item)
VALUES (1, 'coffee'), (2, 'sugar'), (3, 'cream');
```

SCHARP at FRED HUTCH

UPDATE using INSERT

Method 1 – Update Records via Insert

UPDATE using CREATE TABLE AS and INSERT

- Make all foreign keys to the table deferrable.
- BEGIN your transaction
- Create a temp table using CREATE TABLE AS with SELECT
- DELETE rows to be updated
- INSERT updated rows
- DROP the temp table
- COMMIT your transaction, now the foreign key relationship are evaluated to make sure there is no bad references.

```
ALTER TABLE table_d
  ADD CONSTRAINT table_d_id_fk FOREIGN KEY (id)
     REFERENCES table_c(id)
     ON DELETE NO ACTION
     ON UPDATE NO ACTION
     DEFERRABLE
     INITIALLY DEFERRED;

BEGIN;

CREATE TABLE table_a AS
SELECT field_name, … FROM table_c, table_b;

DELETE FROM table_c WHERE id IN
(SELECT id FROM table_a);

INSERT INTO table_c
SELECT field_name, … FROM table_a;

COMMIT;
```

SCHARP at FRED HUTCH

UPDATE FROM

Method 2 – Update from a Query

UPDATE FROM (Query)

- Normal updates allow you to set the same value, formula, or function result for multiple rows.
- UPDATE FROM allows you to set unique values for each row being updated.
- For example you could update all customer records with their latest invoice number, etc.

```
UPDATE mytable1 mt
SET field1 = a.field1,
    field2 = a.field2
FROM (
    SELECT b.id, b.field1, c.field2
    FROM mytable1 b
    LEFT JOIN mytable2 c
        ON b.id = c.id
) a
WHERE a.id = mt.id
```

SCHARP at FRED HUTCH

UPDATE or INSERT

How to update or insert
a record as a single command
via Writeable CTE

UPDATE / INSERT CTE (Common Table Expressions)

- This example shows how to perform a Bulk UPDATE OR INSERT as a single command.
- The Values command can be replaced by a SELECT simple to complex Query.
- Requires Postgres 9.1 or newer.
- Limitation: It is possible to run into race conditions when a second process has inserted the same new record between the first process's UPDATE and INSERT.

```
WITH new_values (id, field1, field2) AS (
  VALUES
     (1, 'A', 'X'),
     (2, 'B', 'Y'),
     (3, 'C', 'Z')
),
upsert AS
(
    UPDATE mytable m
    SET field1 = nv.field1,
        field2 = nv.field2
    FROM new_values nv
    WHERE m.id = nv.id
    RETURNING m.*
)
INSERT INTO mytable (id, field1, field2)
SELECT id, field1, field2
FROM new_values
WHERE NOT EXISTS (SELECT 1
    FROM upsert up
    WHERE up.id = new_values.id);
```

SCHARP at FRED HUTCH

UPDATE / INSERT CTE (Common Table Expressions)

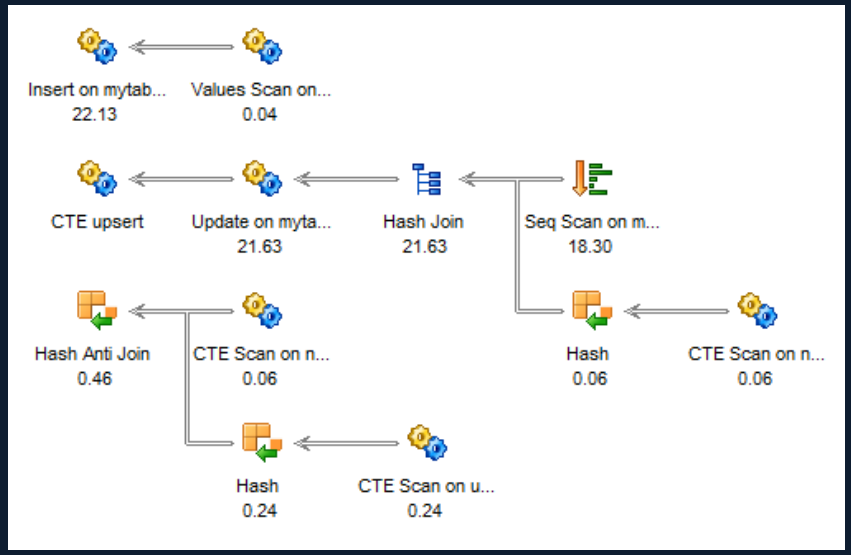
- Here is what is happening behind the scenes.

```
CREATE TABLE mytable (
  id INTEGER,
  field1 TEXT,
  field2 TEXT
);

INSERT INTO mytable VALUES (1, NULL, NULL);
INSERT INTO mytable VALUES (2, NULL, NULL);

-- Run Command from the Previous Page

SELECT * FROM mytable;
```



id	field1	field2
1	A	X
2	B	Y
3	C	Z

SCHARP at FRED HUTCH

UPDATE / INSERT CTE (Common Table Expressions)

More information on Writeable CTE's

- http://www.postgresql.org/docs/9.3/static/queries-with.html
- http://stackoverflow.com/questions/1109061/insert-on-duplicate-update-in-postgresql/6527838#6527838
- http://www.xzilla.net/blog/2011/Mar/Upserting-via-Writeable-CTE.html
- http://www.depesz.com/2011/03/16/waiting-for-9-1-writable-cte/
- http://www.depesz.com/2012/06/10/why-is-upsert-so-complicated/

SCHARP at FRED HUTCH

INSERT / DO UPDATE

How to Insert/Update PostgreSQL 9.5+

INSERT / DO UPDATE

- Here is the new ON CONFLICT part of the INSERT command in PostgreSQL 9.5+.
- There is also a DO NOTHING in case you don't wish to do an update if the INSERT fails.

```
INSERT INTO mytable (id, field1, field2)
  VALUES
     (1, 'A', 'X'),
     (2, 'B', 'Y'),
     (3, 'C', 'Z')
--ON CONFLICT ON CONSTRAINT table_field_pkey DO UPDATE
ON CONFLICT (did) DO UPDATE
SET field1 = EXCLUDED.field1,
    field2 = EXCLUDED.field2;
```

SCHARP at FRED HUTCH

UPDATE comparison

Actual Examples

CREATE TABLE AS (Example)

- Here we generate our results that we want to insert/update and write them to a temp table.
- This first part is the records to updated. The “a” section is the new records and the “b” section is the existing record to get the variable_record_id.

```
CREATE UNLOGGED TABLE temp_qc_record_96_21 AS
  SELECT a.qc_record_id, a.center_number, a.creation_time,
    a.duplicate, a.modification_time, a.name, a.note,
    a.page_number, a.plate_number, a.problem_code, a.ptid,
    a.qc_field_number, a.query, a.raster_id, a.refax_code,
    a.reply_to_query, a.report_number, a.resolution_time, a.status,
    a.df_study_number, a.usage_code, a.validation_level, a.value,
    a.variable_number, a.visit_number, b.variable_record_id
  FROM (

    SELECT DISTINCT ON (df_study_number, plate_number, ptid,
                        variable_number, visit_number) *
    FROM data.qc_record
    WHERE df_study_number = '96'::text AND plate_number = '21'::text
    ORDER BY df_study_number, plate_number, ptid, variable_number,
             visit_number, qc_record_id DESC

  ) a
  LEFT JOIN
  (

    SELECT DISTINCT ON (df_study_number, plate_number, ptid,
                        variable_number, visit_number) *
    FROM data.variable_record
    WHERE df_study_number = '96'::text
      AND plate_number = '21'::text
    ORDER BY df_study_number, plate_number, ptid, variable_number,
             visit_number, variable_record_id DESC

  ) b
  USING (df_study_number, plate_number, ptid, variable_number,
         visit_number)

  UNION ALL
```

SCHARP at FRED HUTCH

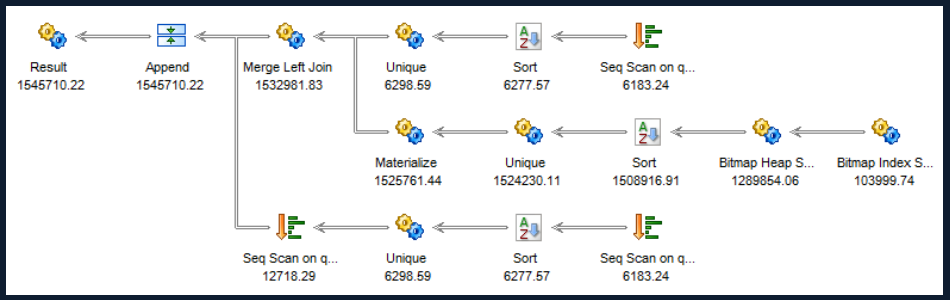
CREATE TABLE AS (Example)

- Here we generate our results that we want to insert/update and write them to a temp table.
- This second part is the new records. We make sure that the record does not already exist, aka updated.
- We now have a temp table with all the records to be inserted and updated with all their keys.

```
SELECT a.qc_record_id, a.center_number, a.creation_time,
       a.duplicate, a.modification_time, a.name, a.note,
       a.page_number, a.plate_number, a.problem_code, a.ptid,
       a.qc_field_number, a.query, a.raster_id, a.refax_code,
       a.reply_to_query, a.report_number, a.resolution_time,
       a.status, a.df_study_number, a.usage_code,
       a.validation_level, a.value, a.variable_number,
       a.visit_number, NULL::BIGINT AS variable_record_id
FROM data.qc_record a
WHERE a.df_study_number = '96'::text AND plate_number = '21'::text
      AND a.qc_record_id NOT IN (

  SELECT DISTINCT ON (df_study_number, plate_number, ptid,
    variable_number, visit_number) qc_record_id
  FROM data.qc_record
  WHERE df_study_number = '96'::text AND plate_number = '21'::text
  ORDER BY df_study_number, plate_number, ptid, variable_number,
           visit_number, qc_record_id DESC

);
```

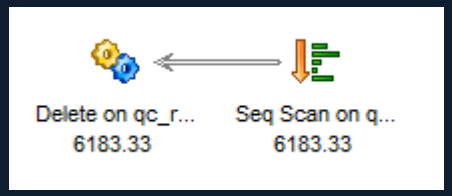


SCHARP at FRED HUTCH

DELETE (Example)

- Now we delete the rows that we will be updating from our table.
- Because other tables have foreign key relationships to this table, we must set the constraints as DEFERRABLE. Enabling us to delete and replace the record without causing any constraint issues.

```
DELETE FROM data.qc_record
WHERE df_study_number = '96'::text
      AND plate_number = '21'::text;
```

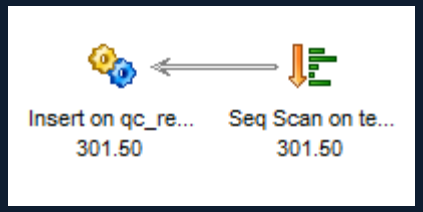


SCHARP at FRED HUTCH

INSERT INTO (Example)

- Instead of doing the update, we insert from our temp table into the production table.
- One benefit for this method is that the time for the DELETE and INSERT is very fast. This means that the LOCKS will released much faster.
- With this single insert, we have done all the INSERT's and UPDATE's.

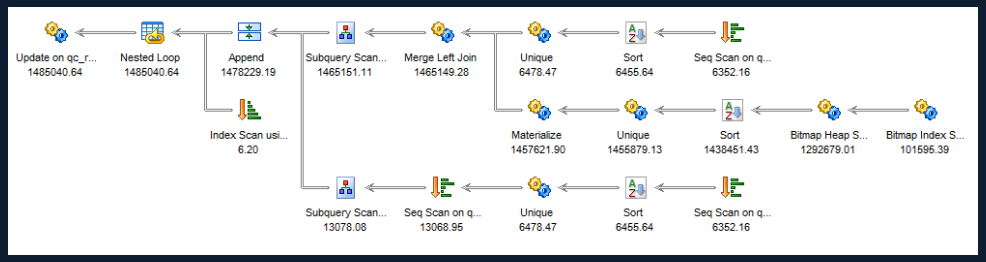
```
INSERT INTO data.qc_record
SELECT * FROM temp_qc_record_96_21;
```



SCHARP at FRED HUTCH

UPDATE FROM (Example - All)

- You will notice that the UPDATE FROM (24.134s) was actually faster than the CREATE TABLE AS (38.079s) with DELETE (78ms), INSERT (47ms) and COMMIT (31ms).
- This is partly due to not needing to return all the fields for the UPDATE. We only need the key field and the fields being updated.

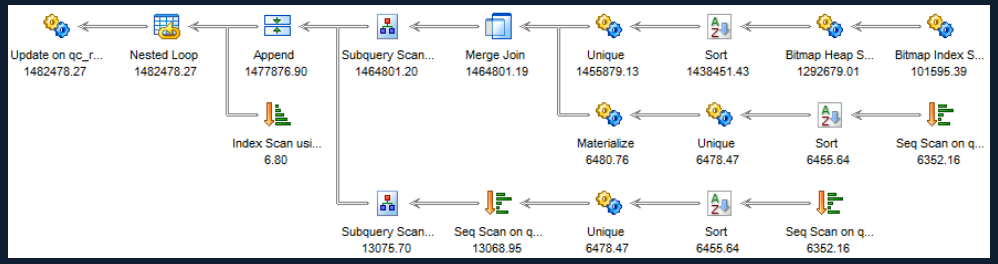



```
UPDATE data.qc_record qcr
  SET variable_record_id = tqcr.variable_record_id, …
  FROM (
    SELECT a.qc_record_id, …, b.variable_record_id
    FROM (
      SELECT DISTINCT ON (df_study_number, plate_number, ptid,
                          variable_number, visit_number)
             df_study_number, plate_number, ptid, variable_number,
             visit_number, qc_record_id, variable_record_id
      FROM data.qc_record
      WHERE df_study_number = '96'::text
            AND plate_number = '21'::text
      ORDER BY df_study_number, plate_number, ptid,
               variable_number, visit_number, qc_record_id DESC
    ) a
    LEFT JOIN (
      SELECT DISTINCT ON (df_study_number, plate_number, ptid,
                          variable_number, visit_number)
             df_study_number, plate_number, ptid, variable_number,
             visit_number, variable_record_id
      FROM data.variable_record
      WHERE df_study_number = '96'::text
            AND plate_number = '21'::text
      ORDER BY df_study_number, plate_number, ptid,
            variable_number, visit_number, variable_record_id DESC
    ) b
    USING (df_study_number, plate_number, ptid, variable_number,
           visit_number)
    UNION ALL
    SELECT a.qc_record_id, …, NULL::BIGINT AS variable_record_id
    FROM data.qc_record a
    WHERE a.df_study_number = '96'::text
      AND plate_number = '21'::text AND a.qc_record_id NOT IN (
      SELECT DISTINCT ON (df_study_number, plate_number, ptid,
        variable_number, visit_number) qc_record_id
      FROM data.qc_record
      WHERE df_study_number = '96'::text
            AND plate_number = '21'::text
      ORDER BY df_study_number, plate_number, ptid, variable_number,
               visit_number, qc_record_id DESC)
    ) AS tqcr
    WHERE qcr.qc_record_id = tqcr.qc_record_id;
```

SCHARP at FRED HUTCH

UPDATE FROM (Example - Filtered)

- If you added where causes to both sides of the UNION ALL so that it restricts the update to only records that need to be updated, this increases the execution time to 46.878s from 24.134s.
- So it is actually faster to update all the records.




```
UPDATE data.qc_record qcr
  SET variable_record_id = tqcr.variable_record_id, …
  FROM (
    SELECT a.qc_record_id, …, b.variable_record_id
    FROM (
      SELECT DISTINCT ON (df_study_number, plate_number, ptid,
                          variable_number, visit_number)
            df_study_number, plate_number, ptid, variable_number,
            visit_number, qc_record_id, variable_record_id
      FROM data.qc_record
      WHERE df_study_number = '96'::text
            AND plate_number = '21'::text
      ORDER BY df_study_number, plate_number, ptid,
               variable_number, visit_number, qc_record_id DESC
    ) a LEFT JOIN (
      SELECT DISTINCT ON (df_study_number, plate_number, ptid,
                          variable_number, visit_number)
            df_study_number, plate_number, ptid, variable_number,
            visit_number, variable_record_id
      FROM data.variable_record
      WHERE df_study_number = '96'::text
            AND plate_number = '21'::text
      ORDER BY df_study_number, plate_number, ptid,
            variable_number, visit_number, variable_record_id DESC
    ) b
    USING (df_study_number, plate_number, ptid, variable_number,
           visit_number)
    UNION ALL
    SELECT a.qc_record_id, …, NULL::BIGINT AS variable_record_id
    FROM data.qc_record a
    WHERE a.df_study_number = '96'::text
      AND plate_number = '21'::text AND
      a.variable_record_id = IS NOT NULL AND a.qc_record_id NOT IN (
        SELECT DISTINCT ON (df_study_number, plate_number, ptid,
          variable_number, visit_number) qc_record_id
        FROM data.qc_record
        WHERE df_study_number = '96'::text
              AND plate_number = '21'::text
        ORDER BY df_study_number, plate_number, ptid,
                 variable_number, visit_number, qc_record_id DESC)
    ) AS tqcr
    WHERE qcr.qc_record_id = tqcr.qc_record_id;
```

SCHARP at FRED HUTCH

Re-Factor Example #1

Re-Factor Functions – Real World Example

Example #1 – Original Code

- This is the function that used to take 10 minutes and has now increased to 3½ hours the md5 hash's were added.
- The code for this function is to long to publish, so we will be looking at parts of it as we re-factor each section.
- So our first job is to fix the hashing speed, any idea's?

```
-- Process Each Row of raw standard enrollment data
#####################################################--
FOR raw_row IN (SELECT * FROM daids_es.std_enr_data_raw) LOOP
  n_total := n_total + 1;
  -- Assign various data elements casting as necessary
  -- to correct the SAS ODBC driver type definitions
  ---------------------------------------------------------
  new_row.network := raw_row.network;
  new_row.site_affiliation := raw_row.site_affiliation;
  new_row.protocol := raw_row.protocol;
  new_row.ptid := raw_row.ptid::bigint;
  new_row.gender := raw_row.gender;
  new_row.genidm := raw_row.genidm;
  new_row.genidf := raw_row.genidf;
  new_row.genidtm := raw_row.genidtm;
  new_row.genidtf := raw_row.genidtf;
  new_row.genido := raw_row.genido;
  new_row.genidds := raw_row.genidds;
  new_row.sex := raw_row.sex;
  new_row.race := raw_row.race;
  new_row.raceotxt := raw_row.raceotxt;
  new_row.nihwhite := raw_row.nihwhite::int;
  new_row.nihblack := raw_row.nihblack::int;
  new_row.nihasian := raw_row.nihasian::int;
  new_row.nihnativ := raw_row.nihnativ::int;
  new_row.nihhawpi := raw_row.nihhawpi::int;
  new_row.nihother := raw_row.nihother::int;
  new_row.hispanic := raw_row.hispanic::int;
  new_row.birthdt := raw_row.birthdt;
  new_row.birthmm := raw_row.birthmm::int;
  new_row.birthdd := raw_row.birthdd::int;
  new_row.birthyy := raw_row.birthyy::int;
  new_row.bdt_est := raw_row.bdt_est::int;
  new_row.age := raw_row.age::int;
  new_row.enrolldt := raw_row.enrolldt;
  new_row.sitedfno := raw_row.sitedfno::int;
  new_row.site := raw_row.site;
  new_row.ptidmom := raw_row.ptidmom::bigint;
  new_row.ptidprim := raw_row.ptidprim::bigint;
  new_row.hvtnpart := raw_row.hvtnpart;
  new_row.hvtngrp := raw_row.hvtngrp;
  new_row.inactdt := raw_row.inactdt;
  new_row.inactrsn := raw_row.inactrsn;
  new_row.txstatus := raw_row.txstatus ;
  new_row.txstopdt := raw_row.txstopdt;
  new_row.daids_site_id := raw_row.daids_site_id;
  new_row.study_status_md5 := daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_study_status', raw_row.ptid::bigint);
  new_row.steps_md5 := daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_step_history', raw_row.ptid::bigint);
  new_row.site_enlistments_md5 := daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_site_enlist', raw_row.ptid::bigint);
  new_row.interventions_md5 := daids_es.calculate_md5_of_data_for_ptid('daids_es','int_hist', raw_row.ptid::bigint);
```

SCHARP at FRED HUTCH

Re-Factor Example #2

Re-Factor Function – calculate_md5_of_data_for_ptid

Example #2 – Original Code

- Here is the function that was written. By itself is fast, but when executed many times, it becomes very slow.
- For any rows found in the specified table matching the given PTID, collapse all the data contained in those rows into a single text field, then calculate and return the md5 hash of that text field in UUID format. The purpose of this is to help detect whether any of the data for this PTID in the specified table has changed since the last time the md5 hash value was calculated.
- How would you speed this function up using the principals that I showed before?

```
CREATE OR REPLACE FUNCTION daids_es.calculate_md5_of_data_for_ptid (
  _schema_name text,
  _table_name text,
  _ptid bigint
)
RETURNS uuid AS $body$
-- EX:
-- SELECT
daids_es.calculate_md5_of_data_for_ptid('daids_es','combined_step_hist',
701229883);
--
DECLARE
  columns_string text := NULL;
  full_table_name text := NULL;
  col RECORD;
  select_query text := NULL;
  rows_to_text_query text := NULL;
  md5_query text := NULL;
  md5_value UUID := NULL;
BEGIN
  FOR col IN SELECT *
    FROM information_schema.columns
    WHERE table_schema = _schema_name AND table_name = _table_name
    ORDER BY ordinal_position
  LOOP
    columns_string = concat_ws(',',columns_string, col.column_name::text);
  END LOOP;
  full_table_name = _schema_name || '.' || _table_name;
  select_query = 'SELECT concat_ws('','','' || columns_string || ') FROM '
|| full_table_name || ' WHERE ptid = ' || _ptid || ' ORDER BY ' ||
columns_string;
  rows_to_text_query = 'array_to_string(ARRAY(' || select_query || '),
'','')';
  md5_query = 'SELECT md5(' || rows_to_text_query || ')::uuid';
  EXECUTE md5_query INTO md5_value;
  RETURN md5_value;
END $body$ LANGUAGE 'plpgsql' VOLATILE CALLED ON NULL INPUT SECURITY
INVOKER;
```

SCHARP at FRED HUTCH

Example #2 – Re-Factored Code

- Here is how you can re-factor it to be faster by having it return the results for all ptid's in the table instead of calling this function for each ptid.
- Ptid = Patient ID.
- Any ideas on how much of a speed increase we will see?

```
CREATE OR REPLACE FUNCTION daids_es.calculate_md5_of_data_for_ptid (
   _schema_name text,
  _table_name text
)
RETURNS TABLE (
  ptid bigint,
  md5 uuid
) AS $body$
-- EX:
-- SELECT
daids_es.calculate_md5_of_data_for_ptid('daids_es','combined_step_hist');
--
DECLARE
  columns_string text := NULL;
  full_table_name text := NULL;
  col RECORD;
  select_query text := NULL;
  rows_to_text_query text := NULL;
  md5_query TEXT;
BEGIN
  FOR col IN SELECT *
    FROM information_schema.columns
    WHERE table_schema = _schema_name AND table_name = _table_name
    ORDER BY ordinal_position
  LOOP
    columns_string = concat_ws(',',columns_string, col.column_name::text);
  END LOOP;
  full_table_name = _schema_name || '.' || _table_name;
  select_query = 'SELECT ptid, concat_ws('','',' || columns_string || ')
FROM ' || full_table_name;
  rows_to_text_query = 'array_to_string(array_agg("concat_ws" ORDER BY
"concat_ws"), '','')';
  md5_query = 'SELECT ptid::bigint, md5(' || rows_to_text_query ||
')::uuid FROM (' || select_query || ') a GROUP BY ptid';
  RETURN QUERY EXECUTE md5_query;
END $body$ LANGUAGE 'plpgsql' VOLATILE CALLED ON NULL INPUT SECURITY
INVOKER;
```

SCHARP at FRED HUTCH

Example #2 – Speed Increase

- As you can see it is much faster now. Over 1 hour, but we were estimating about 3 hours in production, down to 2¼ seconds.
- The speed was increased by a minimum of 790 but we suspect it was really increased by 2,371 times.

```
SELECT DISTINCT ON (ptid) ptid,
daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_study_status',
ptid::bigint) FROM daids_es.sas_study_status;
-- 55208 rows returned (execution time: 00:16:20; total time: 00:16:20)
SELECT DISTINCT ON (ptid) ptid,
daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_step_history',
ptid::bigint) FROM daids_es.sas_step_history;
-- 117 rows returned (execution time: 1.061 sec; total time: 1.061 sec)
SELECT DISTINCT ON (ptid) ptid,
daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_site_enlist', ptid::bigint)
FROM daids_es.sas_site_enlist;
-- Unknown, This would be another long one like the one below.
SELECT DISTINCT ON (ptid) ptid,
daids_es.calculate_md5_of_data_for_ptid('daids_es','int_hist', ptid::bigint) FROM
daids_es.int_hist;
-- 46+ minutes and still going
-- Over 1 Hour Total

SELECT * FROM
daids_es.calculate_md5_of_data_for_ptid_lloyd('daids_es','sas_study_status');
-- 55208 rows returned (execution time: 655 ms; total time: 686 ms)
SELECT * FROM
daids_es.calculate_md5_of_data_for_ptid_lloyd('daids_es','sas_step_history');
-- 117 rows returned (execution time: 15 ms; total time: 15 ms)
SELECT * FROM
daids_es.calculate_md5_of_data_for_ptid_lloyd('daids_es','sas_site_enlist');
-- 51502 rows returned (execution time: 468 ms; total time: 468 ms)
SELECT * FROM daids_es.calculate_md5_of_data_for_ptid_lloyd('daids_es','int_hist');
-- 87638 rows returned (execution time: 1.139 sec; total time: 1.155 sec)
-- Total 2.277 sec
```

SCHARP at FRED HUTCH

Re-Factor Example #1

Re-Factor Functions – Finishing this refactor

Example #1 – Re-Factored Code

- Here is how we use the re-factored hashing function inside of our main code.
- How much of a speed increase do you think this will have?

```
FOR raw_row IN (
    SELECT r.network,
       r.site_affiliation,
       r.protocol,
       r.ptid,
       r.gender,
       r.genidm,
       r.genidf,
       r.genidtm,
       r.genidtf,
       r.genido,
       r.genidds,
       r.sex,
       r.race,
       r.raceotxt,
       r.nihwhite,
       r.nihblack,
       r.nihasian,
       r.nihnativ,
       r.nihhawpi,
       r.nihother,
       r.hispanic,
       r.birthdt,
       r.birthmm,
       r.birthdd,
       r.birthyy,
       r.bdt_est,
       r.age,
       r.enrolldt,
       r.sitedfno,
       r.site,
       r.ptidmom,
       r.ptidprim,
       r.hvtnpart,
       r.hvtngrp,
       r.inactdt,
       r.inactrsn,
       r.txstatus,
       r.txstopdt,
       r.daids_site_id,
       a.md5 AS study_status_md5,
       b.md5 AS steps_md5,
       c.md5 AS site_enlistments_md5,
       d.md5 AS interventions_md5
    FROM daids_es.std_enr_data_raw r
    LEFT JOIN daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_study_status') a USING (ptid)
    LEFT JOIN daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_step_history') b USING (ptid)
    LEFT JOIN daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_site_enlist') c USING (ptid)
    LEFT JOIN daids_es.calculate_md5_of_data_for_ptid('daids_es','int_hist') d USING (ptid)
    )
    LOOP
```

SCHARP at FRED HUTCH

Example #1 – Re-Factored Code

- This changed it from running in 3½ hours back down to the original 10 minutes.
- It is now 10½ times faster that it was originally written.

Re-Factor Example #3

Re-Factor Functions

Example #3 – Original Code

- Here we are grabbing the existing row of data and doing comparison checks on it to the current row of data to see if it needs to be:
  - Undeleted
  - Deleted
  - Updated
  - No Action (count only)
- Any ideas on how to make this faster?

```
-- Look for an existing entry in the pt_accrual table
SELECT INTO exist_row *
  FROM daids_es.pt_accrual
  WHERE raw_row.network = daids_es.pt_accrual.network
    AND raw_row.protocol = daids_es.pt_accrual.protocol
    AND raw_row.ptid = daids_es.pt_accrual.ptid;
  GET DIAGNOSTICS exist_row_cnt = ROW_COUNT;
-- Insert the data as necessary
-- Separate logic based on whether the data already exists in the pt_accrual table
  IF exist_row_cnt = 1 THEN
-- Handle the rare case in which a pt previously marked as delete suddenly appears
in the raw data again
    IF exist_row.delete = 'true' THEN
      -- Reset the delete flag
      -- Update the timestamp
      -- Update pt_accrual table
      -- Log the activity --> Will be used to notify externally
      new_row.DELETE = 'false';
      PERFORM daids_es.pt_update(new_row);
      PERFORM daids_es.log_it('update_pt_accrual',
              exist_row.network,
              exist_row.protocol,
              exist_row.ptid::bigint,
              'REMOVING DELETE FLAG!');
      n_update := n_update +1;
      n_undel := n_undel +1;
    ELSE -- Determine if anything has changed in the data
-- we don't care about differences in the date_last_modified and delete columns
      exist_row.date_last_modified := new_row.date_last_modified;
      exist_row.delete := new_row.delete;
      IF (new_row IS DISTINCT FROM exist_row) THEN
        -- Update the pt_accrual table
        PERFORM daids_es.pt_update(new_row);
        n_update := n_update +1;
      ELSE
        -- If the data did NOT change DO NOTHING
        n_ignore := n_ignore +1;
      END IF;
    END IF;
  ELSE
    -- Insert the new data
    new_row.DELETE = 'false';
    PERFORM daids_es.pt_insert(new_row);
    n_insert := n_insert + 1;
  END IF;
END LOOP;
--Process Each Row of raw standard enrollment data
```

SCHARP at FRED HUTCH

Example #3 – Re-Factor Part 1

- Join the original data that you have to do separate queries to get it.
- Then converted all the IF statements into CASE statements and made them part of the main query.
- Also notice that instead of looping through the data, write it into a temp table. This was it can be dealt with in bulk.
- Any idea's of what to do next?

```
CREATE TEMP TABLE tmp_raw_table AS SELECT r.network,
       r.site_affiliation,
       ……
       r.daids_site_id,
       a.md5 AS study_status_md5,
       b.md5 AS steps_md5,
       c.md5 AS site_enlistments_md5,
       d.md5 AS interventions_md5,
       pa."delete",
       pa.date_last_modified,
       CASE WHEN pa.ptid IS NULL THEN 1 -- new_record -- IF exist_row_cnt = 1 THEN
WHEN pa."delete" = 'true' THEN 2 -- undelete_record -- IF exist_row.delete = 'true' THEN
WHEN (ROW(
r.network, r.site_affiliation, r.protocol, r.ptid, r.gender, r.genidm, r.genidf, r.genidtm,
r.genidtf, r.genido, r.genidds, r.sex, r.race, r.raceotxt, r.nihwhite, r.nihblack,
r.nihasian,r.nihnativ, r.nihhawpi, r.nihother, r.hispanic, r.birthdt, r.birthmm, r.birthdd,
r.birthyy, r.bdt_est, r.age, r.enrolldt, r.sitedfno, r.site, r.ptidmom, r.ptidprim, r.hvtnpart,
r.hvtngrp, r.inactdt, r.inactrsn, r.txstatus, r.txstopdt, r.daids_site_id, a.md5, b.md5, c.md5,
d.md5
       ) IS DISTINCT FROM ROW(
pa.network, pa.site_affiliation, pa.protocol, pa.ptid, pa.gender, pa.genidm, pa.genidf, pa.genidtm,
pa.genidtf, pa.genido, pa.genidds, pa.sex, pa.race, pa.raceotxt, pa.nihwhite, pa.nihblack,
pa.nihasian, pa.nihnativ, pa.nihhawpi, pa.nihother, pa.hispanic, pa.birthdt, pa.birthmm, pa.birthdd,
pa.birthyy, pa.bdt_est, pa.age, pa.enrolldt, pa.sitedfno, pa.site, pa.ptidmom, pa.ptidprim,
pa.hvtnpart, pa.hvtngrp, pa.inactdt, pa.inactrsn, pa.txstatus, pa.txstopdt, pa.daids_site_id,
pa.study_status_md5, pa.steps_md5, pa.site_enlistments_md5, pa.interventions_md5
       )) THEN 3 -- update -- (new_row IS DISTINCT FROM exist_row)
ELSE 4 -- ignore
END AS record_action
    FROM daids_es.std_enr_data_raw r
    LEFT JOIN daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_study_status') a USING (ptid)
    LEFT JOIN daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_step_history') b USING (ptid)
    LEFT JOIN daids_es.calculate_md5_of_data_for_ptid('daids_es','sas_site_enlist') c USING (ptid)
    LEFT JOIN daids_es.calculate_md5_of_data_for_ptid('daids_es','int_hist') d USING (ptid)
    LEFT JOIN daids_es.pt_accrual pa USING (network, protocol, ptid);
GET DIAGNOSTICS n_total = ROW_COUNT;
```

SCHARP at FRED HUTCH

Example #3 – Re-Factor Part 2

- This is using one of the principals that I taught you earlier about inserting in bulk.
- Instead of counting each row inserted in the look we use the GET DIAGNOSTICS to find out how many rows we inserted at once.

```
-- New Record
INSERT INTO daids_es.pt_accrual (
network, protocol, ptid, gender, race, raceotxt, nihwhite, nihblack,
nihasian, nihnativ, nihhawpi, nihother, hispanic, birthdt, birthmm,
birthdd, birthyy, bdt_est, age, enrolldt, sitedfno, site, ptidmom,
ptidprim, inactdt, inactrsn, txstatus, txstopdt, date_last_modified,
"delete", sex, genidm, genidf, genidtm, genidtf, genido, genidds,
daids_site_id, site_affiliation, hvtnpart, hvtngrp, study_status_md5,
steps_md5, site_enlistments_md5, interventions_md5
)

  SELECT network,   protocol, ptid, gender, race, raceotxt, nihwhite,
nihblack, nihasian, nihnativ, nihhawpi, nihother, hispanic, birthdt,
birthmm, birthdd, birthyy, bdt_est, age , enrolldt, sitedfno, site,
ptidmom, ptidprim, inactdt, inactrsn, txstatus, txstopdt,
CURRENT_TIMESTAMP AS date_last_modified, false::boolean AS "delete", sex,
genidm, genidf, genidtm, genidtf, genido, genidds, daids_site_id,
site_affiliation, hvtnpart, hvtngrp, study_status_md5, steps_md5,
site_enlistments_md5, interventions_md5
  FROM tmp_raw_table
  WHERE record_action = 1;


GET DIAGNOSTICS n_insert = ROW_COUNT;
```

SCHARP at FRED HUTCH

Example #3 – Re-Factor Part 3

- This is using one of the principals that I taught you earlier about updating in bulk.
- Instead of counting each row inserted in the look we use the GET DIAGNOSTICS to find out how many rows we inserted at once.
- Notice how I having it logging and updating in the same single statement.

```
-- Undelete Record
UPDATE daids_es.pt_accrual pa
SET network = a.network,
          site_affiliation = a.site_affiliation,
          protocol = a.protocol,
          ptid = a.ptid,
          daids_site_id = a.daids_site_id,
          ……
          study_status_md5 = a.study_status_md5,
          steps_md5 = a.steps_md5,
          site_enlistments_md5 = a.site_enlistments_md5,
          interventions_md5 = a.interventions_md5,
          "delete" = false,
          date_last_modified = a.date_last_modified
      FROM (
            SELECT *, daids_es.log_it('update_pt_accrual'::text,
                                      network::text,
                                      protocol,
                                      ptid::bigint,
                                      'REMOVING DELETE FLAG!'::text)
            FROM tmp_raw_table
            WHERE record_action = 2
      ) a
      WHERE pa.network = a.network AND
            pa.protocol = a.protocol AND
            pa.ptid = a.ptid;

GET DIAGNOSTICS n_undel = ROW_COUNT;
```

SCHARP at FRED HUTCH

Example #3 – Re-Factor Part 4

- This is using one of the principals that I taught you earlier about updating in bulk.
- Instead of counting each row inserted in the look we use the GET DIAGNOSTICS to find out how many rows we inserted at once.
- I also added the undelete to the update count, just like in the original code.

```
-- Undelete Record
UPDATE daids_es.pt_accrual pa
SET network = a.network,
          site_affiliation = a.site_affiliation,
          protocol = a.protocol,
          ptid = a.ptid,
          daids_site_id = a.daids_site_id,
          ……
          study_status_md5 = a.study_status_md5,
          steps_md5 = a.steps_md5,
          site_enlistments_md5 = a.site_enlistments_md5,
          interventions_md5 = a.interventions_md5,
          "delete" = a."delete",
          date_last_modified = a.date_last_modified
      FROM (
            SELECT *,
            FROM tmp_raw_table
            WHERE record_action = 3
      ) a
      WHERE pa.network = a.network AND
            pa.protocol = a.protocol AND
            pa.ptid = a.ptid;

GET DIAGNOSTICS n_update = ROW_COUNT;
n_update := n_undel + n_update;
```

SCHARP at FRED HUTCH

Example #3 – Re-Factor Part 5

- Here we just count the records and that is all we need to do.

```
-- Do Nothing – Record is Current
SELECT count(*) INTO ignore_records
  FROM tmp_raw_table WHERE record_action = 4;
n_ignore := ignore_records.count;
```

SCHARP at FRED HUTCH

Example #3 – Original Code

- Here we are finding records that should be flagged as deleted as they are no longer in the current data set.
- We need to update these records and log that we are deleting these records.
- Any idea's on how to do this? I have shown how to do this one in the previous examples!

```
--#########################################################################--
-- Look for participants that should be marked as DELETE -
--#########################################################################--
FOR del_row IN (SELECT network,protocol,ptid
                    FROM daids_es.pt_accrual
                      WHERE delete != 'true'
                        EXCEPT
                          SELECT network,protocol,ptid
                            FROM daids_es.std_enr_data_raw) LOOP
       -- Set delete flag to true
UPDATE daids_es.pt_accrual
         SET date_last_modified = CURRENT_TIMESTAMP,
             delete = 'true'
         WHERE network = del_row.network AND
               protocol = del_row.protocol AND
               ptid = del_row.ptid;
       -- log the activity --> Will be used to notify me nightly
PERFORM daids_es.log_it('update_pt_accrual'::text,
                    del_row.network::text,
                    del_row.protocol::double precision,
                    del_row.ptid::bigint,
                    'DELETING PARTICIPANT!'::text);
       n_del := n_del + 1;
    END LOOP;
```

SCHARP at FRED HUTCH

Example #3 – Re-Factor Part 6

- This is using one of the principals that I taught you earlier about updating in bulk.
- Instead of counting each row inserted in the look we use the GET DIAGNOSTICS to find out how many rows we inserted at once.
- Notice how I having it logging and updating in the same single statement.

```
--#########################################################################--
-- Look for participants that should be marked as DELETE -
--#########################################################################--
UPDATE daids_es.pt_accrual pa
       SET date_last_modified = CURRENT_TIMESTAMP,
           delete = 'true'
           -- Set delete flag to true
FROM (
             SELECT network,protocol,ptid,
                    daids_es.log_it('update_pt_accrual'::text,
                                    network::text,
                                    protocol::double precision,
                                    ptid::bigint,
                                    'DELETING PARTICIPANT!'::text)
             -- log the activity --> Will be used to notify me nightly
FROM (
                   SELECT network,protocol,ptid
                          FROM daids_es.pt_accrual
                          WHERE delete != 'true'
                   EXCEPT
                   SELECT network,protocol,ptid
                          FROM daids_es.std_enr_data_raw
             ) a
       ) b
       WHERE pa.network = b.network AND
             pa.protocol = b.protocol AND
             pa.ptid = b.ptid;
GET DIAGNOSTICS n_del = ROW_COUNT;
```

SCHARP at FRED HUTCH

Example #3 – Speed Increase

- As you can see it is much faster now. Almost 11 minutes down to less than 6½ seconds.
- The speed was increased by 52 times.
- Overall the re-factoring took a 3½ hour function down to under 6½ seconds. That's a speed increase of 497 times!!!
- Was this worth re-factoring, YES! The amount of time saved by refactoring is made up in less than a week. Plus the code is fast enough that the programmers can now test with it multiple times a day instead of only making changes once a day and then running the test overnight.

```
BEGIN;
SELECT daids_es.update_pt_accrual();
-- TOTAL:64591 INSERT:20 UPDATE:64571 IGNORE:0 DELETE:0 UNDELETE:0
-- 1 rows returned (execution time: 00:10:52; total time: 00:10:52)
ROLLBACK;

BEGIN;
SELECT daids_es.update_pt_accrual();
-- TOTAL:64591 INSERT:20 UPDATE:64571 IGNORE:0 DELETE:0 UNDELETE:0
-- 1 rows returned (execution time: 6.334 sec; total time: 6.334 sec)
ROLLBACK;
```

SCHARP at FRED HUTCH

THANK YOU

SCHARP
at FRED HUTCH