认识PostgreSQL12

姜瑞海
ruihaijiang@msn.com
http://ruihaijiang.net

内容纲要

- PostgreSQL简介
- 快速安装使用
  - 开发者如何使用PostgreSQL
  - CentOS8安装使用PostgreSQL12
  - 数据对象层次结构
  - 简单的用户管理策略
- 数据类型
  - 内置的数据类型
  - JSON举例
  - XML举例
- 特异功能表
  - 表继承
  - 分区表
  - 外部表
- 索引/Index
- 视图/View
- 过程语言/Procedural Language
- 编程接口
- PostgreSQL内幕
  - 内存和文件
  - MVCC((Multi-Version Concurrency Control) /WAL(Write-Ahead Logging)
  - 流复制(Stream Replication)
- 数据备份和恢复
  - pg_dump, pg_dumpall
  - 基于文件系统的备份
  - 增量备份
  - 辅助工具
- 高可用
- 应用场景
  - 单节点
  - 一主多从
  - 分布式集群

PostgreSQL简介

12 (2019-10-03)	8.0 (2005-01-19)
11 (2018-10-18)	7.4 (2003-11-17)
10 (2017-10-05)	7.3 (2002-11-27)
9.6 (2016-09-29)	7.2 (2002-02-04)
9.5 (2016-01-07)	7.0 (2000-05-08)
9.4 (2014-12-18)	6.5 (1999-06-09)
9.3 (2013-09-09)	6.4 (1998-10-30)
9.2 (2012-09-10)	6.3 (1998-03-01)
9.1 (2011-09-12)	6.2 (1997-10-02)
9.0 (2010-09-20)	6.0 (1997-01-29)
8.4 (2009-07-01)	1.0 (1995-0A
8.3 (2008-02-04)	
8.2 (2006-12-05)	
8.1 (2005-11-08)	

官网：https://www.postgresql.org

中文社区：http://www.postgres.cn/

功能表 https://www.postgresql.org/about/featurematrix/

官方下载：https://www.postgresql.org/download/

在线文档：https://www.postgresql.org/docs/

在线其它学习资源：https://www.postgresql.org/docs/online-resources/

官网代码库：https://git.postgresql.org/gitweb/?p=postgresql.git;a=summary

官网邮件列表：https://www.postgresql.org/list/

官网的技术分享：https://wiki.postgresql.org/wiki/Main_Page

- 开源/免费，关系型数据库
- 功能强大，性能卓越，稳定可靠
- 用户众多，全世界都在用它
- 被众多顶级公司使用、改造
- 强力的社区支援
- 众多的服务提供者

PostgreSQL简介：谁在使用PostgreSQL

Who uses PostgreSQL?

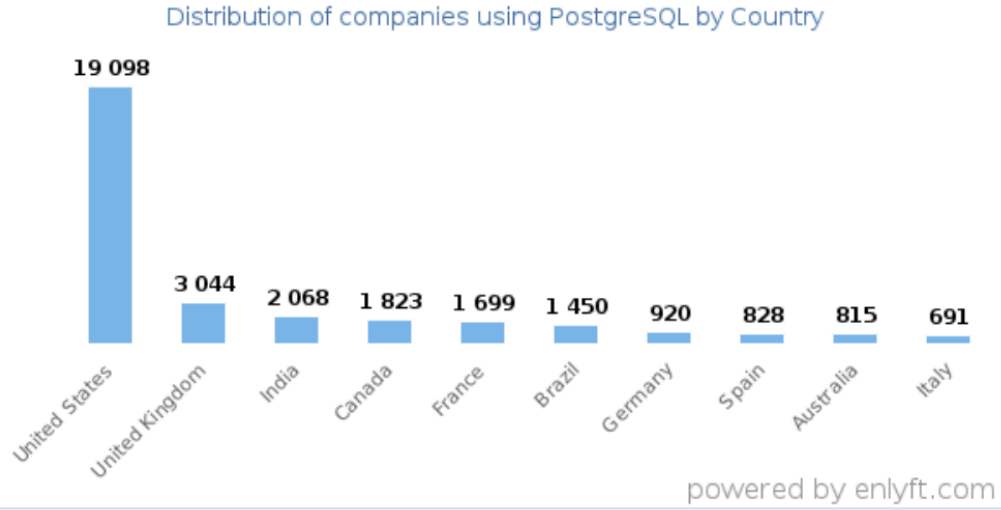
List of the top companies using PostgreSQL :

Company	Website	Country	Revenue	Company Size
Udacity, Inc.	udacity.com	United States	50M-100M	200-500
Red Hat Inc	redhat.com	United States	>1000M	>10000
TripAdvisor Inc	tripadvisor.com	United States	>1000M	1000-5000
Capital Numbers Infotech Pvt Ltd	capitalnumbers.com	India	50M-100M	200-500
Gilt Groupe, Inc.	gilt.com	United States	100M-200M	500-1000

PostgreSQL简介：谁在使用PostgreSQL

Top Countries that use PostgreSQL

40% of PostgreSQL customers are in United States and 7% are in United Kingdom.



PostgreSQL简介：谁在使用PostgreSQL

Companies that use PostgreSQL, by industry :

Industry	Number of companies
Computer Software	10940
Information Technology and Services	5532
Staffing and Recruiting	1625
Internet	1287
Hospital & Health Care	1176
Higher Education	1087
Financial Services	944
Telecommunications	843
Computer Hardware	802
Marketing and Advertising	791

PostgreSQL简介：PostgreSQL功能点概括

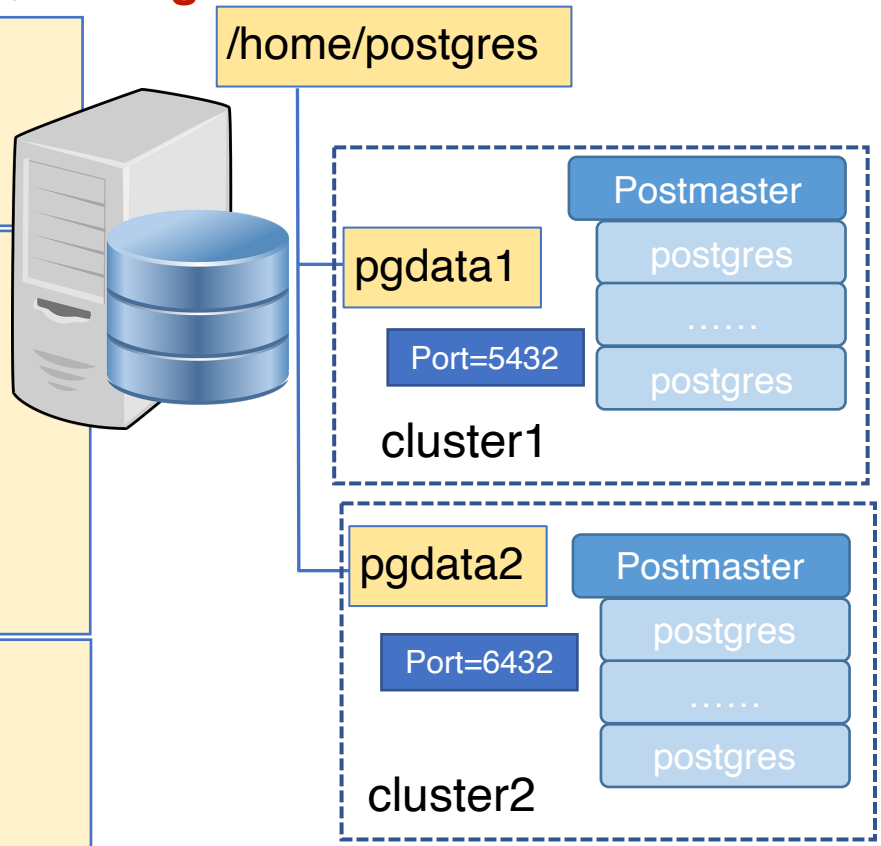
- **Data Types**
  - Primitives: Integer, Numeric, String, Boolean
  - Structured: Date/Time, Array, Range, UUID
  - Document: JSON/JSONB, XML, Key-value (Hstore)
  - Geometry: Point, Line, Circle, Polygon
  - Customizations: Composite, Custom Types
- **Data Integrity**
  - UNIQUE, NOT NULL
  - Primary Keys
  - Foreign Keys
  - Exclusion Constraints
  - Explicit Locks, Advisory Locks
- **Concurrency, Performance**
  - Indexing: B-tree, Multicolumn, Expressions, Partial
  - Advanced Indexing: GiST, SP-Gist, KNN Gist, GIN, BRIN, Covering indexes, Bloom filters
  - Sophisticated query planner / optimizer, index-only scans, multicolumn statistics
  - Transactions, Nested Transactions (via savepoints)
  - Multi-Version concurrency Control (MVCC)
  - Parallelization of read queries and building B-tree indexes
  - Table partitioning
  - All transaction isolation levels defined in the SQL standard, including Serializable
  - Just-in-time (JIT) compilation of expressions
- **Reliability, Disaster Recovery**
  - Write-ahead Logging (WAL)
  - Replication: Asynchronous, Synchronous, Logical
  - Point-in-time-recovery (PITR), active standbys
  - Tablespaces
- **Security**
  - Authentication: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificate, and more
  - Robust access-control system
  - Column and row-level security
  - Multi-factor authentication with certificates and an additional method
- **Extensibility**
  - Stored functions and procedures
  - Procedural Languages: PL/PGSQL, Perl, Python (and many more)
  - SQL/JSON path expressions
  - Foreign data wrappers: connect to other databases or streams with a standard SQL interface
  - Customizable storage interface for tables
  - Many extensions that provide additional functionality, including PostGIS
- **Internationalization, Text Search**
  - Support for international character sets, e.g. through ICU collations
  - Case-insensitive and accent-insensitive collations
  - Full-text search

快速安装使用：一个开发者这样使用PostgreSQL

```
tar zxvf postgresql-9.6.3.tar.gz
cd postgresql-9.6.3
./configure
make
sudo make install
```

```
Linux user=postgres

cd /home/postgres
mkdir  pgdata1
/usr/local/pgsql/bin/initdb –D ./pgdata1
/usr/local/pgsql/bin/pg_ctl –D ./pgdata1 start –l pglog1

./pgdata1/postgresql.conf: port = 5432

/usr/local/pgsql/bin/psql –p 5432 => create table t(id int, name varchar(50));
/usr/local/pgsql/bin/pg_ctl –D ./pgdata1 stop
```

```
cd /home/postgres
mkdir  pgdata2
/usr/local/pgsql/bin/initdb –D ./pgdata2
/usr/local/pgsql/bin/pg_ctl –D ./pgdata2 start –l pglog2

./pgdata2/postgresql.conf: port = 6432

/usr/local/pgsql/bin/psql –p 6432 => select * from t where id=123;
/usr/local/pgsql/bin/pg_ctl –D ./pgdata2 stop
```



快速安装使用： CentOS 8 从PostgreSQL官方yum源安装PostgreSQL-12

```
yum -y install https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
yum -qy module disable postgresql
yum install postgresql12-server postgresql12
```

```
[root@localhost ~]# cat /etc/passwd | grep postgres
postgres:x:26:26:PostgreSQL server:/var/lib/pgsql:/bin/bash

[root@localhost ~]# which psql
/usr/bin/psql
[root@localhost ~]# ls -l /usr/bin/psql
lrwxrwxrwx. 1 root root 28 Nov 10 23:44 /usr/bin/psql -> /etc/alternatives/pgsql-psql
[root@localhost ~]# ls -l /etc/alternatives/pgsql-psql
lrwxrwxrwx. 1 root root 22 Nov 10 23:44 /etc/alternatives/pgsql-psql -> /usr/pgsql-12/bin/psql
[root@localhost ~]#
```

快速安装使用： CentOS-8 从PostgreSQL官方yum源安装 PostgreSQL-12

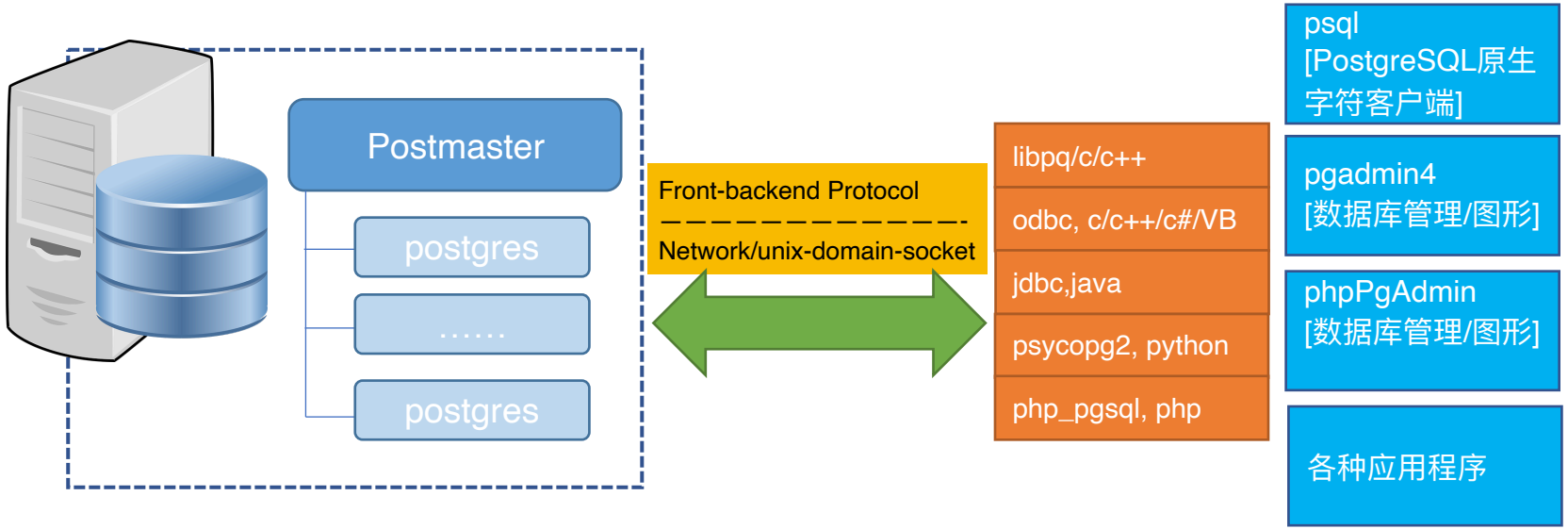
```
[root@localhost ~]# /usr/pgsql-12/bin/postgresql-12-setup initdb

[root@localhost ~]# systemctl enable --now postgresql-12
Created symlink /etc/systemd/system/multi-user.target.wants/postgresql-12.service → /usr/lib/systemd/system/postgresql-12.service


.
[root@localhost ~]# ps -eflgrep postgres
postgres  37668      1  0 23:49 ?        00:00:00 /usr/pgsql-12/bin/postmaster -D /var/lib/pgsql/12/data/
postgres  37670  37668  0 23:49 ?        00:00:00 postgres: logger
postgres  37672  37668  0 23:49 ?        00:00:00 postgres: checkpointer
postgres  37673  37668  0 23:49 ?        00:00:00 postgres: background writer
postgres  37674  37668  0 23:49 ?        00:00:00 postgres: walwriter
postgres  37675  37668  0 23:49 ?        00:00:00 postgres: autovacuum launcher
postgres  37676  37668  0 23:49 ?        00:00:00 postgres: stats collector
postgres  37677  37668  0 23:49 ?        00:00:00 postgres: logical replication launcher


[root@localhost ~]# netstat -na l grep 5432
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN
tcp6       0      0 ::1:5432                :::*                    LISTEN
unix  2      [ ACC ]     STREAM     LISTENING     196419   /tmp/.s.PGSQL.5432
unix  2      [ ACC ]     STREAM     LISTENING     196417   /var/run/postgresql/.s.PGSQL.5432
```

快速安装使用： CentOS-8 从PostgreSQL官方yum源安装 PostgreSQL-12

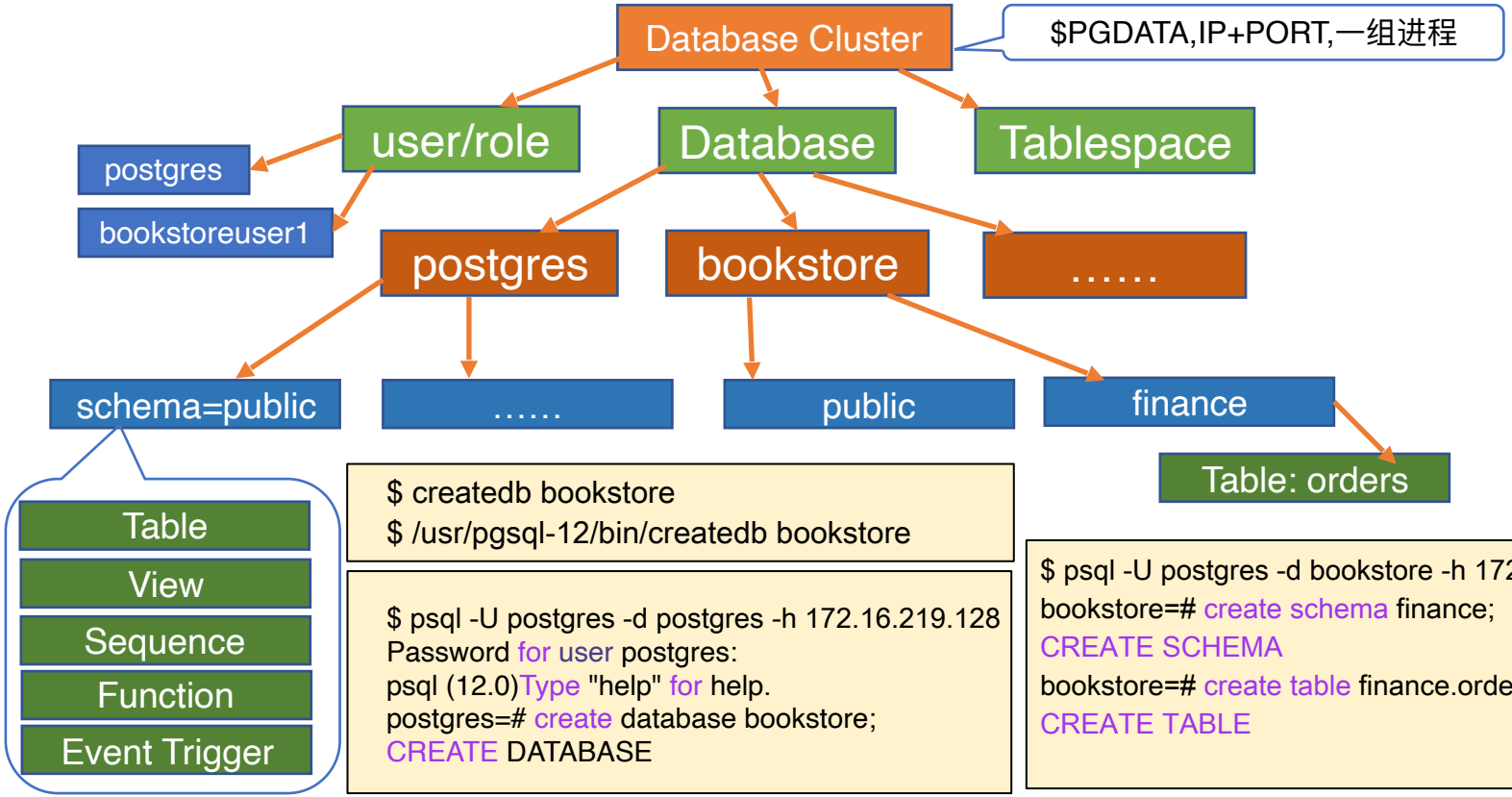
```
[root@localhost ~]# su – postgres
[postgres@localhost ~]$ pwd
/var/lib/pgsql
[postgres@localhost ~]$ echo $PGDATA
/var/lib/pgsql/12/data
[postgres@localhost ~]$
```

```
[postgres@localhost ~]$ psql
psql (12.0)
Type "help" for help.
postgres=# create table test ( id int, name varchar(50));
CREATE TABLE
postgres=# insert into test values( 1,'Tom'),(2,'Jack’);
INSERT 0 2
postgres=# select * from test;
id | name
----+------
1 | Tom
2 | Jack
(2 rows)
postgres=#
```

快速安装使用：应用程序如何与PostgreSQL协作



快速安装使用：数据对象的层次结构




```
$ createdb bookstore
$ /usr/pgsql-12/bin/createdb bookstore
```

```
$ psql -U postgres -d postgres -h 172.16.219.128
Password for user postgres:
psql (12.0)Type "help" for help.
postgres=# create database bookstore;
CREATE DATABASE
```

```
$ psql -U postgres -d bookstore -h 172.16.219.128
bookstore=# create schema finance;
CREATE SCHEMA
bookstore=# create table finance.orders(id int, amount int);
CREATE TABLE
```

快速安装使用：简单的访问控制策略

默认的数据库超级用户：postgres
为PostgreSQL数据库创建的操作系统账号：postgres
默认情况下：只能使用操作系统的账号”postgres”访问数据库，在数据库中的身份是：数据库超级用户”posrgres”。
可以更改配置，使得数据库账号“postgres”和操作系统账号不绑定。

数据库管理：本地 or 远程

- 仅本地管理：使用已有的postgres账号
- 允许远端管理：
    - 创建专用的数据库超级用户，用于远程访问；
    - 或者更改配置，使“postgres”账号可以远程访问和管理数据库。

数据访问：

- 使用数据库管理员(默认postgres)创建其它账号，用于访问数据
- 根据业务场景，可以创建多个不同的数据访问账号。
- 可以有只读账号和读写账号。
- 可以把账号限制在某个/些库，某个/些schema，某个/些表，某个/些列。

快速安装使用：如何让数据库账号postgres可以远程访问数据库

设置防火墙，允许PostgreSQL服务端口

```
firewall-cmd --add-service=postgresql –permanent
firewall-cmd --reload
```

为数据库账号“postgres”设置一个密码

```
[root@localhost ~]# su - postgres
[postgres@localhost ~]$ psql
psql (12.0)
Type "help" for help.
postgres=# alter user postgres with password 'StrongPassword‘;
ALTER ROLE
```

让PostgreSQL侦听外部IP地址 vim /var/lib/pgsql/12/data/postgresql.conf

```
#listen_addresses = 'localhost'
```

```
listen_addresses = ‘*’
```

修改基于外部主机的访问控制vim /var/lib/pgsql/12/data/pg_hba.conf

```
host    all     all             172.16.219.0/24         md5
```

重新启动PostgreSQL服务

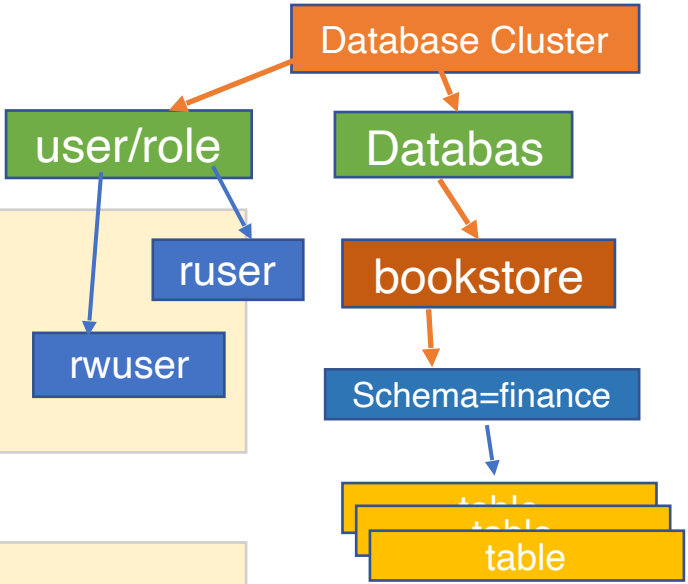
```
systemctl restart postgresql-12
```

在远端或本地使用网络地址访问数据库

```
psql -U postgres -d postgres -h 172.16.219.128
```

快速安装使用：数据库账号规划举例

- 为一个应用创建了数据库bookstore。
- 数据库bookstore包含一个schema,名字是finance。
- 创建一个数据库账号: ‘rwuser’,读、写bookstore里面的数据。
- 创建一个数据库账号: ‘ruser’,读bookstore里面的数据，不能更改数据。



psql -U postgres -d bookstore -h 172.16.219.128

创建只读账号

```
CREATE USER ruser WITH PASSWORD 'secret_passwd';
GRANT CONNECT ON DATABASE bookstore TO ruser;
GRANT USAGE ON SCHEMA finance TO ruser;
GRANT SELECT ON ALL TABLES IN SCHEMA finance TO ruser;
```

创建读写账号

```
CREATE USER rwuser WITH PASSWORD 'secret_passwd';
GRANT CONNECT ON DATABASE bookstore TO rwuser;
GRANT USAGE ON SCHEMA finance TO rwuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA finance TO rwuser;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA finance TO rwuser;
```

数据类型：PostgreSQL内置数据类型

https://www.postgresql.org/docs/12/datatype.html

自定义数据类型

- https://www.postgresql.org/docs/12/xfunc-c.html#XFUNC-C-BASETYPE
- Typical example: PostGIS: http://postgis.net

数据类型：使用数据类型定义和处理数据

```
CREATE TABLE COMPANY(
   ID INT PRIMARY KEY NOT NULL,
   NAME VARCHAR(50) NOT NULL,
   AGE INT NOT NULL,
   ADDRESS VARCHAR(100),
   SALARY REAL,
   JOIN_DATE DATE );

INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY,JOIN_DATE) VALUES (1, 'Paul', 32, 'California', 20000.00,'2001-07-13');
INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,JOIN_DATE) VALUES (2, 'Allen', 25, 'Texas', '2007-12-13');
INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY,JOIN_DATE) VALUES (3, 'Teddy', 23, 'Norway', 20000.00, DEFAULT );
INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY,JOIN_DATE) VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00, '2007-12-13' ), (5, 'David', 27, 'Texas',
85000.00, '2007-12-13');

SELECT * FROM COMPANY;

ID NAME AGE ADDRESS SALARY JOIN_DATE
 ---- ---------- ----- ---------- ------- --------
1 Paul 32 California 20000.0 2001-07-13
2 Allen 25 Texas 2007-12-13
3 Teddy 23 Norway 20000.0
4 Mark 25 Rich-Mond 65000.0 2007-12-13
5 David 27 Texas 85000.0 2007-12-13

UPDATE COMPANY SET AGE=40 WHERE NAME='Paul';
DELETE FROM COMPANY WHERE ID=2;
DROP TABLE COMPANY;
```

数据类型：JSON举例

```
CREATE TABLE users (
    id serial NOT NULL PRIMARY KEY,
    info json NOT NULL
);

INSERT INTO users (info) VALUES (
'{
    "name": "Jane Doe",
    "email": "janedoe@example.com",
    "personalDetails": {"age":33, "gender":"F"}
 }');

INSERT INTO users (info) VALUES (
'{
     "name": "Jane Doe",
    "email": "janedoe@example.com",
    "personalDetails": {"age":33, "gender":"F"}
  }');
```

```
SELECT info ->> 'email' FROM users;
id |              ?column?
---+-----------------------------------
1 | johndoe@example.com
2 | janedoe@example.com

SELECT info -> 'personalDetails' -> 'gender' FROM users;


?column?
----------
"M"
"F"
(2 rows)
```

数据类型：JSONB举例

```
CREATE TABLE cards (
  id integer NOT NULL,
  board_id integer NOT NULL,
  data jsonb
);

INSERT INTO cards VALUES (1, 1, '{"name": "Paint house", "tags": ["Improvements", "Office"], "finished": true}');
INSERT INTO cards VALUES (2, 1, '{"name": "Wash dishes", "tags": ["Clean", "Kitchen"], "finished": false}');
INSERT INTO cards VALUES (3, 1, '{"name": "Cook lunch", "tags": ["Cook", "Kitchen", "Tacos"], "ingredients": ["Tortillas",
"Guacamole"], "finished": false}');
INSERT INTO cards VALUES (4, 1, '{"name": "Vacuum", "tags": ["Clean", "Bedroom", "Office"], "finished": false}');
INSERT INTO cards VALUES (5, 1, '{"name": "Hang paintings", "tags": ["Improvements", "Office"], "finished": false}');

SELECT data->>'name' AS name FROM cards;
 name
----------------
Paint house
Wash dishes
Cook lunch
Vacuum
Hang paintings
(5 rows)

CREATE INDEX idxgintags ON cards USING gin ((data->'tags'));
```

数据类型：XML举例

```
CREATE TABLE table1(
 id integer PRIMARY KEY,
 created timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
 xdata xml );

INSERT INTO table1 (id, xdata) VALUES( 1,
'<dept xmlns:smpl="http://example.com" smpl:did="DPT011-IT">
   <name>IT</name>
  <persons>
     <person smpl:pid="111">
        <name>John Smith</name>
        <age>24</age>
     </person>
    <person smpl:pid="112">
        <name>Michael Black</name>
        <age>28</age>
    </person>
  </persons>
  </dept>'
 );
SELECT * FROM table1 WHERE (xpath('//person/name/text()', xdata))[1]::text = 'John Smith';
SELECT * FROM table1 WHERE (xpath('//person/@smpl:pid', xdata, ARRAY[ARRAY['smpl', = '111', 'http://example.com']]))::text;
CREATE INDEX i_table1_xdata ON table1 USING btree ( xpath('//person/@name', xdata) );
```

特异功能表：表继承

```
CREATE TABLE cities (
    name text,
    population float,
   altitude int );
CREATE TABLE capitals
( state char(2) ) INHERITS (cities);
INSERT INTO cities values ('Las Vegas', 1.53, 2174);
INSERT INTO cities values ('Mariposa', 3.30, 1953);
INSERT INTO capitals values ('Madison', 4.34, 845,'WI');

SELECT name, altitude FROM cities WHERE altitude > 500;
name | altitude
-----+----
Las Vegas | 2174
Mariposa | 1953
Madison | 845
(3 rows)

SELECT name, altitude FROM capitals WHERE altitude > 500;
name | altitude
-----+----
Madison | 845 (1 row)

SELECT name,altitude FROM ONLY cities WHERE altitude > 500;
name | altitude
-----+----
Las Vegas | 2174 Mariposa | 1953
(2 rows)
```

Table cities

name	type
name	text
population	float
altitude	int

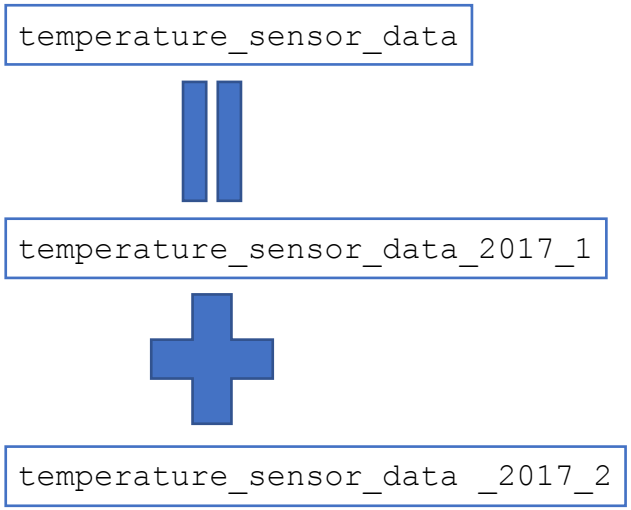
Table capitals

name	type
name	text
population	float
altitude	int
state	char(2)

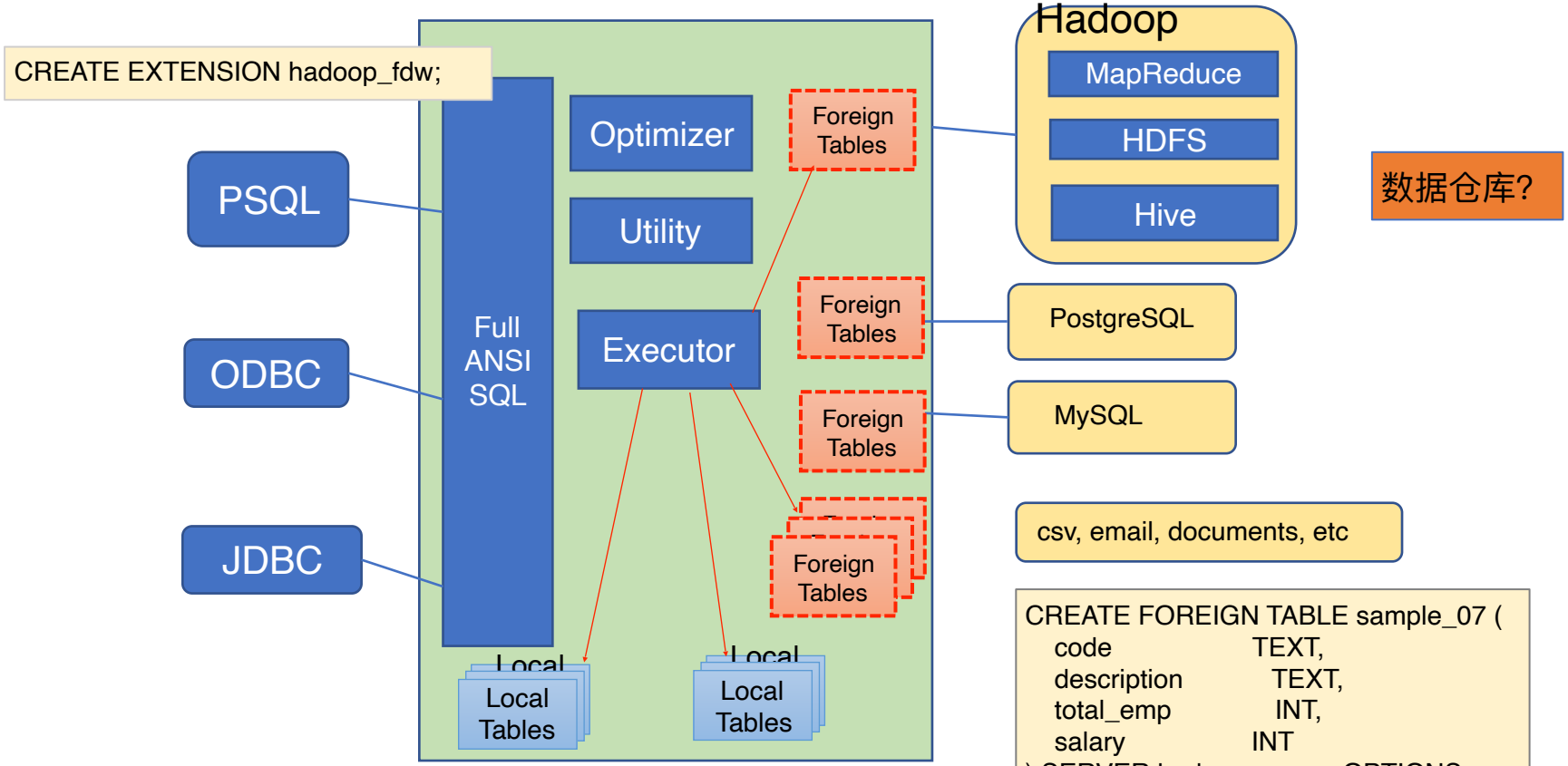
对象数据库

特异功能表：表分区

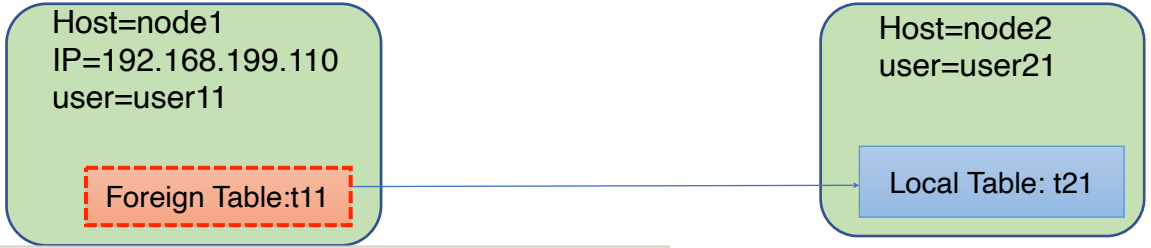
```
CREATE TABLE temperature_sensor_data (
    sensor_id       integer   NOT NULL,
    timestamp       timestamp NOT NULL,
    temperature     decimal(5,2) NOT NULL
)  PARTITION BY RANGE (timestamp);

CREATE TABLE temperature_sensor_data_2017_1
    PARTITION OF temperature_sensor_data
    FOR VALUES FROM ('2017-01-01') TO ('2017-02-01');

CREATE TABLE temperature_sensor_data_2017_2
    PARTITION OF temperature_sensor_data
    FOR VALUES FROM ('2017-02-01') TO ('2017-03-01');

--insert data to the parent
insert into  temperature_sensor_data values (…);

--select from one partition
select sensor_id, timestamp, temperature from
temperature_sensor_data where timestamp =
'2017-02-02 12:10:00'::timestamp;

--select data from all partitions
select sensor_id, timestamp, temperature from
temperature_sensor_data;
```



特异功能表：FDW(Foreign Data Wrapper)



特异功能表：postgres-fdw




```
psql -h node1 -p 5432 -d postgres
create extension postgres_fdw;
create role user11 with login password '#@1qw';
grant all privileges on database postgres to user11;
grant usage on foreign data wrapper postgres_fdw to user11;
```

```
psql –U user11 -h node1 -p 5432 -d postgres

create server fnode2    foreign data wrapper postgres_fdw   options
          (host 'node2', port '6432', dbname 'postgres');

create user mapping for user11   server fnode2  options (
                  user 'user21', password 'pwd#@1'    );

create foreign table t11 (  id int,  name varchar(100)   )
       server fnode2   options ( schema_name  'public',  table_name
't21' );
```

```
insert into t11 values(2,'Jack');
select * from t11;
```

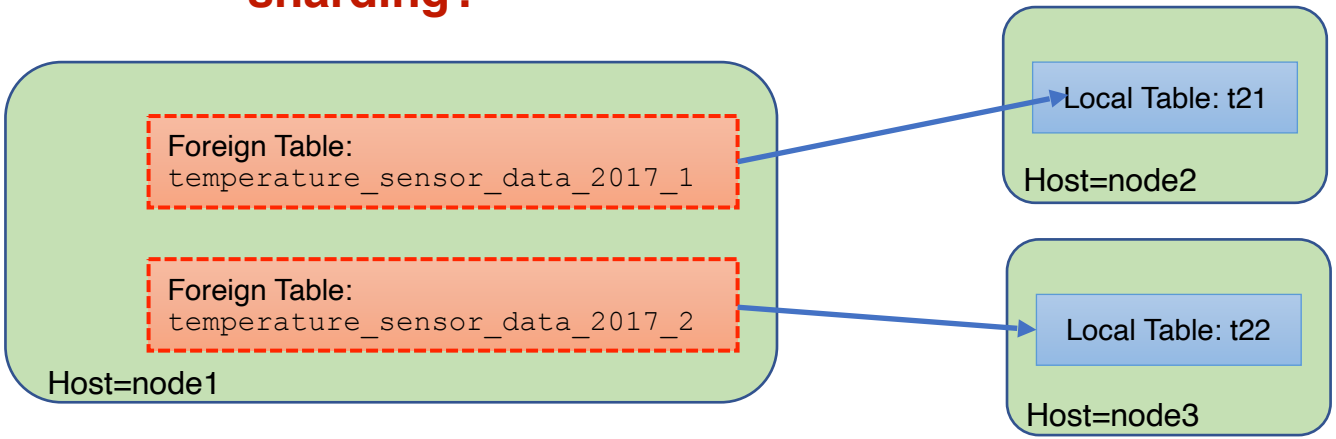
```
psql -h node2 -p 6432 -d postgres
create role user21 with login password 'pwd#@1';
grant all privileges on database postgres to user21;
```

pg_hba.conf

```
host    postgres        user21        192.168.199.110/32    md5
```

```
psql –U user21 -h node2 -p 6432 -d postgres
create table t21 ( id int, name varchar(100));
insert into t21 values(1,'Tom');
```

特异功能表：(partitioned table) + postgres-fdw = sharding?



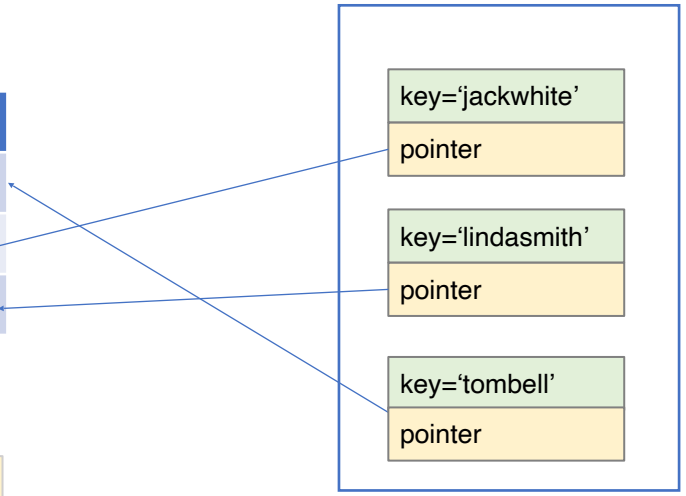
temperature_sensor_data = {
  temperature_sensor_data_2017_1,
  temperature_sensor_data_2017_2
}

```
--select from one partition
select sensor_id, timestamp, temperature from temperature_sensor_data
where timestamp = '2017-02-02 12:10:00'::timestamp;

--select data from all partitions
select sensor_id, timestamp, temperature from temperature_sensor_data;
```

- 当前的分区是串行扫描的，有待改进。
- 我们也实现了并行扫描。

使用索引，加快搜索速度

id	username	firstname	lastname	email	phone
1	tombell	Tom	Bell	tom@abc.com	123-45567
2	jackwhite	Jack	White	jackw@ted.com	125-23522
3	lindasmith	Linda	Smith	lindas@tell.com	532-539021




```
create table users (
    id bigint,
    username varchar(50),
    firstname varchar(20),
    lastname varchar(20),
    email varchar(50),
    phone varchar(20)
);

insert into users values
(1, 'tombell' , 'Tom' , 'Bell' , 'tom@abc.com' , '123-45567'),
(2, 'jackwhite' , 'Jack' , 'White' , 'jackw@ted.com' , '125-23522'),
(3, 'lindasmith' , 'Linda' , 'Smith' , 'lindas@tell.com' , '532-539021');
```

```
create index user_id_idx on users( id );
select * from users where id=12342155566;

create index user_username_idx on users( username );
select * from users where username='tombell';

drop index user_id_idx;
drop index user_username_idx;
```

B-tree, Hash, GiST, SP-GiST, GIN and BRIN

视图/View，物化视图/Materialized View

```
create table employee_info
(
   employee_id  integer not null,
   first_name  varchar(100),
   last_name   varchar(100),
   email       varchar(50),
   mobile_num  varchar(20),
   hire_date   timestamp without time zone not null,
   salary      int,
   bank_number varchar(20),
   department_id  smallint not null,
   supervisor_id  integer,
   password    varchar(50)
);

create view employee_contact  as
     select employee_id,
       first_name ||' '|| last_name as name,
       mobile_num,
       email from employee_info;

select * from employee_contact;
```

```
create materialized view mat_sales_dept_contact
  as select employee_id,
            first_name ||' '|| last_name as name,
            mobile_num,
            email from employee_info
               where department_id=6;

select * from mat_sales_dept_contact;
```

```
drop view employee_contact;
drop view mat_sales_dept_contact;
```

物化视图+FDW

事务/Transaction

```
CREATE TABLE accounts (
id INT GENERATED BY DEFAULT AS IDENTITY,
name VARCHAR(100) NOT NULL,
balance DEC(15,2) NOT NULL,
PRIMARY KEY(id) );

insert into accounts (name, balance)
values ( 'Tom', 1000),('Jack', 2000);

postgres=# select * from accounts;
id | name | balance ----+------+---------
 1 | Tom | 1000.00
 2 | Jack | 2000.00
 (2 rows)

begin;
update accounts set balance=balance-100 where name='Tom';
update accounts set balance=balance+100 where name='Jack';
commit;

postgres=# select * from accounts;
 id | name | balance
 ----+------+---------
 1 | Tom | 900.00
 2 | Jack | 2100.00
 (2 rows)
```

```
begin;
update accounts set balance=balance-100 where name='Tom';
update accounts set balance=balance+100 where name='Jack ;
rollback;

postgres=# select * from accounts;
id | name | balance
----+------+---------
1 | Tom | 900.00
2 | Jack | 2100.00
(2 rows)

begin;
update accounts set balance=balance-100 where name='Tom';
savepoint p1;
update accounts set balance=balance+100 where name='Jack ;
rollback to savepoint p1; --rollback
commit;

begin;
update accounts set balance=balance-100 where name='Tom';
savepoint p1;
update accounts set balance=balance+100 where name='Jack ;
release savepoint p1; --commit
commit;
```

过程语言/Procedural Language

PL/pgSQL - SQL Procedural Language
PL/Tcl - Tcl Procedural Language
PL/Perl - Perl Procedural Language
PL/Python - Python Procedural Language
PLV8: Javascript
Plgo: Go language

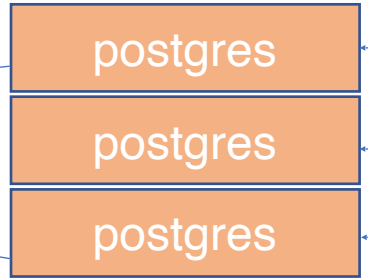
过程语言/Procedural Language: PL/pgsql

```
create table test ( id int );
insert into test select * from generate_series(1,10);


CREATE OR REPLACE FUNCTION count_test ( )
RETURNS integer AS $$
    declare total integer;
    BEGIN SELECT count(*) into total FROM test;
        RETURN total;
    END;
$$ LANGUAGE plpgsql;


postgres=# select count_test();
count_test
------------
10
(1 row)

DROP FUNCTION count_test;
```

过程语言/Procedural Language: PL/pgsql

```
CREATE TABLE emp ( empname text NOT NULL, salary integer );
CREATE TABLE emp_audit( operation char(1) NOT NULL,
    stamp timestamp NOT NULL, userid text NOT NULL,
    empname text NOT NULL, salary integer );

CREATE OR REPLACE FUNCTION process_emp_audit() RETURNS TRIGGER AS $emp_audit$
BEGIN
   -- Create a row in emp_audit to reflect the operation performed on emp,
   -- making use of the special variable TG_OP to work out the operation.
    IF (TG_OP = 'DELETE') THEN
       INSERT INTO emp_audit SELECT 'D', now(), user, OLD.*;
    ELSIF (TG_OP = 'UPDATE') THEN
       INSERT INTO emp_audit SELECT 'U', now(), user, NEW.*;
    ELSIF (TG_OP = 'INSERT') THEN
       INSERT INTO emp_audit SELECT 'I', now(), user, NEW.*;
    END IF;
    RETURN NULL; -- result is ignored since this is an AFTER trigger
END;
$emp_audit$ LANGUAGE plpgsql;

CREATE TRIGGER emp_audit
    AFTER INSERT OR UPDATE OR DELETE ON emp
FOR EACH ROW EXECUTE FUNCTION process_emp_audit();

DROP TRIGGER emp_audit on emp;
DROP FUNCTION process_emp_audit;
```

PostgreSQL内幕：内存和文件



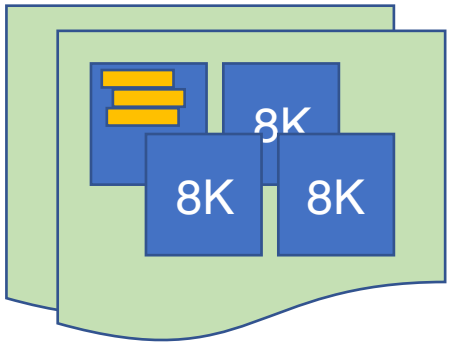
select * from t;

Update t set …

insert into t values(1,'Tom');

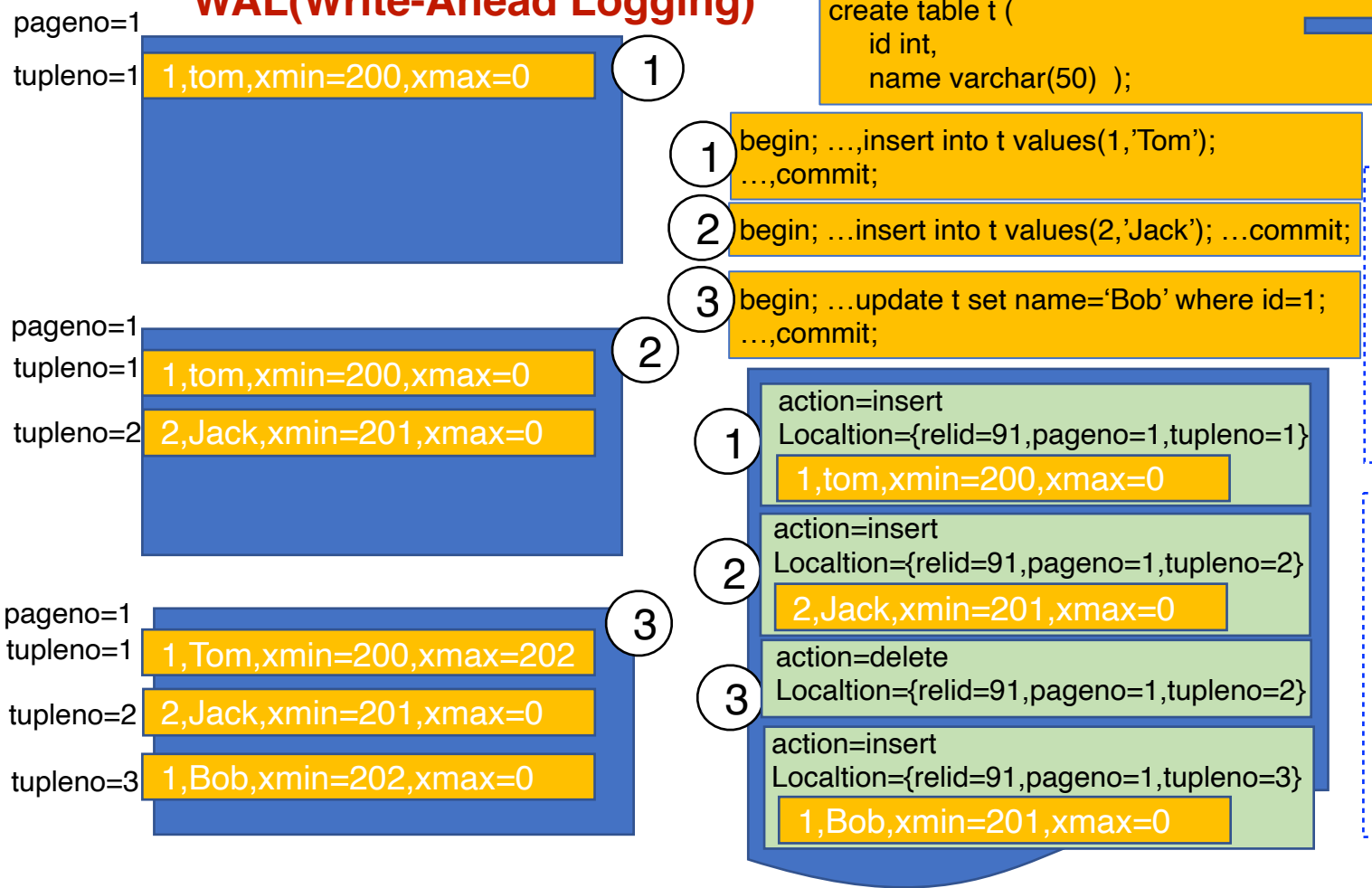
共享内存

文件



- 数据持久化保存在磁盘上
- 需要时候，数据以页（page)为单位被加载到内存：
  - 搜索
  - 更改
  - 插入
- 内存中更改的数据页在适当时候被写入文件
- 最关键的数据文件类型：
  - 表文件
  - 索引文件
  - WAL（Write Ahead Log）文件
- 每个一个表或索引可能对应多个文件，单个文件大小1G，每个数据页(page)默认大小是8K，每个数据文件有唯一编号，每个数据页有唯一编号。
- 一个8K数据页(page)包含多行数据；每一行数据的所有列连续存放，形成一个连续的数据块，叫做tuple。每一个tuple在数据页内有统一编号。
- (page编号，页内tuple序号)=>唯一标识表内的一个行，这就是索引的指针。

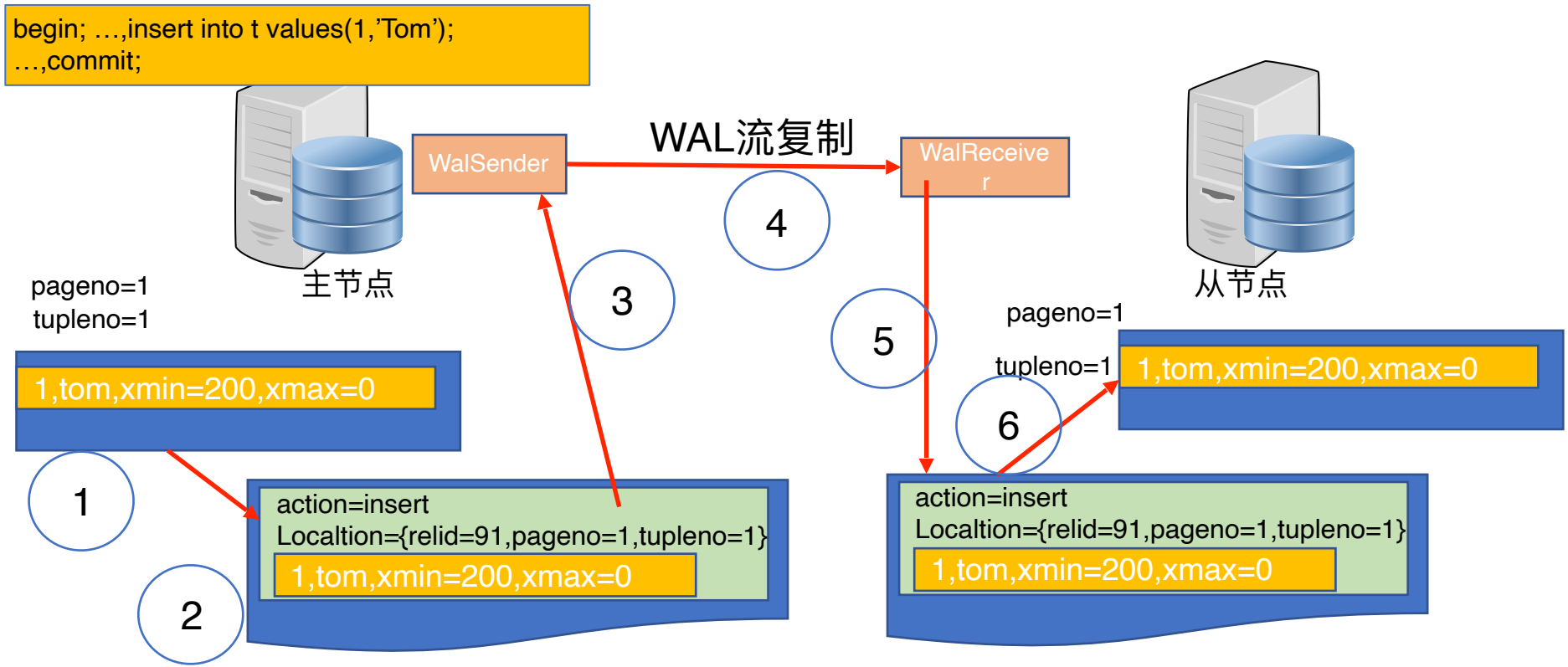
PostgreSQL内幕：MVCC((Multi-Version Concurrency Control) / WAL(Write-Ahead Logging)



- WAL记录了tuple变更的物理拷贝
- WAL写是顺序追加，比数据页落盘更高效。
- WAL先于数据页落盘，保证数据不丢失。
- WAL比数据页更重要。

- MVCC意味着一行记录有多个版本同时存在。
- 更新、删除数据时，老数据依然存在。
- 读和写相互不阻碍。
- 第三步updat操作完成之前，其它的都操作得到的是'Tom'，而不是'Bob'

PostgreSQL内幕：流复制(Stream Replication)+持续数据备份



编程接口

- 1 C
  - 1.1 libpq
  - 1.2 ECPG
- 2 C++
  - 2.1 libpqxx
  - 2.2 QtSql
  - 2.3 Pgfe
  - 2.4 OZO
- 3 Elixir
  - 3.1 Postgrex
- 4 Go
  - 4.1 pq
  - 4.2 pgx
  - 4.3 go-pg
- 5 Haskell
  - 5.1 postgresql-simple
- 6 Java
  - 6.1 PostgreSQL JDBC Driver
- 7 Javascript
  - 7.1 node-postgres
  - 7.2 pg-promise
- 8 Common Lisp
  - 8.1 Postmodern
  - 8.2 CLSQL
- 9 .Net
  - 9.1 npgsql
- 10 ODBC
- 11 Perl
  - 11.1 DBD::Pg
  - 11.2 DBD::PgPP
- 12 PHP
  - 12.1 Pomm
  - 12.2 ext-pq
- 13 Python
  - 13.1 psycopg2
- 14 R
  - 14.1 RPostgreSQL
- 15 Ruby
  - 15.1 ruby-pg
- 16 Rust
  - 16.1 rust-postgres
- 17 tcl
  - 17.1 pgtcl
  - 17.2 pgtclng

PostgreSQL扩展(extension)

什么是extension?

经过多年的发展和完善，PostgreSQL社区开发设计了一套编写扩展的规范。按照该规范，开发者能够设计一些PostgreSQL不存在的功能，这些功能会通过PostgreSQL架构中预留的钩子函数，或者通过SQL中的函数被调用。这些新的功能，以扩展包的方式提供。扩展包包括：编译的成动态代码库和相关的支撑文件，可以被PostgreSQL动态加载，这样，就可以在不修改和编译PostgreSQL代码的情况下，动态加载或卸载扩展。

该网站登记了大多数PostgreSQL扩展包：https://pgxn.org

著名的扩展

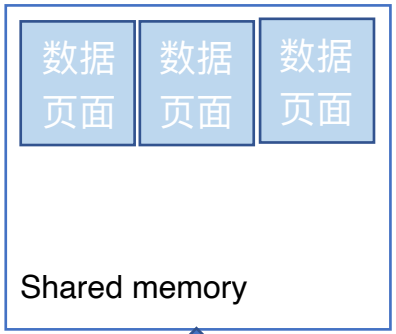
- PostGIS，http://postgis.net：地理位置数据处理。
- Citus：最成功的分布式数据库解决方案。
- Bidirectional Replication (BDR)， http://bdr-project.org/docs/stable/
- Key-value store，Hstore，https://www.postgresql.org/docs/12/hstore.html
- Postgres-fdw，https://www.postgresql.org/docs/12/postgres-fdw.html
- 列式存储：Cstore_fdw is an open source columnar store extension for PostgreSQL. https://github.com/citusdata/cstore_fdw
- mongo_fdw, MongoDB Foreign Data Wrapper for PostgreSQLhttps://github.com/EnterpriseDB/mongo_fdw
- 时间序列：Pipelinedb, High-performance time-series aggregation for PostgreSQL.https://www.pipelinedb.com
- 时间序列： TimescaleDB, is implemented as an extension on PostgreSQL, which means that it runs within an overall PostgreSQL instance. https://www.timescale.com
- 利用GPU加速查询：pg_strom，https://github.com/heterodb/pg-strom

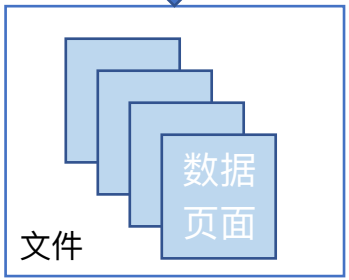
PostgreSQL扩展: PostGIS

```
CREATE TABLE geometries (
  name varchar,
  geom geometry);

INSERT INTO geometries VALUES
  ('Point', 'POINT(0 0)'),
  ('Linestring', 'LINESTRING(0 0, 1 1, 2 1, 2 2)'),
  ('Polygon', 'POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),
  ('PolygonWithHole', 'POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),(1 1, 1 2, 2 2, 2 1, 1 1))'),
  ('Collection', 'GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0, 1 0, 1 1, 0 1, 0 0)))');
```

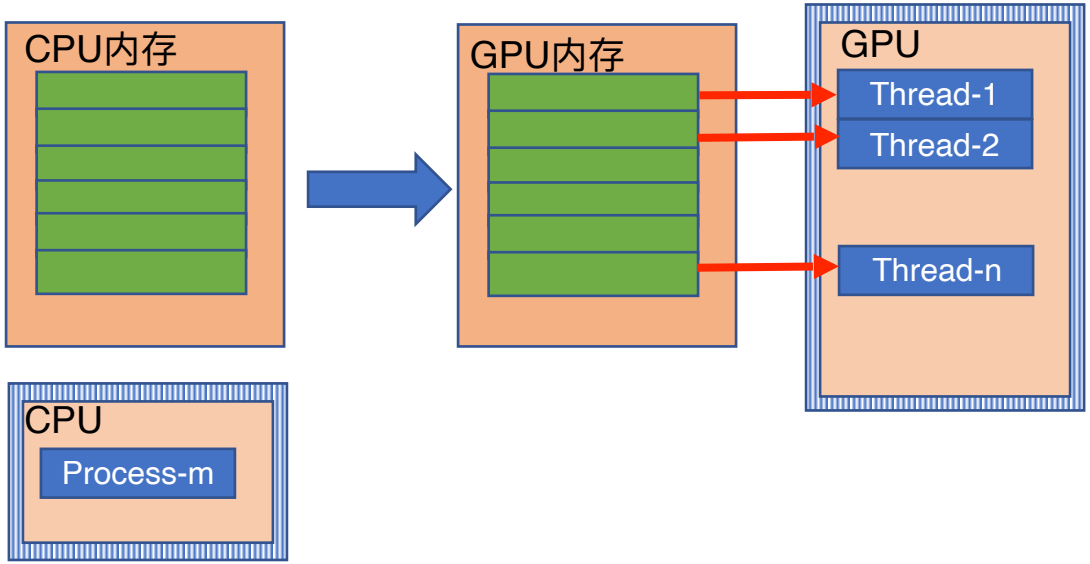
```
CREATE TABLE school (
   ID int4,
  name char(10),
  geom geometry(POINT)
);

CREATE TABLE hospital(
  ID int4, name char(10),
  geom geometry(POINT)
);

insert into school values (1,'65 middle','POINT(1 2)'); insert into hospital values (1,'shanda','POINT(1 2)');

SELECT s.ID, s.name, s.geom, h.name FROM
school s LEFT JOIN hospital h ON ST_DWithin(s.geom, h.geom, 3000) ORDER BY s.ID, ST_Distance(s.geom, h.geom);
```

PostgreSQL扩展: pg_strom



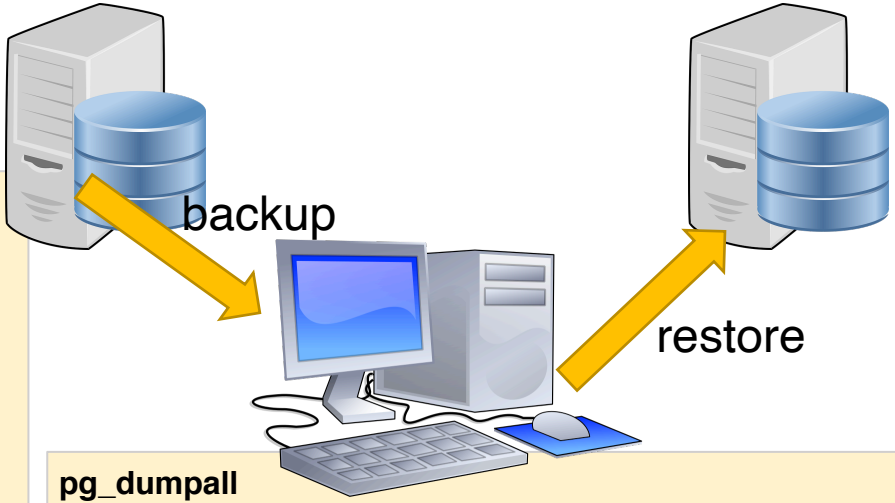


select * from t where a < 10;



- Postgres进程：CPU进程。
- Postgres进程生成GPU代码，编译上传可执行代码到GPU。
- Postgres进程从数据文件中读取批量tuple到内存。
- 批量的tuple总内存传入GPU内存。
- GPU内多个线程同时工作，每个线程处理各自的tuple。

数据备份和恢复：pg_dump, pg_dumpall，逻辑备份



pg_dump
把单个数据库导出到文件。导出的数据不包含用户和表空间。

使用举例1：
导出数据：pg_dump dbname > dumpfile
导入数据：psql dbname < dumpfile

使用举例2：
导出数据：pg_dump -Fc dbname > filename
导入数据：pg_restore -d dbname filename

使用举例3：
导出数据：pg_dump dbname | gzip > filename.gz
导入数据：gunzip -c filename.gz | psql dbname

使用举例4：
导出数据：pg_dump dbname | split -b 1m – filename
导入数据：cat filename* | psql dbname

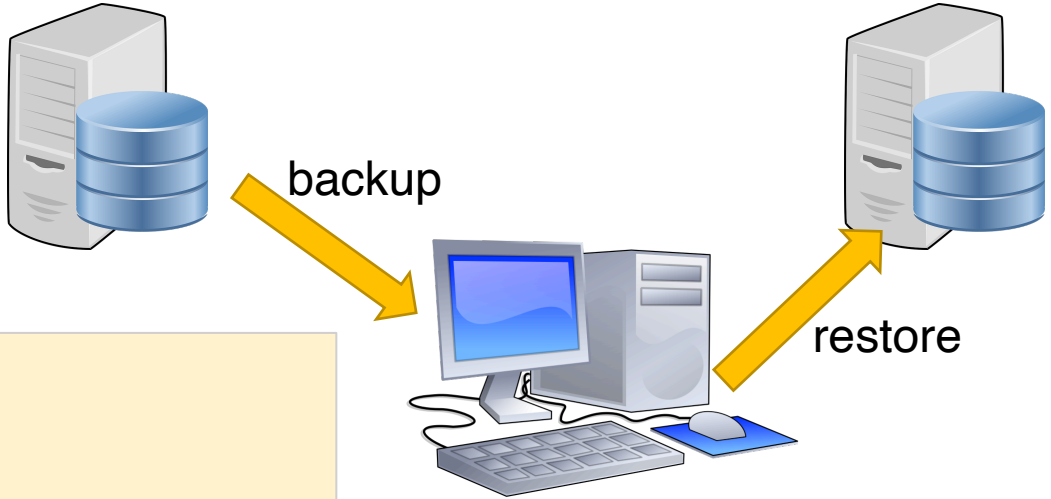
pg_dumpall
可以把整个cluster导出。

使用举例1：
导出数据：/usr/pgsql-11/bin/pg_dumpall > /tmp/dumpall.sql
导入数据：/usr/pgsql-11/bin/psql -p 5433 -f /tmp/dumpall.sql

使用举例2：
pg_dumpall -p 5432 | psql -p 5433

应用举例3：
pg_dumpall -p 5432 -h source_server | psql -p 5433 -h target_server

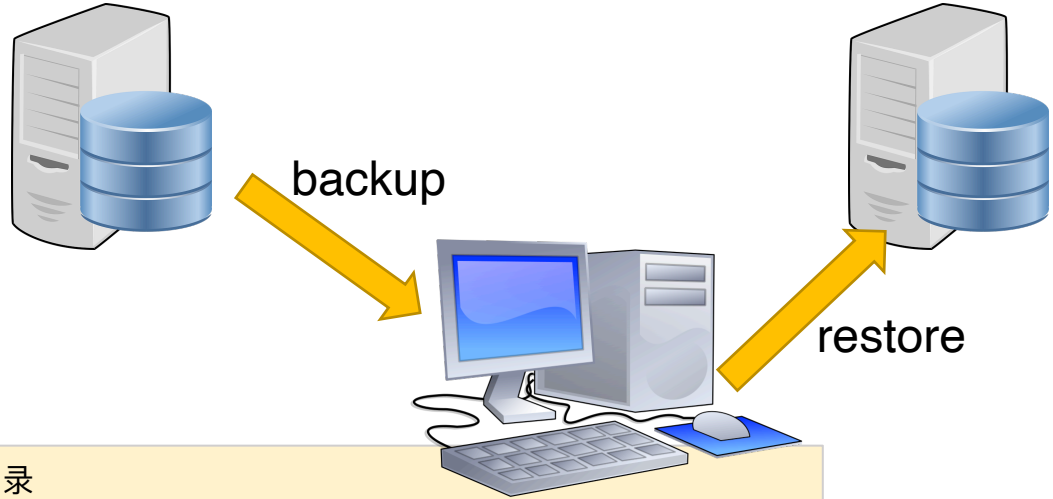
数据备份和恢复：基于文件系统的备份，物理备份



备份数据：host1
systemctl stop postgresql-12
tar -cvf backup.tar /var/pgsql/12/data

恢复数据：host2
tar –xvf backup.tar
systemctl start postgresql-12

数据备份和恢复：pg_basebackup,在线物理备份



把节点mydbserver上的数据库备份到本地目录
/usr/local/pgsql/datapg_basebackup -h mydbserver -D /usr/local/pgsql/data

把本地数据库的备份成tar文件，保存在本地目录
pg_basebackup -D backup -Ft -z -P

备份得到的数据，释放到目标服务器的指定目录，在目标服务器上启动PostgreSQL。

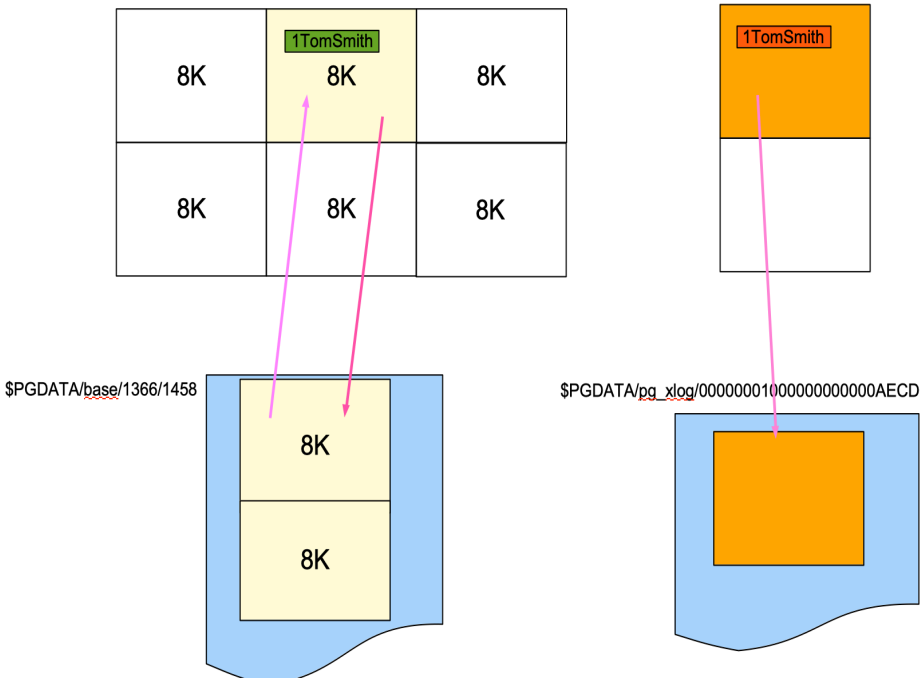
数据备份和恢复：增量备份：[pg_basebackup,在线物理备份]+主从流复制

1. 使用pg_basebackup，备份主节点上数据。
2. 使用备份得到的数据，创建从节点。
3. 配置主从节点之间的流复制。
4. 根据需要，重新启动主从节点。
5. 主节点上的数据更改会持续不断地传递到从节点。
6. 主节点提供数据读和写；从节点可以提供数据读(hot-standby)。

数据备份和恢复：增量备份:WAL

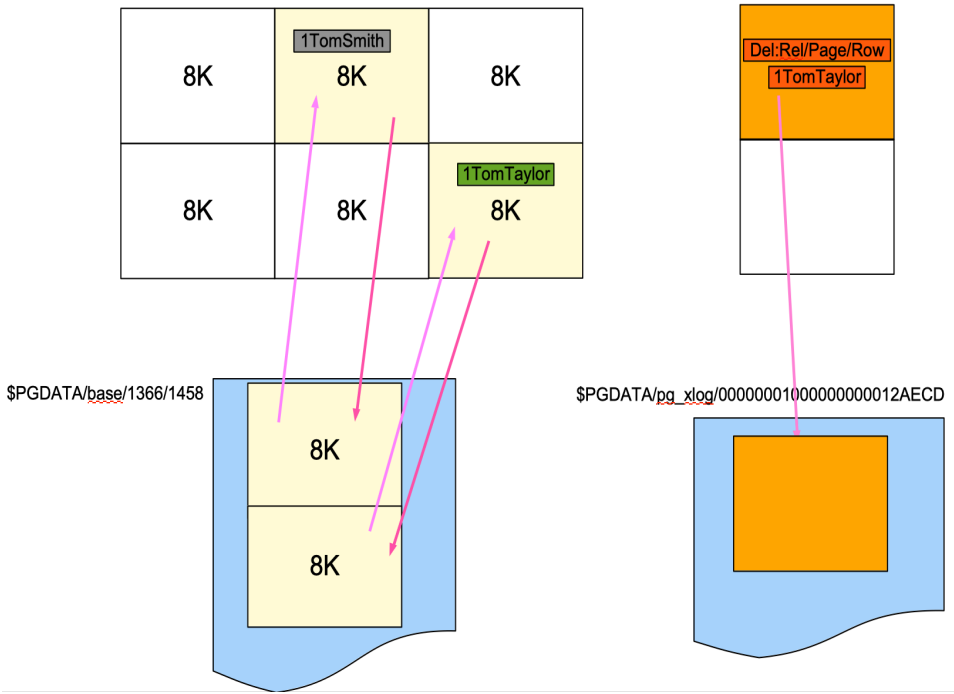
insert:Tuple,Page and XLog

```
create table guestbook( id int, first_name varchar(100), last_name varchar(100));
insert into t values (1,'Tom','Smith');
```



update:Tuple,Page and XLog

```
Update t set last_name='Taylor' where id=1;
```



数据备份和恢复：自动化

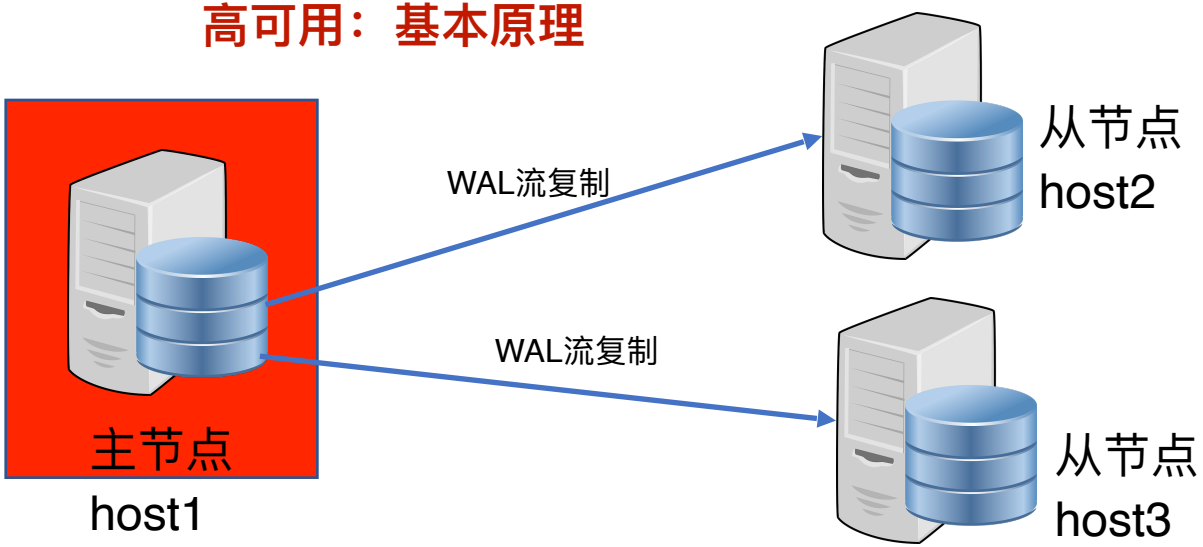
Barman https://www.pgbarman.org/index.html

pgBackRest
https://pgbackrest.org/

编写你自己的自动化备份工具

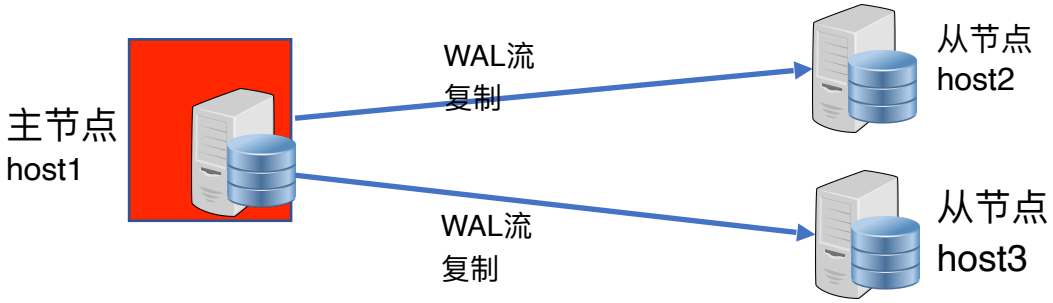
云上的PostgreSQL服务

高可用：基本原理



- 配置一个数据库主节点，数据可读、可写。
- 根据需要配置1个或多个从节点。
- 当主节点故障时，把一个从节点提升为主。任何一个节点都可能成为主节点。
- 业务量很大时，可以让从节点提供数据只读服务，以降低主节点的压力。

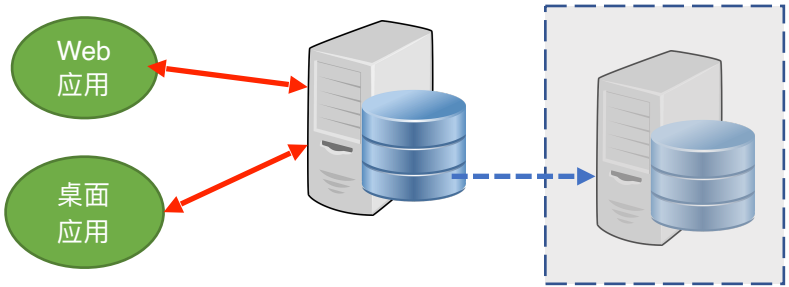
高可用：辅助工具，自动监控+自动切换。

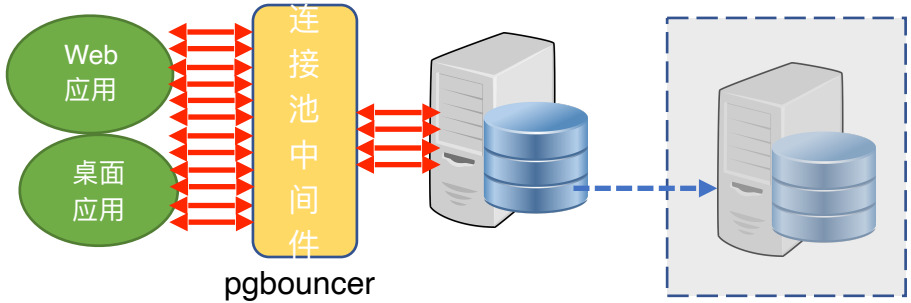


系统监控+自动切换+故障恢复

- Etcd+patroni
https://github.com/zalando/patroni
- pg_auto_failover
https://github.com/citusdata/pg_auto_failover
- Pacemaker + Corosync + PAF(PostgreSQL Automatic Failover)
https://clusterlabs.github.io/PAF/
- repmgr
https://repmgr.org/docs/4.4/index.html

应用场景：单节点





pgbouncer
http://www.pgbouncer.org

特点

- 单节点能够应付并发数量、数据量和查询量。
- 扩展方式：增加内存，磁盘，CPU。
- 大多数的应用都是这样的。

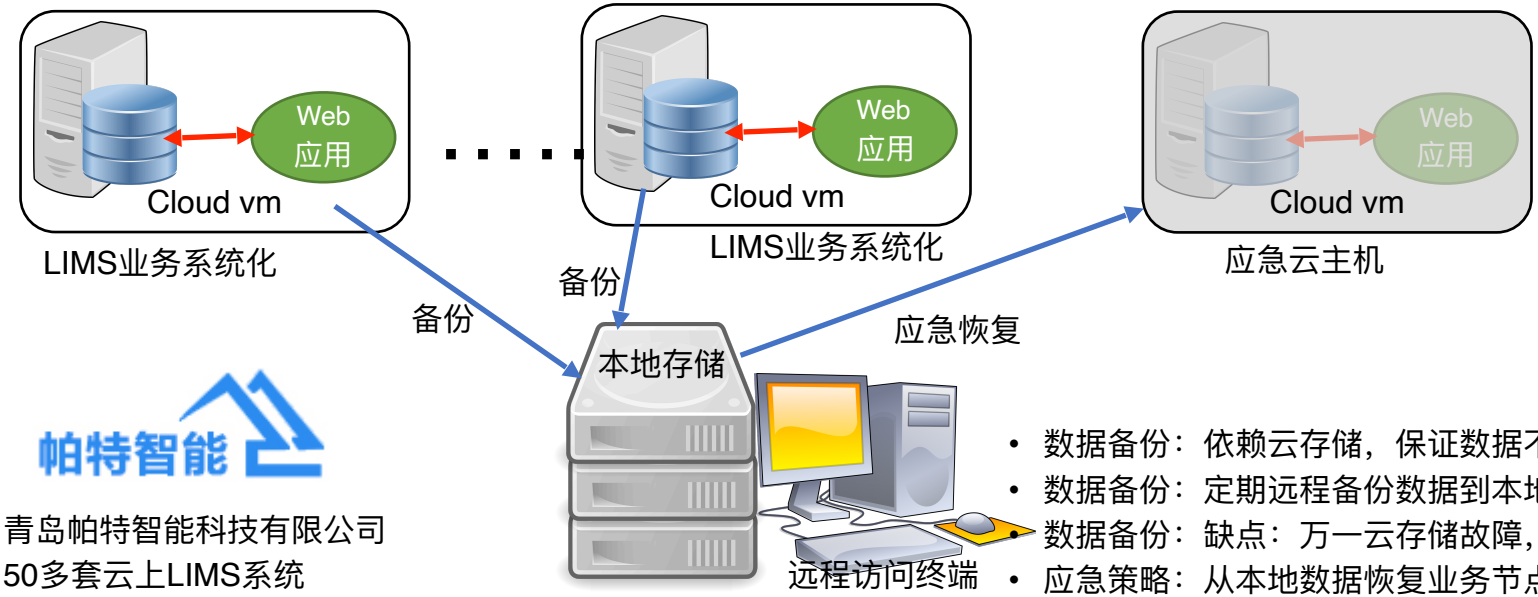
备份方案：

- 根据应用的重要程度，可以使用pg_dump做定期备份，或者使用持续增量备份。
- 自动化的工具会更加有效。

高可用方案：根据应用的重要程度，

- 可以主节点宕机时重建数据库节点，
- 使用Warm-standby
- 使用hot-standby

应用场景：单节点应用案例分析



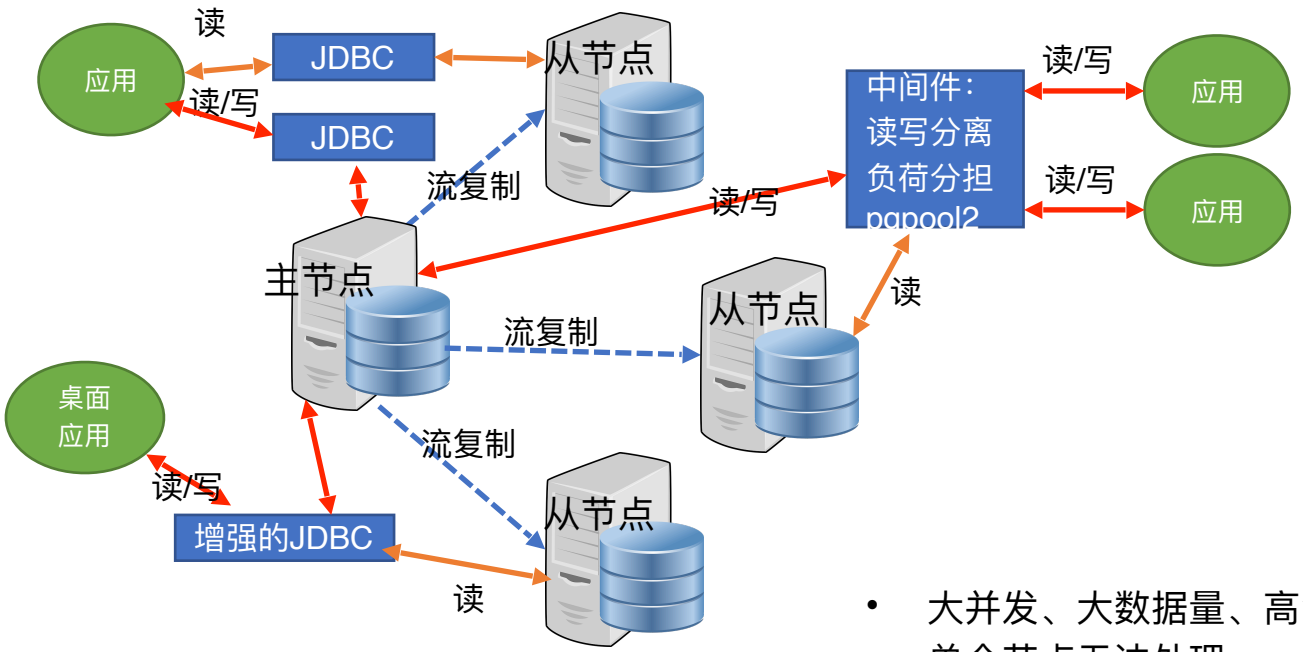
帕特智能

青岛帕特智能科技有限公司
50多套云上LIMS系统
LIMS：食品质量检测企业的业务系统

- 每个云主机上部署： JavaWeb应用+PostgreSQL
- 每个应用的数据量：GB
- 每个应用的并发量：100
- 数据库并发量：20
- 数据库TPS：<100

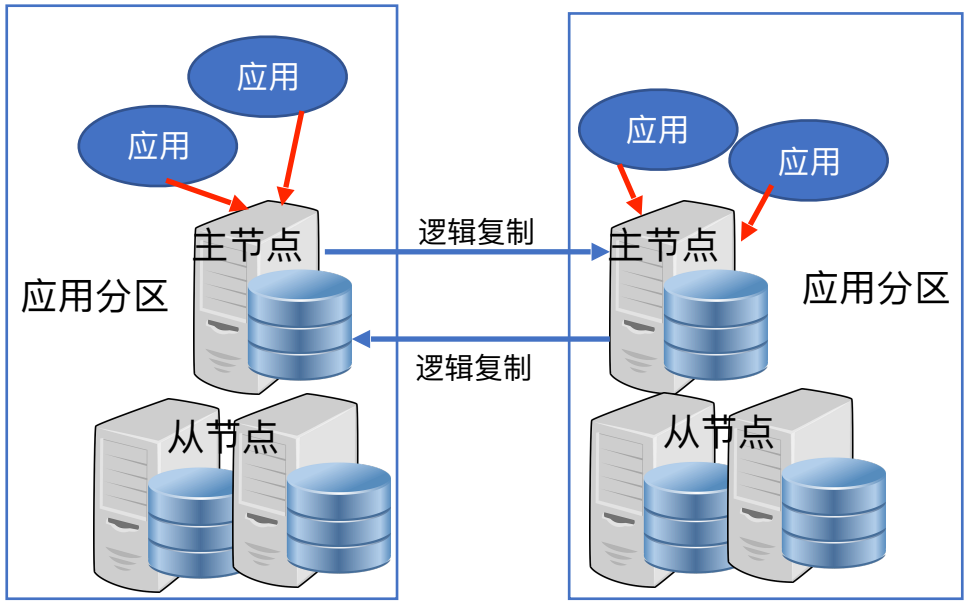
- 数据备份：依赖云存储，保证数据不丢失
- 数据备份：定期远程备份数据到本地。
- 数据备份：缺点：万一云存储故障，本地的数据是老版本
- 应急策略：从本地数据恢复业务节点。
- 编写大量的自动备份恢复工具。
- 建议使用同步流复制。

应用场景：一主多从+读写分离+自动监控+自动切换



- 大并发、大数据量、高负荷
- 单个节点无法处理
- 必须部署多个数据库节点，系统才能运行。
- 系统监控，节点提升，故障恢复

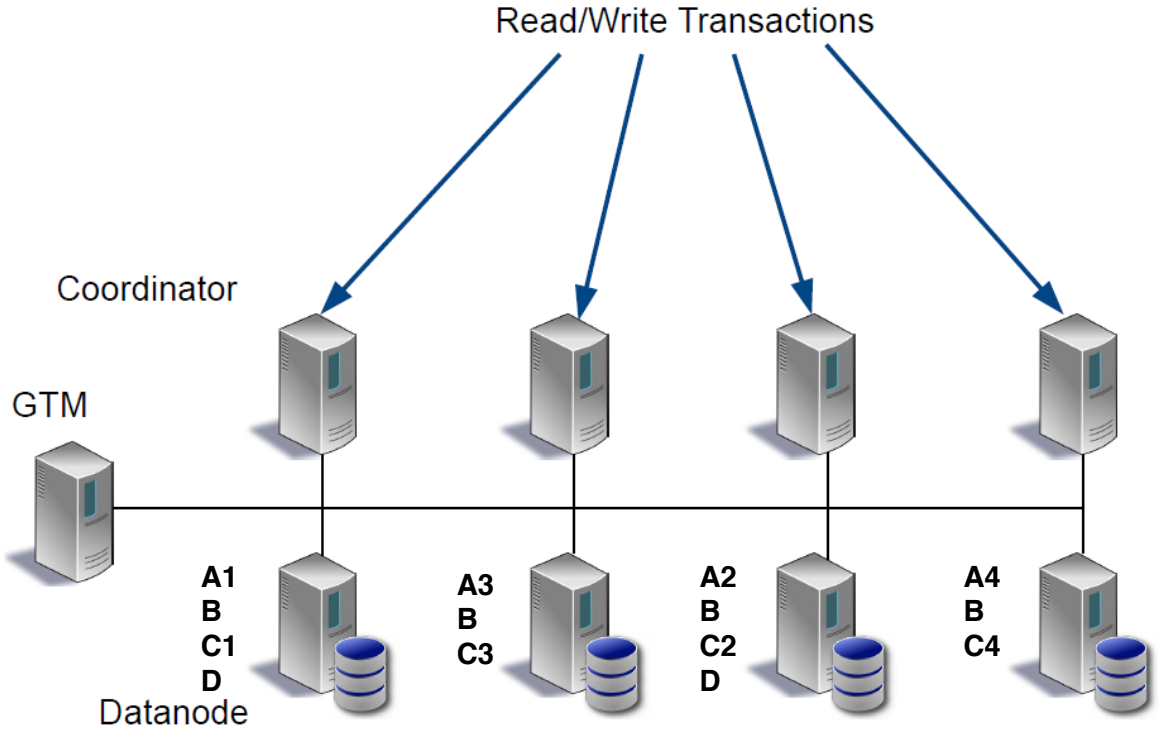
应用场景：多主节点(multimaster with logical replication)



- 业务和数据能够根据业务键分区
- 分区内：
  - 应用对本分区数据读写。
  - 有时需要读取其它分区的数据，允许一定的延时。针对某些表。
  - 一主多从。
- 分区之间：
  - 主节点之间使用逻辑复制，把本分区的数据（仅仅某些表）更新传送给其它分区。
  - 潜在风险：数据冲突。
  - 分区之间是独立的数据库系统。
- 可以手工配置分区之间的逻辑复制，或者编写脚本程序辅助维护。
- 已有工具BDR：http://bdr-project.org/docs/stable/

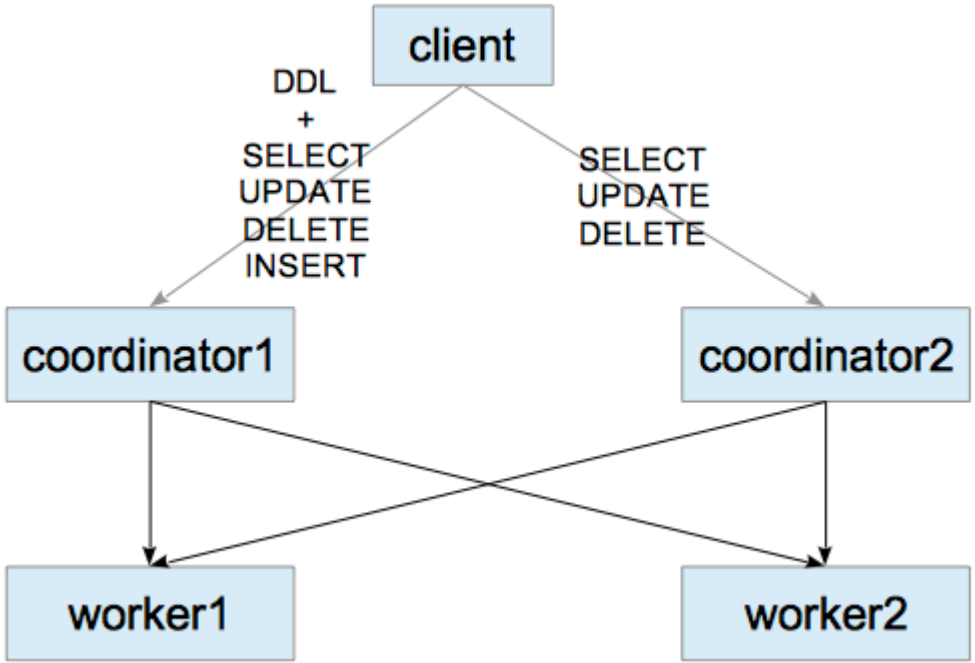
应用场景:分布式数据库集群，Postgre-XL, 基于PostgreSQL改造的

https://www.2ndquadrant.com/en/resources/postgres-xl/



- Coordinators(协调器)
  - 应用程序的访问点
  - 分析SQL语句，生成查询计划
- Data Nodes(数据节点)
  - 实际的数据存储
  - 本地执行查询计划
- Global Transaction Manager(GTM)
  - 统一管理全局范围的事务
  - 使用GTM-proxy进一步提高性能。
- 对应用系统透明。
- 能够灵活水平扩展。
- 把负荷分摊到多个处理节点，实现并行处理。
- 支持OLTP+OLAP业务。
- 实现了全局的事务处理。
- 需改进之处:
  - 执行计划有待于进一步优化
  - 性能优化
  - 节点备份、高可用
  - 管理维护
  - 数据冗余分片
  - 跨节点死锁检查
  - 数据rebalance
- 被大公司改进并使用

应用场景：分布式数据库，Citus，目前最成功的分布式PostgreSQL

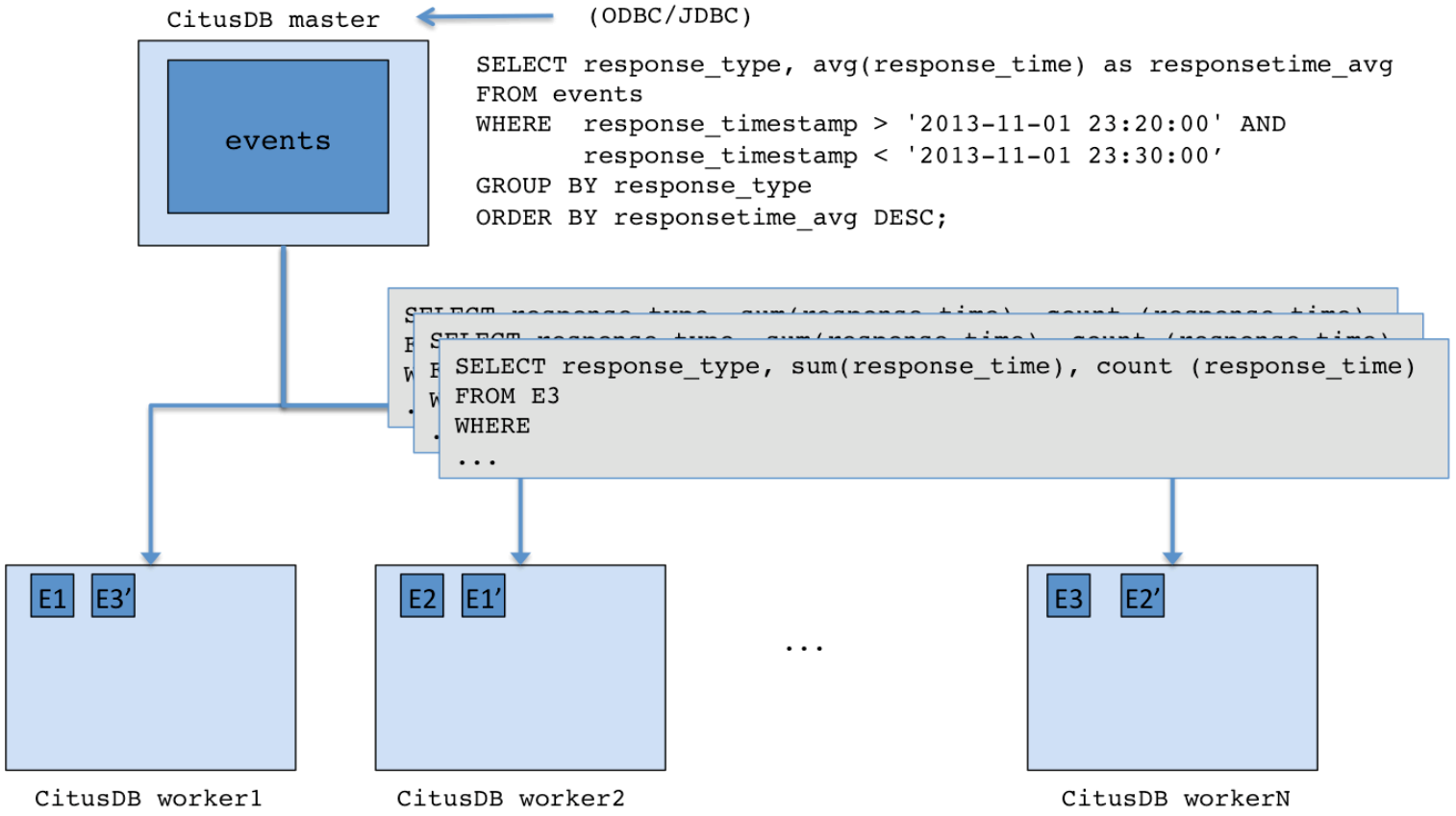


Citus解决的问题和Postgres-XL要解决的问题一样，只是走了不同的路，另外，Postgres-XL团队仅仅开了个头，然后就几乎停滞不前。
Citus提供云服务，及时解决了应用系统的痛点：数据库需要扩展。

目前Citus被广泛使用，且被微软收购。

- 实现了数据在多个节点之间分片。
- 实现了分布式执行计划。
- 实现了全局的事物管理。
- 实现了分布式的死锁检测。

应用场景： Citus，目前最成功的分布式PostgreSQL集群



PostgreSQL12近期期待的新功能

- 内置的连接池
- UNDO log
- Block level parallel vacuum
- Transactions involving multiple postgres foreign servers
- Unix-domain socket support on Windows


thank
you!