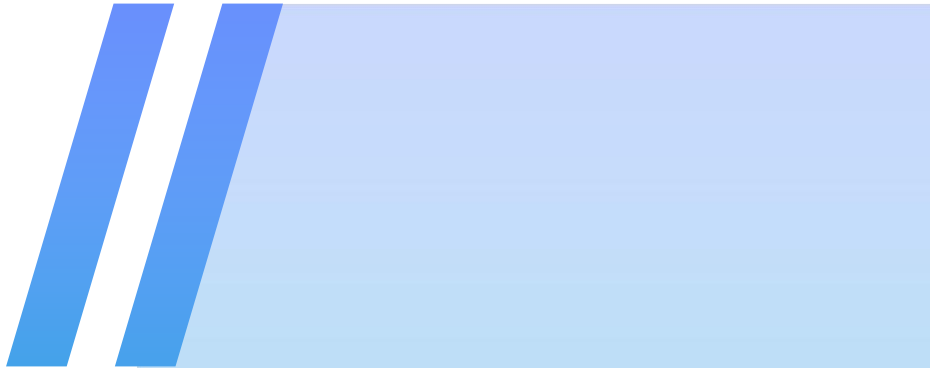
PostgreSQL15 beta1新特性解读

墨天轮

个人简介：尚长军 中兴通讯股份有限公司数据库内核系统工程师，主要从事数据库内核的设计，研发和规划工作。

目录

- 应用
- 工具
- Contrib

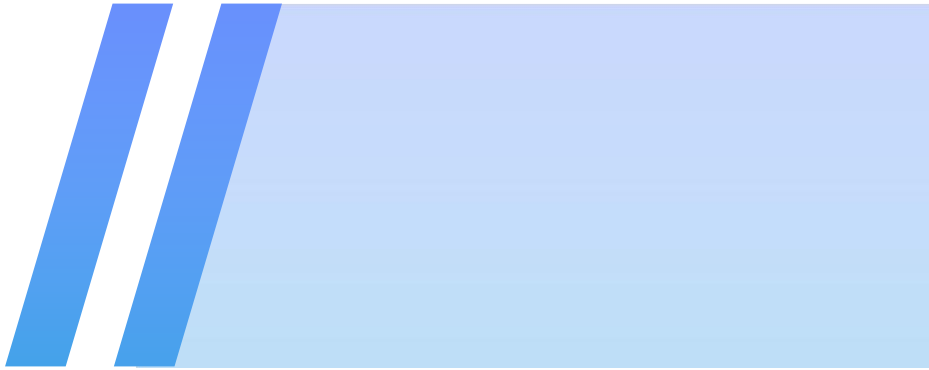


01 应用

Sort排序

◘ 单列排序性能的改进

pg14的查询执行器在对算子执行排序时，必须存储整个tuple。pg15对此进行了优化，即排序算子的结果仅有一列时，仅存储一个Datum，不会拷贝整个tuple到排序的内存，场景如下：



SELECT col1 FROM tab ORDER BY col1;　　①

SELECT col1, col2 FROM tab ORDER BY col1;　　②

①因为结果仅返回一列，应用了此优化，而②则不能。

通过对比测试10,000行整数值排序的性能，仅存储Datum性能提升很明显：pg15比pg14快了近26%。

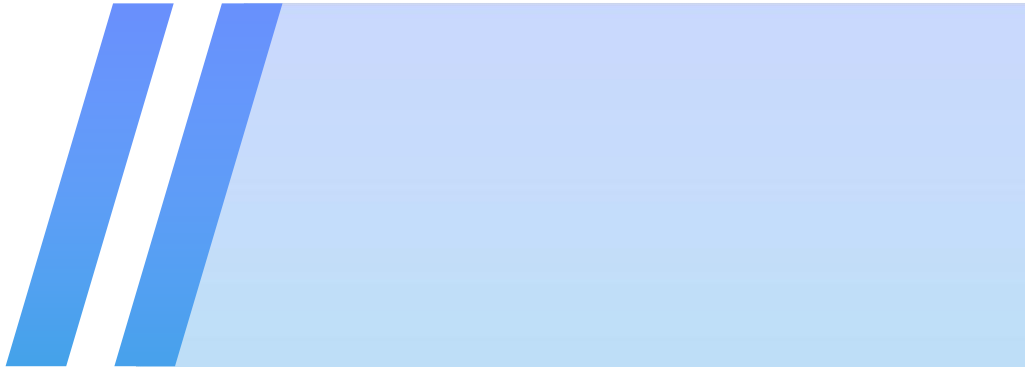
Sort排序

- generation memory context减小内存消耗

pg14及更早版本中，使用“aset”（AllocSet）内存分配器分配给排序记录所需内存：

- 内存分配器分配的内存充当PG和底层操作系统之间的缓冲区
- 将内存分配请求的大小向上取整为2的下一个幂
- PG除非排序数据量很大且要溢出到磁盘时，才会释放排序所占用的内存。

“aset”这种内存分配大小的四舍五入规则会导致平均25%的内存浪费。



pg15采用“generation”的内存分配器：

- 不维护任何空闲链表
- 不四舍五入分配大小
- 假设分配模式是先进先出的
- 当每个block的所有chunk不再需要时，依赖于释放完整的blocks。

“generation”这种不四舍五入的内存分配，可以以更少的内存存储更多记录。这种变化能提高多少性能取决于存储的元组的宽度。略高于 2 次方的元组大小提高最多。例如，pg14 会将一个 36 字节的元组四舍五入为 64 字节——接近所需内存的两倍，而已经是 2 次方大小的元组的性能提升最少。

Sort排序

◘ 为常见的数据类型添加专门的排序routine

pg有很多数据类型，每种数据类型都有一个比较函数，用于排序时的两个值的比较，就产生了开销。pg15增加了一组新的快速排序函数，这些快速排序函数的特点是具有内联编译的比较函数，以消除比较函数的调用开销。可以通过以下方式检查 pg15 中排序的数据类型是否使用这些新的快速排序函数之一：

```
postgres=# set client_min_messages TO 'debug1';
SET
postgres=# explain analyze select id from test order by id;
```

如果调试消息显示：

- qsort_tuple_unsigned
- qsort_tuple_signed
- qsort_tuple_int32

则说明采用了快速排序函数，如果调试消息显示其他内容，则排序使用原始（较慢）快速排序功能。添加的这 3 个快速排序函数不仅适合一些常见的数据类型，而且还涵盖了时间戳和所有使用缩写键的数据类型，其中包括使用 C 排序规则的 TEXT 类型。

Sort排序

用多路归并(N-way merge)替代多阶段合并(polyphase merge)算法

- N-way merge算法：算法思想是采用N个输入设备和输出设备，读取N个输入，并归并产生N个输出，这里的N-way是指可以同时对N个设备进行处理，即并发。
- polyphase merge算法：与N-way类似，但只需要一个空闲设备，可以节省空间，但并发性降低。

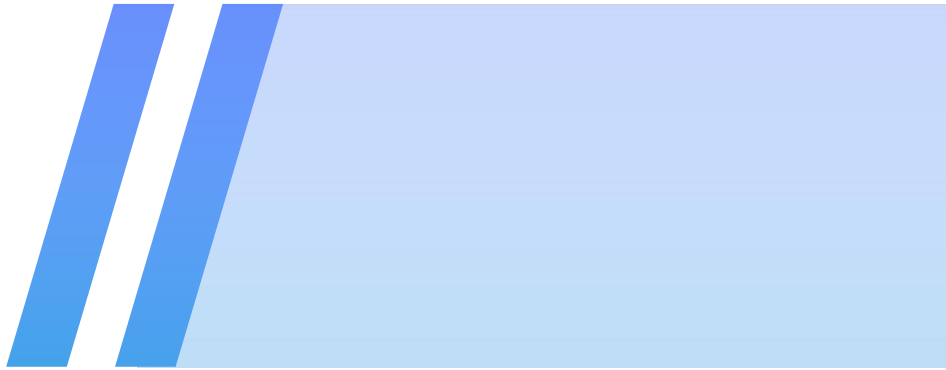
pg15针对超过work_mem大小的更大规模的排序，由合并单个tape的算法已更改为使用多路合并。当tape数量很大时，所需的 I/O 比原来的多阶段合并算法要少。

这种优化对大型排序的执行速度提升了近43%。针对这种work_mem较小且需要进行大数据量排序时，pg 15 比pg14具有更高性能。 随着work_mem尺寸的增加，性能差距缩小。

WAL压缩算法

pg15之前的版本只有一种压缩类型pglz，且默认是关闭的。pg15增加了lz4和zstd两种新的压缩算法，可以通过--wal_compression选项加以指定。

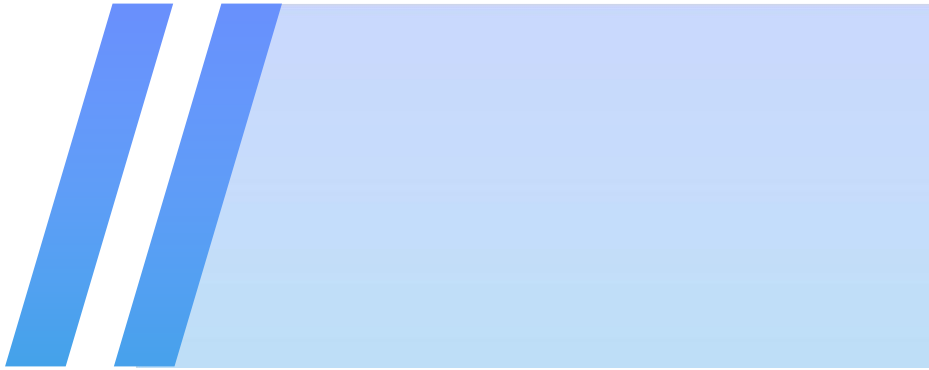
```
$ ./configure --with-zstd --with-lz4
postgres=# select name, setting, vartype, enumvals from pg_settings where
name='wal_compression' ;
-[ RECORD 1 ]--------------
name | wal_compression
setting | off
vartype | enum
enumvals | {pglz,lz4,zstd,on,off}
```



\Dconfig命令

\dconfig命令可以显示实例的参数设置。如果不指定任何选项，将显示已从默认值更改的参数列表。如果输入参数名，可以查看具体的参数值。参数名可以使用通配符，如果需要显示详细信息，使用\dconfig+，\dconfig命令示例如下：

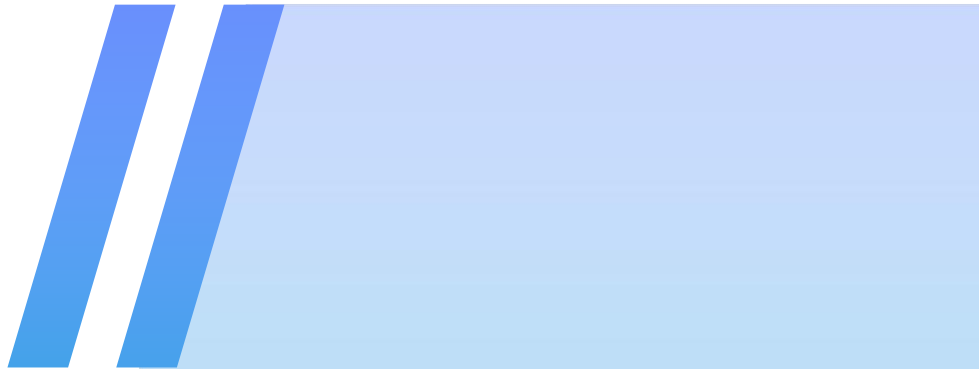
```
postgres=> \dconfig work_mem
List of configuration parameters
Parameter | Value
----------+-------
work_mem | 4MB
(1 row)
postgres=> \dconfig log_*connect*
List of configuration parameters
 Parameter | Value
-------------------+-------
log_connections | off
log_disconnections | off
(2 rows)
```



“--”注释

pg15针对SQL语句中的注释做了一些改变：
SQL语句中的“--”注释内容，在pg15之前的版本，是不记录在服务器端的日志中的，而pg15中则将注释的内容记录到日志中了。

```
postgres=# select 'shangchangjun',--that's good
postgres-# 2;
   ?column?    | ?column?
---------------+----------
 shangchangjun |        2
(1 row)

2022-06-14 18:43:49.932 JST [20606] LOG:  statement: SELECT
'shangchangjun', --that's good 2;
```



\watch命令

\watch是一种监控SQL语句动态执行的分页显示工具。pg15针对 \watch 命令增加了寻呼机选项 (Pavel Stehule, Thomas Munro)。由 PSQL_WATCH_PAGER 环境变量控制，在环境变量中设置分页器的命令：export PSQL_WATCH_PAGER="less”

```
postgres=# SELECT generate_series(1,100), clock_timestamp();
 generate_series |       clock_timestamp
-----------------+-------------------------------
               1 | 2022-06-20 10:26:18.249584+08
               2 | 2022-06-20 10:26:18.249605+08
               3 | 2022-06-20 10:26:18.249607+08
               .....
(100 rows)
postgres=# \watch 1
   Mon 20 Jun 2022 10:26:44 AM CST (every 1s)
 generate_series |       clock_timestamp
-----------------+-------------------------------
               1 | 2022-06-20 10:26:44.552742+08
               2 | 2022-06-20 10:26:44.552754+08
               3 | 2022-06-20 10:26:44.552756+08
               .....
 :10
```

以每次10条记录向下翻。

\getenv命令

pg15增加了\getenv命令来获取环境变量。指定psql变量名称和环境变量名称以保存变量的值。如果指定了不存在的环境变量名称也不会导致错误。

```
\getenv psql_variable_name Environment_variable_name

postgres=> \getenv home HOME
postgres=> \echo :home
/home/postgres

postgres=> \getenv val BADENV
postgres=> \echo :val
:val
```

SQL输出结果的设置

pg15之前的版本对带“\”SQL查询，默认服务器返回多个结果集，pg15版本可通过“\set SHOW_ALL_RESULTS off”命令，只显示最后的结果。

```
postgres=# SELECT 1 \; SELECT 2 \; SELECT 3;
 ?column?
----------
        1
(1 row)
 ?column?
----------
        2
(1 row)
 ?column?
----------
        3
(1 row)
postgres=# \set SHOW_ALL_RESULTS off
postgres=# SELECT 1 \; SELECT 2 \; SELECT  3;
 ?column?
----------
        3
(1 row)
```

02 工具

pg_basebackup

pg15在pg_basebackup命令中扩展了以下选项：

- Target选项

pg15增加了用于指定备份输出目的地的--target选项。
可以指定以下格式：

目标	描述
blackhole	无输出。测试选项
server:/path	输出到服务器上的指定目录。目前，输出格式为tar（输出到/左示例中的路径）。

服务器端备份可由具有SUPERUSER或pg_write_server_files角色的用户执行。

pg_basebackup

压缩选项

pg15中通过--compress选项，在可压缩性方面做了一些扩展：

- 增加了lz4，ztsd压缩算法
- 既可以指定压缩算法，又可以指定压缩级别

如果服务器端压缩(server-gzip)指定为(-Fp)纯格式，则仅对传输的数据进行压缩和解压缩并存储在客户端。此设置在网络带宽较小时有用。

设置	描述	备注
none	不压缩	
gzip	默认客户端Gzip压缩（如果没有指定）	
lz4	默认客户端LZ4压缩（如果没有指定）	
zstd	客户端的Zstandard压缩（如果不是目标）	
client-gzip	客户端侧gzip压缩	
server-gzip	服务器侧Gzip压缩	
client-lz4	客户端侧LZ4压缩	
server-lz4	服务器侧LZ4压缩	
client-zstd	客户端侧Zstandard压缩	
server-zstd	服务器侧的Zstandard压缩	

pg_basebackup

- 并行服务器端备份压缩

pg15支持ztsd多线程并行压缩。尽管lz4 的压缩不如 gzip，但它是最快的单线程算法。单线程ztsd比 gzip 压缩得更好更快，但不如单线程 lz4快。多线程的ztsd压缩明显快于lz4.

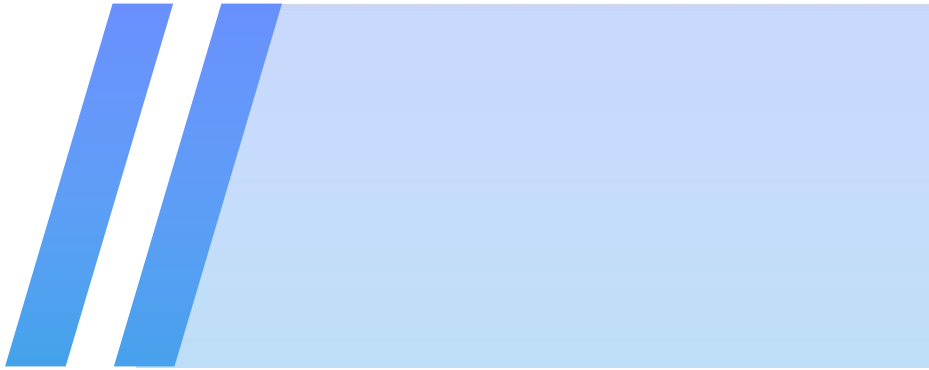
命令示例：

```
$ pg_basebackup --compress=none --format=tar -D back.1
$ pg_basebackup --compress=zstd:level=9,workers=2 --format=tar -D back.2
```

pg_amcheck

pg_amcheck是pg的数据和索引逻辑错误检测插件。pg14增加对heap表的数据页逻辑错误检测功能。

pg15新增--install-missing=schema选项，若指定此选项，如果应用此命令时检测到尚未安装amcheck，则能够自动安装到指定的schema中。如果未指定schema，则安装到pg_catalog下。



pg_recvlogical

pg_recvlogical是一种wal日志解析工具，主要用于获取数据库的增量更新信息。pg15在pg_recvlogical命令中增加了--two--phase选项，以支持在创建复制槽的时候，增加对two-phase事务日志的解析，示例如下：

```
$ pg_recvlogical --create-slot --slot=slot1 --two-phase --dbname=postgres
$ psql
postgres=# SELECT two_phase FROM pg_replication_slots WHERE
slot_name='slot1' ;
two_phase
----------
t
```

pg_receivewal

pg15针对采用流复制协议进行wal日志转储工具pg_receivewal的--compress选项做了一些改进，新增了lz4压缩算法，并且既可以指定压缩方法，也可以指定压缩级别。其命令格式如下：

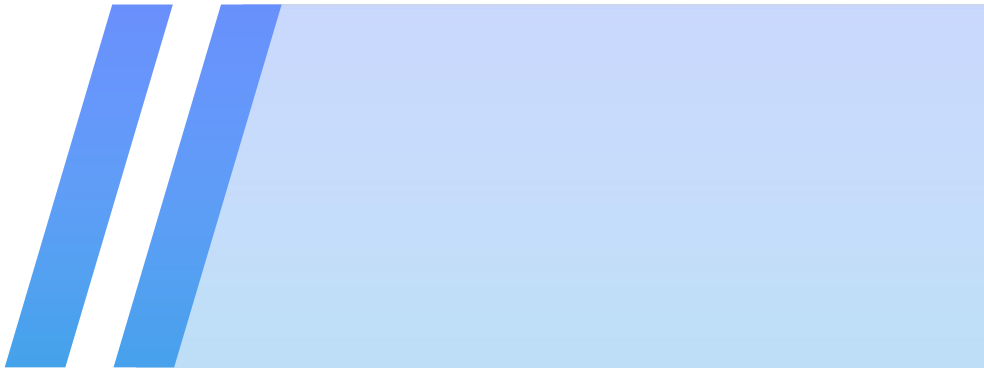
```
$pg_receivewal --compress=METHOD[:DETAIL]

METHOD = {'gzip', 'none', 'lz4'}

DETAIL = level=[1-9]
```

压缩的有效选项为gzip、lz4或none。可指定可选压缩级别，压缩级别的范围是1-9。

pg_receivewal

应用示例：

```
$ pg_receivewal -D wal --compress=lz4:level=9
^Cpg_receivewal: not renaming "000000010000000000000028.lz4.partial",
segment
is not complete
$ ls wal
000000010000000000000022.lz4 000000010000000000000026.lz4
000000010000000000000023.lz4 000000010000000000000027.lz4
000000010000000000000024.lz4 000000010000000000000028.lz4.partial
000000010000000000000025.lz4
```



pg_resetwal

pg_resetwal是一种wal日志强制重置的工具。可以通过截断日志方式强行恢复数据库。在pg15中增加了--oldest-transaction-id（缩写-u）选项，可以手工指定最早的事务ID。格式：

--oldest-transaction-id=xid。

确定安全值的方法：在数据目录下的pg_xact目录中查找最小的数字文件名，然后乘以1048576（0x100000）。

注意，文件名是十六进制的。通常用十六进制指定选项值是最容易的。

例如，如果0007是pg_xact中的最小条目，-u 0x700000将起作用。

pg_rewind

pg_rewind是一种主持数据库间数据目录的同步工具，其特点：

- 只复制表数据文件中更改的块，而其他所有文件都会被完整复制
- 不需要读取数据库中未更改的块，在只有小的更新情况下，速度会非常快

主要用于在集群的时间线发生偏离后，主从同步失败的修复。一个典型的场景是，在故障切换后，将旧的主服务器作为新主服务器之后的备用服务器重新联机。

pg15中增加了-config-file选项以指定服务器配置文件postgresql.conf的路径。当配置文件在数据目录之外时，此选项很有用。

pg_upgrade

pg15针对大版本升级的pg_upgrade命令进行了做了一些优化：

- -No-sync选项

增加--no-sync选项（缩写-N）。默认情况下，pg_upgrade要等待升级集群的所有文件安全写入磁盘。此选项会使得pg_upgrade可以不进行上述等待的情况下返回，速度更快，但意味着随后的系统崩溃可能会导致数据目录损坏。此选项主要应用于测试，避免在生产环境使用。

- 临时目录

升级期间创建的临时文件将在新数据库群集的pg_upgrade_output.d目录中创建。

pg_waldump

作为一种wal日志分析工具，pg15中针对pg_waldump命令增加了如下选项：

- 可选项

pg15新增加以下选项，可以指定多个--rmgr选项。

选项名称	缩写	描述
--block=N	-B	显示--relation选项指定的表的块信息
--fork=N	-F	按照fork number过滤
--relation=N/N/N	-R	按照relation file node过滤
--fullpage	-w	仅显示全页写信息

pg_waldump

信号

当接收到SIGINT信号（如Ctrl+C）过程结束时，输出的汇总信息终止于结束时间点。

增加输出信息

在输出结果中增加事务提交的remote_apply信息和PREPARE TRANSACTION的origin的信息。

```
$ pg_waldump data/pg_wal/000000010000000000000008 | grep origin
rmgr: Transaction len (rec/tot): 117/ 117, tx: 775, lsn:
0/080025F0, prev 0/080025C0, desc: COMMIT 2022-05-20 10:01:46.390500 JST;
inval
msgs: catcache 57 catcache 56 catcache 66; origin: node 2, lsn 0/0, at 2022-05-
20 10:01:46.382305 JST
rmgr: Transaction len (rec/tot): 85/ 85, tx: 776, lsn:
0/08002770, prev 0/08002740, desc: COMMIT 2022-05-20 10:01:46.391231 JST;
inval
msgs: catcache 66; origin: node 2, lsn 0/0, at 2022-05-20 10:01:46.382305 JST
…
```

03 Contrib模块

Basebackup_to_shell

pg15增加了basebackup_to_shell插件，pg_basebackup中相应增加--target选项：pg_basebackup --target=shell 。
使用此功能时，需要在系统参数配置文件中的shared_preload_library中指定basebackup_to_shell。
可以指定以下参数：

参数名	描述
basebackup_to_shell.命令	要执行的命令路径
basebackup_to_shell.required_role	执行所需的角色

File_fdw

pg15在create foreign table中的header选项中指定match值。此选项检查文本文件中的标题行和列名之间的匹配。

```
postgres=# \! cat /tmp/data1.csv
c1,c3
1,data1
2,data2
postgres=# create extension file_fdw ;
CREATE EXTENSION
postgres=# create server filesvr1 foreign data wrapper file_fdw ;
CREATE SERVER
postgres=# create foreign table head1 (c1 int, c2 text) server filesvr1
options (format 'csv', filename '/tmp/data1.csv', delimiter ',',
header 'match') ;
CREATE FOREIGN TABLE
postgres=#select * from  head1 ;
ERROR: column name mismatch in header line field 2: got "c3", expected "c2"
CONTEXT: COPY head1, line 1: "c1,c3"
```

pg_stat_statements

- 临时文件的I/O信息

pg15增加了对临时文件的读/写时间的监控。已将以下列添加到pg_stat_statements视图中。

列名称	数据类型	描述
temp_blk_read_time	双精度	临时块读取时间
temp_blk_write_time	双精度	临时块写入时间

pg_stat_statements

- 增加JIT相关信息

即时（Just-In-Time，JIT）编译是将某种形式的解释程序计算转变成原生程序的过程，并且这一过程是在运行时完成的。例如，与使用能够计算任意SQL表达式的通用代码来计算一个特定的SQL谓词（如WHERE a.col = 3）不同，可以产生一个专门针对该表达式的函数并且可以由CPU原生执行，从而得到加速。

PG15增加了对JIT相关信息的监控，pg_stat_statements视图中增加如下监控列：

列名称	数据类型	描述
jit_function	bigint	JIT编译函数执行次数
jit_generated_time	double precision	JIT代码生成时间（毫秒）
jit_inlining_count	bigint	内联函数的执行次数
jit_inlining_time	double precision	内联函数执行时间（毫秒）
jit_optimization_count	bigint	优化的语句执行计数
jit_optimization_time	double precision	优化语句执行时间（毫秒）
jit_emission_count	bigint	代码发布次数
jit_emission_time	double precision	代码发布的时间（毫秒）

pg_walinspect

pg15新增的pg_walInspection模块提供了以SQL函数的方式查询wal日志信息的功能。实现类似pg_waldump的功能。这些函数只能由具有superuser属性或pg_read_server_files角色的用户执行。

函数名称	描述
pg_get_wal_record_info	返回指定LSN的wal信息
pg_get_wal_records_info	返回指定LSN之间的wal信息
pg_get_wal_record_info_till_end_of_wal	从指定的LSN一直到最后返回wal信息
pg_get_wal_stats	返回指定LSN之间的wal统计占比信息（统计占比，比如heap，全页写，索引wal日志占比）
pg_get_wal_stats_till_end_of_wal	返回从指定LSN到末尾的WAL统计占比信息

pg_walinspect

应用示例：

```
postgres=# CREATE EXTENSION pg_walinspect ;
CREATE EXTENSION

postgres=# select start_lsn, end_lsn, prev_lsn,xid, resource_manager, record_type,
record_length,main_data_length, fpi_length, description from
pg_get_wal_records_info('0/14F9A30', '0/15011D7');
 start_lsn | end_lsn | prev_lsn | xid | resource_manager |record_type | record_length |
main_data_length | fpi_length |description
-----------+-----------+-----------+-----+------------------+-------------+--------------+-----------------+-
-
 0/14FA118 | 0/14FB4B0 | 0/14F9958 | 725 | Btree |INSERT_LEAF | 5013 | 2 | 4960 |off 246
 0/14FB4B0 | 0/14FD050 | 0/14FA118 | 725 | Btree |INSERT_LEAF | 7045 | 2 | 6992 |off 130
 0/14FD0A8 | 0/14FD0F0 | 0/14FD050 | 725 | Btree |INSERT_LEAF | 72 | 2 | 0 |off 155
 0/14FD0F0 | 0/14FD138 | 0/14FD0A8 | 725 | Btree |INSERT_LEAF | 72 | 2 | 0 |off 134
 0/14FD138 | 0/14FD210 | 0/14FD0F0 | 725 | Heap |INSERT | 211 | 3 | 0 |off 11 flags 0x00
 0/14FD210 | 0/14FD250 | 0/14FD138 | 725 | Btree |INSERT_LEAF | 64 | 2 | 0 |off 246
```

pg_walinspect

应用示例：

```
postgres=# select * from pg_get_wal_stats('0/14AFC30','0/15011D7', true) where count > 0;
resource_manager/record_type | count | count_percentage| record_size | record_size_percentage
| fpi_size |fpi_size_percentage | combined_size | combined_size_percentage
-----------------------------+-------+-----------------+-------------+-----------------------+----------+---
---

 XLOG/CHECKPOINT_SHUTDOWN | 1 | 0.32894737 |114 | 0.22891566 | 0 | 0| 114 |
0.03534489
 XLOG/CHECKPOINT_ONLINE | 4 | 1.3157895 |456 | 0.91566265 | 0 | 0| 456 | 0.14137957
 XLOG/NEXTOID | 1 | 0.32894737 |30 | 0.060240965 | 0 | 0| 30 | 0.009301287
 Transaction/COMMIT | 9 | 2.9605262 |1173 | 2.3554218 | 0 | 0 | 1173 | 0.36368033
 Storage/CREATE | 1 | 0.32894737 | 42 | 0.084337346 | 0 | 0| 42 | 0.0130218025
 Database/CREATE_FILE_COPY | 2 | 0.65789473 |84 | 0.16867469 | 0 | 0| 84 | 0.026043605
 Standby/RUNNING_XACTS | 6 | 1.9736842 |316 | 0.6345382 | 0 | 0 | 316 | 0.09797356
 Standby/INVALIDATIONS | 45 | 14.802631 |4018 | 8.068274 | 0 | 0 | 4018 | 1.2457525
```

postgres_fdw

PG15中对postgres_fdw进行功能增强和优化：

◘ 指定application_name

可以为远程连接指定application_name。可以更改postgres_fdw.application_name参数，示例如下：

```
postgres=# SET postgres_fdw.application_name TO 'fdw_name1' ;
SET
postgres=# CREATE SERVER svr5432 FOREIGN DATA WRAPPER postgres_fdw
OPTIONS
(host 'remhost1', port '5432', dbname 'postgres') ;
CREATE SERVER
```

postgres_fdw

可以在application name中指定以下转义符：

转义序列	描述	备注
%a	本地应用名称	
%c	会话ID	
%C	集群名称（cluster_name）	
%d	本地数据库名称	
%u	本地用户名	
%p	后端进程ID	
%%	%characters	

Postgres_fdw

◘ Parallel_commit选项

pg15增加了parallel_commit选项以控制如何执行远程事务提交。将此选项设置为“true”允许远程事务并行提交。默认为“关闭”，即串行执行提交，应用示例如下：

```
postgres=# CREATE SERVER svr5433 FOREIGN DATA WRAPPER postgres_fdw
OPTIONS
( host 'remhost1', port '5433', dbname 'postgres', parallel_commit 'true') ;
CREATE SERVER

postgres=# SELECT srvoptions FROM pg_foreign_server WHERE srvname='svr5433' ;
 srvoptions
-----------------------------------------------------------------------------------
{host=localhost,port=5433,dbname=postgres,parallel_commit=true}
(1 row)
```

Postgres_fdw

case表达式

CASE表达式现在推送到远程实例，示例如下：

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT COUNT(*) FROM remote1
WHERE
CASE WHEN c1 > 100 THEN c1 END < 100 ;
 QUERY PLAN
-----------------------------------------------------------------------------------------------
Foreign Scan (cost=102.84..164.07 rows=1 width=8) (actual time=1.364..1.365
rows=1 loops=1)
 Output: (count(*))
 Relations: Aggregate on (public.remote1)
 Remote SQL: SELECT count(*) FROM public.remote1 WHERE (((CASE WHEN (c1 >
100::numeric) THEN c1 ELSE NULL::numeric END) < 100::numeric))
Planning Time: 0.077 ms
Execution Time: 1.620 ms
(6 rows)
```

可以放联系方式等

墨天轮 观看视频回放