NAME

CREATE_SUBSCRIPTION − define a new subscription

SYNOPSIS

CREATE SUBSCRIPTION *subscription_name*
CONNECTION '*conninfo*'
PUBLICATION *publication_name* [, ...]
[WITH (*subscription_parameter* [= *value*] [, ...])]

DESCRIPTION

CREATE SUBSCRIPTION adds a new logical−replication subscription. The user that creates a subscription becomes the owner of the subscription. The subscription name must be distinct from the name of any existing subscription in the current database.

A subscription represents a replication connection to the publisher. Hence, in addition to adding definitions in the local catalogs, this command normally creates a replication slot on the publisher.

A logical replication worker will be started to replicate data for the new subscription at the commit of the transaction where this command is run, unless the subscription is initially disabled.

To be able to create a subscription, you must have the privileges of the pg_create_subscription role, as well as CREATE privileges on the current database.

Additional information about subscriptions and logical replication as a whole is available at Section 29.2 and Chapter 29.

PARAMETERS

subscription_name
The name of the new subscription.

CONNECTION '*conninfo*'
The libpq connection string defining how to connect to the publisher database. For details see Section 32.1.1.

PUBLICATION *publication_name* [, ...]
Names of the publications on the publisher to subscribe to.

WITH (*subscription_parameter* [= *value*] [, ...])
This clause specifies optional parameters for a subscription.

The following parameters control what happens during subscription creation:

connect (boolean)
Specifies whether the **CREATE SUBSCRIPTION** command should connect to the publisher at all. The default is true. Setting this to false will force the values of create_slot, enabled and copy_data to false. (You cannot combine setting connect to false with setting create_slot, enabled, or copy_data to true.)

Since no connection is made when this option is false, no tables are subscribed. To initiate replication, you must manually create the replication slot, enable the failover if required, enable the subscription, and refresh the subscription. See Section 29.2.3 for examples.

create_slot (boolean)
Specifies whether the command should create the replication slot on the publisher. The default is true.

If set to false, you are responsible for creating the publisher's slot in some other way. See Section 29.2.3 for examples.

enabled (boolean)
Specifies whether the subscription should be actively replicating or whether it should just be set up but not started yet. The default is true.

slot_name (string)
: Name of the publisher's replication slot to use. The default is to use the name of the subscription for the slot name.

  Setting slot_name to NONE means there will be no replication slot associated with the subscription. Such subscriptions must also have both enabled and create_slot set to false. Use this when you will be creating the replication slot later manually. See Section 29.2.3 for examples.

  When setting slot_name to a valid name and create_slot to false, the failover property value of the named slot may differ from the counterpart failover parameter specified in the subscription. Always ensure the slot property failover matches the counterpart parameter of the subscription and vice versa. Otherwise, the slot on the publisher may behave differently from what these subscription options say: for example, the slot on the publisher could either be synced to the standbys even when the subscription's failover option is disabled or could be disabled for sync even when the subscription's failover option is enabled.

The following parameters control the subscription's replication behavior after it has been created:

binary (boolean)
: Specifies whether the subscription will request the publisher to send the data in binary format (as opposed to text). The default is false. Any initial table synchronization copy (see copy_data) also uses the same format. Binary format can be faster than the text format, but it is less portable across machine architectures and PostgreSQL versions. Binary format is very data type specific; for example, it will not allow copying from a smallint column to an integer column, even though that would work fine in text format. Even when this option is enabled, only data types having binary send and receive functions will be transferred in binary. Note that the initial synchronization requires all data types to have binary send and receive functions, otherwise the synchronization will fail (see CREATE TYPE (**CREATE_TYPE**(7)) for more about send/receive functions).

  When doing cross−version replication, it could be that the publisher has a binary send function for some data type, but the subscriber lacks a binary receive function for that type. In such a case, data transfer will fail, and the binary option cannot be used.

  If the publisher is a PostgreSQL version before 16, then any initial table synchronization will use text format even if binary = true.

copy_data (boolean)
: Specifies whether to copy pre−existing data in the publications that are being subscribed to when the replication starts. The default is true.

  If the publications contain WHERE clauses, it will affect what data is copied. Refer to the Notes for details.

  See Notes for details of how copy_data = true can interact with the origin parameter.

streaming (enum)
: Specifies whether to enable streaming of in−progress transactions for this subscription. The default value is parallel, meaning incoming changes are directly applied via one of the parallel apply workers, if available. If no parallel apply worker is free to handle streaming transactions then the changes are written to temporary files and applied after the transaction is committed. Note that if an error happens in a parallel apply worker, the finish LSN of the remote transaction might not be reported in the server log.

  Caution

  There is a risk of deadlock when the schemas of the publisher and subscriber differ, although such cases are rare. The apply worker is equipped to retry these transactions automatically.

If set to on, the incoming changes are written to temporary files and then applied only after the transaction is committed on the publisher and received by the subscriber.

If set to off, all transactions are fully decoded on the publisher and only then sent to the subscriber as a whole.

synchronous_commit (enum)
The value of this parameter overrides the synchronous_commit setting within this subscription's apply worker processes. The default value is off.

It is safe to use off for logical replication: If the subscriber loses transactions because of missing synchronization, the data will be sent again from the publisher.

A different setting might be appropriate when doing synchronous logical replication. The logical replication workers report the positions of writes and flushes to the publisher, and when using synchronous replication, the publisher will wait for the actual flush. This means that setting synchronous_commit for the subscriber to off when the subscription is used for synchronous replication might increase the latency for **COMMIT** on the publisher. In this scenario, it can be advantageous to set synchronous_commit to local or higher.

two_phase (boolean)
Specifies whether two−phase commit is enabled for this subscription. The default is false.

When two−phase commit is enabled, prepared transactions are sent to the subscriber at the time of **PREPARE TRANSACTION**, and are processed as two−phase transactions on the subscriber too. Otherwise, prepared transactions are sent to the subscriber only when committed, and are then processed immediately by the subscriber.

The implementation of two−phase commit requires that replication has successfully finished the initial table synchronization phase. So even when two_phase is enabled for a subscription, the internal two−phase state remains temporarily “pending” until the initialization phase completes. See column subtwophasestate of pg_subscription to know the actual two−phase state.

disable_on_error (boolean)
Specifies whether the subscription should be automatically disabled if any errors are detected by subscription workers during data replication from the publisher. The default is false.

password_required (boolean)
If set to true, connections to the publisher made as a result of this subscription must use password authentication and the password must be specified as a part of the connection string. This setting is ignored when the subscription is owned by a superuser. The default is true. Only superusers can set this value to false.

run_as_owner (boolean)
If true, all replication actions are performed as the subscription owner. If false, replication workers will perform actions on each table as the owner of that table. The latter configuration is generally much more secure; for details, see Section 29.11. The default is false.

origin (string)
Specifies whether the subscription will request the publisher to only send changes that don't have an origin or send changes regardless of origin. Setting origin to none means that the subscription will request the publisher to only send changes that don't have an origin. Setting origin to any means that the publisher sends changes regardless of their origin. The default is any.

See Notes for details of how copy_data = true can interact with the origin parameter.

failover (boolean)
Specifies whether the replication slots associated with the subscription are enabled to be synced to the standbys so that logical replication can be resumed from the new primary after failover.

The default is false.

When specifying a parameter of type boolean, the = *value* part can be omitted, which is equivalent to specifying TRUE.

NOTES

See Section 29.11 for details on how to configure access control between the subscription and the publication instance.

When creating a replication slot (the default behavior), **CREATE SUBSCRIPTION** cannot be executed inside a transaction block.

Creating a subscription that connects to the same database cluster (for example, to replicate between databases in the same cluster or to replicate within the same database) will only succeed if the replication slot is not created as part of the same command. Otherwise, the **CREATE SUBSCRIPTION** call will hang. To make this work, create the replication slot separately (using the function **pg_create_logical_replication_slot** with the plugin name pgoutput) and create the subscription using the parameter create_slot = false. See Section 29.2.3 for examples. This is an implementation restriction that might be lifted in a future release.

If any table in the publication has a WHERE clause, rows for which the *expression* evaluates to false or NULL will not be published. If the subscription has several publications in which the same table has been published with different WHERE clauses, a row will be published if any of the expressions (referring to that publish operation) are satisfied. In the case of different WHERE clauses, if one of the publications has no WHERE clause (referring to that publish operation) or the publication is declared as FOR ALL TABLES or FOR TABLES IN SCHEMA, rows are always published regardless of the definition of the other expressions. If the subscriber is a PostgreSQL version before 15, then any row filtering is ignored during the initial data synchronization phase. For this case, the user might want to consider deleting any initially copied data that would be incompatible with subsequent filtering. Because initial data synchronization does not take into account the publication publish parameter when copying existing table data, some rows may be copied that would not be replicated using DML. See Section 29.2.2 for examples.

Subscriptions having several publications in which the same table has been published with different column lists are not supported.

We allow non−existent publications to be specified so that users can add those later. This means pg_subscription can have non−existent publications.

When using a subscription parameter combination of copy_data = true and origin = NONE, the initial sync table data is copied directly from the publisher, meaning that knowledge of the true origin of that data is not possible. If the publisher also has subscriptions then the copied table data might have originated from further upstream. This scenario is detected and a WARNING is logged to the user, but the warning is only an indication of a potential problem; it is the user's responsibility to make the necessary checks to ensure the copied data origins are really as wanted or not.

To find which tables might potentially include non−local origins (due to other subscriptions created on the publisher) try this SQL query:

```
# substitute <pub−names> below with your publication name(s) to be queried
SELECT DISTINCT PT.schemaname, PT.tablename
FROM pg_publication_tables PT
    JOIN pg_class C ON (C.relname = PT.tablename)
    JOIN pg_namespace N ON (N.nspname = PT.schemaname),
    pg_subscription_rel PS
WHERE C.relnamespace = N.oid AND
     (PS.srrelid = C.oid OR
     C.oid IN (SELECT relid FROM pg_partition_ancestors(PS.srrelid) UNION
            SELECT relid FROM pg_partition_tree(PS.srrelid))) AND
     PT.pubname IN (<pub−names>);
```

EXAMPLES

Create a subscription to a remote server that replicates tables in the publications mypublication and insert_only and starts replicating immediately on commit:

```
CREATE SUBSCRIPTION mysub
    CONNECTION 'host=192.168.1.50 port=5432 user=foo dbname=foodb'
    PUBLICATION mypublication, insert_only;
```

Create a subscription to a remote server that replicates tables in the insert_only publication and does not start replicating until enabled at a later time.

```
CREATE SUBSCRIPTION mysub
    CONNECTION 'host=192.168.1.50 port=5432 user=foo dbname=foodb'
    PUBLICATION insert_only
        WITH (enabled = false);
```

COMPATIBILITY

CREATE SUBSCRIPTION is a PostgreSQL extension.

SEE ALSO

ALTER SUBSCRIPTION (**ALTER_SUBSCRIPTION**(7)), DROP SUBSCRIPTION (**DROP_SUBSCRIPTION**(7)), CREATE PUBLICATION (**CREATE_PUBLICATION**(7)), ALTER PUBLICATION (**ALTER_PUBLICATION**(7))