PostgreSQL Materialized Views

Summary: in this tutorial, you will learn about PostgreSQL materialized views that store the result of a query physically and refresh the data from base tables periodically.

Introduction to the PostgreSQL materialized views

In PostgreSQL, views are virtual tables that represent data of the underlying tables. Simple views can be updatable.

PostgreSQL extends the view concept to the next level which allows views to store data physically. These views are called **materialized views**.

Materialized views cache the result set of an expensive query and allow you to refresh data periodically.

The materialized views can be useful in many cases that require fast data access. Therefore, you often find them in data warehouses and business intelligence applications.

Creating materialized views

To create a materialized view, you use the `CREATE MATERIALIZED VIEW` statement as follows:

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] view_name
AS
query
WITH [NO] DATA;
```

How it works.

First, specify the `view_name` after the `CREATE MATERIALIZED VIEW` clause

Second, add the query that retrieves data from the underlying tables after the `AS` keyword.

Third, if you want to load data into the materialized view at the creation time, use the `WITH DATA` option; otherwise, you use `WITH NO DATA` option. If you use the `WITH NO DATA` option, the view is flagged as unreadable. It means that you cannot query data from the view until you load data into it.

Finally, use the IF NOT EXISTS option to conditionally create a view only if it does not exist.

Refreshing data for materialized views

To load data into a materialized view, you use the `REFRESH MATERIALIZED VIEW` statement:

```
REFRESH MATERIALIZED VIEW view_name;
```

When you refresh data for a materialized view, PostgreSQL locks the underlying tables. Consequently, you will not be able to retrieve data from underlying tables while data is loading into the view.

To avoid this, you can use the `CONCURRENTLY` option.

```
REFRESH MATERIALIZED VIEW CONCURRENTLY view_name;
```

With the `CONCURRENTLY` option, PostgreSQL creates a temporary updated version of the materialized view, compares two versions, and performs INSERT and UPDATE only the differences.

PostgreSQL allows you to retrieve data from a materialized view while it is being updated. One requirement for using `CONCURRENTLY` option is that the materialized view must have a `UNIQUE` index.

Notice that `CONCURRENTLY` option is only available in PostgreSQL 9.4 or later.

Removing materialized views

To remove a materialized view, you use the `DROP MATERIALIZED VIEW` statement:

```
DROP MATERIALIZED VIEW view_name;
```

In this syntax, you specify the name of the materialized view that you want to drop after the `DROP MATERIALIZED VIEW` keywords.

PostgreSQL materialized views example

We'll use the tables in the sample database for creating a materialized view.

First, create a materialized view named `rental_by_category` using the `CREATE MATERIALIZED VIEW` statement:

```
CREATE MATERIALIZED VIEW rental_by_category
AS
 SELECT c.name AS category,
    sum(p.amount) AS total_sales
   FROM (((((payment p
     JOIN rental r ON ((p.rental_id = r.rental_id)))
     JOIN inventory i ON ((r.inventory_id = i.inventory_id)))
     JOIN film f ON ((i.film_id = f.film_id)))
     JOIN film_category fc ON ((f.film_id = fc.film_id)))
     JOIN category c ON ((fc.category_id = c.category_id)))
  GROUP BY c.name
  ORDER BY sum(p.amount) DESC
WITH NO DATA;
```

Because of the `WITH NO DATA` option, you cannot query data from the view. If you attempt to do so, you'll get the following error message:

```
SELECT * FROM rental_by_category;
```

Output:

```
[Err] ERROR: materialized view "rental_by_category" has not been populated
HINT: Use the REFRESH MATERIALIZED VIEW command.
```

PostgreSQL is helpful to give you a hint to ask for loading data into the view.

Second, load data into the materialized view using the `REFRESH MATERIALIZED VIEW` statement:

```
REFRESH MATERIALIZED VIEW rental_by_category;
```

Third, retrieve data from the materialized view:

```
SELECT * FROM rental_by_category;
```

Output:

```
category    | total_sales
-------------+-------------
 Sports      |     4892.19
 Sci-Fi      |     4336.01
 Animation   |     4245.31
 Drama       |     4118.46
 Comedy      |     4002.48
 New         |     3966.38
 Action      |     3951.84
 Foreign     |     3934.47
 Games       |     3922.18
 Family      |     3830.15
 Documentary |     3749.65
 Horror      |     3401.27
 Classics    |     3353.38
 Children    |     3309.39
 Travel      |     3227.36
 Music       |     3071.52
(16 rows)
```

From now on, you can refresh the data in the `rental_by_category` view using the `REFRESH MATERIALIZED VIEW` statement.

However, to refresh it with `CONCURRENTLY` option, you need to create a `UNIQUE` index for the view first.

```
CREATE UNIQUE INDEX rental_category
ON rental_by_category (category);
```

Let's refresh data concurrently for the `rental_by_category` view.

```
REFRESH MATERIALIZED VIEW CONCURRENTLY rental_by_category;
```

Summary

- A materialized view is a view that stores data that comes from the base tables.
- Use the `CREATE MATERIALIZED VIEW` statement to create a materialized view.
- Use the `REFRESH MATERIALIZED VIEW` statement to load data from the base tables into the view.
- Use the `DROP MATERIALIZED VIEW` statement to drop a materialized view.

← Previous
PostgreSQL ALTER VIEW Statement

Next →
PostgreSQL Recursive View

Last updated on March 16, 2024

Was this page helpful? Yes No