Usefull procedures for data migration

Often you run into situation when you need to change DB structure, which results in replicating tables from one table to another. Usually algo is following.

1. Adding logic to a code that start writing to a new table. Release.
2. Replicating old data to new table.
3. Changing read usages in code to a new table. Release.
4. Droping old data after few releases.

Very often you can't just replicate all data using one query cause it can lead to long locks. Here is procedures that may be useful.

Updating table untill some condition is met

```
CREATE OR REPLACE PROCEDURE uam.migrateData(get_query text, update_query text)
LANGUAGE plpgsql
AS $$
declare
    ids_to_update int[];
    counter int:= 0;
begin
    execute concat('SELECT ARRAY(', get_query, ')') into ids_to_update;
    WHILE array_length(ids_to_update, 1) > 0 LOOP

    execute format(update_query, array_to_string(ids_to_update, ','));
    raise info 'Iteration executed: %', counter;
    counter := counter + 1;
    COMMIT;
    execute concat('SELECT ARRAY(', get_query, ')') into ids_to_update;
    END LOOP ;
END;
$$;
```

```
-- set type
call uam.migrateData(
    'SELECT pm.id from payment.payment_methods pm where pm.type is null AND pm."method"->>''type''  is not null LIMIT 10',
    'UPDATE payment.payment_methods SET type = "method"->>''type'' WHERE id in (%s)'
);
```

Using cursor to iterate some result set

This one is not a universal procedure and must be created per case, cause row type can't be set via function params.

```
CREATE OR REPLACE PROCEDURE uam.migrate_accounts_old(get_query text, update_query text) LANGUAGE plpgsql AS
$$
DECLARE
   c refcursor := 'curs';
   /* set correct table type */
   selected_val uam.accounts%rowtype;
   counter int:= 0;
BEGIN
   /* dynamic SQL to create the cursor */
   EXECUTE format('DECLARE curs CURSOR WITH HOLD FOR %s FOR READ ONLY', get_query);
   LOOP
      FETCH c INTO selected_val;

      EXIT WHEN NOT FOUND;

      /*
       * We need to make sure that the cursor is closed
       * in the case of an error.  For that, we need an
       * extra block, because COMMIT cannot be used in
       * a block with an EXCEPTION clause.
       */
      BEGIN
          EXECUTE update_query using selected_val;
         /* avoid SQL injection */
      EXCEPTION
         WHEN OTHERS THEN
            CLOSE c;
            RAISE;
         WHEN query_canceled THEN
            CLOSE c;
            RAISE;
      END;
      COMMIT;

```

```
      /* log execution */
      IF counter % 100 = 0 then
        RAISE INFO 'Items processed: %', counter;
      END IF;
      counter := counter + 1;
   END LOOP;
   RAISE INFO 'Items processed: %', counter;

   /* we need to close the cursor */
   CLOSE c;
END;
$$;
```

```
call migrate_accounts(
'SELECT * FROM uam.accounts where admin is true',
'UPDATE uam.accounts SET locked = true WHERE id = $1.id'
);
```

Using cursor to iterate some result set and inserted id

```
CREATE OR REPLACE PROCEDURE iteratePaymentMethodsPayWithMyBank() LANGUAGE plpgsql AS
$$DECLARE
   get_query text:= 'select pm.* from payment.payment_methods pm
               where  pm.ach_payment_method_id is null
                  and pm."method"->>''type'' = ''PayWithMyBank''
                  and pm."method" is not null';
   insert_query text:= 'INSERT INTO payment.ach_payment_method(account_name, account_number, account_routing_number, account_type, bank_name, payment_provider_id,
created_at)
                                                   VALUES(
                                                          $1."method"->>''accountName'',
                                                          $1."method"->>''accountNumber'',
                                                          $1."method"->>''accountRoutingNumber'',
                                                          CASE WHEN ($1."method"->>''accountType'')::int= 1 THEN ''checking''
                                                                 ELSE ''saving''
                                                                 END,
                                                          $1."method"->>''paymentProviderName'',
                                                          $1."method"->>''paymentProviderId'',
                                                          current_timestamp
                                                   ) RETURNING id';
   update_query text:= 'update payment.payment_methods pm set ach_payment_method_id = %s where pm.id = %s';
   c refcursor := 'curs';
   insertedId payment.card_payment_method.id%TYPE;
     /* set correct table type */
   selected_val payment.payment_methods%rowtype;
   counter int:= 0;
BEGIN
   /* dynamic SQL to create the cursor */
   EXECUTE format('DECLARE curs CURSOR WITH HOLD FOR %s FOR READ ONLY', get_query);
   LOOP
      FETCH c INTO selected_val;

      EXIT WHEN NOT FOUND;

      /*
       * We need to make sure that the cursor is closed
       * in the case of an error.  For that, we need an
       * extra block, because COMMIT cannot be used in
       * a block with an EXCEPTION clause.
       */
      begin
          EXECUTE insert_query INTO insertedId using selected_val;
          EXECUTE format(update_query,insertedId, selected_val.id);
        /* avoid SQL injection */
      EXCEPTION
         WHEN OTHERS THEN
            CLOSE c;
            RAISE;
         WHEN query_canceled THEN
            CLOSE c;
            RAISE;
      END;
      COMMIT;

      /* log execution */
      IF counter % 100 = 0 then
        RAISE info 'Items processed: %', counter;
      END IF;
      counter := counter + 1;
   END LOOP;

```

```
   /* we need to close the cursor */
   CLOSE c;
END;$$;
```

```
call iteratePaymentMethodsPayWithMyBank();
```

```
DO $$
 DECLARE
    c refcursor := 'curs';
    curr_reward uam.bonus_rewards%rowtype;
    found_campaign_id bigint;
    counter int:= 0;
    nf_counter int:= 0;
    offer_code text;
    declare get_affected_rewards_query text := 'select * from uam.bonus_rewards where campaign_id in (699, 700, 701) and at between ''2024-06-03''::date and ''2024-06-05''::date' ;--b2prod
BEGIN
    EXECUTE format('DECLARE curs CURSOR WITH HOLD FOR %s FOR READ ONLY', get_affected_rewards_query);
    LOOP
        FETCH c INTO curr_reward;
        EXIT WHEN NOT FOUND;

        BEGIN
            select offer.code into offer_code
                from payment.payment_orders o
                join payment.offer_templates offer on o.offer_id = offer.id
                                                                  where (o.success = true or chargeback_status='chargeback')--nuivei
                                                                    and o.account_id = curr_reward.account_id
                                                                    and o.at = curr_reward.at
                                                                    and o.sc_amount - o.amount = curr_reward.sweepstake_amount
                order by curr_reward.created_at - o.created_at limit 1;-----??
            if NOT FOUND then
                nf_counter = nf_counter + 1;
                RAISE info 'NOT FOUND OFFER FOR REWARD: %', curr_reward.id;
            else
                counter = counter + 1;
                select id into found_campaign_id from uam.reward_campaigns where code=offer_code and brand_id = (select acc.brand_id from uam.accounts acc where acc.id = curr_reward.account_id);
                if NOT FOUND then
                    RAISE info 'CAMPAIGN NOT FOUND with code %', offer_code ;
                else
                    UPDATE uam.bonus_rewards set campaign_id = found_campaign_id where id = curr_reward.id;
                end if;
            end if;

        EXCEPTION
            WHEN OTHERS THEN
                CLOSE c;
                RAISE;
            WHEN query_canceled THEN
                CLOSE c;
                RAISE;
        END;
        COMMIT;
        IF counter % 1000 = 0 then
            RAISE info 'debug - rewards processed: %', counter;
        END IF;
    END LOOP;
    CLOSE c;
    raise info 'total processed rewards %', counter::text;
    raise info 'total NOT processed rewards %', nf_counter::text;
END;$$;
```