

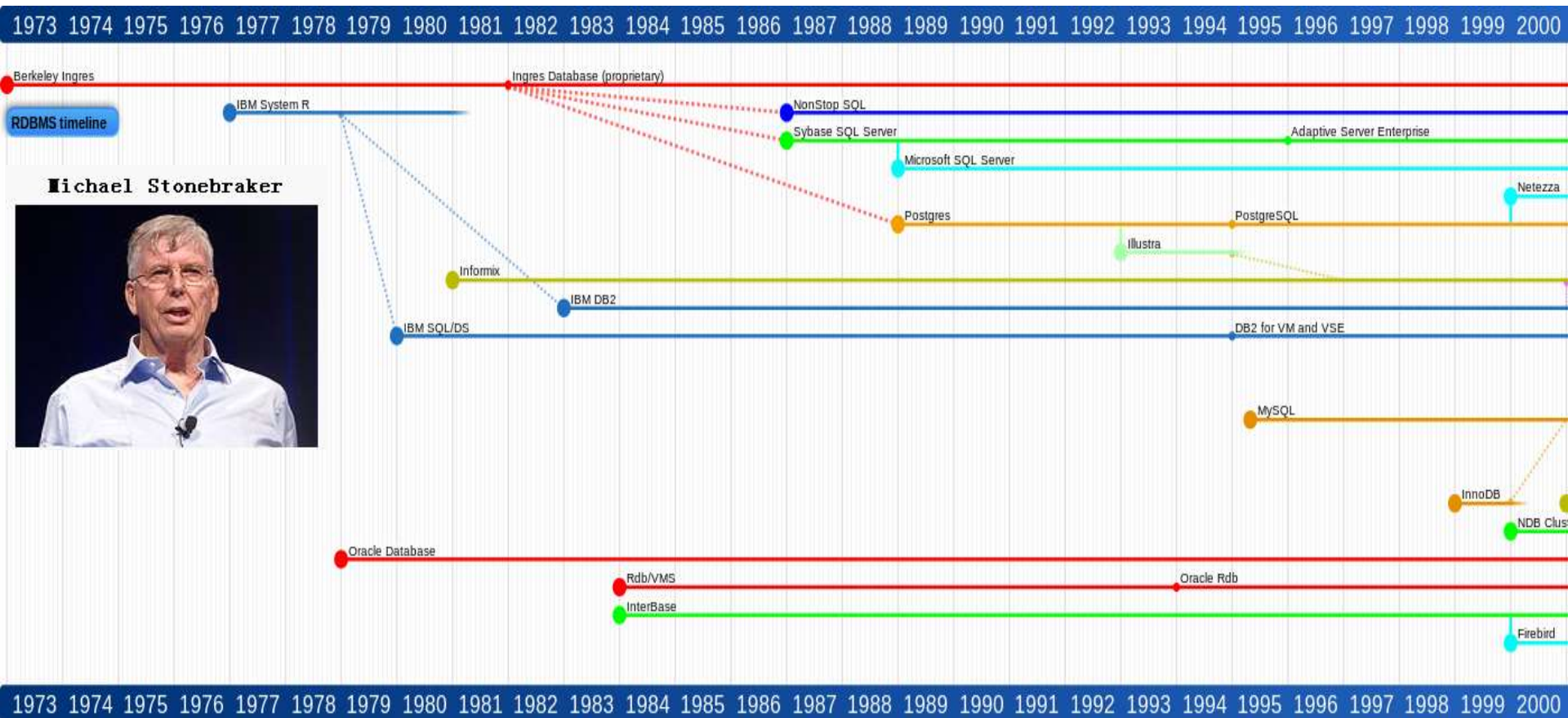
阿里云 ApsaraDB PostgreSQL、 PPAS、HDB for PG 产品指导、开发实践、应用案例

阿里云
digoal

目录

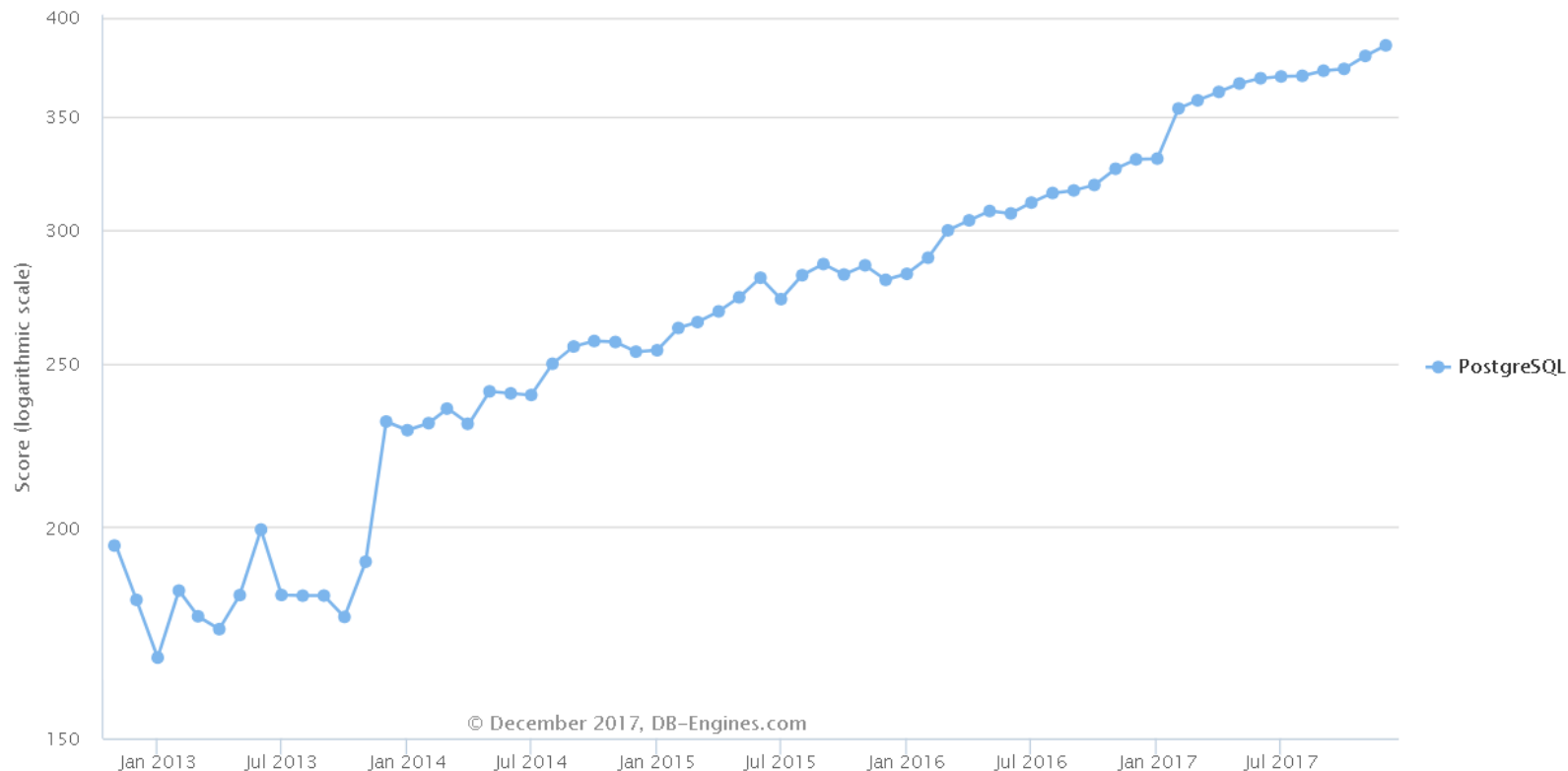
- 产品介绍
- 生态介绍
- 应用案例
- 开发实践
- 参考文档

PostgreSQL 起源

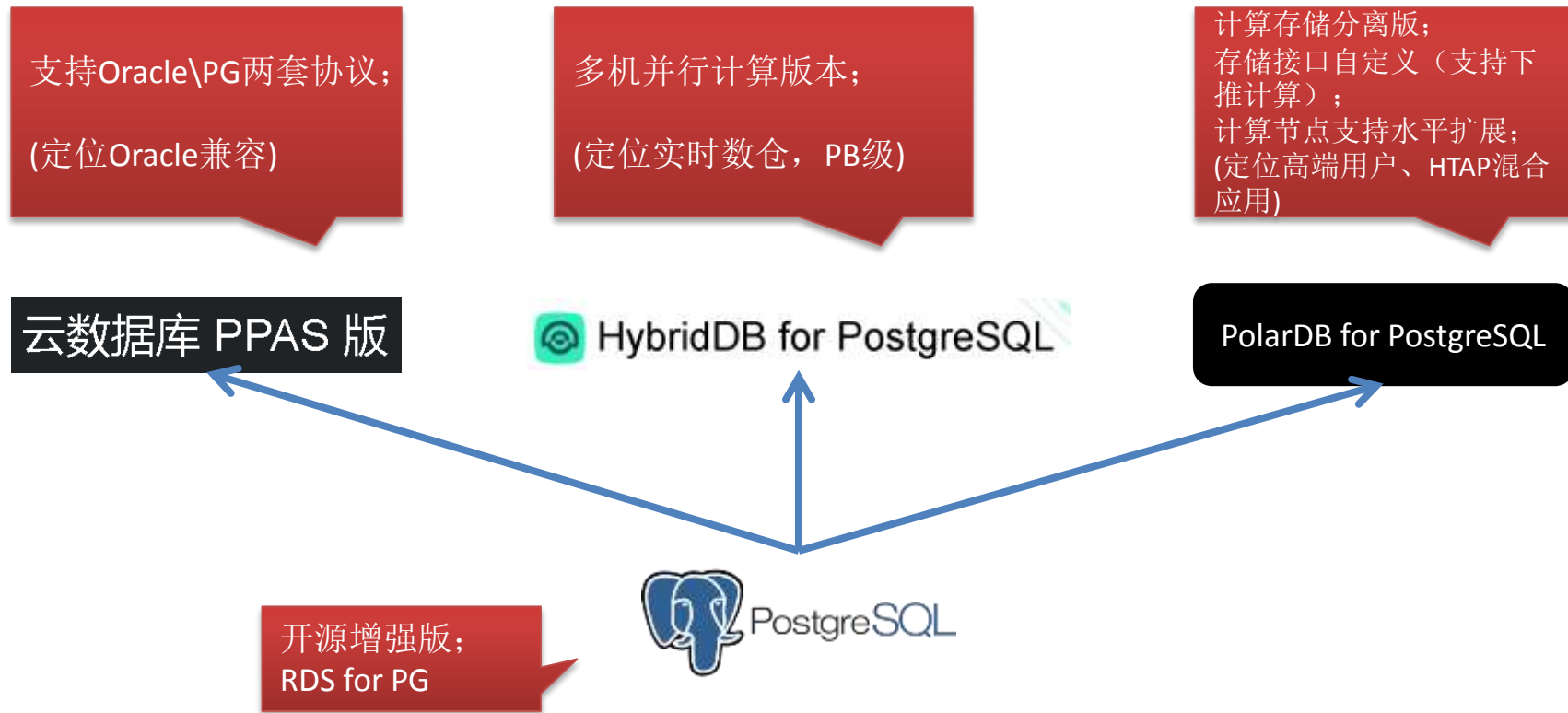


PostgreSQL 趋势

DB-Engines Ranking of PostgreSQL



阿里云PostgreSQL产品体系



产品体系 - RDS for PG



PostgreSQL



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

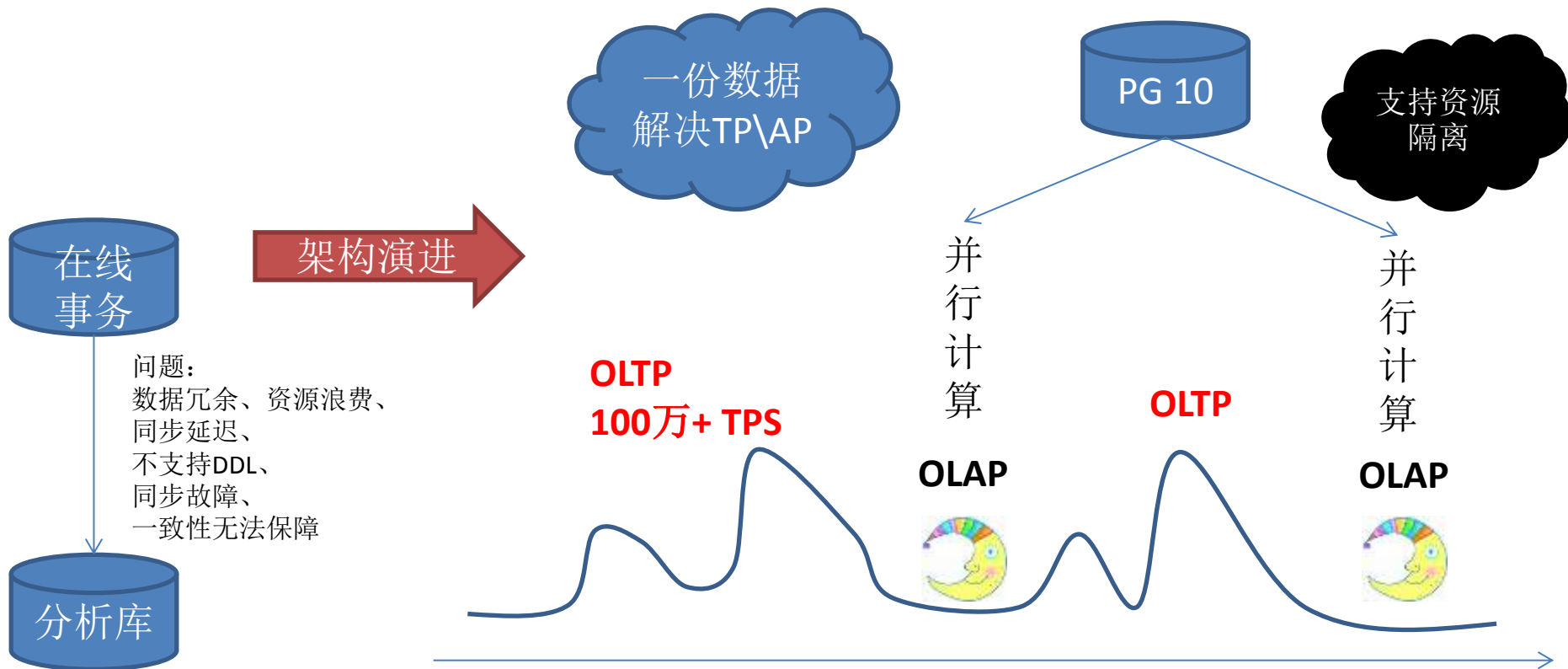
基于PostgreSQL的FDW功能，阿里云深度整合优化了对OSS云存储的外部表管理功能，可以支持2TB以上存储空间无限扩展。



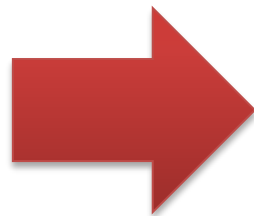
支持数据仓库

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。

RDS PG 10 HTAP (ALL IN ONE)



RDS PG 10 HTAP (**All In One**)



产品体系 - RDS for PG



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



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

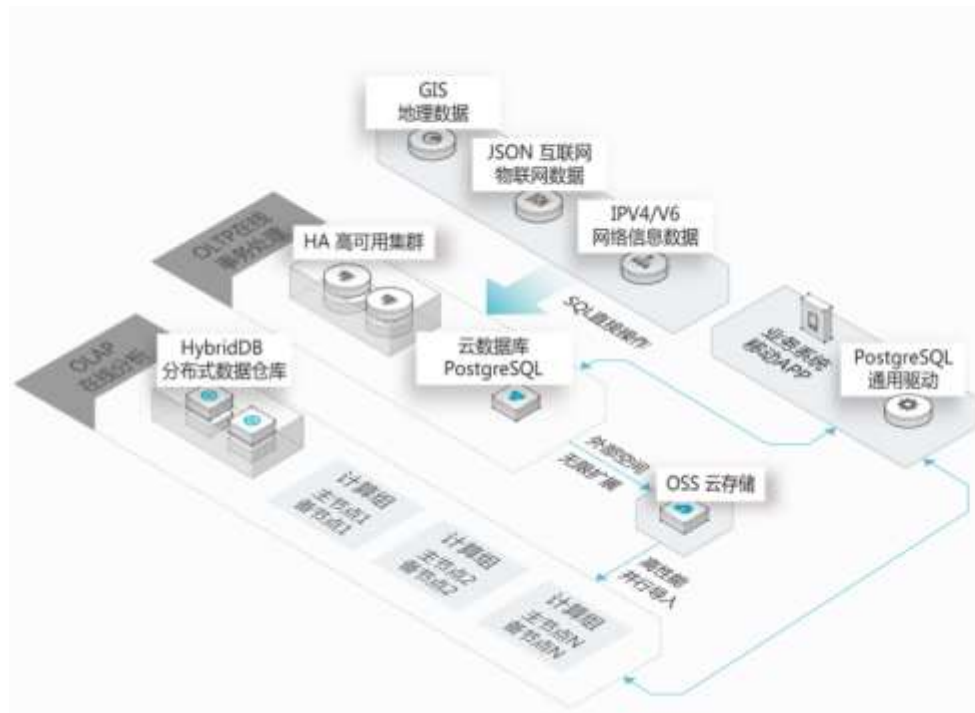
精准定位，支持2D、3D、路径、范围分析简单易用

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外部存储将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



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。

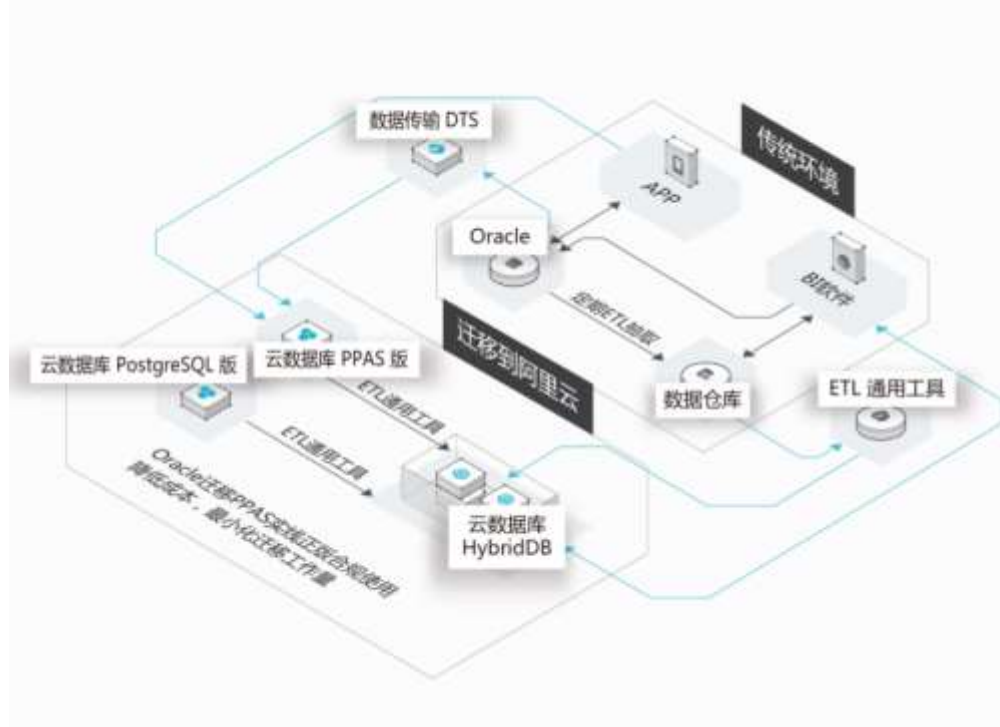


数据仓库

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上云迁移

让上云迁移变得更有价值

Oracle上云, 迁移只是第一步。阿里云协助用户实现正版化, 解决上市的合规审查风险, 提供企业级HA同城容灾, 整合HybridDB, 可实现按需扩展的OLAP决策分析。

方案优势

易于迁移

原有应用程序调整, 相比迁移到其他数据库工作量降低30%

更具价值

实现同城容灾, 提高HA安全级别, 保护升级价格不变。

OLAP扩展

结合HybridDB实现100倍于传统方案的OLAP能力实现实时报表

推荐搭配使用



HybridDB



OSS



数据传输



ECS

产品体系 - RDS for PPAS

云数据库 PPAS 版

OLTP+OLAP组合场景

企业全系统迁移方案

得益于PPAS的Oracle兼容特性，最少化现有程序的修改量，节省迁移人力。通过“数据传输(DTS)”工具实现增量数据导入，有助于降低业务系统割接过程的停机时间，同时目标系统可以在不影响生产业务的前提下实现并行测试。用于BI及报表分析的数据导入到HybridDB，通过MPP分布式数据仓库系统实现快速分析，为企业决策提供信息参考。

能够解决

数据迁移

DTS持续增量数据迁移，方便割接及异构功能验证

无限扩展

通过OSS无限扩展存储空间，突破RDS存储上限

OLAP分析

通过HybridDB实现20-100倍SQL分析性能提升

推荐搭配使用



HybridDB



OSS



DTS

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

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。

ADAM for PPAS 专版



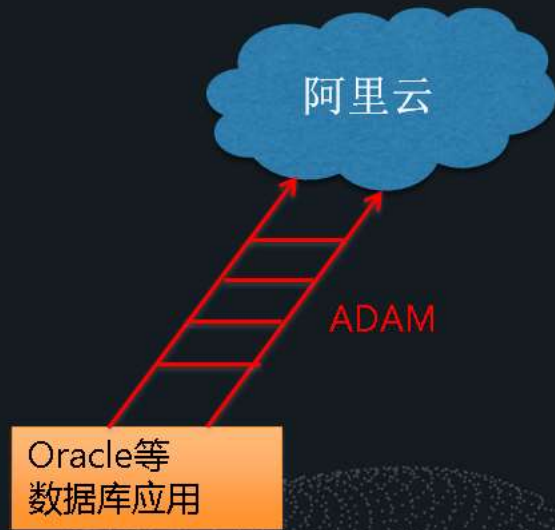
<https://www.aliyun.com/product/adam>

去O生态 - ADAM PPAS专版

Advanced Database & Application Migration

(简称 ADAM)

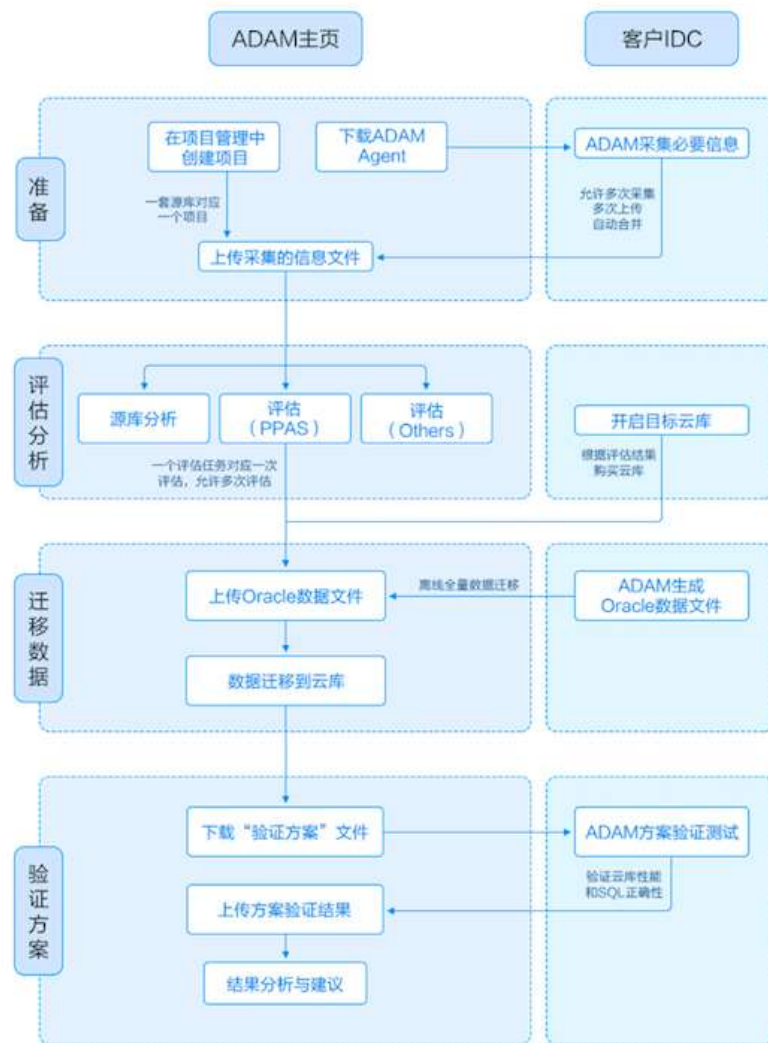
定位：简单地把云下Oracle等数据库应用迁移上云



去O生态 - ADAM PPAS专版

ADAM覆盖迁移上云全生命周期





某GaTrKgl去O上云



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
	48	0	0	1	0	4	0
	404	34	19	68	12	80	1
	43	1	0	15	1	28	0
	AT	111	1	0	0	0	0
	EW	53	0	23	1	32	2
	9	1	0	4	0	7	0
	9	0	0	0	0	6	0
	51	0	0	15	1	32	3
	14	0	0	4	0	6	0
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	16Cores64G Memory 512G Disk	962	

部分报告截图

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	2,467,889,152
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	867,172,362
1	Z	EW	3,641,592	1,363,869,696
1	X		3,293,384	2,193,883,136
1	X		3,000,000	1,000,000,000

部分报告截图

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

风险点列表

慢SQL(执行时间超10秒)

SQL ID	dnr4p3cjppzsz
执行时间(秒)	127

部分报告截图

PROCEDURE详细评估报告

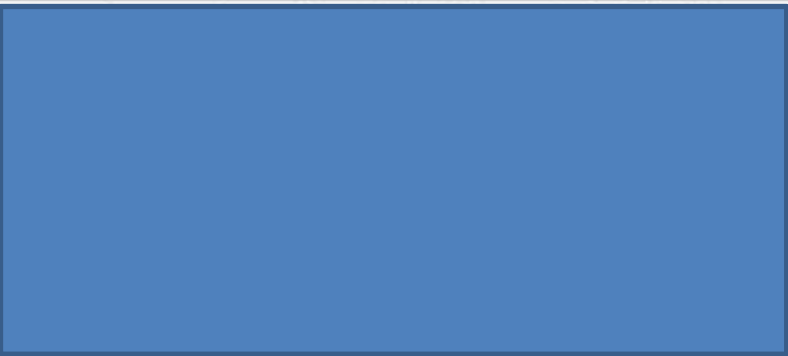
(企业版)

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	

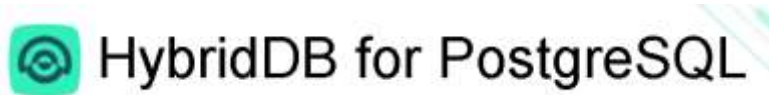
产品体系 - HybridDB for PostgreSQL



HybridDB for PostgreSQL



产品体系 - HybridDB for PostgreSQL



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**，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）

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

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

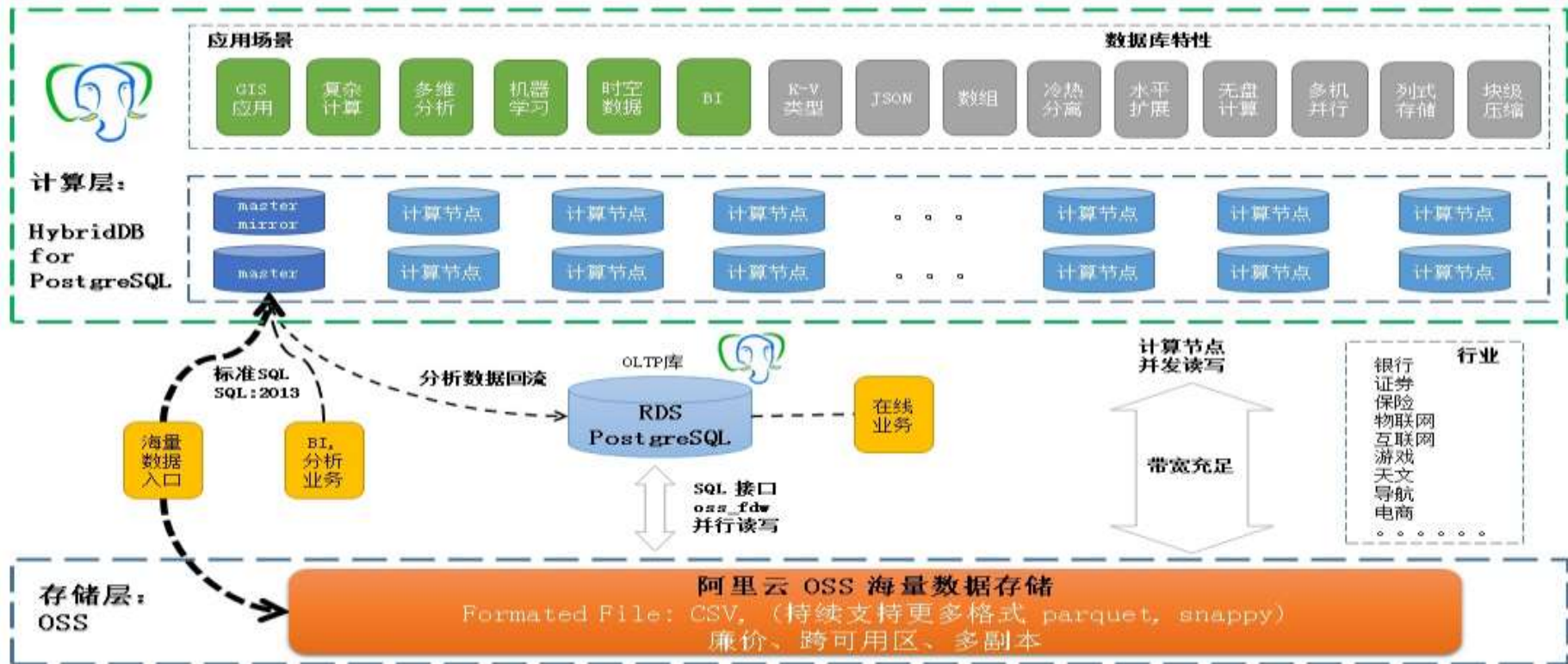
目录

- 产品介绍
- 生态介绍
- 应用案例
- 开发实践
- 参考文档

产品生态



OLTP+海量OLAP+分级存储->技术栈



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。

```
<dataSource type="org.mybatis.c3p0.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>
```

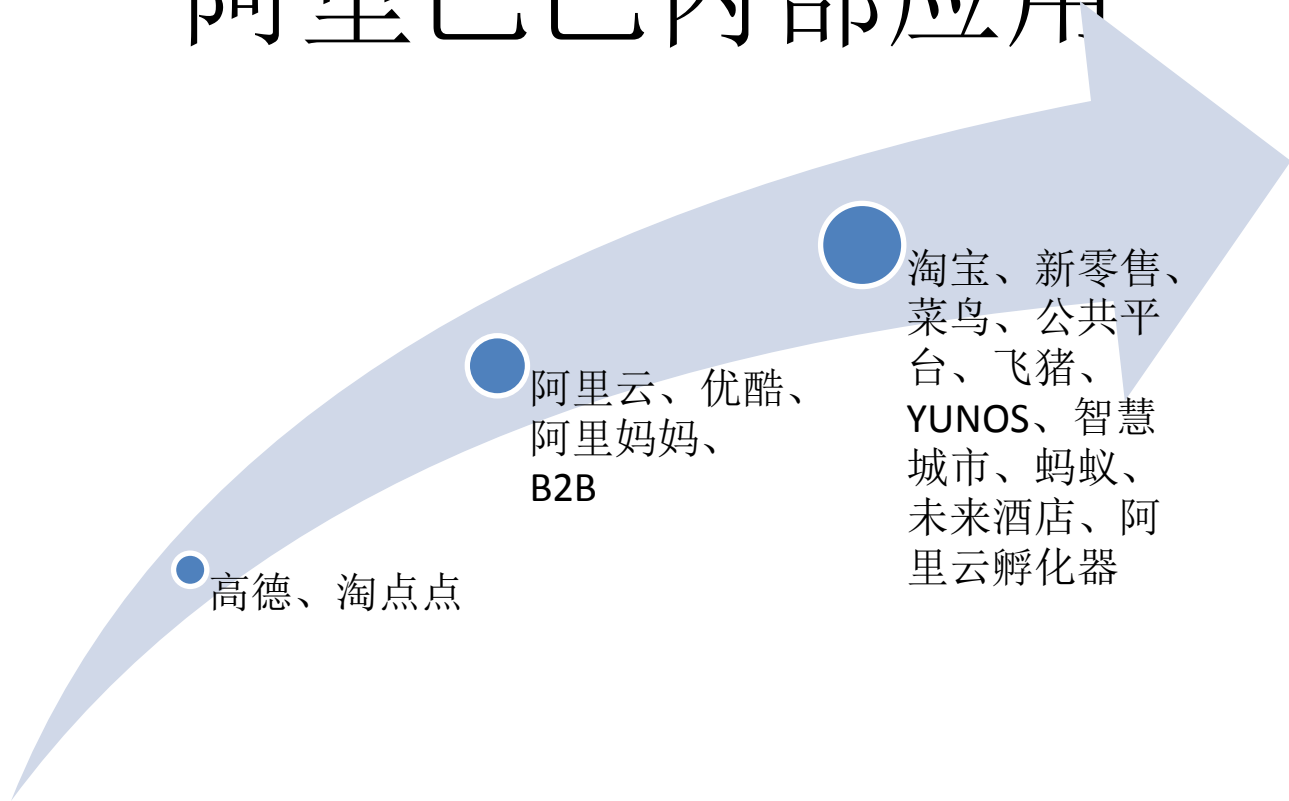
mysql, pg 类型映射

- 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

- 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



内部应用

- **RDS PostgreSQL**

内部应用

- **HybridDB PostgreSQL**

目录

- 产品介绍
- 生态介绍
- 应用案例
- 开发实践
- 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, 时间字段索引)
 - 每轮测试插入12TB数据
 - 506万行/s, 1.78 GB/s, 全天插入4372亿, 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tps (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 社交关系应用)（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)

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
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
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	1亿, 128个随机字符	112	14万	0.8毫秒
字符串查询、后缀	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	1亿, 128个随机字符	56	8.1万	0.68毫秒

OLTP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
字符串 查询 、相似查询	1亿，128个随机字符	56	1531	35毫秒
字符串 查询 、全文检索	1亿	56	5.14万	1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
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
合并 写入 (有则更新、无则写入)	1亿	56	22.8万	0.245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
IN\EXISTS 查询	1亿，IN(1,10,100,...100万个元素)		1毫秒~	380毫秒
NOT IN\NOT EXISTS 查询	1亿，NOT IN(1,10,100,...100万个元素)		27秒~	35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
键值更新	1亿	56	22万	0.25毫秒
空间数据、位置更新(滴、菜鸟、饿)	1亿	56	18万	0.3毫秒
秒杀 - 单条记录并发更新	1	56	23万	0.48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

OLTP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
单表, 有索引, 单事务多条写入	单行110字节, 每次提交1000条	56	16.7万行/s	334毫秒
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-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
排序	1亿			1.4秒
建索引	1亿			38秒

OLAP Benchmark(56 Core ECS, PG 10)

CASE	数据量	并发	TPS	平均响应时间
并行扫描	1亿	32		0.88秒
并行JOIN+聚合	1亿 JOIN 1亿 (无条件JOIN)	32		17秒
并行聚合	1亿	32		0.9秒
并行过滤	1亿	32		1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

测试详情:

<https://github.com/digoal/blog/blob/master/201711/readme.md>

RDS PG+HybridDB PG+OSS

海量实时处理

CDN-离线分析
CDN-域名违规整理

金融应用

平安
风控

模糊查询、相似搜索、正则搜索

网络广告-域名搜索
导购-实时盗文判定

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
智能车

任意字段实时搜索

网CRM

任意维度数据透视、钻取

设备、会员透视、精细化运营
销售
营销

时间、空间、对象多维搜索、透视

销售-选址、线上线下地推

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

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
- **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]**



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

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
 - 10亿级记录多字段搜索
- 新零售-营销、分销链路
 - 多值类型检索
 - 10亿级记录多值类型搜索



难以加速
难以同步一致性

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**

实例名

Select...

Q

高级搜索

实例名:

用户连接:

用户信息:

可用区:

请选择

集群名:

请选择

库类型:

请选择

库版本:

请选择

实例类型:

请选择

网络类型:

请选择

VIP类型:

请选择

TOP类型:

请选择

实例状态:

请选择

锁定模式:

请选择

服务状态:

请选择

是否非标:

请选择

SQL WALL:

请选择

业务类型:

请选择

实例角色:

请选择

网络MODE:

请选择

过期时间:

至

创建时间:

至

搜索

云产品方案、效果

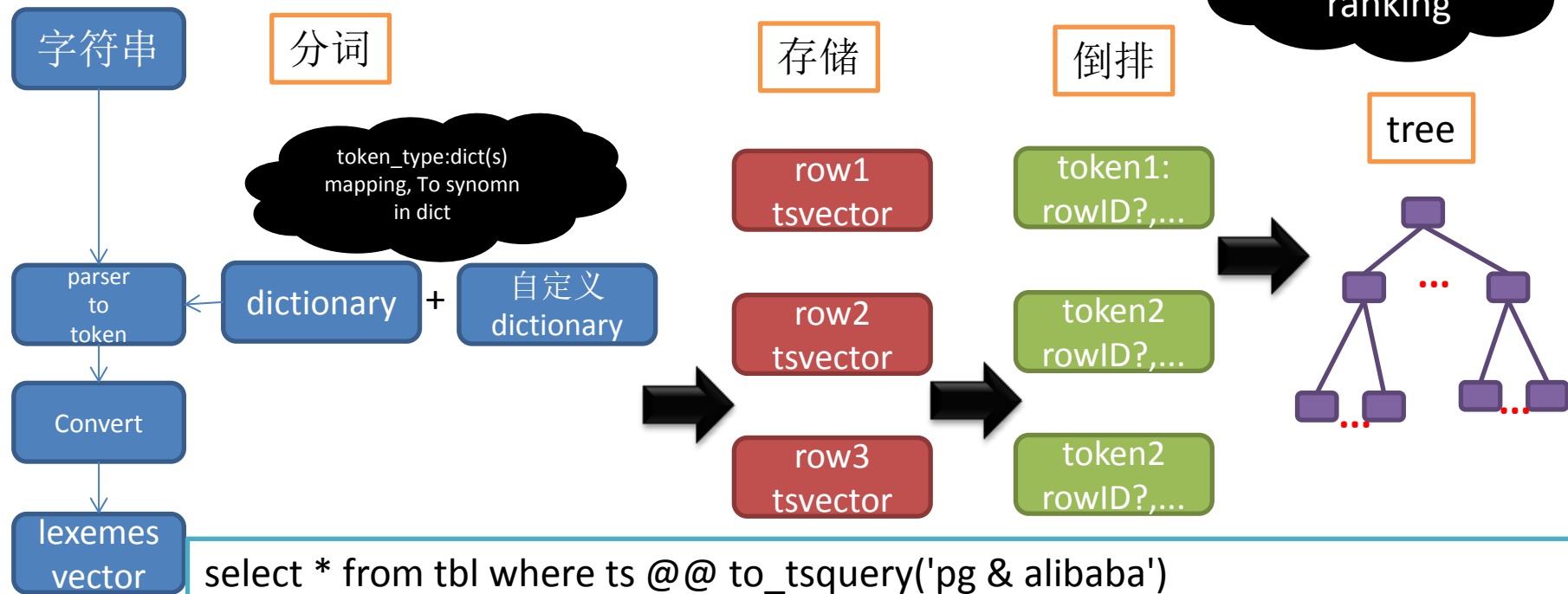
- RDS PG
 - gin 倒排索引，
 - 支持多值类型、多字段任意条件检索
 - bloom索引
 - 支持多字段任意组合等值条件过滤
 - 多索引 bitmap scan
 - 多个索引合并扫描
 - pg_trgm 支持
 - 模糊查询、相似查询、正则查询
 - zhparser中文分词插件支持中文分词



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

支持精排、
微调
ranking



```
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 - 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

内置ranking
算法

```
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

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

内置ranking
算法

```
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

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 [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 on 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;
```


任意字段组合条件搜索

- 多个独立的索引的BITMAP SCAN（或单个GIN多字段复合索引）
 - 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。

```
+-----+
|10000000000100000001000000000000111100000000| bitmap 1
|000001000001000100010000000001000010000000010| bitmap 2
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
|000000000000100000001000000000000001000000000| Combined bitmap
+-----+
          |          |          |
          v          v          v
Used to scan the heap only for matching pages:
+-----+
|_____X_____X_____X_____|
+-----+
```

详细链接

- 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>
- 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>
- 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>

Case3(特征、相似)

- 相似
 - 数组相似
 - 文本特征值相似
 - 图片相似

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



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



难以加速

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

- imgsmllr插件

- 相似图片

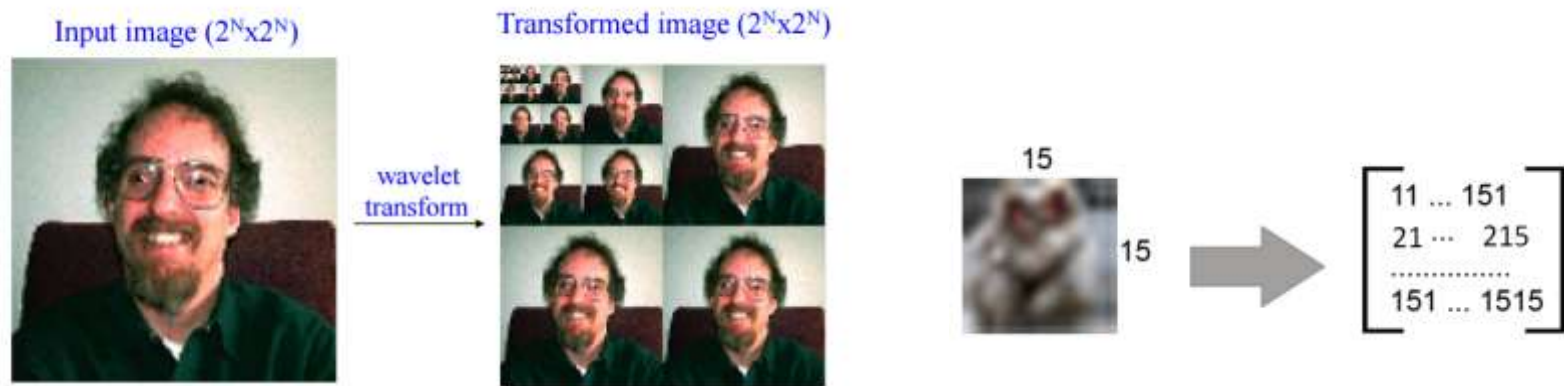
Heap ID	key Page	1	35	44	...	101	109	1000
1		1	0	0	:	1	0	0
3		0	0	1	:	0	1	1
101		0	0	1	:	1	0	0
198		0	1	0	:	1	1	0
798		1	0	0	:	1	0	1
1000		0	0	1	:	0	0	0
		2	1	3	...	4	2	2

海明码切分

id	hmv	hmarr
1	0101011111110100000010010111011011000111110111101101100000011	{1_010101111111010,2_000001001011011,3_011000111110111,4_111011111011011}
2	11010110000100000000000000011010111101101110100000010101011	{1_1101011000010000,2_000000000000110,3_11011110110111,4_011011101000000}
3	0101000010110110110010001010100010101001010111111011000110011	{1_0101000010110110,2_1100100010101000,3_1010100100101011,4_001010111110110}
4	0111000111100011110001001110000110111111000001110101101000100	{1_011100011110001,2_1110001001110000,3_1110111111000001,4_110000011101011}
5	0010111010101011111010011101100100111011111110011101110010011	{1_0010111010101011,2_111010011101100,3_100111011111111,4_111111100111011}
6	0110111100111001001100101110100000000110001100001110001010110	{1_011011110011100,2_1001100101110100,3_000000110001100,4_100011000011100}
7	010011010011100111001101111010011101110101001000101010110110	{1_0100110100111001,2_110011011101001,3_1110111010100100,4_1010010001010101}
8	011001011100110011100001101101110000110011111101111011010100010	{1_0110010111001100,2_1110000110110111,3_000011001111110,4_111111011110110}
9	011011101100000010101111000101110000010011100011100101000100	{1_011011101100000,2_0010101011110001,3_0111000001001110,4_0100111000111001}
10	010110100000011010010110001111111000101110001010011100110101011	{1_0101101000000110,2_1001011000111111,3_1100010111000101,4_1100010100111001}

(10 rows)

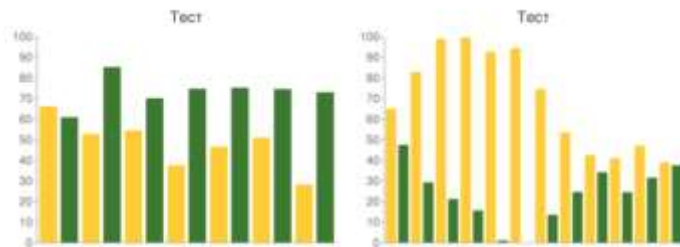
图像特征值提取与存储



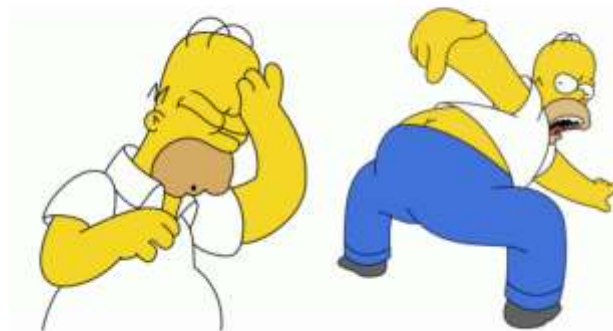
图像特征值比对



87,55% similarity



61,33% similarity



23,56% similarity

RDS PostgreSQL

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

详细链接

- 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
 - https://github.com/digoal/blog/blob/master/201607/20160726_01.md

Case4(画像、特徴)

- 画像システム



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， 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。



效率低

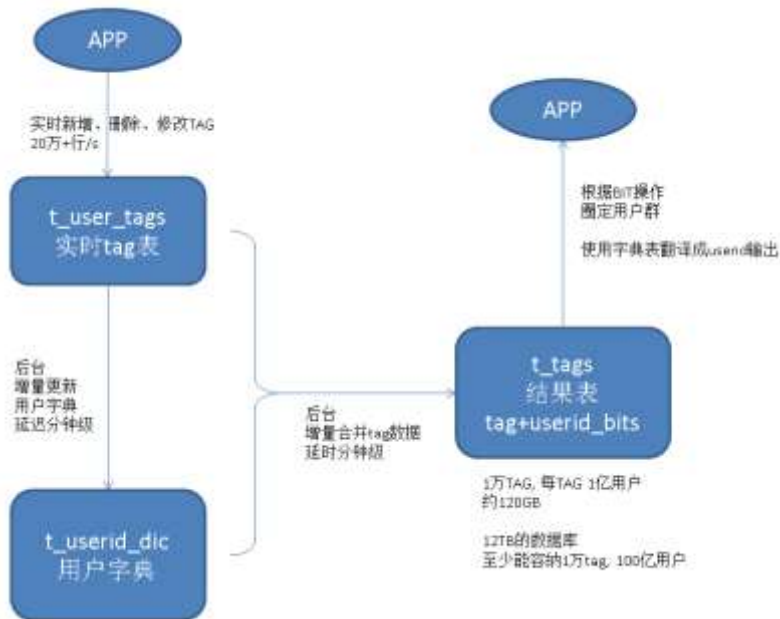
云产品方案、效果

- RDS PG

- 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



详细链接

- varbitx

- 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

Case5(模拟股票交易系统)

- SchemaLess

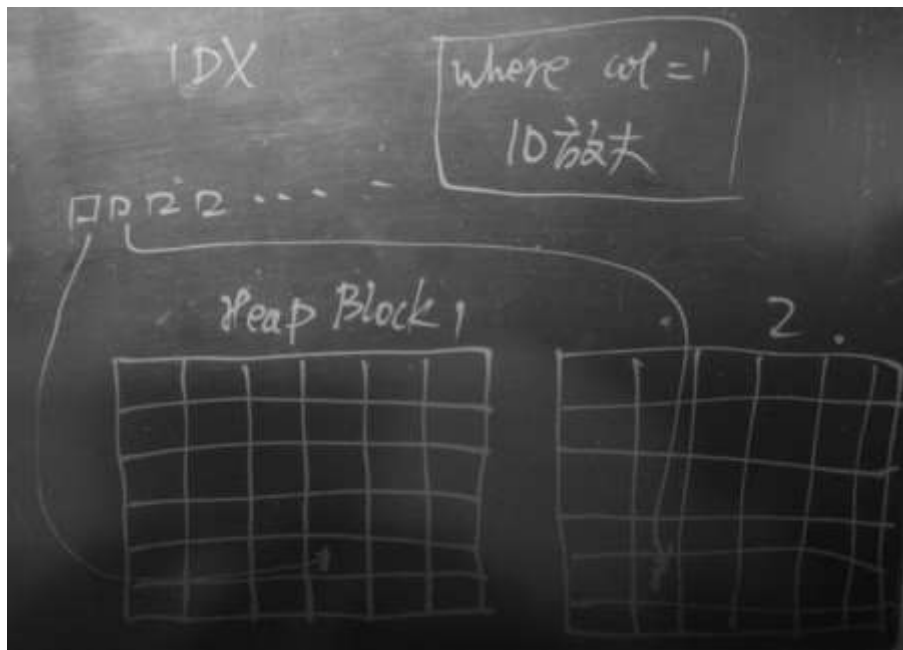
▼	代码	名称	• 涨幅%	现价	涨跌	买价	卖价	总量
1	600030	中信证券	-0.50	18.20	-1.69	18.19	18.20	364.9万
2	601198	东兴证券	-0.12	24.79	-2.19	24.79	24.81	746274
3	002736	国信证券	-7.72	19.95	-1.67	19.91	19.95	621341
4	600958	东方证券	-7.30	23.33	-1.06	23.33	23.34	457017
5	601555	东吴证券	-7.35	15.63	-1.24	15.63	15.65	711700
6	000728	国元证券	-7.23	23.73	-1.05	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

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

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。

详细链接

- 自动切片

- https://github.com/digoal/blog/blob/master/201704/20170417_01.md
- 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

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
 - 放牧监控
 - 菜鸟-包裹快递员分配管理



画出电子围栏，并设置
围栏名称和报警设置



电子围栏信息查看，以及
为电子围栏关联组或者终端

区域报警设置:

围栏名称:	中山公园
报警设置:	出区域报警
<input type="button" value="保存"/> <input type="button" value="取消"/>	

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**



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



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

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



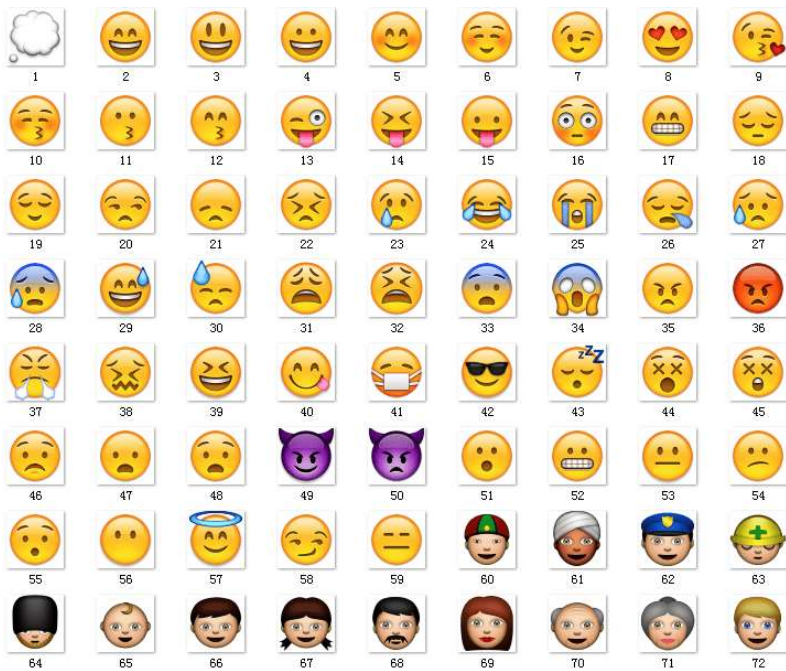
空间数据
管理

详细链接

- 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
- 商旅问题
 - https://github.com/digoal/blog/blob/master/201704/20170409_01.md

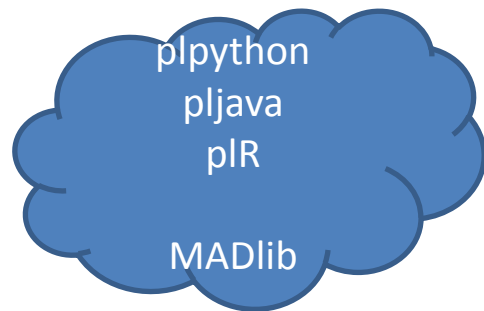
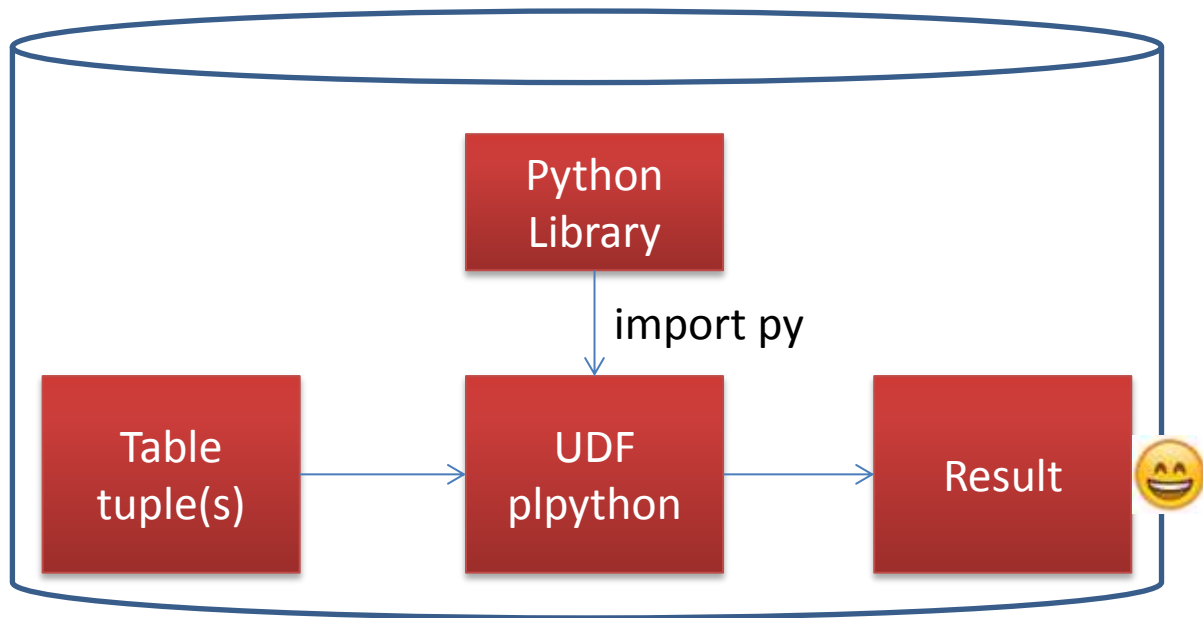
Case7(文本情感)

• 实时文本情感分析



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

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（比如税务）
 - 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

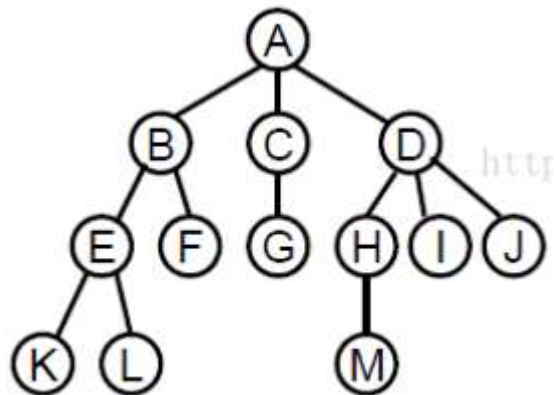


痛点

- 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

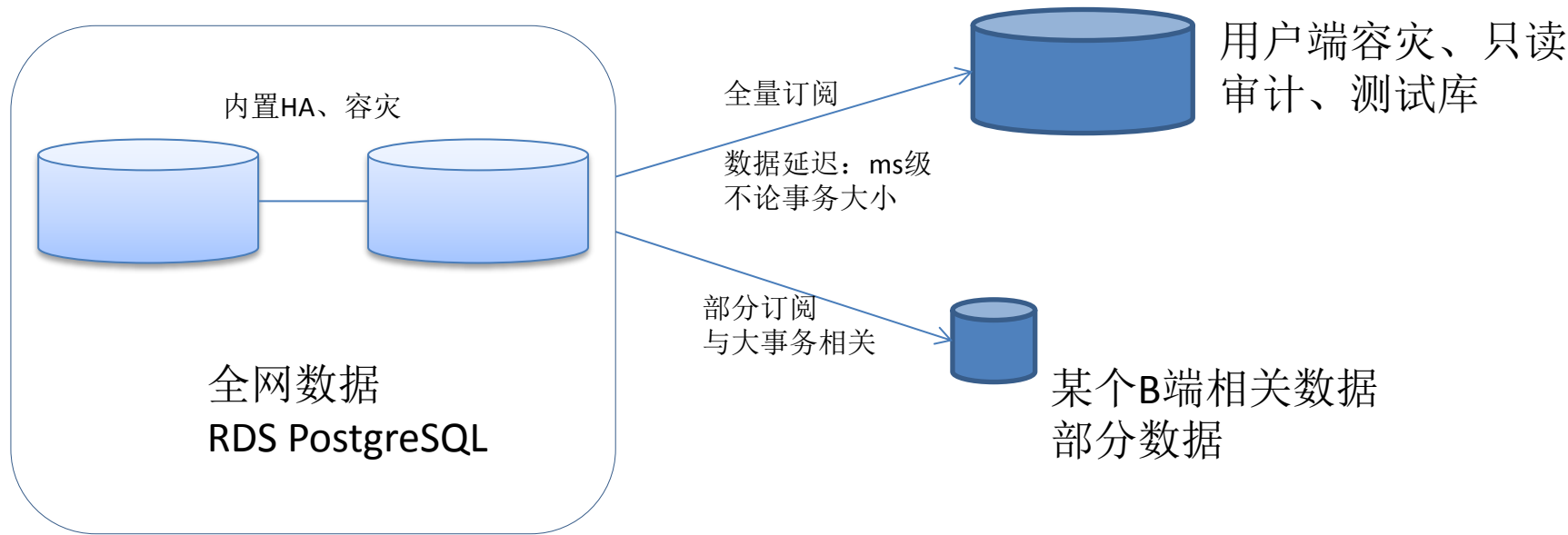
RDS PG

- 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)



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




详细链接

- 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
- 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
- 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
- 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

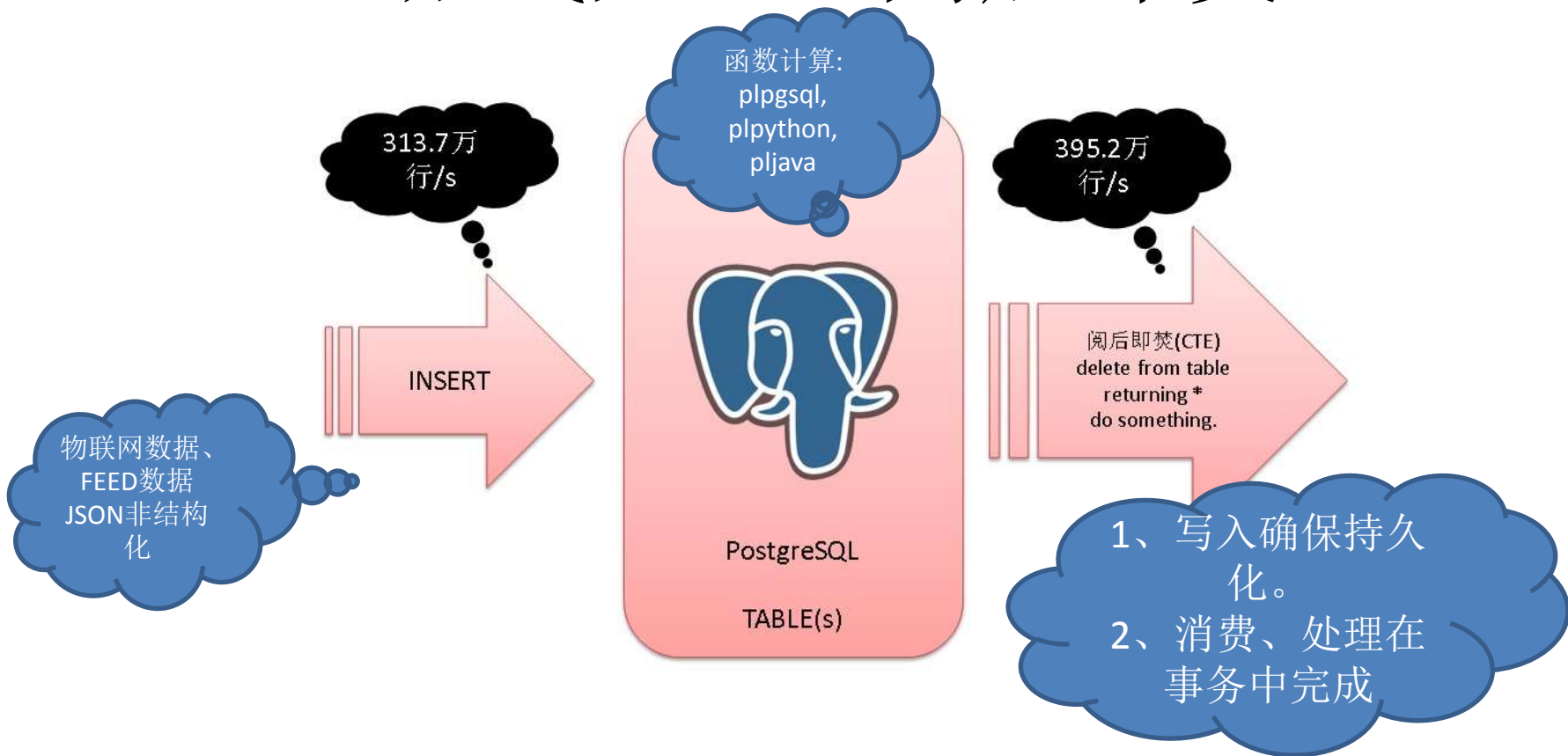
case10 (流式处理 - 阅后即焚)

- 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



高效、可靠、流
式、事务

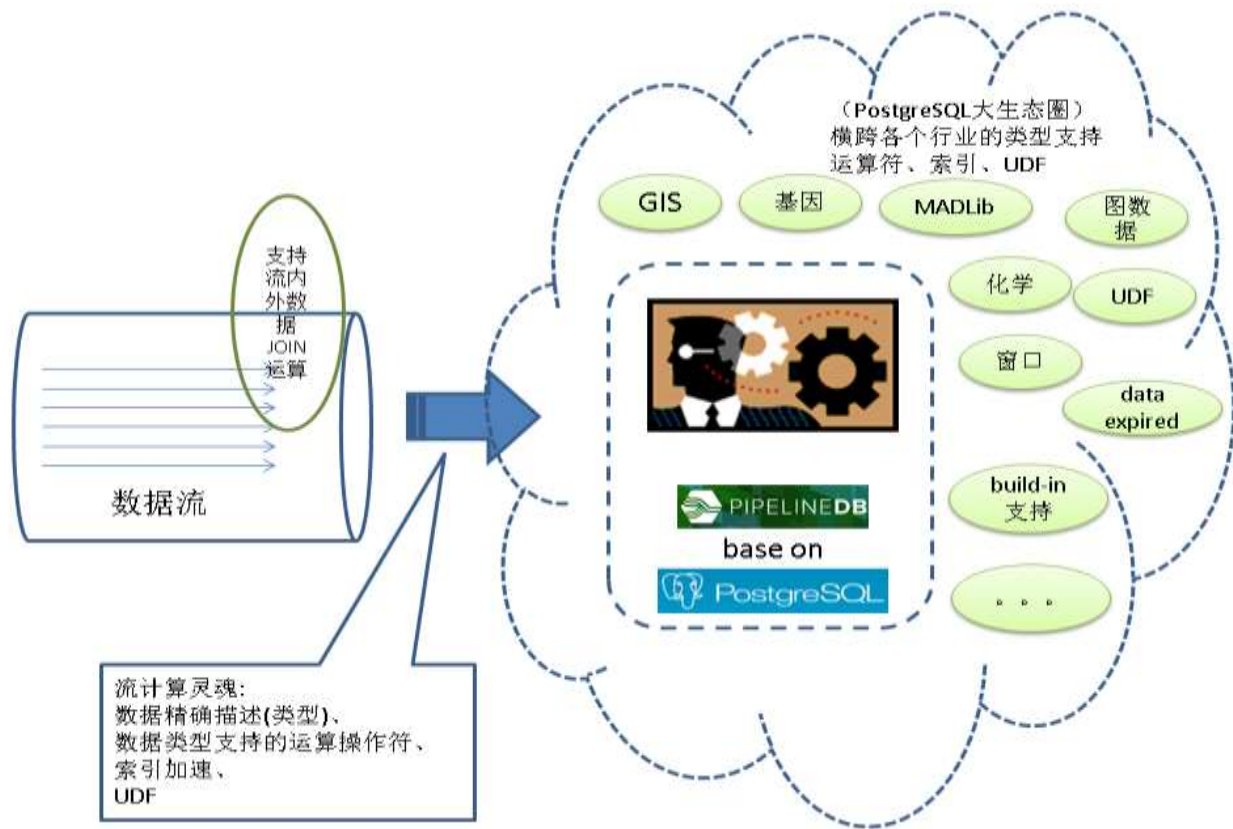
流式处理 - 阅后即焚



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)

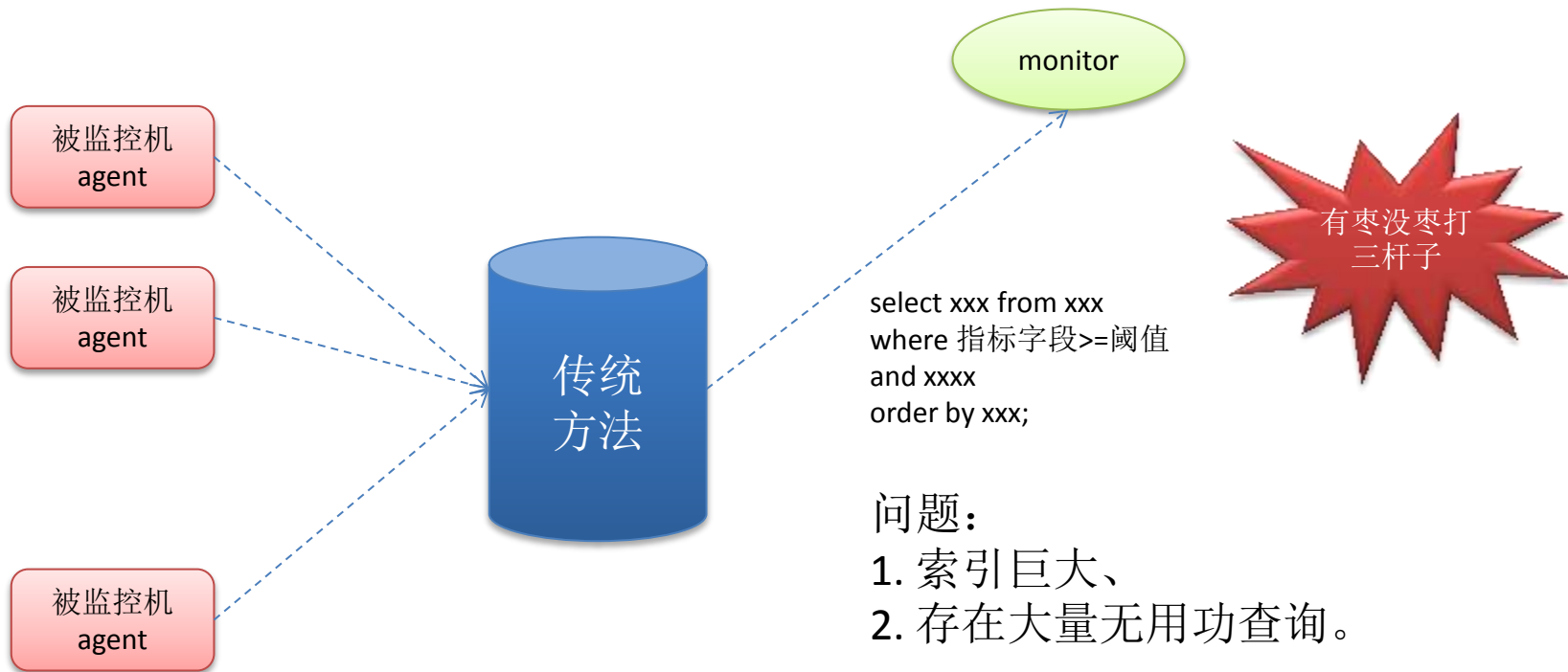
方法3、insert on conflict (实时处理，实时，小吞吐，单步写30+万行/s，批量写100+万行/s)

方法4、阅后即焚 (批处理，低延迟，大吞吐，100+万行/s)

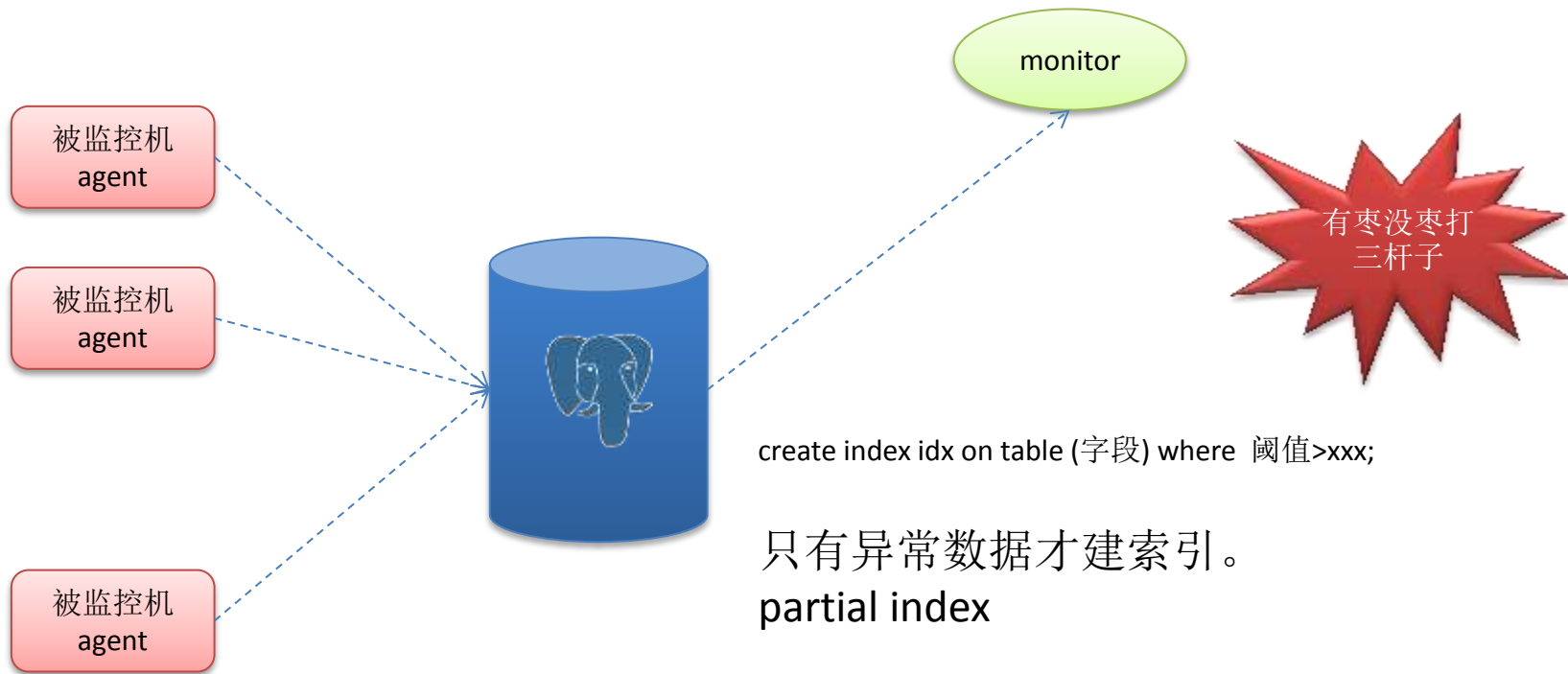
案例1-流式预警

- 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



案例1-流式预警



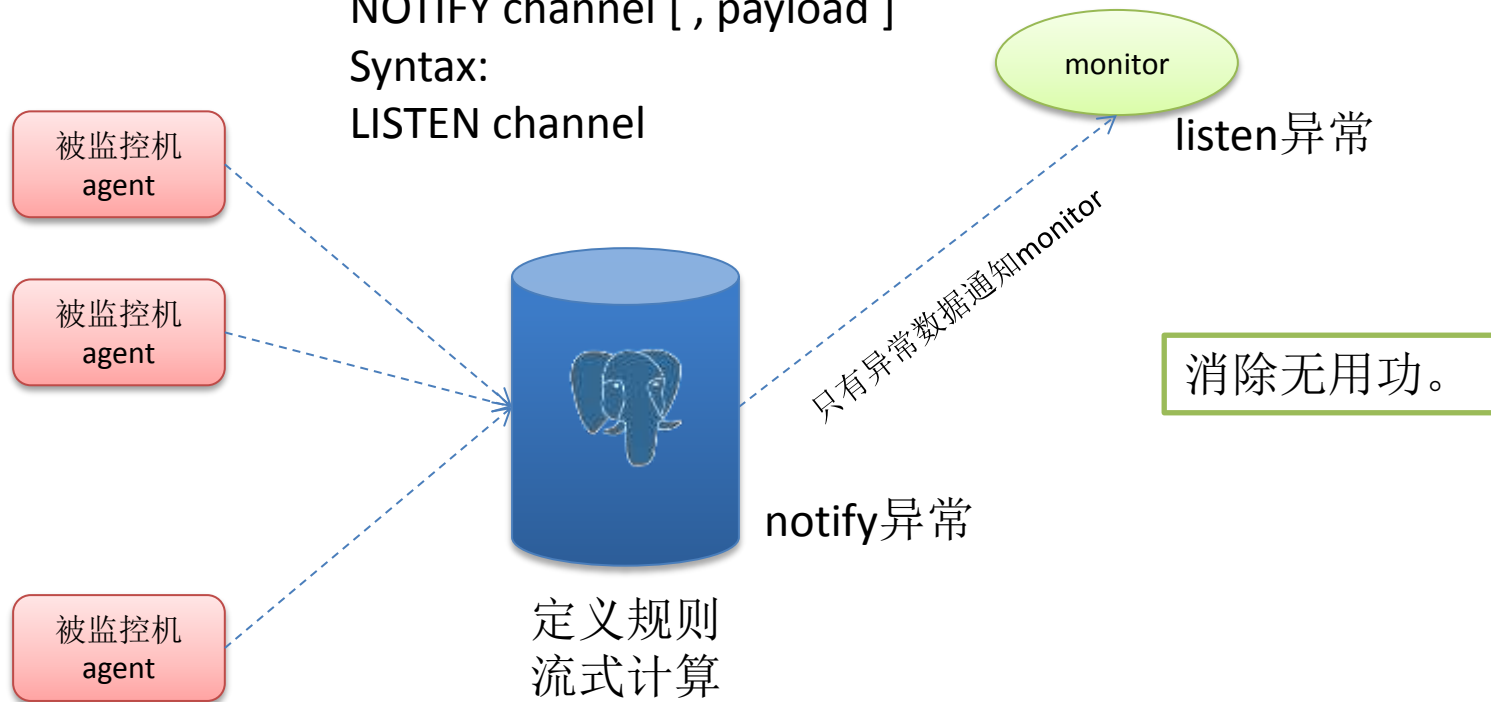
案例1-流式预警

Syntax:

NOTIFY channel [, payload]

Syntax:

LISTEN channel



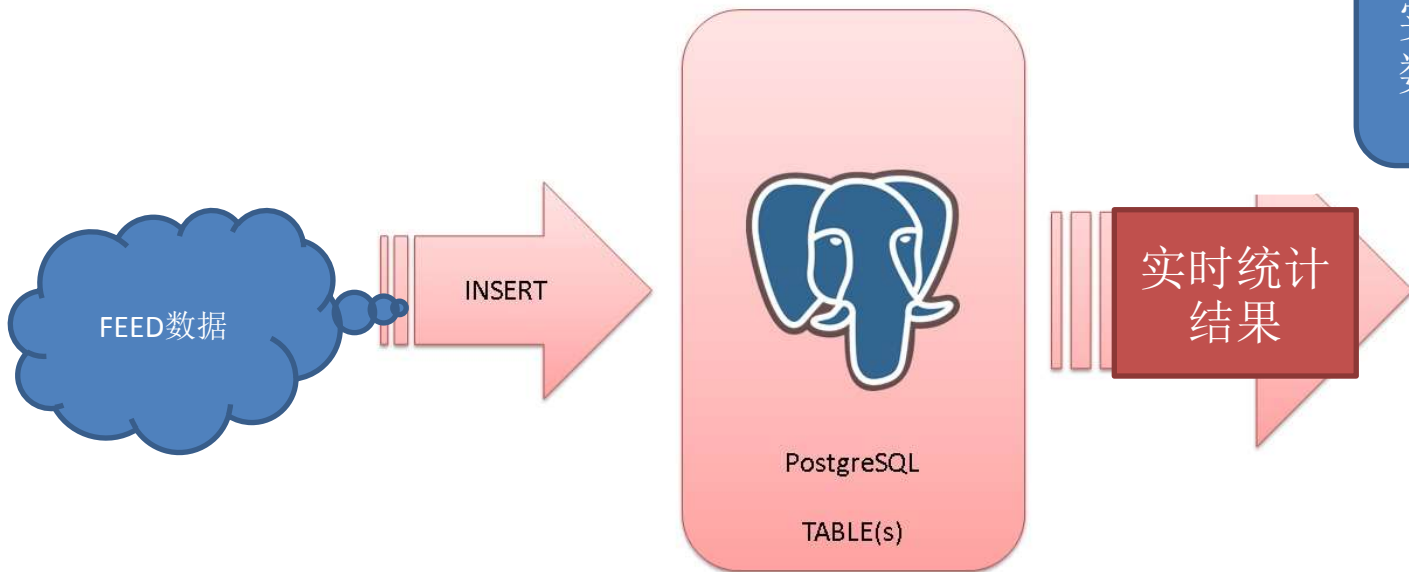
案例1-流式预警

- `create table tbl(sid int, content jsonb);`
- `create or replace function monitor(jsonb) returns boolean as $$`
- `declare`
- `begin`
 - `if xxx then return true; else return false; end if;`
 - `.....`
- `end;`
- `$$ language plpgsql strict;`
- `create or replace rule r1 as on insert to tbl where (monitor(NEW.content)) do also select pg_notify('channel_name', (NEW.content)::text);`
- `client:`
 - `listen 'channel_name';`

案例2 - 流式统计(avg,count,min,max,sum)

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

业务举例：
实时报表、实时在线
数、实时大屏监控、
实时UV估算



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 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 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

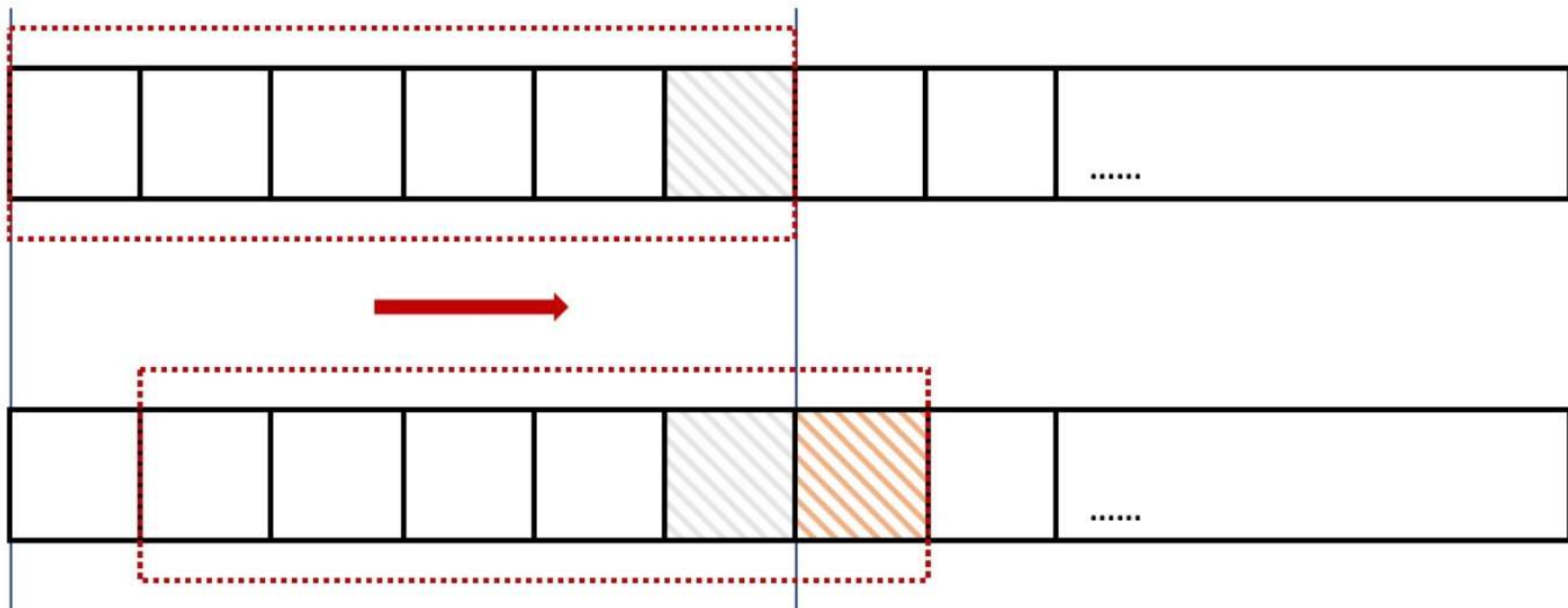
- https://github.com/digoal/blog/blob/master/201711/20171123_02.md
- **HLL插件**
- **create extension hll;**
- create table tbl (
 - grpid int, userid int, dt date,
 - cnt int8 default 1,
 - hll_userid hll default hll_empty(), -- 估算字段
 - primary key (grpid, dt)
 -);
- insert into tbl (grpid, userid, dt) 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);
```


每日新增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		<pre>hll_add(users, hll_hash_integer(123)) or users hll_hash_integer(123) or hll_hash_integer(123) users</pre>
hll_cardinality	#	<pre>hll_cardinality(users) or #users</pre>
hll_union		<pre>hll_union(male_users, female_users) or male_users female_users or female_users male_users</pre>

案例4-流式阅后即焚

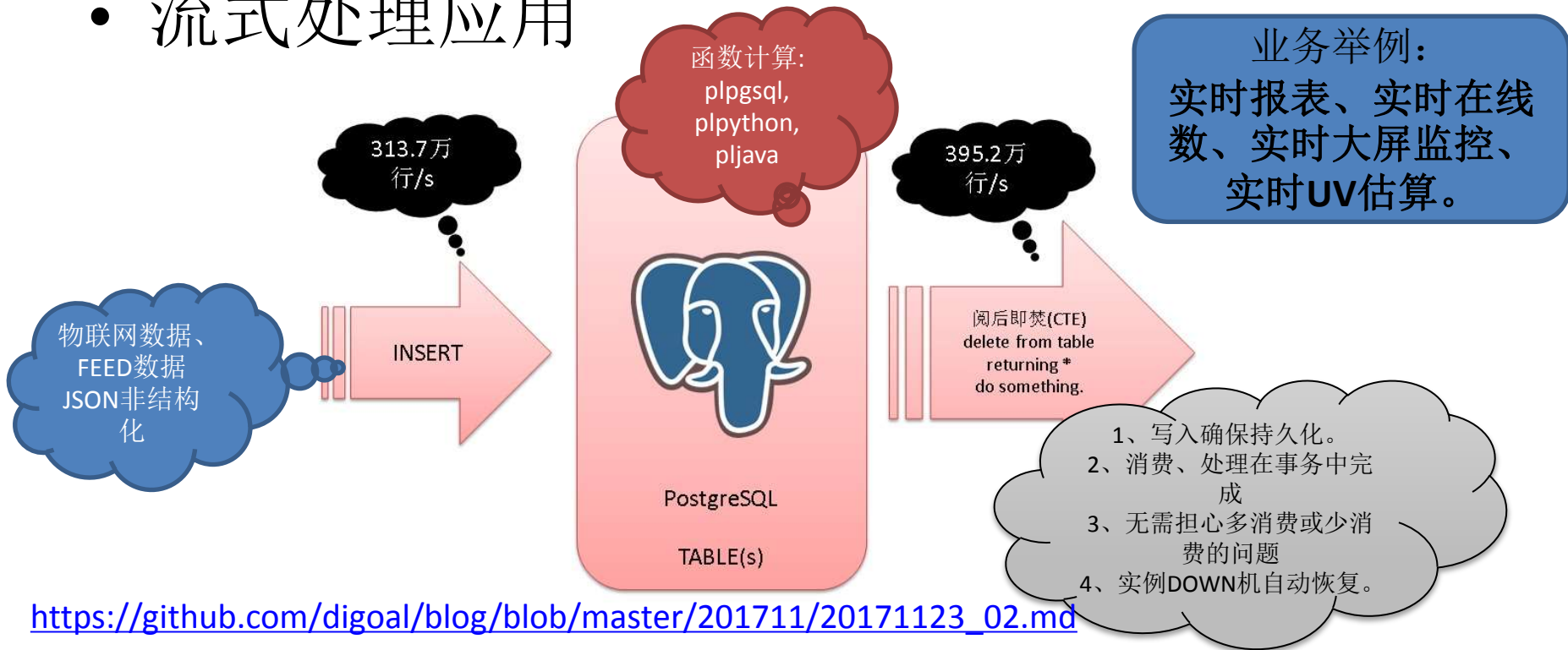
- 流式处理，高并发写入，快速消费处理。
- 处理后的数据被删除。
- 要求：
 - 数据快速写入
 - 数据写入后必须持久化
 - 快速消费被写入的记录（例如订阅，或者用于业务上的流式计算，计算结果保留）
 - 消费和计算必须在一个事务完成
 - https://github.com/digoal/blog/blob/master/201711/20171107_32.md



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



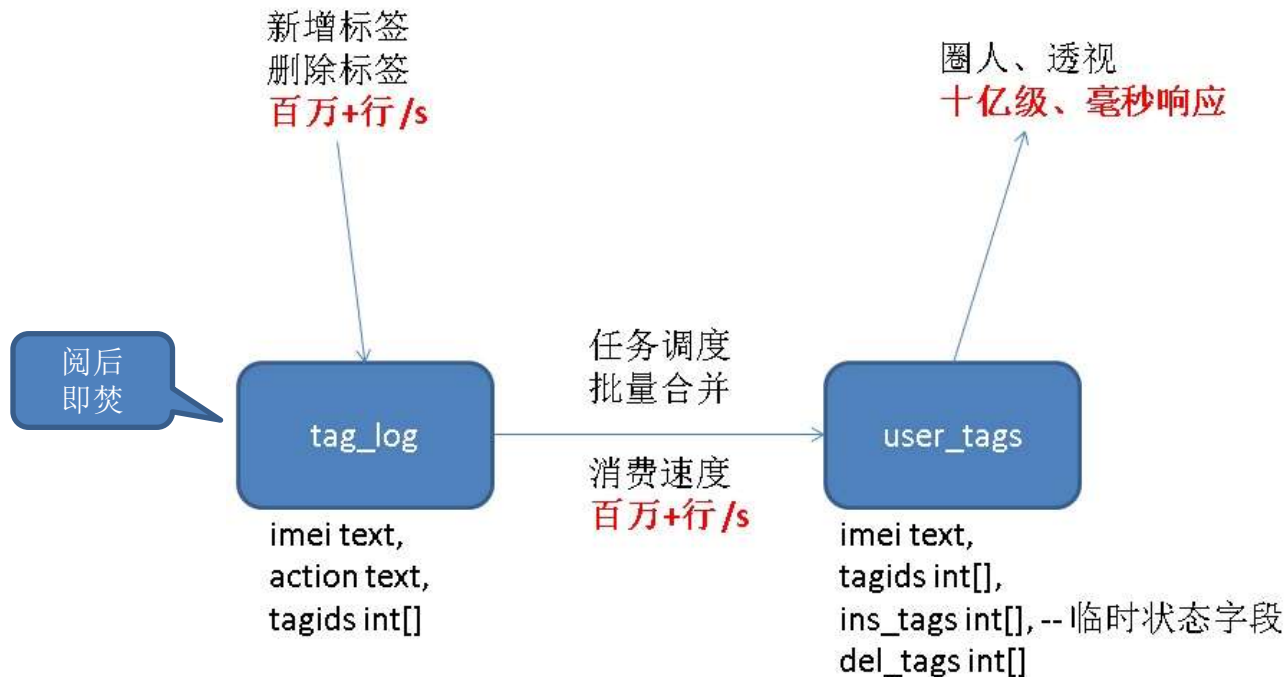
案例4-流式阅后即焚

- 流式处理应用
- create table tbl (id int, info jsonb);
- insert into tbl;
- 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-实时经营分析1

- 经营分析系统、决策系统

- https://github.com/digoal/blog/blob/master/201711/20171126_01.md



详细链接

- 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

Case11 (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

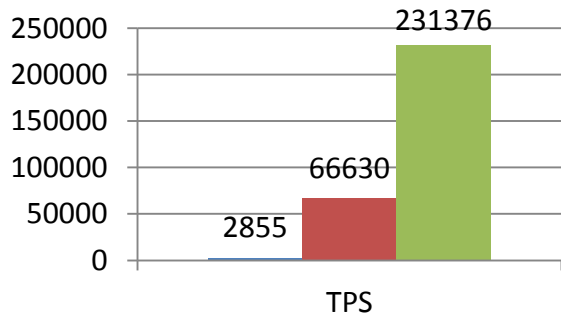
传统 - 行锁弊端

1. 无效等待多

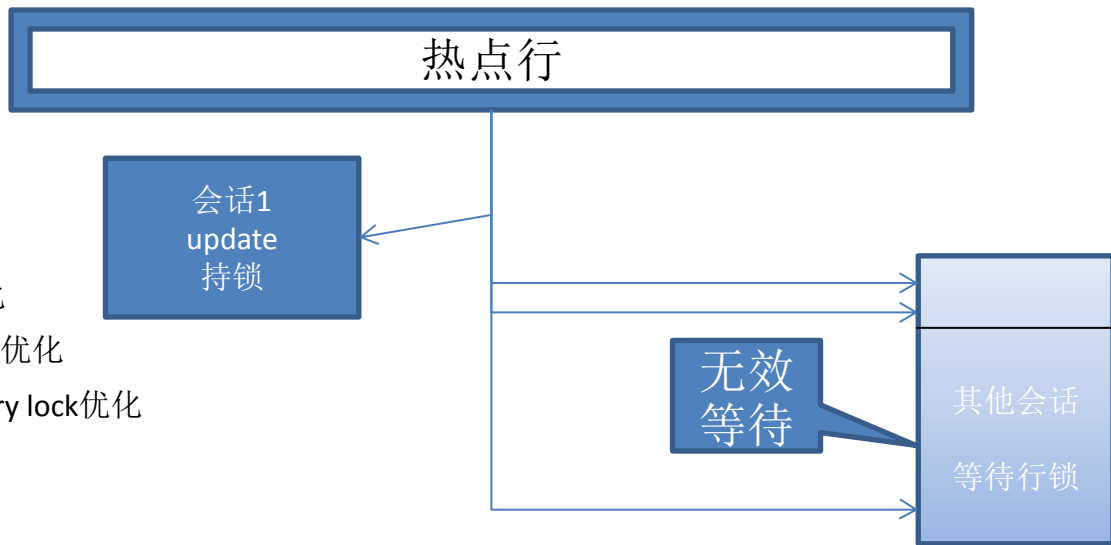
2. 无效等待用户

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

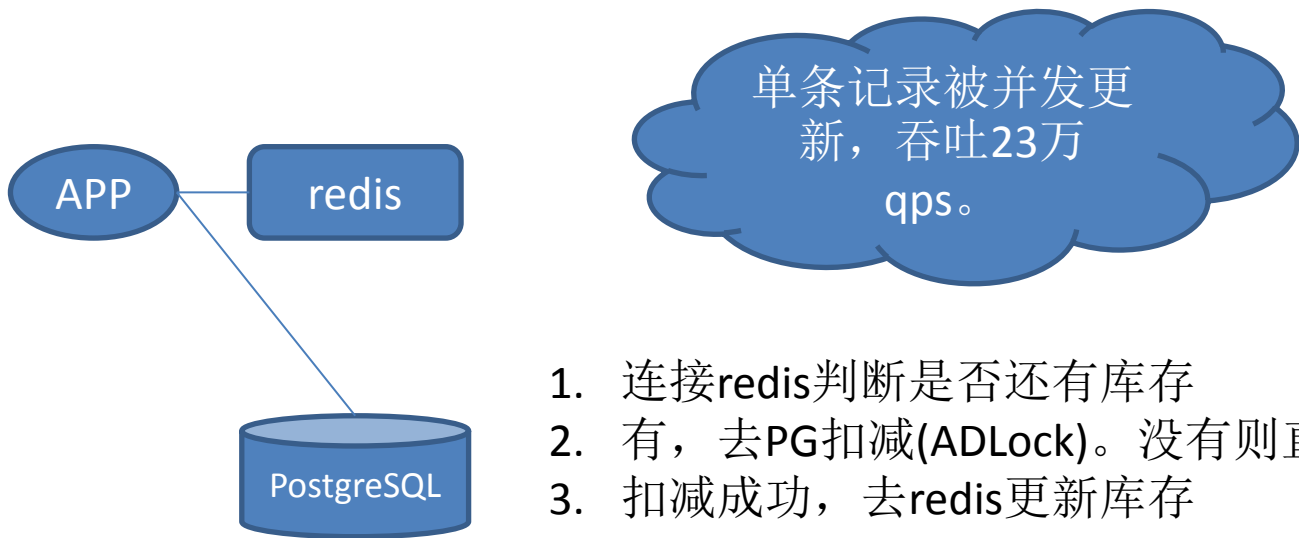


■ 未优化
■ nowait优化
■ advisory lock优化



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 *;`

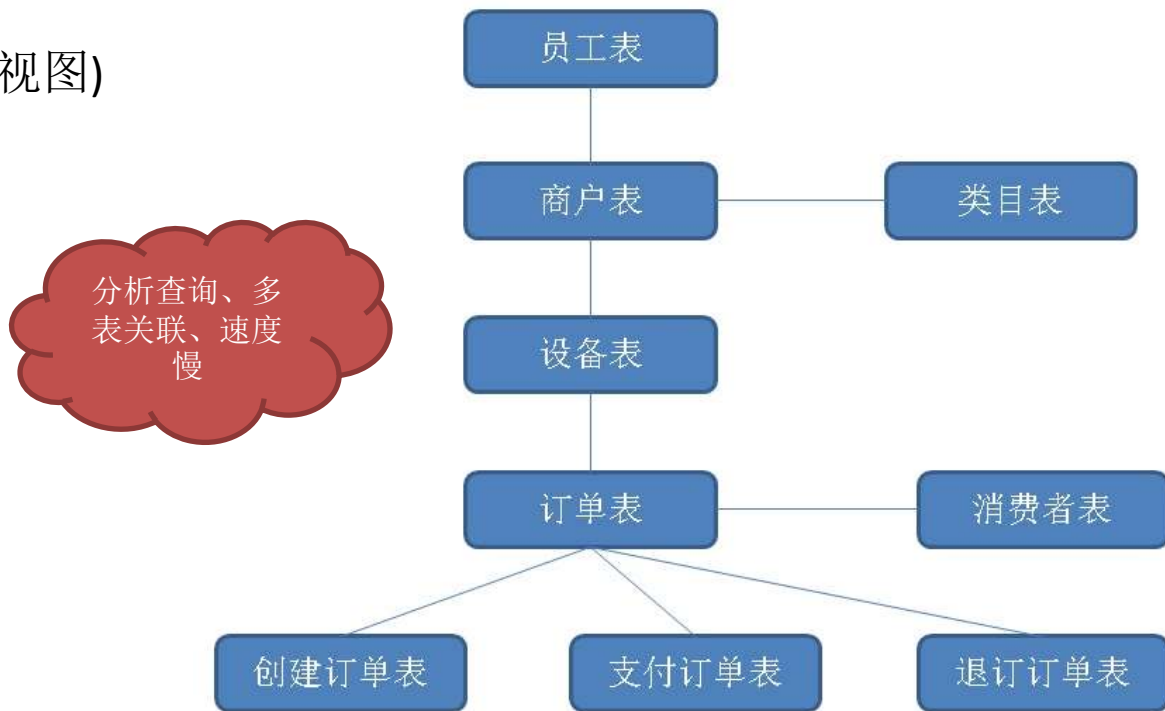


详细链接

- 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

Case12 共享充电宝(多表JOIN实时分析)

- Itree 树类型、消除 JOIN
 - (空间换时间、物化视图)



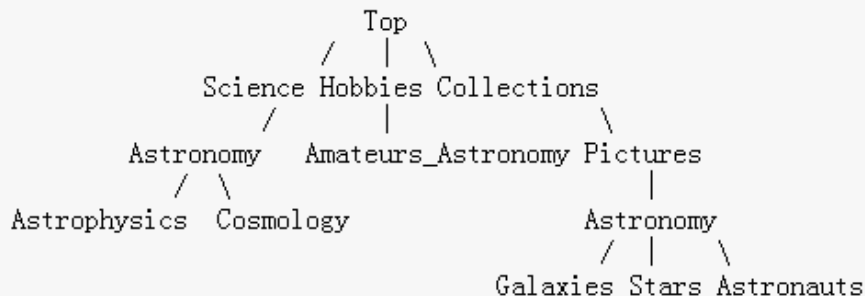
Itree 树类型 + R-T索引

- Itree

- <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



```
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 共享充电宝(多表JOIN实时分析)

- Itree 树类型、消除 JOIN
 - (空间换时间、物化视图)

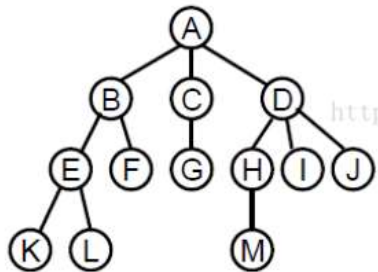
前后对比:

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



详细链接

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

RDS PG + HDB PG应用案例

Case 1 优士-智能广告

<div><div>*任务名称</div><div>终端</div><div>业务</div><div>*资源位</div></div>		<div>入口图片</div>	
<div><div>*选择素材</div></div>		<div>投放维度</div> <div>*内容分类</div>	

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

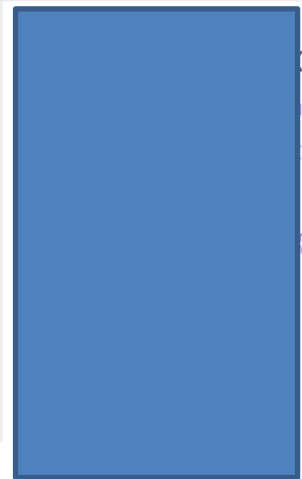
曝光目标设置

曝光目标人群

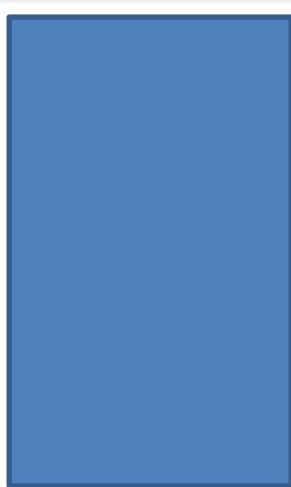
曝光频次控制

优士-智能广告

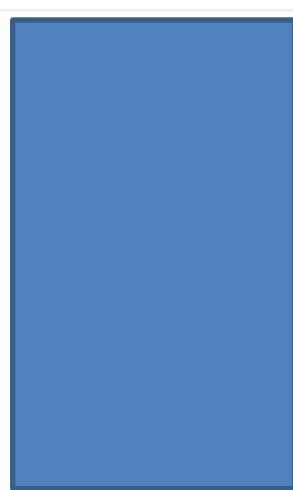
用户曝光次数小时表



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 × 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

曝光量

1000W

750W

500W

250W

0

前贴
视频首页
banner1首页
banner2中插
视频

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 × 100%

点击UV/点击 × 100%

456

七月预告

七月预热投放

2

七月安生预热

123

前贴视频

优士-智能广告

- **RDS PostgreSQL 角色**

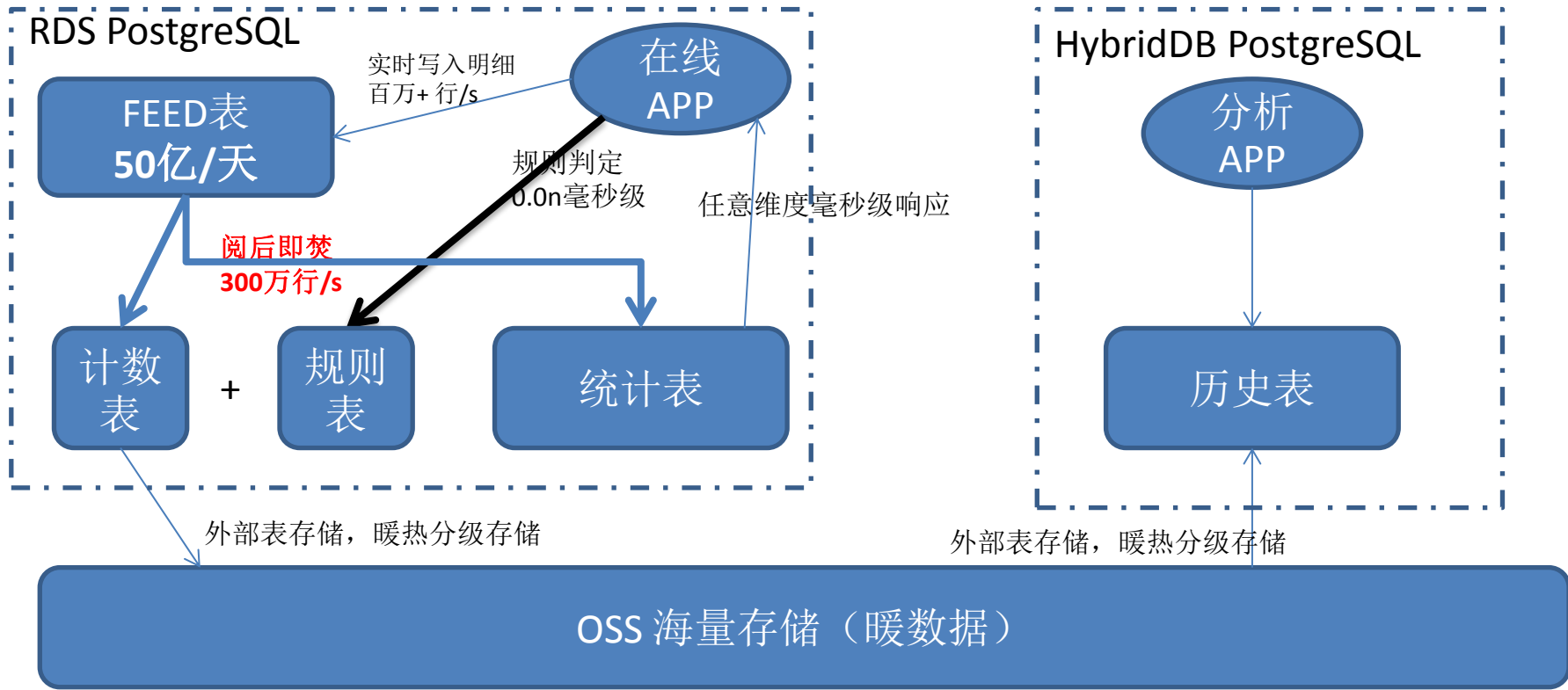
- FEED表
 - 命中规则，写入FEED。(100万+ 行/s)
- 计数表
 - 分区：小时、天、周
 - 实时写入、阅后即焚(300万行+/s)，合并到计数表
- 实时统计表
 - 阅后即焚(300万行+/s)、合并到实时统计表，提供毫秒级任意维度查询
- OSS存储
 - 计数表，上一小时、天、周，调度写入OSS。200MB/s 并行读写速度。

- **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



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
 - $y_1 = b_0 + b_1x_{11} + b_2x_{12} + \dots + b_px_{1p} + \epsilon_1$
 - $y_2 = b_0 + b_1x_{21} + b_2x_{22} + \dots + b_px_{2p} + \epsilon_2$
 -
 - 求截距，斜率。
 - 预测 y_n
 - $y_n = b_0 + b_1x_{n1} + b_2x_{n2} + \dots + b_px_{np} + \epsilon_n$

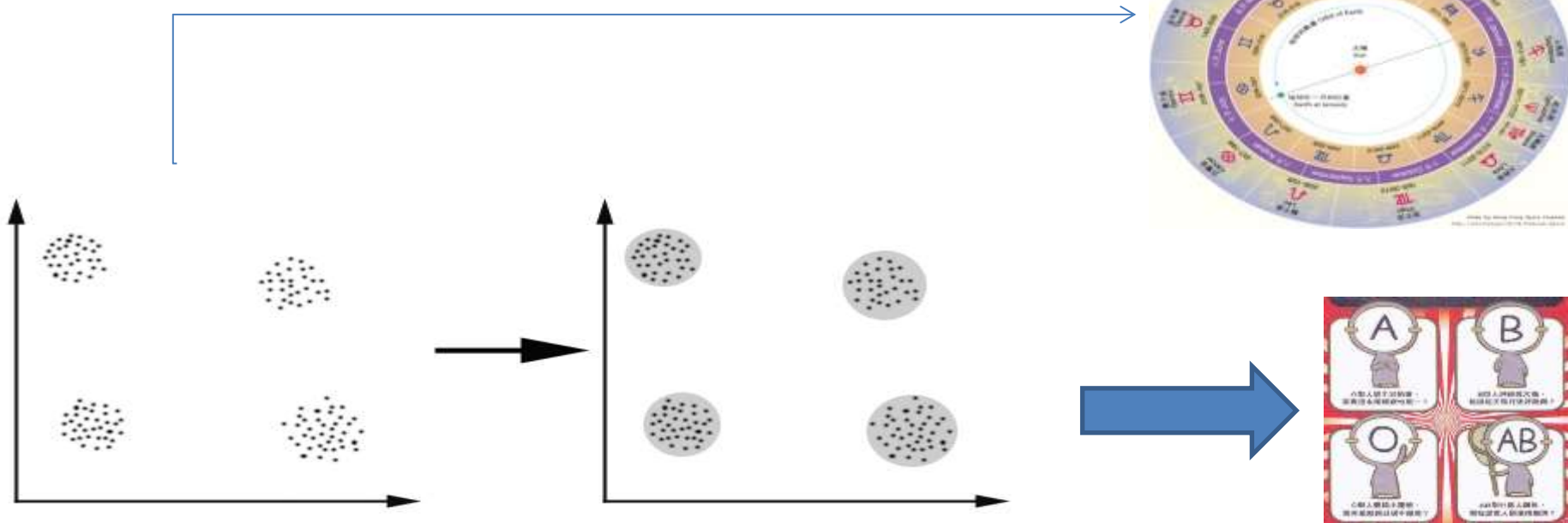
股市有风险
入市需谨慎

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



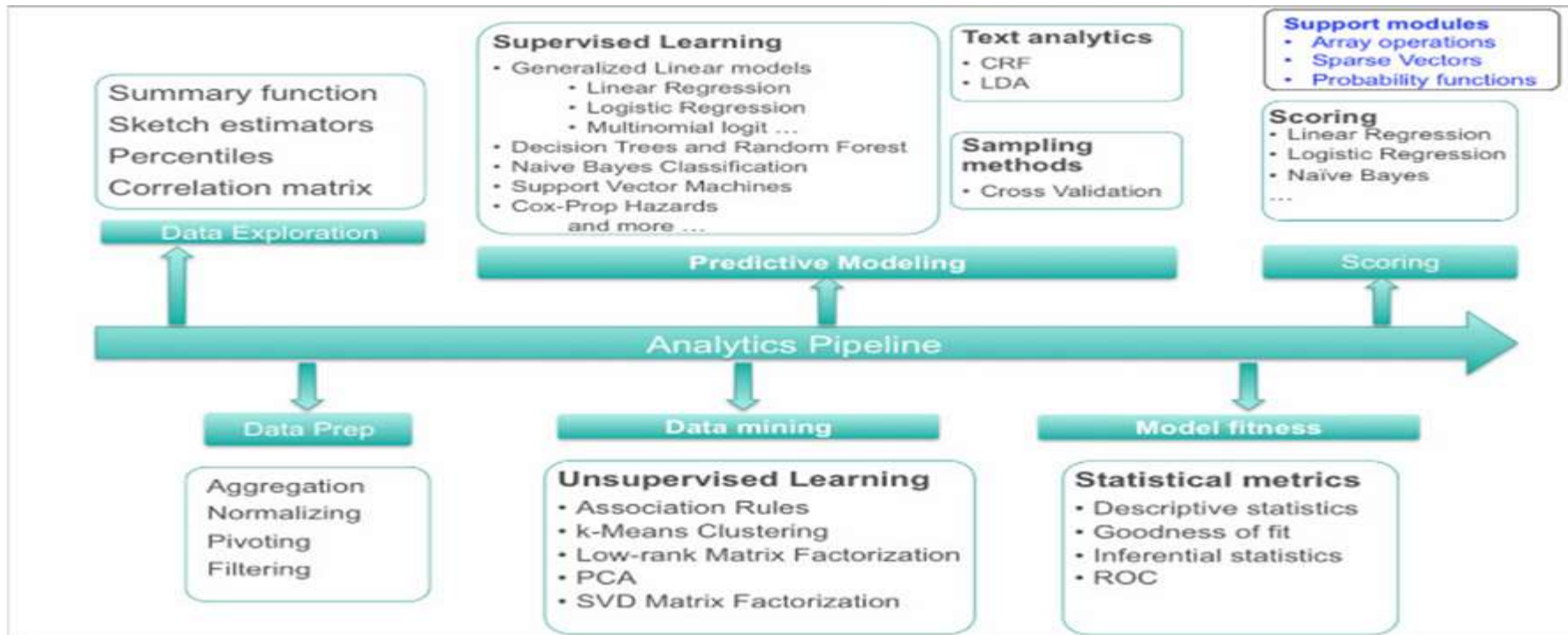
优土-智能广告(衍生需求)

- 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



详细链接

- 阅后即焚 - 流式处理
 - https://github.com/digoal/blog/blob/master/201711/20171126_01.md
 - https://github.com/digoal/blog/blob/master/201711/20171123_02.md
 - https://github.com/digoal/blog/blob/master/201711/20171107_33.md
 - https://github.com/digoal/blog/blob/master/201711/20171107_32.md
 - https://github.com/digoal/blog/blob/master/201711/20171107_28.md
- 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>

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

背景:
JSON很大,
包含一些标
准列、需要
提取拆分、
提高统计性
能、分区、
列存

游戏场
景埋点
feed实
时变化

实时写入动态结构

INSERT、
COPY

OSS

通过UDF转换为:
固定条件字段+其他JSON字段

JSON

A

+

B

+

JSON

支持列存储
分区、分片
JSON 索引

标准字段切分、列存储、
分区 等

实时分析, 举例(不限于此):

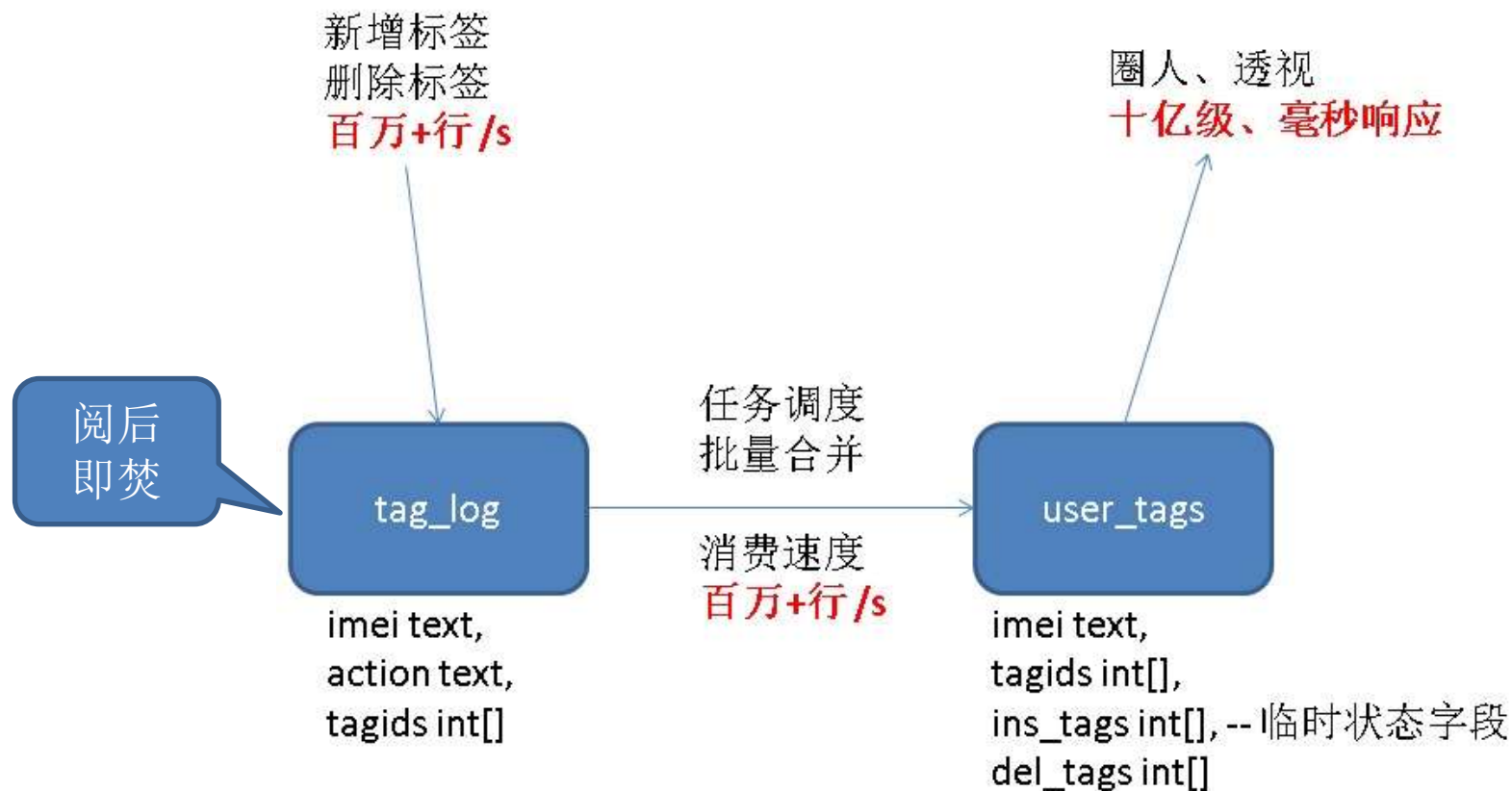
指定维度、日UV, 新增UV, ... 透视

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

详细链接

- <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 oXXo - 经营分析系统



详细链接

- https://github.com/digoal/blog/blob/master/201711/20171126_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: 9亿。(已上线)、1500亿数据。(DOING)
 - 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

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



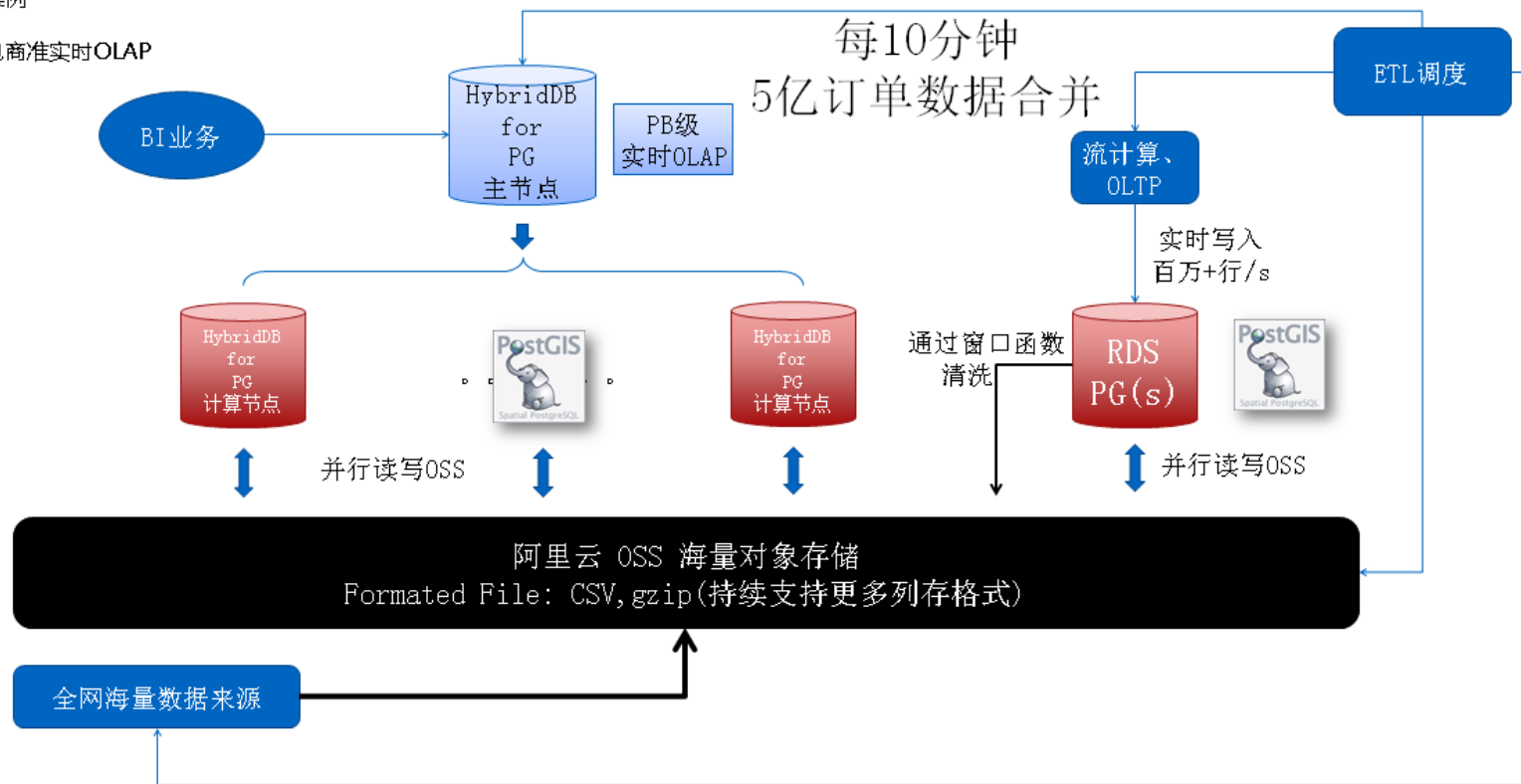
详细链接

- 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

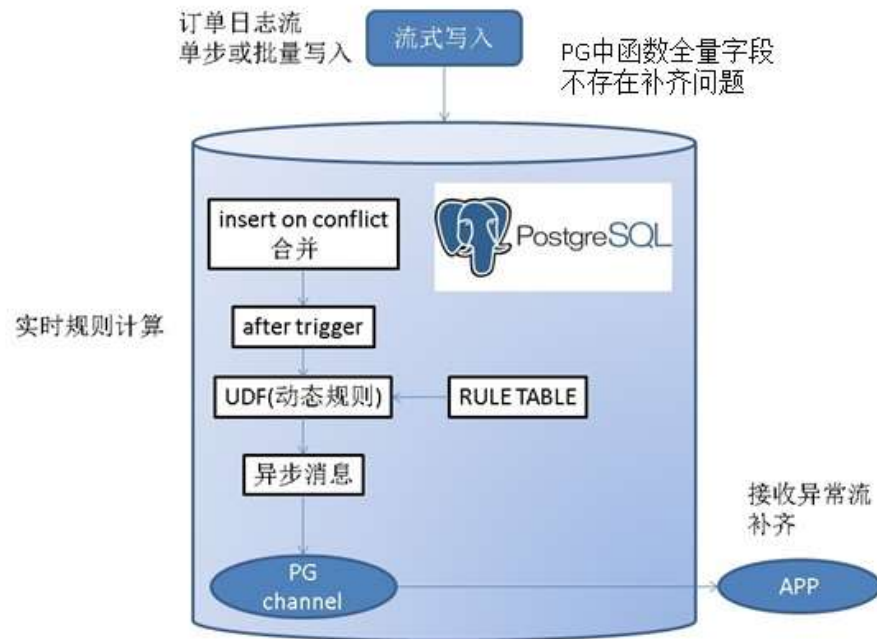
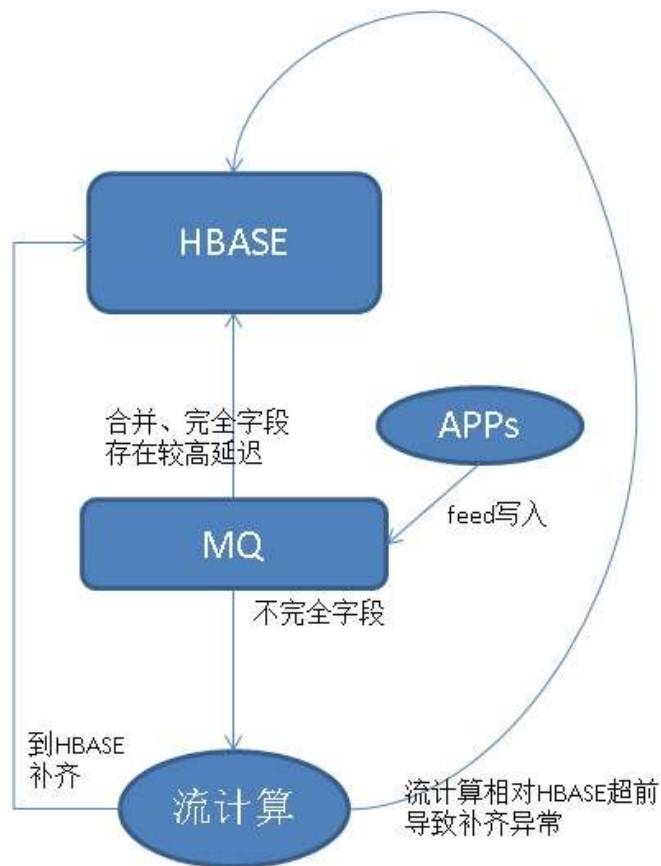
Case5 准实时订单分析系统(双十一业务)

案例

电商准实时OLAP



实时分析 - 架构演进



详细链接

- https://github.com/digoal/blog/blob/master/201707/20170728_01.md
- https://github.com/digoal/blog/blob/master/201711/20171111_01.md

详细链接

- 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

目录

- 产品介绍
- 生态介绍
- 应用案例
- 开发实践
- 参考文档

开发实践

- RDS PostgreSQL\PPAS
- HybridDB for PostgreSQL

场景映射与特性匹配

- 非结构化数据类型选择
 - json, hstore, xml类型
- 全文检索需求
 - 采用tsvector类型
- 模糊查询
 - 含前缀时, 使用b-tree索引
 - 含后缀时, 使用reverse(col) b-tree表达式索引
- 前后模糊
 - 采用pg_trgm插件, gin索引。
- 相似搜索
 - 采用pg_trgm插件, gin索引。
- 短文特征向量, 海明相似
 - 采用smlar插件, gin索引。
- 多字段任意搜索
 - 采用gin复合索引。
 - 采用多个单列索引。
- 多值类型搜索
 - 采用数组类型。
 - 采用gin索引。

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
 - 数据库互通。
 - file
 - oracle
 - mysql
 - sqlserver
 - hadoop.....

物化视图

- 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 ?;

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

字段加密

- create extension **pgcrypto**;
- digoal=# create table userpwd(userid int8 primary key, pwd text);
- CREATE TABLE
- digoal=# insert into userpwd (userid,pwd) values (1, crypt('this is a pwd source', gen_salt('bf',10)));
- 可逆加密
- 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

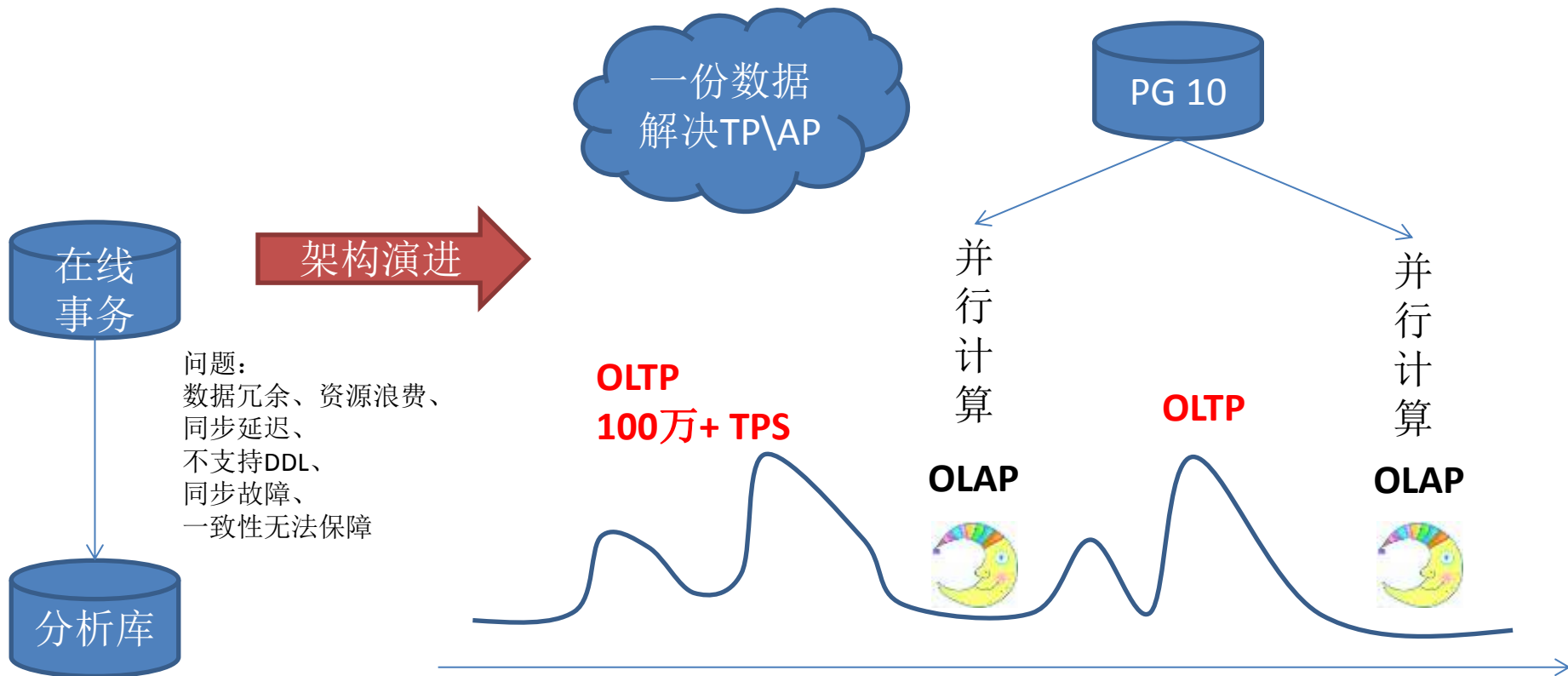
约束种类与用法

- 唯一, unique
- 非空, not null
- 排他, (例如, 空间不相交, 地图应用, 范围不相交, 边界限制。)
 - 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 &&)
 -);
- check, check(a>0);
- 外键,

数据去重大法

- https://github.com/digoal/blog/blob/master/201706/20170602_01.md

RDS PG 10 HTAP (一份数据)



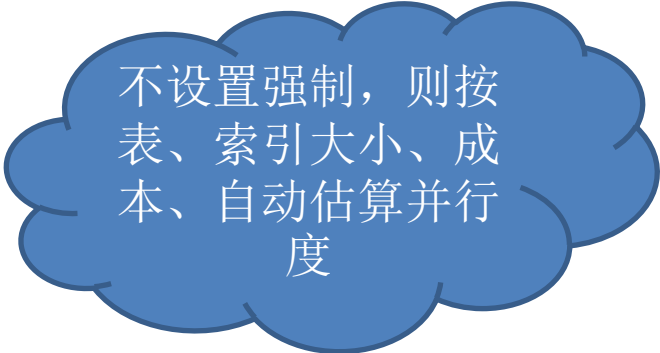
并行计算资源控制

- 单个并行节点并行度
 - `#max_parallel_workers_per_gather = 2`
- 全局并行度
 - `#max_parallel_workers = 8`

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=# 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

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;


KNN、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
 - UDF
 - https://github.com/digoal/blog/blob/master/201308/20130806_01.md



索引选择

- B-Tree
 - 等值、区间、排序
- Hash
 - 等值、LONG STRING
- GIN
 - 多值类型、倒排
 - 多列，任意列组合查询
- GiST
 - 空间、异构数据（范围）
- SP-GiST
 - 空间、异构数据
- BRIN
 - 线性数据、时序数据
- 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) = ?
- 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';



条件索引应用举例：
客服、文本质检
create index idx on tbl
where content !~ '怒
骂、。。。可枚举值'

b-tree复合索引

- 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
 - GIN
 - bloom

函数稳定性

- 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。

并发创建索引

- 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]

并发创建索引
不堵塞DML的方法

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`

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	

TOP SQL

- `create extension pg_stat_statements;`
- `select total_time,calls,total_time/calls,query
from pg_stat_statements order by 1 desc limit
5;`

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' now()-xact_start > interval '?s';`
- 长空闲事务
 - `select * from pg_stat_activity where state='idle in transaction' now()-xact_start > interval '?s';`
- 长2PC事务
 - `select * from pg_prepared_xacts where now()-prepared > interval '?s';`

历史慢SQL

[AWR](#)

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

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

日期时间: -

问题SQL

慢SQL执行计划详情

- auto_explain
- 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=9999999 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=9999999 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=9999999 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)
```

plpgsql函数诊断

- 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

- pldebugger extension + pgadmin
- 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)

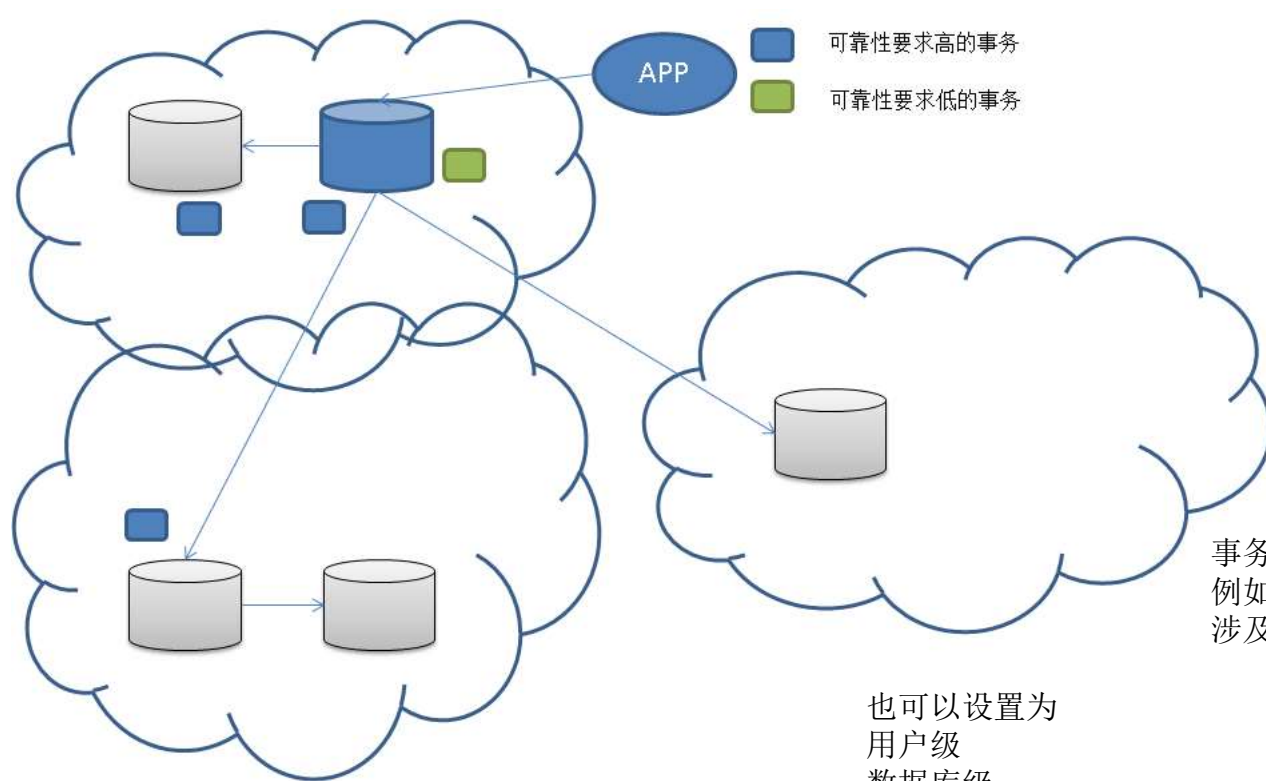
Name	Type	Description
ROW_COUNT	bigint	the number of rows processed by the most recent SQL command
RESULT_OID	oid	the OID of the last row inserted by the most recent SQL command (only useful after an INSERT command into a table having OIDs)
PG_CONTEXT	text	line(s) of text describing the current call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

可靠性与性能自由组合



鱼与熊
掌兼得

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
用户级
数据库级

https://github.com/digoal/blog/blob/master/201712/20171207_01.md

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)

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_transaction...

logical_replication_...

database_ages

table_ages

table_ages_remain

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)`

防雪崩

- `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

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 '?s' order by query_start offset ?;`
 - 或 `pg_cancel_backend(pid)`

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 username=? and pid<>pg_backend_pid();`
- 杀QUERY
 - `select pg_cancel_backend($pid);`

防DDoS

- auth_delay
 - auth_delay.milliseconds = '500'

是否被DDoS

- V1、`select count(*) from pg_stat_activity;`
- V2、`show max_connections;`
- V3、`netstat -anp | grep -c $xxxx`
- $V2 = V3 > V1$

数据同步

- DTS
- DATAX
- rds_dbsync

跨库访问

- dblink
 - PPAS支持PostgreSQL, Oracle 两种DBLINK

外部表

- 基于 dblink 的视图
 - PPAS支持PostgreSQL, Oracle 两种DBLINK
- postgres_fdw 外部表
- oracle_fdw 外部表

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

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
- DDL订阅
 - https://github.com/digoal/blog/blob/master/201712/20171204_04.md

执行计划

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

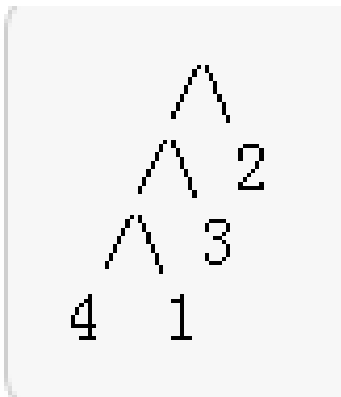
- Aggregate (cost=2.85..2.86 rows=1 width=8) (actual time=0.543..0.543 rows=1 loops=1)
- Output: count(*)
- Buffers: shared read=4
- -> Index Only Scan using a_pkey on public.a (cost=0.43..2.85 rows=1 width=0) (actual time=0.532..0.533 rows=1 loops=1)
- Output: id
- Index Cond: (a.id = 1)
- Heap Fetches: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

成本因子

- `#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
 - TSP (traveling salesman problem)



固定JOIN顺序

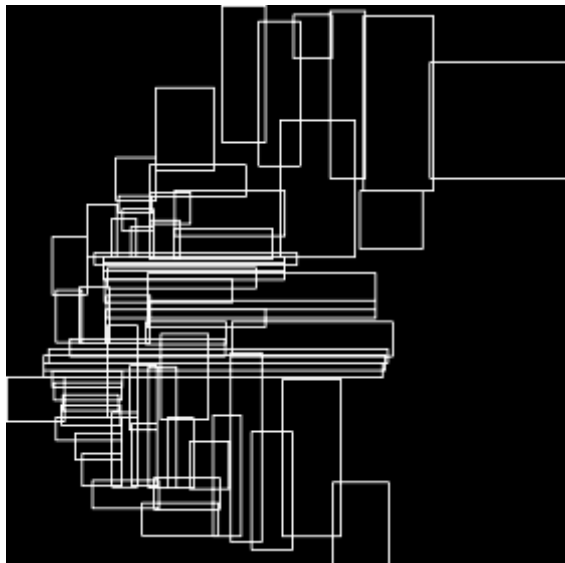
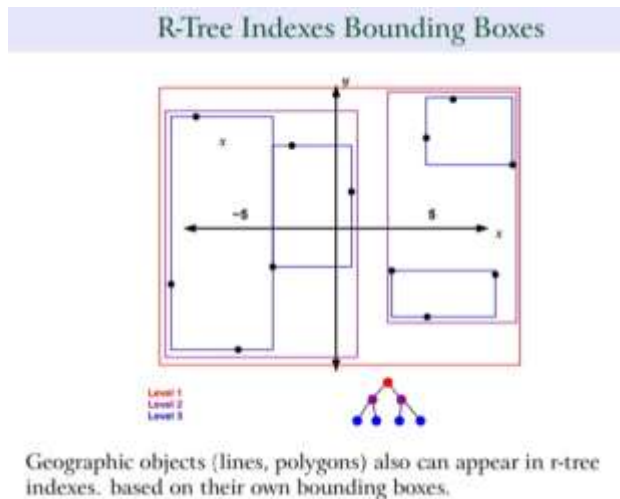
- 控制子查询提升
 - # from_collapse_limit = 8
- 控制显示INNER JOIN顺序
 - # join_collapse_limit = 8 # 1 disables
collapsing of explicit # JOIN clauses

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)

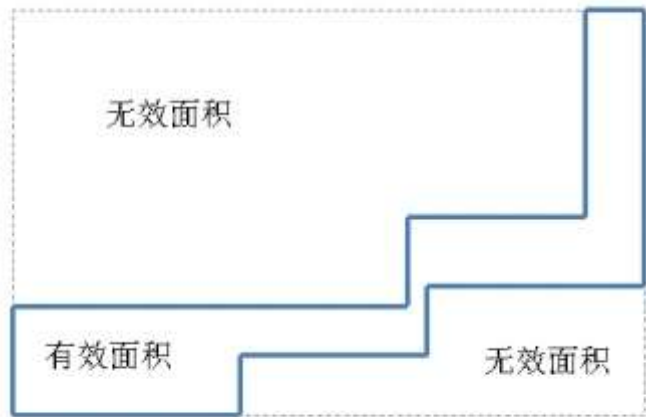
空间查询优化

- 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



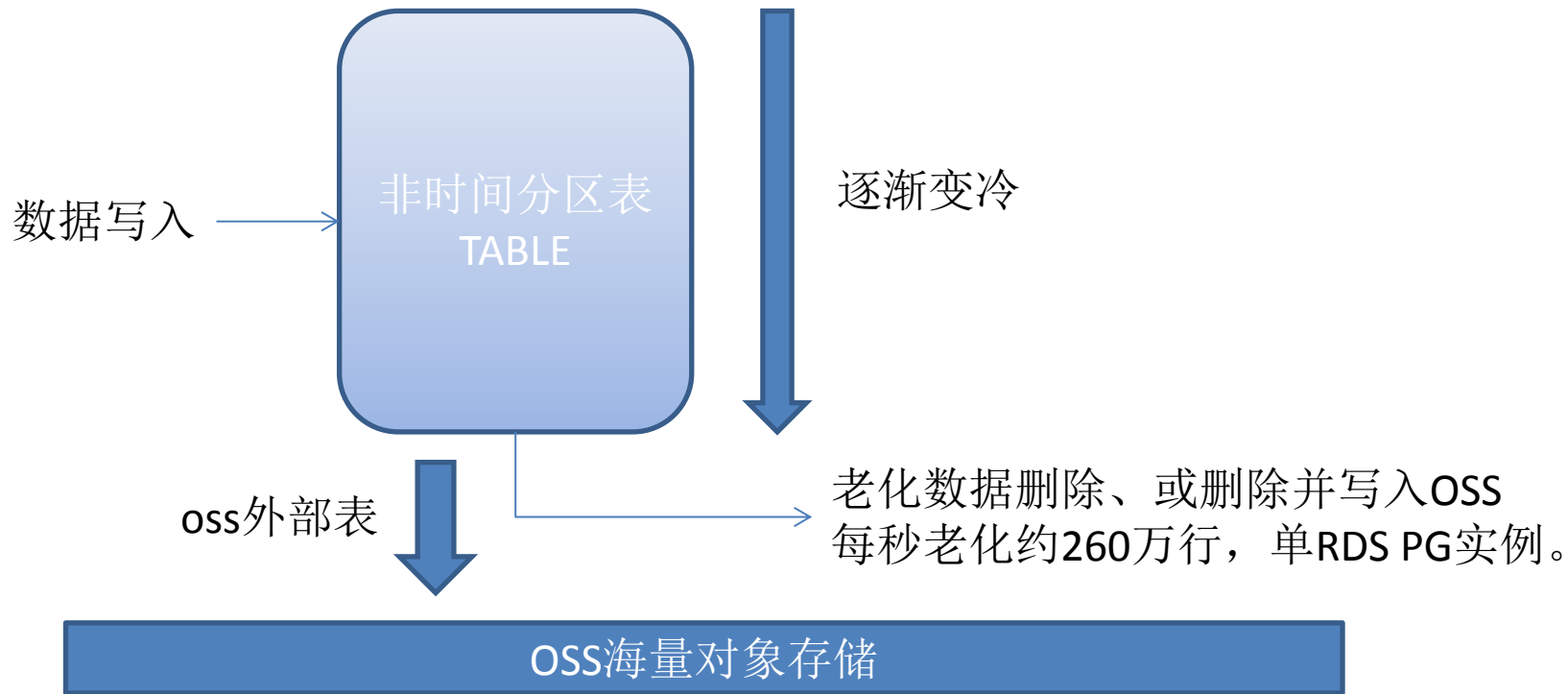
空间查询优化

- 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

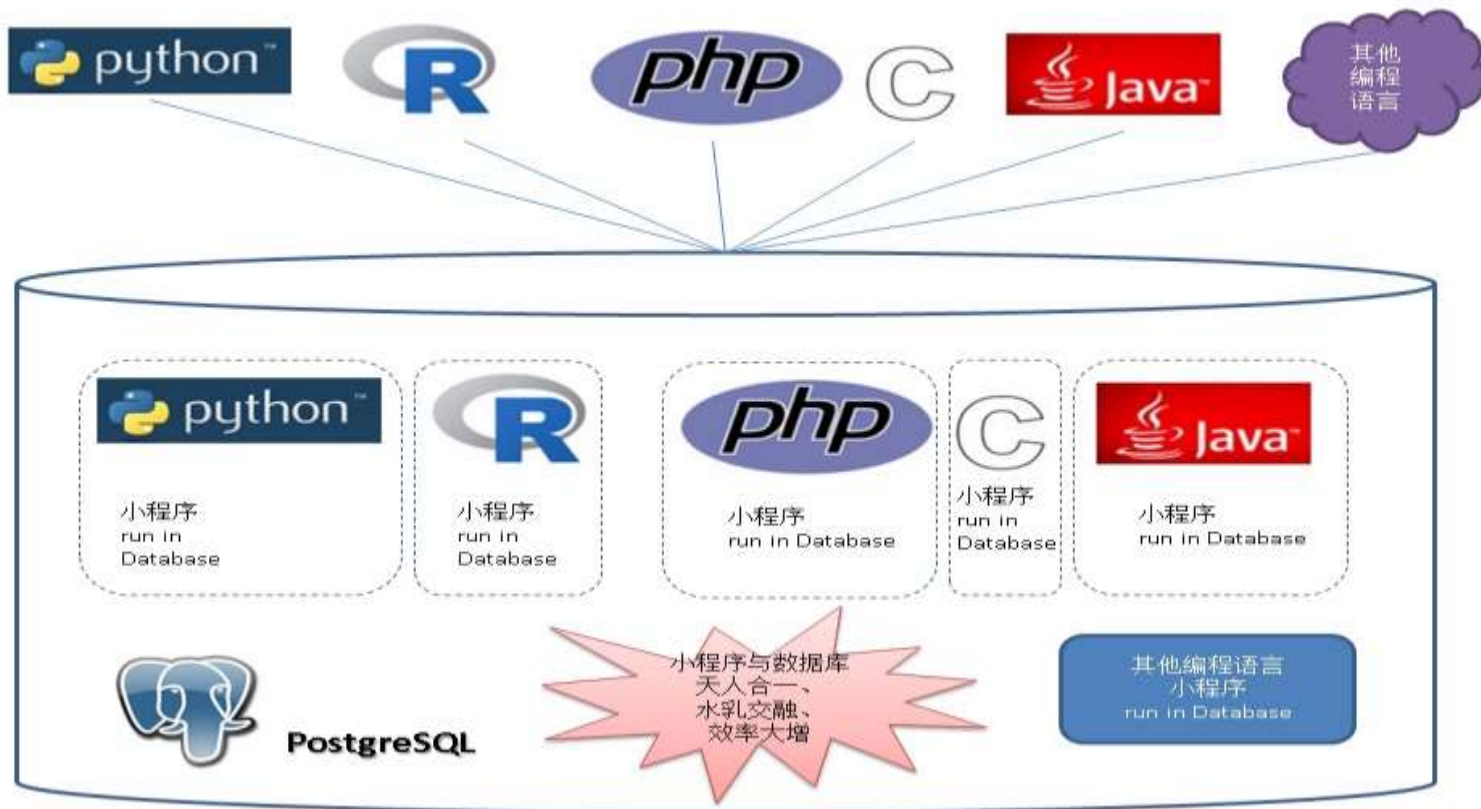
数据老化实践

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



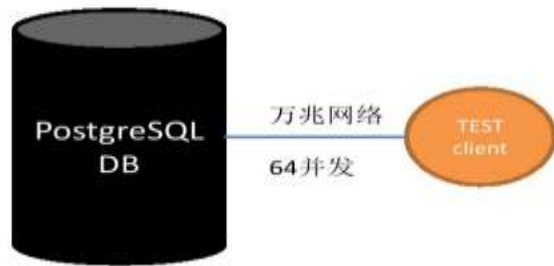
复杂业务逻辑延迟问题优化

- 一、
- 数
- 据
- 库
- 端
- 编
- 程



复杂业务逻辑延迟问题优化

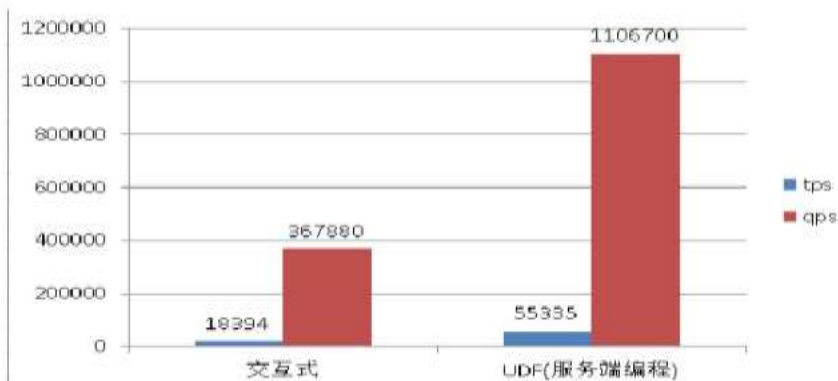
- 一、
- 数
- 据
- 库
- 端
- 编
- 程



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）



行级权限控制

- RLS
 - <https://www.postgresql.org/docs/10/static/sql-createpolicy.html>
 - 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

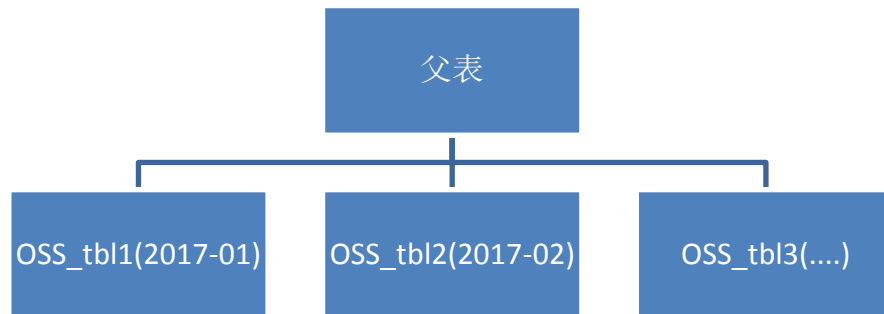
- GRANT { { SELECT | INSERT | UPDATE | REFERENCES } (*column_name* [, ...]) [, ...] | ALL [PRIVILEGES] (*column_name* [, ...]) } ON [TABLE] *table_name* [, ...] TO *role_specification* [, ...] [WITH GRANT OPTION]

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）
- 敏感信息加密
 - 对称加密
- 密码
 - 不可逆加密

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



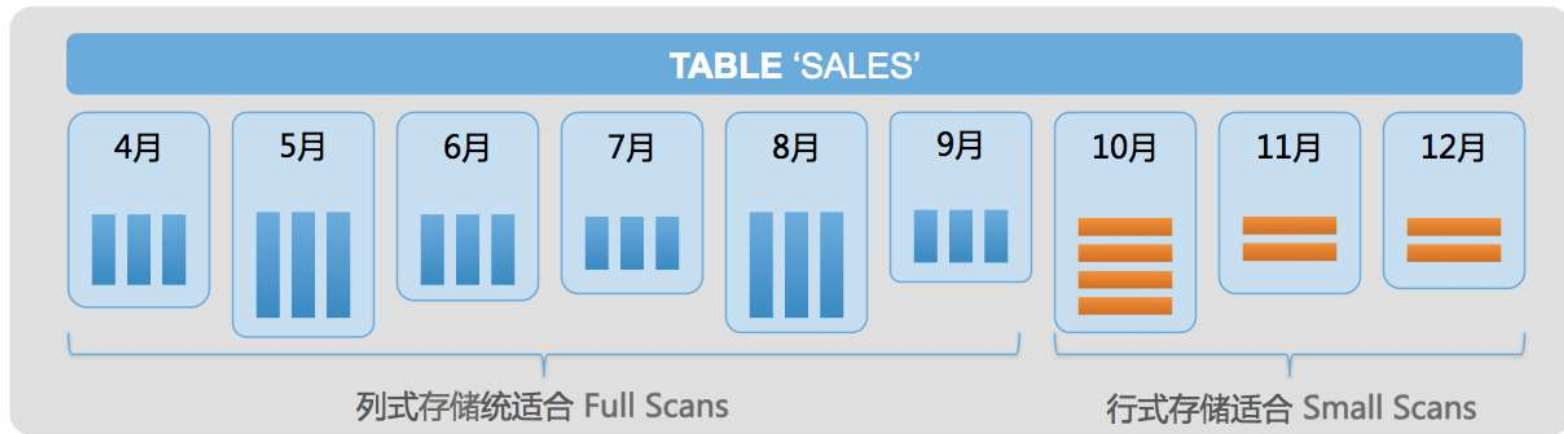
快速构造海量测试数据

- https://github.com/digoal/blog/blob/master/201711/20171121_01.md

开发实践

- RDS PostgreSQL\PPAS
- HybridDB for PostgreSQL

行存与列存的选择



- 支持索引类型:

- B-tree
- Bitmap
- GIST

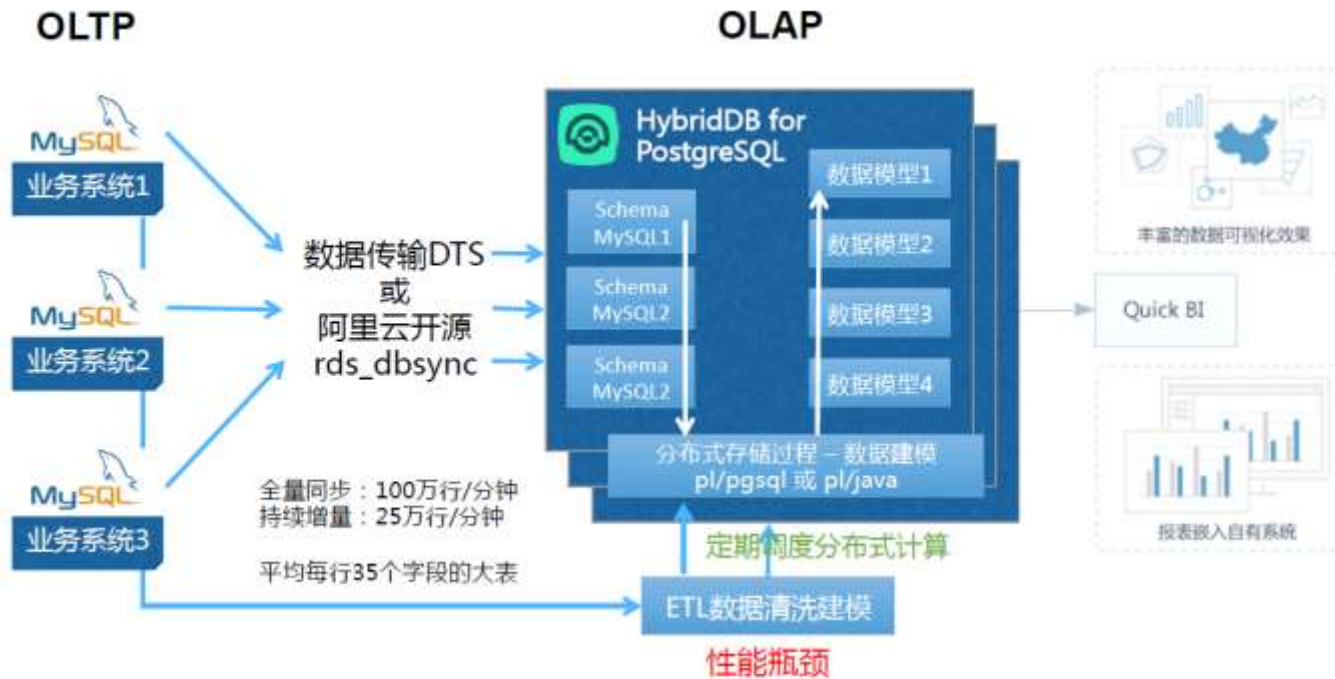
- 支持行列存储混合转换
- 支持按列存储数据库, 及列数据库索引
- 透明实时数据压缩类型

行存与列存的选择

- 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



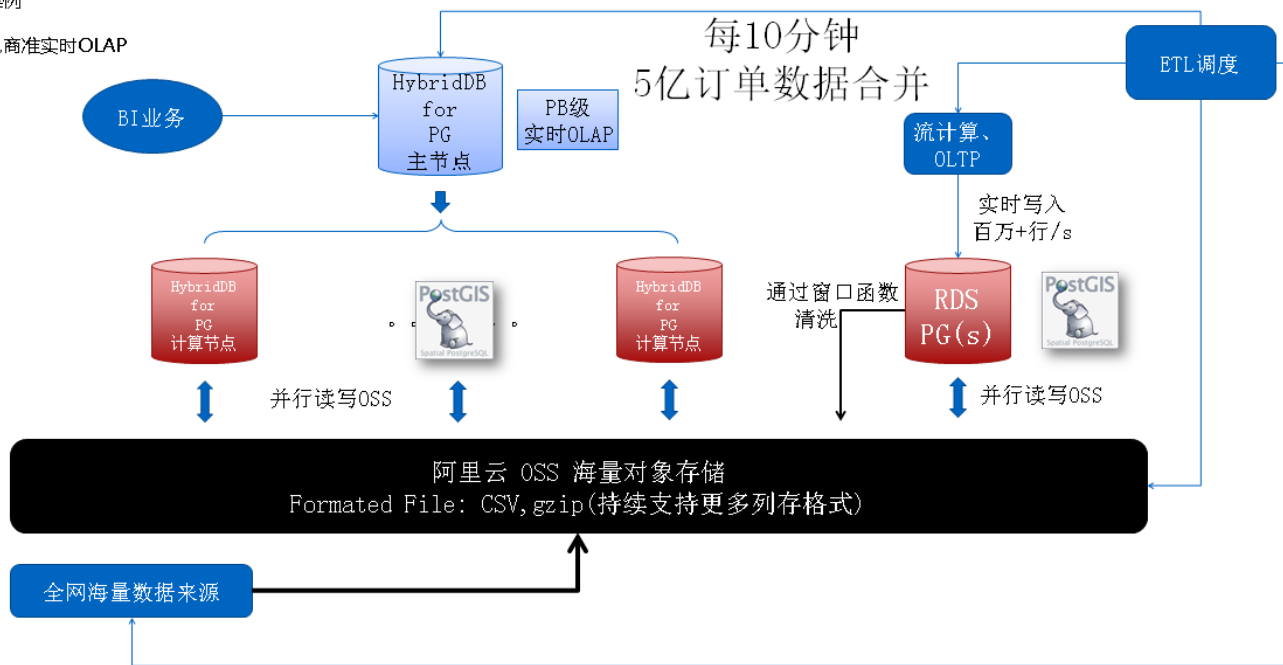
数据同步通道1



数据同步通道2

案例

电商准实时OLAP



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

非分布键 group by 和 distinct

- 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)

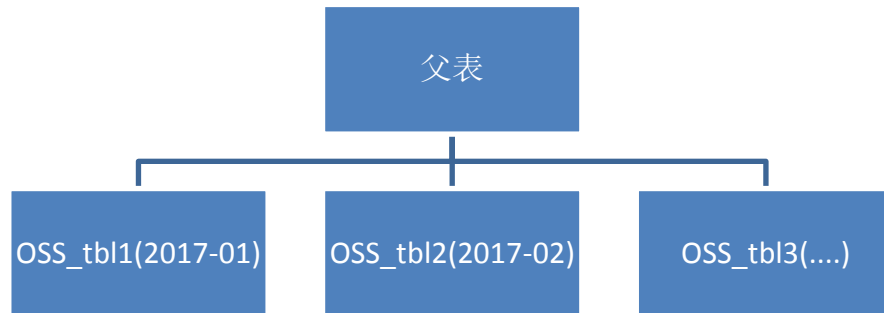
```

# 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](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

- GiST
  - 空间数据
- B-Tree
  - 等值、区间、排序
- Bitmap
  - 类似倒排
  - value: 所有行号对应的bitmap
  - 含100到1万个唯一值的列

# 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](https://github.com/digoal/blog/blob/master/201708/20170821_01.md)



资源使用  
隔离、  
控制

# 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](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）

2、sortKey

3、index

1.1、分区（list, range）

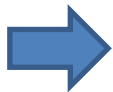


[https://help.aliyun.com/knowledge\\_detail/59195.html](https://help.aliyun.com/knowledge_detail/59195.html)  
详细介绍

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



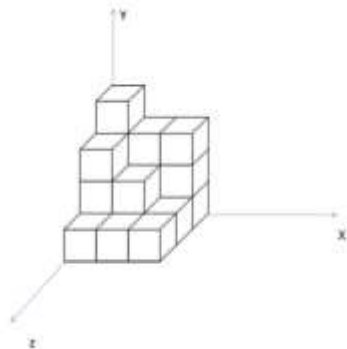
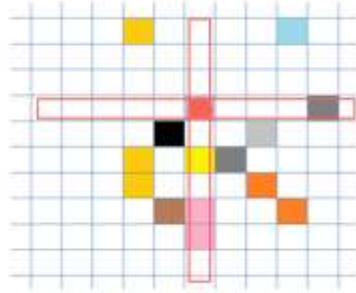
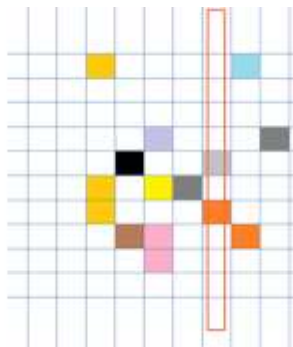
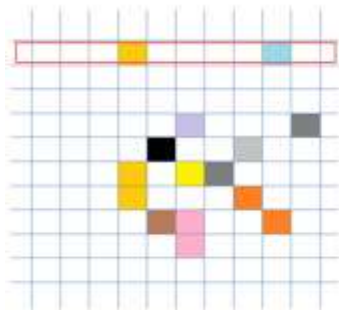
# metascan+sort Key+index实践

| 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

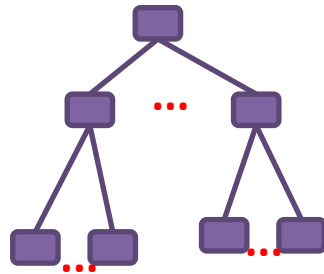
| 工单ID | 用户ID | 订单ID | 其他字段 | 其他字段 |
|------|------|------|------|------|
|      |      |      |      |      |

## 1、分布

### 1.1、分区

## 2、sortKey

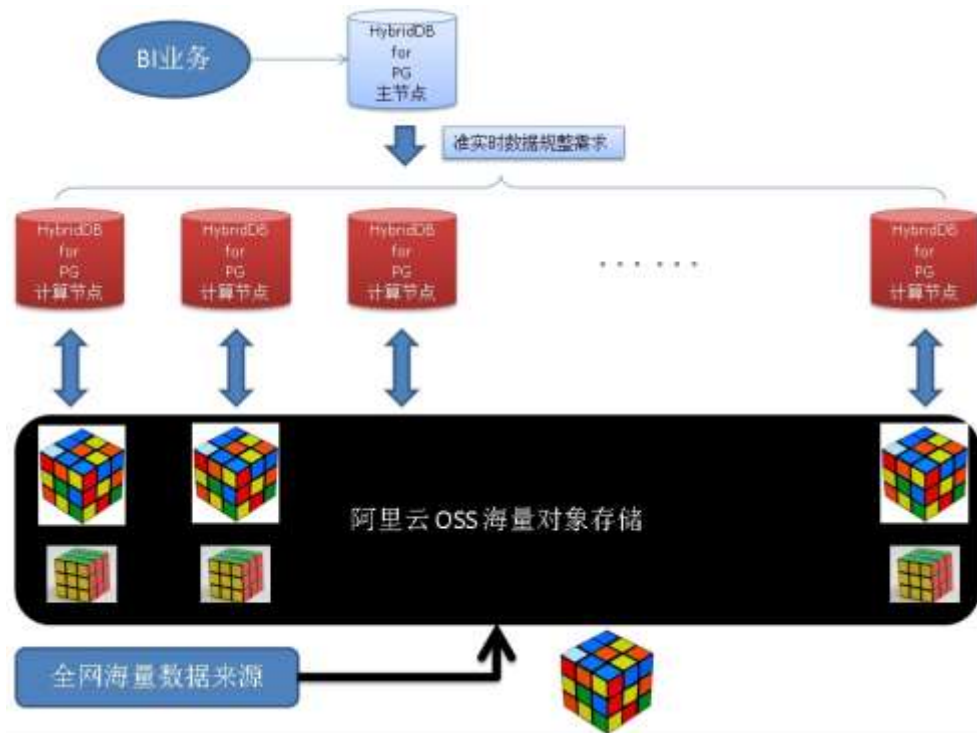
### 3、 index



# 大吞吐输出场景开发实践

[https://github.com/digoal/blog/blob/master/201707/20170726\\_01.md](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);
```

每日新增UV

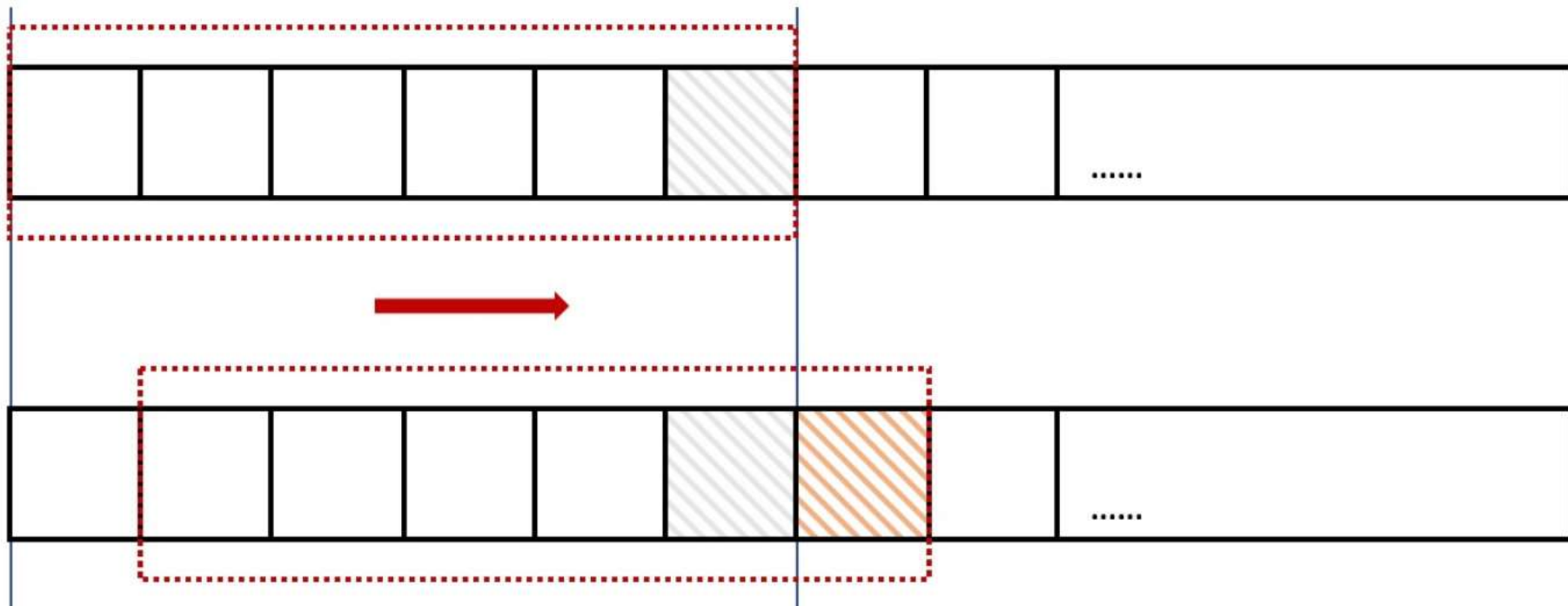
```
SELECT date, (#hll_union_agg(users) OVER two_days) - #users AS lost_uniques
FROM daily_uniques WINDOW two_days AS (ORDER BY date ASC ROWS 1 PRECEDING);
```

| Function        | Operator | Example                                                                                                                              |
|-----------------|----------|--------------------------------------------------------------------------------------------------------------------------------------|
| hll_add         |          | <pre>hll_add(users, hll_hash_integer(123))<br/>or<br/>users    hll_hash_integer(123)<br/>or<br/>hll_hash_integer(123)    users</pre> |
| hll_cardinality | #        | <pre>hll_cardinality(users)<br/>or<br/>#users</pre>                                                                                  |
| hll_union       |          | <pre>hll_union(male_users, female_users)<br/>or<br/>male_users    female_users<br/>or<br/>female_users    male_users</pre>           |

# 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



# 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/708/20170821\\_02.md](https://github.com/digoal/blog/708/20170821_02.md)



# 查看锁等待

- [https://github.com/digoal/blog/blob/master/201705/20170521\\_01.md](https://github.com/digoal/blog/blob/master/201705/20170521_01.md)



# 查看数据膨胀

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

# 目录

- 产品介绍
- 生态介绍
- 应用案例
- 开发实践
- 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



我这儿有本秘笈

那本书是本秘笈

# RDS PG+HybridDB PG+OSS

海量实时处理

CDN-离线分析  
CDN-域名违规整理

金融应用

平安  
风控

模糊查询、相似搜索、正则搜索

网络广告-域名搜索  
导购-实时盗文判定

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  
智能车院

任意字段实时搜索

网CRM

任意维度数据透视、钻取

设备、会员透视、精细化运营  
销售  
营销

时间、空间、对象多维搜索、透视

销售-选址、线上线下地推

# Thanks



我这儿有本秘笈

<https://yq.aliyun.com/articles/98539>

# 开发规约

- [https://github.com/digoal/blog/blob/master/201609/20160926\\_01.md](https://github.com/digoal/blog/blob/master/201609/20160926_01.md)

# 资料分享

- **# 实例管理**

- RDS PG管理客户端pgadmin: <https://www.pgadmin.org/download/>
- HDB PG管理客户端pgadmin:  
<https://www.postgresql.org/ftp/pgadmin/pgadmin3/v1.6.3/>
- PG 客户端驱动: <https://www.postgresql.org/docs/10/static/external-interfaces.html>

- **# 产品文档**

- RDS PG 产品文档: [https://help.aliyun.com/document\\_detail/26152.html](https://help.aliyun.com/document_detail/26152.html)
- HDB PG 产品文档:  
[https://help.aliyun.com/document\\_detail/49912.html](https://help.aliyun.com/document_detail/49912.html)



# 资料分享

- # 社区文档

- RDS PG 社区官方手册: <https://www.postgresql.org/docs/10/static/index.html>
- 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/>
- MADlib SQL机器学习库: <http://madlib.apache.org/documentation.html>
- PostGIS文档: <http://postgis.net/docs/manual-2.3/>
- PostGIS入门: <http://workshops.boundlessgeo.com/postgis-intro/>
- pgrouting文档: <http://pgrouting.org/documentation.html>
- PG开发手册: <http://www.postgresqltutorial.com/>
- <https://www.tutorialspoint.com/postgresql/>

- # 开发指南

- Java: [https://github.com/digoal/blog/blob/master/201701/20170106\\_05.md](https://github.com/digoal/blog/blob/master/201701/20170106_05.md)
- PHP: [https://github.com/digoal/blog/blob/master/201701/20170106\\_08.md](https://github.com/digoal/blog/blob/master/201701/20170106_08.md)
- Ruby: [https://github.com/digoal/blog/blob/master/201701/20170106\\_07.md](https://github.com/digoal/blog/blob/master/201701/20170106_07.md)
- Python: [https://github.com/digoal/blog/blob/master/201701/20170106\\_06.md](https://github.com/digoal/blog/blob/master/201701/20170106_06.md)
- C: [https://github.com/digoal/blog/blob/master/201701/20170106\\_09.md](https://github.com/digoal/blog/blob/master/201701/20170106_09.md)

# 资料分享

- # 最佳实践
- HDB PG OSS 外部表文档: [https://help.aliyun.com/document\\_detail/35457.html](https://help.aliyun.com/document_detail/35457.html)
- HDB PG 批量更新 (Upsert) 方法: <https://yq.aliyun.com/articles/86604>
- HDB PG 分区键的选择: [https://github.com/digoal/blog/blob/master/201607/20160719\\_02.md](https://github.com/digoal/blog/blob/master/201607/20160719_02.md)
- HDB PG 数据倾斜的监测和实践:  
[https://github.com/digoal/blog/blob/master/201708/20170821\\_02.md](https://github.com/digoal/blog/blob/master/201708/20170821_02.md)
- RDS PG OSS 外部表文档: [https://help.aliyun.com/knowledge\\_detail/43352.html](https://help.aliyun.com/knowledge_detail/43352.html)
- HDB PG 负载管理(资源队列管理):  
[https://github.com/digoal/blog/blob/master/201708/20170821\\_01.md](https://github.com/digoal/blog/blob/master/201708/20170821_01.md)
- mysql->RDS PG 全量、增量同步: [https://github.com/aliyun/rds\\_dbsync](https://github.com/aliyun/rds_dbsync)
- mysql->RDS PG 全量、增量同步: <https://help.aliyun.com/product/26590.html>
- RDS PG TOP SQL 分析: [https://github.com/digoal/blog/blob/master/201704/20170424\\_06.md](https://github.com/digoal/blog/blob/master/201704/20170424_06.md)

# 资料分享

- # 最佳实践
- 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](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](https://github.com/digoal/blog/blob/master/201608/20160825_02.md)
- UPSERT的用法：[https://github.com/digoal/blog/blob/master/201704/20170424\\_04.md](https://github.com/digoal/blog/blob/master/201704/20170424_04.md)
- sharding 开发框架：
- <https://github.com/dangdangdotcom/sharding-jdbc>
- <https://github.com/go-pg/sharding/>
- PostGIS例子：<http://revenant.ca/www/postgis/workshop/indexing.html>
- 锁等待查询：[https://github.com/digoal/blog/blob/master/201705/20170521\\_01.md](https://github.com/digoal/blog/blob/master/201705/20170521_01.md)
- 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://github.com/digoal/blog/blob/master/201706/20170601_02.md)