2020 PostgreSQL中文社区钉钉线上直播活动

今日直播主题

PostgreSQL逻辑复制探究

分享嘉宾：何敏

扫描钉钉直播二维码，免费看直播

逻辑复制概述

- ◆ 从2014年发布的9.4版本开始，PostgreSQL就支持逻辑复制了，只是一直没有将其引入内核。
- ◆ 2017年发布的10.0，在内核层面支持基于REDO流的逻辑复制。
- ✓ 逻辑复制是PostgreSQL V10重量级新特性，支持内置的逻辑复制。
- ✓ 可基于表级别复制，是一种粒度可细的复制。
- ✓ 你可以针对同一个数据库实例，同时使用逻辑复制和物理复制，因为他们都是基于REDO的。

逻辑复制与物理复制的比较

物理复制的优越性及局限性：

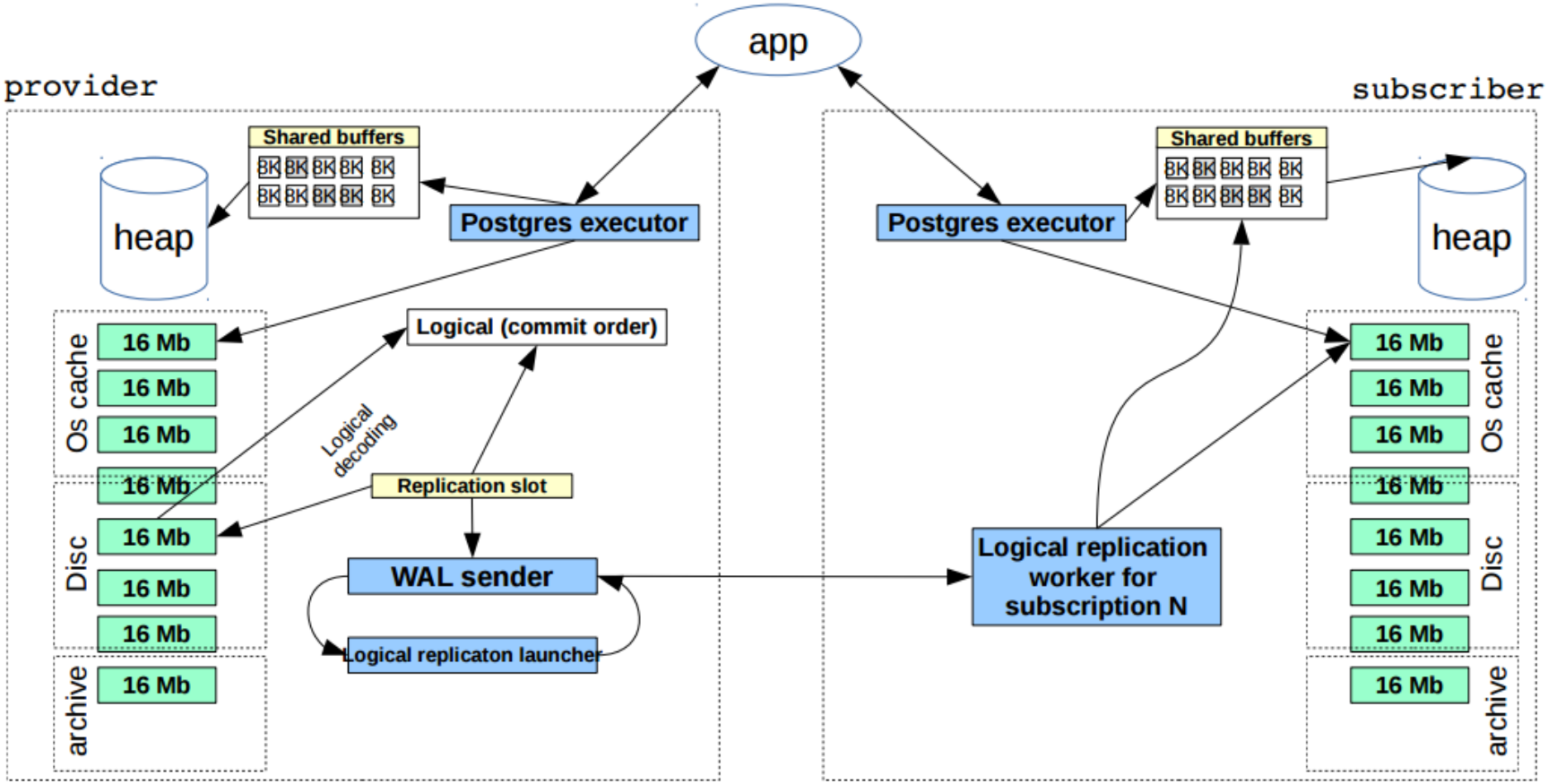
1. 物理层面、逻辑层面完全一致：主备数据库的物理块完全一样，逻辑层面也完全一样，达到金融级需求。
2. 延迟极低：任何大事务，主库完成提交，备库能立即提交。事务过程中产生的每条record会实时同步到备库进行回放，因此，备库不会因为回放大事务而使得延迟很大。
3. 整个实例级别同步：物理复制针对整个实例进行同步，无法仅针对单个表或者数据库进行同步。
4. 同步级别无法单独设置：数据库同步级别针对整个流复制，无法针对某个表进行设置流复制级别。往往一个数据库中，仅有几张最重要表需要最高级别的同步等级，其他表异步即可。

逻辑复制的特点及限制

逻辑复制的特点：

1.根据发布、订阅的方式实现逻辑复制，达到数据库同步。
2.也是基于wal日志，进行流复制的，传输可以选择二进制也可以选择字符串。
3.同步的最小单元是表，因此可以很灵活的进行数据库同步。
4.可以对不同的订阅单元设置不同的同步级别，需要完全一致的表配置为同步，没那么强的设置为异步。
5.支持多对一、一对多、多对多进行逻辑复制，以此能够很灵活的实现各种同步需求。
6.支持在不同大版本数据库之间进行逻辑复制。
7.支持在不同架构服务器之间数据同步：windows、Linux之间实现数据同步。
8.订阅端，这里不能叫备库了，是可以写的。而且可以再次对数据表进行发布。

逻辑复制的架构（引）


app
provider
subscriber
Shared buffers
heap
Postgres executor
Logical (commit order)
Logical decoding
Replication slot
WAL sender
Logical replicaton launcher
Os cache
Disc
archive
16 Mb
Logical replication worker for subscription N

逻辑复制的关键步骤

Logical Decoding
PostgreSQL 的逻辑日志来源于解析物理 WAL 日志。解析 WAL 成为逻辑数据的过程叫 Logical Decoding。

Replication Slots
保存逻辑或物理流复制的基础信息。一个逻辑 slot 创建后，它的相关信息可以通过 pg_replication_slots 系统视图获取。
如果它在 active 状态，则可以通过系统视图 pg_stat_replication 看到一些 slot 的实时的状态信息。

Output Plugins
PostgreSQL 的逻辑流复制协议开放一组可编程接口，用于自定义输数据到客户端的逻辑数据的格式。这部分实现使用插件的方式被内核集成和使用，称作 Output Plugins，例如系统自带的test_decoding。

Exported Snapshots
当一个逻辑流复制 slot 被创建时，系统会产生一个快照。客户端可以通过它订阅到数据库任意时间点的数据变化。

Logical Decoding输出的数据格式

Logical Decoding 是把 WAL 日志解析成逻辑日志的过程。这个过程输出的是数据格式可以描述为：

1.事务开始：任何的变化总是在一个事务中，所以订阅的数据变化的开始是一个事务被启动的消息，他包括了事务ID，LSN，开始时间等信息。

2.数据的变化：包括当前事物中修改的数据。即对某些表的 insert update delete 操作来带的数据变化。

1）一个事务内可以包含任意个表和任意行数据的变化。

2）输出的数据的格式和对应表的定义和索引相关。

3.事务的提交：包括事物提交的 LSN，时间等相关信息。

Output Plugins 需要完成的工作

上一节的内容提到的数据需要通过 Output Plugins 确定最终的数据格式，再发送给客户端

这部分的接口表现为下列回调函数：

1.LogicalDecodeStartupCB startup_cb;
2.LogicalDecodeBeginCB begin_cb;
3.LogicalDecodeChangeCB change_cb;
4.LogicalDecodeCommitCB commit_cb;
5.LogicalDecodeShutdownCB shutdown_cb;

函数 1，5 是分配和清理插件所需的内存结构。此外的 2，3，4 三个回调函数完全对应
上一节中的三个部分。

开发者可以根据需求实现这一组函数。

```
(0/165C970,495,"BEGIN 495")
(0/165C970,495,"table public.test: INSERT: id[integer]:1 info[character varying]:'test4' crt_time[timestamp without time zone]:'2020-03-26 1
(0/165CC18,495,"COMMIT 495")
(0/165CC50,496,"BEGIN 496")
(0/165CC50,496,"table public.test: UPDATE: id[integer]:1 info[character varying]:'test update' crt_time[timestamp without time zone]:'2020-0
(0/165CCE0,496,"COMMIT 496")
34 rows)
```

Output Plugins 需要完成的工作

REPLICA IDENTITY 如何影响输出数据的格式

逻辑流复制利用索引的方式优化传输数据的效率，它们可以按表为单位定制。大致分为三种情况:

1. 如果修改的表有 primary key, 则表的变化的逻辑数据只会包括该表变化的列和pk列数据, 如果 pk 列被修改，则还会输出老的 pk 列数据。

2. 如果修改的表没有 primary key，则可以使用 alter table 指定一个 REPLICA index，同时需要这个索引列为非空，其产生的效果和 1 相同。

3. 如果修改的表不满足上面的两个条件，而又要做同步，可以使用 alter table 设置这个表的 REPLICA IDENTITY 为 FULL。于是系统在表修改时会记录修改行的所有列，不会做任何的优化。

很明显，给对应的表设置 PK 或指定索引，在数据同步时效率更高。我们可以安需定制同步策略。在实现功能的过程中需要考虑这部分变化。

Output Plugins 需要完成的工作

xlog.c - postgresql

walwriter.c ● | xlog.c ×

d > access > transam > C xlog.c > XLogWrite(XLogwrtRqst, bool)

```
        nleft -= written;
        from += written;
        startoffset += written;
    } while (nleft > 0);

    npages = 0;

    /*
     * If we just wrote the whole last page of a logfile segmen
     * fsync the segment immediately.  This avoids having to go
     * and re-open prior segments when an fsync request comes a
     * later. Doing it here ensures that one and only one backe
     * perform this fsync.
     *
     * This is also the right place to notify the Archiver that
     * segment is ready to copy to archival storage, and to upd
     * timer for archive_timeout, and to signal for a checkpoin
     * too many logfile segments have been used since the last
     * checkpoint.
     */
    if (finishing_seg)
    {
        issue_xlog_fsync(openLogFile, openLogSegNo);

        /* signal that we need to wakeup walsenders later */
        WalSndWakeupRequest();

        LogwrtResult.Flush = LogwrtResult.Write;    /* end of page */

        if (XLogArchivingActive())
```

```
void
WalSndWakeup(void)
{
    int         i;

    for (i = 0; i < max_wal_senders; i++)
    {
        Latch      *latch;
        WalSnd     *walsnd = &WalSndCtl->walsnds[i];

        /*
         * Get latch pointer with spinlock held, for the unlikely case that
         * pointer reads aren't atomic (as they're 8 bytes).
         */
        SpinLockAcquire(&walsnd->mutex);
        latch = walsnd->latch;
        SpinLockRelease(&walsnd->mutex);

        if (latch != NULL)
            SetLatch(latch);
    }
}
```

逻辑复制-角色

逻辑复制架构图中最重要的两个角色为Publication和Subscription。

Publication（发布）：

- 可以定义在任何可读写的PostgreSQL实例上，对于已创建Publication的数据库称为发布节点。
- 一个数据库中允许创建多个发布，目前允许加入发布的对象只有表，允许将多个表注册到一个发布中。
- 加入发布的表通常需要有复制标识（replica identity），从而使逻辑主库表上的DELETE/UPDAE操作可以标记到相应数据行并复制到逻辑备库上的相应表，默认情况下使用主键作为复制标识，如果没有主键，也可是唯一索引，如果没有主键或唯一索引，可设置复制标识为full，意思是整行数据作为键值，这种情况下复制效率会降低。
- 如果加入发布的表没有指定复制标识，表上的UPDATE/DELETE将会报错。

逻辑复制-角色

逻辑复制架构图中最重要的两个角色为Publication和Subscription。

Subscription（订阅）：

- 实时同步指定发布者的表数据，位于逻辑复制的下游节点。
- 对于已创建Subscription的数据库称为订阅节点，订阅节点的数据库上同时也能创建发布。
- 发布节点上发布的表的DDL不会被复制，因此，如果发布节点上发布的表结构更改了，订阅节点上需手工对订阅的表进行DDL操作。
- 订阅节点最好通过逻辑复制槽获取发布节点发送的WAL数据变化。

逻辑复制-数据库配置

Publication（发布）节点

wal_level = logical
max_replication_slots = 8　　#每一个订阅需要消耗一个slot
max_wal_senders = 10　　#每一个订阅要使用一个wal sender
max_worker_processes=128　　#必须大于等于max_wal_senders加并行计算进程，或者其他插件需要fork的进程数。

pg_hba.conf：

host　　replication　　postgres　　0.0.0.0/0　　md5

Subscription（订阅）节点

postgersql.conf:

max_replication_slots = 8
#保证不低于该实例总共需要创建的订阅数，同步进程+apply进程
max_logical_replication_workers = 8
#大于等于max_logical_replication_workers + 1 + CPU并行计算 + 其他插件需要fork的进程数
max_worker_processes=128

逻辑复制-发布者

发布语法：

```
CREATE PUBLICATION name
    [ FOR TABLE [ ONLY ] table_name [ * ] [, ...]
      | FOR ALL TABLES ]
    [ WITH ( publication_parameter [= value] [, ... ] ) ]
```

```
postgres=# \h alter publication
Command:     ALTER PUBLICATION
Description: change the definition of a publication
Syntax:
ALTER PUBLICATION name ADD TABLE [ ONLY ] table_name [ * ] [, ...]
ALTER PUBLICATION name SET TABLE [ ONLY ] table_name [ * ] [, ...]
ALTER PUBLICATION name DROP TABLE [ ONLY ] table_name [ * ] [, ...]
ALTER PUBLICATION name SET ( publication_parameter [= value] [, ... ] )
ALTER PUBLICATION name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
ALTER PUBLICATION name RENAME TO new_name
```

URL: https://www.postgresql.org/docs/12/sql-alterpublication.html

逻辑复制-发布者参数

Parameters

Name

The name of the new publication.

FOR TABLE

Specifies a list of tables to add to the publication. If ONLY is specified before the table name, only that table is added to the publication. **If ONLY is not specified, the table and all its descendant(继承表) tables (if any) are added**. Optionally, * can be specified after the table name to explicitly indicate that descendant tables are included.

Only persistent base tables can be part of a publication. **Temporary tables, unlogged tables, foreign tables, materialized views, regular views, and partitioned tables cannot be part of a publication**. To replicate a partitioned table, add the individual partitions to the publication.

FOR ALL TABLES

Marks the publication as one that replicates changes for all tables in the database, **including tables created in the future**.

WITH (***publication_parameter*** [= ***value***] [, ...])

publish (string)This parameter determines which DML operations will be published by the new publication to the subscribers. The allowed operations are **insert, update, delete, and truncate**. **The default is to publish all actions, and so the default value for this option is 'insert, update, delete, truncate** '.

发布表的复制标识-REPLICA IDENTITY

1. DEFAULT (the default for non-system tables) records the old values of the columns of the primary key, if any.

2. USING INDEX records the old values of the columns covered by the named index, **which must be unique, not partial, not deferrable, and include only columns marked NOT NULL**.
记录指定索引列（索引的所有列须是not null列，其实和PK一样，但是某些情况下，你可以选一个比PK更小的UK）

Alter table test replica identity using index idx_test_id;

3. FULL records the old values of all columns in the row.
默认为replica identity nothing。如果这种表发布出去了，允许insert，但是执行delete或者update时，会报错。

4. NOTHING records no information about the old row (This is the default for system tables.)
In all cases, no old values are logged unless at least one of the columns that would be logged differs between the old and new versions of the row.
仅仅当数据有变更时才会记录old value，比如delete。 或者update前后old.*<>new.*。

逻辑复制-订阅者

订阅者语法：

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

postgres=# \h alter subscription

ALTER SUBSCRIPTION name CONNECTION 'conninfo'
ALTER SUBSCRIPTION name SET PUBLICATION publication_name [, ...] [WITH
(set_publication_option [= value] [, ...])]
ALTER SUBSCRIPTION name REFRESH PUBLICATION [WITH (refresh_option [= value]
[, ...])]
ALTER SUBSCRIPTION name ENABLE
ALTER SUBSCRIPTION name DISABLE
ALTER SUBSCRIPTION name SET (subscription_parameter [= value] [, ...])
ALTER SUBSCRIPTION name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
ALTER SUBSCRIPTION name RENAME TO new_name

URL: https://www.postgresql.org/docs/12/sql-altersubscription.html

逻辑复制-订阅者者

Parameter：
copy_data (boolean)
Specifies whether the existing data in the publications that are being subscribed to should be copied once the replication starts. The default is true.

create_slot (boolean)
Specifies whether the command should create the replication slot on the publisher. The default is true.
enabled (boolean)Specifies whether the subscription should be actively replicating, or whether it should be just setup but not started yet. The default is true.

slot_name (string)
Name of the replication slot to use. The default behavior is to use the name of the subscription for the slot name.
When slot_name is set to NONE, there will be no replication slot associated with the subscription. This can be used if the replication slot will be created later manually. Such subscriptions must also have
both enabled and create_slot set to false.

逻辑复制-订阅者者

synchronous_commit (enum)
The value of this parameter overrides the **synchronous_commit** setting. **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 resent 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.

connect (boolean)
Specifies whether the CREATE SUBSCRIPTION should connect to the publisher at all. Setting this to false will change default values of enabled, create_slot and copy_data to false.
It is not allowed to combine connect set to false and enabled, create_slot, or copy_data set to true.

Since no connection is made when this option is set to false, the tables are not subscribed, and so after you enable the subscription nothing will be replicated. It is required to run ALTER SUBSCRIPTION ... REFRESH PUBLICATION in order for tables to be subscribed.

逻辑复制-查看发布情况

发布端：

- pg_publication
- pg_replication_slots
- pg_stat_replication

select pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_insert_lsn(), sent_lsn)) send_gap, pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_insert_lsn(), replay_lsn)) apply_gap, * from pg_stat_replication ;

订阅端：

- pg_subscription
- pg_stat_subscription

select pg_size_pretty(pg_wal_lsn_diff(received_lsn, latest_end_lsn)), * from pg_stat_subscription ; pg_replication_origin_status

- **pg_replication_origin_status**

逻辑复制-冲突处理

逻辑复制，本质上是事务层级的复制。

- 如果订阅端apply失败（或者说引发了任何错误，包括约束等），都会导致该订阅暂停。用户可以在订阅端数据库日志中查看错误原因。

注意，update, delete没有匹配的记录时，不会报错，也不会导致订阅暂停。

- 冲突修复方法

1. 通过修改订阅端的数据，解决冲突。例如insert违反了唯一约束时，可以删除订阅端造成唯一约束冲突的记录先DELETE掉。
   --常用方法

2. 在订阅端调用pg_replication_origin_advance(node_name text, pos pg_lsn)函数，node_name就是subscription name，pos指重新开始的LSN，从而跳过有冲突的事务。
   --可以在错误日志中查看，也可以通过视图查看。

逻辑复制-权限控制

- The role used for the **replication connection** must have the **REPLICATION attribute (or be a superuser)**. **Access for the role** must be configured in pg_hba.conf and it must have the **LOGIN attribute**.
- In order to be able to copy the initial table data, the role used for the replication **connection must have the SELECT privilege** on a published table (or be a superuser).
- To create a publication, the user must have the CREATE privilege in the database.
- To add tables to a publication, the user must have ownership rights on the table. To create a publication that publishes all tables automatically, the user must be a superuser.
- To create a subscription, the user must be a superuser.
- The subscription apply process will run in the local database with the privileges of a superuser.
- Privileges are only checked once at the start of a replication connection. They are not re-checked as each change record is read from the publisher, nor are they re-checked for each change when applied.

https://www.postgresql.org/docs/12/logical-replication-security.html

逻辑复制-一些限制

- schema和DDL不会被同步，需要在订阅端创建表，包含要同步的字段。
- Sequence不会被同步：sequence的数据肯定会在表上进行同步，但是订阅节点的sequence不会进行增长。
- Truncate限制：truncate会进行同步，但需要注意的是如果truncate的对象是一个通过外键关联的对象组时，truncate会将整个组的表都重置。如果订阅端的表是其他表的外键（不在发布端的组内），则会同步失败。**而且当单个表有多个发布端的情况，也会truncate表**。
- 大对象不会同步。
- 只支持基本表的同步，不支持视图、物化视图、分区根表、外部表、**无日志表**的同步。分区表需要对子表一一建立单独的同步。
- 不建议双向复制，会导致WAL循环，除非用特殊的函数进行处理（通过序列划分、触发器判断）。
- 不支持在备机（流复制集群的）上创建订阅。
- 订阅表上没有合适的REPLICA IDENTITY时，发布端执行UPDATE/DELETE会报错。

逻辑复制-实践

1) 发布端数据库配置：

postgresql.conf：

wal_level = logical
max_replication_slots = 8
max_wal_senders = 10
max_worker_processes=128

pg_hba.conf：

host replication postgres 0.0.0.0/0 md5

订阅端：

mkdir data1
initdb –D data1
pg_ctl –D data1 start

逻辑复制-实践

2）发布端创建表并发布：

```
postgres=# create table test(id int primary key, info text, crt_time timestamp);
CREATE TABLE
postgres=# \d test
                    Table "public.test"
  Column  |            Type             | Collation | Nullable | Default
----------+-----------------------------+-----------+----------+---------
 id       | integer                     |           | not null |
 info     | text                        |           |          |
 crt_time | timestamp without time zone |           |          |
Indexes:
    "test_pkey" PRIMARY KEY, btree (id)

postgres=# alter table test replica identity using index test_pkey ;
ALTER TABLE
postgres=# \d test
                    Table "public.test"
  Column  |            Type             | Collation | Nullable | Default
----------+-----------------------------+-----------+----------+---------
 id       | integer                     |           | not null |
 info     | text                        |           |          |
 crt_time | timestamp without time zone |           |          |
Indexes:
    "test_pkey" PRIMARY KEY, btree (id) REPLICA IDENTITY
```

逻辑复制-实践

--如果不设置发布参数，**默认发布：insert, update, delete, truncate**
postgres=# create publication pub1 for table test;
CREATE PUBLICATION
postgres=# select * from pg_publication;
  oid | pubname | pubowner | puballtables | pubinsert | pubupdate | pubdelete | pubtruncate
-------+---------+----------+--------------+-----------+-----------+-----------+-------------
 16409 | pub1 | 10 | f | t | t | t | t
(1 row)

3）订阅端创建订阅：

postgres=# create table test(id int primary key, info text, crt_time timestamp);
CREATE TABLE
--如果不设置parameter，**默认使用：with(copy_data=true, create_slot=true, synchronous_commit=off, connect=true)**
postgres=# create subscription sub_from_pub1 connection 'host=localhost port=5432 user=postgres password=postgres' publication pub1;
2020-03-27 16:52:53.515 CST [48960] LOG: logical decoding found consistent point at 0/16A94C8
2020-03-27 16:52:53.515 CST [48960] DETAIL: There are no running transactions.
NOTICE: created replication slot "sub_from_pub1" on publisher
CREATE SUBSCRIPTION

逻辑复制-查看发布情况

```
postgres=# select * from pg_stat_replication ;
-[ RECORD 1 ]----+------------------------------
--
pid              | 48962
usesysid         | 10
usename          | postgres
application_name | sub_from_pub1
client_addr      | ::1
client_hostname  |
client_port      | 54384
backend_start    | 2020-03-27 16:52:53.533975+08
backend_xmin     |
state            | streaming
sent_lsn         | 0/16B02C0
write_lsn        | 0/16B02C0
flush_lsn        | 0/16B02C0
replay_lsn       | 0/16B02C0
write_lag        |
flush_lag        |
replay_lag       |
sync_priority    | 0
sync_state       | async
reply_time       | 2020-03-27 17:35:43.729501+08
```

```
postgres=# \x
Expanded display is on.
postgres=# select * from pg_replication_slots ;
-[ RECORD 1 ]-------+--------------
slot_name           | sub_from_pub1
plugin              | pgoutput
slot_type           | logical
datoid              | 13543
database            | postgres
temporary           | f
active              | t
active_pid          | 48962
xmin                |
catalog_xmin        | 519
restart_lsn         | 0/16B0288
confirmed_flush_lsn | 0/16B02C0

postgres=# select * from pg_publication;
-[ RECORD 1 ]+------
oid          | 16409
pubname      | pub1
pubowner     | 10
puballtables | f
pubinsert    | t
pubupdate    | t
pubdelete    | t
pubtruncate  | t
```

逻辑复制-查看逻辑复制延迟

```
postgres=# select state,
backend_start,
pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_insert_lsn(), sent_lsn)) send_gap,
pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_insert_lsn(), replay_lsn)) apply_gap,
sent_lsn,
write_lsn,
flush_lsn,
replay_lsn,
sync_state,
reply_time
from pg_stat_replication ;
-[ RECORD 1 ]-+------------------------------
state                | streaming
backend_start        | 2020-03-27 16:52:53.533975+08
send_gap             | 0 bytes
apply_gap            | 0 bytes
sent_lsn             | 0/16B03A8
write_lsn            | 0/16B03A8
flush_lsn            | 0/16B03A8
replay_lsn           | 0/16B03A8
sync_state           | async
reply_time           | 2020-03-27 17:45:29.766881+08
```

逻辑复制-查看订阅情况

查看订阅端属性：

```
postgres=# \x
Expanded display is on.
postgres=# select * from pg_subscription;
-[ RECORD 1 ]---+--------------------------------------------------
-----
oid             | 16409
subdbid         | 13543
subname         | sub_from_pub1
subowner        | 10
subenabled      | t
subconninfo     | host=localhost port=5432 user=postgres
password=postgres
subslotname     | sub_from_pub1
subsynccommit   | off
subpublications | {pub1}

Expanded display is off.
postgres=# select * from pg_replication_origin_status;
 local_id | external_id | remote_lsn | local_lsn
----------+-------------+------------+-----------
        1 | pg_16409    | 0/16B0288  | 0/16BDF78
(1 row)
```

查看订阅端复制情况：

```
postgres=# select
pg_size_pretty(pg_wal_lsn_diff(received_lsn, latest_end_lsn)) ,
* from pg_stat_subscription ;
-[ RECORD 1 ]---------+-----------------------------
pg_size_pretty        | 0 bytes
subid                 | 16409
subname               | sub_from_pub1
pid                   | 48961
relid                 |
received_lsn          | 0/16B03A8
last_msg_send_time    | 2020-03-27 17:52:30.870722+08
last_msg_receipt_time | 2020-03-27 17:52:30.870847+08
latest_end_lsn        | 0/16B03A8
latest_end_time       | 2020-03-27 17:52:30.870722+08
```

remote_lsn指已复制到订阅者的发布者的LSN。
local_lsn指本地已持久化REDO的LSN，只有在这个LSN之前的DIRTY PAGE才能flush到磁盘。

逻辑复制-查看订阅情况

4）发布端插入数据

```
postgres=# insert into test select generate_series(1, 10), 'test', now();
INSERT 0 10
```

5）订阅端看到数据已经同步过来：

```
postgres=# \x
Expanded display is off.
postgres=# select * from test;
 id | info |          crt_time
----+------+----------------------------
  1 | test | 2020-03-27 16:57:59.637935
  2 | test | 2020-03-27 16:57:59.637935
  3 | test | 2020-03-27 16:57:59.637935
  4 | test | 2020-03-27 16:57:59.637935
  5 | test | 2020-03-27 16:57:59.637935
  6 | test | 2020-03-27 16:57:59.637935
  7 | test | 2020-03-27 16:57:59.637935
  8 | test | 2020-03-27 16:57:59.637935
  9 | test | 2020-03-27 16:57:59.637935
 10 | test | 2020-03-27 16:57:59.637935
(10 rows)
```

逻辑复制-确认项

1.订阅表有历史数据会怎么处理？有冲突怎么办，是更新还是如何？

1）订阅表有历史数据，会被保留。
2）当发布端、订阅端都有主键时，发布端插入一条订阅端主键冲突的数据，订阅端会报错，提示主键冲突。当把冲突的行删除掉，发布端插入的数据就同步到订阅端了。冲突时订阅没有中断，**删除行之后，不需要刷新订阅**。
3）发布端有主键、订阅端没有时，插入数据不会冲突，会同步到订阅端。
4）发布端有主键、订阅端没有时，更新上面插入的数据，（发布端主键值在订阅端有两条），订阅端会报错。--当在订阅端删除两条数据中的一条，并创建主键，冲突解决，自动同步过来。
5）那么要同步update、delete，订阅端一定要有主键吗？
是的，订阅端要么有replica indentity，要么有主键，否则会报错:
ERROR: logical replication target relation "public.test1" has neither REPLICA IDENTITY index nor PRIMARY KEY and published relation does not have REPLICA IDENTITY FULL
创建主键后，自动将更新同步过来。

逻辑复制-确认项

2.发布表中有历史数据，会先同步到本地吗，还是只同步订阅开始之后的增量数据?

copy_data (boolean)
Specifies whether the existing data in the publications that are being subscribed to should be copied once the replication starts. The default is true.

逻辑复制-确认项

3.订阅端是一个一个事务接收并触发回放，那么事务回放的顺序就是按照接收的顺序，也就是事务完成的顺序，而不是事务ID顺序?

实际上PostgreSQL的逻辑复制也是流式的，事务没有结束时，发布端事务过程中产生的变更会实时的同步并在订阅端APPLY。
但是先完成的事务会先在订阅端APPLY，有点类似于串行处理事务的感觉。

100 未结束
101 结束，先回放完成

逻辑复制-确认项

4.DDL不会同步，那么发布端添加一列怎么处理的？订阅端添加一列如何处理？

1）发布端加一列，当插入新的数据时，会报错提示订阅端缺少某个列，即使还是插入的原来的列的值。
2）订阅端添加一列，没有影响。

逻辑复制-确认项

5.删除订阅，订阅端的表不会删除，数据也不会删除。

对的

```
postgres=# drop subscription sub_from_pub1 ;
NOTICE:  dropped replication slot "sub_from_pub1" on publisher
DROP SUBSCRIPTION
postgres=# select * from test1;
 id |   name    |          crt_time
----+-----------+----------------------------
  1 | from pub1 |
  2 | from pub1 |
  3 | from pub1 |
  4 | from pub1 |
  5 | from pub1 |
  6 | from pub1 |
  7 | from pub1 |
  8 | from pub1 |
 10 | from pub1 |
  9 | updated   |
 11 | from pub1 |
 12 | from pub1 | 2020-03-30 17:32:54.789677
(12 rows)
```

逻辑复制-确认项

6.删除订阅后，重新订阅，订阅端原来的数据怎么处理？如果订阅端表有多个源，会删除其他源的数据吗？

历史数据如果跟发布端数据有主键冲突，会报错，需要删除冲突数据，或者跳过，或者设置订阅端copy_data为false。

```
postgres=# create subscription sub_from_pub1 connection 'host=localhost user=postgres' publication pub1 with (copy_data=false);
NOTICE:  created replication slot "sub_from_pub1" on publisher
CREATE SUBSCRIPTION
postgres=# select * from test1;
 id |   name    |          crt_time
----+-----------+----------------------------
  1 | from pub1 |
  2 | from pub1 |
  3 | from pub1 |
  4 | from pub1 |
  5 | from pub1 |
  6 | from pub1 |
  7 | from pub1 |
  8 | from pub1 |
 10 | from pub1 |
 13 | from pub1 | 2020-03-30 17:39:54.748625
  9 | updated   |
 11 | from pub1 |
 12 | from pub1 | 2020-03-30 17:32:54.789677
(13 rows)
```

逻辑复制-确认项

7.pg_dump 加上- -include-subscription才会备份订阅信息。

8.如果被订阅的数据有主外键约束，怎么处理？
没有关系，可以被当做普通表进行同步。

truncate操作是否会影响订阅端外键表？不会

```
postgres=# \d test3
                 Table "public.test3"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 id     | integer               |           |          |
 name   | character varying(32) |           |          |
Foreign-key constraints:
    "test3_id_fkey" FOREIGN KEY (id) REFERENCES test2(id)
Publications:
    "pub1"

postgres=# insert into test2 values(2);
INSERT 0 1
postgres=# insert into test3 values(2, 'info');
INSERT 0 1
postgres=# truncate test2 cascade;
NOTICE:  truncate cascades to table "test3"
TRUNCATE TABLE
```

```
postgres=# \d test3
                                  Table "public.test3"
 Column |         Type          | Collation | Nullable | Default | Storage  | Stats target |
Description
--------+-----------------------+-----------+----------+---------+----------+--------------+-------
-+-------------+-------------
 id     | integer               |           |          |         | plain    |              |
 name   | character varying(32) |           |          |         | extended |              |

postgres=# select * from test3;
 id | name
----+------
  1 | info
  2 | info
(2 rows)
```

逻辑复制-确认项

9. 因为逻辑复制在订阅端实际上是逻辑写操作，请尽量避免锁冲突。比如truncate , alter table

10. 在订阅端，delete, update 不存在记录，会不会报错，会不会中断订阅，例如没有同步历史数据，在做删除和更新是没有数据的。

不会

11.逻辑复制全是异步的吗？

在订阅创建时，可以指定，默认是异步，不过同步复制延迟很大
synchronous_commit (enum)

12.逻辑复制的延迟大不大？

很大

逻辑复制-确认项

13.对发布端添加一个张表，订阅端要刷新订阅。如果设置的是全部发布，需要刷新吗？

需要

发布端：

```
postgres=# drop publication pub1 ;
DROP PUBLICATION
postgres=# create publication pub1 for all tables ;
CREATE PUBLICATION
postgres=# create table test5(id int, crt_time timestamp);
CREATE TABLE
postgres=# \d test5
                    Table "public.test5"
  Column  |            Type             | Collation | Nullable | Default
----------+-----------------------------+-----------+----------+--
 id       | integer                     |           |          |
 crt_time | timestamp without time zone |           |          |
Publications:
    "pub1"

postgres=# insert into test5 values(1, now());
INSERT 0 1
```

订阅端：

```
postgres=# create table test5(id int, crt_time timestamp);
CREATE TABLE
postgres=#
postgres=# select * from test5;
 id | crt_time
----+----------
(0 rows)

postgres=# alter subscription sub_from_pub1 refresh publication ;
ALTER SUBSCRIPTION
postgres=# select * from test5;
 id |         crt_time
----+----------------------------
  1 | 2020-03-30 18:02:33.988226
(1 row)
```

逻辑复制-确认项

14.默认发布端会发布truncate，如果订阅端有自己插入的数据，会不会被删除？

验证结论：会，关联的表也会被重置

1）发布端truncate test表；
2）订阅端插入1条数据；
3）发布端插入10条数据；
4）发布端truncate test表；
5）订阅端查询，test表数据会被清空；

```
postgres=# truncate test;
TRUNCATE TABLE
postgres=# insert into test select generate_series(1, 10), 'test', now();
INSERT 0 10
postgres=# truncate test;
TRUNCATE TABLE
postgres=#
```

```
postgres=# insert into test values(100, 'test other', now());
INSERT 0 1
postgres=# select * from test;
 id  |    info    |          crt_time
-----+------------+----------------------------
 100 | test other | 2020-03-27 17:04:20.782016
   1 | test       | 2020-03-27 17:04:28.446478
   2 | test       | 2020-03-27 17:04:28.446478
   3 | test       | 2020-03-27 17:04:28.446478
   4 | test       | 2020-03-27 17:04:28.446478
   5 | test       | 2020-03-27 17:04:28.446478
   6 | test       | 2020-03-27 17:04:28.446478
   7 | test       | 2020-03-27 17:04:28.446478
   8 | test       | 2020-03-27 17:04:28.446478
   9 | test       | 2020-03-27 17:04:28.446478
  10 | test       | 2020-03-27 17:04:28.446478
(11 rows)

postgres=# select * from test;
 id | info | crt_time
----+------+----------
(0 rows)
```

PG13新特性

PostgreSQL 13 版本的逻辑复制新增对分区表的支持，如下:
1、可以显式地对分区表进行发布，自动发布所有分区。
2、从分区表中添加或删除分区将自动从发布者中添加或删除。

Allow partitioned tables to be logically replicated via publications (Amit Langote)
Previously, partitions had to be replicated individually. Now partitioned tables can be published explicitly causing all partitions to be automatically published. Addition/removal of partitions from partitioned tables are automatically added/removed from publications. The CREATE PUBLICATION option publish*via*partition_root controls whether changes to partitions are published as their own or their ancestors.
Allow logical replication into partitioned tables on subscribers (Amit Langote)
Previously, subscribers could only receive rows into non-partitioned tables.