阿里云数据库 PostgreSQL、PPAS、HDB for PG

云生态、产品指南、企业全栈应用案例、开发管理实践

阿里云
digoal

内容简介

- 1、PostgreSQL, GPDB 技术发展与云生态。
- 2、PostgreSQL与GPDB在OLTP|OLAP、空间数据管理、图式搜索、全文检索、文本搜索、特征搜索、时序应用、流式数据处理、用户画像分析等场景的应用案例。
  - 案例横跨 物联网、电商、生物科技、游戏、传统企业、CRM、ERP、ZF、GA、物流、音视频、BI、社交、金融、证券、手机、天文等行业。
- 3、开发与管理实践。

目录

- 产品介绍
- 生态介绍
- 应用案例
- 开发、管理实践
- 数据库原理
- 参考文档

PostgreSQL 起源

数据库界
唯一图灵奖

PostgreSQL 趋势

阿里云PostgreSQL产品体系

支持Oracle\PG两套协议；

(定位Oracle兼容)

多机并行计算版本；

(定位实时数仓，PB级)

完全兼容PostgreSQL。
计算存储分离版；
存储接口自定义（支持下推计算）；
计算节点支持水平扩展；
(定位高端用户、HTAP混合应用)

云数据库 PPAS 版

HybridDB for PostgreSQL

PolarDB for PostgreSQL

产品间支持平滑过渡

开源增强版；
RDS for PG
定位10TB级TP+AP

PostgreSQL

产品体系 - RDS for PG

PostgreSQL

全面支持：
增删改查、
操作符、
索引、
编程语言
(plv8)

轻松处理空间信息

通过PostGIS插件，可以轻松支持2D、3D地址信息模型，更支持地球不规则球体的偏移量，实现达到国际OpenGIS标准的精确定位。

强大NoSQL兼容

基于SQL支持JSON、XML、Key-Value等非结构化数据类型，实现另类的Not Only SQL(NOSQL)解决方案

支持全文搜索

通过全文搜索，应用将不再需要额外搭建搜索引擎，只通过SQL操作即可实现全文检索(Full Text Search)及模糊查询。

支持OSS云存储扩展

基于PostgreSQL的FDW功能，阿里云深度整合优化了对OSS云存储的外部表管理功能，可以支 3TB now 上存储空间无限扩展。

支持数据仓库

通过PostgreSQL除了在高可用方面能够满足OLTP在线应用的要求外，需要进行实时分析的数据，还可以扩展支持OLAP数据仓库的业务。

精准GIS模型

不规则椭球体。
半径变化。
.....
SPHEROID["Krasovsky_1940",**6378245.000000**,298.299997264589]]

测绘、天文、定位、自动驾驶、。。。要求精准模型。

空间优势

- 2D、3D、4D类型
  - 点、线、面、组合空间对象
- geometry，高效、精准（指定SRID区间）
- geography，全球精准（跨国、航空、航海）
- raster(栅格数据)，广泛应用于 测绘、航天、气象、3D城市。。。
- Topology，路径规划（导航）
- 索引
  - 空间索引、BRIN索引
  - https://github.com/digoal/blog/blob/master/201804/20180417_01.md **(为什么geometry+GIST 比geohash+BTREE更适合空间搜索 - 多出的不仅仅是20倍性能提升)**
- 生态
  - ArcGIS唯一支持的开源数据库
  - 专业空间业务唯一选择（天文、测绘、气象、3D城市、国土资源 ...）
- 多维查询支持
  - 时空、其他类型 多维组合搜索 - 索引

传统架构痛点

跨数据库痛点：同步延迟、一致性、原子性、可靠性等问题。
成本痛点：软件成本、硬件成本、研发成本。

RDS PG HTAP (**All In One全栈数据库**)

一个产品搞定。

PostgreSQL 优势

- PG支持热插拔扩展

Watch 89 | Unstar 712 | Fork 88

pg-strom GPU加速

Unwatch 111 | Unstar 1,752 | Fork 151

pipelinedb 流计算

Watch 156 | Unstar 2,871 | Fork 207

citus sharding

Watch 77 | Unstar 529 | Fork 216

PostGIS 时空

Watch 234 | Unstar 5,443 | Fork 259

timescaleDB 时序

Unwatch 385 | Unstar 2,645 | Fork 816

GPDB MPP

RDS PG HTAP (**All In One全栈数据库**)

- Why
  - PG具有先天架构优势
- VS
  - 性能、功能
- 扩展能力
  - 水平、垂直均可扩展

单实例数据管理能力

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB **128TB(32K block_size)**
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250 - 1600 depending on column types
Maximum Indexes per Table	Unlimited

单表管理能力

- 单表十亿性能**几乎无衰减** (仅增加索引深度开销)

数据量	写入吞吐	查询tps	更新tps
1000万	58万行/s	67万	23.1万
1亿	53.2万行/s	63.4万	24.5万
10亿	162.6万行/s	60.6万	23.4万

https://github.com/digoal/blog/blob/master/201711/20171107_46.md

垂直扩展能力

一份数据
解决TP\AP
PG 9.6+
支持资源
隔离
并行计算
并行计算
OLTP
100万+ TPS
OLAP
OLTP
OLAP

跨机房容灾能力

发布：
create publication ...;
add tbl to publication ...;

https://github.com/digoal/blog/blob/master/201702/20170227_01.md

订阅：
create subscrib...;
支持断点续传、支持双向订阅
支持一对多订阅、支持多对一订阅
支持多对多订阅

水平扩展能力 + 存储扩展能力

- 传统企业数据库上云，突破单库容量限制
  - 多库组集群、相互可访问、可写、可同步。
  - 功能点：FDW(外部表、远程表)、DBLINK、匿名、逻辑订阅。

FDW外部表：
访问异库表，犹如访问本地表，没有限制。
支持读写、JOIN等，支持PUSHDOWN算子。
偶尔访问的异地数据，采用dblink、或fdw，简化逻辑。

逻辑订阅（单元化）：
经常访问的异地数据，使用订阅功能订阅到本地。

OSS 无限对象存储：
用于冷存储，支持多实例共享访问。

库与库不再孤立，形成整体。

sharding 1 (FDW)

继承、自动数据路由、支持跨库访问、
PUSH DOWN(JOIN、sort、where、UDF。。。)

sharding2 (Citus)

shard table,
reference table

https://github.com/digoal/blog/blob/master/201808/20180824_02.md

1\choose column
2\choose distribute policy
3\set shard count
4\choose colocate group
5\shard hash value range
6\balance to worker node

	events shards		page shards	
	shard	hash range	shard	hash range
NODE 1	1	-2147483648 ≤ x ≤ -1073741825	5	-2147483648 ≤ x ≤ -1073741825
	2	-1073741824 ≤ x ≤ -1	6	-1073741824 ≤ x ≤ -1
NODE 2	3	0 ≤ x ≤ 1073741823	7	0 ≤ x ≤ 1073741823
	4	1073741824 ≤ x ≤ 2147483647	8	1073741824 ≤ x ≤ 2147483647

x = hash(distribution_column)

sharding2 (citus)

sharding2 (citus)

citus
MX特性
指定worker携带metadata

sharding2 (citus)

产品体系 - RDS for PG

PostgreSQL

无限存储空间扩展

打破RDS TB级存储空间极限

现基于OSS云存储的外部表实现存储空间扩展，需要经常访问的“热数据”直接放在RDS的SSD存储空间，不常用的数据转存到OSS云存储，并可基于标准gzip压缩算法进行OSS中的数据压缩，进一步节省历史数据存储成本。结合同样基于PostgreSQL内核的云数据库HybridDB，可扩展支持PB级别的OLAP在线分析业务，性能及存储空间同时线性扩展。

方案优势

历史日志存储

结合OSS云存储实现无限空间扩展

BI海量数据分析

结合HybridDB通过相同的SQL语法实现高性能分析...

推荐搭配使用

OSS　HybridDB

产品体系 - RDS for PG

PostgreSQL

GIS+JSON助力IoT高速发展

让开发人员及DBA基于SQL提高生产力

JSON数据类型及GIS地理信息数据类型都是IoT业务处理的必备手段，传统方案中开发人员及DBA需要在NoSQL数据库和专用的GIS数据分析软件中进行多次的硬编码开发。而在PostgreSQL中所有这些操作都可以在SQL中完成，无需来回进行数据导入，提高开发效率。

方案优势

传感器JSON数据

SQL中直接实现关联查询降低开发成本

GIS地理信息数据

精准定位，支持2D、3D、路径、范围分析简单易用

推荐搭配使用

HybridDB

产品体系 - RDS for PG

企业PostgreSQL标配

核心业务数据库

基于阿里云多可用区架构，支持同城容灾，实现企业级数据库安全稳定。PostgreSQL相比其它开源数据库，进行更优事务处理，保障多表关联JOIN时的系统性能。

能够解决

大表支持

单表上亿数据性能平稳，基于Hash JOIN支持高性能多表查询

OLAP方案

提供TB级以下OLAP支持，海量数据使用HybridDB横向扩展

无限空间扩展

基于OSS外部表将冷数据保存到云存储并可与HybridDB交互

推荐搭配使用

HybridDB　OSS　ECS

产品体系 - RDS for PG

PostgreSQL

OLTP

- JSON、数组、K-V等扩展类型
- 函数、存储过程
- 全文检索、模糊查询
- 数组、文本、图像 相似搜索
- 2d,3d,4d 空间数据, 兼容 ArcGIS
- 逻辑订阅（单元化）
- 流式主备复制、延迟低至毫秒(不担心大事务)
- AWR报告、扩展插件

OLAP

- 多核并行计算(9.6+)
- TB级 实时分析
- 复杂JOIN
- 任意字段组合 Ad-Hoc 查询
- SQL流式计算
- FDW based sharding
  - pushdown select-clause, where, sort, join, agg, operator, function

分级存储

- 本地存储上限 3TB
- OSS存储无上限
  - 每线程30MB/s

产品体系 - RDS for PPAS

云数据库 PPAS 版

高度兼容

兼容Oracle数据类型、PL/SQL
比迁移其他数据库降低90%工作量

自动化部署

5分钟完成所有部署配置
全自动化处理避免人为失误

同城容灾

HA企业级架构全双冗余保障
同城双中心，同等价格加倍安全

正版合规

含License，避免合规审计风险
协助企业完成数据库正版化

产品体系 - RDS for PPAS

云数据库 PPAS 版

增量数据迁移

通过阿里云“数据传输”服务，用户线下的Oracle数据库可以实现 云数据库PPAS版 的增量数据同步，缩短生产系统迁移中的停机时间。

高性能表分区

提供高性能表分区实现，兼容Oracle语法，在100+表分区的情况下性能相比原生PostgreSQL性能提高50倍以上

全文搜索及GIS支持

在同一个SQL引擎中直接完成包括全文检索及GIS地理信息处理功能，减轻开发人员的研发难度，免除多系统交互带来的复杂性。

OSS云存储扩展

打破RDS系统中3TB空间的物理限制，基于OSS云存储实现外部表处理，用户空间无限扩展。

数据仓库

支持与HybridDB通过OSS进行数据交换，使用同样基于PostgreSQL生态的OLAP分析型数据库实现数据仓库业务。

产品体系 - RDS for PPAS

云数据库 PPAS 版

Oracle兼容

- 语法
- 内置函数
- PL/SQL存储过程
- AWR报告功能

- ADAM评估去O风险
  - 1周输出详细报告

OLTP+OLAP

- 复杂SQL
- 函数、存储过程
- 全文检索、模糊查询
- 空间数据GIS
- JSON
- 多核并行计算(9.6+)
- TB级实时分析
- FDW based sharding
  - pushdown select-clause, where, sort, join, agg, operator, function

增强功能

- SQL防火墙
- 自动索引推荐
- CPU\IO 资源组管理

分级存储

- 本地存储3TB
- OSS存储无上限
  - 每线程30MB/s

去O生态 - ADAM PPAS专版

真正做到心里
有底的去O。

https://www.aliyun.com/product/adam

去O生态 - ADAM PPAS专版

Advanced Database & Application Migration

（简称 ADAM）

定位： 简单地把云下Oracle等数据库应用迁移上云

去O生态 - ADAM PPAS专版

依赖分析

SQL类型分析

自动拆库

垂直拆

水平拆

按SQL类型拆

某GaTrKgl去O上云(2017)

评估配置

schema名称	表	视图	物化视图	存储过程	触发器	函数	自定义
[redacted]	48	0	0	1	0	4	0
[redacted]	494	34	19	68	12	80	1
[redacted]	43	1	0	15	1	28	0
[redacted]AT	111	1	0	0	0	0	0
[redacted]EW	53	0	0	23	1	32	2
[redacted]	9	1	0	4	0	7	0
[redacted]	9	0	0	0	0	6	0
[redacted]	51	0	0	15	1	32	3
[redacted]	14	0	0	4	0	6	0
[redacted]A	207	8	0	25	7	59	0
总计	1,039	45	19	155	22	254	6

目标数据库方案

编号	类型	数据库规格	表数量	备注
1	PPAS	16 Cores 64 G Memory 512 G Disk	962	

部分报告截图 - 对象详情

跨库对象统计

对象类型	Table	SQL	Trigger	Procedure	View	M-View	Transaction
总计	0	0	0	0	0	0	0

信息详情

表组列表

表组编号	Schema名	表名	表行数	表容量
1	X		6,316,590	2,467,899,152
1	X		6,222,630	1,016,070,144
1	X		6,208,278	1,962,016,768
1	X		4,762,086	1,017,118,720
1	J		4,669,665	511,705,088
1	X		4,084,688	867,172,352
1	Z EW		3,641,592	1,363,869,696
1	X		3,293,384	2,193,883,136

部分报告截图 - 兼容性、工作量详情

兼容性分析

编号	数据库类型	对象类型	对象总数	兼容	不兼容	改动后兼容
1	PPAS	MATERIALIZED_VIEW	0	0	0	0
1	PPAS	PROCEDURE	165	162	3	0
1	PPAS	SQL	5,578	5,578	0	0
1	PPAS	TABLE	948	948	0	0
1	PPAS	TRANSACTION	0	0	0	0
1	PPAS	VIEW	38	37	1	0
	总计		6,729	6,725	4	0

工作量评估

DBA工作量(人天)	开发人员工作量(人天)	测试和上线保障工作量(人天)
0	0	22

部分报告截图 - 风险点详情

风险点列表

慢SQL(执行时间超10秒)

SQL ID	dnr4p3cjppzsz
执行时间(秒)	127
	[redacted]

部分报告截图 - 不兼容修改建议

需要修改SQL详细信息
不兼容sql详细信息
兼容sql详细信息

编号	1
SQL定义	[redacted]
兼容性	oracle object:最低修改难度系数，不需要修改

编号	2
SQL定义	[redacted]
兼容性	oracle object:最低修改难度系数，不需要修改

部分报告截图

PROCEDURE详细评估报告

（企业版）

评估时间: 2017-10-31 20:50:38

总计评估PROCEDURE数量为165，匹配到特性的DDL有165，其中，不兼容的DDL数量有3，修改后兼容的DDL数量为0，兼容的DDL数量为162。

需要修改SQL详细信息

不兼容sql详细信息

编号	1
SQL定义	[redacted]

产品体系 - HybridDB for PostgreSQL

HybridDB for PostgreSQL

传统分库分表架构缺陷

传统分库分表
架构：
限制、缺陷。

1. 扩容不方便（数据重分布）
2. 分布键变更很麻烦
3. 分布键选择（架构设计）谨慎
4. 跨库JOIN性能差
5. 分布式事务性能差
6. SQL限制多、功能缺失多
7. 应用改造成本巨大
8. 全局一致性时间点恢复几乎不可实现

shard之间隔绝、需要跨节点JOIN的话要在中间件层实现、效率低

产品体系 - HybridDB for PostgreSQL

HybridDB for PostgreSQL

产品体系 - HybridDB for PostgreSQL

多阶段聚合-降低网络开销

```
postgres=# explain analyze select count(*),c2 from a group by c2;
                                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 Gather Motion 48:1  (slice2; segments: 48)  (cost=15733096.77..15733098.04 rows=101 width=12)
   Rows out:  101 rows at destination with 3823 ms to end, start offset by 216 ms.
   ->  HashAggregate  (cost=15733096.77..15733098.04 rows=3 width=12)
         Group By: a.c2
         Rows out:  Avg 2.5 rows x 41 workers.  Max 4 rows (seg9) with 0.001 ms to first row, 1324 ms to end, start offset by 253 ms.
         ->  Redistribute Motion 48:48  (slice1; segments: 48)  (cost=15733093.24..15733095.26 rows=3 width=12)
               Hash Key: a.c2
               Rows out:  Avg 118.2 rows x 41 workers at destination.  Max 192 rows (seg9) with 2462 ms to end, start offset by 253 ms.
               ->  HashAggregate  (cost=15733093.24..15733093.24 rows=3 width=12)
                     Group By: a.c2
                     Rows out:  Avg 101.0 rows x 48 workers.  Max 101 rows (seg0) with 0.003 ms to first row, 2432 ms to end, start offset by 268 ms.
                     ->  Append-only Columnar Scan on a  (cost=0.00..10733143.16 rows=20833126 width=4)
                           Rows out:  0 rows (seg0) with 33 ms to end, start offset by 265 ms.
 Slice statistics:
   (slice0)    Executor memory: 327K bytes.
   (slice1)    Executor memory: 828K bytes avg x 48 workers, 828K bytes max (seg0).
   (slice2)    Executor memory: 353K bytes avg x 48 workers, 356K bytes max (seg0).
 Statement statistics:
   Memory used: 128000K bytes
 Settings:  enable_bitmapscan=off; enable_seqscan=off; optimizer=off
 Optimizer status: legacy query optimizer
 Total runtime: 4040.718 ms
```

产品体系 - HybridDB for PostgreSQL

HybridDB for PostgreSQL

标准关系数据库
+
实时分析能力

支持任意数量级精准count(distinct)，
支持返回大结果集(没有记录数限制)，
支持翻页查询，
支持游标。

数据写入实时，查询无延迟，
实时BUILD索引，查询无延迟，
无需索引维护，
支持事务，
完全兼容ACID标准，不会出现数据陡降异常问题，
数据两副本，支持高可用，
支持多表任意列JOIN、GROUP BY，无需维表，
支持UDF函数，
支持UDF函数排序，
支持UDF函数索引，
支持表达式索引，
支持修改表结构，
支持单条、多条、批量UPDATE，
支持多值列的存储、搜索、显示查询（内容可见），

产品体系 - HybridDB for PostgreSQL

HybridDB for PostgreSQL

分布式

MPP架构

基于分布式大规模并行处理，随计算单元的添加线性扩展存储及计算能力，充分发挥每个计算单元的OLAP计算效能

分布式事务

支持分布式的SQL OLAP统计及窗口函数，支持分布式PL/pgSQL存储过程、触发器，实现数据库端分布式计算过程开发

学习分析

MADlib机器学习

为数据科学用户提供基于SQL的海量数据机器学习工具，支持50多种数据库内置学习算法

GIS地理分析

符合国际OpenGIS标准的地理数据混合分析，通过单条SQL即可从海量数据中进行地理信息的分析，如：人流量、面积统计、行踪等分析

产品体系 - HybridDB for PostgreSQL

HybridDB for PostgreSQL

数据互通

异构数据导入

通过MySQL数据库可以通过mysql2pgsql进行高性能数据导入，同时业界流行的ETL工具均可支持以HybridDB为目标的ETL数据导入

OSS异构存储

可将存储于OSS中的格式化文件作为数据源，通过外部表模式进行实时操作，使用标准SQL语法实现数据查询

透明数据复制

支持数据从PostgreSQL/PPAS透明流入，持续增量无需编程处理，简化维护工作，数据入库后可再进行高性能内部数据建模及数据清洗

安全性

IP白名单配置

最多支持配置1000个允许连接RDS实例的服务器IP地址，从访问源进行直接的风险控制。

DDOS防护

在网络入口实时监测，当发现超大流量攻击时，对源IP进行清洗，清洗无效情况下可以直接拉进黑洞。

产品体系 - HybridDB for PostgreSQL

HybridDB for PostgreSQL

分析SQL兼容

- 复杂分析SQL、UDF
- 窗口查询、Grouping Sets
- plpython, pljava，函数计算
- 机器学习库 MADlib
- 空间数据PostGIS
- JSON、数组、全文检索
- 估值计算 HLL
- 扩展插件（无限可能）

海量OLAP

- 多机并行计算
- 行、列混存、支持压缩
- 复合分布键、随机分布键
- 范围、枚举分区
- hash\merge\nestloop JOIN
- 多阶段JOIN
- 支持任意字段 JOIN
- MetaScan

分级存储

- 本地SSD存储
  - 4TB/主机(有效)
- OSS二级存储
  - 30MB/s/线程
- 物理Mirror
  - 大事务，低延迟

产品体系 - PolarDB for PostgreSQL

传统分库分表架构缺陷

传统分库分表
架构：
限制、缺陷。

1. 扩容不方便（数据重分布）
2. 分布键变更很麻烦
3. 分布键选择（架构设计）谨慎
4. 跨库JOIN性能差
5. 分布式事务性能差
6. SQL限制多、功能缺失多
7. 应用改造成本巨大
8. 全局一致性时间点恢复几乎不可实现

shard之间隔绝、需要跨节点JOIN的话要在中间件层实现、效率低

产品体系 - PolarDB for PostgreSQL

产品体系 - PolarDB for PostgreSQL

产品体系 - PolarDB for PostgreSQL

PolarDB

Median **HTAP**

1. 实例扩容方便（秒级新增只读节点）
2. 存储扩容方便（几乎无限扩展IO、带宽）
3. 不存在分布键问题
4. 不存在跨库JOIN的效率问题
5. 不存在分布式事务问题
6. SQL没有任何限制
7. 应用无需改造
8. 支持全局一致性时间点恢复
9. 只读节点延迟毫秒内
10. 所有节点都支持并行计算

100TB+
千亿+

PostgreSQL

阿里云PostgreSQL产品生态

OLTP+海量OLAP+分级存储->统一PG技术栈

阿里云PostgreSQL产品形态

- PG,PPAS
  - 单节点云盘版
  - 双节点
  - 三节点
  - 只读节点
  - 计算存储分离版
- HDB PG
  - ssd
  - hdd

云数据库优势

- 自动化备份
- 克隆实例
- 弹性扩容
- 内置同城双机HA
- 允许用户基于RDS搭建物理、逻辑备库
- 行业特性提炼、功能产品化
  - varbitx
  - oss_fdw (分级存储、冷暖分离、无限扩容)
  - ali decode
- 无缝对接云平台其他产品
- 开源生态插件整合，BUG修复
  - PostGIS, pgrouting, HLL, MADlib, zhparser, rum, PLpython, PLjava ... ...

云数据库优势

- 云生态，合作伙伴生态，用户生态
- 内核能力，持续交付能力
- ADAM，持续进化能力
- 开源生态

RDS for PostgreSQL优势

- 性能
  - 检查点smooth化
  - 解决drop 大量对象时的备库延迟(高德case)
  - 高并发性能提升240倍
  - GPU加速空间计算
- 功能
  - release cache,
  - varbitx,
  - oss_fdw分级存储,
  - alidecode,
  - cloud DBA

RDS for PPAS优势

分级存储：

热数据(本地表)

暖数据(外部表)

HybridDB for PostgreSQL优势

- 性能
  - sortkey
    - 单独使用可避免排序， 可结合metascan
  - metascan
    - 优于brin的地方: 实时更新、树结构搜索高效
  - 向量化执行引擎
- 功能
  - create extension;
  - 标签类型 - roaring bitmap 4阶段化,

阿里云PostgreSQL生态开源

- 开源
  - rds_dbsync
  - gpdb extension

PostgreSQL **AP特性**

- JIT
- 向量计算
- 单机
  - parallel CPU加速
  - GPU 加速
- 多机
  - citus
  - pg shard
  - … …

TPC-H
200G
20min
单机 ECS 测试

PostgreSQL **TP特性**

PG vs 商用数据库（1/12)

Feature	PG	商用数据库
SQL 标准	SQL:2011	
窗口查询	支持	
grouping sets	支持	
CTE	支持	
递归调用	支持	
存储过程	支持	
存储过程 扩展语言	支持	

PG vs 商用数据库 (2/12)

Feature	PG	商用数据库
nestloop join	支持	
hash join	支持	
merge join	支持	
分组聚合	支持	
哈希聚合	支持	

PG vs 商用数据库 (3/12)

Feature	PG	商用数据库
继承表	支持	
分区表	支持	
外部表	支持	
check约束	支持	
排他约束	支持	
online DDL	支持	
dblink	支持	
几何类型	支持	

PG vs 商用数据库 (4/12)

Feature	PG	商用数据库
网络类型	支持	
范围类型	支持	
全文检索类型	支持	
JSON类型	支持	
数组类型	支持	
树结构类型	支持	
自定义类型	支持	
自定义转换	支持	

PG vs 商用数据库 (5/12)

Feature	PG	商用数据库
b-tree索引	支持	
hash索引	支持	
gin索引	支持	
gist索引	支持	
SP-GiST索引	支持	
BRIN索引	支持	
rum索引	支持	
bloom索引	支持	

PG vs 商用数据库 (6/12)

Feature	PG	商用数据库
表达式索引	支持	
多列复合索引	支持	
多索引合并扫描	支持	
分区(条件)索引	支持	

PG vs 商用数据库 (7/12)

Feature	PG	商用数据库
PDB	支持	
流式物理备库	支持	
流式逻辑备库	支持	
逻辑订阅	支持	
多副本强同步	支持	
PITR	支持	
一主多从	支持	
级联从库	支持	
多MASTER	支持	

PG vs 商用数据库 (8/12)

Feature	PG	商用数据库
外部表 条件下推	支持	
同构外部表	支持	
异构外部表	支持	

PG vs 商用数据库 (9/12)

Feature	PG	商用数据库
ECPG	支持	
扩展类型	支持	
扩展操作符	支持	
扩展索引	支持	
扩展函数	支持	
扩展聚合	支持	
扩展插件	支持	
扩展函数语言	支持	

PG vs 商用数据库 (10/12)

Feature	PG	商用数据库
扩展订阅接口	支持	
扩展FDW接口	支持	
扩展采样接口	支持	
扩展数据扫描接口	支持	
扩展WAL接口	支持	
钩子	支持	
性能探针	支持	
扩展存储接口	12+ 支持	

PG vs 商用数据库 (11/12)

Feature	PG	商用数据库
2D空间数据	支持	
3D，4D空间数据	支持	
拓扑数据	支持	
栅格数据	支持	
空间图式搜索	支持	
实时全文检索	支持	
实时模糊查询	支持	
实时ADHoC查询	支持	
关系图式搜索	支持	

PG vs 商用数据库 (12/12)

Feature	PG	商用数据库
实时流计算	支持	
时序数据插件	支持	
估值计算	支持	
实时相似查询	支持	

HDB PG vs. AWS Redshift (1/3)

Feature	HybridDB for PostgreSQL	AWS Redshift
SQL base	PostgreSQL 8.3 & Greenplum 4.3	PostgreSQL 8.0
Oracle Compatibility	20+ functions	No
Spatial data types and functions	Yes	No
Table Partitioning	Yes	Do not for local table Just support at external table
Hybrid, Column, and Row Orientation	Yes	No
Vacuum/Analyze	Allow multi processors	Just one processor
Indexes	Yes	No
Machine Learning	Yes (MADlib)	No (reload to other product)

HDB PG vs. AWS Redshift (2/3)

Data Type	HybridDB for PostgreSQL	AWS Redshift
11 General Data types: SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP, TIMESTAMPTZ	Yes	Yes
Spatial GIS (box, circle, lseg, path, point, polygon)	Yes	No
Network IPv4 / IPv6 (cidr, inet, macaddr)	Yes	No
JSON	Yes	No
uuid	Yes	No
xml	Yes	No

HDB PG vs. AWS Redshift (3/3)

	HybridDB for PostgreSQL	AWS Redshift
Massively parallel processing (MPP)	MPP	MPP
Data compression	Yes	Yes
Query optimizer	Yes	Yes
Compiled code	JDBC / ODBC / libq	JDBC / ODBC / libq
Row-oriented table	Yes	No
Column-oriented table	Yes	Yes
External table	Yes	Yes
User-Defined Functions	PL/pgsql, PL/Java	No

目录

- 产品介绍
- 生态介绍
- 应用案例
- 开发、管理实践
- 数据库原理
- 参考文档

产品生态

生态
云市场
专家
服务
SaaS
QBI,DataV
...
数据传输、
迁移服务
MySQL
ORACLE
ODPS， ADS
Others ... ...
集团业务
公有云
业务
专有云
混合云
业务
阿里云
RDS 中台
OLTP
+
OLAP
HybridDB for PostgreSQL
分级存储：
本地
OSS表：并行读写、压缩、行存、列存
阿里云OSS (海量分布式对象存储)
对象存储

mybatis

http://www.mybatis.org/mybatis-3/configuration.html

mybatis等框架，不支持的语法都可以通过UDF来实现，例如批量更新。
https://github.com/digoal/blog/blob/master/201712/20171214_01.md

```
<dataSource type="org.myproject.C3P0DataSourceFactory">
  <property name="driver" value="org.postgresql.Driver"/>
  <property name="url" value="jdbc:postgresql:mydb"/>
  <property name="username" value="postgres"/>
  <property name="password" value="root"/>
</dataSource>
```

mysql, pg 类型映射

- MySQL 'enum' THEN LOWER(CONCAT(c.COLUMN_NAME, '_t'))
- MySQL 'tinyint' THEN 'smallint'
- MySQL 'mediumint' THEN 'integer'
- MySQL 'tinyint unsigned' THEN 'smallint'
- MySQL 'smallint unsigned' THEN 'integer'
- MySQL 'mediumint unsigned' THEN 'integer'
- MySQL 'int unsigned' THEN 'bigint'
- MySQL 'bigint unsigned' THEN 'numeric(20)'
- MySQL 'double' THEN 'double precision'
- MySQL 'float' THEN 'real'
- MySQL 'datetime' THEN 'timestamp'
- MySQL 'longtext' THEN 'text'
- MySQL 'mediumtext' THEN 'text'
- MySQL 'blob' THEN 'bytea'
- MySQL 'mediumblob' THEN 'bytea'

mysql, pg 语法差异

- **类型、语法差异详情：**
- https://github.com/digoal/blog/blob/master/201801/20180117_01.md
- https://github.com/digoal/blog/blob/master/201801/20180117_02.md

- 常见mysql,pg 语法差异
  - mysql: [LIMIT {[*offset*,] *row_count* | *row_count* OFFSET *offset*}]
  - pg: [LIMIT { *count* | ALL }] [OFFSET *start* [ROW | ROWS]]
- PG扩展语法、操作符、函数、类型、索引接口、插件
  - https://www.postgresql.org/docs/10/static/sql-commands.html
  - https://www.postgresql.org/docs/10/static/functions.html
  - https://www.postgresql.org/docs/10/static/datatype.html
  - B-tree, hash, GiST, SP-GiST, GIN, and BRIN, bloom.
    - https://www.postgresql.org/docs/10/static/sql-createindex.html
  - https://www.postgresql.org/docs/10/static/contrib.html

阿里巴巴内部应用

高德、淘点点
阿里云、优酷、阿里妈妈、B2B
淘宝、新零售、菜鸟、公共平台、飞猪、YUNOS、智慧城市、蚂蚁、未来酒店、阿里云孵化器

目录

- 产品介绍
- 生态介绍
- 应用案例
- 开发、管理实践
- 数据库原理
- 参考文档

PostgreSQL Benchmark

- **数据装载（时序类）**
  - 32Core, 512G, 2*Aliflash SSD
  - 连续24小时多轮数据批量导入测试(平均每条记录长度360字节, 时间字段创建时序索引(BRIN index))
  - 每轮测试插入12TB数据
  - 506万行/s，1.78 GB/s，全天插入4372亿行，154TB数据
- **TPC-B** (1 Select : 3 Update : 1 Insert)
  - 32Core, 512G, 2*Aliflash SSD 10亿数据量, 11万tps, 77万qps
  - Select-Only 100万qps (即使应用缓存失效，也无大碍^_^)
- **TPC-C** (新建订单45,支付43,订单查询4,发货4,库存查询4)
  - 4000个仓库, 400GB数据, 平均每笔事务10几条SQL
  - 32Core, 256GB, Nvme, 100万+ TPmC
- **LinkBench** (Facebook 应用)（图式搜索类）
  - 1亿个node，4亿条关系，(32Core，2 SSD, 512G)
  - (添加NODE，更新NODE，删除NODE，获取NODE信息，添加关系，删除关系，更新关系，关系总数查询，获取多个关系，获取关系列表)
  - 12万 ops (默认测试用例)

TPC-B (ECS 32C, 512G, SSD)

- https://github.com/digoal/blog/blob/master/201809/20180916_01.md
- **1、100亿TPCB 初始化**
- 耗时：8385秒。
- 速度：约119万行/s。
- **2、100亿TPCB 创建索引（24并行）**
- 耗时：43分50秒。
- 速度：约380万行/s。
- **3、100亿TPCB 空间占用**
- 表：1.251 TB
- 索引：209 GB
- **4、100亿TPCB 只读3600秒**
- TPS: 118053
- QPS: 118053
- **5、100亿TPCB 读写3600秒**
- TPS: 42058
- QPS: 210290

1、10亿TPCB 初始化

耗时：879秒。

速度：约113万行/s。

2、10亿TPCB 创建索引（24并行）

耗时：145秒。

速度：约690万行/s。

3、10亿TPCB 空间占用

表：125 GB

索引：21 GB

4、10亿TPCB 只读3600秒

TPS: 109万

QPS: 109万

5、10亿TPCB 读写3600秒

TPS: 93729

QPS: 468645

TPC-B (ECS 32C, 512G, ESSD)

- https://github.com/digoal/blog/blob/master/201809/20180917_01.md
- **1、1000亿TPCB 初始化**
- 耗时：25小时 52分。
- 速度：约107万行/s。
- **2、1000亿TPCB 创建索引（64并行）**
- 耗时：10小时 50分。
- 速度：约250万行/s。
- **3、1000亿TPCB 空间占用**
- 表：12.51 TB
- 索引：2 TB

单表数据量	TEST CASE	QPS	TPS
1000亿	tpcb 活跃数据10亿 只读	998818	998818
1000亿	tpcb 活跃数据100亿 只读	597877	597877
1000亿	tpcb 活跃数据500亿 只读	66678	66678
1000亿	tpcb 活跃数据1000亿 只读	67295	67295
1000亿	tpcb 活跃数据10亿 读写	475595	95119
1000亿	tpcb 活跃数据100亿 读写	426390	85278
1000亿	tpcb 活跃数据500亿 读写	191505	38301
1000亿	tpcb 活跃数据1000亿 读写	175945	35189

TPC-B (ECS 32C, 512G, ESSD)

- https://github.com/digoal/blog/blob/master/201809/20180917_01.md
- **1、10000亿TPCB 初始化**
- 耗时：810688秒。
- 速度：约123万行/s。
- **2、10000亿TPCB 创建索引（64并行）**
- 耗时：427800秒。
- 速度：约233万行/s。
- **3、10000亿TPCB 空间占用**
- 表：120 TB
- 索引：20 TB

单表数据量	TEST CASE	QPS	TPS
1万亿	tpcb 活跃数据1亿 只读	1068052	1068052
1万亿	tpcb 活跃数据10亿 只读	1039962	1039962
1万亿	tpcb 活跃数据100亿 只读	590692	590692
1万亿	tpcb 活跃数据1000亿 只读	23101	23101
1万亿	tpcb 活跃数据10000亿 只读	21492	21492
1万亿	tpcb 活跃数据1亿 读写	393235	78647
1万亿	tpcb 活跃数据10亿 读写	410740	82148
1万亿	tpcb 活跃数据100亿 读写	375010	75002
1万亿	tpcb 活跃数据1000亿 读写	90630	18126
1万亿	tpcb 活跃数据10000亿 读写	80715	16143

TPC-C (ECS 32C, 512G, SSD)

- https://github.com/digoal/blog/blob/master/201809/20180913_01.md
- (1000W数据) 10*100W
- 103万 tpmC

TPC-H (ECS 32C, 512G, SSD)

- https://github.com/digoal/blog/blob/master/201808/20180823_01.md
- SF=10 TPCH 150秒
- SF=200 TPCH 39 min

SF	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18	q19	q20	q21	q22
10	3	2	2	5	3	4	2	2	9	3	1	2	4	2	6	10	27	46	2	10	13	2
200	18	38	25	32	57	8	52	24	66	38	24	26	98	13	58	114	732	595	12	213	124	14

OLTP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
点查，KEY值查询	1亿	112	69万	0.16毫秒
键值更新	1亿	56	22万	0.25毫秒
合并写入(有则更新、无则写入)	1亿	56	22.8万	0.245毫秒
秒杀 - 单条记录并发更新	1	56	23万	0.48毫秒
空间包含，菜鸟精准分包、共享单车.等	1亿个多边形	112	27.9万	0.4毫秒
搜索空间附近对象，LBS，O2O	10亿个经纬度点	112	13.7万	0.8毫秒
空间数据、位置更新(滴、菜鸟、饿)	1亿	56	18万	0.3毫秒
图式搜索（N度搜索、最短路径）	50亿，3度搜索，单次响应2.1毫秒	64	1万	6.6 毫秒

OLTP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
全文检索、写入、实时索引	500万词库，每行64个词，并发写入	56	9.3万	0.6毫秒
数组、写入、实时索引	500万词库，每行64个词，并发写入	56	10万	0.5毫秒
前后模糊查询、实时索引、并发写	128个随机字符	56	14.4万	0.38毫秒
字符串查询、前缀 like 'x%'	1亿，128个随机字符	112	14万	0.8毫秒
字符串查询、后缀 like '%x'	1亿，128个随机字符	112	17.8万	0.63毫秒
字符串查询、前后百分号 like '%x%'	1亿，128个随机字符	56	8.1万	0.68毫秒
字符串查询、相似查询 similary(str, 'xx')	1亿，128个随机字符	56	1531	35毫秒
字符串查询、全文检索 ts @@ tsquery	1亿	56	5.14万	1毫秒

OLTP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
文本特征向量**搜索**	1亿海明码	56	4.9万	1.14毫秒
数组相似**搜索**	1亿，每行24个数组元素	56	1909	29毫秒
时序数据并发**写入**(含时序索引)	批量写入 313.7 万行/s，单步写入27.3万行/s。	56	3137	17.8毫秒
区间**查询**，返回5万条记录 **返回N条，可按比例计算响应时间。**	1亿 （3160万行/s 吞吐）	16	630	25毫秒
IN\EXISTS **查询**	1亿，IN(1,10,100,...100万个元素)		1毫秒~	380毫秒
NOT IN **查询 (无需索引) parallel Hashjoin**	1亿，NOT IN(1,10,100,...100万个元素)		0.8秒~	1.8秒
NOT EXISTS **查询 (无需索引) parallel Hashjoin**	1亿，NOT exists(100万个元素)		0.9秒~	1.2秒

OLTP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
流式处理 - 阅后即焚 - 消费	10亿，消费 **395.2 万行/s**	56	3952	14毫秒
物联网-阅后即焚-读写并测	写入：193万行/s，消费：418万行/s	56		
物理网-阅后即焚-JSON+函数流计算-读写并测	写入：180万行/s，消费：145.8万行/s	56		
单表，无索引，单事务单条写入	单行110字节	56	26万	0.2毫秒
单表，有索引，单事务单条写入	单行110字节	56	10万	0.5毫秒
单表，无索引，单事务多条批量写入	单行110字节，每次提交1000条	56	180万行/s	30.9毫秒
单表，有索引，单事务多条批量写入	单行110字节，每次提交1000条	56	16.7万行/s	334毫秒

OLTP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
多表，无索引，单事务单条写入	动态SQL成为瓶颈	56	18万行/s	0.3毫秒
多表，有索引，单事务单条写入	动态SQL成为瓶颈	56	17万行/s	0.3毫秒
多表，无索引，单事务多条写入	每次提交1000条	56	262.7万行/s	21毫秒
多表，有索引，单事务多条写入	每次提交1000条	56	145万行/s	38毫秒
无日志多表，无索引，单事务多条写入	每次提交1000条	56	813万行/s	6.8毫秒
无日志多表，有索引，单事务多条写入	每次提交1000条	56	733万行/s	7.6毫秒

OLAP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
多表JOIN	10张1000万的表	112	10万	1.08毫秒
大表JOIN、统计	2张1亿，1张100万	56	2.2万	2.5毫秒
大表OUTER JOIN、统计	1千万 OUTER JOIN 1亿			1千万 left join 1亿：11秒 反之：8秒
用户画像-数组包含、透视	1亿，每行16个标签	56	1773	31毫秒
用户画像-数组相交、透视	1亿，每行16个标签	56	113	492毫秒
用户画像-varbitx	1万行，2000亿BIT，与或非			2.5秒
用户画像-多字段任意搜索\聚合、透视	1亿，32个字段，任意字段组合查询	56	3.6万	1.56毫秒
物联网-线性数据-区间实时聚合、统计	1万传感器，10亿记录	56	6266	8.9毫秒

OLAP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
排序	1亿	32		1.4秒
建索引	1亿			PG 10: 38秒； PG11: 15.5秒
并行扫描	1亿	32		0.88秒
并行聚合	1亿	32		0.9秒
并行过滤	1亿	32		1秒
并行JOIN+聚合	1000 万 JOIN 1000 万	32		PG 10: 5.4秒； PG 11: 1秒
并行JOIN+聚合	1亿 JOIN 1亿 (双表过滤到1000万)	32		PG 10: 6.3秒； PG 11: 1.2秒
并行JOIN+聚合	1亿 JOIN 1亿 (单表过滤到1000万)	32		PG 10: 8.5秒； PG 11: 2秒
并行JOIN+聚合	1亿 JOIN 1亿 (无条件JOIN)	32		PG 10: 58.3秒； PG 11: 10.7秒
并行JOIN+聚合	10亿 JOIN 10亿 (双表过滤到1000万)	32		PG 10: 12秒； PG 11: 1秒

OLAP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
并行HASH AGG	10亿 (PG 11)	64		11秒
VOPS + 异步并行	10亿，聚合查询（PG 10）	56		2秒
多表并行扫描 (parallel append)	2亿 (PG 11)	64		0.6 秒
求TOP-K	100亿 (PG 11)	64		40秒

OLAP Benchmark(56 Core ECS, PG 10)

测试详情：
https://github.com/digoal/blog/blob/master/201711/readme.md
https://github.com/digoal/blog/blob/master/201801/20180102_04.md
https://github.com/digoal/blog/blob/master/201802/20180204_01.md
https://github.com/digoal/blog/blob/master/201802/20180204_03.md
https://github.com/digoal/blog/blob/master/201802/20180201_01.md
https://github.com/digoal/blog/blob/master/201802/20180202_02.md
https://github.com/digoal/blog/blob/master/201802/20180210_01.md
https://github.com/digoal/blog/blob/master/201808/20180823_01.md

HDB PG OLAP Benchmark(1 Host)

CASE	数据量	并发	TPS	平均响应时间
非分布键 group by , TOP-K	10亿、列存、压缩、101个元素	1		1.8秒
非分布键 count(distinct)	10亿、列存、压缩、101个元素	1		1.8秒
非分布键 order by, TOP-K	10亿、行存、压缩、索引	1		50毫秒
非分布键 JOIN，group by，TOP-K	10亿、列存、压缩、101个元素	1		56秒
非分布键 JOIN，count(distinct)	10亿、列存、压缩、101个元素	1		53秒
非分布键 JOIN，order by, TOP-K	10亿、列存、压缩	1		141秒
分布键 group by, TOP-K	10亿、列存、压缩、10亿元素	1		1.67秒
分布键 order by, TOP-K	10亿、行存、压缩、索引	1		42毫秒
分布键 JOIN, order by , TOP-K	10亿、列存、压缩、10亿元素	1		119秒

HDB PG OLAP Benchmark(50 Hosts)

CASE	数据量	并发	TPS	平均响应时间
海量数据导入，走OSS（带统计干扰时）	100亿，5.5 TB，单行576字节。	<1%	重复度	1250秒（压缩后360秒）
海量数据内部生成	1000亿，55 TB，单行576字节。			2112秒
1000亿，前后模糊查询				246秒
并发前缀查询100亿 + 写入目标（2.5亿）	100亿，5个并发前缀查询，写入			149秒
复杂统计 + 模糊查询 + 写入目标（0.5亿）	100亿，统计伴随，模糊查询写			64秒
海量导入 + 模糊查询 + 写入目标（0.5亿）	100亿，伴随导入，模糊查询写			56秒
海量导入+510字节模糊查询+写入（0.5亿）	100亿，伴随导入，模糊查询写			59秒
100亿 JOIN 5000万，双可变字段510字节+写	写0.5亿			109秒
100亿，双可变字段510字节，聚合写	写100亿			481秒

HDB PG OLAP Benchmark(50 Hosts)

CASE	数据量	并发	TPS	平均响应时间
100亿，批量（JOIN）删除100万				53秒
100亿，批量（join）更新5000万	更新41秒，更新结果写入B,49秒			
100亿，流式返回5000万末尾100万行	游标技术			1.4秒
100亿，任意字段组合排序				54秒
100亿，表达式排序				
100亿，任意经纬度点。任意多边形包含查询	返回最近100条			0.16秒

HDB PG OLAP Benchmark(50 Hosts)

CASE	数据量	并发	TPS	平均响应时间
100亿，任意经纬度点。任意多边形包含查询	聚合			4.5 秒
修改表结构-加字段加默认值	100亿			3.5秒
修改表结构-扩展长度	100亿			17秒
修改表结构-删除字段	100亿			0.01秒
非结构化数据，100亿。	展示功能			

RDS PG应用案例

Case1(标签 - 多值类型)

- 多值类型与GIN索引应用
  - Array, Hstore, JSON

用户画像-数组包含、透视	1亿，每行16个标签	56	1773	31毫秒

案例

- XX单车、新零售-XX小店。
- 人群标签
  - {标签:结束时间, ...}
- 透视人群、求交并差
  - 包月人群
  - 促销人群
- 痛点
  - 无法结构化
- **RDS for PG**
  - **多值类型解决结构化难点问题**
  - **arr @> array[?,?,...]**
  - **arr && array[?,?,...]**
  - **not arr @> array[?,?,...]**
- **搜索加速**
  - **GIN倒排，(标签元素倒排索引)**
  - **udf1(标签s) -> [1维数组]　，　udf2(标签s)->[2维数组]**

```
postgres=# select * from tbl;
 id |        info
----+------------------
  1 | {a:100,b:10}
  2 | {a:15,b:20,c:99}
  3 | {c:78,b:100}
(3 rows)
```

```
postgres=# select * from tbl where get_label(info) @> array['a'] and get_weight(info, 'a')::float8 >20;
 id |      info
----+--------------
  1 | {a:100,b:10}
(1 row)
```

案例

- xx生物科技
  - 几万列浮点
  - 几十万行
  - 100GB左右
  - 计算任意组合(物种、序列分段)的欧氏空间距离

几万列、
超过单表
列宽

  - 采用float8[]存储几万列浮点，UDF，计算XX距离。

```
postgres=# select * from compute_eu_dist(array[1,2,3,4,5]);
 kind1 | kind2 |     euc_dist
-------+-------+------------------
     2 |     1 | 57768.4024741692
     1 |     3 | 57866.2845528097
     1 |     4 | 57632.9837382263
     5 |     1 |   57779.36595061
     3 |     2 | 58004.3926579964
     4 |     2 | 57593.0783041254
     5 |     2 | 57802.9690538283
     3 |     4 | 57837.6707750057
     3 |     5 | 57921.5524014271
     4 |     5 | 57818.9181109456
(10 rows)

Time: 100.582 ms
```

案例-架构设计、代码、实操手册

- 数组类型和操作符、函数、索引
  - https://www.postgresql.org/docs/10/static/arrays.html
  - https://www.postgresql.org/docs/10/static/functions-array.html
- 多重含义数组UDF检索
  - https://github.com/digoal/blog/blob/master/201801/20180124_02.md
- 生物科技
  - https://github.com/digoal/blog/blob/master/201712/20171227_01.md

Case2(搜索 - GIN)

- 搜索需求分类：
  - 全文检索
  - 模糊搜索、前缀、后缀、前后模糊
  - 相似搜索
  - 任意字段组合搜索

Case2(搜索 - GIN)

全文检索、写入、实时索引	500万词库，每行64个词，并发写入	56	9.3万	0.6毫秒
前后模糊查询、实时索引、并发写	128个随机字符	56	14.4万	0.38毫秒
字符串查询、前缀 like 'x%'	1亿，128个随机字符	112	14万	0.8毫秒
字符串查询、后缀 like '%x'	1亿，128个随机字符	112	17.8万	0.63毫秒
字符串查询、前后百分号 like '%x%'	1亿，128个随机字符	56	8.1万	0.68毫秒
字符串查询、全文检索 ts @@ tsquery	1亿	56	5.14万	1毫秒
字符串查询、相似查询 similary(str, 'xx')	1亿，128个随机字符	56	1531	35毫秒
用户画像-多字段任意搜索\聚合、透视	1亿，32个字段，任意字段组合查询	56	3.6万	1.56毫秒

案例

- XXX域名服务
  - 模糊查询、相似查询
  - 10亿级记录模糊搜索
- XXX某CRM系统
  - 任意字段全文检索、模糊查询
  - 词汇(phase)查询
  - 10亿级记录多字段任意组合搜索
- 新零售-营销、分销链路
  - 多值类型检索
  - 10亿级记录数组&|搜索
- **QA（医疗行业、搜索、知识库）**
  - **相似问题、相似地址、相似病例。。。**
- **GA**
  - **车牌模糊搜索**

痛点

- 全文检索**无法支持"模糊查询"**
  - （例如**域名并非分词**）

- 数据库与搜索引擎**一致性维护麻烦**

云产品方案、效果

- RDS PG
  - gin 倒排索引，
    - 支持多值类型、多字段任意条件检索
  - bloom索引
    - 支持多字段任意组合等值条件过滤
  - 多索引 bitmap scan
    - 多个索引合并扫描
  - pg_trgm 支持
    - 模糊查询、相似查询、正则查询
  - zhparser中文分词插件支持中文分词

1. GIN 复合(倒排+聚集)索引
2. 分词索引
3. bloom复合索引
4. pg_trgm, fuzzymatch

全文检索技术

```
select * from tbl where ts @@ to_tsquery('pg & alibaba')
  order by ts_rank(ts, to_tsquery('pg & alibaba') );
  -- order by ts_rank_cd(ts, to_tsquery('pg & alibaba') )
```

https://www.postgresql.org/docs/10/static/textsearch-controls.html#TEXTSEARCH-RANKING

全文检索技术 - 词距离条件

```
select * from tbl where ts @@ '速度 <距离值> 激情'::tsquery;

如

select * from tbl where ts @@ '速度 <1> 激情'::tsquery;
```

```
|                                                
```

按距离范围搜索

自定义UDF，RANGE相交操作判断。

```
-- 测试，取距离是1到2（不含2）的

postgres=# select get_lexeme_pos_range(ts, '速度', '激情'), * from ts_test where ts @@ tsquery '速度 & 激情' and get_lexeme_pos_range(ts, '速度', '激情') && int4range(1,2) limit 1;
 get_lexeme_pos_range | id |          info          |                    ts
----------------------+----+------------------------+-------------------------------------------
 [1,2)                |  1 | 电影速度与激情8的票房破亿 | '激情':3 '电影':1 '破':5 '票房':4 '速度':2
(1 row)

Time: 0.713 ms

-- 测试，取距离是2到5（不含5）的

postgres=# select get_lexeme_pos_range(ts, '速度', '激情'), * from ts_test where ts @@ tsquery '速度 & 激情' and get_lexeme_pos_range(ts, '速度', '激情') && int4range(2,5) limit 1;
 get_lexeme_pos_range | id |                          info                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
postgres=# select get_lexeme_pos_range(ts, '速度', '激情'), * from ts_test where ts @@ tsquery '速度 & 激情' and get_lexeme_pos_range(ts, '速度', '激情') && int4range(2,5) limit 1;
 get_lexeme_pos_range | id |                          info                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
 [3,4)                |  1 | 激情，创新，坚持，速度-- 北京北方创业出租汽车公司优质服务小花絮 | '优质服务':9 '公司':8 '出租汽车':7 '创业':6 '创新':2 '北京':5 '坚持':3 '小花':10 '激情':1 '絮':11 '速度':4
(1 row)

Time: 0.682 ms
```

全文检索技术 - 词距离条件

带距离

```
postgres=# select * from ts_test where ts @@ '速度 <1> 激情'::tsquery;
 id |          info            |                    ts
----+--------------------------+-------------------------------------------
  1 | 电影速度与激情8的票房破亿 | '激情':3 '电影':1 '破':5 '票房':4 '速度':2
(1 row)
```

不带距离

```
postgres=# select * from ts_test where ts @@ '速度 & 激情'::tsquery limit 5;
 id |                              info                              |
----+----------------------------------------------------------------+-----------------------------------------
  1 | 激情，创新，坚持，速度-- 北京北方创业出租汽车公司优质服务小花絮 | '优质服务':9 '公司':8 '出租汽车':7 '创业':6
  1 | 电影速度与激情8的票房破亿                                      | '激情':3 '电影':1 '破':5 '票房':4 '速度':2
  2 | 激情，创新，坚持，速度-- 北京北方创业出租汽车公司优质服务小花絮 | '优质服务':9 '公司':8 '出租汽车':7 '创业':6
  2 | 激情，创新，坚持，速度-- 北京北方创业出租汽车公司优质服务小花絮 | '优质服务':9 '公司':8 '出租汽车':7 '创业':6
  2 | 激情，创新，坚持，速度-- 北京北方创业出租汽车公司优质服务小花絮 | '优质服务':9 '公司':8 '出租汽车':7 '创业':6
(5 rows)
```

全文检索技术 - 内置ranking

```
UPDATE tt SET ti =
    setweight(to_tsvector(coalesce(title,'')), 'A')    ||
    setweight(to_tsvector(coalesce(keyword,'')), 'B')  ||
    setweight(to_tsvector(coalesce(abstract,'')), 'C') ||
    setweight(to_tsvector(coalesce(body,'')), 'D');
```

支持4种weight：
标题、作者、摘要、
内容

```
SELECT title, ts_rank_cd(textsearch, query) AS rank
FROM apod, to_tsquery('neutrino|(dark & matter)') query
WHERE query @@ textsearch
ORDER BY rank DESC
LIMIT 10;
                    title                        |   rank
-------------------------------------------------+----------
 Neutrinos in the Sun                            |      3.1
 The Sudbury Neutrino Detector                   |      2.4
 A MACHO View of Galactic Dark Matter            |  2.01317
 Hot Gas and Dark Matter                         |  1.91171
 The Virgo Cluster: Hot Plasma and Dark Matter   |  1.90953
 Rafting for Solar Neutrinos                     |      1.9
 NGC 4650A: Strange Galaxy and Dark Matter       |  1.85774
 Hot Gas and Dark Matter                         |   1.6123
 Ice Fishing for Cosmic Neutrinos                |      1.6
 Weak Lensing Distorts the Universe              | 0.818218
```

内置ranking
算法

全文检索技术 - ranking掩码

Both ranking functions take an integer *normalization* option that specifies whether and how a document's length should impact its rank.

0 (the default) ignores the document length

1 divides the rank by 1 + the logarithm of the document length

2 divides the rank by the document length

4 divides the rank by the mean harmonic distance between extents
(this is implemented only by ts_rank_cd)

8 divides the rank by the number of unique words in document

16 divides the rank by 1 + the logarithm of the number of unique words in document

32 divides the rank by itself + 1

```
SELECT title, ts_rank_cd(textsearch, query, 32 /* rank/(rank+1) */ ) AS rank
FROM apod, to_tsquery('neutrino|(dark & matter)') query
WHERE  query @@ textsearch
ORDER BY rank DESC
LIMIT 10;
                     title                      |        rank
------------------------------------------------+-------------------
 Neutrinos in the Sun                           | 0.756097569485493
 The Sudbury Neutrino Detector                  | 0.705882361190954
 A MACHO View of Galactic Dark Matter           | 0.668123210574724
 Hot Gas and Dark Matter                        |  0.65655958650282
 The Virgo Cluster: Hot Plasma and Dark Matter  | 0.656301290640973
 Rafting for Solar Neutrinos                    | 0.655172410958162
 NGC 4650A: Strange Galaxy and Dark Matter      | 0.650072921219637
 Hot Gas and Dark Matter                        | 0.617195790024749
 Ice Fishing for Cosmic Neutrinos               | 0.615384618911517
 Weak Lensing Distorts the Universe             | 0.450010798361481
```

内置ranking
算法

全文检索技术 - 内置ranking

The two ranking functions currently available are:

`ts_rank([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ]) returns float4`

Ranks vectors based on the frequency of their matching lexemes.

`ts_rank_cd([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ]) returns float4`

This function computes the *cover density* ranking for the given document vector and query, as described in Clarke, Cormack, Processing and Management", 1999. Cover density is similar to `ts_rank` ranking except that the proximity of matching lexeme

This function requires lexeme positional information to perform its calculation. Therefore, it ignores any "stripped" lexemes in t Section 12.4.1 for more information about the `strip` function and positional information in `tsvectors`.)

For both these functions, the optional *weights* argument offers the ability to weigh word instances more or less heavily depending or word, in the order:

```
{D-weight, C-weight, B-weight, A-weight}
```

If no *weights* are provided, then these defaults are used:

```
{0.1, 0.2, 0.4, 1.0}
```

全文检索技术 - 自定义ranking

1、店铺标签表：

```
create table tbl (
  shop_id int8 primary key,   -- 店铺 ID
  tags text[],                -- 数组，标签1,标签2,.....
  scores float8[]             -- 数组，评分1,评分2,.....
);

create index idx_tbl_1 on tbl using gin(tags);
```

```
国民_足浴,国民_餐饮,娱乐_KTV

0.99,0.1,0.45
```

2、标签权值表：

```
create table tbl_weight (
  tagid int primary key,   -- 标签ID
  tagname name,            -- 标签名
  desc text,               -- 标签描述
  weight float8            -- 标签权值
);

create index idx_tbl_weight_1 on tbl_weight (tagname);
```

```
create or replace function cat_ranking(tsquery) returns float8 as $$
declare

begin
  for each x in array (contains_element) loop
    search hit element's score.
    search hit element's weight.
    cat ranking and increment
  end loop;
  return res;
end;
$$ language plpgsql strict;
```

ranking sort index

- rum index am
  - CREATE EXTENSION rum;
  - CREATE INDEX rumidx ON test_rum USING rum (a rum_tsvector_ops);

```
SELECT t, a <=> to_tsquery('english', 'beautiful | place') AS rank
    FROM test_rum
    WHERE a @@ to_tsquery('english', 'beautiful | place')
    ORDER BY a <=> to_tsquery('english', 'beautiful | place');
                t                 |  rank
----------------------------------+---------
 It looks like a beautiful place | 8.22467
 The situation is most beautiful | 16.4493
 It is a beautiful               | 16.4493
(3 rows)
```

前缀模糊查询背景技术

- create index idx on tbl (**col text_patten_ops**);
- select * from tbl where **{col ~ '^前缀' | like '前缀%'};**
- 自动
  - postgres=# explain select * from pre where c1 like '你%';
  - QUERY PLAN
  - ----------
  - Index Scan using idx_pre on pre (cost=0.29..2.71 rows=1 width=21)
  - Index Cond: ((c1 ~>=~ '你'::text) AND (c1 ~<~ '伧'::text))
  - Filter: (c1 ~~ '你%'::text)
  - (3 rows)

  - postgres=# select **chr(ascii('你')+1);**
  - chr
  - -----
  - 伧
  - (1 row)

后缀模糊查询背景技术

- postgres=# create index idx_pre1 on pre (**reverse(c1) text_pattern_ops**);
- CREATE INDEX
  - postgres=# select **reverse('你好abcd ');**
  - reverse
  - ----------
  - dcba好你
  - (1 row)

  - postgres=# explain select * from pre where **reverse(c1) like reverse('结尾')||'%';**
  - QUERY PLAN
  - ----------
  - Index Scan using idx_pre1 on pre (cost=0.29..45.93 rows=50 width=21)
  - Index Cond: ((reverse(c1) ~>=~ '尾结'::text) AND (reverse(c1) ~<~ '尾绔'::text))
  - Filter: (reverse(c1) ~~ '尾结%'::text)
  - (3 rows)

前后模糊查询背景技术

- postgres=# create index idx_pre2 on pre using **gin (c1 gin_trgm_ops);**
- CREATE INDEX
  - postgres=# select **show_trgm('abcde');**
  - show_trgm
  - ---
  - {" a"," ab",abc,bcd,cde,"de "}
  - (1 row)

  - postgres=# explain select * from pre where **c1 like '%abc%';**
  - QUERY PLAN
  - ---
  - Bitmap Heap Scan on pre (cost=3.61..4.82 rows=1 width=21)
  - Recheck Cond: (c1 **~~ '%abc%'::text**)
  - -> Bitmap Index Scan on idx_pre2 (cost=0.00..3.61 rows=1 width=0)
  - Index Cond: (c1 **~~ '%abc%'::text**)
  - (4 rows)

前后模糊查询背景技术

- 当前后模糊查询低于3个字符时，需要使用表达式索引, split
  - https://github.com/digoal/blog/blob/master/201704/20170426_01.md

支持中文的前后模糊查询背景技术

- https://github.com/digoal/blog/blob/master/201605/20160506_02.md
- Command: CREATE DATABASE
- Description: create a new database
- Syntax:
- CREATE DATABASE name
- [[WITH] [OWNER [=] user_name]
- [TEMPLATE [=] template]
- [ENCODING [=] encoding]
- [LC_COLLATE [=] lc_collate] 不能=C
- [LC_CTYPE [=] lc_ctype] 不能=C
- [TABLESPACE [=] tablespace_name]
- [ALLOW_CONNECTIONS [=] allowconn]
- [CONNECTION LIMIT [=] connlimit]
- [IS_TEMPLATE [=] istemplate]]

相似查询

```
create or replace function get_res(
  text,    -- 要按相似搜的文本
  int8,    -- 限制返回多少条
  float4 default 0.3,  -- 相似度阈值，低于这个值不再搜搜
  float4 default 0.1   -- 相似度递减步长，直至阈值
) returns setof record as $$
declare
  lim float4 := 1;
begin
  -- 判定
  if not ($3 <= 1 and $3 > 0) then
    raise notice '$3 must >0 and <=1';
    return;
  end if;

  if not ($4 > 0 and $4 < 1) then
    raise notice '$4 must >0 and <=1';
    return;
  end if;
```

相似查询

```
loop
  -- 设置相似度阈值
  perform set_limit(lim);

  return query select similarity(info, $1) as sml, * from tbl where info % $1 order by sml desc limit $2;

  -- 如果有，则退出loop
  if found then
    return;
  end if;

  -- 否则继续，降低阈值
  -- 当阈值小于0.3时，不再降阈值搜索，认为没有相似。
  if lim < $3 then
    return;
  else
    lim := lim - $4;
  end if;
 end loop;
end;
$$ language plpgsql strict;
```

相似查询

- select * from get_res(
  - '输入搜索文本',
  - 输入限制条数,
  - 输入阈值,
  - 输入步长
-) as t(sml float4, id int, info text);

相似查询

- postgres=# select * from get_res('睹掇贼展跃鬠哈四餧麾鄟賃青乖浈鳋揃摦垭岮操彴淒鋺約韉夗缝特鐍夗鯩垭縳墙靰禮徛亦猰庴釅恮嚕鉞翱鈒嘹雍岈擦寵淽蒸彼鴙糜娕籹偌勹浰', 10, 0.4, 0.05) as t(sml float4, id int, info text);
- -[RECORD 1]----------
- sml | 0.882353
- id | 1
- info | 彿睹掇贼展跃鬠哈四餧麾鄟賃青乖浈鳋揃摦垭岮操彴淒鋺約韉夗缝特鐍夗鯩垭縳墙靰禮徛亦猰庴釅恮嚕鉞翱鈒嘹雍岈擦寵淽蒸彼鴙糜娕籹偌勹浰鶙

- Time: 52.852 ms

任意字段组合条件搜索-内部加速技术1 bitmap scan

- 多个独立的索引的BITMAP SCAN（或单个GIN多字段复合索引(using gin (c1,c2,...))）
  - select * from table where col1 = ? and col2 = ?;
  - 合并扫描后，访问的数据块非常少，速度很快。

```
+-----------------------------------------------+
|1000000000010000000100000000000001111000000000| bitmap 1
|0000010000010001000100000000010000100000000010| bitmap 2
 &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
|0000000000010000000100000000000000100000000000| Combined bitmap
+-----------+-------+--------------+----------+
            |       |              |
            v       v              v
Used to scan the heap only for matching pages:
+-----------------------------------------------+
|___________X_______X______________X__________|
+-----------------------------------------------+
```

ADHoc搜索-加速技术2 - rum index

- 20亿行，每行50个字段，任意字段AND\OR组合查询，**毫秒级响应，1万+TPS。**
  - https://github.com/digoal/blog/blob/master/201802/20180228_01.md

案例-架构设计、代码、实操手册

- 全文检索
  - https://github.com/digoal/blog/blob/master/201603/20160310_01.md
  - https://github.com/digoal/blog/blob/master/201712/20171206_01.md
  - https://github.com/digoal/blog/blob/master/201712/20171205_02.md
  - https://www.postgresql.org/docs/10/static/textsearch.html
  - https://github.com/digoal/blog/blob/master/201801/20180123_01.md (含距离搜索)
  - https://github.com/digoal/blog/blob/master/201612/20161231_01.md
- 模糊、正则查询
  - https://www.postgresql.org/docs/10/static/pgtrgm.html
- 相似查询
  - https://www.postgresql.org/docs/10/static/pgtrgm.html
  - 算法：contrib/pg_trgm/trgm_regexp.c
  - https://github.com/digoal/blog/blob/master/201802/20180202_01.md
  - https://github.com/digoal/blog/blob/master/201802/20180205_03.md
- 多字段任意组合查询
  - https://www.postgresql.org/docs/10/static/indexes-bitmap-scans.html
  - https://www.postgresql.org/docs/10/static/bloom.html
  - https://www.postgresql.org/docs/10/static/btree-gin.html
  - https://github.com/digoal/blog/blob/master/201802/20180228_01.md
  - https://github.com/digoal/blog/blob/master/201805/20180518_02.md

Case3(特征、相似)

- 相似
  - 数组相似
  - 文本特征值相似
  - 图片相似

文本特征向量搜索	1亿海明码	56	4.9万	1.14毫秒
数组相似搜索	1亿，每行24个数组元素	56	1909	29毫秒

案例

- 导购系统
  - 1亿历史导购文章：**数组（商品ID）相似判断**
  - 实时判定盗文
  - 毫秒级
- 新零售-商品相关短文相似查询
  - 10亿级短文
  - **短文特征值海明码相似识别**
  - 切分，通过smlar插件overlap求相似
  - 毫秒级
- 图像搜索系统
  - 10亿级图片
  - **相似图片识别**
  - 对象识别（doing）
  - 毫秒级

相似度
去重

猜你喜欢

超酷FRONTIART 1:18 诺米克概念车 one电动车跑车汽车模

浏览轨迹(图、文本向量)、推荐相似商品

痛点

- 多值存储和高效检索
  - 海量多值数据，相似查询，毫秒响应
  - 海量短文相似查询，毫秒响应
- 图像特征值存储和高效检索
  - 图片相似查询，毫秒响应
  - 图像识别，毫秒响应

云产品方案、效果

- RDS PG
  - smlar插件
    - 相似文本、数组
    - 海明码切片(转码)相似
  - imgsmlr插件
    - 相似图片

海明码切分

id	hmval	hmarr
1	0101011111111010000001001011101101100011111101111101101100000011	{1_0101011111111010,2_0000010010111011,3_0110001111110111,4_1111011111011011}
2	1101011000010000000000000000011101101111101110111010000001010101 1	{1_1101011000010000,2_0000000000001110,3_1101111101110111,4_0111011101000000}
3	0101000010110110110010001010100010101001001010111111011000110011	{1_0101000010110110,2_1100100010101000,3_1010100100101011,4_0010101111110110}
4	0111000111100011111000100111000011101111110000011110101101000100	{1_0111000111100011,2_1110001001110000,3_1110111111000001,4_1100000111101011}
5	0010111010101011111010011110110010011110111111111001110111001001 1	{1_0010111010101011,2_1110100111101100,3_1001111011111111,4_1111111100111011}
6	0110111110011100100110010111101000000001110001100001111001010110	{1_0110111110011100,2_1001100101110100,3_0000001110001100,4_1000110000111100}
7	0100110100111001110011011111010011110111010100100010101011011011 0	{1_0100110100111001,2_1100110111101001,3_1110111010100100,4_1010010001010101}
8	0110010111001100111000011011011100001100111111101111011101010001 0	{1_0110010111001100,2_1110000110110111,3_0000110011111110,4_1111111011110110}
9	0110111010110000001010101111000101110000010011100011100101000100	{1_0110111010110000,2_0010101011110001,3_0111000001001110,4_0100111000111001}
10	0101101000000110100101100011111111000101110001010011100110101011	{1_0101101000000110,2_1001011000111111,3_1100010111000101,4_1100010100111001}

(10 rows)

图像特征值提取与存储

Input image (2^Nx2^N)
wavelet transform
Transformed image (2^Nx2^N)

15
15
11 ... 151
21 ··· 215
.....
151 ... 1515

图像特征值比对

61,33% similarity

87,55% similarity

23,56% similarity

RDS PostgreSQL

- smlar插件
  - cosine, overlap, tdidf相似
- imgsmlr插件
  - 图像特征值，图像相似搜
  - 图像识别(doing)
- smlar插件
  - 短文海明距离<N，相似性
  - 海明码切片+smlar overlap高速检索

案例-架构设计、代码、实操手册

- 数组相似
  - https://github.com/digoal/blog/blob/master/201701/20170116_02.md
  - https://github.com/digoal/blog/blob/master/201701/20170116_03.md
  - https://github.com/digoal/blog/blob/master/201701/20170116_04.md
  - https://github.com/digoal/blog/blob/master/201701/20170112_02.md
- 海明码相似
  - https://github.com/digoal/blog/blob/master/201708/20170804_01.md
- 图片相似搜索
  - https://github.com/digoal/blog/blob/master/201607/20160726_01.md
  - https://github.com/digoal/blog/blob/master/201809/20180904_02.md
  - https://github.com/digoal/blog/blob/master/201809/20180904_03.md
  - https://github.com/digoal/blog/blob/master/201809/20180904_04.md

Case4(画像、特征、透视)

- 画像系统

用户画像-数组包含、透视	1亿，每行16个标签	56	1773	31毫秒
用户画像-数组相交、透视	1亿，每行16个标签	56	113	492毫秒
用户画像-varbitx	1万行，2000亿BIT，与或非			2.5秒

案例

- 心选
- 生意参谋
- 优酷
- B2B卖家智能运营
- 菜鸟
- 友盟
  - 多值列标签+任意字段组合 圈选

1、多值列：标签
2、多列任意组合搜索Adhoc SQL

案例

- XXXpush
  - 业务背景： ToB 实时圈人系统
  - 数据来源：实时标签数据
  - 数据规模：单表10亿条记录，单个B-1亿用户，1万个标签字段。
  - 数据描述：每个用户的标签数据
  - 查询需求：任意标签组合圈人
    - 100毫秒级响应
  - 并发需求：200+
  - DML需求：实时标签分钟级体现到查询中

痛点

- 1万个TAG，大宽表。
  - 目前没有数据库支持。需要拆分成多表。
- 原方案成本高，收益低。
  - 8台，数据延迟**天级**别，响应时间接近**分钟级**，并发不到100。

云产品方案、效果

- RDS PG
  - 阿里云varbitx插件
  - 翻转存储 tag, userid_varbit
  - 用户ID字典化
- 单台RDS PG
  - 标签数据合并延迟**10分钟**级
  - 查询响应**毫秒级**
  - 支持并发**500+**
  - 裸空间节省80倍算上索引至少240倍节省

案例

- 数据银行项目(一期)
- 20亿+用户，万级标签，大屏展示（100+标签组合圈选透视）
  - 1、求COUNT，2000亿（20亿用户，100个标签组合）USER_IDS，响应速度**2.6秒**。
  - 2、求USERID明细，返回500万用户ID位置，**692毫秒**。
  - 3、求USERID明细，返回BITMAP，500万个BIT，**224毫秒**。

案例

- 数据银行项目(一期)
- 超大BIT分段，加速
- bitand，同时包含
- bitor，包含任意
- bitand bitxor，包含1但是不包含2

案例

- 数据银行项目(二期)
- HDB PG
- roaring bitmap

- https://github.com/digoal/blog/blob/master/201801/20180127_01.md

- roaring bitmap vs varbit
- 1、压缩
- 2、BITMAP操作接口
- 3、分阶段聚合

案例-架构设计、代码、实操手册

- roaring bitmap
  - https://github.com/digoal/blog/blob/master/201801/20180127_01.md
- 阿里云varbitx插件
  - https://github.com/digoal/blog/blob/master/201712/20171212_01.md
  - https://github.com/digoal/blog/blob/master/201610/20161021_01.md
  - https://github.com/digoal/blog/blob/master/201705/20170502_01.md
  - https://github.com/digoal/blog/blob/master/201706/20170612_01.md
  - https://github.com/digoal/blog/blob/master/201712/20171223_01.md

Case5(模拟股票交易系统-时序数据)

- SchemaLess
- 同类业务
  - 轨迹
  - 时序特性

	代码	名称	涨幅%	现价	涨跌	买价	卖价	总量
1	600030	中信证券	-8.50	18.20	-1.69	18.19	18.20	364.9万
2	601198	东兴证券	-8.12	24.79	-2.19	24.79	24.81	746274
3	002736	国信证券	-7.72	19.95	-1.67	19.91	19.95	621341
4	600958	东方证券	-7.38	23.33	-1.86	23.33	23.34	457017
5	601555	东吴证券	-7.35	15.63	-1.24	15.63	15.65	711700
6	000728	国元证券	-7.23	23.73	-1.85	23.70	23.73	463199
7	600109	国金证券	-7.03	16.01	-1.21	16.00	16.01	919679
8	601688	华泰证券	-6.74	19.37	-1.40	19.37	19.39	680707
9	000776	广发证券	-6.68	18.01	-1.29	17.99	18.01	684825
10	002500	山西证券	-6.36	15.45	-1.05	15.43	15.45	675172
11	600999	招商证券	-6.26	21.13	-1.41	21.13	21.14	452137
12	002673	西部证券	-6.12	35.44	-2.31	35.41	35.42	627193
13	601788	光大证券	-6.00	23.35	-1.49	23.32	23.33	604189
14	601901	方正证券	-5.78	9.61	-0.59	9.62	9.63	132.8万
15	601099	太平洋	-5.67	9.48	-0.57	9.47	9.48	144.5万
16	000166	申万宏源	-5.67	10.98	-0.66	10.97	10.98	611083
17	000783	长江证券	-5.65	12.02	-0.72	12.01	12.02	753265
18	000750	国海证券	-5.43	12.71	-0.73	12.72	12.74	538391
19	601377	兴业证券	-5.34	11.17	-0.63	11.16	11.17	125.5万
20	601211	国泰君安	-5.20	22.78	-1.25	22.77	22.78	595793
21	600369	西南证券	-4.98	9.73	-0.51	9.72	9.73	102.3万
22	600061	国投安信	-4.96	25.46	-1.33	25.46	25.49	290470
23	000686	东北证券	-1.56	18.31	-0.29	18.29	18.31	998157
24	600837	海通证券	-	-	-	-	-	0

时序数据并发**写入**(含时序索引)	批量写入 313.7 万行/s，单步写入27.3万行/s。	56	3137	17.8毫秒
区间**查询**，返回5万条记录 **返回N条，可按比例计算响应时间。**	1亿 （3160万行/s 吞吐）	16	630	25毫秒

案例

- 证券模拟交易系统
  - 业务背景：模拟股票交易
  - 数据来源：实时股票数据
  - 数据规模：300亿
  - 数据描述：股票交易数据，大宽表。
  - 查询需求：查询任意股票任意时间区间的数据，要求返回60条数据10毫秒以内
  - 并发需求：1000+
  - DML需求：准实时写入

痛点

- 按任意时间滑动查询。
- 写入、查询延迟要求低。

IO放大问题

索引庞大问题

云产品方案、效果

- RDS PG
  - Schemaless方案(UDF)
  - 任意股票任意时间段查询响应时间0.04毫秒
    - 同行竞品为10毫秒
  - 股票数据写入速度约22万行/s，远超业务需求。
  - 以十年的股票数据来计算，约300亿数据。单机可以搞定。
  - BRIN 时序索引，**10亿~1MB。**

时序数据
时序索引

timescaleDB 插件

- 自动切片，写入性能无损耗
- merge append(时序窗口查询性能优化)
- 数据维护API
- https://github.com/timescale/timescaledb

菜鸟 - 实时FEED LOG监测

- **海量FEED LOG实时质量统计(若干台RDS PG)**
  - avg, min, max, sum, **percentile_disc**
  - **单rds pg指标: 15亿point/分钟**
  - **intarray,** aggs_for_arrays, list partition

案例-架构设计、代码、实操手册

- 证券案例
  - https://github.com/digoal/blog/blob/master/201704/20170417_01.md
- 自动切片
  - https://github.com/digoal/blog/blob/master/201711/20171102_02.md
  - https://github.com/digoal/blog/blob/master/201705/20170511_01.md
  - https://github.com/digoal/blog/blob/master/201709/20170927_03.md
- 菜鸟-实时FEED LOG检测
  - https://github.com/digoal/blog/blob/master/201802/20180205_04.md
- timescaleDB插件
  - https://github.com/digoal/blog/blob/master/201801/20180129_01.md
  - https://github.com/digoal/blog/blob/master/201704/20170409_05.md

Case6(空间应用)

- **电子围栏、LBS、AOI、POI、路网、导航、自动驾驶、路径规划**

空间包含，菜鸟精准分包、共享单车.等	1亿个多边形	112	27.9万	0.4毫秒
搜索空间附近对象，LBS，O2O	10亿个经纬度点	112	13.7万	0.8毫秒
空间数据、位置更新(滴、菜鸟、饿)	1亿	56	18万	0.3毫秒

Case6(空间应用)

- GIS空间数据管理
- 电子围栏(不规则多边形)
  - 共享自行车还车点管理
  - 公务用车限行管理
  - 车辆限行区域管理
  - 放牧区域管理
  - 菜鸟-包裹快递员分配管理
  - 基于实时位置的广告营销
  - 智能家居（IoT）
  - 封印

案例 - 不规则多边形

- 不具备空间索引的数据库，**编码索引。存在弊端：（因实际建筑规划不可能完全与编码匹配，相邻小区存在编码重叠），一个实体小区的召回可能会跨快递员。**

裹裹Geohash四边形网格地图
—附近小件员召回

滴滴六边形格网地图
—实时运力供需

案例 -不规则多边形

- 菜鸟aoi
  - AOI库的构建，
  - 精准分单
- 共享单车
  - 限制还车地点

空间数据
管理

成都市全域AOI切割产品

场景 - LBS

- 探探、微信、陌陌、订餐、POI（附近对象"如加油站"搜索）

场景 - LBS

- https://github.com/digoal/blog/blob/master/201804/20180417_01.md

繁荣的五道口，一公里圈10家小意思。 300公里外才有一家，新疆人民吴军在厕所

半径大了性能差

半径小了圈不着

GeoHash:
圈多大完全
摸不到头脑

场景 - LBS

天目山路 Tian Mu Shan Road
黄龙体育中心
杭州市第十五中学
杭州市陈经纶体育学校
黄龙体育中心西大道
玉古路
求是路
松木场
曙光路
西溪路
黄龙路 Huanglong Road
© OpenStree

场景 - LBS

geometry+GiST
不存在问题。

where st_dwithin (pos,中心,1000) order by pos <-> 中心;

Case 餐饮配送

- 骑手
  - 实时状态、位置更新
- 配送
  - 时空多维度实时搜索

相比ES性能提升50倍。

https://github.com/digoal/blog/blob/master/201711/20171107_48.md

空间复合搜索

- btree_gist, intarray
  - create index idx_tbl_1 on tbl using gist(xx,xxx,pos);
  - 普通字段，数组标签包含，空间包含，KNN。
- partial index
  - create index idx_tbl_1 on tbl using gist(xx,pos) where xxx > xx;
- https://github.com/digoal/blog/blob/master/201711/20171107_48.md
- where
- xx=xx and
- xxx between x and x and
- st_contain(polygon, pos)
- order by pos <-> ? limit xx;

热力图

- https://github.com/digoal/blog/blob/master/201807/20180725_02.md

Case

Case

- 计算
  - 每幅卫星图包含2w*2w = 4亿个点（原始数据）
  - 运算算法，使用至少30次拍摄的卫星图（至少120亿个点）进行计算，得到最多4亿个有效点的数据（目前是3千万左右）。
  - 计算后的4亿点数据，有100多个属性，其中包括一个经纬度数据。
- 查询
  - 1、给定图层（有效值快照），输入一个点，搜索最近的点的一行数据（100多个属性值，绘图）。
    - RDS PG响应速度1毫秒左右。
  - 给定一个区域（客户端使用线段、多边形圈出来的不规则多边形），输出里面的所有点的数据。绘图。
  - 多表JOIN的查询，TPS万级，TP类请求

案例 - 点云、路径规划

- 菜鸟 - 自动配送机器人
  - pointcloud
- 高德
  - 地图、导航
- 淘点点
  - 路径规划

案例 - 路径拟合

- Gxxx，路径补全
  - 监控盲点路径补全
  - pgrouting路径插件
- Gxxx，人车拟合
  - 拟合司机、乘客
  - 时间、空间圈选计算

空间数据管理

pgrouting

- All Pairs Shortest Path, Johnson’s Algorithm
- All Pairs Shortest Path, Floyd-Warshall Algorithm
- Shortest Path A*
- Bi-directional Dijkstra Shortest Path
- Bi-directional A* Shortest Path
- Shortest Path Dijkstra
- Driving Distance
- K-Shortest Path, Multiple Alternative Paths
- K-Dijkstra, One to Many Shortest Path
- Traveling Sales Person
- Turn Restriction Shortest Path (TRSP)

(OpenStreetMap)OSM & POI

- **osm2pgrouting_10.x86_64** : Import tool for OpenStreetMap data to pgRouting
- **osm2pgsql.x86_64** : Imports map data from OpenStreetMap to a PostgreSQL database

- 兴趣点相关业务（导航、...）

- 导出POI，导入PostgreSQL(PostGIS)
- https://wiki.openstreetmap.org/wiki/Osmosis/Installation
- http://wiki.openstreetmap.org/wiki/Osmosis/Detailed_Usage
- https://help.openstreetmap.org/questions/4065/getting-specific-poi-data-and-keeping-them-up-to-date

案例-架构设计、代码、实操手册

- 电子围栏
  - https://github.com/digoal/blog/blob/master/201710/20171031_01.md
  - https://github.com/digoal/blog/blob/master/201708/20170803_01.md
- 多边形搜索
  - https://github.com/digoal/blog/blob/master/201710/20171004_01.md
  - https://github.com/digoal/blog/blob/master/201710/20171005_01.md
  - https://github.com/digoal/blog/blob/master/201711/20171107_06.md
- 点云
  - https://github.com/digoal/blog/blob/master/201705/20170519_02.md
  - https://github.com/digoal/blog/blob/master/201705/20170523_01.md
- 路径规划
  - https://github.com/digoal/blog/blob/master/201508/20150813_03.md
  - http://pgrouting.org/
- KNN搜索
  - https://github.com/digoal/blog/blob/master/201711/20171107_07.md
  - https://github.com/digoal/blog/blob/master/201308/20130806_01.md
  - https://github.com/digoal/blog/blob/master/201804/20180417_01.md
- 商旅问题
  - https://github.com/digoal/blog/blob/master/201704/20170409_01.md

Case7(文本情感)

- 实时文本情感分析
  - 聊天记录
  - 评论
  - XF\FK
  - …

涉黄、虚假评论、恐怖袭击、不文明用语、…

案例 - 实时舆情分析

- 实时文本情感分析、舆情系统

详细链接

- 实时文本情感分析、舆情系统
- 导入 java\python lib
  - https://help.aliyun.com/document_detail/50594.html
- plpython开发手册
  - https://www.postgresql.org/docs/10/static/plpython.html
  - https://gpdb.docs.pivotal.io/43100/ref_guide/extensions/pl_python.html
- pljava 开发手册
  - https://gpdb.docs.pivotal.io/43100/ref_guide/extensions/pl_java.html
- plR开发手册
  - https://gpdb.docs.pivotal.io/43100/ref_guide/extensions/pl_r.html
- MADlib SQL机器学习库手册
  - http://madlib.apache.org/docs/latest/index.html
- R MADlib对接手册
  - https://cran.r-project.org/web/packages/PivotalR/
- python MADlib库对接手册
  - https://pypi.python.org/pypi/pymadlib/0.1.7

Case8(树、多表关联、多值、图搜)

- 树形结构
  - 复杂JOIN
  - 递归查询
  - 图式搜索(graph search)

金融风控、
QA系统、
好友关系、
舆情系统、
药品监管(溯源)、
串货监管(溯源) 、

图式搜索（N度搜索、最短路径）	50亿，3度搜索，单次响应2.1毫秒	64	1万	6.6 毫秒

Case8(树、多表关联、多值、图搜)

- https://github.com/digoal/blog/blob/master/201804/20180408_03.md

案例 - 图式搜索、伴随分析

- xxx小微金融项目
  - 业务背景：中xx小微金融项目
  - 数据来源：
    - 爬虫、合作平台（税务、蚂蚁、银行...）
  - 数据规模：
    - 全网100亿，中xx10亿级
    - 大平台、线上线下多份存储
  - 数据描述：
    - 企业信息、法人信息、爬虫爬到的相关信息、纳税信息、。。。展示图谱，辅助评估贷款风险
  - 查询需求：
    - 企业多级关系查询、图谱展示
  - 并发需求：100+
  - DML需求：实时写入

痛点

- 树形结构数据，递归查询
- 众多关联企业信息
  - 多表JOIN，关联关系复杂
- 输出多级关联企业（类似人脉关系）
- 要求高速响应图式搜索需求

RDS PG

- ltree数据类型
- 多表JOIN
- 递归查询
  - CTE
- 数组
  - key1:{val1, val2, val3, ...}
  - GIN索引
  - 包含、相交
  - = any(array)

案例 - 知识图谱

```
create table a(
  c1 int,                 -- 点1
  c2 int,                 -- 点2
  prop jsonb,             -- 点1,2对应的边的属性，使用JSON存储，包括权重，关系等等。
  primary key (c1,c2)     -- 主键
);

create index idx_a_2 on a(c1, COALESCE(((prop ->> 'weight'::text))::float8, 0));
```

```
create or replace function graph_search1(
  IN i_root int,                          -- 根据哪个节点开始搜
  IN i_depth int  default 99999,          -- 搜索层级、深度限制
  IN i_limit int8 default 2000000000,     -- 限制每一层返回的记录数
  IN i_weight float8 default 0,           -- 限制权重
  OUT o_path int[],                       -- 输出：路径，ID 组成的数组
  OUT o_point1 int,                       -- 输出：点1 ID
  OUT o_point2 int,                       -- 输出：点2 ID
  OUT o_link_prop jsonb,                  -- 输出：当前两点之间的连接属性
  OUT o_depth int                         -- 输出：当前深度、层级
) returns setof record as $$
```

API:
N度搜索；
最短路径；
流式返回；

百亿关系网、
3层关系查询、
2.5毫秒

适用场景

- 广度搜索、深度搜索（不含权重）
- 好友关系系统
- 好友推荐系统
- 陌生人交友
- 知识图谱
- 风控

不适用

AgensGraph

- http://bitnine.net/agensgraph/
- 专业图数据库 base on PG

- pgrouting插件
- http://pgrouting.org/

- All Pairs Shortest Path, Johnson's Algorithm
- All Pairs Shortest Path, Floyd-Warshall Algorithm
- Shortest Path A*
- Bi-directional Dijkstra Shortest Path
- Bi-directional A* Shortest Path
- Shortest Path Dijkstra
- Driving Distance
- K-Shortest Path, Multiple Alternative Paths
- K-Dijkstra, One to Many Shortest Path
- Traveling Sales Person
- Turn Restriction Shortest Path (TRSP)

案例-架构设计、代码、实操手册

- 图式搜索 案例链接
  - https://github.com/digoal/blog/blob/master/201708/20170801_01.md
  - https://github.com/digoal/blog/blob/master/201612/20161213_01.md
  - https://github.com/digoal/blog/blob/master/201801/20180102_04.md
  - https://github.com/digoal/blog/blob/master/201804/20180408_03.md
- ltree 树类型
  - https://github.com/digoal/blog/blob/master/201105/20110527_01.md
  - https://github.com/digoal/blog/blob/master/201709/20170923_01.md
- 递归查询
  - https://github.com/digoal/blog/blob/master/201705/20170519_01.md
  - https://github.com/digoal/blog/blob/master/201703/20170324_01.md
  - https://github.com/digoal/blog/blob/master/201612/20161201_01.md
  - https://github.com/digoal/blog/blob/master/201611/20161128_02.md
  - https://github.com/digoal/blog/blob/master/201611/20161128_01.md
  - https://github.com/digoal/blog/blob/master/201607/20160725_01.md
  - https://github.com/digoal/blog/blob/master/201607/20160723_01.md
  - https://github.com/digoal/blog/blob/master/201604/20160405_01.md
  - https://github.com/digoal/blog/blob/master/201512/20151221_02.md
  - https://github.com/digoal/blog/blob/master/201210/20121009_01.md
  - https://github.com/digoal/blog/blob/master/201209/20120914_01.md
- JSON, plv8
  - https://github.com/plv8/plv8/blob/master/doc/plv8.md#scalar-function-calls

(阿里云RDS PG 内置plv8语言)

- 数组
  - https://github.com/digoal/blog/blob/master/201711/20171107_18.md
  - https://github.com/digoal/blog/blob/master/201711/20171107_19.md
  - https://github.com/digoal/blog/blob/master/201711/20171107_20.md

case9(订阅、单元化、容灾、多写)

- 线上RDS PG
- 逻辑订阅、或物理订阅 到用户端PG

案例-架构设计、代码、实操手册

- 订阅功能(单元化)
  - https://github.com/digoal/blog/blob/master/201702/20170227_01.md
  - https://github.com/digoal/blog/blob/master/201707/20170711_01.md

case 10 (跨域、跨库、sharding)

- 传统企业数据库上云，突破单库容量限制
  - 多库组集群、相互可访问、可写、可同步。
  - 功能点：FDW(外部表、远程表)、DBLINK、匿名、逻辑订阅。

FDW外部表：
访问异库表，犹如访问本地表，没有限制。
支持读写、JOIN等，支持PUSHDOWN算子。
偶尔访问的异地数据，采用dblink、或fdw，简化逻辑。

逻辑订阅（单元化）：
经常访问的异地数据，使用订阅功能订阅到本地。

OSS 无限对象存储：
用于冷存储，支持多实例共享访问。

库与库不再孤立，形成整体。

sharding1 (FDW)

继承、自动数据路由、支持跨库访问、
PUSH DOWN(JOIN、sort、where、UDF。。。)

sharding2 (Citus)

shard table,
reference table

https://github.com/digoal/blog/blob/master/201808/20180824_02.md

1\choose column
2\choose distribute policy
3\set shard count
4\choose colocate group
5\shard hash value range
6\balance to worker node

	events shards		page shards	
	shard	hash range	shard	hash range
NODE 1	1	-2147483648 ≤ x ≤ -1073741825	5	-2147483648 ≤ x ≤ -1073741825
	2	-1073741824 ≤ x ≤ -1	6	-1073741824 ≤ x ≤ -1
NODE 2	3	0 ≤ x ≤ 1073741823	7	0 ≤ x ≤ 1073741823
	4	1073741824 ≤ x ≤ 2147483647	8	1073741824 ≤ x ≤ 2147483647

x = hash(distribution_column)

sharding2 (citus)

sharding2 (citus)

citus
MX特性
指定worker携带metadata

sharding2 (citus)

案例-架构设计、代码、实操手册

- PG Sharding
  - https://github.com/digoal/blog/blob/master/201808/20180824_02.md
  - https://github.com/digoal/blog/blob/master/201809/20180905_01.md
- FDW
  - 总览: https://wiki.postgresql.org/wiki/Fdw
  - 文件fdw: https://www.postgresql.org/docs/10/static/file-fdw.html
  - PG fdw: https://www.postgresql.org/docs/10/static/postgres-fdw.html
  - MySQL fdw: https://github.com/EnterpriseDB/mysql_fdw
  - Oracle FDW: https://pgxn.org/dist/oracle_fdw/
  - SQL Server FDW: https://pgxn.org/dist/tds_fdw/
- DBLINK
  - https://www.postgresql.org/docs/10/static/dblink.html
- 订阅功能(单元化)
  - https://github.com/digoal/blog/blob/master/201702/20170227_01.md
  - https://github.com/digoal/blog/blob/master/201707/20170711_01.md

case11 (流式处理 - 阅后即焚)

流式处理 - 阅后即焚 - 消费	10亿，消费 **395.2 万行/s**	56	3952	14毫秒
物联网-阅后即焚-读写并测	写入：193万行/s，消费：418万行/s	56		
物理网-阅后即焚-JSON+函数流计算-读写并测	写入：180万行/s，消费：145.8万行/s	56		

case11 (流式处理 - 阅后即焚)

- 流式处理，高并发写入，快速消费处理。
- 处理后的数据被删除。
- 要求：
  - 数据快速写入
  - 数据写入后必须持久化
  - 快速消费被写入的记录（例如订阅，或者用于业务上的流式计算，计算结果保留）
  - 消费和计算必须在一个事务完成

流式处理 - 阅后即焚

函数计算:
plpgsql,
plpython,
pljava
313.7万
行/s
395.2万
行/s
INSERT
阅后即焚(CTE)
delete from table
returning *
do something.
物联网数据、
FEED数据
JSON非结构
化
PostgreSQL
TABLE(s)
1、写入确保持久
化。
2、消费、处理在
事务中完成

为什么需要流计算

- 实时分析需求
  - 大查询的时效性
- 过滤有效数据
  - 实时数据清洗
- 预警需求
  - 实时数据预警，电子围栏、物联网异常指标、监控系统

流计算和数据库有关系吗？

PostgreSQL流计算原理

方法1、pipelineDB (批处理，低延迟，大吞吐，100+万行/s)

方法2、rule、trigger (实时处理，实时，小吞吐，单步写30+万行/s，批量写100+万行/s)

方法3、insert on conflict (实时处理，实时，小吞吐，单步写30+万行/s ，批量写100+万行/s)

方法4、阅后即焚 (批处理，低延迟，大吞吐，100+万行/s)

案例1-流式预警

- 根据规则发现数据异常，并通知应用程序
- 传统手段
  - 异步查询、实效性较差、重复劳动较多
- 流计算手段
  - 实时或异步、异步消息通道

案例1-流式预警

select xxx from xxx
where 指标字段>=阈值
and xxxx
order by xxx;

问题：

1. 索引巨大、
2. 存在大量无用功查询。

案例1-流式预警

create index idx on table (字段) where 阈值>xxx;

只有异常数据才建索引。

partial index

案例1-流式预警

Syntax:
NOTIFY channel [, payload]
Syntax:
LISTEN channel

案例1-流式预警

- create table tbl(sid int, content jsonb);

- create or replace function monitor(jsonb) returns boolean as $$
- declare
- begin
  - **if xxx then return true; -- 条件1**
  - **else if xxx then return true; -- 条件2，**
  - **else if .....; --条件N**
  - **else return false;**
  - **end if;**
- end;
- $$ language plpgsql strict;

- create or replace rule r1 as on insert to tbl **do {also|instead}** select pg_notify('channel_name', (NEW.content)::text) **where monitor(NEW.content)**;
  - **可落明细、可不落明细、可按需落明细，实时压缩(例如5分钟落一个明细点，其他只落告警点)。**

- client:
  - listen 'channel_name';

案例1-流式预警

吞吐较小、性价比低

百亿+/天 吞吐

案例2 - 流式统计(avg,count,min,max,sum)

- https://github.com/digoal/blog/blob/master/201711/20171123_02.md

案例2 - 流式统计(avg,count,min,max,sum)

- https://github.com/digoal/blog/blob/master/201711/20171123_02.md

```
create table tbl (
  sid int primary key,
  v1 int,
  crt_time timestamp,
  cnt int8 default 1,                        -- 统计值, 默认为1, 等于1时表示第一条记录
  sum_v float8 default 0,                    -- 统计值, 默认为0
  min_v float8 default float8 'Infinity',    -- 统计值, 默认设置为这个类型的最大值
  max_v float8 default float8 '-Infinity'    -- 统计值, 默认设置为这个类型的最小值
);
```

案例2 - 流式统计(avg,count,min,max,sum)

- https://github.com/digoal/blog/blob/master/201711/20171123_02.md

```
insert into tbl (sid, v1, crt_time) values (:sid, :v1, now())
on conflict (sid) do update set
  v1=excluded.v1,
  crt_time=excluded.crt_time,
  cnt=tbl.cnt+1,
  sum_v=case tbl.cnt when 1 then tbl.v1+excluded.v1 else tbl.sum_v+excluded.v1 end,
  min_v=least(tbl.min_v, excluded.v1),
  max_v=greatest(tbl.max_v, excluded.v1)
;
```

分区+批量写入
336万 行/s

案例2 - 流式统计(avg,count,min,max,sum)

- https://github.com/digoal/blog/blob/master/201711/20171123_02.md

```
postgres=# select * from tbl order by sid limit 10;
 sid |    v1    |          crt_time          | cnt |   sum_v   |  min_v   |  max_v
-----+----------+----------------------------+-----+-----------+----------+----------
   1 | 26479786 | 2017-11-23 20:27:43.134594 |  14 | 740544728 | 11165285 | 90619042
   2 | 25755108 | 2017-11-23 20:27:43.442651 |  10 | 414224202 |  2813223 | 83077953
   3 | 51068648 | 2017-11-23 20:27:48.118906 |  11 | 501992396 | 13861878 | 79000001
   4 | 81160224 | 2017-11-23 20:27:37.183186 |  17 | 902219309 |    23429 | 99312338
   5 | 70208701 | 2017-11-23 20:27:35.399063 |   6 | 374351692 | 40289886 | 96340616
   6 | 77536576 | 2017-11-23 20:27:46.04372  |  15 | 649447876 | 12987896 | 80478126
   7 | 31153753 | 2017-11-23 20:27:46.54858  |   8 | 386687697 | 19697861 | 95097076
   8 | 11339236 | 2017-11-23 20:27:40.947561 |  12 | 657650588 | 11339236 | 97211546
   9 | 46103803 | 2017-11-23 20:27:38.450889 |  10 | 594843053 |  9192864 | 92049544
  10 | 55630877 | 2017-11-23 20:27:28.944168 |   9 | 383123573 |  3877866 | 76604940
(10 rows)
```

多维流式计算

- 1、定义明细表
- create table tbl(c1 int not null, c2 int not null, c3 int not null, c4 int not null, c5 int not null);
- 2、定义每个维度的目标统计表
- create table cv1_tbl (c1 int primary key, cnt int8 default 1);
- create table cv2_tbl (c2 int, c3 int, c5 int, sum_v float8 default 0, cnt int8 default 1, primary key (c2,c3)) ;
- .....
- 其他维度
- 3、定义维度表的insert on conflict SQL
- insert into cv1_tbl (c1) values (NEW.c1) on conflict (c1) do update set cnt=cv1_tbl.cnt+1;
- insert into cv2_tbl (c2,c3,c5) values (NEW.c2, NEW.c3, NEW.c5) on conflict (c2,c3) do update set cnt=cv2_tbl.cnt+1, sum_v=case cv2_tbl.cnt when 1 then cv2_tbl.c5+excluded.c5 else cv2_tbl.sum_v+excluded.c5 end;
- 4、定义明细表trigger或rule，顺序调用insert on conflict 写入多个维度表
- create rule r1 as on insert to tbl **do {instead|also} insert** into cv1_tbl (c1) values (NEW.c1) on conflict (c1) do update set cnt=cv1_tbl.cnt+1;
- create rule r2 as on insert to tbl **do {instead|also} insert** into cv2_tbl (c2,c3,c5) values (NEW.c2, NEW.c3, NEW.c5) on conflict (c2,c3) do update set cnt=cv2_tbl.cnt+1, sum_v=case cv2_tbl.cnt when 1 then cv2_tbl.c5+excluded.c5 else cv2_tbl.sum_v+excluded.c5 end;

并行、多维、流式计算

- 1、定义明细分区表
- 2、定义每个维度的目标统计表
- 3、定义维度表的insert on conflict SQL
- 4、定义明细分区表trigger或rule，顺序调用insert on conflict 写入多个维度表

案例3-流式概率计算

- http://docs.pipelinedb.com/probabilistic.html
  - bloom filter: distinct概率判定, select **distinct (...) 快速判断**
  - count-min sketch: **per element's freq 概率 select col,count(*) group by col 快速统计+判定**
  - filtered-space saving **top-k**: element's bucket & freq 概率
  - HLL: **COUNT(DISTINCT ...)**
  - T-Digest: Rank base statistic. **percentile_cont 水位值.**

案例3-流式实时估算

- https://github.com/digoal/blog/blob/master/201711/20171123_02.md
- **HLL插件**
- **create extension hll;**
- create table tbl (
  - grpid int, userid int, dt date,
  - cnt int8 default 1,
  - hll_userid hll default hll_empty(), -- 估算字段
  - primary key (grpid, dt)
  -);
- insert into tbl (grpid, userid, dt) values () on conflict (grpid, dt)
- do update set
- cnt=tbl.cnt+1,
- hll_userid= case tbl.cnt when 1 then hll_add(hll_add(tbl.hll_userid, hll_hash_integer(tbl.userid)), hll_hash_integer(excluded.userid)) else hll_add(tbl.hll_userid, hll_hash_integer(excluded.userid)) end ;

案例3-流式实时估算

```
postgres=# select grpid,userid,cnt,hll_cardinality(hll_userid) from tbl limit 10;
 grpid  |  userid   | cnt | hll_cardinality
--------+-----------+-----+-----------------
 775333 | 642518584 |  13 |              13
  17670 | 542792727 |  11 |              11
  30079 | 311255630 |  14 |              14
  61741 | 945239318 |  10 |              10
 808051 | 422418318 |  14 |              14
 620850 | 461130760 |  12 |              12
 256591 | 415325936 |  15 |              15
 801374 | 373207765 |   9 |               9
 314023 | 553568037 |  12 |              12
```

滑窗分析 - RDS PG与HDB PG都适用

滑窗分析 - RDS PG与HDB PG都适用

- 估值滑窗(最近7天UV)
  - SELECT date, #hll_union_agg(users) OVER seven_days FROM daily_uniques **WINDOW seven_days AS (ORDER BY date ASC ROWS 6 PRECEDING);**
- 统计滑窗(最近7天精确UV， SUM，AVG。。。)
  - SELECT date, count(distinct users) OVER seven_days, sum(x) OVER seven_days, avg(x) OVER seven_days FROM daily_uniques **WINDOW seven_days AS (ORDER BY date ASC ROWS 6 PRECEDING);**

估值计算

- 求UV（唯一值）
- 求UV增量（唯一值增量）
- HLL估值插件
- https://github.com/digoal/blog/bl

毫秒级

日UV

select count(distinct uid) from t where dt='2017-11-11';

select # hll_uid from t where dt='2017-11-11';

滑动分析：最近N天UV

SELECT date, #hll_union_agg(users) OVER seven_days

FROM daily_uniques WINDOW seven_days AS (ORDER BY date ASC ROWS 6 PRECEDING);

每日流失UV

SELECT date, (#hll_union_agg(users) OVER two_days) - #users AS lost_uniques

FROM daily_uniques WINDOW two_days AS (ORDER BY date ASC ROWS 1 PRECEDING);

Function	Operator	Example
`hll_add`	`||`	`hll_add(users, hll_hash_integer(123))` or `users || hll_hash_integer(123)` or `hll_hash_integer(123) || users`
`hll_cardinality`	`#`	`hll_cardinality(users)` or `#users`
`hll_union`	`||`	`hll_union(male_users, female_users)` or `male_users || female_users` or `female_users || male_users`

案例4-流式阅后即焚

- 流式处理，高并发写入，快速消费处理。
- 处理后的数据被删除。
- 要求：
  - 数据快速写入
  - 数据写入后必须持久化
  - 快速消费被写入的记录（例如订阅，或者用于业务上的流式计算，计算结果保留）
  - 消费和计算必须在一个事务完成
  - https://github.com/digoal/blog/blob/master/201711/20171107_32.md

可靠、事务、原子、
低延迟、大吞吐

案例4-实时监控

- 流式处理应用

https://github.com/digoal/blog/blob/master/201711/20171123_02.md

案例4-流式阅后即焚

- 流式处理应用
- create table tbl (id int, info jsonb);
- insert into tbl ....;

具备
一致性、原子性。

- 异步处理
- **UDF写法，UDF内实现阅后即焚**
- **CTE写法，单个SQL实现阅后即焚**
  - **with t1 as (delete from tbl where ctid = any (array(select ctid from tbl limit 10)) returning *)**
  - **select pg_notify('channel_name', values) from t1;**
  - **-- deal with t1's values;**

案例5-实时画像

- 经营分析系统、决策系统
  - https://github.com/digoal/blog/blob/master/201711/20171126_01.md

案例-架构设计、代码、实操手册

- https://github.com/digoal/blog/blob/master/201711/20171111_01.md
- https://github.com/digoal/blog/blob/master/201711/20171107_28.md
- https://github.com/digoal/blog/blob/master/201711/20171107_32.md
- https://github.com/digoal/blog/blob/master/201711/20171107_33.md
- https://github.com/digoal/blog/blob/master/201608/20160827_01.md
- https://github.com/digoal/blog/blob/master/201711/20171123_02.md
- https://github.com/digoal/blog/blob/master/201711/20171126_01.md

- pipelinedb集群
- https://github.com/digoal/blog/blob/master/201803/20180314_04.md

PipelineDB 集群架构

创建一个CV，设置创建2个SHARD

为两个shard映射pp node。
规则：
一个node只能为一个cv提供一个shard。
因此任何一个cv的shard数
不可能大于node个数

PipelineDB 集群架构 - CV路由策略(读优化)

中间聚合结果，按group分发到不同的shard

PipelineDB 集群架构 - CV路由策略(写优化)

CV的中间聚合结果，落本地。
除非该CV未分配本地node作为shard，则分发到其他节点上的shard。

PipelineDB 集群架构 - CV shard合并(写优化)

PipelineDB 集群架构 - HA，LB

例子 CV1:
num_shards=3，三份shard
num_replicas=3，三个副本

写高可用取决于shard数，坏num_shards-1=2个NODE，不影响写。
读高可用取决于副本数，同一个shard，坏num_replicas=3个shard，不影响读。
读高可用，需设置pipeline_cluster.primary_only=FALSE.
num_replicas最大可设置为nodes数减1。

replica采取单向逻辑复制

Case12 (秒杀)

- 超轻锁 (advisory LOCK) 解决高并发锁竞争问题
  - 手段： 在CPU运算发现行锁之前就知道是不是有冲突，大大缩短CPU计算资源，等待资源

传统 - 行锁弊端
1. 无效等待多
2. 无效等待用户长时间占用会话资源
3. 发现锁冲突的代码路径长需要进行大量CPU运算

ADLock代替行锁 - 秒杀

- 高并发扣减库存
- 高并发争抢锁
  - update tbl set x=x where id=? and pg_try_advisory_xact_lock(id) returning *;

单条记录被并发更新，吞吐23万qps。

1. 连接redis判断是否还有库存
2. 有，去PG扣减(ADLock)。没有则直接返回。
3. 扣减成功，去redis更新库存

案例-架构设计、代码、实操手册

- https://github.com/digoal/blog/blob/master/201711/20171107_31.md
- https://github.com/digoal/blog/blob/master/201611/20161117_01.md
- https://github.com/digoal/blog/blob/master/201509/20150914_01.md

Case13 共享xxx(多表JOIN实时分析)

- ltree 树类型、消除 JOIN
  - (空间换时间、物化视图)

ltree 树类型 + R-T索引

- ltree
  - https://www.postgresql.org/docs/10/static/ltree.html
  - SELECT path FROM test WHERE path ~ '*.Astronomy.*';
  - SELECT path FROM test WHERE path ~ '*.!pictures@.*.Astronomy.*';
  - SELECT path FROM test WHERE path @ 'Astro*% & !pictures@';
  - SELECT path FROM test WHERE path @ 'Astro* & !pictures@';

GiST
R-Tree索引

```
                    Top
                  /  |  \
         Science Hobbies Collections
            /       |              \
     Astronomy   Amateurs_Astronomy Pictures
        /  \                          |
Astrophysics  Cosmology             Astronomy
                                     /  |   \
                              Galaxies Stars Astronauts
```

```
ltreetest=> SELECT path FROM test WHERE path <@ 'Top.Science';
                path
-----------------------------------
 Top.Science
 Top.Science.Astronomy
 Top.Science.Astronomy.Astrophysics
 Top.Science.Astronomy.Cosmology
(4 rows)
```

Case 共享xxx(多表JOIN实时分析)

- ltree 树类型、消除 JOIN
  - (空间换时间、物化视图)

前后对比：

方案	用例	响应时间
PostgreSQL 10 方案1	全量透视	77 毫秒
PostgreSQL 10 方案1	类目 TOP	446 毫秒
PostgreSQL 10 方案1	我的总销量（包括所有下属）	464 毫秒
PostgreSQL 10 方案1	我的直接下属，TOP	2.6 秒
PostgreSQL 10 方案1	我的所有下属(递归)，TOP	642 毫秒
-	-	-
PostgreSQL 10 方案2	全量透视	74 毫秒
PostgreSQL 10 方案2	类目 TOP	41 毫秒
PostgreSQL 10 方案2	我的总销量（包括所有下属）	41 毫秒
PostgreSQL 10 方案2	我的直接下属，TOP	41 毫秒
PostgreSQL 10 方案2	我的所有下属(递归)，TOP	41 毫秒

物化、树化
分析毫秒级。
流式实时补齐，
并行计算，空间
换时间。

其他表，JOIN补齐

员工表树结构
物化补齐

案例-架构设计、代码、实操手册

- https://github.com/digoal/blog/blob/master/201709/20170923_01.md

Case14 实时推荐(ADhoc+SORT)

- 任意维度组合查询
- 求某字段(分值)TOP-K
- https://github.com/digoal/blog/blob/master/201804/20180424_04.md
- 技术
  - 多字段抽象为数组
  - RUM索引
  - 分段RUM索引
  - UDF，分段查询，取到TOP-K为止

PostgreSQL 画像

- HTAP(OLTP、OLAP)
- OLAP
  - GPU、FPGA加速
  - 多核并行计算
  - 复杂场景SQL，处理能力强悍
- OLTP
  - SQL标准覆盖全面
  - 索引接口丰富（内置已达8种）
- Oracle兼容
- 全栈数据库
  - 插件化
  - 开放自定义接口（函数、类型、操作符、索引、存储引擎、。。。）
  - NoSQL（数组、JSON、XML、KV）
  - GIS，空间数据，路由，栅格，TOP，轨迹，多媒体，点云
  - 全文检索、模糊查询、相似查询
  - 时空、任意字段组合条件查询
  - 流计算、估值计算
  - 图式计算、搜索
  - 向量计算、索引加速搜索
    - 人脸识别、图像特征、推荐、指纹、基因匹配
  - IoT 时序数据、时序索引、加速搜索
  - pl/xxx 数据库函数式编程 (java,python,lua,perl,tcl,v8,... ...)
  - MPP (GPDB)
  - SQL on Hadoop (HAWQ)
  - sharding (citus)
  - 可靠性 - 同步复制(quorum based 多副本)

产品改进

- 内核改进点
  - 事务号32位，改成64或引入其他tuple BIT向下兼容
    - https://postgrespro.com/products/postgrespro/enterprise
  - 旧版本与新版本在同堆
    - https://github.com/EnterpriseDB/zheap
- 功能增强点
  - https://github.com/digoal/blog/blob/master/201710/20171017_01.md
  - 列存、内存表 ...

HDB PG应用案例

- 双十一业务，实时复杂 OLAP
- 实时游戏运营，JSON破解FEED难题
- 实时画像， roaring bitmap化繁为简
- 新零售，时空数据实时分析
- 数据大批量处理+输出，OSS外表接口承载大流量
- 广告分析，madlib缩短开发周期

Case1 准实时订单分析系统(双十一业务)

案例

电商准实时OLAP

30TB+数据；
单表100亿级；
5张表JOIN，秒级响应。

Case1 实时订单分析系统(双十一业务)

同时兼具OLTP+OLAP：
1、大吞吐实时写入
2、海量实时分析（MPP架构）
3、行列混合存储
4、任意JOIN、Group by、Distinct
5、GIS数据管理
6、JSON
7、HLL估值
8、全文检索
9、多值(array,k-v)类型
10、游标
11、传统数据库ACID标准
12、MADlib机器学习

案例-架构设计、代码、实操手册

- 案例
- https://github.com/digoal/blog/blob/master/201707/20170728_01.md
- https://github.com/digoal/blog/blob/master/201711/20171111_01.md
- OSS外部表
  - RDS PG OSS 外部表文档：https://help.aliyun.com/knowledge_detail/43352.html
  - HDB PG OSS 外部表文档：https://help.aliyun.com/document_detail/35457.html

Case2 阿里游戏 - 单款游戏日增量亿+级 动态分析

- 需求：
- JSON内字段检索、索引、统计
- UDF - JSON内容分裂

实时分析，**举例(不限于此)**：
指定维度、日UV，新增UV，。。。透视

```
select x,count(distinct y) from tbl where dt=? group by x;
```

案例-架构设计、代码、实操手册

- https://www.postgresql.org/docs/10/static/datatype-json.html
- https://www.postgresql.org/docs/10/static/functions-json.html
- RDS PG OSS 外部表文档：https://help.aliyun.com/knowledge_detail/43352.html
- HDB PG OSS 外部表文档：https://help.aliyun.com/document_detail/35457.html

Case3(画像、特征、透视)

- 画像系统

背景

- 业务背景： ToB 实时圈人系统
- 数据来源：实时标签数据
- 数据规模：单表10亿条记录，单个B-1亿用户，1万个标签字段。
- 数据描述：每个用户的标签数据
- 查询需求：任意标签组合圈人
  - 100毫秒级响应
- 并发需求：200+
- DML需求：实时标签分钟级体现到查询中

痛点

- 1万个TAG，大宽表。
  - 目前没有数据库支持。需要拆分成多表。
- 原方案成本高，收益低。
  - 8台，数据延迟**天级**别，响应时间接近**分钟级**，并发不到100。

案例

- 数据银行项目
- 20亿+用户，万级标签，大屏展示（100+标签组合圈选透视）
  - 1、求COUNT，2000亿（20亿用户，100个标签组合）USER_IDS，响应速度**2.6秒**。
  - 2、求USERID明细，返回500万用户ID位置，**692毫秒**。
  - 3、求USERID明细，返回BITMAP，500万个BIT，**224毫秒**。

roaring bitmap 插件

- 提供：bitmap新增元素、删除元素接口。BITMAP计算接口。聚合接口。
- bitand，同时包含
- bitor，包含任意
- bitand bitxor，包含1但是不包含2

案例-架构设计、代码、实操手册

- roaringbitmap插件使用
  - https://github.com/digoal/blog/blob/master/201801/20180127_01.md

Case4 (时空分析)

- GIS与新零售

案例

- 新零售 - LBS数据应用, 网格化运营
  - 业务背景：LBS透视、圈人
  - 数据来源：ODPS
  - 数据规模：千亿+
  - 数据描述：商铺位置、用户轨迹数据，保留3个月。
  - 查询需求：
    - 选址，分析某个商圈的对象透视，秒级
    - 商铺地推，商圈周边的潜在目标人群，秒级
    - 时间区间、空间覆盖查询，秒级
  - 并发需求：100+
  - DML需求：OSS批量写入
- 黄金策
  - 精准筛选（任意字段、标签过滤、透视分析），高效分析

痛点

- 数据量较大
- 时间、空间、对象属性多维透视
- 有空间需求
- 透视实时响应
- 存储乱序、IO放大

云产品方案、效果

案例

LBS（GIS）新零售OLAP

千亿级空间数据
秒级响应

用户
- 时间
- 位置
- 标签

地标
- 路网
- 多边形

HDB、RDS PG

PostGIS
Spatial PostgreSQL

Geometry, Raster, Top, ...

多边形+时间
圈选人群
人群透视

020
会员圈选、透视

多边形+时间
圈选人群
人群透视

选址
圈选、透视

案例-架构设计、代码、实操手册

- https://github.com/digoal/blog/blob/master/201706/20170629_01.md
- https://github.com/digoal/blog/blob/master/201709/20170918_02.md
- https://github.com/digoal/blog/blob/master/201708/20170820_02.md
- https://github.com/digoal/blog/blob/master/201708/20170820_01.md
- https://github.com/digoal/blog/blob/master/201707/20170722_01.md
- https://github.com/digoal/blog/blob/master/201703/20170328_04.md

Case5 批量数据圈选与导出

- CDN，数据归集
- 某物流公司，数据归集
- 品牌圈人，导出
- 速度
  - ~ 30MB/s/Segment

海量数据实时归集、导出。prefix区分归集.

OSS 对象存储

Case5 批量数据圈选与导出

- DEMO
- 详见阿里云官网, HDB PG产品手册
  - create table tbl_output_struct(prefix text, struct json);
  - insert into tbl_output_struct values ('label1', '{col:type, col2:type, ...}');
  - create writable table tbl_output (prefix text, content json) bucket ....;
  - insert into tbl_output select 'label1' as prefix, row_to_json(t) (select ... from tblxxx where .....) as t;

案例-架构设计、代码、实操手册

- https://github.com/digoal/blog/blob/master/201801/20180109_01.md

Case 6 优土-智能广告

*任务名称

终端

业务

*资源位

*选择素材

入口图片

投放维度

*内容分类

优土-智能广告

投放日期 ： 2015-10-02 ~ 2015-10-10

投放时间 ： 全天候投放 指定时间

全量独占资源位

本次投放级别

曝光目标设置

曝光目标人群

曝光频次控制

优土-智能广告

用户曝光次数小时表

用户曝光次数天表

用户曝光次数周表

素材ID	素材名称	素材类型	所属项目名称	总曝光量	总曝光UV	曝光UV占比	总点击量	总点击UV	总点击UV占比
1	前贴视频	投放任务[121]	七月安生项目[32]						

资源位id	资源位名称	所属投放任务	所属投放项目	曝光	曝光UV	曝光UV占比	点击	点击UV
123	七月花絮	七月预热任务	七月投放项目			曝光UV/曝光 x 100%		
456	七月预告	七月预热任务	七月投放项目					

项目名称 ：七月与安生预热投放项目

项目描述 ：七月与安生预热的项目描述

项目总曝光量

821,846

日曝光量 12,423

任务id	投放任务名称	资源位id	资源位名称	总曝光量	总曝光UV	曝光UV占比	总点击量	总点击UV	总点击UV占比
1	七月安生预热	123	前贴视频						

素材id	素材名称	所属投放任务	曝光	曝光UV	曝光UV占比	点击	点击UV	点击UV占比
123	七月花絮	七月预热投放			曝光UV/曝光 x 100%			点击UV/点击 x 100%
456	七月预告	七月预热投放						

2	七月安生预热	123	前贴视频

优土-智能广告

- **RDS PostgreSQL 角色**
  - FEED表
    - 命中规则，写入FEED。(100万+ 行/s)
  - 计数表
    - 分区：小时、天、周
    - 实时写入、阅后即焚(300万行+/s)，合并到计数表
  - 实时统计表
    - 阅后即焚(300万行+/s) 、合并到实时统计表，提供毫秒级任意维度查询
  - OSS存储
    - 计数表，上一小时、天、周，调度写入OSS。200MB/s 并行读写速度。
- **HDB PostgreSQL 角色**
  - 对接OSS(30MB/s/segment节点读写速度)、实时分析。无限扩容。
- https://github.com/digoal/blog/blob/master/201711/20171126_01.md

优土-智能广告

RDS PostgreSQL
实时写入明细
百万+ 行/s
在线
APP
FEED表
50亿/天
规则判定
0.0n毫秒级
任意维度毫秒级响应
阅后即焚
300万行/s
计数
表
+
规则
表
统计表
HybridDB PostgreSQL
分析
APP
历史表
外部表存储，暖热分级存储
外部表存储，暖热分级存储
OSS 海量存储（暖数据）

优土-智能广告(衍生需求)

- MADLib库(支持几百个机器学习库函数、对应**各种数学模型**)，PL/R，PL/Python
- 例子
  - p元线性回归
  - y1=b0+b1x11+b2x12+…+bpx1p+ε1
  - y2=b0+b1x21+b2x22+…+bpx2p+ε2
  - ……
  - 求截距，斜率。
  - 预测yn
  - yn=b0+b1xn1+b2xn2+…+bpxnp+εn

```
linregr_train( source_table,
               out_table,
               dependent_varname,
               independent_varname,
               grouping_cols,
               heteroskedasticity_option
             )
```

优土-智能广告(衍生需求)

- 一条SQL搞定聚类分析
- SELECT kmeans(ARRAY[columns,...], K) OVER (...), * FROM samples;

优土-智能广告(衍生需求)

- SQL接口机器学习库MADLib (支持几百个机器学习库函数、对应**各种数学模型**)，PL/R，PL/Python

案例-架构设计、代码、实操手册

- MADlib机器学习库
  - https://github.com/digoal/blog/blob/master/201511/20151111_01.md
  - http://madlib.apache.org/
  - https://cran.r-project.org/web/packages/PivotalR/vignettes/pivotalr.pdf
  - https://pypi.python.org/pypi/pymadlib/0.1.4

案例小结

行业

- 物联网、电商、生物、游戏
- 企业CRM、传统行业、ZF
- 物流、音视频、BI、证券、金融
- 手机、

技术点

- 搜索（ADHoc, 全文检索、模糊、数组）
- 相似搜索（图片、文本、数组）
- 用户画像、空间、时空、挖掘
- 图式搜索、流计算、秒杀、树结构、非结构化
- 单元化部署、sharding

目录

- 产品介绍
- 生态介绍
- 应用案例
- 开发、管理实践
- 数据库原理
- 参考文档

开发、管理实践

- RDS PostgreSQL\PPAS
- HybridDB for PostgreSQL

开发、管理实践

- RDS PPAS
- RDS PostgreSQL
- HybridDB for PostgreSQL

PPAS

- SQL防火墙
  - https://github.com/digoal/blog/blob/master/201801/20180116_02.md
- CPU\IO 资源隔离
  - https://github.com/digoal/blog/blob/master/201801/20180113_01.md
- 索引推荐
  - https://github.com/digoal/blog/blob/master/201801/20180113_02.md
- AWR
  - https://github.com/digoal/blog/blob/master/201606/20160628_01.md
- VPD(RLS)
  - https://www.enterprisedb.com/docs/en/10.0/EPAS_Guide_v10/EDB_Postgres_Advanced_Server_Guide.1.52.html#pID0E04QC0HA

PPAS AWR

- 全面的系统报告
- edbreport(beginning_id, ending_id)

- 数据库报告
- stat_db_rpt(beginning_id, ending_id)

- 指定范围的表级报告
- stat_tables_rpt(beginning_id, ending_id, top_n, scope)
- scope=ALL, USER, SYS

- 指定范围的表级IO报告
- statio_tables_rpt(beginning_id, ending_id, top_n, scope)

- 指定范围的索引级报告
- stat_indexes_rpt(beginning_id, ending_id, top_n, scope)

- 指定范围的索引级IO报告
- statio_indexes_rpt(beginning_id, ending_id, top_n, scope)

阿里云RDS PPAS AWR API

- 数据库报告
- sys.rds_stat_db_rpt(beginning_id, ending_id)

- 指定范围的表级报告
- sys.rds_stat_tables_rpt(beginning_id, ending_id, top_n, scope)
- scope=ALL, USER, SYS

- 指定范围的表级IO报告
- sys.rds_statio_tables_rpt(beginning_id, ending_id, top_n, scope)

- 指定范围的索引级报告
- sys.rds_stat_indexes_rpt(beginning_id, ending_id, top_n, scope)

- 指定范围的索引级IO报告
- sys.rds_statio_indexes_rpt(beginning_id, ending_id, top_n, scope)

阿里云RDS PPAS AWR API

- 全面的系统报告
- sys.rds_report(beginsnap bigint, endsnap bigint)

- 删除指定范围快照
- sys.rds_purgesnap(beginsnap int, endsnap int)

- 删除所有快照
- sys.rds_truncsnap()

- 查看已有快照
- sys.rds_get_snaps()

- 创建快照
- sys.rds_snap()

实例版本管理与插件管理

- 查看当前实例版本
  - select version();
- 升级实例版本
  - 控制台，重启实例。
- 列出支持哪些插件
  - select pg_get_functiondef('sys.rds_manage_extension'::regproc);
- 创建插件
  - select rds_manage_extension('create','dblink');
- 升级插件版本
  - select rds_manage_extension('update','dblink');
- 删除插件
  - select rds_manage_extension('drop','dblink');

会话管理

- 查看当前会话状态, pid为会话ID
  - select * from sys.rds_pg_stat_activity()
- KILL 会话
  - select sys.rds_pg_terminate_backend(upid int)
- KILL QUERY
  - select sys.rds_pg_cancel_backend(upid int)

TOP SQL查询

- TOP SQL查询接口
  - https://github.com/digoal/blog/blob/master/201704/20170424_06.md
  - sys.rds_pg_stat_statements()
- 重置SQL统计信息
  - select sys.rds_pg_stat_statements_reset()

重置统计信息计数

- select sys.rds_pg_stat_reset()

设置所有用户的用户级参数

- 查看有哪些参数
  - select * from pg_settings;
- 查看参数当前设置
  - select current_setting('参数名');
- 设置所有用户的用户级参数
  - select sys.rds_set_conf_for_all_roles(param text, val text)

开发、管理实践

- RDS PPAS
- RDS PostgreSQL
- HybridDB for PostgreSQL

实例版本管理与插件管理

- 查看当前实例版本
  - select version();
  - show rds_release_date;
- 升级实例版本
  - 控制台，重启实例。
- 列出支持哪些插件
  - show rds_internal_allowed_extensions;
  - show rds_available_extensions;
- 创建插件
  - create extension 插件名;
- 升级插件版本
  - alter extension 插件名 UPDATE;
- 删除插件
  - drop extension 插件名;

审计

- 审计开关
  - #log_statement = 'none' # none, ddl, mod, all
- DDL审计+NOTIFY
  - https://github.com/digoal/blog/blob/master/201709/20170925_02.md
  - https://github.com/digoal/blog/blob/master/201412/20141211_01.md
- 关键TABLE的DML审计
  - https://github.com/digoal/blog/blob/master/201206/20120625_01.md
  - https://github.com/digoal/blog/blob/master/201408/20140828_01.md

SQL高级用法

- https://github.com/digoal/blog/blob/master/201802/20180226_05.md
  - CTE(递归)
  - LATERAL
  - ORDINALITY (SRF)
  - WINDOW
  - SKIP LOCKED
  - DISTINCT ON
  - GROUPING SETS , CUBE , ROLLUP

物化视图

- https://www.postgresql.org/docs/devel/static/sql-creatematerializedview.html
- 预计算，支持索引。
- CREATE MATERIALIZED VIEW [IF NOT EXISTS] table_name
- [(column_name [, ...])]
- [WITH (storage_parameter [= value] [, ...])]
- [TABLESPACE tablespace_name]
- AS query
- [WITH [NO] DATA]

- 刷新物化视图
- REFRESH MATERIALIZED VIEW [CONCURRENTLY] name
- [WITH [NO] DATA]

分页

- 每一页都丝般柔滑的方法
  - 1、使用游标
    - declare cur1 cursor for select * from table where xxx order by xx;
    - fetch 10 from cur1;
    - ....
  - 2、使用位点，每次取值区间以上一次的最后位点为开始点。
    - select * from table where xx>上一次最大点 and xxxx order by xx limit ?;
- https://github.com/digoal/blog/blob/master/201605/20160506_01.md
- https://github.com/digoal/blog/blob/master/201509/20150919_02.md
- https://github.com/digoal/blog/blob/master/201402/20140211_01.md
- https://github.com/digoal/blog/blob/master/201206/20120620_01.md
- https://github.com/digoal/blog/blob/master/201102/20110216_02.md

频繁更新的TABLE优化

- 索引较多、并且：
  - 频繁insert on conflict\更新、插入+删除

- 设置fillfactor，尽量走HOT，减少IO放大。
- 使用分区表，降低索引页分裂时的锁冲突带来的性能影响。提高vacuum并发性。
  - https://github.com/digoal/blog/blob/master/201803/20180301_01.md
- 对于gin索引，设置足够的pending list size。
  - https://www.postgresql.org/docs/devel/static/gin-tips.html
- 同时需要注意freeze风暴。
  - https://github.com/digoal/blog/blob/master/201801/20180117_03.md
  - https://github.com/digoal/blog/blob/master/201606/20160612_01.md

实时数据清洗、转换

- https://github.com/digoal/blog/blob/master/201706/20170619_02.md
- rule
- 创建来源表结构
  - postgres=# create table nt(id int, **c1 numeric, c2 numeric**);
  - CREATE TABLE
- 创建目标表结构
  - postgres=# create table nt_geo (id int, **geo geometry**);
  - CREATE TABLE
- 对来源表创建规则或触发器，例如
  - postgres=# create rule r1 as on insert to nt do instead insert into nt_geo values (NEW.id, ST_MakePoint(NEW.c1,NEW.c2));
  - CREATE RULE
- 使用来源数据结构，将数据插入来源数据表
  - postgres=# insert into nt values (1,1,1);
  - INSERT 0 1

实时数据清洗、转换

- rule
- 源表，JSONB非结构化
  - postgres=# create table t1 (id int, info text, j jsonb);
- 目标表，结构化
  - postgres=# create table t2 (id int, info text, c1 int, c2 int, c3 text);
- 在源表创建规则，自动将JSONB非结构化数据，转换为结构化数据插入
  - postgres=# create rule r1 as on insert to t1 do instead insert into t2 values (NEW.ID, NEW.INFO, ((NEW.J)->>'c1')::int, ((NEW.j)->>'c2')::int, (NEW.j)->>'c3');
  - postgres=# insert into t1 values (1,'test',**jsonb '{"c1":1, "c2":2, "c3":"text"}'**);
  - postgres=# select * from t1;
  - (0 rows)
  - postgres=# select * from t2;
  - id | info | c1 | c2 | c3
  - ----+------+----+----+------
  - 1 | test | 1 | 2 | text
  - (1 row)

数据采样

- 使用采样算法
  - 行级随机采样(BERNOULLI(百分比))
    - select * from test TABLESAMPLE bernoulli (1);
  - 块级随机采样(SYSTEM(百分比))
    - select * from test TABLESAMPLE system (1);
- https://github.com/digoal/blog/blob/master/201706/20170602_02.md
  - 采样应用：估值计算、统计信息、测试环境
- https://github.com/digoal/blog/blob/master/201709/20170911_02.md

数据加密

- pgcrypto
  - https://www.postgresql.org/docs/10/static/pgcrypto.html
- 加密后的查询加速（等值查询）
- 加密
  - 对称、非对称、混淆（使用非对称加密需要交换的对称加密密钥，然后使用对称加密）加密
  - https://github.com/digoal/blog/blob/master/201802/20180226_01.md

字段加密

- digoal=# create extension **pgcrypto;**
- 可逆加密
  - digoal=# insert into userpwd (userid,pwd) values (1, crypt('this is a pwd source', gen_salt('bf',10)));
  - digoal=# create table userpwd(userid int8 primary key, pwd text);
  - CREATE TABLE
- 不可逆加密
  - https://github.com/digoal/blog/blob/master/201607/20160727_02.md
  - https://github.com/digoal/blog/blob/master/201711/20171127_02.md

数据脱敏

- https://github.com/digoal/blog/blob/master/201706/20170602_02.md

约束种类与用法

- 唯一, unique
- 非空, not null
- check, check(exp);
- 外键,
- 排他, (例如, 空间不相交, 地图应用, 范围不相交, 边界限制。)
  - https://github.com/digoal/blog/blob/master/201712/20171223_02.md
  - CREATE TABLE reservation
  - (during tsrange,
  - EXCLUDE USING GIST (during WITH &&)
  -);

  - CREATE EXTENSION btree_gist;
  - CREATE TABLE room_reservation
  - (room text,
  - during tsrange,
  - EXCLUDE USING GIST (room WITH =, during WITH &&)
  -);

数据去重大法

- https://github.com/digoal/blog/blob/master/201706/20170602_01.md
  - 单列去重
  - 多列去重
  - 行去重
  - 多列混合去重
- 窗口、行号、= any(array())、数组

模糊查询

- 单、双字搜索
  - postgres=# create or replace function split_12(text) returns text[] as $$
  - declare
  - res text[];
  - begin
  - select regexp_split_to_array($1, '') into res;
  - for i in 1..length($1)-1 loop
  - res := array_append(res, substring($1, i, 2));
  - end loop;
  - return res;
  - end;
  - $$ language plpgsql strict immutable;
  - CREATE FUNCTION
  - postgres=# select split_12('abc你好');
  - split_12
  - ----------
  - {a,b,c,你,好,ab,bc,c你,你好}
  - (1 row)

  - create index idx2 on tbl using gin (split_12(col));
  -
  - select * from tbl where split_12(col) @> array['单字或双字'];

模糊查询

- 大于2个字
  - https://github.com/digoal/blog/blob/master/201704/20170426_01.md
  - create extension pg_trgm;
  - create index idx on tbl using gin (col gin_trgm_ops);
  - select * from tbl where col like '%xxx%';

多字段范围检索加速

- 范围查询，联合索引效果并不好。
- 使用range类型，GIST索引加速。
  - 开始时间，结束时间范围
  - 经纬度范围
  - DNS匹配（开始IP，结束IP 范围）
    - https://github.com/digoal/blog/blob/master/201206/20120607_01.md

RDS PG 10 HTAP (一份数据)

并行计算资源控制

- 单个并行节点并行度
  - #max_parallel_workers_per_gather = 2
- 全局并行度
  - #max_parallel_workers = 8
- 并行度算法
  - https://github.com/digoal/blog/blob/master/201610/20161002_01.md

强制设置并行度

- postgres=# set max_parallel_workers_per_gather =32;
- SET
- postgres=# set parallel_setup_cost =0;
- SET
- postgres=# set parallel_tuple_cost =0;
- SET
- postgres=# set min_parallel_table_scan_size =0;
- SET
- postgres=# set min_parallel_index_scan_size =0;
- SET
- postgres=# alter table a set (parallel_workers =32);
- ALTER TABLE
- postgres=# explain select count(*) from a;
- QUERY PLAN
- ----------
- Finalize Aggregate (cost=86811.94..86811.95 rows=1 width=8)
- -> Gather (cost=86811.85..86811.86 rows=32 width=8)
- Workers Planned: 32
- -> Partial Aggregate (cost=86811.85..86811.86 rows=1 width=8)
- -> Parallel Index Only Scan using a_pkey on a (cost=0.43..86030.60 rows=312500 width=0)
- (5 rows)

不设置强制，则按表、索引大小、成本、自动估算并行度

```
postgres=# show max_worker_processes ;
 max_worker_processes
----------------------
 128
(1 row)

postgres=# show max_parallel_workers;
 max_parallel_workers
----------------------
 128
(1 row)
```

批量DML

- https://github.com/digoal/blog/blob/master/201704/20170424_05.md
- 批量插入
  - insert into tbl values (),(),...();
  - copy
- 批量更新
  - update tbl from tmp set x=tmp.x where tbl.id=tmp.id;
- 批量删除
  - delete from tbl using tmp where tmp.id=tbl.id;

索引接口的选择

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] name ] ON [ ONLY ] table_name [ USING method ]
    ( { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
    [ INCLUDE ( column_name [, ...] ) ]
    [ WITH ( storage_parameter = value [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ]
```

select * from pg_am;
https://www.postgresql.org/docs/devel/static/sql-createindex.html
https://github.com/digoal/blog/blob/master/201706/20170627_01.md
自动选择索引接口：
https://github.com/digoal/blog/blob/master/201706/20170617_01.md

索引接口的选择

- BTree
  - =, >, >=, <, <=、排序
- Hash (KEY值为HASH值，可以适合很长的字符串)
  - =
- GIN
  - 多值类型(数组、全文检索、枚举、网络地址类型)：包含、相交
  - JSON类型：包含 https://www.postgresql.org/docs/devel/static/datatype-json.html#JSON-INDEXING
  - 普通类型(通过btree_gin 插件支持)：与B-Tree类似
  - 字符串(通过pg_trgm 插件支持)：模糊查询、相似查询
  - 多列：任意列组合查询
- GiST
  - 空间类型：方位（上、下、左、右），空间关系（相交、包含），空间距离排序(KNN)
  - 范围数据： =, &&, <@, @>, <<, >>, -|-, &<, and &>
  - 普通类型(通过btree_gist 插件支持)：与B-Tree类似，增加空间类型类似操作符
  - 数组类型(通过intarray 插件支持)：与GIN类似
  - 多列：任意列组合查询
- SPGiST
  - 平面几何类型：与GiST类似
  - 范围类型：与GiST类似

索引接口的选择

- RUM
  - 多值类型(数组、全文检索类型)：包含、相交、相似排序
  - 普通类型：与B-Tree类似
- BRIN
  - 适合 线性数据、时序数据、（HEAP PAGE之间边界清晰的数据）。
  - 普通类型：与B-Tree类似
  - 空间类型：包含
- Bloom
  - 多列：任意列组合，等值查询
- 表达式索引
  - 搜索条件为表达式时。
  - where abs(a+b) = ?
  - create index idx on tbl ((abs(a+b)));
- 条件索引(定向索引)
  - 搜索时，强制过滤某些条件时。
  - where status='active' and col=? 。
  - create index idx on tbl (col) where status='active';

多列复合索引字段顺序原则

- https://github.com/digoal/blog/blob/master/201803/20180314_02.md

非驱动列查询的优化

- https://github.com/digoal/blog/blob/master/201803/20180323_03.md

ADHoc查询

- 单值字段、空间字段、时间字段、多值字段 任意组合搜索
  - https://github.com/digoal/blog/blob/master/201802/20180207_02.md
  - https://github.com/digoal/blog/blob/master/201802/20180208_01.md
- 单索引复合顺序选择
  - 驱动列优先选择等值条件列
- 任意字段组合扫描需求，不适合复合索引
  - 多个b-tree索引支持bitmap scan
    - https://github.com/digoal/blog/blob/master/201702/20170221_02.md
  - GIN
  - bloom
  - rum
    - https://github.com/digoal/blog/blob/master/201802/20180228_01.md

函数稳定性

- https://github.com/digoal/blog/blob/master/201212/20121226_01.md

- 稳定性
  - volatile，不稳定，每次都会被触发调用。（select * from tbl where id=func(); 有多少记录，就会被触发多少次调用func(). ）
  - stable，稳定，在事务中只调用一次。
  - immutable, 超级稳定，执行计划中，直接转换为常量。
- 索引表达式
  - 必须是immutable稳定性的函数或操作符
- 使用索引
  - 必须是stable以上稳定性的函数或操作符
  - select * from tbl where a=now();
  - now(), = 都是stable以上操作符。
- 绑定变量
  - stable ，每次execute被调用。
  - immutable，prepare时转换为常量，不再被调用。

索引维护

- PostgreSQL支持重复索引（例如一个字段，可以创建多个一样的索引）
  - 在线新建索引
  - 删除旧索引

在线创建索引(不堵塞dml)

- https://www.postgresql.org/docs/devel/static/sql-createindex.html

并发创建索引
不堵塞DML的方法

- Command: CREATE INDEX
- Description: define a new index
- Syntax:
- CREATE [UNIQUE] INDEX [**CONCURRENTLY**] [[IF NOT EXISTS] name] ON table_name [USING method]
- ({ column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS { FIRST | LAST }] [, ...])
- [WITH (storage_parameter = value [, ...])]
- [TABLESPACE tablespace_name]
- [WHERE predicate]

在线加列

- 不要加默认值
- 如果有默认值，建议后台分批更新

在线回收空间

- https://github.com/reorg/pg_repack/

- https://github.com/digoal/blog/blob/master/201610/20161030_02.md

count(*) 优化

- partial index
- index only scan
- 预计算
- 流计算
- 采样估算
  - https://github.com/digoal/blog/blob/master/201804/20180403_03.md
- 统计信息估算
  - pg_class.reltuples

SQL Hint

- create extension pg_hint_plan;

- /*+
- NestLoop(ir d)
- NestLoop(ir d rd)
- Leading(((ir d) rd))
- IndexScan(rd "def")
- IndexScan(d "bcd")
- IndexScan(ir "abc")
- */
- SELECT xxxx
- FROM
- "test01" AS rd
- INNER JOIN "test02" AS d ON (rd.test02_uuid = d.uuid)
- INNER JOIN "test03" AS ir ON (d.test03_uuid = ir.uuid)
- WHERE
- d.status = 'normal'
- AND ir.u_uuid = 'tttttttt' and (d.test02_status in ('test02ed','checked'))
- and d.is_sub = false and d.is_filter = false ORDER BY d.test02_time desc limit 10 offset 0

http://pghintplan.osdn.jp/

SQL Hint

Group	Format	Description
Scan method	SeqScan(table)	Forces sequential scan on the table
	TidScan(table)	Forces TID scan on the table.
	IndexScan(table[index...])	Forces index scan on the table. Restricts to specified indexes if any.
	IndexOnlyScan(table[index...])	Forces index only scan on the table. Rstricts to specfied indexes if any. Index scan may be used if index only scan is not available. Available for PostgreSQL 9.2 and later.
	BitmapScan(table[index...])	Forces bitmap scan on the table. Restoricts to specfied indexes if any.
	NoSeqScan(table)	Forces not to do sequential scan on the table.
	NoTidScan(table)	Forces not to do TID scan on the table.
	NoIndexScan(table)	Forces not to do index scan and index only scan (For PostgreSQL 9.2 and later) on the table.
	NoIndexOnlyScan(table)	Forces not to do index only scan on the table. Available for PostgreSQL 9.2 and later.
	NoBitmapScan(table)	Forces not to do bitmap scan on the table.
Join method	NestLoop(table table[table...])	Forces nested loop for the joins consist of the specifiled tables.
	HashJoin(table table[table...])	Forces hash join for the joins consist of the specifiled tables.
	MergeJoin(table table[table...])	Forces merge join for the joins consist of the specifiled tables.
	NoNestLoop(table table[table...])	Forces not to do nested loop for the joins consist of the specifiled tables.
	NoHashJoin(table table[table...])	Forces not to do hash join for the joins consist of the specifiled tables.
	NoMergeJoin(table table[table...])	Forces not to do merge join for the joins consist of the specifiled tables.
Join order	Leading(table table[table...])	Forces join order as specified.
	Leading(<join pair>)	Forces join order and directions as specified. A join pair is a pair of tables and/or other join pairs enclosed by parentheses, which can make a nested structure.
Row number correction	Rows(table table[table...] correction)	Corrects row number of a result of the joins consist of the specfied tables. The available correction methods are absolute (#<n>), addition (+<n>), subtract (-<n>) and multiplication (*<n>). <n> should be a string that strtod() can read.
GUC	Set(GUC-param value)	Set the GUC parameter to the value while planner is running.

性能分析利器1 - TOP SQL

View "public.pg_stat_statements"

Column	Type	Collation
userid	oid	
dbid	oid	
queryid	bigint	
query	text	
calls	bigint	
total_time	double precision	
min_time	double precision	
max_time	double precision	
mean_time	double precision	
stddev_time	double precision	
rows	bigint	
shared_blks_hit	bigint	
shared_blks_read	bigint	
shared_blks_dirtied	bigint	
shared_blks_written	bigint	
local_blks_hit	bigint	
local_blks_read	bigint	
local_blks_dirtied	bigint	
local_blks_written	bigint	
temp_blks_read	bigint	
temp_blks_written	bigint	
blk_read_time	double precision	
blk_write_time	double precision	

性能分析利器1 - TOP SQL

- https://github.com/digoal/blog/blob/master/201704/20170424_06.md
- create extension pg_stat_statements;

- **最耗IO SQL**
- 单次调用最耗IO SQL TOP 5
- select userid::regrole, dbid, query from pg_stat_statements order by (blk_read_time+blk_write_time)/calls desc limit 5;
- 总最耗IO SQL TOP 5
- select userid::regrole, dbid, query from pg_stat_statements order by (blk_read_time+blk_write_time) desc limit 5;

- **最耗时 SQL**
- 单次调用最耗时 SQL TOP 5
- select userid::regrole, dbid, query from pg_stat_statements order by mean_time desc limit 5;
- 总最耗时 SQL TOP 5
- select userid::regrole, dbid, query from pg_stat_statements order by total_time desc limit 5;

- **响应时间抖动最严重 SQL**
- select userid::regrole, dbid, query from pg_stat_statements order by stddev_time desc limit 5;

- **最耗共享内存 SQL**
- select userid::regrole, dbid, query from pg_stat_statements order by (shared_blks_hit+shared_blks_dirtied) desc limit 5;

- **最耗临时空间 SQL**
- select userid::regrole, dbid, query from pg_stat_statements order by temp_blks_written desc limit 5;

性能分析利器2 - explain

- https://www.postgresql.org/docs/10/static/sql-explain.html
- https://github.com/digoal/blog/blob/master/201704/20170424_06.md
- HDB PG
- https://github.com/digoal/blog/blob/master/201712/20171204_02.md

设置需要TRACE的开关。
查看详细的输出，逐个分析。
最重要的是分析explain (analyze,verbose,timing,costs,buffers的结果)

```
EXPLAIN [ ( option [, ...] ) ] statement
EXPLAIN [ ANALYZE ] [ VERBOSE ] statement

where option can be one of:

    ANALYZE [ boolean ]
    VERBOSE [ boolean ]
    COSTS [ boolean ]
    BUFFERS [ boolean ]
    TIMING [ boolean ]
    SUMMARY [ boolean ]
    FORMAT { TEXT | XML | JSON | YAML }
```

性能分析利器2 - explain

- set client_min_messages=debug5;
- set log_checkpoints = on;
- set log_error_verbosity = verbose ;
- set log_lock_waits = on;
- set log_replication_commands = off;
- set log_temp_files = 0;
- set track_activities = on;
- set track_counts = on;
- set track_io_timing = on;
- set track_functions = 'all';
- set trace_sort=on;
- set log_statement_stats = off;
- set log_parser_stats = on;
- set log_planner_stats = on;
- set log_executor_stats = on;
- set log_autovacuum_min_duration=0;
- set deadlock_timeout = '1s';
- set debug_print_parse = off;
- set debug_print_rewritten = off;
- set debug_print_plan = off;
- set debug_pretty_print = on;
- explain (analyze,verbose,timing,costs,buffers) select count(*),relkind from pg_class group by relkind order by count(*) desc limit 1;

设置需要TRACE的开关。
查看详细的输出，逐个分析。
最重要的是分析explain (analyze,verbose,timing,costs,buffers的结果)

性能分析利器3 - perf

- perf
- https://github.com/digoal/blog/blob/master/201611/20161129_01.md
- 例子(分区过多导致的性能问题)
  - https://github.com/digoal/blog/blob/master/201801/20180124_01.md

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9482 root      20   0 1504m  52m 6508 S 46.1  0.0   3611:23 [redacted]
52630 [redacted] 20   0  120g 291m 249m S 25.6  0.1   0:08.30 postgres: [redacted] BIND
52660 [redacted] 20   0  120g 287m 248m S 25.6  0.1   0:04.39 postgres: [redacted] BIND
53155 [redacted] 20   0  120g 288m 249m S 25.6  0.1   0:04.80 postgres: [redacted] BIND
53180 [redacted] 20   0  120g 318m 249m S 25.6  0.1   0:07.54 postgres: [redacted] BIND
52369 [redacted] 20   0  120g 306m 249m S 23.9  0.1   0:04.97 postgres: [redacted] BIND
```

```
+      5.29%   postgres  postgres                    [.] AllocSetAlloc
+      5.09%   postgres  postgres                    [.] hash_search_with_hash_value
+      4.92%   postgres  postgres                    [.] SearchCatCache
+      3.70%   postgres  postgres                    [.] _bt_checkkeys
+      2.50%   postgres  postgres                    [.] pglz_decompress
+      1.87%   postgres  [kernel.kallsyms]           [k] _spin_lock_irq
+      1.86%   postgres  [kernel.kallsyms]           [k] _spin_lock_irqsave
+      1.73%   postgres  [kernel.kallsyms]           [k] _spin_lock
```

性能分析利器3 - oprofile

- oprofile
  - https://github.com/digoal/blog/blob/master/201610/20161012_01.md
  - https://github.com/digoal/blog/blob/master/201505/20150509_01.md

```
[root@digoal data06]# opreport -l -f -g -w -x -t 1 /opt/pgsql/bin/postgres
Using /data06/oprofile_data/samples/ for samples directory.
CPU: Intel Core/i7, speed 1995.14 MHz (estimated)
Counted CPU_CLK_UNHALTED events (Clock cycles when not halted) with a unit mask of 0x00 (No unit mask) count 100000
vma      samples  %        linenr info                 symbol name
007827a0 2091381  26.7572  /opt/soft_bak/postgresql-9.4.1/src/backend/utils/time/tqual.c:1116 HeapTupleSatisfiesVacuum
00490300 988600   12.6482  /opt/soft_bak/postgresql-9.4.1/src/backend/access/heap/pruneheap.c:174 heap_page_prune
0078a8c0 698665    8.9387  /opt/soft_bak/postgresql-9.4.1/src/port/qsort.c:104 pg_qsort
0058afb0 676022    8.6491  /opt/soft_bak/postgresql-9.4.1/src/backend/commands/vacuumlazy.c:1728 vac_cmp_itemptr
0058baf0 385039    4.9262  /opt/soft_bak/postgresql-9.4.1/src/backend/commands/vacuumlazy.c:172 lazy_vacuum_rel
004c4d00 365497    4.6762  /opt/soft_bak/postgresql-9.4.1/src/backend/access/transam/xlog.c:844 XLogInsert
00675420 229805    2.9401  /opt/soft_bak/postgresql-9.4.1/src/backend/storage/page/bufpage.c:415 itemoffcompare
00675d20 184668    2.3626  /opt/soft_bak/postgresql-9.4.1/src/backend/storage/page/bufpage.c:433 PageRepairFragmentation
0078a7e0 169808    2.1725  /opt/soft_bak/postgresql-9.4.1/src/port/qsort.c:78 swapfunc
00655590 147647    1.8890  /opt/soft_bak/postgresql-9.4.1/src/backend/storage/buffer/bufmgr.c:1898 BufferGetBlockNumber
00488940 139389    1.7833  /opt/soft_bak/postgresql-9.4.1/src/backend/access/heap/heapam.c:5756 heap_prepare_freeze_tuple
007624d0 86239     1.1033  /opt/soft_bak/postgresql-9.4.1/src/backend/utils/hash/dynahash.c:824 hash_search_with_hash_val
```

性能分析利器3 - dtract\stap

- stap
  - https://github.com/digoal/blog/blob/master/201310/20131017_01.md
  - https://www.postgresql.org/docs/current/static/dynamic-trace.html

性能分析利器3 - strace\pstack

- strace
- pstack

性能分析利器4 - 当前慢SQL

- 运行中慢SQL
  - select * from pg_stat_activity where now()-query_start > interval '?s';
- 长运行中事务
  - select * from pg_stat_activity where state='active' and now()-xact_start > interval '?s';
- 长空闲事务
  - select * from pg_stat_activity where state='idle in transaction' and now()-xact_start > interval '?s';
- 长2PC事务
  - select * from pg_prepared_xacts where now()-prepared > interval '?s';

性能分析利器5 -历史慢SQL

AWR

https://github.com/digoal/blog/blob/master/201604/20160421_01.md

性能分析利器5 -跟踪慢SQL为什么慢?

- auto_explain
  - https://www.postgresql.org/docs/devel/static/auto-explain.html
- 慢SQL执行计划详情
  - plan
  - node time
  - buffers, hints
  - filter

慢SQL执行计划详情

```
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=16494388.74..16494388.94 rows=10 width=12) (actual time=153006.839..153006.839 rows=0 loops=1)
   Output: id, (count(DISTINCT c1))
   Buffers: shared hit=95131 read=1041241, temp read=220164 written=220195
   ->  GroupAggregate  (cost=16494388.74..18494388.72 rows=99999999 width=12) (actual time=153006.838..153006.838 rows=0 loops=1)
         Output: id, count(DISTINCT c1)
         Group Key: t1.id
         Filter: (count(*) > 1)
         Rows Removed by Filter: 100000000
         Buffers: shared hit=95131 read=1041241, temp read=220164 written=220195
         ->  Sort  (cost=16494388.74..16744388.74 rows=99999999 width=8) (actual time=47686.082..62240.952 rows=100000000 loops=1)
               Output: id, c1
               Sort Key: t1.id
               Sort Method: external merge  Disk: 1761312kB
               Buffers: shared hit=95123 read=1041241, temp read=220164 written=220195
               ->  Seq Scan on public.t1  (cost=0.00..2386364.00 rows=99999999 width=8) (actual time=0.011..26439.157 rows=100000000 loops=1)
                     Output: id, c1
                     Filter: (t1.c2 <> 'abc'::text)
                     Buffers: shared hit=95123 read=1041241
 Planning time: 0.092 ms
 Execution time: 153357.601 ms
(20 rows)
```

性能分析利器6 - 跟踪活动日志

- log_autovacuum_min_duration = 0
  - vacuum,analyze活动，消耗
- log_checkpoints = on
  - 跟踪检查点开销
- log_lock_waits=on
  - 跟踪锁等待
- lock_timeout=1s
  - 锁等待时间阈值
- track_io_timing = on
  - 跟踪IO耗时
- track_counts = on
  - 计数器, pg_stat, pg_statio依赖

性能分析利器7 - 查看当前锁等待

- 代码： https://github.com/digoal/blog/blob/master/201705/20170521_01.md

```
with
t_wait as
(
  select a.mode,a.locktype,a.database,a.relation,a.page,a.tuple,a.classid,a.granted,
  a.objid,a.objsubid,a.pid,a.virtualtransaction,a.virtualxid,a.transactionid,a.fastpath,
  b.state,b.query,b.xact_start,b.query_start,b.usename,b.datname,b.client_addr,b.client_port,b.application_name
    from pg_locks a,pg_stat_activity b where a.pid=b.pid and not a.granted
),
t_run as
(
  select a.mode,a.locktype,a.database,a.relation,a.page,a.tuple,a.classid,a.granted,
  a.objid,a.objsubid,a.pid,a.virtualtransaction,a.virtualxid,a.transactionid,a.fastpath,
  b.state,b.query,b.xact_start,b.query_start,b.usename,b.datname,b.client_addr,b.client_port,b.application_name
    from pg_locks a,pg_stat_activity b where a.pid=b.pid and a.granted
),
t_overlap as
(
  select r.* from t_wait w join t_run r on
  (
    r.locktype is not distinct from w.locktype and
    r.database is not distinct from w.database and
    r.relation is not distinct from w.relation and
    r.page is not distinct from w.page and
    r.tuple is not distinct from w.tuple and
    r.virtualxid is not distinct from w.virtualxid and
    r.transactionid is not distinct from w.transactionid and
    r.classid is not distinct from w.classid and
    r.objid is not distinct from w.objid and
    r.objsubid is not distinct from w.objsubid and
    r.pid <> w.pid
  )
),
```

性能分析利器7 - 查看当前锁等待

```
t_unionall as
(
  select r.* from t_overlap r
  union all
  select w.* from t_wait w
)
select locktype,datname,relation::regclass,page,tuple,virtualxid,transactionid::text,classid::regclass,objid,objsubid,
string_agg(
'Pid: '||case when pid is null then 'NULL' else pid::text end||chr(10)||
'Lock_Granted: '||case when granted is null then 'NULL' else granted::text end||' , Mode: '||case when mode is null then
'Username: '||case when usename is null then 'NULL' else usename::text end||' , Database: '||case when datname is null th
'Xact_Start: '||case when xact_start is null then 'NULL' else xact_start::text end||' , Query_Start: '||case when query_s
'SQL (Current SQL in Transaction): '||chr(10)||
case when query is null then 'NULL' else query::text end,
chr(10)||'--------'||chr(10)
order by
  (  case mode
    when 'INVALID' then 0
    when 'AccessShareLock' then 1
    when 'RowShareLock' then 2
    when 'RowExclusiveLock' then 3
    when 'ShareUpdateExclusiveLock' then 4
    when 'ShareLock' then 5
    when 'ShareRowExclusiveLock' then 6
    when 'ExclusiveLock' then 7
    when 'AccessExclusiveLock' then 8
    else 0
  end  ) desc,
  (case when granted then 0 else 1 end)
```

性能分析利器7 - 查看当前锁等待

```
) as lock_conflict
from t_unionall
group by
locktype,datname,relation,page,tuple,virtualxid,transactionid::text,classid,objid,objsubid ;
```

性能分析利器7 - 查看当前锁等待

```
postgres=# select * from v_locks_monitor ;
-[ RECORD 1 ]-+-----------------------------------------------------------------------------------------------------------
locktype      | relation
datname       | postgres
relation      | locktest
page          |
tuple         |
virtualxid    |
transactionid |
classid       |
objid         |
objsubid      |
string_agg    | Pid: 23043
              | Granted: false , Mode: AccessExclusiveLock , FastPath: false , VirtualTransaction: 4/1450064 , Session_St
              | Username: postgres , Database: postgres , Client_Addr: NULL , Client_Port: -1 , Application_Name: psql
              | Xact_Start: 2017-05-21 21:43:43.735829+08 , Query_Start: 2017-05-21 21:43:50.965797+08 , Xact_Elapse: 00:
              | Query: truncate locktest ;
              | --------
              | Pid: 40698
              | Granted: true , Mode: RowExclusiveLock , FastPath: false , VirtualTransaction: 6/1031925 , Session_State:
              | Username: postgres , Database: postgres , Client_Addr: NULL , Client_Port: -1 , Application_Name: psql
              | Xact_Start: 2017-05-21 21:43:15.173798+08 , Query_Start: 2017-05-21 21:43:24.338804+08 , Xact_Elapse: 00:
              | Query: insert into locktest values (2,'test');
              | --------
              | Pid: 17515
              | Granted: true , Mode: RowExclusiveLock , FastPath: false , VirtualTransaction: 3/5671759 , Session_State:
              | Username: postgres , Database: postgres , Client_Addr: NULL , Client_Port: -1 , Application_Name: psql
              | Xact_Start: 2017-05-21 21:42:19.199124+08 , Query_Start: 2017-05-21 21:42:47.820125+08 , Xact_Elapse: 00:
              | Query: select * from locktest ;
              | --------
              | Pid: 17515
```

性能分析利器7 - 查看当前锁等待

```
| --------
| Pid: 24781
| Granted: false , Mode: AccessShareLock , FastPath: false , VirtualTransaction: 7/1025270 , Session_State:
| Username: postgres , Database: postgres , Client_Addr: NULL , Client_Port: -1 , Application_Name: psql
| Xact_Start: 2017-05-21 21:44:20.725834+08 , Query_Start: 2017-05-21 21:44:20.725834+08 , Xact_Elapse: 00:
| Query: select * from locktest ;
```

性能分析利器8 - AWR

- PostgreSQL
  - https://github.com/digoal/blog/blob/master/201611/20161123_01.md
- PPAS
  - https://github.com/digoal/blog/blob/master/201606/20160628_01.md

IO、CPU飙升分析、优化

- OS
  - cgroup 检查throtte
  - top find PID
  - iotop find PID
  - iostat， 是否到达瓶颈、IOHANG问题
  - dmesg
- DB
  - pid -> find pg_stat_activity
  - freeze 风暴问题
    - https://github.com/digoal/blog/blob/master/201804/20180411_01.md
    - https://github.com/digoal/blog/blob/master/201801/20180117_03.md
    - https://github.com/digoal/blog/blob/master/201606/20160612_01.md
    - https://github.com/digoal/blog/blob/master/201605/20160520_01.md
  - autovacuum_naptime , 长事务, 长 2PC, long sql, hot_standby_feedback(备库长事务, 长 2PC, long sql) , vacuum_defer_cleanup_age
    - https://github.com/digoal/blog/blob/master/201704/20170410_03.md
    - https://github.com/digoal/blog/blob/master/201505/20150503_01.md

plpgsql函数性能诊断

- auto_explain
- plpgsql函数中每一个调用的详细执行计划
- https://github.com/digoal/blog/blob/master/201611/20161121_02.md

plpgsql函数debug

- https://www.postgresql.org/docs/devel/static/plpgsql-control-structures.html
- https://www.postgresql.org/docs/devel/static/plpgsql-statements.html#PLPGSQL-STATEMENTS-DIAGNOSTICS
- https://www.postgresql.org/docs/devel/static/plpgsql-errors-and-messages.html

- pldebugger extension + pgadmin
  - https://github.com/digoal/blog/blob/master/201704/20170424_02.md
- raise notice
- print stack
  - GET STACKED DIAGNOSTICS *variable* { = | := } *item* [, ...];
  - GET [CURRENT] DIAGNOSTICS *variable* { = | := } *item* [, ...];

plpgsql函数debug

Name	Type	Description
RETURNED_SQLSTATE	text	the SQLSTATE error code of the exception
COLUMN_NAME	text	the name of the column related to exception
CONSTRAINT_NAME	text	the name of the constraint related to exception
PG_DATATYPE_NAME	text	the name of the data type related to exception
MESSAGE_TEXT	text	the text of the exception's primary message
TABLE_NAME	text	the name of the table related to exception
SCHEMA_NAME	text	the name of the schema related to exception
PG_EXCEPTION_DETAIL	text	the text of the exception's detail message, if any
PG_EXCEPTION_HINT	text	the text of the exception's hint message, if any
PG_EXCEPTION_CONTEXT	text	line(s) of text describing the call stack at the time of the exception (see Section 42.6.7)

Name	Type	Description
ROW_COUNT	bigint	the number of rows processed by the most recent SQL command
RESULT_OID	oid	the OID of the last row inserted by the most recent SQL command (only useful after an INSERT command into a table having OIDs)
PG_CONTEXT	text	line(s) of text describing the current call stack (see Section 42.6.7)

plpgsql 判断有无满足条件记录

- perform 1 from tbl where .... limit 1;
- if FOUND then
- ...
- else
- ...
- end if;

- **DON'T use**
  - select count(*) into var from tbl where ....;
  - if var >= 1 then
  - else
  - end if;

事务可靠性、可用性级别设置

事务级 提交模式可调。
例如涉及金额较大的使用同步模式
涉及金额较小的使用异步模式

也可以设置为：
1、用户级
2、数据库级
https://github.com/digoal/blog/blob/master/201712/20171207_01.md

事务可靠性、可用性级别设置

- https://www.postgresql.org/docs/10/static/runtime-config-wal.html#GUC-SYNCHRONOUS-COMMIT
  - synchronous_commit
    - on, remote_apply, remote_write, local, off
- https://www.postgresql.org/docs/10/static/runtime-config-replication.html#RUNTIME-CONFIG-REPLICATION-MASTER
  - synchronous_standby_names
    - [FIRST] *num_sync* (*standby_name* [, ...])
    - ANY *num_sync* (*standby_name* [, ...])
    - *standby_name* [, ...]

分布式事务(2PC)

- 分布式事务
- https://github.com/digoal/blog/blob/master/201102/20110214_01.md
- 勿滥用2PC
- https://github.com/digoal/blog/blob/master/201509/20150924_01.md
- max_prepared_transactions = 1000

- postgres=# begin;
- postgres=# insert into t values (0,'abc');
- postgres=# **prepare transaction 'pxt1';**
- PREPARE TRANSACTION
- postgres=# select * from pg_prepared_xacts ;
- transaction | gid | prepared | owner | database
- ----------+------+----------+----------+----------
- 2756839324 | pxt1 | 2018-04-18 15:08:15.901434+08 | postgres | postgres
- (1 row)
- postgres=# **commit prepared 'pxt1';**
- COMMIT PREPARED
- postgres=# select * from pg_prepared_xacts ;
- transaction | gid | prepared | owner | database
- ----------+-----+----------+-------+----------
- (0 rows)

RDS PG资源使用

实例链路信息 实例Proxy连接日志 实例性能信息 实例卡慢诊断 执行SQL

查询区间： 最近3小时 最近6小时 最近12小时 最近1天 最近2天 2017-11-15 09:33:40~2017-11-15 12:33:40

abc 磁盘空间(MB)

RDS PG资源使用

磁盘空间

磁盘空间详情

iops

物理内存(MB)

mem_usage

ms_latency

conn_count

连接数利用率

disk_usage

IOPS利用率

cpu利用率

cpu_throttled_averag...

cpu_throttled_averag...

mem_cache

pg_slave_io

memory_alloc_failcnt

conn_active_count

conn_waiting_count

long_transaction_cou...

long_idle_transactio...

logical_replication_...

database_ages

table_ages

table_ages_remain

防雪崩

- https://github.com/digoal/blog/blob/master/201712/20171211_02.md
- statement_timeout
  - 语句超时，防止雪崩
- lock_timeout
  - 锁超时
- deadlock_timeout
  - 死锁超时
- idle_in_transaction_session_timeout
  - 空闲中事务超时

DDL操作建议 - 防止DDL锁等待引发雪崩

- 锁等待机制介绍
  - https://github.com/digoal/blog/blob/master/201705/20170521_01.md

- DDL、大锁建议
  - begin;
  - set lock_timeout='ns';
  - DDL
  - end;

AB 表切换

- https://github.com/digoal/blog/blob/master/201807/20180725_04.md

限制慢SQL并发度

- 杀掉最近发起的慢SQL，老的慢SQL继续，保证N个慢SQL并发
  - select pg_terminate_backend(pid) from pg_stat_activity where now()-query_start > interval **'? second'** order by query_start offset **$N**;
  - 或 pg_cancel_backend(pid)
  - https://www.postgresql.org/docs/devel/static/functions-admin.html#FUNCTIONS-ADMIN-SIGNAL

杀会话、杀QUERY

- 杀会话
  - select **pg_terminate_backend**(pid);
    - 杀某个会话
  - select pg_terminate_backend(pid) from pg_stat_activity where pg_backend_pid()<>pid;
    - 杀所有会话
    - 杀某个用户的所有会话
      - select pg_terminate_backend(pid) from pg_stat_activity where usename=? and pid<>pg_backend_pid();
- 杀QUERY
  - select **pg_cancel_backend**($pid);

杀会话、杀QUERY

Name	Return Type	Description
pg_cancel_backend(*pid* int)	boolean	Cancel a backend's current query. This is also allowed if the calling role is a member of the role whose backend is being canceled or the calling role has been granted pg_signal_backend, however only superusers can cancel superuser backends.
pg_reload_conf()	boolean	Cause server processes to reload their configuration files
pg_rotate_logfile()	boolean	Rotate server's log file
pg_terminate_backend(*pid* int)	boolean	Terminate a backend. This is also allowed if the calling role is a member of the role whose backend is being terminated or the calling role has been granted pg_signal_backend, however only superusers can terminate superuser backends.

防DDoS、暴力破解

- DDoS
  - https://github.com/digoal/blog/blob/master/201706/20170629_02.md
  - authentication_timeout
    - authentication_timeout= '1s'
- 暴力破解
  - https://github.com/digoal/blog/blob/master/201410/20141009_01.md
  - auth_delay
    - auth_delay.milliseconds = '500'

是否被DDoS

- V1、当前总连接数：
  - select count(*) from pg_stat_activity;
- V2、最大允许连接数：
  - show max_connections;
- V3、当前已占用slot数：
  - netstat -anp|grep -c $xxxx

- DDoS判断标准：
  - (V2 = V3) > (V1+superuser_reserved_connections)
  - https://www.postgresql.org/docs/devel/static/runtime-config-connection.html#RUNTIME-CONFIG-CONNECTION-SETTINGS

数据同步

- DTS
- DATAX
  - https://github.com/alibaba/DataX
- rds_dbsync
  - https://github.com/aliyun/rds_dbsync
- PG 10 订阅功能
  - https://github.com/digoal/blog/blob/master/201702/20170227_01.md

数据订阅

- 集群级订阅
  - https://github.com/digoal/blog/blob/master/201707/20170711_01.md
- 表级订阅
  - https://github.com/digoal/blog/blob/master/201702/20170227_01.md
- 多通道订阅
  - https://github.com/digoal/blog/blob/master/201706/20170624_01.md
- DDL订阅
  - https://github.com/digoal/blog/blob/master/201712/20171204_04.md

跨库访问

- dblink
  - create extension dblink;
    - https://www.postgresql.org/docs/devel/static/dblink.html
  - PPAS支持PostgreSQL, Oracle 两种DBLINK
    - https://github.com/digoal/blog/blob/master/201801/20180119_01.md
    - CREATE DATABASE LINK chicago CONNECT TO admin IDENTIFIED BY 'mypassword' USING oci '//127.0.0.1/acctg';
    - CREATE DATABASE LINK boston CONNECT TO admin IDENTIFIED BY 'mypassword' USING libpq 'host=127.0.0.1 dbname=sales';

外部表

- 基于 dblink 的视图
  - PPAS支持PostgreSQL, Oracle 两种DBLINK
  - https://www.postgresql.org/docs/10/static/dblink.html
- postgres_fdw 外部表
  - https://www.postgresql.org/docs/10/static/postgres-fdw.html
- oracle_fdw 外部表
  - https://pgxn.org/dist/oracle_fdw/

FDW 外部表 - 数据融合

- 用于分级存储、数据库互通
- OSS外部表
  - 分级存储：
  - RDS PG OSS 外部表文档：https://help.aliyun.com/knowledge_detail/43352.html
  - HDB PG OSS 外部表文档：https://help.aliyun.com/document_detail/35457.html
- 其他外部表
  - https://wiki.postgresql.org/wiki/Fdw
  - 数据库互通。
  - file
  - oracle
  - mysql
  - sqlserver
  - hadoop.....

分区表

- PG 内置分区表语法
  - https://github.com/digoal/blog/blob/master/201612/20161215_01.md
  - https://github.com/digoal/blog/blob/master/201802/20180205_02.md
- PG 传统分区表
  - https://github.com/digoal/blog/blob/master/201711/20171122_02.md
- pg_pathman分区表
  - https://github.com/digoal/blog/blob/master/201710/20171015_01.md
  - https://github.com/digoal/blog/blob/master/201610/20161024_01.md

定时任务

- Data Studio
- Crontab
  - https://github.com/digoal/blog/blob/master/201305/20130531_02.md
- pgagent
  - https://github.com/digoal/blog/blob/master/201305/20130531_01.md

执行计划

- postgres=# explain **(analyze,verbose,timing,costs,buffers)** select count(*) from a where id=1;

```
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=2.85..2.86 rows=1 width=8) (actual time=0.543..0.543 rows=1 loops=1)
   Output: count(*)
   Buffers: shared read=4
   ->  Index Only Scan using a_pkey on public.a  (cost=0.43..2.85 rows=1 width=0) (actual time=0.532..0.533 rows=1 loops=1)
         Output: id
         Index Cond: (a.id = 1)
         Heap Fetches: 1
         Buffers: shared read=4
 Planning time: 0.914 ms
 Execution time: 0.591 ms
(10 rows)
```

CBO 成本因子

- https://www.postgresql.org/docs/devel/static/runtime-config-query.html#RUNTIME-CONFIG-QUERY-CONSTANTS
- #seq_page_cost = 1.0 # measured on an arbitrary scale
- #random_page_cost = 4.0 # same scale as above
- #cpu_tuple_cost = 0.01 # same scale as above
- #cpu_index_tuple_cost = 0.005 # same scale as above
- #cpu_operator_cost = 0.0025 # same scale as above
- #parallel_tuple_cost = 0.1 # same scale as above
- #parallel_setup_cost = 1000.0 # same scale as above
- #min_parallel_table_scan_size = 8MB
- #min_parallel_index_scan_size = 512kB
- #effective_cache_size = 4GB

Join 介绍

- Join 方法
  - https://www.postgresql.org/docs/devel/static/planner-optimizer.html
- Join 背景原理
  - https://github.com/digoal/blog/blob/master/201802/20180205_01.md
  - https://www.postgresql.org/docs/devel/static/queries-table-expressions.html#QUERIES-JOIN
- 并行 Join
  - https://github.com/digoal/blog/blob/master/201802/20180202_02.md
  - https://github.com/digoal/blog/blob/master/201802/20180201_02.md
  - https://github.com/digoal/blog/blob/master/201802/20180201_01.md

JOIN优化

- 多表JOIN时，JOIN顺序直接决定了最终成本
  - 类似经典的商旅问题(走完所有的点)，实际更复杂
  - TSP (traveling salesman problem)

强制固定JOIN顺序

- 1 disables collapsing of explicit JOIN clauses
- https://www.postgresql.org/docs/devel/static/runtime-config-query.html#RUNTIME-CONFIG-QUERY-OTHER

- 控制子查询提升
  - # from_collapse_limit = 8

- 控制显示INNER JOIN顺序
  - # join_collapse_limit = 8

优化器遗传算法设置

- 解决多表JOIN优化器穷举带来的性能问题
- GEQO (geqo_threshold)
  - traveling salesman problem (TSP)
  - D. Whitley's Genitor algorithm
  - https://www.postgresql.org/docs/10/static/geqo.html

- All Pairs Shortest Path, Johnson's Algorithm
- All Pairs Shortest Path, Floyd-Warshall Algorithm
- Shortest Path A*
- Bi-directional Dijkstra Shortest Path
- Bi-directional A* Shortest Path
- Shortest Path Dijkstra
- Driving Distance
- K-Shortest Path, Multiple Alternative Paths
- K-Dijkstra, One to Many Shortest Path
- Traveling Sales Person
- Turn Restriction Shortest Path (TRSP)

计算若干个相邻节点的开销。
图算法，找到跑完所有节点的TSP最小开销。

GEQO：NP完全问题，近似求解。
相比穷举计算所有组合的开销，更低。

窗口、帧查询

- 窗口查询语法
  - https://www.postgresql.org/docs/devel/static/sql-expressions.html#SYNTAX-WINDOW-FUNCTIONS
  - https://github.com/digoal/blog/blob/master/201802/20180224_01.md
- 窗口函数
  - https://www.postgresql.org/docs/devel/static/functions-window.html
- 聚合函数
  - https://www.postgresql.org/docs/devel/static/functions-aggregate.html

窗口、帧查询例子

- 与第一名分差
  - select id, **first_value(score) over(partition by sub order by score desc)** - score, score, sub from t order by sub,id;
- 每门课程排名
  - select id,sub,score,**rank() over (partition by sub order by score desc)** from t order by sub,id;
- 滑窗分析-每条记录附近10(11)条记录的平均值
  - select id,class,score,**avg(score) over (partition by class rows between 5 preceding and 5 following)** from t order by class,id;
- 滑窗分析-每一天相比前一天的新增UV，最近7天新增UV
  - SELECT date, **(# hll_union_agg(users) OVER two_days) - (# lag(users) over (ORDER BY date ASC))** AS new_uniques FROM daily_uniques **WINDOW two_days AS (ORDER BY date ASC ROWS 1 PRECEDING);**
  - SELECT date, **# hll_union_agg(users) OVER seven_days** FROM daily_uniques **WINDOW seven_days AS (ORDER BY date ASC ROWS 6 PRECEDING)**;
- 数据去重
  - delete from tbl where ctid = any (array(
  - select ctid from
  - (select **ctid,row_number() over (partition by id order by crt_time desc) as rn** from tbl) t
  - where t.rn<>1
  -));

递归查询

- UNION 去重复(去重复时NULL 视为等同)
- 图中所有输出都涉及UNION [ALL]的操作, 包含以往返回的记录和当前返回的记录

递归查询案例 - 图式搜索

https://github.com/digoal/blog/blob/master/201706/20170601_02.md
https://github.com/digoal/blog/blob/master/201801/20180102_04.md

```
create or replace function graph_search1(
  IN i_root int,                        -- 根据哪个节点开始搜
  IN i_depth int  default 99999,        -- 搜索层级、深度限制
  IN i_limit int8 default 2000000000,   -- 限制每一层返回的记录数
  IN i_weight float8 default 0,         -- 限制权重
  OUT o_path int[],                     -- 输出：路径，ID 组成的数组
  OUT o_point1 int,                     -- 输出：点1 ID
  OUT o_point2 int,                     -- 输出：点2 ID
  OUT o_link_prop jsonb,                -- 输出：当前两点之间的连接属性
  OUT o_depth int                       -- 输出：当前深度、层级
) returns setof record as $$
declare
  sql text;
begin
sql := format($_$
```

递归查询案例 - 图式搜索

```
WITH RECURSIVE search_graph(
  c1,     -- 点1
  c2,     -- 点2
  prop,   -- 当前边的属性
  depth,  -- 当前深度，从1开始
  path,   -- 路径，数组存储
  cycle   -- 是否循环
) AS (
        select c1,c2,prop,depth,path,cycle from (
        SELECT                              -- ROOT节点查询
          g.c1,                             -- 点1
          g.c2,                             -- 点2
          g.prop,                           -- 边的属性
          1 depth,                          -- 初始深度=1
          ARRAY[g.c1] path,                 -- 初始路径
          false  as cycle                   -- 是否循环(初始为否)
        FROM a AS g
        WHERE
          c1 = %s                                                   -- ROOT节点=?
          AND coalesce((g.prop->>'weight')::float8,0) >= %s         -- 相关性权重
          ORDER BY coalesce((g.prop->>'weight')::float8,0) desc     -- 可以使用ORDER BY，例如返回权重排在前面的N条。
          limit %s                          -- 每个层级限制多少条？
        ) t
```

递归查询案例 - 图式搜索

```
      UNION ALL
        select c1,c2,prop,depth,path,cycle from (
        SELECT                                -- 递归子句
          g.c1,                               -- 点1
          g.c2,                               -- 点2
          g.prop,                             -- 边的属性
          sg.depth + 1 depth,                 -- 深度+1
          path || g.c1 path,                  -- 路径中加入新的点
          (g.c1 = ANY(path)) as cycle         -- 是否循环，判断新点是否已经在之前的路径中
        FROM a AS g, search_graph AS sg       -- 循环 INNER JOIN
        WHERE
          g.c1 = sg.c2                        -- 递归JOIN条件
          AND NOT cycle                       -- 防止循环
          AND sg.depth <= %s                  -- 搜索深度=?
          AND coalesce((g.prop->>'weight')::float8,0) >= %s          -- 相关性权重
          ORDER BY coalesce((g.prop->>'weight')::float8,0) desc      -- 可以使用ORDER BY，例如返回权重排在前面的N条。
          limit %s                            -- 每个层级限制多少条？
        ) t
)
```

递归查询案例 - 图式搜索

```
SELECT path||c2 as o_path, c1 as o_point1, c2 as o_point2, prop as o_link_prop, depth as o_depth
FROM search_graph;                          -- 查询递归表，可以加LIMIT输出，也可以使用游标
$_$, i_root, i_weight, i_limit, i_depth, i_weight, i_limit
);

return query execute sql;

end;
$$ language plpgsql strict;
```

递归查询案例 - 图式搜索

```
postgres=# select * from graph_search1(31208, 3, 100, 0);
             o_path              | o_point1 | o_point2 | o_link_prop | o_depth
---------------------------------+----------+----------+-------------+---------
 {31208,344710}                  |    31208 |   344710 |             |       1
 {31208,319951}                  |    31208 |   319951 |             |       1
 {31208,340938}                  |    31208 |   340938 |             |       1
 {31208,325272}                  |    31208 |   325272 |             |       1
 {31208,346519}                  |    31208 |   346519 |             |       1
 {31208,314594}                  |    31208 |   314594 |             |       1
 {31208,307217}                  |    31208 |   307217 |             |       1
 {31208,348009}                  |    31208 |   348009 |             |       1
 {31208,300046}                  |    31208 |   300046 |             |       1
 {31208,344359}                  |    31208 |   344359 |             |       1
 {31208,318790}                  |    31208 |   318790 |             |       1
 {31208,321034}                  |    31208 |   321034 |             |       1
 {31208,301609}                  |    31208 |   301609 |             |       1
 {31208,344339}                  |    31208 |   344339 |             |       1
 {31208,314087}                  |    31208 |   314087 |             |       1
------
 {31208,319951,3199647,31991191} |  3199647 | 31991191 |             |       3
 {31208,319951,3199647,31954904} |  3199647 | 31954904 |             |       3
 {31208,319951,3199647,31986691} |  3199647 | 31986691 |             |       3
 {31208,319951,3199647,31986448} |  3199647 | 31986448 |             |       3
 {31208,319951,3199647,31993624} |  3199647 | 31993624 |             |       3
 {31208,319951,3199647,31997771} |  3199647 | 31997771 |             |       3
 {31208,319951,3199647,31982764} |  3199647 | 31982764 |             |       3
 {31208,319951,3199647,31993420} |  3199647 | 31993420 |             |       3
 {31208,319951,3199647,31962666} |  3199647 | 31962666 |             |       3
 {31208,319951,3199647,31957536} |  3199647 | 31957536 |             |       3
(300 rows)
```

空间查询优化

- GIST索引面收敛查询优化
  - https://github.com/digoal/blog/blob/master/201711/20171122_03.md

AOI优化

- GiST空间索引结构为bound box，对于不规则多边形，会引入一些边界放大问题
  - CPU放大
  - IO放大
- 优化方法
  - 空间SPLIT
    - https://github.com/digoal/blog/blob/master/201710/20171004_01.md

空间查询优化

- GIST索引面收敛查询优化
  - https://github.com/digoal/blog/blob/master/201711/20171122_03.md

空间查询优化

- GIST索引面收敛查询优化
  - https://github.com/digoal/blog/blob/master/201711/20171122_03.md
  - 菜鸟、高德、HELLOBIKE、新零售、空间透视分析、。。。。。。

优化前	优化1（空间聚集）	优化1,2(SPLIT多边形)
访问35323块	访问1648块	访问243块
过滤26590条	过滤26590条	过滤0条

空间数据使用建议

- https://github.com/digoal/blog/blob/master/201710/20171018_02.md
- https://github.com/digoal/blog/blob/master/201708/20170809_01.md

- 球面距离计算
  - **ST_DistanceSpheroid(geom,ST_SetSRID(ST_Point(102,24),4326),'SPHEROID["WGS84",6378137,298.257223563]')**
  - vspheroid := 'SPHEROID["WGS84",6378137,298.257223563]' ;
    - --WGS84椭球体参数定义
  - vcurrentpoint := ST_SetSRID(ST_Point(ix,iy),4326);
    - 设置 SRID
- KNN查询优化
  - https://github.com/digoal/blog/blob/master/201308/20130806_01.md

空间数据使用建议

```
digoal=# do language plpgsql $$
declare
  v_rec record;
  v_limit int := 1000;
begin
  set local enable_seqscan=off;  -- 强制索引，因为扫描行数够就退出.
  for v_rec in
    select *,ST_DistanceSpheroid(jwd, ST_SetSRID(ST_Point(120.19, 30.26),4326), 'SPHEROID["WGS84",6378137,298.257223563]') AS dist
    from cust_jw order by jwd <-> ST_SetSRID(ST_Point(120.19, 30.26),4326)
  loop
    if v_limit <=0 then
      raise notice '已经取足数据';
      return;
    end if;
    if v_rec.dist > 20000 then
      raise notice '满足条件的点已输出完毕';
      return;
    else
      raise notice 'do someting, v_rec:%', v_rec;
    end if;
    v_limit := v_limit -1;
  end loop;
end;
$$;
NOTICE:  do someting, v_rec:(杭州,0101000020730800004C94087D5D4F54C173AA7759E8FB5D41,0)
NOTICE:  do someting, v_rec:(余杭,0101000020730800000E6E5A20494854C121FC688DA9EF5D41,14483.9823187612)
NOTICE:  满足条件的点已输出完毕
DO
```

空间数据使用建议

- 时间、空间、对象 多维搜索
  - 位图扫描
    - gist 索引
    - btree_gist 插件
    - https://www.postgresql.org/docs/devel/static/btree-gist.html
    - https://github.com/digoal/blog/blob/master/201702/20170221_02.md
  - 分区
    - https://github.com/digoal/blog/blob/master/201711/20171122_03.md
      https://github.com/digoal/blog/blob/master/201710/20171005_01.md
    - https://github.com/digoal/blog/blob/master/201710/20171004_01.md
  - BRIN
    - https://github.com/digoal/blog/blob/master/201708/20170820_01.md

数据老化实践

https://github.com/digoal/blog/blob/master/201712/20171208_01.md

分级存储

- https://help.aliyun.com/knowledge_detail/43352.html
- 热数据
  - 实例本地存储
- 访问频次较低数据
  - oss外部表存储
  - 压缩格式选择
- 继承与分区约束
  - 每个OSS外部表负责一部分数据
  - 使用约束
  - 建立OSS外部表继承关系

复杂业务逻辑延迟问题优化

- 一、
- 数
- 据
- 库
- 端
- 编
- 程

复杂业务逻辑延迟问题优化

https://github.com/digoal/blog/blob/master/201509/20150910_01.md

- 一、
- 数
- 据
- 库
- 端
- 编
- 程

对比20条QUERY(pk I,U,D,S)的事务TPS
1.交互式（业务逻辑在client完成）
2.服务端编程模式（业务逻辑封装在数据库FUNC完成）

逻辑结构

- https://github.com/digoal/blog/blob/master/201605/20160510_01.md

权限体系

https://www.postgresql.org/docs/devel/static/sql-grant.html

实例权限	• 修改pg_hba.conf
数据库权限	• grant赋予是否允许连接或创建schema的权限 • revoke 回收
schema权限	• grant赋予允许查询schema中的对象,或在schema中创建对象 • revoke 回收
object权限	• grant赋予 • revoke 回收
表空间	• grant 赋予允许在对应表空间创建表，物化视图，索引，临时表 • revoke 回收

行级权限控制

- RLS
  - https://www.postgresql.org/docs/10/static/sql-createpolicy.html
  - https://github.com/digoal/blog/blob/master/201602/20160203_03.md
  - https://github.com/digoal/blog/blob/master/201504/20150409_01.md
  - CREATE POLICY *name* ON *table_name* [AS { PERMISSIVE | RESTRICTIVE }] [FOR { ALL | SELECT | INSERT | UPDATE | DELETE }] [TO { *role_name* | PUBLIC | CURRENT_USER | SESSION_USER } [, ...]] [USING (*using_expression*)] [**WITH CHECK (*check_expression*)**]

列级权限控制

- GRANT { { SELECT | INSERT | UPDATE | REFERENCES } (***column_name*** [, ...]) [, ...] | ALL [PRIVILEGES] (*column_name* [, ...]) } ON [TABLE] *table_name* [, ...] TO *role_specification* [, ...] [WITH GRANT OPTION]

为什么会膨胀

- 表、索引、物化视图为什么会膨胀

- 如何检查膨胀
  - https://github.com/digoal/blog/blob/master/201306/20130628_01.md
  - https://raw.githubusercontent.com/digoal/pgsql_admin_script/master/generate_report.sh
- 膨胀如何处理
  - vacuum full , 在线repack
  - https://www.postgresql.org/docs/devel/static/sql-vacuum.html
  - https://github.com/digoal/blog/blob/master/201610/20161030_02.md
- 如何预防膨胀
  - https://github.com/digoal/blog/blob/master/201511/20151109_01.md
  - https://github.com/digoal/blog/blob/master/201504/20150429_02.md
  - https://github.com/digoal/blog/blob/master/201704/20170410_03.md

freeze 风暴、追溯

- https://github.com/digoal/blog/blob/master/201606/20160612_01.md
- https://github.com/digoal/blog/blob/master/201801/20180117_03.md
- 业务空闲、But IO\CPU偏高。
  - select age(a.relfrozenxid), last_autovacuum,last_vacuum,schemaname,a.relname,pg_size_pretty(pg_total_relation_size(relid)) from pg_class a, pg_stat_all_tables b where a.oid=b.relid and a.relkind in ('r', 'm') order by last_autovacuum nulls last;

freeze 风暴、追溯

可以大概推测。

```
   age    | last_autovacuum | last_vacuum |     schemaname     |         relname         | pg_size_pretty
----------+-----------------+-------------+--------------------+-------------------------+----------------
       46 |                 |             | public             | test                    | 5608 MB
       43 |                 |             | public             | test1                   | 5784 kB
 80593695 |                 |             | pg_catalog         | pg_statistic            | 248 kB
 80593695 |                 |             | pg_catalog         | pg_type                 | 184 kB
       39 |                 |             | public             | a                       | 48 kB
       32 |                 |             | public             | b                       | 16 kB
 80593695 |                 |             | pg_catalog         | pg_policy               | 16 kB
       22 |                 |             | public             | c                       | 48 kB
 80593695 |                 |             | pg_catalog         | pg_authid               | 72 kB
...............
```

预测、预防FREEZE风暴

```
psql -c "drop table pred_io; create table pred_io(crt_time timestamp, bytes int8, left_live int8);"
for db in `psql -A -t -q -c "select datname from pg_database where datname <> 'template0'"`
do
psql -d $db -c " copy (
select now(), bytes, case when max_age>age then max_age-age else 0 end as xids from
(select block_size*relpages bytes,
case when d_max_age is not null and d_max_age<max_age then d_max_age else max_age end as max_age,
age from
(select
(select setting from pg_settings where name='block_size')::int8 as block_size,
(select setting from pg_settings where name='autovacuum_freeze_max_age')::int8 as max_age,
relpages,
substring(reloptions::text,'autovacuum_freeze_max_age=(\d+)')::int8 as d_max_age,
age(relfrozenxid) age
from pg_class where relkind in ('r', 't')) t) t
) to stdout;" | psql -d $PGDATABASE -c "copy pred_io from stdin"
done
```

预测、预防FREEZE风暴

```
postgres=# select * from pred_io limit 10;
          crt_time          | bytes  | left_live
----------------------------+--------+-----------
 2016-06-12 13:24:08.666995 | 131072 | 199999672
 2016-06-12 13:24:08.666995 |  65536 | 199999672
 2016-06-12 13:24:08.666995 |      0 | 199999672
 2016-06-12 13:24:08.666995 |      0 | 199999672
 2016-06-12 13:24:08.666995 |      0 | 199999672
 2016-06-12 13:24:08.666995 |      0 | 199999672
...
```

```
create view pred_tbased_io as with a as (
select crt_time, xids as s_xids, lead(xids)
over(order by crt_time) as e_xids from v_pred_xids)
select a.crt_time, sum(b.bytes) bytes from a, pred_io b
where b.left_live >=a.s_xids and b.left_live < a.e_xids
group by a.crt_time order by a.crt_time;
```

未来一天的freeze io bytes分时走势

得到的结果可能是这样的

```
postgres=# select * from pred_tbased_io
          crt_time          |  bytes
----------------------------+---------
 2016-06-13 12:36:13.201315 |   65536
 2016-06-13 12:37:13.216002 |  581632
 2016-06-13 12:38:13.240739 |       0
 2016-06-13 12:39:13.259203 |       0
 2016-06-13 12:40:13.300604 |       0
 2016-06-13 12:41:13.325874 |       0
 2016-06-13 12:43:15.481609 |  106496
 2016-06-13 12:43:24.133055 |    8192
 2016-06-13 12:45:24.193318 |       0
 2016-06-13 12:46:24.225559 |   16384
 2016-06-13 12:48:24.296223 | 13434880
 2016-06-13 12:49:24.325652 |   24576
 2016-06-13 12:50:24.367232 |  401408
 2016-06-13 12:51:24.426199 |       0
 2016-06-13 12:52:24.457375 |  393216
------
```

预测、预防FREEZE风暴

- 预防
  - https://www.postgresql.org/docs/devel/static/sql-createtable.html
  - https://www.postgresql.org/docs/devel/static/runtime-config-autovacuum.html
  - 重点关注几个表级 autovacuum freeze 参数
  - AUTOVACUUM_ANALYZE_SCALE_FACTOR
  - AUTOVACUUM_VACUUM_SCALE_FACTOR
  - AUTOVACUUM_ENABLED
  - AUTOVACUUM_FREEZE_MIN_AGE
  - AUTOVACUUM_FREEZE_TABLE_AGE
  - AUTOVACUUM_FREEZE_MAX_AGE
  - AUTOVACUUM_VACUUM_COST_DELAY
  - AUTOVACUUM_VACUUM_COST_LIMIT

分区建议

- 单表多大建议使用分区表
  - 非常频繁更新的表（考虑到autovacuum的速度）
    - 2亿
    - 指表中频繁被更新的记录数在2亿以内，表可以大到20亿不分区。
  - 更新、删除不频繁或毫无的表（考虑到设计rewrite的DDL，建索引，逻辑备份等的速度）
    - 20亿

可定义SLA的备份恢复架构设计

blog.163.com/digoal@126

可定义SLA的备份恢复架构设计

https://github.com/digoal/blog/blob/master/201711/20171129_02.md

跨版本升级 - 增量平滑升级

- pg_upgrade + zfs, 支持8.3+
  - https://github.com/digoal/blog/blob/master/201412/20141219_01.md
- 工具 - 支持9.4+
  - https://github.com/aliyun/rds_dbsync
  - https://www.2ndquadrant.com/en/resources/pglogical/
  - FDW
    - https://github.com/digoal/blog/blob/master/201710/20171027_01.md
    - https://github.com/digoal/blog/blob/master/201710/20171027_02.md
- 订阅-逻辑复制, 支持10+
  - https://github.com/digoal/blog/blob/master/201704/20170413_01.md
  - https://github.com/digoal/blog/blob/master/201702/20170227_01.md
  - https://github.com/digoal/blog/blob/master/201712/20171204_04.md
  - https://github.com/digoal/blog/blob/master/201706/20170624_01.md

数据库监控、健康指标

- https://github.com/digoal/pgsql_admin_script/blob/master/generate_report.sh
- 操作系统信息(资源限制、防火墙、硬件错误、磁盘寿命)
- 数据库
- 错误日志
- 慢SQL
- 空间利用率
- 空间变化趋势
- TOP 对象
- 数据库连接（总、已使用、剩余、认证中、库级、用户级）
- TOP SQL（IO\CPU\CALLS\timing）
- 未使用索引
- 索引数超过N的对象
- 回滚比例
- QPS\TPS
- 膨胀对象
- 垃圾比例
- 未引用大对象
- 剩余年龄
- 长事务、2PC
- 归档状态
- 备库延迟
- HA状态
- 订阅延迟
- 简单密码
- 序列剩余
- 锁等待
- 安全风险（触发器、事件触发器、规则、SQL注入、密码错误尝试、对象名）

多实例管理、监控

- https://github.com/dalibo/pgbadger
- https://www.pgadmin.org/
- http://pgstatsinfo.sourceforge.net/

快速构造海量测试数据

- https://github.com/digoal/blog/blob/master/201711/20171121_01.md
- pgbench
- https://www.postgresql.org/docs/devel/static/pgbench.html

Converting from other Databases to PostgreSQL

- https://wiki.postgresql.org/wiki/Converting_from_other_Databases_to_PostgreSQL
- https://github.com/aliyun/rds_dbsync
- https://yq.aliyun.com/articles/225946
- EDB MTK
  - https://www.enterprisedb.com/resources/product-documentation

常见热门问题

- 1、PG多机房高可用方案
- 2、ADHoC查询
- 3、数据导入、同步，实时性与一致性
  - https://github.com/digoal/blog/blob/master/201702/20170227_01.md
  - https://www.2ndquadrant.com/en/resources/pglogical/
  - https://github.com/aliyun/rds_dbsync
- 4、分布式事务(2PC)
- 5、数据容器类型 - JSONB
- 6、如何在线新增索引、新增列
- 7、评估什么时候该主备读写分离与应用实践
- 8、用户画像：带有属性的标签（复合标签，二维、多维数组）
- 9、PG的库端编程能力在什么场景适用可以帮到业务
  - 一个事务需要与数据库交互很多次，并且网络延迟已成为瓶颈(oddo)
  - 需要拉取大量数据到应用端进行处理

开发、管理实践

- RDS PPAS
- RDS PostgreSQL
- HybridDB for PostgreSQL

最佳实践(规约)

- 大锁操作尽量避免在业务时段进行（DDL操作），务必设置超时，避免影响业务。
- 业务上避免长事务，避免引起膨胀
- 定期监控膨胀，及时回收垃圾
- 无法收缩空间时，重建索引或vacuum full TABLE
- vacuum full，不能回收膨胀的索引空间，索引建议重建
- 合理配置统计信息收集调度，避免统计信息不准确
- 避免存储倾斜、避免查询倾斜
- 根据实际场景，合理选择存储格式（HEAP、AO、行、列存）
- 避免分区过多导致执行计划耗时较多，单个segment单个分区大小=单个分区记录数/segment数。 **单个分区建议不要低于200万行。**
- 大批量导入数据，建议通过OSS并行加载，建议使用GZIP压缩文件格式
- AO表避免单步INSERT，导致性能下降，文件稀疏影响性能。
  - https://github.com/digoal/blog/blob/master/201711/20171116_01.md

最佳实践(规约)

- OSS并行加载，建议的文件数与SEGMENT个数一致，大小一致，达到最好的并行度和性能
- DML为表级锁，避免并发对单表执行DML操作（UPDATE，DELETE）
- 链路模式，始终使用SLB模式，不要使用PROXY模式
- 建议使用连接池，减少连接数(降低内存开销、进程切换开销)，避免短连接。
- 如果MASTER、SEGMENT 资源利用率长期100%，建议使用资源队列，降低同时执行的并发
- 业务并发确实较高时，建议给SEGMENT，MASTER配置更多的资源（默认是1核）
- 大量使用临时表时，长期使用会导致元数据膨胀，需要vacuum full与reindex来回收空间，否则性能会受到影响。
  - pg_class, pg_attribute等，通过膨胀检测查看
- 需要使用索引、HASH JOIN 时，务必保障查询的类型对齐，JOIN的类型对齐
- 数值类型选择建议：可以使用float8时，尽量不使用numeric
- 复杂SQL使用orca优化器：set optimizer=on;

1

- **1 最佳使用实践**
  - 《Greenplum 最佳实践 - 三张图读懂OLAP数据库在企业的正确使用姿势》
  - 《Greenplum 行存、列存，堆表、AO表的原理和选择》
  - 《Greenplum 行存、列存，堆表、AO表性能对比 - 阿里云HDB for PostgreSQL最佳实践》
  - 《Greenplum 最佳实践 - 行存与列存的选择以及转换方法》
  - 《Greenplum 清理垃圾、修改存储模式（行列变换） 平滑方法 - 交换数据、交互分区》
  - 《Greenplum 最佳实践 - 数据分布黄金法则 - 分布列与分区的选择》
  - 《Greenplum 类型一致性使用注意 - 索引条件、JOIN的类型一致性限制》
  - 《Greenplum 跨库数据JOIN需求 - dblink的使用和弊端以及解决方案》

阿里云ApsaraDB GP用户推荐用法

HDB PG规格说明

计算组规格	1C SSD	2C SSD	16C SSD
	2Cores/16GBMem/160GB SSD	规格	
计算组节点	2	4	8

【注意：本产品不支持降级操作。另外如需购买更大规格或高并发规格，请通过工单系统，或客户代表提出申请】

HDB PG资源监控

- Master\Segment
- Cpu使用率
- 内存使用率
- 磁盘使用率
- iops使用率

2,3,4

- **2 估值计算实践**
  - 《Greenplum 最佳实践 - 估值插件hll的使用(以及hll分式聚合函数优化)》
  - 《PostgreSQL、Greenplum 滑动窗口 分析SQL 实践》
- **3 评估实践**
  - **《Greenplum 计算能力估算 - 暨多大表需要分区，单个分区多大适宜》**
  - 《Greenplum 性能评估公式 - 阿里云HybridDB for PostgreSQL最佳实践》
  - 《如何评估Greenplum master 空间以及segment元数据占用的空间》
- **4 索引使用实践**
  - 《Greenplum 最佳实践 - 什么时候选择bitmap索引》
  - 《PostgreSQL 9种索引的原理和应用场景》
  - 《Greenplum 空间(GIS)数据检索 b-tree & GiST 索引实践 - 阿里云HybridDB for PostgreSQL最佳实践》

5

- **5 开发实践**
  - 《Greenplum 排序nulls first|last的 SQL写法实现》
  - 《Greenplum 最佳实践 - 如何支持反转索引(reverse, orafunc)》
  - 《Greenplum 最佳实践 - 多维分析的使用(CUBE, ROLLUP, GROUPING SETS in GreenPlum and Oracle)》
  - 《Greenplum 的Oracle兼容性之 - orafunc (orafce)》
  - 《Greenplum 最佳实践 - 函数内嵌套查询在query中调用的替代方案》
  - 《PivotalR between R & PostgreSQL-like Databases(for exp : Greenplum, hadoop access by hawq)》
  - 《为什么啤酒和纸尿裤最搭 - 用HybridDB/PostgreSQL查询商品营销最佳组合》
  - 《Greenplum 模糊查询 实践》

6,7,8,9

- **6 OSS+GP实践**
  - 《打造云端流计算、在线业务、数据分析的业务数据闭环 - 阿里云RDS、HybridDB for PostgreSQL最佳实践》
- **7 日常维护实践**
  - 《Greenplum通过gp_dist_random('gp_id') 在所有节点调用某个函数》
  - 《PostgreSQL、Greenplum 日常监控 和 维护任务 - 最佳实践》
  - 《Greenplum 列存表(AO表)的膨胀、垃圾检查与空间收缩》
  - 《如何检测、清理Greenplum HEAP对象膨胀、垃圾 - 阿里云HybridDB for PG最佳实践》
- **8 数据同步实践**
  - 《MySQL准实时同步到PostgreSQL, Greenplum的方案之一 - rds_dbsync》
  - 《Greenplum, PostgreSQL 数据实时订阅的几种方式》
- **9 数据导入实践**
  - 《Greenplum insert的性能(单步\批量\copy) - 暨推荐使用gpfdist、阿里云oss外部表并行导入》

8.1

- **8.1 阿里云RDS PG，HDB PG，ODPS数据同步**
  - # 数据同步
  - MySQL, PG, Oracle, ... -> HDB PG：https://www.atatech.org/articles/92271/
  - ODPS 导出到OSS：https://www.atatech.org/articles/7100 2
  - ODPS->RDS PG PPAS,HDB PG 数据同步(支持文本、GZIP压缩格式)：https://www.atatech.org/articles/87003
  - mysql->RDS PG全量、增量同步：https://github.com/aliyun/rds_dbsync
  - mysql->RDS PG增量同步(DataX)：https://help.aliyun.com/document_detail/28291.html
  - mysql->RDS PG全量、增量迁移：https://help.aliyun.com/product/26590.html
  - 使用DataX一键快速将异构数据源数据同步到GP：https://www.atatech.org/articles/92271
  - RDS PG OSS 外部表文档：https://help.aliyun.com/knowledge_detail/43352.html
  - HDB PG OSS 外部表文档：https://help.aliyun.com/document_detail/35457.html
- mysql, sqlserver, sqlite, csv, text, pbf 迁移到PG, HDB PG
  - 高效、支持DDL
  - https://pgloader.io/

10,11,12

- **10 数据合并实践**
  - 《PostgreSQL、Greenplum DML合并操作 最佳实践》
  - 《Greenplum merge insert 用法与性能 (insert on conflict) - 2》
  - 《Greenplum merge insert 用法与性能 (insert on conflict) - 1》
- **11 优化**
  - 《Greenplum explain analyze 解读 + 深度明细开关》
  - 《大量使用临时表带来的系统表如pg_attribute膨胀问题，替代方案，以及如何擦屁股 - Greenplum, PostgreSQL最佳实践》
- **12 诊断**
  - 《Greenplum 统计信息收集参数 - 暨统计信息不准引入的broadcast motion一例》
  - 《Greenplum segment级锁问题排查方法 - 阿里云HybridDB for PostgreSQL最佳实践》
  - 《Greenplum RT高的原因分析 和 优化方法》
  - 《PostgreSQL 锁等待监控 珍藏级SQL - 谁堵塞了谁》

13

- **13 GP OLTP 实践**
  - 《Greenplum 连接池实践》
  - https://greenplum.org/docs/admin_guide/access_db/topics/pgbouncer.html
  - 《让greenplum的oltp性能飞起来 - 直接读写数据节点》
  - 《Greenplum segment节点直接读写配置与性能》
  - 《Greenplum 点查询的优化(分布键)》
  - 《Greenplum 点查(按PK查询)性能与提升空间》
  - 《Use pgbouncer connect to GreenPlum's segment node》

14

- **14 原理**
  - 《轻松打爆netfilter conntrack table的Greenplum 烂SQL》
  - 《Greenplum , HAWQ outer join与motion问题讲解》
  - 《PostgreSQL distinct 与 Greenplum distinct 的实现与优化》
  - 《Greenplum vacuum ao表和heap表的区别》
  - 《PostgreSQL vs Greenplum Hash outer join hash表的选择》
  - 《分布式DB(Greenplum)中数据倾斜的原因和解法 - 阿里云HybridDB for PostgreSQL最佳实践》
  - 《Greenplum 列存储加字段现象 - AO列存储未使用相对偏移》
  - 《阿里云HDB for PostgreSQL数据库metascan特性(存储级、块级、batch级过滤与数据编排)》
  - 《Greenplum 内存与负载管理(resource queue)最佳实践》
  - 《Greenplum 资源隔离的原理与源码分析》

- 《Greenplum ORCA 优化器的编译安装与使用》
- 《PostgreSQL和Greenplum的临时表空间介绍》
- 《Greenplum 表空间和filespace的用法》
- 节约98%的数据存储成本的方法
- 《PostgreSQL n阶乘计算, 排列组合计算 - 如何计算可变参数中有没有重复参数》
- 《PostgreSQL 计算 任意类型 字段之间的线性相关性》
- 《一个简单算法可以帮助物联网,金融 用户 节约98%的数据存储成本(PostgreSQL,Greenplum帮你做到)》
- 《Greenplum hash分布算法》
- 《Greenplum "Sort、Group、distinct 聚合、JOIN" 不惧怕数据倾斜的黑科技和原理 - 多阶段聚合》

15

- **15 问题集锦**
  - 《PostgreSQL, Greenplum ETL 之 - 非法字符(如0x00)过滤、转换(blob2text, bytea2text)》
  - 《Greenplum , PostgreSQL pgcrypto 加密算法、mode、PAD的选择 - 与Oracle, MySQL的差异(安全性差异)》
  - 《PostgreSQL 和 Greenplum pgcrypto 加解密bytea处理差异(convert, convert_from)》
  - 《PostGIS 多点几何类型 空字符构造异常CASE - parse error - invalid geometry (lwgeom_pg.c:96)》

- **PgAdmin**
  - 点一下表名，可能会发出count(*)到后台执行。

Use the options on the *Properties* dialog to specify display properties:

- **Count rows if estimated less than** - Include a value in the *Count rows if estimated less than* field to perform a SELECT count(*) if the estimated number of rows in a table (as read from the table statistics) is below the specified limit. After performing the SELECT count(*), pgAdmin will display the row count.

16

- **16 案例**
  - 《空间|时间|对象 圈人 + 透视 - 暨PostgreSQL 10与Greenplum的对比和选择》
  - 《音视图(泛内容)网站透视分析 DB设计 - 阿里云(RDS、HybridDB) for PostgreSQL最佳实践》
  - 《泛电网系统 海量实时计算+OLTP+OLAP DB设计 - 阿里云(RDS、HybridDB) for PostgreSQL最佳实践》
  - 《记录动态格式化输出(ToB日志转换业务) - 阿里云RDS PostgreSQL, HybridDB for PostgreSQL最佳实践》
  - 《(新零售)商户网格化(基于位置GIS)运营 - 阿里云RDS PostgreSQL、HybridDB for PostgreSQL最佳实践》
  - 《强制数据分布与导出prefix - 阿里云pg, hdb pg oss快速数据规整外部表导出实践案例》
  - 《日增量万亿+级 实时分析、数据规整 - 阿里云HybridDB for PostgreSQL最佳实践》
  - 《ApsaraDB的左右互搏(PgSQL+HybridDB+OSS) - 解决OLTP+OLAP混合需求》
  - 《Greenplum roaring bitmap与业务场景 (类阿里云RDS PG varbitx, 应用于海量用户 实时画像和圈选、透视)》

数据同步通道 - 1

https://github.com/aliyun/rds_dbsync

数据同步通道 - 2

OSS外部表优化：

1、文件数=计算节点数
2、开启压缩
3、允许错误条数
4、超时参数

https://help.aliyun.com/document_detail/35457.html

数据同步通道 - 3

目标数据源选择PostgreSQL

https://help.aliyun.com/document_detail/47677.html
https://help.aliyun.com/document_detail/30269.html

数据写入实践

- 批量 or 单步
- 为什么AO表写入有IO放大
  - https://github.com/digoal/blog/blob/master/201711/20171116_01.md
- 如何提速

HDB PG 分布键选择建议

- 允许随机分布
- 分布键允许多列
- 如果有唯一、主键约束，必须与分布键一致
- 分布键选择，确保不出现倾斜
- 大表、建议经常被用于JOIN的列
- https://github.com/digoal/blog/blob/master/201607/20160719_02.md

行存与列存的选择

- 支持索引类型:
  - B-tree
  - Bitmap
  - GIST
- 支持行列存储混合转换
- 支持按列存储数据库，及列数据库索引
- 透明实时数据压缩类型

行存与列存的选择 - 阿里云扩展

- Row oriented faster when returning all columns
- HEAP for many updates and deletes
- Use indexes for drill through queries

- Columnar storage compresses better
- Optimized for retrieving a subset of the columns when querying
- Compression can be set differently per column: gzip (1-9)

- Less accessed partitions on OSS with external partitions to seamlessly query all data
- CSV format

行存与列存的选择

- https://github.com/digoal/blog/blob/master/201708/20170825_02.md
- https://github.com/digoal/blog/blob/master/201708/20170818_02.md
- https://github.com/digoal/blog/blob/master/201608/20160815_01.md
- append only table
  - 批量写入、含少量DML
  - 行存
    - 查询较多字段、输出较多记录。
  - 列存
    - 统计、JOIN、少量列查询
- heap row table
  - 单步写入、含部分DML

非分布键 group by 和 distinct

- https://github.com/digoal/blog/blob/master/201711/20171123_01.md
- 对于非分布键的分组聚合请求，Greenplum采用了多阶段聚合如下：
-
- 第一阶段，在SEGMENT本地聚合。
-
- 第二阶段，根据分组字段，将结果数据重分布。
-
- 第三阶段，再次在SEGMENT本地聚合。
-
- 第四阶段，返回结果给master，有必要的话master节点调用聚合函数的final func（已经是很少的记录数和运算量）。

非分布键 内部多阶段 JOIN

- HDB PG全自动、无任何JOIN限制
- 1、数据节点本地JOIN - **解决网络开销问题**
- 2、数据节点间自动重分布
  - 小表自动广播
  - 大表，按JOIN字段自动重分布
- 3、数据节点本地JOIN
- 4、返回JOIN结果

非分布键 内部多阶段 JOIN

```
postgres=# explain analyze select count(*),c1 from a group by c1 ;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Gather Motion 48:1  (slice2; segments: 48)  (cost=1561155.53..1561156.80 rows=101 width=12)  -- 第四阶段、上报结果
   Rows out:  101 rows at destination with 4004 ms to end, start offset by 1.742 ms.
   -> HashAggregate  (cost=1561155.53..1561156.80 rows=3 width=12)  -- 第三阶段、本地聚合
        Group By: a.c1
        Rows out:  Avg 2.5 rows x 41 workers.  Max 4 rows (seg9) with 0.004 ms to first row, 1277 ms to end, start offset by 26 ms.
        -> Redistribute Motion 48:48  (slice1; segments: 48)  (cost=1561152.00..1561154.02 rows=3 width=12)  -- 第二阶段、重分布（仅少量数据走网络重分布）
             Hash Key: a.c1
             Rows out:  Avg 118.2 rows x 41 workers at destination.  Max 192 rows (seg9) with 2702 ms to end, start offset by 26 ms.
             -> HashAggregate  (cost=1561152.00..1561152.00 rows=3 width=12)  -- 第一阶段、本地聚合，收敛数据区间
                  Group By: a.c1
                  Rows out:  Avg 101.0 rows x 48 workers.  Max 101 rows (seg0) with 0.007 ms to first row, 2638 ms to end, start offset by 50 ms.
                  -> Append-only Columnar Scan on a  (cost=0.00..1061152.00 rows=2083334 width=4)
                       Rows out:  0 rows (seg0) with 25 ms to end, start offset by 52 ms.
 Slice statistics:
   (slice0)    Executor memory: 263K bytes.
   (slice1)    Executor memory: 764K bytes avg x 48 workers, 764K bytes max (seg0).
   (slice2)    Executor memory: 289K bytes avg x 48 workers, 292K bytes max (seg0).
 Statement statistics:
   Memory used: 128000K bytes
 Settings:  enable_bitmapscan=off; enable_seqscan=off; optimizer=off
 Optimizer status: legacy query optimizer
 Total runtime: 4006.957 ms
(22 rows)
```

HDB PG 分区表分区字段选择建议

- 支持范围、枚举分区
- 不建议与分布键一致
- 建议经常用于过滤的列
  - 时间
  - 枚举

分级存储功能

- https://help.aliyun.com/document_detail/35457.html
- 热数据
  - 实例本地存储
- 访问频次较低数据
  - oss外部表存储
  - 压缩格式选择
- 继承与分区约束
  - 每个OSS外部表负责一部分数据
  - 使用约束
  - 建立OSS外部表继承关系

索引选择

- 自动索引选择
  - https://github.com/digoal/blog/blob/master/201706/20170617_01.md
- GiST
  - 空间数据
- B-Tree
  - 等值、区间、排序
- Bitmap
  - https://github.com/digoal/blog/blob/master/201705/20170512_01.md
  - 类似倒排
  - value: 所有行号对应的bitmap
  - 含100到1万个唯一值的列

统计信息采集调度

- 专治SQL执行计划不准。
- gp_autostats_mode
  - none：不收集
  - on_no_stats：没有统计信息时，收集
  - on_change：当写入、更新量超过阈值（gp_autostats_on_change_threshold参数设置的行数，默认为20亿）后，自动收集统计信息。

- https://github.com/digoal/blog/blob/master/201712/20171211_03.md

队列管理

- CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ...])
- where queue_attribute is:
- ACTIVE_STATEMENTS=integer
- [MAX_COST=float [COST_OVERCOMMIT={TRUE|FALSE}]]
- [MIN_COST=float]
- [PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}]
- [MEMORY_LIMIT='memory_units']
- | MAX_COST=float [COST_OVERCOMMIT={TRUE|FALSE}]
- [ACTIVE_STATEMENTS=integer]
- [MIN_COST=float]
- [PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}]
- [MEMORY_LIMIT='memory_units']

- https://github.com/digoal/blog/blob/master/201708/20170821_01.md

执行计划

```
postgres=# explain analyze select count(*) from t group BY c2;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Gather Motion 48:1  (slice2; segments: 48)  (cost=7469.24..7469.26 rows=1 width=16)
   Rows out:  1 rows at destination with 75 ms to end, start offset by 1.500 ms.
   -> HashAggregate  (cost=7469.24..7469.26 rows=1 width=16)
        Group By: t.c2
        Rows out:  1 rows (seg42) with 0.003 ms to first row, 27 ms to end, start offset by 30 ms.
        -> Redistribute Motion 48:48  (slice1; segments: 48)  (cost=7469.21..7469.23 rows=1 width=16)
             Hash Key: t.c2
             Rows out:  48 rows at destination (seg42) with 16 ms to end, start offset by 30 ms.
             -> HashAggregate  (cost=7469.21..7469.21 rows=1 width=16)
                  Group By: t.c2
                  Rows out:  Avg 1.0 rows x 48 workers.  Max 1 rows (seg0) with 0.006 ms to first row, 24 ms to end, start offset by 29 ms.
                  -> Seq Scan on t  (cost=0.00..6710.14 rows=3163 width=8)
                       Rows out:  Avg 3166.7 rows x 48 workers.  Max 3281 rows (seg7) with 16 ms to first row, 17 ms to end, start offset by 30 ms.
 Slice statistics:
   (slice0)    Executor memory: 327K bytes.
   (slice1)    Executor memory: 660K bytes avg x 48 workers, 660K bytes max (seg0).
   (slice2)    Executor memory: 276K bytes avg x 48 workers, 292K bytes max (seg42).
 Statement statistics:
   Memory used: 128000K bytes
 Settings:  enable_bitmapscan=off; enable_seqscan=off; optimizer=off
 Optimizer status: legacy query optimizer
 Total runtime: 77.142 ms
(22 rows)
```

https://github.com/digoal/blog/blob/master/201712/20171204_02.md

metascan+sort Key+index实践

工单ID	用户ID	订单ID	其他字段	其他字段

1、分布键（hash,随机）
1.1、分区（list, range）

2、sortKey

3、index

https://help.aliyun.com/knowledge_detail/59195.html
详细介绍

例子：
分布键：工单ID
分区键：用户ID，范围分区

where 工单id=?，扫描单个数据节点
where 用户id=?，扫描所有数据节点，单个分区
where 订单id=?，扫描所有数据节点，所有分区

metascan+sort Key+index实践

工单ID	用户ID	订单ID	其他字段	其他字段

1、分布
1.1、分区

2、sortKey

3、index

metascan+sort Key+index实践

工单ID	用户ID	订单ID	其他字段	其他字段

1、分布
1.1、分区

2、sortKey

3、index

大吞吐输出场景开发实践

https://github.com/digoal/blog/blob/master/201707/20170726_01.md

大数据并行计算，
高吞吐并行写OSS.
30MB/s/数据节点

估值计算

- 求UV（唯一值）
- 求UV增量（唯一值增量）
- HLL估值插件
- https://github.com/digoal/blog/bl

毫秒级

日UV

select count(distinct uid) from t where dt='2017-11-11';

select # hll_uid from t where dt='2017-11-11';

滑动分析：最近N天UV

SELECT date, #hll_union_agg(users) OVER seven_days

FROM daily_uniques WINDOW seven_days AS (ORDER BY date ASC ROWS 6 PRECEDING);

每日流失UV

SELECT date, (#hll_union_agg(users) OVER two_days) - #users AS lost_uniques

FROM daily_uniques WINDOW two_days AS (ORDER BY date ASC ROWS 1 PRECEDING);

Function	Operator	Example
`hll_add`	`||`	`hll_add(users, hll_hash_integer(123))` or `users || hll_hash_integer(123)` or `hll_hash_integer(123) || users`
`hll_cardinality`	`#`	`hll_cardinality(users)` or `#users`
`hll_union`	`||`	`hll_union(male_users, female_users)` or `male_users || female_users` or `female_users || male_users`

滑窗分析 - RDS PG与HDB PG都适用

- 估值滑窗(最近7天UV)
  - SELECT date, #hll_union_agg(users) OVER seven_days FROM daily_uniques **WINDOW seven_days AS (ORDER BY date ASC ROWS 6 PRECEDING)**;
- 统计滑窗(最近7天精确UV， SUM，AVG。。。)
  - SELECT date, count(distinct users) OVER seven_days, sum(x) OVER seven_days, avg(x) OVER seven_days FROM daily_uniques **WINDOW seven_days AS (ORDER BY date ASC ROWS 6 PRECEDING)**;

滑窗分析 - RDS PG与HDB PG都适用

https://github.com/digoal/blog/blob/master/201711/20171129_01.md

查看数据倾斜

- 数据分布不均匀，导致性能差、存储空间受限、木桶效应。
  - https://github.com/digoal/blog/1708/20170821_02.md

查看锁等待

- https://github.com/digoal/blog/blob/master/201705/20170521_01.md

查看数据膨胀、清理膨胀

- 堆表膨胀检测
  - https://github.com/digoal/blog/blob/master/201708/20170817_01.md
- AO表膨胀检测
  - https://github.com/digoal/blog/blob/master/201708/20170817_03.md

清理垃圾，行存、列存切换

- https://github.com/digoal/blog/blob/master/201712/20171208_04.md
- https://github.com/digoal/blog/blob/master/201708/20170817_03.md
- https://github.com/digoal/blog/blob/master/201708/20170817_01.md
- https://github.com/digoal/blog/blob/master/201608/20160815_01.md

数值类型的选择

- 如果有除法，并且需要确保精度，建议float8或numeric
- 海量数据处理，建议采用float8或int8
- 数值类型
  - numeric性能较低（内部实现的数据类型，有大量memcpy）
  - float4, float8, int, int8性能较高

连接池

- pgbouncer
  - https://github.com/digoal/blog/blob/master/201801/20180128_04.md
  - https://www.linkedin.com/pulse/scaling-greenplum-pgbouncer-sandeep-katta-/?articleId=6128769027482402816
  - https://pgbouncer.github.io/
- pgpool-II
  - http://pgpool.net/mediawiki/index.php/Main_Page

常见热门问题

- 1. HybridDB for PostgreSQL表的分布键（distribute key）是否支持设置4个列？文档上看到可以支持多列，但不知道最多可支持设置多少个列？分布键可选数据类型是否有限制？
  - 可以任意个列，类型不受限制。

- 2. HybridDB for PostgreSQL 分区表中分区个数最好设置在什么范围内（按行存储和按列存储是否有区别）？每个分区的记录数最佳实践是多少条？
  - 列存单个分区在单个节点上的记录数建议不要超过1千万条记录。
    - 例如100亿的表，规格是1024个SEGMENT，那么建议100/1024/0.1=1即不需要分区。
  - 行存单个分区在单个节点上的记录数建议不要超过200万行。
  - https://github.com/digoal/blog/blob/master/201803/20180328_01.md

常见热门问题

- 3. HybridDB for PostgreSQL 按列存储的表达到什么数量级时需要进行表压缩？
  - 建议开启压缩。

- 4. 行存表和列存表JOIN是否可行，是否有性能影响？
  - 可以，不影响性能。
  - 列存是按列存储的，如果经常访问少量列，建议使用列存。

常见热门问题

- 5. HybridDB for PostgreSQL 按列存储表压缩时， 是选择全表压缩还是选择只对某些列进行压缩？选择的标准是什么？
  - 对于不经常访问的列，建议压缩，节约存储。
  - 对于经常访问的列，如果CPU不是瓶颈，但IO是瓶颈，可以选择压缩。
  - 默认的块大小和压缩比就可以了。如果对性能不关心，只关心压缩比，可以选择“列存、最大块、压缩级别选最高”。

- 6. HybridDB for PostgreSQL 仅支持 zlib 和 RLE_TYPE 压缩算法，这两种压缩算法分别使用在什么场景下？RLE_TYPE是否只适用于有大量重复数据的列？
  - Rle_type适合重复值较多，同时紧挨着存储的列。
  - Zlib通用。

常见热门问题

- 7. HybridDB for PostgreSQL表压缩时，设置COMPRESSLEVEL时，以什么标准设置COMPRESSLEVEL的值？比如某列的数据量达到多少时，我要把COMPRESSLEVEL最好设置成4而不是2？
  - 和用户关心的压缩比有关，通常建议不设置，选择默认压缩比和性能较为均衡。

- 8. appendonly表，是否只能insert，不能update、delete。按列存、压缩的前提条件是不是设置成appendonly？
  - Appendonly表可以update、delete。
  - 列存仅支持appendonly，所以如果要选择列存，必须是AO表。

常见热门问题

- 9. 用户的事实表需要进行聚合运算，又有可能会被UPDATE（但更新频率不高），这种表怎么如何设计？（按列存但不满足APPENDONLY条件，按行存性能低）。在实际中，人社地市局有可能对操作数据进行回退\更正操作。
  - AO表支持更新和删除，可以选择列存AO表。
  - 但是需注意，对同一张表的更新和删除操作不建议并发，建议采用单线程批量更新和删除。因为目前HDB PG的更新、删除是锁表的，并发会等待。

- 10. GP建表语句中的BLOCKSIZE属性在什么情况下需要用户进行设置，通常情况下是否可使用默认值不设置？
  - 通常不需要设置，仅当用户需要非常大的压缩比时，可以使用列存，并设置blocksize为最大值。

- 11. AO表使用建议
  - AO表建议批量写入，不建议来一条，写一条。性能会非常差。而且会膨胀。
  - 对于经常有更新、删除的表，建议每天做一次vacuum。或者在更新、删除大量数据后，人为执行一下vacuum。
    - vacuum table_name;

目录

- 产品介绍
- 生态介绍
- 应用案例
- 开发、管理实践
- 数据库原理
- 参考文档

数据库物理架构

instance
database
public/other schema
pg_catalog schema
information_schema schema
table
view,func,...
system tbl\view global obj(s)
pg_default/other tablespace
pg_global tablespace
xact, log, stats, slot, seri, conf ...
WAL
归档
DataFILE(s)
VM,FSM
DIR
DataFILE(s)
VM,FSM
DIR

数据库物理架构

```
base
current_logfiles
global
log
pg_commit_ts
pg_dynshmem
pg_hba.conf
pg_ident.conf
pg_logical
pg_multixact
pg_notify
pg_replslot
pg_serial
pg_snapshots
pg_stat
pg_stat_tmp
pg_subtrans
pg_tblspc
pg_twophase
PG_VERSION
pg_wal
pg_xact
postgresql.auto.conf
postgresql.conf
postmaster.opts
postmaster.pid
```

```
cd pg_tblspc
ll

Nov  8 16:10 17912 -> /data02/pg/tbs1
```

数据库进程结构

- postmaster -- 所有数据库进程的主进程(负责监听和fork子进程)
- startup -- 主要用于数据库恢复的进程
- syslogger -- 记录系统日志
- pgstat -- 收集统计信息
- pgarch -- 如果开启了归档, 那么postmaster会fork一个归档进程.
- checkpointer -- 负责检查点的进程
- bgwriter -- 负责把shared buffer中的脏数据写入磁盘的进程
- autovacuum lanucher -- 负责回收垃圾数据的进程, 如果开启了autovacuum的话,那么postmaster会fork这个进程.
- autovacuum worker -- 负责回收垃圾数据的worker进程, 是lanucher进程fork出来的.
- bgworker -- 分为很多种worker，例如logical replication worker launcher. parallel worker. replication worker等
- wal sender -- 逻辑复制、流式物理复制的WAL发送进程
- wal receiver -- 逻辑复制、流式物理复制的WAL接收进程
- work process -- 工作进程，动态fork，例如并行计算的进程。

数据库进程结构

APP
Shared Memory Area
IPC
query(s)
Handshake
&
authentication
postmaster
autovacuum launcher
fork
autovacuum worker
parallel
workers
parallel exec
backend process
Shared buffer
bgwriter
WAL buffer
ckpt
WAL writer
wal
sender
XLOGs
wal
receiver
Datafiles
Archiver
ARCH FILEs

QUERY处理流程

https://www.postgresql.org/developer/backend/

数据页结构layout

https://www.postgresql.org/docs/de

```
postgres=# select pg_relation_filepath('aa');
 pg_relation_filepath
----------------------
 base/20699/66326
(1 row)
```

数据文件结构

One DataFile(s) Per Table or Index .
BlockID :
sequentially, 0 to 0xFFFFFFFE

page layout

PageHeaderData(24 Bytes)

ItemIdData(Array of (offset,flag,length) pairs pointing to the actual items. 4 bytes per item)

Free space(The unallocated space.
New item pointers are allocated from the start of this area, new items from the end.)

Items (The actual items themselves.)

Special space (Index access method specific data. Different methods store different data. Empty in ordinary tables.)(an access method should always initialize its pages with PageInit and then set its own opaque fields.)

page header

Field	Type	Length	Description
pd_lsn	XLogRecPtr	8 bytes	LSN: next byte after last byte of xlog record for last change to this page
pd_tli	uint16	2 bytes	TimeLineID of last change (only its lowest 16 bits)
pd_flags	uint16	2 bytes	Flag bits
pd_lower	LocationIndex	2 bytes	Offset to start of free space
pd_upper	LocationIndex	2 bytes	Offset to end of free space
pd_special	LocationIndex	2 bytes	Offset to start of special space
pd_pagesize_version	uint16	2 bytes	Page size and layout version number information
pd_prune_xid	TransactionId	4 bytes	Oldest unpruned XMAX on page, or zero if none

tuple结构

HeapTupleHeaderData(23 Bytes on most machine)

null bitmap(Optional,The null bitmap is only present
if the *HEAP_HASNULL* bit is set in t_infomask.
occupies enough bytes to have one bit per data column.
a 1 bit indicates not-null, a 0 bit is a null)

Padding(Optional, Any padding needed to make t_hoff a MAXALIGN
multiple will appear between the null bitmap and the object ID)

object ID(Optional, only present if the
HEAP_HASOID bit is set in t_infomask)

ColumnData

详情：
src/include/access/htup_details.h
https://www.postgresql.org/docs/devel/static/storage.html

tuple结构 layout

tuple header

Field	Type	Length	Description
t_xmin	TransactionId	4 bytes	insert XID stamp
t_xmax	TransactionId	4 bytes	delete XID stamp
t_cid	CommandId	4 bytes	insert and/or delete CID stamp (overlays with t_xvac)
t_xvac	TransactionId	4 bytes	XID for VACUUM operation moving a row version
t_ctid	ItemPointerData	6 bytes	current TID of this or newer row version
t_infomask2	int16	2 bytes	number of attributes, plus various flag bits
t_infomask	uint16	2 bytes	various flag bits
t_hoff	uint8	1 byte	offset to user data

TOAST介绍

- 当变长字段压缩后超过1/4个PAGE
- 转存到TOAST，**TUPLE中存储地址**
- 通过toast macro访问toast内的数据
- 一个变长字段可以存储1GB（压缩后），例如字符串，数组，bytea，varbit等

大对象介绍

- 类似TOAST的切片存储格式
- 最大存储4TB一个大对象
- 支持offset操作

思考、问题

- Oracle里有几种索引
- 分别适用什么场景，支持什么操作
- 索引引入的成本有哪些
- 每种索引有什么不适合的场景

- 遇到 模糊查询、全文检索、相似查询 怎么办
- 遇到正排、倒排、NULL怎么处理(双向链表)
- 空间数据，几何运算、距离排序
- 长字符串检索(hash)
- 计算型检索(表达式)
- 只检索某些数据如何优化(partial index)
- 多字段任意组合检索(bloom,rum,gin)
- 时序数据范围检索(range)
- Oracle全文索引的延迟？

索引接口的选择

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] name ] ON [ ONLY ] table_name [ USING method ]
    ( { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
    [ INCLUDE ( column_name [, ...] ) ]
    [ WITH ( storage_parameter = value [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ]
```

select * from pg_am;
https://www.postgresql.org/docs/devel/static/sql-createindex.html
https://github.com/digoal/blog/blob/master/201706/20170627_01.md
自动选择索引接口：
https://github.com/digoal/blog/blob/master/201706/20170617_01.md

索引接口的选择

- BTree
  - =, >, >=, <, <=、排序
- Hash (KEY值为HASH值，可以适合很长的字符串)
  - =
- GIN
  - 多值类型(数组、全文检索、枚举、网络地址类型)：包含、相交
  - JSON类型：包含 https://www.postgresql.org/docs/devel/static/datatype-json.html#JSON-INDEXING
  - 普通类型(通过btree_gin 插件支持)：与B-Tree类似
  - 字符串(通过pg_trgm 插件支持)：模糊查询、相似查询
  - 多列：任意列组合查询
- GiST
  - 空间类型：方位（上、下、左、右），空间关系（相交、包含），空间距离排序(KNN)
  - 范围数据： =, &&, <@, @>, <<, >>, -|-, &<, and &>
  - 普通类型(通过btree_gist 插件支持)：与B-Tree类似，增加空间类型类似操作符
  - 数组类型(通过intarray 插件支持)：与GIN类似
  - 多列：任意列组合查询
- SPGiST
  - 平面几何类型：与GiST类似
  - 范围类型：与GiST类似

索引接口的选择

- RUM
  - 多值类型(数组、全文检索类型)：包含、相交、相似排序
  - 普通类型：与B-Tree类似
- BRIN
  - 适合 线性数据、时序数据、（HEAP PAGE之间边界清晰的数据）。
  - 普通类型：与B-Tree类似
  - 空间类型：包含
- Bloom
  - 多列：任意列组合，等值查询
- 表达式索引
  - 搜索条件为表达式时。
  - where abs(a+b) = ?
  - create index idx on tbl ((abs(a+b)));
- 条件索引(定向索引)
  - 搜索时，强制过滤某些条件时。
  - where status='active' and col=? 。
  - create index idx on tbl (col) where status='active';

btree索引结构

- https://github.com/digoal/blog/blob/master/201605/20160528_01.md
- src/backend/access/nbtree/README

btree索引结构

类推：

0级结构最多记录285条记录
1级结构最多记录285^2条记录
2级结构最多记录285^3条记录
.....

btree索引结构

2级 索引，包括1个root page，1或多个branch page，多个leaf page。
最多存储285^3条记录。
branch page是"双向链表"。
B-Tree INDEX
meta page
root page
point upto 285 leaf page items
branch page 1
point upto 285
heap items
branch page 2
point upto 285
heap items
branch page 2
point upto 285
heap items
leaf page 1
point upto 285
heap items
leaf page 2
point upto 285
heap items
leaf page n
point upto 285
heap items
TABLE
heap page(s)

gin索引结构

- src/backend/access/gin/README
- 倒排结构
  - 多值类型 element tree
  - ctids (posting list or tree)
- 写入优化
  - element pending list
  - gin_pending_list_limit
- 大结果集查询优化 - fuzzy match
  - gin_fuzzy_search_limit

https://www.postgresql.org/docs/10/static/gin.html

gin索引结构

- https:/
  postgre
  search-

CYBERTEC
The PostgreSQL Database Company

hash索引结构

- hash值转换，hash值映射到某个bucket。
- bucket数量为2的N次方。**至少包括2个bucket。**
- metapage，page zero。包括控制信息。
- 每个bucket内**至少一个primary page**。放不下时，增加overflow page。
- hash index支持长字符串。page内存储的是HASH VALUE。
- 每个page内，hash value有序存放，支持binary search. 跨page不保证有序。
- 分裂优化，**增加bucket时，hash mapping会变化，需要分裂。**2^n映射。有一定的优化策略
- (切成4个部分，增量进行split)。
- src/backend/access/hash/README
- src/backend/utils/hash/dynahash.c

special space

double link

bitmap page:
标记overflow page 状态
(reuse,free)

gist, spgist索引结构

Geographic objects (lines, polygons) also can appear in r-tree indexes. based on their own bounding boxes.

gist, spgist索引结构

- [空间分区]通用索引结构
- r-tree base on gist
- src/backend/access/gist/README
- src/backend/access/spgist/README
- https://github.com/digoal/blog/blob/master/201708/20170824_02.md
- https://github.com/digoal/blog/blob/master/201708/20170820_01.md
- https://github.com/digoal/blog/blob/master/201709/20170905_01.md
- https://github.com/digoal/blog/blob/master/201708/20170825_01.md

brin索引结构

- src/backend/access/brin/README
- 定义粒度
  - N个连续的块
- 索引字段值在连续N个块内的边界值
  - 普通边界
  - RANGE边界
  - 空间边界（BOUND BOX）
  - PostgreSQL 11 优化(分段 bound box)
    - https://github.com/digoal/blog/blob/master/201803/20180323_05.md

其他索引结构

- rum
  - https://github.com/postgrespro/rum
- bloom
  - https://www.postgresql.org/docs/devel/static/bloom.html
- zombodb
  - https://github.com/zombodb/zombodb

cluster

Index Page Structure
Internal
Leaf
Heap
Page Header
Item
Item
Item
>= N
< F
< N
Special
E
L
A
C
G
K
M C I A G E P K W L

Page Header
Item
Item
Item
>= N
< F
< N
Special
E
L
A
C
G
K
A C D D D E G K K L

思考、问题

- 多列复合索引如何设计字段顺序
- 非驱动列查询，如何使用索引扫描

多列复合索引字段顺序原则

- https://github.com/digoal/blog/blob/master/201803/20180314_02.md

非驱动列查询的优化

- https://github.com/digoal/blog/blob/master/201803/20180323_03.md

思考

- 为什么DELETE后表还是那么大？
  - 理解水位概念

PAGE的回收、REUSE机制

- HEAP PAGE
  - 末页没有任何数据，释放真实空间。
  - FSM，见缝插针。只要有空间就可以使用。但不一定回收物理空间。
  - VACUUM回收垃圾TUPLE。
- INDEX PAGE
  - 不同索引实现不一样。（nbtree，任何情况下都不回收真实占用空间，仅做REUSE。）
  - 没有任何引用的页，REUSE。但不一定释放空间。
    - 双向链接。仅有3个ITEM的情况下，这个PAGE也不会REUSE。

例子

- create table test (id int);
- insert into test select generate_series(1,10000000);
- delete from test where ctid <> (1,1);
- vacuum test;

- -- 观察表大小
- insert into test select generate_series(1,10000000);
- delete from test where ctid <> (100,1);
- vacuum test;
- -- 观察表大小

- 为什么？

FSM 结构

- src/backend/storage/freespace/README
- https://github.com/digoal/blog/blob/master/201005/20100511_02.md
- https://github.com/digoal/blog/blob/master/201306/20130628_01.md
- 用途
  - 加速空闲页查找
  - 降低热点页

VM 结构

- 结构
  - 每个HEAP PAGE，2个比特位
  - /* Flags for bit map */
  - #define VISIBILITYMAP_ALL_VISIBLE 0x01
  - #define VISIBILITYMAP_ALL_FROZEN 0x02
  - #define VISIBILITYMAP_VALID_BITS 0x03 /* OR of all valid visibilitymap
  - * flags bits */
- 用途
  - index only scan
  - vacuum 跳过all visible， all frozen页
  - vacuum freeze跳过all frozen页

思考、问题

- 如何访问数据
- 每种访问方法的原理
- 数据库如何选择扫描方法、依据是什么

扫描方法介绍

- seqscan
- index only scan
- index scan
- bitmap scan
- ctid scan

seqscan

分批处理，并不会把shared buffer塞满

seqscan+synchronize_seqscans=on

分批处理，并不会把shared buffer塞满

https://github.com/digoal/blog/blob/master/201804/20180414_02.md

index only scan

index scan

Btree Index Scan
Index
< Key = >
< Key = >
< Key = >
Heap
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA

index skip scan

- https://github.com/digoal/blog/blob/master/201803/20180323_03.md
- 从150多毫秒，降低到了0.256毫秒

```
create table t (
  sex int,
  name text
);
insert into t select random(),
md5(random()::text) from
generate_series(1,10000000);
create index idx_t on t(sex,name);
select * from t where name='abc';
```

index skip scan

```
postgres=# explain (analyze,verbose,timing,costs,buffers) select * from t where name='abc';
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Index Only Scan using idx_t on public.t  (cost=0.56..154297.59 rows=1 width=37) (actual time=259.064..259.064 rows=0 loops=1)
   Output: sex, name
   Index Cond: (t.name = 'abc'::text)
   Heap Fetches: 0
   Buffers: shared hit=71432
 Planning time: 0.299 ms
 Execution time: 259.092 ms
(7 rows)

Time: 259.999 ms
postgres=# explain (analyze,verbose,timing,costs,buffers) select * from t where name='abc' and sex in (0,1);
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Index Only Scan using idx_t on public.t  (cost=0.56..4.45 rows=1 width=37) (actual time=0.079..0.079 rows=0 loops=1)
   Output: sex, name
   Index Cond: ((t.sex = ANY ('{0,1}'::integer[])) AND (t.name = 'abc'::text))
   Heap Fetches: 0
   Buffers: shared hit=7 read=4
 Planning time: 0.176 ms
 Execution time: 0.099 ms
(7 rows)

Time: 0.792 ms
postgres=# explain (analyze,verbose,timing,costs,buffers) select * from t where name='abc' and sex = any (array( select * from (values (0),(1)) t (sex) ));
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Index Only Scan using idx_t on public.t  (cost=0.59..17.84 rows=1 width=37) (actual time=0.059..0.059 rows=0 loops=1)
   Output: t.sex, t.name
   Index Cond: ((t.sex = ANY ($0)) AND (t.name = 'abc'::text))
   Heap Fetches: 0
   Buffers: shared hit=8
   InitPlan 1 (returns $0)
     ->  Values Scan on "*VALUES*"  (cost=0.00..0.03 rows=2 width=4) (actual time=0.002..0.003 rows=2 loops=1)
           Output: "*VALUES*".column1
 Planning time: 0.127 ms
 Execution time: 0.090 ms
(10 rows)

Time: 0.636 ms
```

bitmap scan

1、multi-index combine **OR** internal combine(GIN)

2、消除离散、重复读HEAP

bitmap scan

- index scan -> sort heap blockid -> scan heap block -> recheck index Cond.
- 优化器参考指标，相关性
- IO 放大问题消除
- SQL例子
  - https://github.com/digoal/blog/blob/master/201804/20180402_01.md

bitmap scan

```
-[ RECORD 2 ]----------+-----------------
schemaname             | public
tablename              | corr_test
attname                | c2
inherited              | f
null_frac              | 0
avg_width              | 4
n_distinct             | -0.51138
most_common_vals       | {426318,766194,85
most_common_freqs      | {6.66667e-05,6.66
histogram_bounds       | {271,106225,20156
correlation            | 0.00410469  # 线性
most_common_elems      |
most_common_elem_freqs |
elem_count_histogram   |
```

bitmap scan

- **离散扫描，每个BLOCK几乎都被重复扫描了140次，一个BLOCK刚好存储140条记录，说明这140条记录在顺序上完全离散。**
- postgres=# explain (analyze,verbose,timing,costs,buffers) select * from corr_test where c2 between 1 and 10000000;
- QUERY PLAN
- ----------
- Index Scan using idx_corr_test_2 on public.corr_test (cost=0.43..36296.14 rows=50000 width=8) (actual time=0.029..6563.525 rows=9999999 loops=1)
- Output: c1, c2
- Index Cond: ((corr_test.c2 >= 1) AND (corr_test.c2 <= 10000000))
- Buffers: shared hit=**10027095**
- Planning time: 0.089 ms
- Execution time: 7421.801 ms
- (6 rows)

bitmap scan

- 使用位图扫描，位图扫描的原理是从索引中得到**HEAP BLOCK ID**，然后按**HEAP BLOCK ID** 排序后顺序扫描。
- postgres=# explain (analyze,verbose,timing,costs,buffers) select * from corr_test where c2 between 1 and 10000000;
- QUERY PLAN
- ----------
- Bitmap Heap Scan on public.corr_test (cost=2844700.76..3038949.24 rows=10000032 width=8) (actual time=688.150..1939.259 rows=9999998 loops=1)
- Output: c1, c2
- Recheck Cond: ((corr_test.c2 >= 1) AND (corr_test.c2 <= 10000000))
- **Heap Blocks: exact=44248**
- Buffers: shared hit=71573
- -> Bitmap Index Scan on idx_corr_test_2 (cost=0.00..2842200.75 rows=10000032 width=0) (actual time=681.488..681.488 rows=9999998 loops=1)
- Index Cond: ((corr_test.c2 >= 1) AND (corr_test.c2 <= 10000000))
- **Buffers: shared hit=27325**
- Planning time: 0.147 ms
- Execution time: 2758.621 ms
- (10 rows)

ctid scan

- 根据给定行号，直接扫描HEAP PAGE。
- postgres=# explain (analyze,verbose,timing,costs,buffers) select * from car where ctid='(0,1)';
- QUERY PLAN
- ----------
- Tid Scan on public.car (cost=0.00..1.11 rows=1 width=61) (actual time=0.006..0.007 rows=1 loops=1)
- Output: id, pos, sites, rest_sites, mod_time, order_pos
- TID Cond: (car.ctid = '(0,1)'::tid)
- Buffers: shared hit=1
- Planning time: 0.183 ms
- Execution time: 0.028 ms
- (6 rows)

ctid scan

- 根据给定行号，直接扫描HEAP PAGE。
- postgres=# explain (analyze,verbose,timing,costs,buffers) select * from car where ctid = any(array['(0,1)'::tid, '(0,2)'::tid, '(100,1)'::tid]);
- QUERY PLAN
- ----------
- Tid Scan on public.car (cost=0.00..3.33 rows=3 width=61) (actual time=0.005..0.032 rows=3 loops=1)
- Output: id, pos, sites, rest_sites, mod_time, order_pos
- TID Cond: (car.ctid = ANY ('{"(0,1)","(0,2)","(100,1)"}'::tid[]))
- Buffers: shared hit=2 read=1
- Planning time: 0.182 ms
- Execution time: 0.049 ms
- (6 rows)

思考、问题

- 如何选择JOIN方法
- 有哪几种JOIN方法
- 多表JOIN如何选择JOIN顺序
- 除了穷举，还有那些方法
- 如何固定JOIN顺序
- JOIN子查询如何优化

JOIN方法介绍

- nestloop join
- merge join
- hash join

- https://momjian.us/main/writings/pgsql/performance.pdf

Nestloop Join

Nested Loop Join with Inner Sequential Scan

No Setup Required

Used For Small Tables

Nested Loop Join with Inner Index Scan

No Setup Required

Index Must Already Exist

MergeJoin

Ideal for Large Tables

An Index Can Be Used to Eliminate the Sort

HashJoin

Hash Join
Outer
Inner
aay
aag
aak
aar
aak
aas
aam
aay
aar
aao
aaw
Hashed
Must fit in Main Memory
大表
小表

ParallelHashJoin

思考、问题

- 半夜跑分析SQL很慢怎么办
- 业务高峰，如果有很耗资源的SQL怎么办

并行计算

- 并行全表扫描
- 并行索引扫描
- 并行排序
- 并行创建索引
- 并行聚合
- 并行JOIN
- 并行分区JOIN
- 并行append merge

```
postgres=# explain select count(*) from aa;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=510375.42..510375.43 rows=1 width=8)
   ->  Gather  (cost=510375.00..510375.41 rows=4 width=8)
         Workers Planned: 4
         ->  Partial Aggregate  (cost=509375.00..509375.01 rows=1 width=8)
               ->  Parallel Seq Scan on aa  (cost=0.00..507500.00 rows=750000 width=0)
(5 rows)
```

多阶段
并行聚合

https://github.com/digoal/blog/blob/master/201801/20180119_04.md

并行计算资源控制

- 单个并行节点并行度
  - #max_parallel_workers_per_gather = 2
- 全局并行度
  - #max_parallel_workers = 8
- 并行度算法
  - https://github.com/digoal/blog/blob/master/201610/20161002_01.md

强制设置并行度

- postgres=# set max_parallel_workers_per_gather =32;
- SET
- postgres=# set parallel_setup_cost =0;
- SET
- postgres=# set parallel_tuple_cost =0;
- SET
- postgres=# set min_parallel_table_scan_size =0;
- SET
- postgres=# set min_parallel_index_scan_size =0;
- SET
- postgres=# alter table a set (parallel_workers =32);
- ALTER TABLE
- postgres=# explain select count(*) from a;
- QUERY PLAN
- ----------
- Finalize Aggregate (cost=86811.94..86811.95 rows=1 width=8)
- -> Gather (cost=86811.85..86811.86 rows=32 width=8)
- Workers Planned: 32
- -> Partial Aggregate (cost=86811.85..86811.86 rows=1 width=8)
- -> Parallel Index Only Scan using a_pkey on a (cost=0.43..86030.60 rows=312500 width=0)
- (5 rows)

不设置强制，则按表、索引大小、成本、自动估算并行度

```
postgres=# show max_worker_processes ;
 max_worker_processes
----------------------
 128
(1 row)

postgres=# show max_parallel_workers;
 max_parallel_workers
----------------------
 128
(1 row)
```

思考、问题

- Oracle发现索引有膨胀怎么办？
- 发现数据膨胀，高水位，如何处理？
- 如何做到加列不堵塞DML？
- 如何实现DELETE,UPDATE limit
- 什么时候建议使用批量SQL
- 什么时间建议使用COPY

索引维护

- PostgreSQL支持重复索引（例如一个字段，可以创建多个一样的索引）
  - 在线新建索引
  - 删除旧索引

在线创建索引(不堵塞dml)

- https://www.postgresql.org/docs/devel/static/sql-createindex.html

并发创建索引
不堵塞DML的方法

- Command: CREATE INDEX
- Description: define a new index
- Syntax:
- CREATE [UNIQUE] INDEX [**CONCURRENTLY**] [[IF NOT EXISTS] name] ON table_name [USING method]
- ({ column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS { FIRST | LAST }] [, ...])
- [WITH (storage_parameter = value [, ...])]
- [TABLESPACE tablespace_name]
- [WHERE predicate]

在线加列

- 不要加默认值
- 如果有默认值，建议后台分批更新

在线回收空间

- https://github.com/reorg/pg_repack/

- https://github.com/digoal/blog/blob/master/201610/20161030_02.md

批量DML

- https://github.com/digoal/blog/blob/master/201704/20170424_05.md
- 批量插入
  - insert into tbl values (),(),...();
  - copy
- 批量更新
  - update tbl from tmp set x=tmp.x where tbl.id=tmp.id;
- 批量删除
  - delete from tbl using tmp where tmp.id=tbl.id;

DELETE\UPDATE limit

- delete from tbl where ctid=any(array(
- select ctid from tbl where xxxx limit n
-));

- update from tbl set xx=xx where ctid=any(array(
- select ctid from tbl where xxxx limit n
-));

思考、问题

- 如何分析性能瓶颈

数据库优化

- 擒贼先擒王（TOP SQL）
- SQL执行计划分析、优化
- 数据库内核代码层瓶颈分析
- 慢SQL定位、优化
- 数据库活动信息跟踪参数
- 锁等待问题定位
- 等待事件透视
- 全面健康报告
- 9种索引接口的适应场景与选择
- 复合索引顺序
- 非驱动列扫描优化
- 优化器开关介绍（因子、开关）
- HINT的使用

性能分析利器1 - TOP SQL

pg_stat_statements
插件

View "public.pg_stat_statements"

Column	Type	Collation
userid	oid	
dbid	oid	
queryid	bigint	
query	text	
calls	bigint	
total_time	double precision	
min_time	double precision	
max_time	double precision	
mean_time	double precision	
stddev_time	double precision	
rows	bigint	
shared_blks_hit	bigint	
shared_blks_read	bigint	
shared_blks_dirtied	bigint	
shared_blks_written	bigint	
local_blks_hit	bigint	
local_blks_read	bigint	
local_blks_dirtied	bigint	
local_blks_written	bigint	
temp_blks_read	bigint	
temp_blks_written	bigint	
blk_read_time	double precision	
blk_write_time	double precision	

性能分析利器1 - TOP SQL

- https://github.com/digoal/blog/blob/master/201704/20170424_06.md
- create extension pg_stat_statements;

- **最耗时 SQL (整体数据库优化，建议以此结果逐条优化，效果立竿见影)**
- 总最耗时 SQL TOP 5
- select dbid, query from pg_stat_statements order by total_time desc limit 5;

- **其他**
  - **最耗IO SQL**
  - 总最耗IO SQL TOP 5
  - select dbid, query from pg_stat_statements order by (blk_read_time+blk_write_time) desc limit 5;

  - **响应时间抖动最严重 SQL**
  - select dbid, query from pg_stat_statements order by stddev_time desc limit 5;

  - **最耗共享内存 SQL**
  - select dbid, query from pg_stat_statements order by (shared_blks_hit+shared_blks_dirtied) desc limit 5;

  - **最耗临时空间 SQL**
  - select dbid, query from pg_stat_statements order by temp_blks_written desc limit 5;

性能分析利器2 - explain

- https://www.postgresql.org/docs/10/static/sql-explain.html
- https://github.com/digoal/blog/blob/master/201704/20170424_06.md
- HDB PG
- https://github.com/digoal/blog/blob/master/201712/20171204_02.md

设置需要TRACE的开关。
查看详细的输出，逐个分析。
最重要的是分析
explain
(analyze,verbose,timing,costs,buffers的结果)

```
EXPLAIN [ ( option [, ...] ) ] statement
EXPLAIN [ ANALYZE ] [ VERBOSE ] statement

where option can be one of:

    ANALYZE [ boolean ]
    VERBOSE [ boolean ]
    COSTS [ boolean ]
    BUFFERS [ boolean ]
    TIMING [ boolean ]
    SUMMARY [ boolean ]
    FORMAT { TEXT | XML | JSON | YAML }
```

性能分析利器2 - explain

- set client_min_messages=debug5;
- set log_checkpoints = on;
- set log_error_verbosity = verbose ;
- set log_lock_waits = on;
- set log_replication_commands = off;
- set log_temp_files = 0;
- set track_activities = on;
- set track_counts = on;
- set track_io_timing = on;
- set track_functions = 'all';
- set trace_sort=on;
- set log_statement_stats = off;
- set log_parser_stats = on;
- set log_planner_stats = on;
- set log_executor_stats = on;
- set log_autovacuum_min_duration=0;
- set deadlock_timeout = '1s';
- set debug_print_parse = off;
- set debug_print_rewritten = off;
- set debug_print_plan = off;
- set debug_pretty_print = on;
- explain (analyze,verbose,timing,costs,buffers) select count(*),relkind from pg_class group by relkind order by count(*) desc limit 1;

设置需要TRACE的开关。
查看详细的输出，逐个分析。
最重要的是分析explain (analyze,verbose,timing,costs,buffers的结果)

数据库优化

- 擒贼先擒王（TOP SQL）
- SQL执行计划分析、优化
- 数据库内核代码层瓶颈分析
- 慢SQL定位、优化
- 数据库活动信息跟踪参数
- 锁等待问题定位
- 等待事件透视
- 全面健康报告
- 9种索引接口的适应场景与选择
- 复合索引顺序
- 非驱动列扫描优化
- 优化器开关介绍（因子、开关）
- HINT的使用

性能分析利器3 - perf

- perf
- https://github.com/digoal/blog/blob/master/201611/20161129_01.md
- 例子(分区过多导致的性能问题)
  - https://github.com/digoal/blog/blob/master/201801/20180124_01.md

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9482 root      20   0 1504m  52m 6508 S 46.1  0.0   3611:23 [redacted]
52630 [redacted] 20   0  120g 291m 249m S 25.6  0.1   0:08.30 postgres: [redacted] BIND
52660 [redacted] 20   0  120g 287m 248m S 25.6  0.1   0:04.39 postgres: [redacted] BIND
53155 [redacted] 20   0  120g 288m 249m S 25.6  0.1   0:04.80 postgres: [redacted] BIND
53180 [redacted] 20   0  120g 318m 249m S 25.6  0.1   0:07.54 postgres: [redacted] BIND
52369 [redacted] 20   0  120g 306m 249m S 23.9  0.1   0:04.97 postgres: [redacted] BIND
```

```
+      5.29%   postgres  postgres                    [.] AllocSetAlloc
+      5.09%   postgres  postgres                    [.] hash_search_with_hash_value
+      4.92%   postgres  postgres                    [.] SearchCatCache
+      3.70%   postgres  postgres                    [.] _bt_checkkeys
+      2.50%   postgres  postgres                    [.] pglz_decompress
+      1.87%   postgres  [kernel.kallsyms]           [k] _spin_lock_irq
+      1.86%   postgres  [kernel.kallsyms]           [k] _spin_lock_irqsave
+      1.73%   postgres  [kernel.kallsyms]           [k] _spin_lock
```

性能分析利器3 - Oprofile

- https://github.com/digoal/blog/blob/master/201505/20150509_01.md
- https://github.com/digoal/blog/blob/master/201601/20160125_01.md

```
[root@digoal data06]# opreport -l -f -g -w -x -t 1 /opt/pgsql/bin/postgres
Using /data06/oprofile_data/samples/ for samples directory.
CPU: Intel Core/i7, speed 1995.14 MHz (estimated)
Counted CPU_CLK_UNHALTED events (Clock cycles when not halted) with a unit mask of 0x00 (No unit mask) count 100000
vma      samples  %        linenr info                 symbol name
007827a0 2091381  26.7572  /opt/soft_bak/postgresql-9.4.1/src/backend/utils/time/tqual.c:1116 HeapTupleSatisfiesVacuum
00490300 988600   12.6482  /opt/soft_bak/postgresql-9.4.1/src/backend/access/heap/pruneheap.c:174 heap_page_prune
0078a8c0 698665    8.9387  /opt/soft_bak/postgresql-9.4.1/src/port/qsort.c:104 pg_qsort
0058afb0 676022    8.6491  /opt/soft_bak/postgresql-9.4.1/src/backend/commands/vacuumlazy.c:1728 vac_cmp_itemptr
0058baf0 385039    4.9262  /opt/soft_bak/postgresql-9.4.1/src/backend/commands/vacuumlazy.c:172 lazy_vacuum_rel
004c4d00 365497    4.6762  /opt/soft_bak/postgresql-9.4.1/src/backend/access/transam/xlog.c:844 XLogInsert
00675420 229805    2.9401  /opt/soft_bak/postgresql-9.4.1/src/backend/storage/page/bufpage.c:415 itemoffcompare
00675d20 184668    2.3626  /opt/soft_bak/postgresql-9.4.1/src/backend/storage/page/bufpage.c:433 PageRepairFragmentation
0078a7e0 169808    2.1725  /opt/soft_bak/postgresql-9.4.1/src/port/qsort.c:78 swapfunc
00655590 147647    1.8890  /opt/soft_bak/postgresql-9.4.1/src/backend/storage/buffer/bufmgr.c:1898 BufferGetBlockNumber
00488940 139389    1.7833  /opt/soft_bak/postgresql-9.4.1/src/backend/access/heap/heapam.c:5756 heap_prepare_freeze_tuple
007624d0 86239     1.1033  /opt/soft_bak/postgresql-9.4.1/src/backend/utils/hash/dynahash.c:824 hash_search_with_hash_val
```

性能分析利器3 - dtract\stap

- stap
  - https://github.com/digoal/blog/blob/master/201310/20131017_01.md
  - https://www.postgresql.org/docs/current/static/dynamic-trace.html

性能分析利器3 - strace\pstack

- strace
- pstack

数据库优化

- 擒贼先擒王（TOP SQL）
- SQL执行计划分析、优化
- 数据库内核代码层瓶颈分析
- 慢SQL定位、优化
- 数据库活动信息跟踪参数
- 锁等待问题定位
- 等待事件透视
- 全面健康报告
- 9种索引接口的适应场景与选择
- 复合索引顺序
- 非驱动列扫描优化
- 优化器开关介绍（因子、开关）
- HINT的使用

性能分析利器4 - 当前慢SQL

- 运行中慢SQL
  - select * from pg_stat_activity where now()-query_start > interval '?s';
- 长运行中事务
  - select * from pg_stat_activity where state='active' and now()-xact_start > interval '?s';
- 长空闲事务
  - select * from pg_stat_activity where state='idle in transaction' and now()-xact_start > interval '?s';
- 长2PC事务
  - select * from pg_prepared_xacts where now()-prepared > interval '?s';

杀死当前慢SQL

- 如果数据库因为慢SQL耗尽资源，接近Hang死，杀死当前慢SQL通常立竿见影
  - superuser可以杀死所有用户的BACKEND
  - rds superuser可以杀死所有普通用户的BACKEND
  - 杀死所有用户，执行时间超过10秒的慢SQL
    - select pg_terminate_backend(pid) from pg_stat_activity where now()-query_start >= interval '10 s';
  - 杀死当前用户，执行时间超过10秒的慢SQL
    - select pg_terminate_backend(pid) from pg_stat_activity where usename=current_user and now()-query_start >= interval '10 s';

性能分析利器4 -历史慢SQL

AWR

https://github.com/digoal/blog/blob/master/201604/20160421_01.md

性能分析利器5 -跟踪慢SQL为什么慢?

- auto_explain
  - https://www.postgresql.org/docs/devel/static/auto-explain.html
- 慢SQL执行计划详情
  - plan
  - node time
  - buffers, hints
  - filter

auto_explain会打印慢SQL执行计划详情

```
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=16494388.74..16494388.94 rows=10 width=12) (actual time=153006.839..153006.839 rows=0 loops=1)
   Output: id, (count(DISTINCT c1))
   Buffers: shared hit=95131 read=1041241, temp read=220164 written=220195
   ->  GroupAggregate  (cost=16494388.74..18494388.72 rows=99999999 width=12) (actual time=153006.838..153006.838 rows=0 loops=1)
         Output: id, count(DISTINCT c1)
         Group Key: t1.id
         Filter: (count(*) > 1)
         Rows Removed by Filter: 100000000
         Buffers: shared hit=95131 read=1041241, temp read=220164 written=220195
         ->  Sort  (cost=16494388.74..16744388.74 rows=99999999 width=8) (actual time=47686.082..62240.952 rows=100000000 loops=1)
               Output: id, c1
               Sort Key: t1.id
               Sort Method: external merge  Disk: 1761312kB
               Buffers: shared hit=95123 read=1041241, temp read=220164 written=220195
               ->  Seq Scan on public.t1  (cost=0.00..2386364.00 rows=99999999 width=8) (actual time=0.011..26439.157 rows=100000000 loops=1)
                     Output: id, c1
                     Filter: (t1.c2 <> 'abc'::text)
                     Buffers: shared hit=95123 read=1041241
 Planning time: 0.092 ms
 Execution time: 153357.601 ms
(20 rows)
```

数据库优化

- 擒贼先擒王（TOP SQL）
- SQL执行计划分析、优化
- 数据库内核代码层瓶颈分析
- 慢SQL定位、优化
- 数据库活动信息跟踪参数
- 锁等待问题定位
- 等待事件透视
- 全面健康报告
- 9种索引接口的适应场景与选择
- 复合索引顺序
- 非驱动列扫描优化
- 优化器开关介绍（因子、开关）
- HINT的使用

性能分析利器6 - 跟踪活动日志

- log_autovacuum_min_duration = 0
  - vacuum,analyze活动，消耗
- log_checkpoints = on
  - 跟踪检查点开销
- log_lock_waits=on
  - 跟踪锁等待
- lock_timeout=1s
  - 锁等待时间阈值
- track_io_timing = on
  - 跟踪IO耗时
- track_counts = on
  - 计数器, pg_stat, pg_statio依赖

数据库优化

- 擒贼先擒王（TOP SQL）
- SQL执行计划分析、优化
- 数据库内核代码层瓶颈分析
- 慢SQL定位、优化
- 数据库活动信息跟踪参数
- 锁等待问题定位
- 等待事件透视
- 全面健康报告
- 9种索引接口的适应场景与选择
- 复合索引顺序
- 非驱动列扫描优化
- 优化器开关介绍（因子、开关）
- HINT的使用

性能分析利器7 - 查看当前锁等待

- 查看锁等待，谁堵塞了谁
- https://github.com/digoal/blog/blob/master/201705/20170521_01.md
- https://github.com/digoal/blog/blob/master/201806/20180622_02.md

- Greenplum(HDB PG) segment级锁问题排查
- https://github.com/digoal/blog/blob/master/201806/20180620_01.md
- https://github.com/digoal/blog/blob/master/201708/20170822_01.md

性能分析利器7 - 查看当前锁等待

```
postgres=# select * from v_locks_monitor ;
-[ RECORD 1 ]-+-----------------------------------------------------------------------------------------------------------
locktype      | relation
datname       | postgres
relation      | locktest
page          |
tuple         |
virtualxid    |
transactionid |
classid       |
objid         |
objsubid      |
string_agg    | Pid: 23043
              | Granted: false , Mode: AccessExclusiveLock , FastPath: false , VirtualTransaction: 4/1450064 , Session_St
              | Username: postgres , Database: postgres , Client_Addr: NULL , Client_Port: -1 , Application_Name: psql
              | Xact_Start: 2017-05-21 21:43:43.735829+08 , Query_Start: 2017-05-21 21:43:50.965797+08 , Xact_Elapse: 00:
              | Query: truncate locktest ;
              | --------
              | Pid: 40698
              | Granted: true , Mode: RowExclusiveLock , FastPath: false , VirtualTransaction: 6/1031925 , Session_State:
              | Username: postgres , Database: postgres , Client_Addr: NULL , Client_Port: -1 , Application_Name: psql
              | Xact_Start: 2017-05-21 21:43:15.173798+08 , Query_Start: 2017-05-21 21:43:24.338804+08 , Xact_Elapse: 00:
              | Query: insert into locktest values (2,'test');
              | --------
              | Pid: 17515
              | Granted: true , Mode: RowExclusiveLock , FastPath: false , VirtualTransaction: 3/5671759 , Session_State:
              | Username: postgres , Database: postgres , Client_Addr: NULL , Client_Port: -1 , Application_Name: psql
              | Xact_Start: 2017-05-21 21:42:19.199124+08 , Query_Start: 2017-05-21 21:42:47.820125+08 , Xact_Elapse: 00:
              | Query: select * from locktest ;
              | --------
              | Pid: 17515
```

性能分析利器7 - 查看当前锁等待

```
| --------
| Pid: 24781
| Granted: false , Mode: AccessShareLock , FastPath: false , VirtualTransaction: 7/1025270 , Session_State:
| Username: postgres , Database: postgres , Client_Addr: NULL , Client_Port: -1 , Application_Name: psql
| Xact_Start: 2017-05-21 21:44:20.725834+08 , Query_Start: 2017-05-21 21:44:20.725834+08 , Xact_Elapse: 00:
| Query: select * from locktest ;
```

性能分析利器7 - 当前等待事件透视

https://www.postgresql.org/docs/devel/static/monitoring-stats.html#MONITORING-STATS-VIEWS

```
postgres=# select usename, wait_event_type, wait_event, count(*) from
pg_stat_activity group by 1,2,3 order by count(*) desc;
 usename  | wait_event_type |     wait_event      | count
----------+-----------------+---------------------+-------
 postgres | Activity        | LogicalApplyMain    |     1
          | Activity        | BgWriterHibernate   |     1
          | Activity        | WalWriterMain       |     1
          | Activity        | CheckpointerMain    |     1
          | Activity        | AutoVacuumMain      |     1
 postgres |                 |                     |     1
 postgres | Activity        | LogicalLauncherMain |     1
(7 rows)
```

数据库优化

- 擒贼先擒王（TOP SQL）
- SQL执行计划分析、优化
- 数据库内核代码层瓶颈分析
- 慢SQL定位、优化
- 数据库活动信息跟踪参数
- 锁等待问题定位
- 等待事件透视
- 全面健康报告
- 9种索引接口的适应场景与选择
- 复合索引顺序
- 非驱动列扫描优化
- 优化器开关介绍（因子、开关）
- HINT的使用

性能分析利器8 - AWR

- PostgreSQL
  - https://github.com/digoal/blog/blob/master/201611/20161123_01.md
- PPAS
  - https://github.com/digoal/blog/blob/master/201606/20160628_01.md
- Cloud DBA
  - 敬请期待

IO、CPU飙升分析、优化

- OS
  - cgroup 检查throtte
  - top find PID
  - iotop find PID
  - iostat， 是否到达瓶颈、IOHANG问题
  - dmesg
- DB
  - pid -> find pg_stat_activity
  - freeze 风暴问题
    - https://github.com/digoal/blog/blob/master/201804/20180411_01.md
    - https://github.com/digoal/blog/blob/master/201801/20180117_03.md
    - https://github.com/digoal/blog/blob/master/201606/20160612_01.md
    - https://github.com/digoal/blog/blob/master/201605/20160520_01.md
  - autovacuum_naptime , 长事务, 长 2PC, long sql, hot_standby_feedback(备库长事务, 长 2PC, long sql) , vacuum_defer_cleanup_age
    - https://github.com/digoal/blog/blob/master/201704/20170410_03.md
    - https://github.com/digoal/blog/blob/master/201505/20150503_01.md

数据库优化

- 擒贼先擒王（TOP SQL）
- SQL执行计划分析、优化
- 数据库内核代码层瓶颈分析
- 慢SQL定位、优化
- 数据库活动信息跟踪参数
- 锁等待问题定位
- 等待事件透视
- 全面健康报告
- 9种索引接口的适应场景与选择
- 复合索引顺序
- 非驱动列扫描优化
- 优化器开关介绍（因子、开关）
- HINT的使用

索引接口的选择

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] name ] ON [ ONLY ] table_name [ USING method ]
    ( { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
    [ INCLUDE ( column_name [, ...] ) ]
    [ WITH ( storage_parameter = value [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ]
```

select * from pg_am;
https://www.postgresql.org/docs/devel/static/sql-createindex.html
https://github.com/digoal/blog/blob/master/201706/20170627_01.md
自动选择索引接口：
https://github.com/digoal/blog/blob/master/201706/20170617_01.md

索引接口的选择

- BTree
  - =, >, >=, <, <=、排序
- Hash (KEY值为HASH值，可以适合很长的字符串)
  - =
- GIN
  - 多值类型(数组、全文检索、枚举、网络地址类型)：包含、相交
  - JSON类型：包含 https://www.postgresql.org/docs/devel/static/datatype-json.html#JSON-INDEXING
  - 普通类型(通过btree_gin 插件支持)：与B-Tree类似
  - 字符串(通过pg_trgm 插件支持)：模糊查询、相似查询
  - 多列：任意列组合查询
- GiST
  - 空间类型：方位（上、下、左、右），空间关系（相交、包含），空间距离排序(KNN)
  - 范围数据： =, &&, <@, @>, <<, >>, -|-, &<, and &>
  - 普通类型(通过btree_gist 插件支持)：与B-Tree类似，增加空间类型类似操作符
  - 数组类型(通过intarray 插件支持)：与GIN类似
  - 多列：任意列组合查询
- SPGiST
  - 平面几何类型：与GiST类似
  - 范围类型：与GiST类似

索引接口的选择

- RUM
  - 多值类型(数组、全文检索类型)：包含、相交、相似排序
  - 普通类型：与B-Tree类似
- BRIN
  - 适合 线性数据、时序数据、（HEAP PAGE之间边界清晰的数据）。
  - 普通类型：与B-Tree类似
  - 空间类型：包含
- Bloom
  - 多列：任意列组合，等值查询
- 表达式索引
  - 搜索条件为表达式时。
  - where abs(a+b) = ?
  - create index idx on tbl ((abs(a+b)));
- 条件索引(定向索引)
  - 搜索时，强制过滤某些条件时。
  - where status='active' and col=? 。
  - create index idx on tbl (col) where status='active';

思考、问题

- 多列复合索引如何设计字段顺序
- 非驱动列查询，如何使用索引扫描

多列复合索引字段顺序原则

- https://github.com/digoal/blog/blob/master/201803/20180314_02.md

非驱动列查询的优化

- https://github.com/digoal/blog/blob/master/201803/20180323_03.md

数据库优化

- 擒贼先擒王（TOP SQL）
- SQL执行计划分析、优化
- 数据库内核代码层瓶颈分析
- 慢SQL定位、优化
- 数据库活动信息跟踪参数
- 锁等待问题定位
- 等待事件透视
- 全面健康报告
- 9种索引接口的适应场景与选择
- 复合索引顺序
- 非驱动列扫描优化
- 优化器开关介绍（因子、开关）
- HINT的使用

优化器开关介绍（因子、开关）

- 优化器开关、成本计算系数因子
  - https://www.postgresql.org/docs/devel/static/runtime-config-query.html
- 校准因子
  - https://github.com/digoal/blog/blob/master/201311/20131126_03.md

SQL HINT

- HINT介绍
  - https://github.com/digoal/blog/blob/master/201607/20160723_02.md
- 阿里云pg_hint_plan插件使用
  - https://github.com/digoal/blog/blob/master/201604/20160401_01.md

数据库优化

- 擒贼先擒王（TOP SQL）
- SQL执行计划分析、优化
- 数据库内核代码层瓶颈分析
- 慢SQL定位、优化
- 数据库活动信息跟踪参数
- 锁等待问题定位
- 等待事件透视
- 全面健康报告
- 9种索引接口的适应场景与选择
- 复合索引顺序
- 非驱动列扫描优化
- 优化器开关介绍（因子、开关）
- HINT的使用
- 函数、存储过程调用优化

plpgsql函数(存储过程)性能诊断、优化

- auto_explain插件
- plpgsql函数中每一个调用的详细执行计划
  - https://github.com/digoal/blog/blob/master/201611/20161121_02.md

数据库优化

- 擒贼先擒王（TOP SQL）
- SQL执行计划分析、优化
- 数据库内核代码层瓶颈分析
- 慢SQL定位、优化
- 数据库活动信息跟踪参数
- 锁等待问题定位
- 等待事件透视
- 全面健康报告
- 9种索引接口的适应场景与选择
- 复合索引顺序
- 非驱动列扫描优化
- 优化器开关介绍（因子、开关）
- HINT的使用
- 函数、存储过程调用优化
- **连接池**

连接池的优化

4.2、 1亿数据量 TPC-B

连接数	社区版本TPS	postgres buildin_pool版本TPS
64	81460	84356
512	72805	81388
2048	10902	80029
8192	7729	79281
19900	971	81247

连接池的优化

客户端连接数据库
postmaster
session_pool_size>0
session_pool_ports=0
fork process
连postgres库
走dedicate server
其他，走dedicate server
dedicate
server
backend
process
backend
process
backend
process
one process per session
shared
server
backend
process
backend
process
backend
process
one process for multi-session

连接池的优化

- pgbouncer
  - https://github.com/digoal/blog/blob/master/201005/20100511_03.md
- 内置连接池
  - https://github.com/digoal/blog/blob/master/201805/20180521_03.md
- Greenplum 连接池实践
  - https://github.com/digoal/blog/blob/master/201801/20180128_04.md
- 阿里云高并发版本
  - https://github.com/digoal/blog/blob/master/201805/20180505_07.md

HDB PG数据倾斜、SQL倾斜监测、优化

- https://github.com/digoal/blog/blob/master/201708/20170821_02.md
- https://github.com/digoal/blog/blob/master/201711/20171123_01.md
- https://github.com/digoal/blog/blob/master/201607/20160719_02.md

思考、问题

- Oracle PL/SQL函数如何调试
- 如何诊断函数内部执行SQL的性能

plpgsql函数性能诊断

- auto_explain
- plpgsql函数中每一个调用的详细执行计划
- https://github.com/digoal/blog/blob/master/201611/20161121_02.md

思考、问题

- 什么情况下可能出现雪崩
- 如何防止雪崩
- 雪崩的紧急处理方法

防雪崩

- https://github.com/digoal/blog/blob/master/201712/20171211_02.md
- statement_timeout
  - 语句超时，防止雪崩
- lock_timeout
  - 锁超时
- deadlock_timeout
  - 死锁超时
- idle_in_transaction_session_timeout
  - 空闲中事务超时

DDL操作建议 - 防止DDL锁等待引发雪崩

- 锁等待机制介绍
  - https://github.com/digoal/blog/blob/master/201705/20170521_01.md

- DDL、大锁建议
  - begin;
  - set lock_timeout='ns';
  - DDL
  - end;

AB 表切换

- https://github.com/digoal/blog/blob/master/201807/20180725_04.md

限制慢SQL并发度

- 杀掉最近发起的慢SQL，老的慢SQL继续，保证N个慢SQL并发
  - select pg_terminate_backend(pid) from pg_stat_activity where now()-query_start > interval **'? second'** order by query_start offset **$N**;
  - 或 pg_cancel_backend(pid)
  - https://www.postgresql.org/docs/devel/static/functions-admin.html#FUNCTIONS-ADMIN-SIGNAL

杀会话、杀QUERY

- 杀会话
  - select **pg_terminate_backend**(pid);
    - 杀某个会话
  - select pg_terminate_backend(pid) from pg_stat_activity where pg_backend_pid()<>pid;
    - 杀所有会话
    - 杀某个用户的所有会话
      - select pg_terminate_backend(pid) from pg_stat_activity where usename=? and pid<>pg_backend_pid();
- 杀QUERY
  - select **pg_cancel_backend**($pid);

杀会话、杀QUERY

Name	Return Type	Description
pg_cancel_backend(*pid* int)	boolean	Cancel a backend's current query. This is also allowed if the calling role is a member of the role whose backend is being canceled or the calling role has been granted pg_signal_backend, however only superusers can cancel superuser backends.
pg_reload_conf()	boolean	Cause server processes to reload their configuration files
pg_rotate_logfile()	boolean	Rotate server's log file
pg_terminate_backend(*pid* int)	boolean	Terminate a backend. This is also allowed if the calling role is a member of the role whose backend is being terminated or the calling role has been granted pg_signal_backend, however only superusers can terminate superuser backends.

WAL(xlog)文件结构

- https://github.com/digoal/blog/blob/master/201302/20130223_01.md

```
Fixed-size header (XLogRecord struct)
XLogRecordBlockHeader struct
XLogRecordBlockHeader struct
...
XLogRecordDataHeader[Short|Long] struct
block data
block data
...
main data
```

CLOG(xact)文件结构

- 结构
  - 每个事务对应2个BIT位
  - #define TRANSACTION_STATUS_IN_PROGRESS 0x00
  - #define TRANSACTION_STATUS_COMMITTED 0x01
  - #define TRANSACTION_STATUS_ABORTED 0x02
  - #define TRANSACTION_STATUS_SUB_COMMITTED 0x03
- 用途
  - 识别事务状态
  - tuple header - hint bit(查询是可能被设置)，避免访问clog bits
    - https://github.com/digoal/blog/blob/master/201509/20150905_01.md

MVCC

- 多版本
  - tuple header(xid, 事务状态)
- 事务快照
  - 当前未提交事务，已分配最大事务
- 可见性判断
  - 判断要素
  - 事务隔离级别，事务快照，TUPLE header, hint bit, clog

锁

```
/*
 * LOCKTAG is the key information needed to look up a LOCK item in the
 * lock hashtable.  A LOCKTAG value uniquely identifies a lockable object.
 *
 * The LockTagType enum defines the different kinds of objects we can lock.
 * We can handle up to 256 different LockTagTypes.
 */
typedef enum LockTagType
{
        LOCKTAG_RELATION,                       /* whole relation */
        /* ID info for a relation is DB OID + REL OID; DB OID = 0 if shared */
        LOCKTAG_RELATION_EXTEND,        /* the right to extend a relation */
        /* same ID info as RELATION */
        LOCKTAG_PAGE,                           /* one page of a relation */
        /* ID info for a page is RELATION info + BlockNumber */
        LOCKTAG_TUPLE,                          /* one physical tuple */
        /* ID info for a tuple is PAGE info + OffsetNumber */
        LOCKTAG_TRANSACTION,            /* transaction (for waiting for xact done) */
        /* ID info for a transaction is its TransactionId */
        LOCKTAG_VIRTUALTRANSACTION, /* virtual transaction (ditto) */
        /* ID info for a virtual transaction is its VirtualTransactionId */
        LOCKTAG_SPECULATIVE_TOKEN,      /* speculative insertion Xid and token */
        /* ID info for a transaction is its TransactionId */
        LOCKTAG_OBJECT,                         /* non-relation database object */
        /* ID info for an object is DB OID + CLASS OID + OBJECT OID + SUBID */

        /*
         * Note: object ID has same representation as in pg_depend and
         * pg_description, but notice that we are constraining SUBID to 16 bits.
         * Also, we use DB OID = 0 for shared objects such as tablespaces.
         */
        LOCKTAG_USERLOCK,                       /* reserved for old contrib/userlock code */
        LOCKTAG_ADVISORY                        /* advisory user locks */
} LockTagType;
```

锁

```
/*
 * These are the valid values of type LOCKMODE for all the standard lock
 * methods (both DEFAULT and USER).
 */

/* NoLock is not a lock mode, but a flag value meaning "don't get a lock" */
#define NoLock                                  0

#define AccessShareLock                 1       /* SELECT */
#define RowShareLock                    2       /* SELECT FOR UPDATE/FOR SHARE */
#define RowExclusiveLock                3       /* INSERT, UPDATE, DELETE */
#define ShareUpdateExclusiveLock 4      /* VACUUM (non-FULL),ANALYZE, CREATE INDEX
                                                                         * CONCURRENTLY */
#define ShareLock                               5       /* CREATE INDEX (WITHOUT CONCURRENTLY) */
#define ShareRowExclusiveLock   6       /* like EXCLUSIVE MODE, but allows ROW
                                                                         * SHARE */
#define ExclusiveLock                   7       /* blocks ROW SHARE/SELECT...FOR UPDATE */
#define AccessExclusiveLock             8       /* ALTER TABLE, DROP TABLE, VACUUM FULL,
                                                                         * and unqualified LOCK TABLE */
```

HOT原理

Heap Page
PageHead
LinePoint 1
LinePoint 2
Free Space
Tuple2
HEAP_HOT_TUPLE
Tuple1
t_ctid, HEAP_HOT_UPDATED
Index Page
PageHead
ItemPoint

HOT原理

Heap Page
PageHead
LP_REDIRECT
LinePoint 1
LinePoint 2
Free Space
Tuple2
HEAP_HOT_TUPLE
Tuple1
Dead,vacuumed
Index Page
PageHead
ItemPoint

事务隔离

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

垃圾回收

- 计数器
  - track_counts = on
- 扫描间隔
  - autovacuum_naptime = 1min
- autovacuum worker process数
  - autovacuum_max_workers = 3
- 垃圾回收worker process休息间隔
  - autovacuum_vacuum_cost_delay = 0ms
  - #autovacuum_vacuum_cost_limit = -1
- 扫描哪些PAGE?
  - /* Flags for bit map */
  - #define VISIBILITYMAP_ALL_VISIBLE 0x01
  - #define VISIBILITYMAP_ALL_FROZEN 0x02
  - #define VISIBILITYMAP_VALID_BITS 0x03 /* OR of all valid visibilitymap
  - * flags bits */
- 可回收哪些垃圾?
  - 比GetOldestXmin更老的垃圾

```
vacuum_cost_delay = 0                    # 0-100 milliseconds
#vacuum_cost_page_hit = 1                # 0-10000 credits
#vacuum_cost_page_miss = 10              # 0-10000 credits
#vacuum_cost_page_dirty = 20             # 0-10000 credits
#vacuum_cost_limit = 200                 # 1-10000 credits
```

事务号冻结

UINT32
xactid
循环使用

zheap存储引擎

- 引擎扩展接口
  - https://commitfest.postgresql.org/17/1283/
- datafile
  - inplace update
  - 即刻回收
- undo file
  - 旧版本

压缩接口

- https://commitfest.postgresql.org/17/1294/
- 内置 pg_lzcompress
  - src/common/pg_lzcompress.c
- lz4
  - https://github.com/digoal/blog/blob/master/201803/20180315_02.md
- zstd
  - https://github.com/digoal/blog/blob/master/201803/20180315_01.md

FDW接口介绍

https://wiki.postgresql.org/wiki/Fdw

数据库逻辑结构

PostgreSQL实例
xxx.xxx.xxx.x
xx:port
数据库
db1
db2
schema
public
userschema1
public
对象
table1
table2
view1
func1
sequence1

数据库权限体系

实例权限	• 修改pg_hba.conf
数据库权限	• grant赋予是否允许连接或创建schema的权限 • revoke回收
schema权限	• grant赋予允许查询schema中的对象,或在schema中创建对象 • revoke回收
object权限	• grant赋予 • revoke回收
表空间	• grant赋予允许在对应表空间创建表，物化视图，索引，临时表 • revoke回收

数据库认证管理

- 有哪些认证方法
  - password, ldap, ……
  - https://www.postgresql.org/docs/10/authentication.html
- ACL
- 认证配置
  - pg_hba.conf

20.3.1. Trust Authentication
20.3.2. Password Authentication
20.3.3. GSSAPI Authentication
20.3.4. SSPI Authentication
20.3.5. Ident Authentication
20.3.6. Peer Authentication
20.3.7. LDAP Authentication
20.3.8. RADIUS Authentication
20.3.9. Certificate Authentication
20.3.10. PAM Authentication
20.3.11. BSD Authentication


```
local      database  user  auth-method  [auth-options]
host       database  user  address  auth-method  [auth-options]
hostssl    database  user  address  auth-method  [auth-options]
hostnossl  database  user  address  auth-method  [auth-options]
host       database  user  IP-address  IP-mask  auth-method  [auth-options]
hostssl    database  user  IP-address  IP-mask  auth-method  [auth-options]
hostnossl  database  user  IP-address  IP-mask  auth-method  [auth-options]
```

数据库认证管理

Roles
Connection Limit
Auth Method
(Trust,
Password,
Ident,
LDAP…)
PostgreSQL
PG_HBA
Listene
Which
Address
TYPE DATABASE USER CIDR-ADDRESS METHOD

数据库可靠性介绍

PostgreSQL调用OS的sync WRITE函数.
wal_sync_method=?

open_datasync (write WAL files with open() option O_DSYNC)

fdatasync (call fdatasync() at each commit)

fsync (call fsync() at each commit)

fsync_writethrough (call fsync() at each commit, forcing write-through of any disk write cache)

open_sync (write WAL files with open() option O_SYNC)

时间点恢复介绍

1.写WAL(或叫XLOG)
2.为了确保数据库在recovery的时候,可以恢复到一个一致的状态,shared buffer中的脏数据页在flush到磁盘数据文件中之前,应该确保这个脏页的改变量已经write through 到XLOG文件了.
3.如何确保先写XLOG再改变DATA-PAGE呢?PAGE头信息里面包含了一个pd_lsn位,用于记录XLOG写该PAGE信息的最后一个字节的下一个字节.
4. 在写脏页到数据文件前只要确保大于或等于pd_lsn的XLOG已经write through到磁盘了就行.
lsn :
log sequence number --- in practice, a WAL file location

主备切换，激活，时间线

支持跨时间线恢复

异步流复制

Standby
2
archlog
xlog
1
WalRcvData->receivedUpto
startup
process
postmaster
wal receiver
Primary
PM_SHUTDOWN_2
PMSignal
send
3
WalRcvData->conninfo
WalRcvData->receiveStart
wal sender
recover
fork
read
datafiles
postmaster
XLOG
WAL buffer
write
fork
read
write
APP
Other backend processes
wal writer

同步流复制

- 同步流复制原理

1. Many standbys
2. Only one sync standby at the same time
3. Sync wide:
   Cluster
   Database
   User
   Session
   Transaction

多副本介绍

- #synchronous_commit = on
  - # synchronization level;
  - # **off, local, remote_write, remote_apply, or on**
  - **介绍每个参数的区别**

- #synchronous_standby_names = ''
  - # standby servers that provide sync rep
  - # method to choose sync standbys, number of sync standbys,
  - # and comma-separated list of application_name
  - # from standby(s); '*' = all
  - **[FIRST] *num_sync* (*standby_name* [, ...])**
  - **ANY *num_sync* (*standby_name* [, ...])**
  - ***standby_name* [, ...]**

多副本介绍

- https://github.com/digoal/blog/blob/master/201803/20180326_01.md

TPC-B 性能

```
latency average = 1.005 ms
latency stddev = 1.494 ms
tps = 55695.800213 (including
tps = 55700.524316 (excluding
```

```
latency average = 1.248 ms
latency stddev = 4.118 ms
tps = 44840.272218 (including
tps = 44853.145859 (excluding
```

OLTP写场景本地持久化 VS 多副本性能对比：

1、多副本模式，RT 提高了0.24毫秒，约20%

2、多副本模式，TPS 下降了10855，约20%

多副本0丢失、无脑裂HA切换流程

- https://github.com/digoal/blog/blob/master/201803/20180326_01.md
- 1、主库心跳异常
- 2、控制N（同步备库数 - 同步副本数 + 1）个或以上备库、冻结接收新的REDO
- 3、选出接收到最多REDO的备库
- 4、修改其他备库配置，使用该备库为新主库
- 5、激活该备库
- 6、漂移VIP、或修改DNS、或修改中间件（视现场HA架构而定）
- 7、原主库恢复后，修改配置，切换为备库，使用该备库为新主库

逻辑订阅流程

https://github.com/digoal/blog/blob/master/201506/20150616_02.md
(原理：PG逻辑订阅已封装)

逻辑订阅(增量复制)

(发布some tables)
publisher A
catalog
WAL
logical decoder
sender
流复制协议
订阅端 1
slot A
receiver
applyer
feedback
slot_name(每个订阅端有一个slog)
catalog xmin (catalog 快照, 逻辑decoder依赖)
restart_lsn(幂等)
plugin(decoder)
confirmed_flush_lsn (feed back by 订阅端)
发布->订阅(一对多)
slot_name->订阅(一对一)
表->发布(多对多)

HA

- 共享存储
- 双节点异步流复制
- 三节点同步流复制

持续备份

- 基础备份+归档
- 基础备份+增量+归档
- 备库+ZFS快照+归档

异地容灾

- 物理流复制

监控、日常维护

健康检查

- 数据库高频监控指标
- https://github.com/digoal/blog/blob/master/201806/20180613_02.md
- 系统监控指标
- https://github.com/digoal/blog/blob/master/201806/20180613_03.md
- 数据库低频监控指标
- https://github.com/digoal/blog/blob/master/201806/20180613_04.md

- AWR报告(PPAS)
  - https://github.com/digoal/blog/blob/master/201606/20160628_01.md
- CLOUD DBA
  - 敬请期待

健康检查

- 资源队列使用监控(HDB PG)
  - https://github.com/digoal/blog/blob/master/201708/20170821_01.md
- 资源队列使用监控(PPAS)
  - https://github.com/digoal/blog/blob/master/201801/20180113_01.md
- 锁等待监控
  - https://github.com/digoal/blog/blob/master/201705/20170521_01.md

维护操作

- CANCEL QUERY
- 杀会话
- 收缩膨胀对象
- 收集统计信息
- 自动收集统计信息的参数

排错

解读错误代码

- 输出详细错误(包括代码位置)
  - **postgres=# set log_error_verbosity=verbose;**
  - **SET**
  - **或**
  - postgres=# **\set VERBOSITY verbose**
  - postgres=# select a;
  - ERROR: 42703: column "a" does not exist
  - LINE 1: select a;
  - ^
  - LOCATION: errorMissingColumn, parse_relation.c:3293
- 追踪出错源码内容，找错误原因
  - parse_relation.c:3293
- 错误代码解读
  - https://www.postgresql.org/docs/devel/static/errcodes-appendix.html
  - https://github.com/digoal/blog/blob/master/201807/20180711_01.md

core dump

- 配置CORE文件输出
  - https://github.com/digoal/blog/blob/master/201611/20161121_01.md
  - kernel.core_pattern= /data01/corefiles/core_%e_%u_%t_%s.%p
- 编译项
  - -g -ggdb -fno-omit-frame-pointer
- gdb --core=xxx.corefile $postgres_绝对路径
  - bt
  - …

pstack

- /proc/$pid/stack

异常、BUG报告

- 上报要点
  - 描述清楚问题、现象。
  - 描述清楚环境（操作系统、内核版本、多少位）。
  - 数据库版本、涉及插件时，描述清楚插件版本。
  - 复现步骤 。
- 如何上报BUG
  - https://www.postgresql.org/docs/current/static/bug-reporting.html
  - https://www.postgresql.org/account/submitbug/
- 开发组邮件列表
  - https://lists.postgresql.org/

优化

- 见前面

PostgreSQL 基因

JSON HINT GIS数据处理 多维分析 FDW数据泵 时序数据处理
异步消息 音频处理 立体几何 平面几何 基因处理 图数据处理 分词
旋转门压缩 数据走势预测 正则走索引 下棋 估值计算 探索宇宙 3D打印
GPU并行计算 流式计算 机器学习 文本挖掘 对接ES\Kafka\... MR LBS应用

一切皆可扩展 - 取自最终用户开源项目(pgxn, github, pgfoundry, sourceforge, 社区.....)

成熟稳定的插件
整合进PG内核

历经43年进化 - 相当成熟的底盘技术

开放式接口

AGG\WINDOW
HOOK language handler
类型扩展(IO,OP,AM,FUNC)
FDW handler scan handler
支持数十种流行编程语言
编写服务端函数

SQL-2011 递归查询 MVCC 优化器
ECPG 窗口查询 HASH JOIN 秒杀
9种索引方法,20种大类
bt,hash,gin,brin,rum
bloom,gist,sp-gist,...

SQL兼容性

可靠性
REDO
流式复制
多副本
块级增量备份&PITR

扩展性
CPU并行计算
读写分离
水平分库
多主同步

PostgreSQL 基因

- 需求：
- 内核开发者
  - 代码质量、可塑性
- 数据库厂商
  - 开源许可，开源生态，版本演进节奏（PG每年一个大版本）
- 最终用户管理员
  - 垂直、水平扩展能力，稳定性、诊断工具、可靠性、安全性、性能
- 最终用户应用开发者
  - SQL标准，DB端编程能力，功能，开发语言支持，框架支持，数据库扩展能力（大多数插件是最终用户贡献的）

PostgreSQL 基因

- 特点：
- 开放式架构
  - 扩展：**类型、操作符、函数、索引接口、函数语言、外部数据源、数据采样、自定义WAL、存储接口、压缩算法接口...**
  - 例子：gis, 全文检索, 图搜索, 流计算, 推荐引擎, 点云, 路由, citus MPP DB, HLL估值, CMS-TOP估值, 图像特征搜索，列存储，Oracle兼容包，可连接地球上所有数据源的FDW，加密类型，DB端python|r|java|perl|lua|go|cuda|opencl...存储过程、逻辑订阅、物理订阅、9种索引接口（多值、单值、时序、空间、范围、文本、JSON等类型加速，多字段任意检索加速），GPU加速接口(pg_strom)，数组、RANGE、平面几何、立体几何、网络、ISBN、等类型扩展，模糊、相似查询等。。。

- SQL标准
  - SQL:2011
  - 大多数开源数据库兼容SQL:92，相差20年。

- SQL标准扩展
  - 组权限管理、INLINE CODE、异步消息、DB端编程语言扩展、DB端操作符扩展、DB端全文检索、数据聚集语法、DB端扩展数据源。。。

- HTAP
  - 支持高并发OLTP业务，同时支持CPU、IO密集型OLAP业务。

- 优化器
  - JOIN（nestloop, hash, merge），多表JOIN遗传优化算法，merge sort支持并行排序和快速OFFSET，向量计算，并行计算（排序、建索引、JOIN、扫描、FILTER、聚合、...），算子复用、计算下推（外部数据源支持下推计算、JOIN、sort, filter, select...）

PostgreSQL 基因

- 存储
  - HEAP + TOAST存储(变长字段超过1/4个BLOCK时压缩存储到切片，使得单行较小)
  - 内置压缩
  - 超大字段索引可以选择HASH INDEX
- 索引效率
  - 记录数再多，索引扫描效率依旧平稳
  - metapage->rootpage->branchpage->leafpage->HEAPpage

  - 相比较而言，B+Tree结构存储引擎，记录数较多后，性能下降严重。
    - 原因，
    - 1. 二级索引需要扫描多棵树。全离散扫描，性能差。
    - 2. 不支持hash index的话，大字段导致索引层级非常多，离散扫描的BLOCK数更加多。
    - 3. 写入索引字段离散数据时，非常容易导致表内BLOCK分裂，IO放大问题严重。

PostgreSQL 基因

```
NODE:
        case T_SeqScan:
        case T_SampleScan:
        case T_IndexScan:
        case T_IndexOnlyScan:
        case T_BitmapHeapScan:
        case T_TidScan:
        case T_SubqueryScan:
        case T_FunctionScan:
        case T_TableFuncScan:
        case T_ValuesScan:
        case T_CteScan:
        case T_NamedTuplestoreScan:
        case T_WorkTableScan:
        case T_ForeignScan:
        case T_CustomScan:
        case T_ModifyTable:
        case T_Result:
        case T_ProjectSet:
        case T_ModifyTable:
                case CMD_INSERT:
                case CMD_UPDATE:
                case CMD_DELETE:
        case T_Append:
        case T_MergeAppend:
        case T_RecursiveUnion:
        case T_BitmapAnd:
        case T_BitmapOr:
        case T_NestLoop:
        case T_MergeJoin:
        case T_HashJoin:
        case T_SeqScan:
        case T_SampleScan:
        case T_Gather:
        case T_GatherMerge:
        case T_IndexScan:
        case T_IndexOnlyScan:
        case T_BitmapIndexScan:
        case T_BitmapHeapScan:
        case T_TidScan:
        case T_SubqueryScan:
        case T_FunctionScan:
        case T_TableFuncScan:
        case T_ValuesScan:
        case T_CteScan:
        case T_NamedTuplestoreScan:
        case T_WorkTableScan:
        case T_ForeignScan:
                case CMD_SELECT:
                case CMD_INSERT:
                case CMD_UPDATE:
                case CMD_DELETE:
        case T_CustomScan:
        case T_Material:
        case T_Sort:
        case T_Group:
        case T_Agg:
                case AGG_PLAIN:
                case AGG_SORTED:
                case AGG_HASHED:
                case AGG_MIXED:
        case T_WindowAgg:
        case T_Unique:
        case T_SetOp:
                case SETOP_SORTED:
                case SETOP_HASHED:
        case T_LockRows:
        case T_Limit:
        case T_Hash:
```

Greenplum 数据库原理

目录

- 产品介绍
- 生态介绍
- 应用案例
- 开发、管理实践
- 数据库原理
- 参考文档

Thanks

- 案例大全
- 数据库原理
- 数据库发展方向、数据库选型
- 问题诊断、性能分析与优化
- 开发技巧
- 备份恢复
- 安全、审计
- DBA技巧

RDS PG+HybridDB PG+OSS
海量实时处理
-离线分析
-域名
规整
金融应用
平安
模糊查询、相似搜索、正则搜索
域名搜索
导购
-实时盗文
全文检索
业务平台
物联网
边缘计算
流式处理
订单实时状态聚合
社交业务
探探
图式搜索
企业图谱
销售助手
-风控
独立事件分析
舆情分析
商品最佳促销组合
冷热分离
异步消息
海量异步监控
多值类型、图像特征值相似搜索
图搜
实时数据清洗
车联网
GIS应用
末端轨迹、自动配送
高德
智能
院
任意字段实时搜索
CRM
任意维度数据透视、钻取
-设备、会员透视、精细化运营
时间、空间、对象多维搜索、透视
售-选址、线上线下地推

Thanks

https://yq.aliyun.com/articles/98539
如来神掌：
https://github.com/digoal/blog/blob/master/201706/20170601_02.md
致DBA、开发者、内核开发者、架构师：
https://github.com/digoal/blog/blob/master/201611/20161101_01.md

开发规约

- https://github.com/digoal/blog/blob/master/201609/20160926_01.md

资料分享

- **# RDS PG, HDB PG 阿里云入口**
- **RDS PG：https://www.aliyun.com/product/rds/postgresql**
- **HDB PG：https://www.aliyun.com/product/gpdb**
- **RDS PG 和 HDB PG产品选型指南：https://github.com/digoal/blog/blob/master/201709/20170918_02.md**

- **# 阿里云RDS PG, HDB PG 产品文档**
- RDS PG 产品文档：https://help.aliyun.com/document_detail/26152.html
- HDB PG 产品文档：https://help.aliyun.com/document_detail/49912.html
- RDS PG OSS 外部表文档：https://help.aliyun.com/knowledge_detail/43352.html
- HDB PG OSS 外部表文档：https://help.aliyun.com/document_detail/35457.html

资料分享

- # 图形化 实例管理
- RDS PG 图形化管理客户端pgadmin：https://www.pgadmin.org/download/
- HDB PG 图形化管理客户端pgadmin：https://www.postgresql.org/ftp/pgadmin/pgadmin3/v1.6.3/

- # PG 开发驱动
- PG 各种开发语言客户端驱动：https://www.postgresql.org/docs/10/static/external-interfaces.html
- PG GO语言 驱动：https://github.com/jackc/pgx
- RDS PG jdbc 驱动：https://jdbc.postgresql.org
- RDS PPAS jdbc 驱动：https://www.enterprisedb.com/advanced-downloads
- HDB PG jdbc 驱动：https://help.aliyun.com/document_detail/35428.html
- .NET PG驱动: http://www.npgsql.org/index.html

资料分享

- # 社区reference文档
- RDS PG 10 社区官方手册：https://www.postgresql.org/docs/10/static/index.html
- RDS PG 9.4 社区官方手册：https://www.postgresql.org/docs/9.4/static/index.html
- RDS PPAS 手册：https://www.enterprisedb.com/resources/product-documentation
- HDB PG 社区官方手册：http://greenplum.org/docs/
- HDB PG 8.2 社区官方文档：https://www.postgresql.org/docs/8.2/static/

资料分享

- **# PG 开发指南**
- Java： https://github.com/digoal/blog/blob/master/201701/20170106_05.md
- PHP： https://github.com/digoal/blog/blob/master/201701/20170106_08.md
- Ruby： https://github.com/digoal/blog/blob/master/201701/20170106_07.md
- Python： https://github.com/digoal/blog/blob/master/201701/20170106_06.md
- C： https://github.com/digoal/blog/blob/master/201701/20170106_09.md

- **# PostgreSQL 开发实践：**
- http://www.postgresqltutorial.com/
- https://www.tutorialspoint.com/postgresql/index.htm

资料分享

- # GIS 相关文档
- PostGIS 文档：http://postgis.net/docs/manual-2.4/
- PostGIS 入门：http://workshops.boundlessgeo.com/postgis-intro/
- PostGIS 例子：http://revenant.ca/www/postgis/workshop/indexing.html
- pgrouting文档：http://pgrouting.org/documentation.html
- QGIS 可视化GIS软件：http://www.qgistutorials.com/en/
- uDig 可视化GIS软件：http://udig.refractions.net/
- GIS 开源项目汇总：http://www.osgeo.org/
- ArcGIS

资料分享

- # 可视化分析软件
- Orange3 可视化分析软件：https://orange.biolab.si/screenshots/
- https://github.com/biolab/orange3
- 拖拽式、算法可扩展

例子

File
SVM
Naive Bayes
Random Forest Classification
Test & Score
Confusion Matrix (SVM)
Confusion Matrix (Naive Bayes)
Confusion Matrix (RF)
Venn Diagram
Scatter Plot
Confusion Matrix (SVM)
Learners
SVM
Naive Bayes
Random Forest Classificatic
Show
Number of instances
Select
Select Correct
Select Misclassified
Clear Selection
Output
Predicted
Actual
1 2 3 Σ
1 58 1 0 59
2 0 70 1 71
3 0 1 47 48
Σ 58 72 48 178
Confusion Matrix (Naive Bayes)
Learners
SVM
Naive Bayes
Random Forest Classificatic
Predicted
Actual
1 2 3 Σ
1 59 0 0 59
2 1 69 1 71
3 0 0 48 48
Σ 60 69 49 178
Confusion Matrix (RF)
Learners
SVM
Naive Bayes
Random Forest Classificatic
Show
Number of instances
Select
Select Correct
Select Misclassified
Clear Selection
Output
Predicted
Actual
1 2 3 Σ
1 58 1 0 59
2 1 67 3 71
3 0 0 48 48
Σ 59 68 51 178
Venn Diagram
Info
3 data sets on input
Data Instance Identifiers
Use instance equality
Use identifiers
Data set: SVM
Data set: Naive Bayes
Data set: Random Forest Classification
Data set #4
Data set #5
SVM
3
Naive Bayes
2
2
0
0
1
0
1
Home
Screenshots
Download
Docs
Blog
Training
Donate
sualizing misclassifications.
Finding common misclassifications of three predictive models.

资料分享

- **# PostgreSQL 认证学习、考试**
  - https://www.enterprisedb.com/training/postgres-certification
  - 考试攻略
    - https://yq.aliyun.com/articles/464038
  - 推荐理由：
    - 去O最热门产品
    - PostgreSQL社区持续贡献的公司之一
    - 阿里云合作伙伴

资料分享

- **# 最佳实践**
- 流式计算、在线业务、数据分析 业务闭环方案：https://github.com/digoal/blog/blob/master/201707/20170728_01.md
- 流计算模块：https://www.pipelinedb.com/
- UPSERT 用法：https://github.com/digoal/blog/blob/master/201704/20170424_04.md
- HDB PG 批量更新（Upsert）方法：https://yq.aliyun.com/articles/86604
- HDB PG 连接池管理1：https://greenplum.org/docs/admin_guide/access_db/topics/pgbouncer.html
- HDB PG 连接池管理2：https://www.linkedin.com/pulse/scaling-greenplum-pgbouncer-sandeep-katta-/?articleId=6128769027482402816
- 分页和评估（游标 或 PK+上一次最大位点开始）：https://github.com/digoal/blog/blob/master/201605/20160506_01.md
- HyperLogLog的使用：https://github.com/digoal/blog/blob/master/201608/20160825_02.md
- sharding 开发框架：
- https://github.com/dangdangdotcom/sharding-jdbc
- https://github.com/go-pg/sharding/
- 分析库选型测试与分析(分析数据库VS)：https://www.atatech.org/articles/90599
- RDS PG,HDB PG案例大全(开发者的《如来神掌》)：
- https://github.com/digoal/blog/blob/master/201706/20170601_02.md
- https://yq.aliyun.com/topic/118
- http://tms-preview.taobao.com/wh/tms/ali/page/markets/aliyun/yq/topic/postgresql

资料分享

- **# SQL 机器学习相关**
- MADlib SQL 机器学习库：http://madlib.apache.org/documentation.html
- https://pypi.python.org/pypi/pymadlib/0.1.4
- https://github.com/pivotalsoftware/PivotalR
- https://cran.r-project.org/web/packages/PivotalR/PivotalR.pdf
- https://cran.r-project.org/web/packages/PivotalR/vignettes/pivotalr.pdf

资料分享

- **# 日常维护、性能诊断**
- RDS PG TOP SQL 分析：https://github.com/digoal/blog/blob/master/201704/20170424_06.md
- RDS PG 自定义函数内SQL性能诊断：https://github.com/digoal/blog/blob/master/201611/20161121_02.md
- 锁等待 分析：https://github.com/digoal/blog/blob/master/201705/20170521_01.md
- HDB PG 慢SQL 诊断：https://www.postgresql.org/docs/10/static/sql-explain.html
- RDS PG 慢SQL 诊断：https://www.postgresql.org/docs/8.3/static/sql-explain.html
- HDB PG 数据倾斜监测：https://github.com/digoal/blog/blob/master/201708/20170821_02.md
- HDB PG 表膨胀监测：
- https://github.com/digoal/blog/blob/master/201708/20170817_01.md
- https://github.com/digoal/blog/blob/master/201708/20170817_03.md
- HDB PG 分区键的选择：https://github.com/digoal/blog/blob/master/201607/20160719_02.md
- HDB PG 负载管理(资源队列管理)：https://github.com/digoal/blog/blob/master/201708/20170821_01.md
- RDS PPAS AWR报告：https://github.com/digoal/blog/blob/master/201606/20160628_01.md
- 阿里云RDS PG, PPAS, HDB PG日常健康监控：
- https://github.com/digoal/blog/blob/master/201709/20170913_01.md
- HDB PG 行存、列存，堆表、AO表的原理和选择：
  https://github.com/digoal/blog/blob/master/201708/20170818_02.md

资料分享

- **# Oracle兼容性**
- https://github.com/digoal/blog/blob/master/class/21.md
- Oracle业务适合用PostgreSQL去O的一些评判标准：
- https://github.com/digoal/blog/blob/master/201707/20170713_02.md
- PostgreSQL Oracle 兼容性之 - 函数、类型、多国语言映射：
  https://github.com/digoal/blog/blob/master/201702/20170217_01.md
- Oracle 迁移到 PG
- https://wiki.postgresql.org/wiki/Oracle_to_Postgres_Conversion
- Oracle PL/SQL 迁移到 PG plpgsql
- https://www.postgresql.org/docs/10/static/plpgsql-porting.html

资料分享

- **# TO 开发者、DBA、架构师 文档**
- https://github.com/digoal/blog/blob/master/201611/20161101_01.md
- **# 培训视频下载**
- http://pan.baidu.com/s/1pKVCgHX

digoal's 微信

培训视频

文档合集

问卷调查

- 最希望数据库提供哪些功能
- 对阿里云RDS PG,HDB PG,PPAS,POLARDB PG有一些什么疑问、需求
- 对社区版本PG，GPDB有什么疑问、需求
- 业务相关问题
  - 行业背景，应用场景，TP OR AP
  - 痛点有哪些
  - 当前痛点有没有解法？是怎么解决的？