BANCO DE DADOS II

Trigger Procedures

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Msc. Eliezio Soares
eliezio.soares@ifrn.edu.br | https://docente.ifrn.edu.br/elieziosoares

BANCOS DE DADOS ATIVOS

Um banco de dados é ativo quando eventos gerados interna ou externamente ao sistema provocam uma resposta do próprio banco de dados (BD), independente da solicitação do usuário. Neste caso, alguma ação é tomada automaticamente dependendo das condições que foram especificadas sobre o estado do banco de dados. Este paradigma é útil para implementar várias funções do próprio banco de dados ou mesmo para estendê-las.

Alguns exemplos de aplicações são: controle de integridade, controle de acesso, políticas de segurança e atualização.

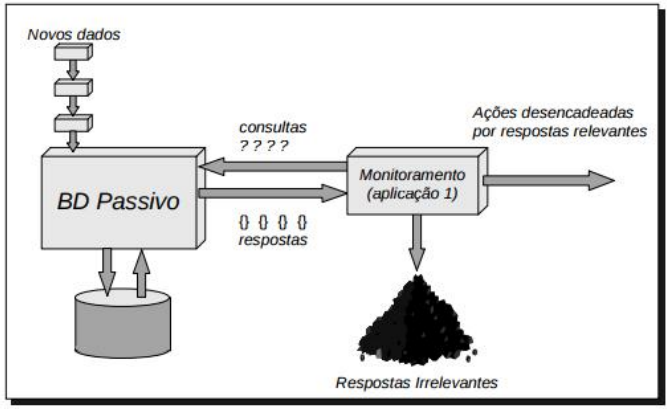

Figura 1. Polling [Buch94].

TRIGGER PROCEDURES

É possível definir para ativar comportamentos específicos quando há mudança nos dados ou eventos no banco de dados.

Uma TRIGGER é criada no postgreSQL através do comando:

- CREATE FUNCTION
- Sem argumentos
- Tipo de retorno: trigger ou event_trigger

A função deve retornar:

- NULL
- Uma linha tendo exatamente a estrutura da tabela

TRIGGER POR MUDANÇA DE DADOS

```
CREATE OR REPLACE FUNCTION log_ator()
RETURNS trigger AS $$
BEGIN
        UPDATE ator
                SET ultima_atualizacao = now()
        WHERE ator_id = NEW.ator_id;

        RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER log_ator AFTER INSERT ON ator
FOR EACH ROW EXECUTE PROCEDURE log_ator();
```

TRIGGER POR MUDANÇA DE DADOS

Deve ser criada:

- Sem argumentos
  - Mesmo se ela espera receber algum argumento.
  - Os argumentos serão passados via o array TG_ARGV
- Tipo de retorno: trigger

No exemplo anterior ao se inserir uma linha na tabela monitorada haverá também uma atualização no campo "ultima_atualizacao".

TRIGGER FUNCTIONS

Quando uma função é executada como uma trigger, algumas variáveis específicas são criadas automaticamente:

- NEW
  - Contém a nova linha do banco de dados para as operações insert e update.
  - Não se aplica a operação delete.
- OLD
  - Contém a antiga linha do banco de dados para as operações delete e update.
  - Não se aplica a operação insert.
- TG_NAME
  - Contém uma string com o nome da trigger disparada.
- TG_WHEN
  - Contém uma string BEFORE, AFTER, or INSTEAD OF, dependendo da definição da trigger.

TRIGGER FUNCTIONS

- TG_LEVEL
  - Contém uma string contendo ROW ou STATEMENT, dependendo da definição da trigger.
- TG_OP
  - Contém uma string INSERT, UPDATE, DELETE, ou TRUNCATE indicando qual operação disparou a trigger.
- TG_RELID
  - Contém o oid (object id) da tabela que gerou o disparo.
- TG_TABLE_NAME
  - Contém o nome da tabela que gerou o evento.
- TG_TABLE_SCHEMA
  - Contém o nome do schema da tabela que gerou o evento.

TRIGGER FUNCTIONS

- TG_NARGS
  - O número de argumentos definidos na declaração CREATE TRIGGER.
- TG_ARGV[]
  - Lista de argumentos passados no evento.

BEFORE, INSTEAD OF, AFTER

O instante de execução da função pode se dar antes, depois do evento motivador do gatilho.

- BEFORE
  - A função será executada antes da execução do INSERT/UPDATE/DELETE.
  - A função pode retornar NULL para sinalizar que a operação deve ser pulada.
  - Retornando um valor não NULL a operação será executada.
  - Para continuar normalmente deve ser retornado o NEW.
  - Em gatilhos Delete o valor de NEW será NULL.

- AFTER
  - A função será executada depois da execução do INSERT/UPDATE/DELETE.

- INSTEAD OF
  - Permite utilizar funções para manipulação de dados em Views.

DESENVOLVENDO TESTES

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);

CREATE TABLE emp_audit(
    operation         char(1)   NOT NULL,
    userid            text      NOT NULL,
    empname           text      NOT NULL,
    salary            integer,
    stamp             timestamp NOT NULL
);
```

TRIGGER BEFORE

```
REATE OR REPLACE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
    BEGIN
        -- Check that empname and salary are given
        IF NEW.empname IS NULL THEN
            RAISE EXCEPTION 'empname cannot be null';
        END IF;
        IF NEW.salary IS NULL THEN
            RAISE EXCEPTION '% cannot have null salary', NEW.empname;
        END IF;

        -- Who works for us when they must pay for it?
        IF NEW.salary <= 0 THEN
            RAISE EXCEPTION '% cannot have a negative or null salary', NEW.empname;
        END IF;

        -- Remember who changed the payroll when
        NEW.last_date := current_timestamp;
        NEW.last_user := current_user;
        RETURN NEW;
    END;
$emp_stamp$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

TRIGGER AFTER

```
CREATE OR REPLACE FUNCTION process_emp_audit() RETURNS TRIGGER AS $emp_audit$
    BEGIN
        --
        -- Create a row in emp_audit to reflect the operation performed on emp,
        -- make use of the special variable TG_OP to work out the operation.
        --
        IF (TG_OP = 'DELETE') THEN
            INSERT INTO emp_audit SELECT 'D', user, OLD.EMPNAME, OLD.SALARY, now();
            RETURN OLD;
        ELSIF (TG_OP = 'UPDATE') THEN
            INSERT INTO emp_audit SELECT 'U', user, NEW.EMPNAME, NEW.SALARY,  now();
            RETURN NEW;
        ELSIF (TG_OP = 'INSERT') THEN
            INSERT INTO emp_audit SELECT 'I', user, NEW.EMPNAME, NEW.SALARY,  now();
            RETURN NEW;
        END IF;
        RETURN NULL; -- result is ignored since this is an AFTER trigger
    END;
$emp_audit$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER emp_audit
AFTER INSERT OR UPDATE OR DELETE
ON emp FOR EACH ROW EXECUTE PROCEDURE process_emp_audit();
```

DÚVIDAS?

REFERÊNCIAS BIBLIOGRÁFICAS

PostgreSQL 9.6.5 Documentation. Disponível em: <https://www.postgresql.org/docs/9.6/static/plpgsql-trigger.html>. Acesso em 28 Set. 2017.

PostgresSQL triggers: Trabalhando com gatilhos em banco de dados. Disponível em: <http://www.devmedia.com.br/trabalhando-com-triggers-no-postgresql/33531> Acesso em 05 Out. 2017.