Database Developmen t and Design

Qinshift Academy

Database Development and Design

Developing and Design of databases using PostgreSQL - Powerful, open-source object-relational database

Agenda

- Session 6
  - Quiz
  - Homework discussion
  - Triggers
    - Create trigger
    - Workshop
  - Indexes
    - Create index
    - Workshop
  - Stored procedures
    - Transactions
    - Create procedure
    - Workshop
  - Knowledge check (Discussion, Homework)

Triggers

Introduction to Triggers 1/2

- A PostgreSQL trigger is a function invoked automatically whenever an event such as insert, update, or delete occurs.
- A trigger is a special user-defined function associated with a table. To create a new trigger, you define a trigger function first, and then bind this trigger function to a table. The difference between a trigger and a user-defined function is that a trigger is automatically invoked when a triggering event occurs.
- PostgreSQL provides two main types of triggers: row and statement-level triggers. The differences between the two kinds are how many times the trigger is invoked and at what time.
- For example, if you issue an UPDATE statement that affects 20 rows, the row-level trigger will be invoked 20 times, while the statement level trigger will be invoked 1 time.

Introduction to Triggers 2/2

- You can specify whether the trigger is invoked before or after an event. If the trigger is invoked before an event, it can skip the operation for the current row or even change the row being updated or inserted. In case the trigger is invoked after the event, all changes are available to the trigger.
- Triggers are useful in case the database is accessed by various applications, and you want to keep the cross-functionality within the database that runs automatically whenever the data of the table is modified. For example, if you want to keep the history of data without requiring the application to have logic to check for every event such as INSERT or UDPATE.
- You can also use triggers to maintain complex data integrity rules which you cannot implement elsewhere except at the database level. For example, when a new row is added into the customer table, other rows must be also created in tables of banks and credits.
- The main drawback of using a trigger is that you must know the trigger exists and understand its logic to figure it out the effects when data changes.

CREATE TRIGGER 1/3

- To create a new trigger in PostgreSQL, you follow these steps:
- First, create a trigger function using CREATE FUNCTION statement.
- Second, bind the trigger function to a table by using CREATE TRIGGER statement.
- A trigger function is similar to a regular user-defined function. However, a trigger function does not take any arguments and has a return value with the type trigger.

```
CREATE FUNCTION trigger_function()
   RETURNS TRIGGER
   LANGUAGE PLPGSQL
AS $$
BEGIN
   -- Trigger logic
END;
$$
```

CREATE TRIGGER 2/3

- A trigger function receives data about its calling environment through a special structure called TriggerData which contains a set of local variables.
- For example, OLD and NEW represent the states of the row in the table before or after the triggering event.
- PostgreSQL also provides other local variables preceded by TG_ such as TG_WHEN, and TG_TABLE_NAME.
- Once you define a trigger function, you can bind it to one or more trigger events such as INSERT, UPDATE, and DELETE.

CREATE TRIGGER 3/3

- The CREATE TRIGGER statement creates a new trigger. The following illustrates the basic syntax of the CREATE TRIGGER statement:

```
CREATE TRIGGER trigger_name -- specify trigger name
   {BEFORE | AFTER} { event } -- fire before or after event and specify the event
                              -- (INSERT, DELETE, UPDATE, TRUNCATE)
   ON table_name -- specify associated table
   [FOR [EACH] { ROW | STATEMENT }] -- type of trigger (for each row/statement)
       EXECUTE PROCEDURE trigger_function -- name of function
```

CREATE TRIGGER 3/3

- The CREATE TRIGGER statement creates a new trigger. The following illustrates the basic syntax of the CREATE TRIGGER statement:

```
CREATE TRIGGER trigger_name -- specify trigger name
   {BEFORE | AFTER} { event } -- fire before or after event and specify the event
                              -- (INSERT, DELETE, UPDATE, TRUNCATE)
   ON table_name -- specify associated table
   [FOR [EACH] { ROW | STATEMENT }] -- type of trigger (for each row/statement)
       EXECUTE PROCEDURE trigger_function -- name of function
```

Indexes

Introduction to indexes 1/2

- PostgreSQL indexes are effective tools to enhance database performance. Indexes help the database server find specific rows much faster than it could do without indexes.
- However, indexes add write and storage overheads to the database system. Therefore, using them appropriately is very important.

Introduction to indexes 2/2

- Assuming that you need to look up for John Doe’s phone number on a phone book. With the understanding that names on the phone book are in alphabetically order, you first look for the page where the last name is Doe, then look for first name John, and finally get his phone number.
- Suppose the names on the phone book were not ordered alphabetically, you would have to go through all pages, check every name until you find John Doe’s phone number. This is called sequential scan which you go over all entries until you find the one that you are looking for.
- Similar to a phonebook, the data stored in the table should be organized in a particular order to speed up various searches. This is why indexes come into play.
- An index is a separated data structure that speeds up the data retrieval on a table at the cost of additional writes and storage to maintain it.

Create index

- Specify the index name after the CREATE INDEX clause. The index name should be meaningful and easy to remember.
- Specify the name of the table to which the index belongs.
- Optionally, specify the index method such as btree, hash, gist, spgist, gin, and brin. PostgreSQL uses btree by default.
- Lastly, list one or more columns that need to be stored in the index. The ASC and DESC specify the sort order. ASC is the default. NULLS FIRST or NULLS LAST specifies nulls sort before or after non-nulls. The NULLS FIRST is the default when DESC is specified and NULLS LAST is the default when DESC is not specified.

```
CREATE INDEX index_name ON table_name [USING method]
(
   column_name [ASC | DESC] [NULLS {FIRST | LAST }],
   ...
);
```

Stored procedures

Transactions

- A database transaction is a single unit of work that consists of one or more operations.
- A classical example of a transaction is a bank transfer from one account to another. A complete transaction must ensure a balance between the sender and receiver accounts. It means that if the sender account transfers X amount, the receiver receives X amount, no more or no less.
- A PostgreSQL transaction is atomic, consistent, isolated, and durable. These properties are often referred to as ACID:
- Atomicity guarantees that the transaction completes in an all-or-nothing manner.
- Consistency ensures the change to data written to the database must be valid and follow predefined rules.
- Isolation determines how transaction integrity is visible to other transactions.
- Durability makes sure that transactions that have been committed will be stored in the database permanently.

Transaction statements

- BEGIN TRANSACTION; or BEGIN WORK; or BEGIN;
- To start the transaction, you could use the above statements. Once a transaction has started, it needs to be ended by using the commit or rollback statements.
- COMMIT WORK; or COMMIT TRANSACTION; or COMMIT;
- To make the change visible to other sessions i.e., commit it (save it), you use the commit statement. After executing the COMMIT statement, PostgreSQL also guarantees that the change will be durable if a crash happens.
- ROLLBACK WORK; or ROLLBACK TRANSACTION; or ROLLBACK;
- To roll back or undo the change of the current transaction, you use any of the above statements.

Create procedure

- A drawback of user-defined functions is that they cannot execute transactions. In other words, inside a user-defined function, you cannot start a transaction, and commit or rollback it.
- PostgreSQL 11 introduced stored procedures that support transactions.
- To define a new stored procedure, you use the create procedure statement.

```
create [or replace] procedure procedure_name(parameter_list)
language plpgsql
as $$
declare
-- variable declaration
begin
-- stored procedure body
end; $$
```

Create procedure

- Parameters in stored procedures can have the in and inout modes. They cannot have the out mode.
- A stored procedure does not return a value. You cannot use the return statement with a value inside a store procedure like this:

```
return expression;
```

- However, you can use the return statement without the expression to stop the stored procedure immediately:

```
return;
```

Homework 5

Homework

- Exercise 1: Write a transaction to insert a new movie and its revenue
  - Insert into movies table
  - Insert into movie_revenues table
  - If any step fails, roll back both operations
- Write a transaction to update movie budget and revenue
  - Update movie budget
  - Update revenue
  - Ensure both updates succeed or none
- Create a trigger to update 'created_at' timestamp whenever a new movie is inserted
- Create a trigger to prevent inserting movies with release dates in the future
- Create a function that returns movie details as a row type
- Create a procedure to add a new movie with its genre

Questions?

Trainer Name

Trainer mail

Assistant Name

Assistant mail

Qinshift Academy