Trying to gain peace of mind by using constraints

Ivan Frolkov

postgrespro.ru

Database tasks

- Reliable data storage
- Convenient access
- **Business process synchronization**

Business process synchronization

- Prevent a single operation on an object from being done multiple times.
  - The refund has been sent to the client twice.
- Prevent loss of objects.
  - Unsent orders accumulate in the warehouse.
- Prevent new objects from appearing
  - The product specified in the database can’t be found in the warehouse.
- Other things
  - The uniqueness of people, debit = credit, 1.5 excavators, etc.

How can they avoid this?

- They need to determine in what state the database **cannot** be.
- To prohibit these states, we need

Integrity Constraints

Integrity Constraints

- Prohibit explicitly incorrect states.
- "Affect performance negatively”:
  - **Well, let it go down.**
  - They need to check **everything.**
    - They need to check it twice.
    - It is better if the checks are written by different people.
- "Inconvenient and inflexible”.
- I only saw foreign key and not null :-(

What it could be?

- not null - often
- check - rarely
- unique/primary key – almost always
  - unique constraint can be deferrable, index cannot be deferrable
  - Tables without a primary key are also common.
- foreign key
  - on delete cascade/set null/default
- exclude
- json schema – **not implemented!**
- Triggers

Granularity of constraints

- String – null, check
  - null/check cannot be deferrable, which is contrary to the standard. However, no one complained.
- Query – primary key, unique, exclude, trigger.
- Transaction - primary key, unique, exclude, trigger
  - deferrable
    - deferrable trigger = constraint trigger

Assertions

- CREATE ASSERTION is a part of the standard.
- Unfortunately, not implemented yet...
  - It's not clear when it will be implemented.
  - ...and whether it will be at all.
  - ...it is not implemented anywhere at all.
- It would be nice:
  - ```
    create assertion limit_debts as
    check((select sum(amount) from debt1)+(select
    sum(amount) from debt2)<1000000)
    ```
  - Etc.
    ```

Triggers

- There should be no unexpected actions in triggers.
- Checks and aggregation only
- Keep in mind parallelism and synchronize
  - Via the auxiliary table (e.g. amounts by group)
  - Via advisory locks

Auxiliary tables

- sales(id, date, amount)
- month_sales(int year>2019 && <=now(), int month between 1 and 12, sales_count>0, total_amount>0, min_amount>0, max_amount>0…)
  - Since the string is blocked during insert / update, no errors will occur during parallel execution.
  - Productivity drops: we either get the wrong result quickly, or slowly get the right one.
- May be useful on its own
- Another trigger is to prevent manual changes (pg_trigger_level())

Only five orders per client

- client(id int, …)
  order(id, client_id,..)

- ```
  create or replace function trg_only_5() returns trigger as
  $code$
  begin
    if tg_op='DELETE' or tg_op='UPDATE' and old.client_id=new.client_id then
      return;
    end if;
    perform from client c where c.id=new.client_id for update;
    if (select count(*) from order o where o.client_id=new.client_id)>5 then
       raise sqlstate '23U01' using message='Too many orders';
    end if;
    return new;
  end
  $code$
  language plpgsql;

  create or replace trigger only5 before insert or update on order for each
  row execute procedure trg_only_5();
  ```
  ```

Only five orders per client

- client(id int, …)
  order(id, client_id,..)

- ```
  create or replace function trg_only_5() returns trigger as
  $code$
  begin
    if tg_op='DELETE' or tg_op='UPDATE' and old.client_id=new.client_id then
      return;
    end if;
    perform pg_xact_advisory_lock(hashint8(new.client_id));
    if (select count(*) from order o where o.client_id=new.client_id)>5 then
       raise sqlstate '23U01' using message='Too many orders';
    end if;
    return new;
  end
  $code$
  language plpgsql;

  create or replace trigger only5 before insert or update on order for each
  row execute procedure trg_only_5();
  ```
  ```

Advisory locks

- Don't want to create temporary tables
- However, they still need to block
- pg_advisory_xact_lock(bigint)
- hashint8, hashtextextended… - undocumented functions for hashing values.

Triggers and referential integrity

- Referential integrity only works with sets of columns of the table.
- some_func(column) – alas, doesn't work
- Trigger only
- When executing, keep in mind for key share - it will prevent the deletion of a string in the parent table

- chart(prefix text not null, …)
- account(nm, ….)
- create or replace function trg_check_nm_prefix() returns trigger as
  $code$
  <<code>>
  declare
   prefix constant text=substring(new.nm from 1 for 5);
  begin
    if tg_op='DELETE' or tg_op='UPDATE' and new.nm=old.nm then
      return;
    end if;
    perform * from chart ca
             where ca.prefix=code.prefix **for key share;**
    if not found then
      raise sqlstate '23U02'
            using message=format('Cannot find chart for %', code.prefix);
    end if;
    return new;
  end;
  $code$
  language plpgsql

Summary

- Nothing non-trivial

- **The database should not allow obviously invalid operations.**

- **The database must detect invalid states.**

Postgres Professional

http://postgrespro.ru/

+7 495 150 06 91

info@postgrespro.ru

postgrespro.ru