DTCC 2010-2022

第十三届中国数据库技术大会

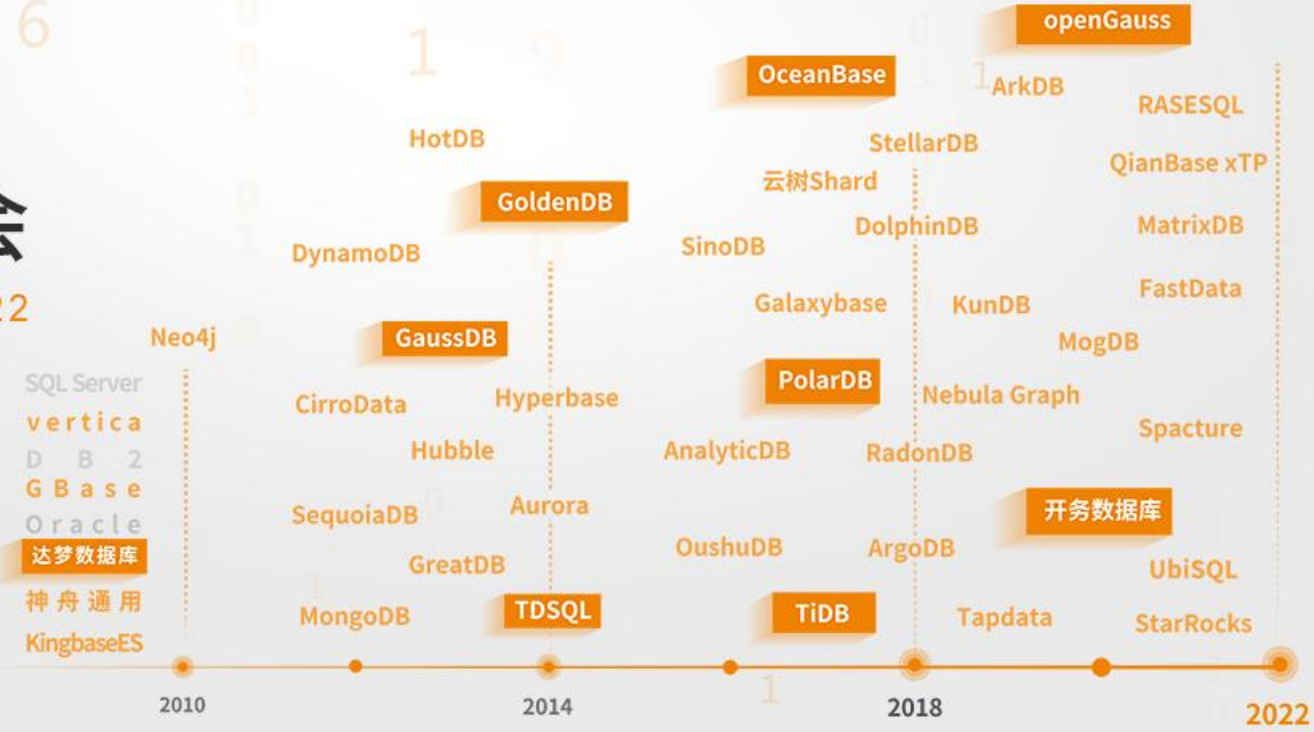
DATABASE TECHNOLOGY CONFERENCE CHINA 2022

数据智能 价值创新

IT168.com ChinaUnix.net ITPUB

线上直播 | 2022/12/14-16

数据来源：数据库产品上市商用时间



PostgreSQL之SQL优化小技巧

傅强
美创科技DBA

目录

CONTENTS

01 表的扫描方式
02 表的连接方式
03 SQL改写
04 数据库配置
05 架构设计
06 相关工具

表的扫描方式

使用合适的表扫描方式

扫描方式简称	扫描方式说明
Seq Scan	顺序扫描整个对象
Parallel Seq Scan	采用并行方式顺序扫描整个对象
Index Scan	采用离散读的方式，利用索引访问某个对象
Index Only Scan	仅通过索引，不访问表快速访问某个对象
Bitmap Index Scan	通过多个索引扫描后形成位图找到符合条件的数据
Bitmap Heap Scan	往往跟随bitmap index scan，使用该扫描生成的位图访问对象

表的扫描方式

使用索引避免表扫描

- 在等值或范围查询、排序及分组查询时使用索引字段，以尽量避免表扫描

```
postgres=# explain analyze select * from tbl_index where a = 557858;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on tbl_index  (cost=0.00..18870.00 rows=2 width=21) (actual time=0.036..671.703 rows=1 loops=1)
   Filter: (a = 557858)
   Rows Removed by Filter: 999999
 Planning Time: 0.344 ms
 Execution Time: 671.862 ms
(5 rows)
```

```
postgres=# explain analyze select * from tbl_index where a = 557858;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Index Scan using tbl_index_a on tbl_index  (cost=0.42..12.46 rows=2 width=21) (actual time=0.145..0.147 rows=1 loops=1)
   Index Cond: (a = 557858)
 Planning Time: 0.367 ms
 Execution Time: 0.235 ms
(4 rows)
```

- 减少使用导致索引扫描失效的SQL语句书写方式（函数，表达式等）

```
postgres=# explain analyze select * from tbl_index where a::varchar = '557858';
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Seq Scan on tbl_index  (cost=0.00..23870.00 rows=5000 width=21) (actual time=0.056..769.264 rows=1 loops=1)
   Filter: (((a)::character varying)::text = '557858'::text)
   Rows Removed by Filter: 999999
 Planning Time: 0.658 ms
 Execution Time: 769.335 ms
(5 rows)
```

表的扫描方式

索引失效

- 索引类型不匹配
- collate不一致
- 数据类型不一致
- 表中数据量少时
- 索引字段在表中占比较高

表的扫描方式

合理使用并行扫描

```
postgres=# explain analyze select * from test where id < 10000;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on test  (cost=0.00..169248.60 rows=10539 width=8) (actual time=0.053..2305.860 rows=9999 loops=1)
   Filter: (id < 10000)
   Rows Removed by Filter: 9990001
 Planning Time: 1.076 ms
 Execution Time: 2307.000 ms
(5 rows)
```

set max_parallel_workers_per_gather = 8;

```
postgres=# explain analyze select * from test where id < 10000;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..77552.05 rows=10539 width=8) (actual time=7.597..831.958 rows=9999 loops=1)
   Workers Planned: 4
   Workers Launched: 4
   ->  Parallel Seq Scan on test  (cost=0.00..75498.15 rows=2635 width=8) (actual time=627.179..787.198 rows=2000 loops=5)
         Filter: (id < 10000)
         Rows Removed by Filter: 1998000
 Planning Time: 0.193 ms
 Execution Time: 845.272 ms
(8 rows)
```

表的扫描方式

使用合理的索引类型

索引类型（按结构划分）

- **B-tree索引：**适合所有的数据类型，支持排序，支持< <= = >= > BETWEEN、IN、IS NULL、is not null等条件。
- **Hash索引：**只能处理等值查询，不写入xlog，数据库崩溃后丢失，不建议使用。
- **GiST索引：**常用于几何、范围、空间等类型（包含、相交、距离排序等）。
- **SP-GiST索引：**适合使用KD树，四叉树，基数树算法的场景。
- **GIN索引：**常用于多值类型，例如数组、全文检索、分词、模糊查询。
- **BRIN索引：**针对数据块级别的索引，索引占用空间非常小，使用与btree索引类似，适用于序列、ctime等于数据物理存储相关性很强的字段的范围查询。

表的扫描方式

索引类型（按用途划分）

- **唯一索引：**字段不允许相同值，主键会自动创建一个唯一索引，空值不等于空值。
- **多列索引：**针对多个字段创建索引，原则：选择度高的字段放到前面性能会更好。
- **函数索引：**基于某个函数或者表达式的值创建的索引。
- **部分索引：**只需要查询表中部分数据的需求，建部分索引能够缩小索引体积，性能更优。查询必须添加部分索引中的筛选条件。
- **覆盖索引：**只需要访问索引的数据就能获得需要的结果，而不需要再次访问表中的数据。

表的扫描方式

- 计算

```
postgres=# explain analyze select * from tbl_index where a + 1 = 100;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on tbl_index  (cost=0.00..21370.00 rows=5000 width=21) (actual time=293.704..293.725 rows=0 loops=1)
   Filter: ((a + 1) = 100)
   Rows Removed by Filter: 1000000
 Planning Time: 0.282 ms
 Execution Time: 293.856 ms
(5 rows)
```

添加索引

```
postgres=# create index tbl_index_a_1 on tbl_index using btree ((a+1));

postgres=# explain analyze select * from tbl_index where a + 1 = 100;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on tbl_index  (cost=91.17..6441.55 rows=5000 width=21) (actual time=0.130..0.131 rows=0 loops=1)
   Recheck Cond: ((a + 1) = 100)
   ->  Bitmap Index Scan on tbl_index_a_1  (cost=0.00..89.92 rows=5000 width=0) (actual time=0.119..0.119 rows=0 loops=1)
         Index Cond: ((a + 1) = 100)
 Planning Time: 0.586 ms
 Execution Time: 0.219 ms
(6 rows)
```

表的扫描方式

- 类型转换

  添加索引

```
postgres=# create index tbl_index_a on tbl_index using btree ((a::varchar));

postgres=# explain analyze select * from tbl_index where a::varchar = '557858';
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on tbl_index  (cost=91.17..6454.05 rows=5000 width=21) (actual time=0.107..0.109 rows=1 loops=1)
   Recheck Cond: (((a)::character varying)::text = '557858'::text)
   Heap Blocks: exact=1
   ->  Bitmap Index Scan on tbl_index_a  (cost=0.00..89.92 rows=5000 width=0) (actual time=0.086..0.087 rows=1 loops=1)
         Index Cond: (((a)::character varying)::text = '557858'::text)
 Planning Time: 0.587 ms
 Execution Time: 0.153 ms
(7 rows)
```

表的连接方式

使用正确的表连接方式

类别	Nested Loop	Hash Join	Merge Join
使用条件	• 小表作为驱动表 • 被驱动表有索引	• 小表用于构造hash桶 • Hash Join不依赖于索引	• 小表作为驱动表 • 适用于很大的表Join
优点	当有高选择性索引时，效率比较高。	当缺乏索引或者索引条件模糊时，Hash Join比Nested Loop有效。通常比Merge Join快。	当缺乏索引或者索引条件模糊时，Merge Join比Nested Loop有效。
缺点	返回的结果集大，效率低。	需要大量内存。	所有表都需要排序。

表的连接方式


```
postgres=# explain select t1.* from test1 t1, test2 t2 where t1.id = t2.id;
                          QUERY PLAN
------------------------------------------------------------------------
 Hash Join  (cost=28.50..19278.50 rows=1000 width=15)
   Hash Cond: (t2.id = t1.id)
   ->  Seq Scan on test2 t2  (cost=0.00..15490.00 rows=1000000 width=4)
   ->  Hash  (cost=16.00..16.00 rows=1000 width=15)
         ->  Seq Scan on test1 t1  (cost=0.00..16.00 rows=1000 width=15)
(5 rows)
```

添加索引

```
postgres=# create index idx_test1 on test1(id);
postgres=# create index idx_test2 on test2(id);
postgres=# explain select t1.* from test1 t1, test2 t2 where t1.id = t2.id;
                                QUERY PLAN
---------------------------------------------------------------------------------------------
 Merge Join  (cost=1.21..86.05 rows=1000 width=15)
   Merge Cond: (t1.id = t2.id)
   ->  Index Scan using idx_test1 on test1 t1  (cost=0.28..44.27 rows=1000 width=15)
   ->  Index Only Scan using idx_test2 on test2 t2  (cost=0.42..25980.42 rows=1000000 width=4)
(4 rows)
```

SQL改写

- UPDATE中包含子查询

```
postgres=# explain (analyze, verbose, timing, buffers) update t1 set info = (select info from t2 where t1.id = t2.id) where t1.id < 9999;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Update on public.t1  (cost=0.00..175135.00 rows=0 width=0) (actual time=3160.258..3160.261 rows=0 loops=1)
   Buffers: shared hit=80118 dirtied=44 written=44
   ->  Seq Scan on public.t1  (cost=0.00..175135.00 rows=9998 width=38) (actual time=0.412..2928.959 rows=9998 loops=1)
         Output: (SubPlan 1), t1.ctid
         Filter: (t1.id < 9999)
         Rows Removed by Filter: 2
         Buffers: shared hit=50035
         SubPlan 1
           ->  Seq Scan on public.t2  (cost=0.00..17.50 rows=1 width=32) (actual time=0.264..0.277 rows=0 loops=9998)
                 Output: t2.info
                 Filter: (t1.id = t2.id)
                 Rows Removed by Filter: 1000
                 Buffers: shared hit=49990
 Planning:
   Buffers: shared hit=19
 Planning Time: 0.761 ms
 Execution Time: 3160.871 ms
(17 rows)
```

SQL改写

将SQL修改为update ... set ... from ... where ...

```
postgres=# explain (analyze, verbose, timing, buffers) update t1 set info = t2.info from t2 where t1.id = t2.id and t1.id < 9999;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Update on public.t1  (cost=27.50..288.99 rows=0 width=0) (actual time=9.784..9.788 rows=0 loops=1)
   Buffers: shared hit=3061
   ->  Hash Join  (cost=27.50..288.99 rows=1000 width=44) (actual time=0.480..4.998 rows=1000 loops=1)
         Output: t2.info, t1.ctid, t2.ctid
         Hash Cond: (t1.id = t2.id)
         Buffers: shared hit=94
         ->  Seq Scan on public.t1  (cost=0.00..214.00 rows=9998 width=10) (actual time=0.093..2.726 rows=9998 loops=1)
               Output: t1.ctid, t1.id
               Filter: (t1.id < 9999)
               Rows Removed by Filter: 2
               Buffers: shared hit=89
         ->  Hash  (cost=15.00..15.00 rows=1000 width=42) (actual time=0.370..0.371 rows=1000 loops=1)
               Output: t2.info, t2.ctid, t2.id
               Buckets: 1024  Batches: 1  Memory Usage: 51kB
               Buffers: shared hit=5
               ->  Seq Scan on public.t2  (cost=0.00..15.00 rows=1000 width=42) (actual time=0.015..0.175 rows=1000 loops=1)
                     Output: t2.info, t2.ctid, t2.id
                     Buffers: shared hit=5
 Planning:
   Buffers: shared hit=44
 Planning Time: 0.833 ms
 Execution Time: 9.926 ms
(22 rows)
```

SQL改写

- 标量子查询

```
postgres=# explain analyze select t1.id,
postgres-# (select t2.num from t2 where t2.id = t1.id) as num
postgres-# from t1;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Seq Scan on t1  (cost=0.00..18515489.00 rows=1000000 width=36) (actual time=0.685..127992.907 rows=1000000 loops=1)
   SubPlan 1
     ->  Seq Scan on t2  (cost=0.00..18.50 rows=1 width=11) (actual time=0.122..0.122 rows=0 loops=1000000)
           Filter: (id = t1.id)
           Rows Removed by Filter: 1000
 Planning Time: 2.558 ms
 Execution Time: 128286.385 ms
(7 rows)
```

改写成外连接

```
postgres=# explain analyze select t1.id, t2.num
postgres-# from t1
postgres-# left join t2 on (t2.id = t1.id);
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Hash Left Join  (cost=28.50..19277.50 rows=1000000 width=15) (actual time=1.948..659.592 rows=1000000 loops=1)
   Hash Cond: (t1.id = t2.id)
   ->  Seq Scan on t1  (cost=0.00..15489.00 rows=1000000 width=4) (actual time=0.026..221.160 rows=1000000 loops=1)
   ->  Hash  (cost=16.00..16.00 rows=1000 width=15) (actual time=1.513..1.514 rows=1000 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 56kB
         ->  Seq Scan on t2  (cost=0.00..16.00 rows=1000 width=15) (actual time=0.044..0.423 rows=1000 loops=1)
 Planning Time: 2.462 ms
 Execution Time: 753.455 ms
(8 rows)
```

SQL改写

- 视图合并

```
postgres=# explain analyze select v.id, v1.num
postgres-# from (select t1.id from t1, t2 where t1.id = t2.id) v, (select t3.id, t3.num from t3 where id < 100) v1
postgres-# where v.id = v1.id;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=3.25..1831.17 rows=1 width=15) (actual time=0.321..71.495 rows=99 loops=1)
   Join Filter: (t1.id = t2.id)
   Rows Removed by Join Filter: 98901
   ->  Hash Join  (cost=3.25..1802.67 rows=1 width=19) (actual time=0.265..26.296 rows=99 loops=1)
         Hash Cond: (t3.id = t1.id)
         ->  Seq Scan on t3  (cost=0.00..1799.00 rows=109 width=15) (actual time=0.086..25.853 rows=99 loops=1)
               Filter: (id < 100)
               Rows Removed by Filter: 99901
         ->  Hash  (cost=2.00..2.00 rows=100 width=4) (actual time=0.147..0.152 rows=100 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 12kB
               ->  Seq Scan on t1  (cost=0.00..2.00 rows=100 width=4) (actual time=0.037..0.079 rows=100 loops=1)
   ->  Seq Scan on t2  (cost=0.00..16.00 rows=1000 width=4) (actual time=0.015..0.237 rows=1000 loops=99)
 Planning Time: 0.566 ms
 Execution Time: 71.636 ms
(14 rows)
```

SQL改写

加上group by后，子查询被固化，视图没有发生合并

```
postgres=# explain analyze select v.id, v1.num
postgres-# from (select t1.id from t1, t2 where t1.id = t2.id group by t1.id) v, (select t3.id, t3.num from t3 where id < 100) v1
postgres-# where v.id = v1.id;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=27.50..1826.79 rows=1 width=15) (actual time=0.904..21.774 rows=99 loops=1)
   Hash Cond: (t3.id = t1.id)
   ->  Seq Scan on t3  (cost=0.00..1799.00 rows=109 width=15) (actual time=0.044..20.824 rows=99 loops=1)
         Filter: (id < 100)
         Rows Removed by Filter: 99901
   ->  Hash  (cost=26.25..26.25 rows=100 width=4) (actual time=0.839..0.848 rows=100 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 12kB
         ->  HashAggregate  (cost=24.25..25.25 rows=100 width=4) (actual time=0.750..0.797 rows=100 loops=1)
               Group Key: t1.id
               Batches: 1  Memory Usage: 24kB
               ->  Hash Join  (cost=3.25..24.00 rows=100 width=4) (actual time=0.162..0.691 rows=100 loops=1)
                     Hash Cond: (t2.id = t1.id)
                     ->  Seq Scan on t2  (cost=0.00..16.00 rows=1000 width=4) (actual time=0.024..0.246 rows=1000 loops=1)
                     ->  Hash  (cost=2.00..2.00 rows=100 width=4) (actual time=0.091..0.094 rows=100 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 12kB
                           ->  Seq Scan on t1  (cost=0.00..2.00 rows=100 width=4) (actual time=0.023..0.050 rows=100 loops=1)
 Planning Time: 0.637 ms
 Execution Time: 21.918 ms
(18 rows)
```

SQL改写

- CTE用法

  ① 非简单子查询提取为CTE，包含聚合函数、窗口函数、集合运算。
  ② 将查询看成一棵树，树的主干是那些被参数筛选的表，通常将主干提取为CTE，其他子查询或者表需与主干关联，以减少结果集。

- 分页

  应用层面优化。

- 尽量减少使用SELECT *

- 窗口函数

  row_number = 1
  row_number <= 1

SQL改写

- or用法

  改写成业务含义等价的SQL

  ```
  postgres=# explain select * from test where id=1 or id=2;
  postgres=# explain select * from test where id in (1, 2);
  ```

- union用法

  结果集不需唯一，使用union all代替union

- in、exists、any用法

  推荐 any > exists > in

数据库配置

干涉执行计划

- 及时收集统计信息

  手动执行analyze

- 更准确的成本预估

  ① 调整成本因子
  ② 扩大统计信息采样范围，重新搜集统计信息，多列统计信息

- 调整函数易变性，使用预估更准的函数

  ① VOLATILE
  ② STABLE
  ③ IMMUTABLE

数据库配置

- GEQO

  当表数量>=geqo_threshold(默认12)，用遗传算法进行最优路径的筛选工作。

- 提升子查询关联等级和指定表连接顺序

  ① from_collapse_limit = 1
  ② join_collapse_limit = 1

- pg_hint_plan插件

数据库配置

- 执行计划节点开关

```
postgres=# select name, setting from pg_settings where name like 'enable_%';
              name               | setting
---------------------------------+---------
 enable_async_append             | on
 enable_bitmapscan               | on
 enable_gathermerge              | on
 enable_hashagg                  | on
 enable_hashjoin                 | on
 enable_incremental_sort         | on
 enable_indexonlyscan            | on
 enable_indexscan                | on
 enable_material                 | on
 enable_memoize                  | on
 enable_mergejoin                | on
 enable_nestloop                 | on
 enable_parallel_append          | on
 enable_parallel_hash            | on
 enable_partition_pruning        | on
 enable_partitionwise_aggregate  | off
 enable_partitionwise_join       | off
 enable_seqscan                  | on
 enable_sort                     | on
 enable_tidscan                  | on
(20 rows)
```

数据库配置

优化内存资源类参数

- maintenance_work_mem
- autovacuum_*等系列参数
- shared_buffers
- work_mem
- dynamic_shared_memory_type
- huge_page

架构设计

- 硬件升级
- 连接池
- 读写分离
- 缓存
- 分布式

架构设计

- 表定义

① 定期归档历史数据
② 使用分区表

什么时候使用分区表

I. 表数据量是否足够大
II. 表是否有合适的分区字段
III. 表内数据是否具有生命周期
IV. 查询语句中是否含有分区字段

相关工具

- pg_stat_activity：当前有哪些活动会话，每个会话正在执行什么SQL以及状态。

```
postgres=# select * from pg_stat_activity where datid is not null;
-[ RECORD 1 ]----+-----------------------------------------------------
datid            | 13892
datname          | postgres
pid              | 31504
leader_pid       |
usesysid         | 10
usename          | postgres
application_name | psql
client_addr      |
client_hostname  |
client_port      | -1
backend_start    | 2022-10-13 15:27:25.333653+08
xact_start       | 2022-10-13 17:52:20.861828+08
query_start      | 2022-10-13 17:52:20.861828+08
state_change     | 2022-10-13 17:52:20.861834+08
wait_event_type  |
wait_event       |
state            | active
backend_xid      |
backend_xmin     | 1854
query_id         |
query            | select * from pg_stat_activity where datid is not null;
backend_type     | client backend
```

相关工具

- pg_stat_statments：记录每个SQL执行次数，平均/最大/最小执行时间。

```
shared_preload_libraries='pg_stat_statements'

track_io_timing = on

track_activity_query_size = 2048

pg_stat_statements.max = 10000

pg_stat_statements.track = all

pg_stat_statements.track_utility = off

pg_stat_statements.save = on
```

欢迎关注美创科技新运维新数据公众号


openGauss
OceanBase
ArkDB
RASESQL
HotDB
StellarDB
QianBase xTP
云树Shard
GoldenDB
DolphinDB
MatrixDB
DynamoDB
SinoDB
FastData
Galaxybase
KunDB
SQL Server
vertica
D B 2
GBase
Oracle
达梦数据库
Spacture
开务数据库
UbiSQL
神舟通用
KingbaseES
MongoDB
TDSQL
TiDB
Tapdata
StarRocks
2010
2014
2018
THANKS