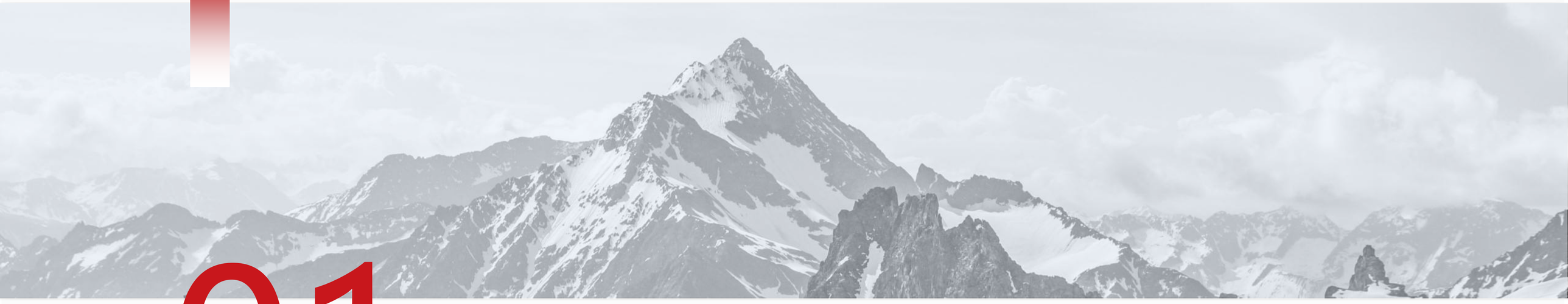
KINGBASE

迁移适配调优

目录

CONTENT



01 产品方案

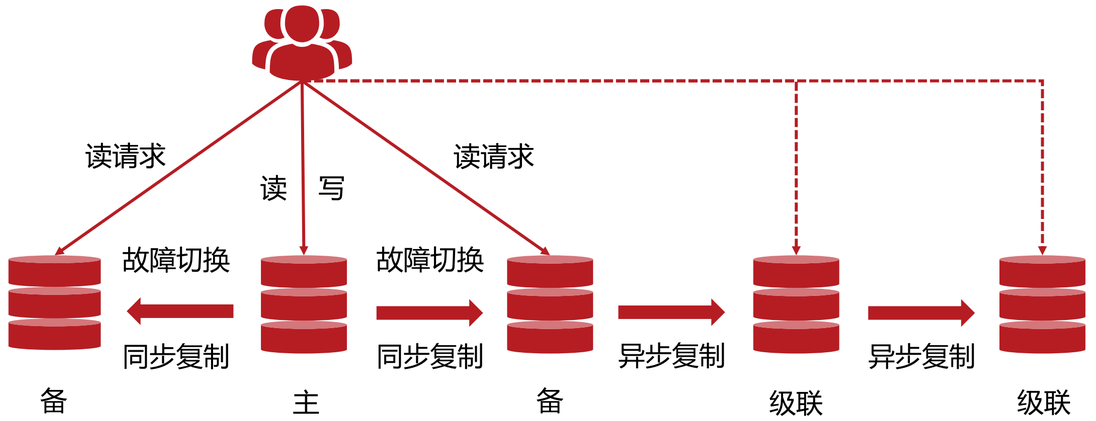
产品体系


KRDS
金仓云数据库服务管控平台
KStudio
金仓数据库开发工具
KMonitor
金仓数据库监控工具
KADB
金仓分析型数据库系统
KES
金仓数据库管理系统
KSOne
金仓HTAP分布式数据库
KDMS
金仓数据库迁移评估系统
KDTS
金仓数据库迁移工具
KFS
金仓异构数据同步软件
KINGBASE®
外部数据源

KES部署架构

集群级

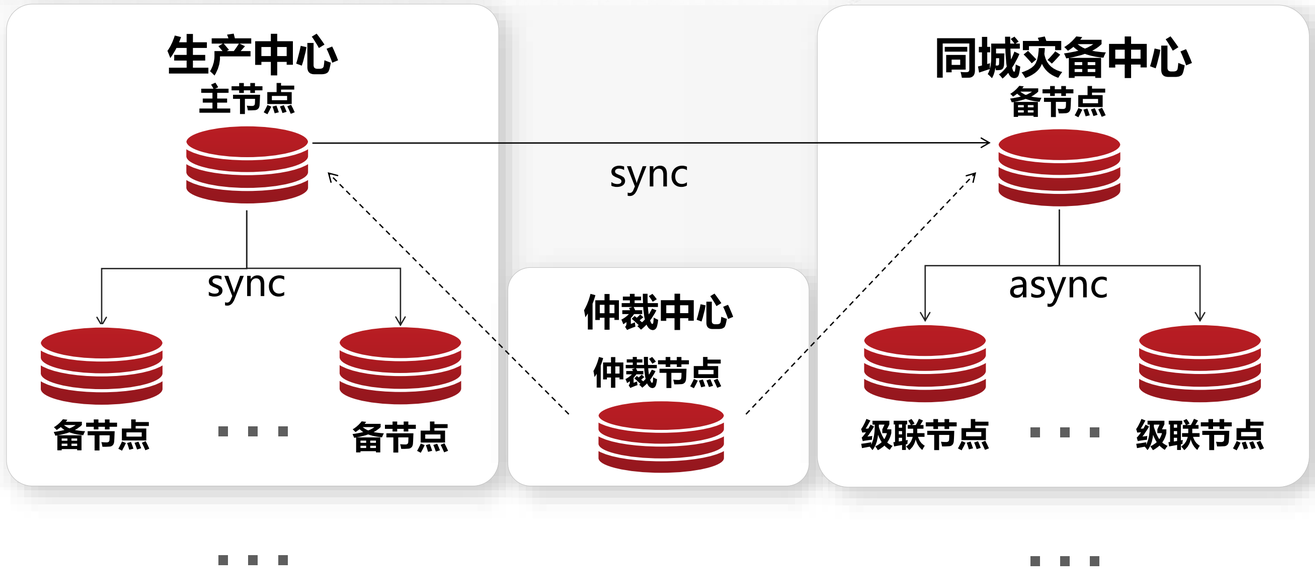
- 基于物理日志的全同步复制，数据不丢失
- 备节点支持异步复制，不阻塞事务，业务持续可用
- 备机日志回放并行加速，故障快速切换
- 增强型一致性协议选主，避免脑裂
- 节点故障恢复后自动回归，保持集群规模稳定

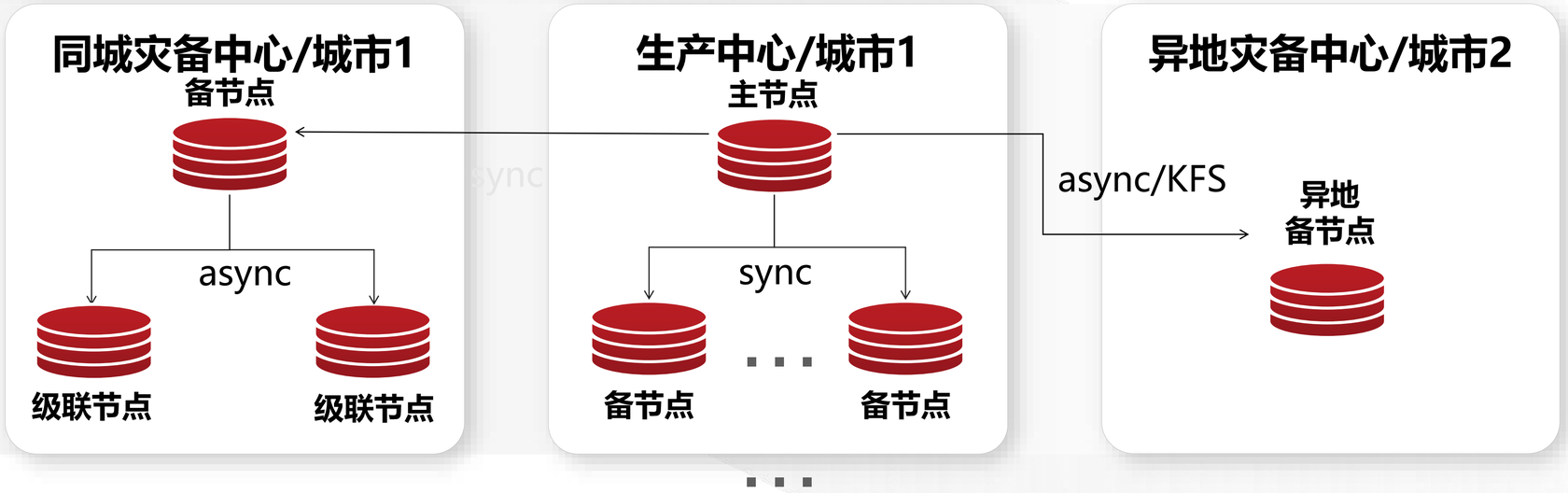


KES容灾架构

同城双中心容灾方案

- 同城跨数据中心物理日志全同步复制，数据“零” 丢失
- 主节点故障中心内优先选主，避免不必要的跨中心切换
- 支持跨中心快速故障切换，秒级极致RTO保障
- 故障节点/中心恢复后，自动回归集群，过程无需干预
- 异步备节点级联复制模式，有效降低带宽压力
- 同异步备机均支持只读业务访问，资源充分利用



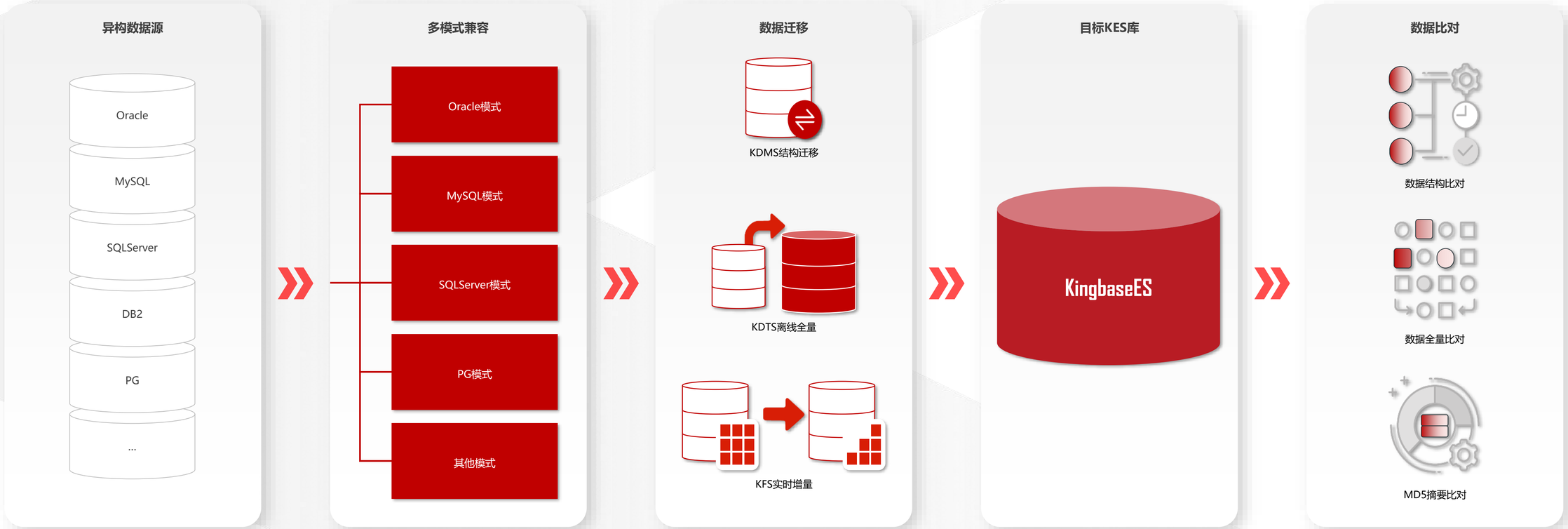


两地三中心容灾方案

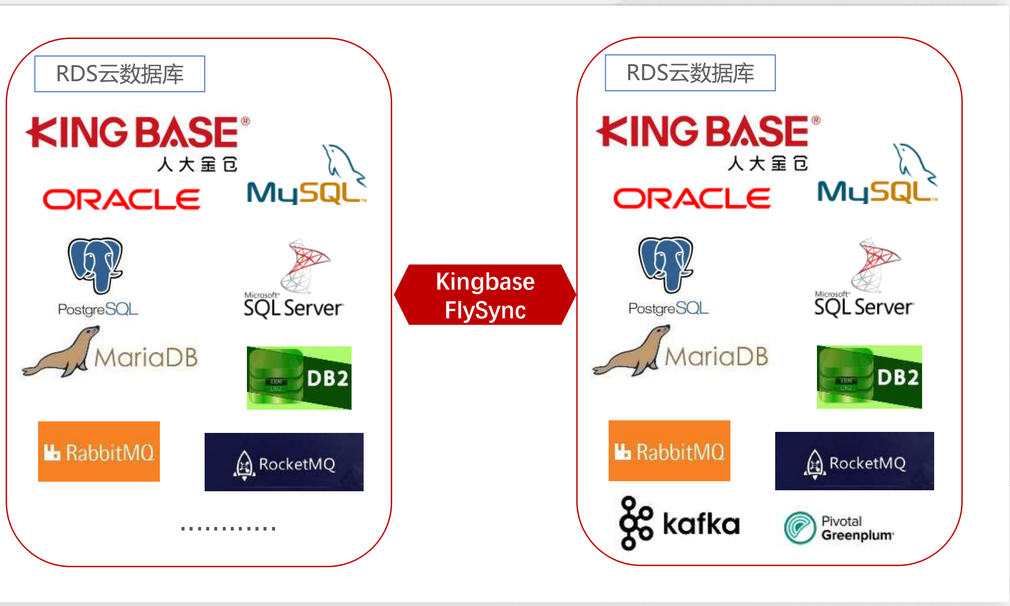
- 同城双中心物理全同步，确保数据“零” 丢失，故障“秒” 切换
- 跨城市物理日志高速复制，最优可实现秒级RPO
- 中心级多数派一致性切换，避免中心级双主

迁移工具

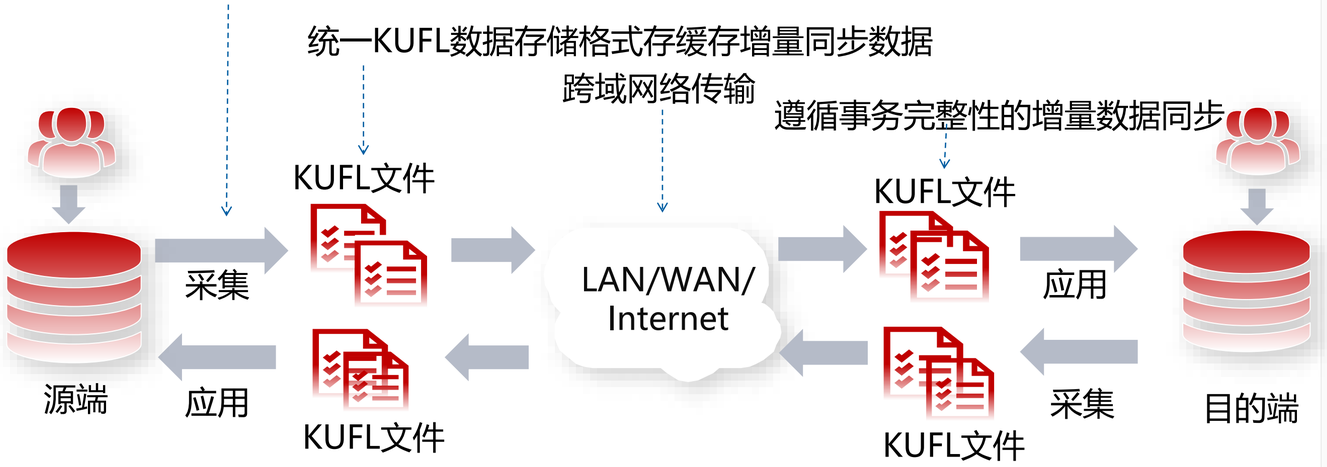
全面覆盖工程化迁移整体流程



KFS - 同步工具



基于物理日志解析的实时增量数据采集

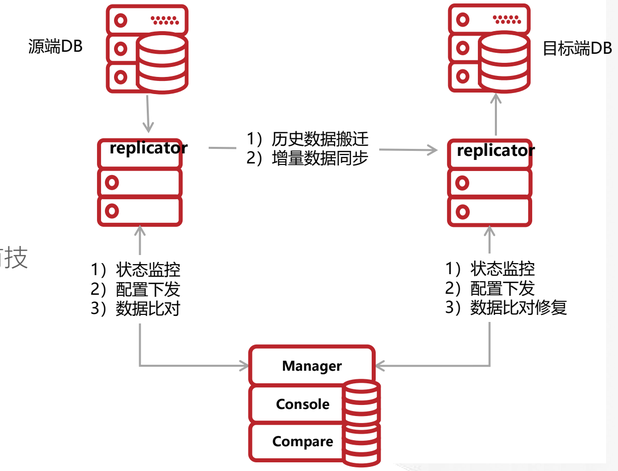


功能：

- 初始搬迁；
- 断点续传；
- 比对和修复；
- 监控及告警；
  - 故障重启；

- 选择同步-支持指定表、列过滤、DDL；
- 支持模式、表、列名称映射（独有技术-源端过滤）；
  - 支持增加指定列数据
  - 支持数据大小写转换

特殊场景：
文件SQL、延时同步



在线迁移场景

> Oracle源端解析：118MB/s

> KES源端解析：101MB/s

> 目标端加载：240MB+/s

> 数据同步延时：40ms

X86 2.8G 6C6T | 16GB | 512GB SSD | 千兆网络

源端解析 + 目标端同步 分段执行

实时灾备场景

- 基础100仓同步性能：
  - KFS：600GB+/天
  - OGG：460GB+/天
- 基础600仓同步性能：
  - KFS：570GB+/天
  - OGG：450GB+/天

X86 2.8G 6C6T | 16GB | 512GB SSD | 千兆网络

标准TPCC场景

迁移方案

柔性迁移方案提供业务不停服迁移能力

准在线

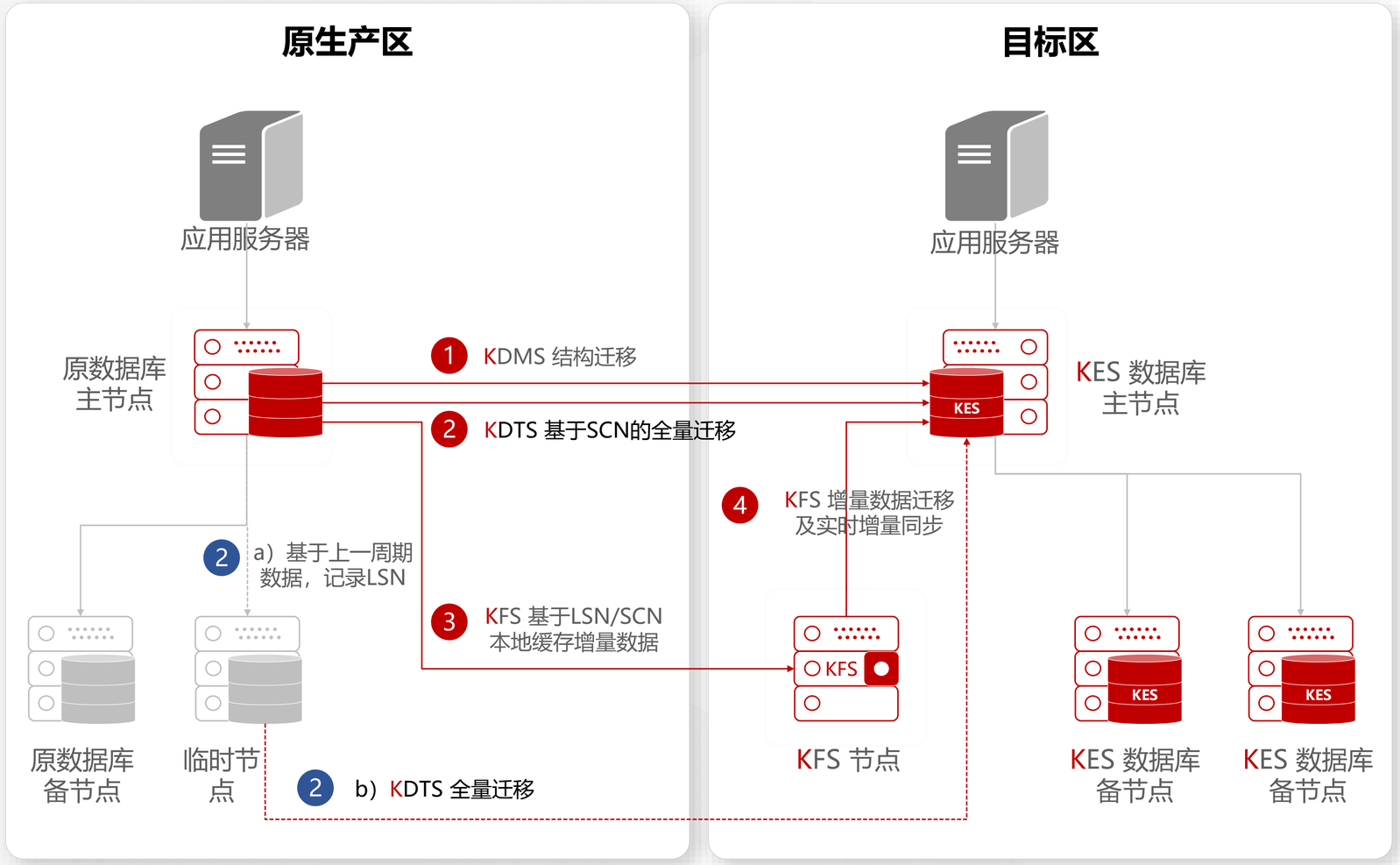
占用停服时间短，TB级存量数据小时级停服占用时间

低侵扰

在线数据迁移和应用适配过程对源端生产业务无侵扰，且不使用源端备库

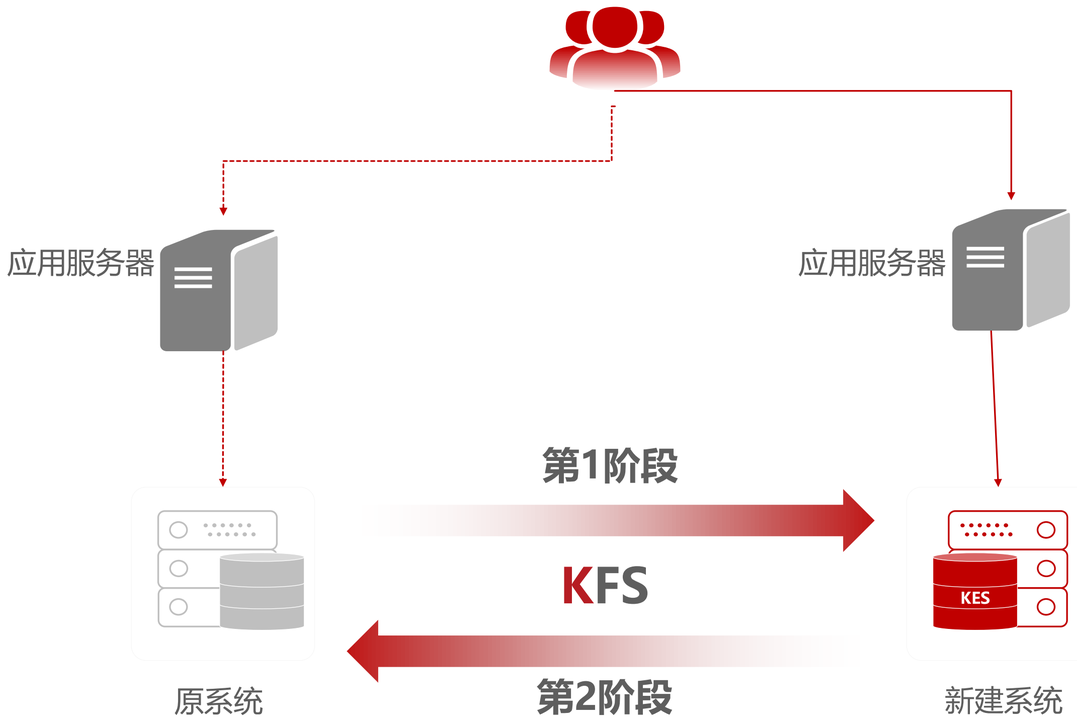
全数据

在线迁移过程中源端业务产生的数据变化（增、删、改）均可捕获，并提供本地缓存策略，可快速追平数据，有效保障数据一致性



双轨并行方案

提供完善回退机制



阶段1 正向同步部署

- 以原系统为业务主系统，KES为备系统
- 通过KFS实现原库与KES数据同步，两端数据一致。
- 此阶段，KES作为备库，可以分担查询业务。

阶段2 反向同步部署

- 不改变原有拓扑
- KES为主系统，原端为备系统
- 通过**KFS**实现KES与原库的反向同步，两端数据一致。
- 此阶段国产环境若发生故障，原始系统可迅速接管。

特性	典型案例
> **实时性** - 秒级延迟	> **企业** - 中国邮政集团 - 8节点RAC替换
> **异构环境支持** - 支持不同平台和数据库厂商及版本	> **运营商** - 中国移动营销资源、一级充值赋能
> **事务性保证** - 物理日志解析，可保证事务一致性	> **金融** - 人行征信融资平台

02 迁移适配

金仓相关资料

文档：https://help.kingbase.com.cn/v9/development/develop-transfer/index.html

安装与升级

基于Windows系统的数据库软件安装指南

基于Linux系统的数据库软件安装指南

KingbaseES数据库docker部署手册

KingbaseES数据库部署工具使用指南

KingbaseES License信息手册

应用开发及迁移

应用开发及迁移指南

客户端编程接口

客户端编程开发框架

常用连接池

SQL和PL/SQL

安全

KingbaseES安全指南

性能

KingbaseES数据库性能调优指南

KingbaseES数据库SQL调优指南

可用性

高可用

备份与恢复

培训网址：https://bbs.kingbase.com.cn/

KCA培训视频合集

共27课 15时58分

免费 213125人学习

KCP培训视频合集

共27课 15时53分

免费 101395人学习

免费 KCA-01-走进kingbase ES

免费 KCA-02-KES安装和启停

免费 KCA-03- 数据库对象管理工具

免费 KCA-04-命令行工具-KSQL

免费 KCA-05-用户与角色

免费 KCA-06-对象访问权限入门

免费 KCA-07-库、模式、表空间

免费 KCA-08-简单运维和巡检

免费 KCA-09-单表查询

免费 KCA-10-多表查询

免费 KCA-11-表的定义

免费 KCA-12-表的约束

免费 KCA-13-外部表

免费 KCA-14-索引的定义

免费 KCA-15-视图管理

免费 KCA-16-物化视图

免费 KCA-17-序列

免费 KCA-18-数据操纵

免费 KCA-19-分区表

免费 KCA-20-集合运算、子查询、伪列

免费 KCA-21-内置单行函数

免费 KCA-22-内置多行函数

免费 KCA-23 -TOAST行外存储

免费 KCA-24-KDMS 数据库迁移评估系统

免费 KCA-25- 数据库迁移工具

免费 KCA-26-PLSQL 简介

免费 KCA-27-KES 的兼容特性

NG BASE®

免费 KCP-08-体系结构（上）

免费 KCP-09-体系结构（下）

免费 KCP-10-服务器配置

免费 KCP-11-客户端认证

免费 KCP-12-逻辑备份和还原

免费 KCP-13-物理备份和还原（上）

免费 KCP-14-物理备份和还原（下）

免费 KCP-15-事务日志与检查点（上）

免费 KCP-16-事务日志与检查点（下）

免费 KCP-17-归档日志

免费 KCP-18-事务基础知识

免费 KCP-19-并发控制

免费 KCP-20-SQL语句的执行计划

免费 KCP-21-索引的应用

免费 KCP-22-统计信息与常用数据字典

免费 KCP-23-对象的访问权限进阶

免费 KCP-24-日常运维

免费 KCP-25-TPCC 性能测试

免费 KCP-26-部署集群V8R6集群（上）

免费 KCP-27-部署集群V8R7集群（下）

项目管理流程

阶段	环节	内容	分工
方案	方案设计	1、架构设计（部署架构（数据库、应用系统）、跨库访问、数据交换） 2、迁移方案-双轨运行 3、备份策略 4、灾备方案 5、数据库设计方案（名称设计、大小写、注释、别名、SQL规范、）	
适配测试	环境准备	数据库部署、业务部署 （硬件、网络、软件）	金仓：数据库软件、部署数据库 ISV: 应用软件、部署应用 客户: 硬件、网络
	功能适配（迁移适配）	数据迁移、驱动、架构配置、功能验证、语法改写	金仓: 数据迁移，配合业务适配 ISV: 功能验证
	性能压测	筛选场景、压测工具、提炼SQL、压测、资源使用收集、慢SQL收集、日志收集、优化	金仓: 配合优化， ISV: 功能模块性能压测
上线	上线准备阶段	生产部署；存量迁移、增量同步；高可用演练、备份恢复演练；	金仓: 数据库部署以及演练 ISV: 上线方案。 客户：审核方案
	上线割接	数据校验、业务测试、业务割接、反向同步	金仓: 上线校验 ISV: 校验内容、业务测试、业务割接 客户: 明确割接时间
运维	运维	巡检策略制定、巡检、故障处理	金仓: 巡检、故障诊断及处理 ISV: 巡检、故障诊断及处理 客户：制定审核巡检策略

金仓安装部署

软件、license下载：
https://www.kingbase.com.cn/xzzx/index.htm

内容	资料
数据库部署、业务部署（硬件、网络、软件）	软件(平台、版本、模式)、授权
数据迁移、驱动、架构配置、功能验证、语法改写	驱动位置（安装目录下的Interface目录）、驱动文档；架构配置文档；迁移文档（客户端）
筛选场景、压测工具、提炼SQL、压测、资源使用收集、慢SQL收集、日志收集、优化	调优文档； 管理工具使用文档； 常规操作指令；
生产部署；存量迁移、增量同步；高可用演练、备份恢复演练；	
数据校验、业务测试、业务割接、反向同步	
巡检策略制定、巡检、故障处理	

操作系统需求

存储规划和目录划分 软件、数据和备份划分。 建议数据目录和备份目录分开
设置且备份和归档放在一个目录下
操作系统字符集 zh.cn-UTF8
服务器防火墙是否可以关闭
需要挂载外存储的，几个服务器kingbase用户的userid一样
集群数据库 使用端口 R6版本:54321
集群用户root用户可ssh免密配置
集群的时间同步且已调整需要时区
确认一下云平台 mac会不会变化

网络要求

集群需要有一个可ping网关或者其他可靠IP地址
集群需要安装软件之前提前做好bond

数据库规划

数据库system密码
字符集编码 UTF8或者GBK
数据库大小写是否敏感，
输入空的处理方法
数据库日志保留时间
备份策略：物理、逻辑
R6多节点集群不需要VIP。R6两节点集群需要一个IP地址做vip
等保需求 审计 三权分立
同步机制
客户端部署

应用需求

应用需要的驱动类型
是否需要创建相应数据
是否需要数据迁移 数据迁移的源数据库类型
是否有数据需要导入（excel等类型）

客户端软件

资料
软件(平台、版本、模式)、授权
驱动位置（安装目录下的Interface目录）、驱动文档；架构配置文档；迁移文档（客户端）
调优文档； 管理工具使用文档； 常规操作指令；

1、工具简介
2、参考文档
3、培训文档

KingbaseES异构数据库移植指南

介绍KingbaseES产品的应用迁移支撑体系，以及应用迁移的过程。

Oracle至KingbaseES迁移最佳实践

指导用户在不同的场景（不同数据库软件版本之间的数据迁移/不同软硬件平台之间的数据迁移）下，进行数据迁移。

KingbaseES与Oracle的兼容性说明

介绍KingbaseES和oracle的兼容性情况 。

MySQL至KingbaseES迁移最佳实践

MySQL至KingbaseES迁移最佳实践

KingbaseES与MySQL的兼容性说明

介绍KingbaseES和MySQL的兼容性情况 。

KingbaseES V8R3至V9迁移最佳实践

指导用户将KingbaseES V8R3 数据库升级为KingbaseES V9 。

KDTS 迁移工具使用指南

介绍了KingbaseES数据库客户端管理工具"数据库迁移工具"BS版、SHELL版的使用说明、部署方式及配置信息。

OCCI迁移指南

KingbaseES 兼容 Oracle 的 OCCI(Oracle C++ Call Interface) 的使用说明文档。

Pro*C迁移指南

KingbaseES 兼容 Oracle 的嵌入式SQL语言（以下简称Pro*C）的使用说明文档。

位置：
Windows：开始菜单KingbaseES V8文件夹
Linux：数据库安装用户开始菜单KingbaseES V8文件夹
KSQL 工具

适配

资料
软件(平台、版本、模式)、授权
驱动位置（安装目录下的Interface目录）、驱动文档；架构配置文档；迁移文档（客户端）
调优文档； 管理工具使用文档； 常规操作指令；

1、端口
2、URL串
3、url参数、数据库参数

4、文档-
https://help.kingbase.com.cn/v9/download.html
5、语法转换-日志
6、扩展
7、调试工具

KingbaseES用户手册	V009R001C001B0024	2023-10-10	PDF	下载PDF
KingbaseES用户手册合集	V009R001C001B0024	2023-10-10	CHM	下载CHM

客户端编程接口

- KingbaseES客户端编程接口指南-ODBC
- KingbaseES客户端编程接口指南-JDBC
- KingbaseES客户端编程接口指南-DCI
- KingbaseES客户端编程接口指南-Perl DBI
- KingbaseES客户端编程接口指南-Gokb
- KingbaseES客户端编程接口指南-PHP PDO
- KingbaseES客户端编程接口指南-Nodejs
- KingbaseES客户端编程接口指南-Python
- KingbaseES客户端编程接口指南-ado.net

KingbaseES常用连接池配置指南

1. 前言
   - 1.1. 适用读者
   - 1.2. 相关文档
   - 1.3. 术语
   - 1.4. 手册约定
2. 概述
3. 常用连接池配置说明
   - 3.1. DBCP配置
   - 3.2. C3P0配置
   - 3.3. Druid配置说明

客户端编程开发框架

- KingbaseES客户端编程开发框架-Activiti
- KingbaseES客户端编程开发框架-Hibernate
- KingbaseES客户端编程开发框架-Hibernate Spatial
- KingbaseES客户端编程开发框架-MyBatis
- KingbaseES客户端编程开发框架-MyBatis-Plus
- KingbaseES客户端编程开发框架-Liquibase
- KingbaseES客户端编程开发框架-Flyway
- KingbaseES客户端编程开发框架-EF6
- KingbaseES客户端编程开发框架-Efcore
- KingbaseES客户端编程开发框架-Qt
- KingbaseES客户端编程开发框架-SQLAlchemy
- KingbaseES客户端编程开发框架-Django
- KingbaseES客户端编程开发框架-DbUnit

KES - 开发

开发语言及架构

语言	JAVA		C/C++	ADO.NET	Perl	PHP	PYTHON	GO	Nodejs
接口	JDBC Hibernate Hibernate spatial Mybatis Mybatis-Plus	Acitviti Flyway Liquibase Dbunit DataX DBCP、C3P0、Druid	ODBC OCI、DCI Pro*C OCCI ESQL QT	OLEDB NDP NETCORE EFCORE EF6	DBI	PDO	Ksycopg2 SQLALCHEME Django	GOKB	nodejskb

数据类型

名称	数量
数字类型	11
货币类型	1
字符类型	8
大对象数据类型	4
日期/时间类型	6+20
布尔类型	1
位串类型	2
几何类型	7
网络类型	5
文本搜索	2
UUID类型	1
XML类型	1
JSON 类型	2
范围类型	6
对象标识符类型	1
枚举、组合、域、对象	
自定义数据类型	

函数及运算符

名称	函数/运算符		名称	函数/运算符	
数学函数和操作符	36	17	数组函数和操作符	17	10
字符串函数和操作符	26	1	范围函数和操作符	6	13
二进制串函数和操作符	19		聚集函数	28	
位串函数和操作符	8	7	窗口函数	12	
模式匹配	4	2	枚举函数	3	
数据类型格式化函数	5		集合返回函数	2	
时间/日期函数和操作符	61	5	系统信息函数	54	3
几何函数和操作符	36	31	系统管理函数	23	
网络地址函数和操作符	6	11	触发器函数	24	
文本搜索函数和操作符	37	9	自动作业功能	19	
XML 函数	27	2	RAW类型转换函数	3	
JSON 函数和操作符	45	36	规则相关函数	7	
序列操作函数	4		空间函数	600+	

SQL典型用法

名称	SQL
q’ 转义	
多行插入	INSERT ALL、INSERT FIRST
判断逻辑SQL	merge into、on conflict do update、insert on duplicate key
行列转换	Pivot/unpivot、case
数据采样	tablesample bernoulli
CASE表达式	case when … then
伪表(dual)、伪列(dummy、rownum、rowid)	
分页查询	OFFSE...LIMIT、 OFFSET...FETCH、LIMIT...OFFSET
层次查询	connect by
分组查询	ROLLUP、CUBE、GROUPING SETS
集合操作	UNION、UNION AL、MINUS、INTERSECT
多表关联	inner join、left join、right join、outer join
with 用法	递归+函数
SQL手动加锁	For UPDATE skip locked、SELECT FOR UPDATE wait

KES-扩展

扩展名称	功能	详细说明
uuid-ossp	数据生成器	uuid生成工具
DBMS_RANDOM		随机数或者字符的生成器
hstore	数据类型	支持hstore类型
seg		为程序提供seg数据类型的使用，带有边界的interval
cube		表示多维立方体，平面中的几何类型一样
ltree		包含ltree和lquery类型的实现
isn		为国际产品编号标准提供数据类型
kdb_oracle_datatype		初始化内置，兼容Oracle数据类型
kdb_oracle_datatype_nls		初始化内置，兼容Oracle日期date数据类型
kdb_date_function		mysql数据库date相关函数的扩展
kdb_raw		raw数据类型的操作
kdb_utils_function	函数	若干字符串函数，若干xml函数、行列转换函数
fuzzystrmatch		提供了两个字符串之间的编辑距离的相关函数
Xml2		提供XPath查询和XSLT功能函数
Intarray		为操作整数的null-free数组提供一些有用的函数和操作符
tsm_system_rows		提供了表采样方法SYSTEM_ROWS
tsm_system_time		提供了表采样方法SYSTEM_TIME
Ftutilx		blob类型字段中抽取文本内容
Ftutils		流版式文件内容抽取
utl_file		补充了一些对于文件操作的函数
Tablefunc		返回多行集合的函数，表(即多行)的各种函数
dbms_lob		CLOB/BLOB大对象的操作
Lo		lo方式存储的大对象数据是独立的对象，有独立的存储。
Earthdistance		支持了与地球表面相关的数据操作
DBMS_OUTPUT		文本行的提取和显示
DBMS_SQL		动态SQL语句的操作
http		数据库内检索网页
sys_trgm		基于 trigram 匹配的函数和操作符，用于在全文搜索中

扩展名称	功能	详细说明
cstore_fdw	列存	列存扩展
pg_partman	分区	分区管理
sys_bulkload	数据加载	文本数据简单、快速的加载
file_fdw	外部表	用来访问存储在外部文件中的数据
dblink	跨库访问	接到其他Kingbase数据库
kdb_database_link		用于访问外部数据库对象
kingbase_fdw		外部数据库
mysql_fdw		用来访问存储在外部Mysql数据库中的数据
oracle_fdw		用来访问存储在外部Oracle数据库中的数据
tds_fdw		使用表格数据流（TDS）协议的数据库
sys_jieba	全文搜索	全文搜索的中文分词
zhparser		全文搜索的中文分词
unaccent		提供了一个能从词位中移除重音的文本搜索字典
dict_int		支持了对于数字的text类型搜索字典
dict_xsyn		提供一个具有附加全文搜索功能的字典模板
kdb_schedule	定时任务	DBMS_JOB和DBMS_SCHEDULER包

03 数据库调优

调优划分

- 01 硬件调优——cpu+网络+内存+io（量）
- 02 操作系统调优——参数+资源+文件系统策略
- 03 数据库参数优化——内存大小、并行
- 04 SQL优化-今日内容

SQL优化

- SQL执行：统计信息、查询优化器、
- 找出慢sql、收集统计信息（ANALYZE）、查看执行计划-找耗时节点
- 设计合理的表结构（分区表）及sql语句结构
- 创建合适的索引-读取、排序（MIN、MAX等）、部分计算
- 手动调整执行计划-读取、计算（过滤+级联+排序+去重+分组）
- 优化并行-多cpu加快计算

抓取慢sql

通过查看过去的执行历史记录，找到消耗了较多系统资源的SQL语句。KingbaseES提供的工具包括：

- sys_stat_statement
- SYS_KWR
- Kbbadger

慢sql记录时间1S以上

#log_min_duration_statement = -1

录慢sql，设置为1000时，则超过1s的慢sql都会打印，不受log_state_statement影响记

常用方法：日志打印

[root@kingbase bin]# ./kbbadger /data/kingbase/R6/data/sys_log/kingbase-08.log

[==========================>] Parsed 16597 bytes of 16597 (100.00%), queries: 32, events: 15

LOG: Ok, generating html report...

抓取运行慢sql

来控制 auto_explain 的行为：

auto_explain.log_min_duration（整数）

auto_explain.log_min_duration 是最短语句执行时间（以毫秒为单位），将此设置为 0 将记录所有计划。-1（默认）禁用计划记录。

auto_explain.log_analyze（boolean）

当记录执行计划时，auto_explain.log_analyze 控制打印 EXPLAIN ANALYZE 输出，而不仅仅是 EXPLAIN 输出。默认情况下，此参数是关闭的。仅超级用户可以更改此设置。

关闭 auto_explain.log_timing 可以降低性能成本，但只能获取较少的信息。

auto_explain.log_buffers（boolean）

auto_explain.log_buffers 控制在记录执行计划时是否打印缓冲区使用情况统计信息。它等效于 EXPLAIN 的 BUFFERS 选项。除非启用了 auto_explain.log_analyze，否则此参数无效。默认情况下，此参数是关闭的。仅超级用户可以更改此设置。

auto_explain.log_timing（boolean）

auto_explain.log_timing 控制在记录执行计划时是否打印每个节点的计时信息。它等效于 EXPLAIN 的 TIMING 选项。重复读取系统时钟的开销可能会严重降低某些系统上的查询速度，因此在仅需要实际行计数而不是确切时间的情况下将此参数设置为 off。除非启用了 auto_explain.log_analyze，否则此参数无效。

auto_explain.log_verbose（boolean）

auto_explain.log_verbose 控制着记录执行计划时是否打印详细信息。它等效于EXPLAIN 的 VERBOSE 选项。默认情况下，此参数是关闭的。仅超级用户可以更改此设置。

auto_explain.sample_rate（boolean）

auto_explain.sample_rate 导致 auto_explain 仅解释每个会话中的一部分语句。默认值为 1，表示解释所有查询。如果是嵌套语句，将全部解释或不解释。仅超级用户可以更改此设置

解读执行计划

```
postgres=# explain analyze select * from a left join b on a.aid = b.bid where a.aid < 10000 order by a.aid;
                                                                 QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=54176.41..54201.72 rows=10125 width=12) (actual time=350.440..350.770 rows=9999 loops=1)
   Sort Key: a.aid
   Sort Method: quicksort  Memory: 853kB
   ->  Hash Right Join  (cost=17051.56..53502.81 rows=10125 width=12) (actual time=67.007..348.612 rows=9999 loops=1)
         Hash Cond: (b.bid = a.aid)
         ->  Seq Scan on b  (cost=0.00..28850.00 rows=2000000 width=4) (actual time=0.009..129.963 rows=2000000 loops=1)
         ->  Hash  (cost=16925.00..16925.00 rows=10125 width=4) (actual time=66.960..66.960 rows=9999 loops=1)
               Buckets: 16384  Batches: 1  Memory Usage: 480kB
               ->  Seq Scan on a  (cost=0.00..16925.00 rows=10125 width=4) (actual time=0.023..63.568 rows=9999 loops=1)
                     Filter: (aid < 10000)
                     Rows Removed by Filter: 990001
 Planning Time: 0.523 ms
 Execution Time: 351.331 ms
(13 rows)
```

执行计划解读

1、由内向外

2、由上到下

3、cost部分显示成本相关的信息，cost=后面有两个数字，中间由..分隔，第一个数字0.00表示启动的成本，也就是说，返回第一行需要多少cost值；第二个数字表示返回所有数据的成本。actual部分显示实际执行相关的信息，

示例中的执行计划解读为：全表扫描表a，过滤出aid小于10000的数据，通过hash连接与全表扫描b的结果连接，最终根据a表的aid进行排序，过程中消耗比较大的步骤是全表扫描b表以及两表关联数据时。

```
postgres=# explain (analyze,verbose,costs,timing,buffers) select * from a left join b on a.aid = b.bid where a.aid < 10000 order by a.aid;
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=54176.41..54201.72 rows=10125 width=12) (actual time=359.615..360.042 rows=9999 loops=1)
   Output: a.aid, b.bid, a.aid
   Sort Key: a.aid
   Sort Method: quicksort  Memory: 853kB
   Buffers: shared hit=13275
   ->  Hash Right Join  (cost=17051.56..53502.81 rows=10125 width=12) (actual time=67.230..358.520 rows=9999 loops=1)
         Output: a.aid, b.bid, a.aid
         Hash Cond: (b.bid = a.aid)
         Buffers: shared hit=13275
         ->  Seq Scan on public.b  (cost=0.00..28850.00 rows=2000000 width=4) (actual time=0.008..133.388 rows=2000000 loops=1)
               Output: b.bid
               Buffers: shared hit=8850
         ->  Hash  (cost=16925.00..16925.00 rows=10125 width=4) (actual time=67.189..67.189 rows=9999 loops=1)
               Output: a.aid
               Buckets: 16384  Batches: 1  Memory Usage: 480kB
               Buffers: shared hit=4425
               ->  Seq Scan on public.a  (cost=0.00..16925.00 rows=10125 width=4) (actual time=0.025..63.982 rows=9999 loops=1)
                     Output: a.aid
                     Filter: (a.aid < 10000)
                     Rows Removed by Filter: 990001
                     Buffers: shared hit=4425
 Planning Time: 0.196 ms
 Execution Time: 360.698 ms
(23 rows)
```

analyze：执行命令并显示执行事件，默认false。没有这个参数 就只能看执行计划生成 看不出来执行时间。

verbose：显示附加信息，比如计划树中每个节点输出的字段名等，默认false 。

costs：显示执行计划的成本，默认true 。

buffers：显示缓冲区的使用信息，包括共享快、本地块和临时读写块，默认false，前置条件是analyze 。“Buffers:shared hit=1 read=xxx written=yyy”，其中“shared hit=1”表示在共享内存中直接读到1个块，从磁盘中读到xxx块，写磁盘yyy块。

timing: 计时，默认true。

查询执行

在kingbase中，节点分为四类，分别是控制节点(Control Node)、扫描节点(ScanNode)、物化节点(Materialization Node)、连接节点(Join Node)。

扫描节点：顾名思义，此类节点用于扫描表、函数结果集、子查询结果集、链表等对象以从中获取元组。

连接节点：此类节点对应于关系代数中的连接操作，可以实现多种连接方式（条件连接、左连接、右连接、全连接、自然连接等），每种节点实现一种连接算法。例如，HashJoin实现了基于Hash的连接算法。

物化节点：这类节点种类比较复杂，但它们有一个共同特点，即能够缓存执行结果到辅助存储中。物化节点会在第一次被执行时生成其中的所有结果元组，然后将这些结果元组缓存起来，等待其上层节点取用；而非物化节点则是每次被执行时生成一个结果元组并返回给上层节点。例如，Sort节点能够获取下层节点返回的所有元组并根据指定的属性进行排序，并将排序结果全部缓存起来，每次上层节点从St节点取元组时就从缓存中按顺序返回下一个元组。

控制节点：是一类用于处理特殊情况的节点，用于实现特殊的执行流程。例如， append，组织多个字表或子查询的执行节点，主要用于union操作

扫描节点

顺序扫描、索引扫描、索引覆盖扫描、bitmap扫描、TID扫描。

Seqscan 表的顺序扫描就是顺序扫描对应表所有页的item指针。全表扫描 数据量多，时间长。

Indexscan 索引扫描，依赖于不同索引类型并和查询中涉及的索引相对应使用不同的数据结构。然后索引扫描获取的条目直接指向heap域中的数据，然后根据隔离级别判断可见性。索引解决的取值和排序的问题，增加随机读。

Index Only Scan 仅索引扫描

bitmap 扫描是Bitmap Index Scan和Bitmap Heap Scan的组合。增加cpu开销和时间。可以进行and或者or的运算。

TID scan ctid 扫描 ctid 是 Kingbase 中标记数据位置的字段，通过这个字段来查找 数据，速度非常快，类似于 Oracle 的 rowid。

扫描节点

•Sample Scan：

数据取样功能，支持查询返回取样数据。当前只在常规表和物化视图上接受TABLESAMPLE子句。

•Subquery Scan：

以另一个查询计划树（子计划）为扫描对象进行元组的扫描，其扫描过程最终被转换为子计划的执行。kingbase子查询主要包含如下几个关键字: EXISTS, IN, NOT IN, ANY/SOME, ALL。

•Function Scan

在KingbaseES中，有一些函数可以返回元组的集合，为了能从这些函数的返回值中获取元组，KingbaseES定义了 FunctionScan节点，其扫描对象为返回元组集的函数。FunctionScan节点在Scan的基础上扩展定义了functions列表字段，里面存放了FuncitonScan涉及的函数，以及funcordinality字段(是否给返回结果加上序号列)。

•Values Scan：

VALUES计算由值表达式指定的一个行值或者一组行值。更常见的是把它用来生成一个大型命令内的“常量表”， 但是它也可以被独自使用。

•CTE Scan：

WITH提供了一种方式来书写在一个大型查询中使用的辅助语句。这些语句通常被称为公共表表达式或CTE，它们可以被看成是定义只在一个查询中存在的临时表。在WITH子句中的每一个辅助语句可以是一个SELECT、INSERT、UPDATE或DELETE，并且WITH子句本身也可以被附加到一个主语句，主语句也可以是SELECT、INSERT、UPDATE或DELETE。

•WorkTable Scan：

WorkTableScan会与RecursiveUnion共同完成递归合并子查询。

•Foreign Scan：

扫描Kingbase外部数据表，用于kingbase_fdw或者dblink和外部数据交互的情况。

•Custom Scan：

自定义扫描接口，用户可以进行自定义扫描方法。

连接节点

连接节点主要是根据算法估算连接代价确定连接顺序。

nestloopjoin
原理：扫描每一条外表的数据（总共m条数据），然后和内表所有的记录（总共n条数据）去连接，时间复杂度是o(mn)
适用情况：数据量不大的情况

Hash join
对内表建立hash表，扫描所有内表数据到各个hash桶里面，建立hash桶，然后一行行扫描外表数据，对外表数据进行hash，hash到某个桶里面，然后跟这个桶里面的数据进行连接。
适用情况：数据分布比较随机无序，重复值不是特别多的情况。

Merge join
对两个表的数据进行排序，然后做连接。
适用情况：两个表的数据都是基本有序

物化节点

结果到缓存中，即第一次被执行时生成的结果元组缓存，等待上层节点使用。包括（Material、Sort、Group、Agg、Unique、Hash、SetOp、Limit、WindowAgg）

Materialize：对下层节点返回的元组进行缓存（如子查询）

```
explain select * from test_dm where id > any (select id from test_new);
QUERY PLAN------------------------------------------------------------------------
Nested Loop Semi Join  (cost=0.00..32201696.41 rows=333333 width=68)
        Join Filter: (test_dm.id > test_new.id)
                ->  Seq Scan on test_dm  (cost=0.00..22346.00 rows=1000000 width=68)
                ->  Materialize  (cost=0.00..58.25 rows=2550 width=4)
                        ->  Seq Scan on test_new  (cost=0.00..35.50 rows=2550 width=4)
```

Sort：对下层返回的节点进行排序（如果内存超过work_mem参数指定大小，则节点工作空间切换到临时文件，性能急剧下降），明确对数据进行排序。Order by 在开始阶段可以用。用索引排序替换。

```
explain select  id,xxx  from test order by id ;

---------------------------------------------------------------------
 Sort  (cost=2904727.34..2929714.96 rows=9995048 width=34)
   Sort Key: id
   ->  Seq Scan on test  (cost=0.00..376141.48 rows=9995048 width=34)
```

物化节点

Group：对下层排序元组进行分组操作。

一般出现在group by，分组主要在于计算 并行方式优化会好一些。

Agg：执行聚集函数（sum/max/min/avg）的group by 操作

```
explain select max(name) from test_dm;
                              QUERY PLAN
-----------------------------------------------------------------------
 Aggregate  (cost=24846.00..24846.01 rows=1 width=45)
   ->  Seq Scan on test_dm  (cost=0.00..22346.00 rows=1000000 width=45)
```

<--- Plain策略分组节点（Group），Aggregate 原理：不分组，直接进行聚集。

```
explain select max(name) from test_dm  group by id order by id;
                                 QUERY PLAN
-----------------------------------------------------------------------------
 GroupAggregate  (cost=122003.84..139503.84 rows=1000000 width=49)
   Group Key: id
   ->  Sort  (cost=122003.84..124503.84 rows=1000000 width=49)
         Sort Key: id
         ->  Seq Scan on test_dm  (cost=0.00..22346.00 rows=1000000 width=49)
```

<--- Sorted策略，GroupAggregate 原理：将排序的数据进行分组，然后执行聚集 。

```
explain select max(name) from test_dm group by id;
                              QUERY PLAN
-----------------------------------------------------------------------
 HashAggregate  (cost=27346.00..37346.00 rows=1000000 width=23)
   Group Key: id
   ->  Seq Scan on test_dm  (cost=0.00..22346.00 rows=1000000 width=23)
```

<--- Hash策略， HashAggregate 原理：通过hash算法，把相同的值hash到同一桶里面，然后求聚集。

物化节点

Unique：执行去重操作

与SQL SELECT子句中的关键字DISTINCT对应，同时SQL的操作符UNION、INTERSECT、EXCEPT默认去重。原理：Unique节点用于对下层节点返回的已排序元组进行去重操作。由于下层节点获取到的元组已经排序，因此在Unique节点的执行过程中只需要缓存上一个返回的元组，判断当前获f的元组是否和上一个元组在指定厲性上重复。如果重复，则忽略当前元组并继续从下层节点获取元组；如果不重复，则输出当前元组并用它替换缓存中的元组。适用情况：Unique节点一般用于处理查询中的DISTINCT关键字，但这不是唯一的处理方式。如果要求去重的属性被“ORDER BY”子句引用时，一般会使用Unique节点进行处理。

```
explain select distinct(id) from test_dm order by id;
------------------------------------------------------------------------------
 Unique  (cost=122003.84..127003.84 rows=1000000 width=4)
   ->  Sort  (cost=122003.84..124503.84 rows=1000000 width=4)
         Sort Key: id
         ->  Seq Scan on test_dm  (cost=0.00..22346.00 rows=1000000 width=4)

explain select distinct(id) from test_dm;
------------------------------------------------------------------------
 HashAggregate  (cost=24846.00..34846.00 rows=1000000 width=4)
   Group Key: id
   ->  Seq Scan on test_dm  (cost=0.00..22346.00 rows=1000000 width=4)
```

并行、索引

物化节点

limit

```
TEST=# explain analyze select max(id) from t_sel;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Result  (cost=0.46..0.47 rows=1 width=4) (actual time=0.045..0.047 rows=1 loops=1)
   InitPlan 1 (returns $0)
     ->  Limit  (cost=0.43..0.46 rows=1 width=4) (actual time=0.036..0.038 rows=1 loops=1)
           ->  Index Only Scan Backward using ddd on t_sel  (cost=0.43..177744.43 rows=6240000 width=4) (actual time=0.035..0.035
rows=1 loops=1)
                 Index Cond: (id IS NOT NULL)
                 Heap Fetches: 0
 Planning Time: 0.251 ms
 Execution Time: 0.084 ms
```

分页常用方法：select * from aa limit 1000,20；
以上sql弊端：表越大后续越慢；
方法找一个主键值，每次分页获得最后一个主键值的基础上显示。
select * from aa where id>1000 limit 20;

控制节点

Append 节点

Append 处理过程会逐个处理这些子计划(forwards or backwards),当一个子计划 返回了所有的结果后，会接着执行链表中的下一个子计划，直到链表中的所有子计划都被执行完。

适用情况：组织多个子表或子查询的执行节点。

RecursiveUnion 节点

对子节点递归进行处理 适用情况：RecursiveUnion 节点用于处理递归定义的 UNION 语句。

表结构及SQL结构

表结构（第四范式）
表结构设计尽量满足设计原则
可以通过冗余列方式，减少多表级联sql。
通过拆表结构的方法，减少处理的数据量。
逻辑设计合理表内尽量不要有重复行。
分区表
物化视图

SQL结构

1）提早过滤，过滤性越高的字段需要越靠前，过滤性低的靠后，条件一致
2）固定值放在最后（不要放在处理中间）
3）禁用select * from，取需要的列。
4）将条件合并，逻辑结构简化，简化计算
5）查询确认值设置变量 尽量不让他循环太多次
6）调用底层函数实现功能。
7）事务提交
8）多表关联时先关联结果集小的表
9）核心 SQL 可以考虑采用覆盖索引 indexonlyscan（更新少）,确保尽可能高效
10）不干扰过滤前提下，order by 排序字段进入索引
11）根据查询条件，建立索引，优化索引、优化访问方式，限制结果集的数据量。索引 应该尽量小，尽量使用字节数小的列建索引，不要对重复值多的列建单一索引。
低效sql（全部记录进行排序后过滤）： select name,count(1) from t1 group by name having name>'abc'
高效sql（过滤数据后再排序，减少中间结果集）： select name,count(1) from t where name>'abc' group by name;

索引

索引分类：Btree、hash、gin、gist等等。

Hash索引使用：hash内部表现为散列表（hash table），其数据结构内每一个元素主要为Key-Value对。Hash索引只能处理等值比较。 create index inx_t1 on t1 using hash(name);

btree索引使用：
索引列查询范围要尽可能小，否则可能用不到索引。等值、范围、模糊查询。
数据量比较大的而且选择比较少的，比如10000条以上记录，选择率在1-20%的。
隐式转换不能使用索引（不同类型字段，db会做隐式函数转换）： select * from t where name=123
正确使用索引（两边数据类型一致）： select * from t where name='123’
btree索引选择：使用表达式索引 使用局部索引 使用联合索引 使用函数索引

Gist 索引使用：
通用索引树 几何， 空间，能实现btree功能，但是效果一般。

Gin索引使用：
通用倒排索引 数组、全文检索等数据类型

brin索引使用：
存储了被索引字段在块级别的边界值（最大值、最小值）以及其他统计信息。用于线性值比较好

索引

索引分类：Btree、hash、gin、gist等等。

Hash索引使用：hash内部表现为散列表（hash table），其数据结构内每一个元素主要为Key-Value对。Hash索引只能处理等值比较。 create index inx_t1 on t1 using hash(name);

btree索引使用：
索引列查询范围要尽可能小，否则可能用不到索引。等值、范围、模糊查询。
数据量比较大的而且选择比较少的，比如10000条以上记录，选择率在1-20%的。
隐式转换不能使用索引（不同类型字段，db会做隐式函数转换）： select * from t where name=123
正确使用索引（两边数据类型一致）： select * from t where name='123’
btree索引选择：使用表达式索引 使用局部索引 使用联合索引 使用函数索引

Gist 索引使用：
通用索引树 几何， 空间，能实现btree功能，但是效果一般。

Gin索引使用：
通用倒排索引 数组、全文检索等数据类型

brin索引使用：
存储了被索引字段在块级别的边界值（最大值、最小值）以及其他统计信息。用于线性值比较好

hint

对于Scan 类型Hint，除了现有的通过修改配置参数的方式给出建议的Hint,增加几种强制的Hint：

SeqScan(table)：强制在表上使用顺序扫描。
TidScan(table)：强制在表上使用 Tid 扫描。
IndexScan(table[index...])：强制在表上使用索引扫描，只限制指定的索引。 IndexOnlyScan(table[index...])：强制在表上使用 Indexonly scan，限制特定的 索引。当 Indexonlyscan 不可用时，可以使用索引扫描。
BitmapScan(table[index...])：强制不在表上使用位图扫描。
NoSeqScan(table)：强制不在表上使用顺序扫描。
NoTidScan(table)：强制不在表上使用 Tid 扫描。
NoIndexScan(table)：强制不在表上使用索引扫描和 indexonlyscan。
NoIndexOnlyScan(table)：强制不在表上使用 indexonlyscan。
NoBitmapScan(table)：强制不在表上使用位图索引。嵌套等方式可以指定

Join

NestLoop(table table[table...])：在对指定的表进行连接时，强制使用循环嵌套 连接。
HashJoin(table table[table...])：在对指定的表进行连接时，强制使用散列连接。
MergeJoin(table table[table...])：在对指定的表进行连接时，强制使用排序合 并连接
NoNestLoop(table table[table...])：在对指定的表进行连接时，强制不使用循环 嵌套连接。
NoHashJoin(table table[table...])：在对指定的表进行连接时，强制不使用散列 连接。
NoMergeJoin(table table[table...])：在对指定的表进行连接时，强制不使用排 序合并连接。
Leading(table table[table...])：强制使用指定的连接顺序。
Leading(<join pair>)：强制使用指定的连接顺序和方向。其中一个 joinpair 可

SELECT /*+seqscan(t1)*/ * FROM t1 UNION ALL SELECT /*+indexscan(t1)*/ * FROM t1;

1）在kingbase.conf配置文件中，配置： a) enable_hint = on
2）启动数据库
3）使用Hint的注释使Hint生效：

并行

max_worker_processes
max_parallel_workers
max_parallel_workers_per_gather
max_parallel_maintenance_workers
并行查询支持的范围 KingbaseES支持并行的操作有：
- 全表扫描（Seq Scan）
- B-tree Index Scan
- Bitmap Heap Scan
- 哈希连接（Hash Join）
- 嵌套循环连接（Nested Loop Join）
- Merge Join
- 聚集（Aggregation）
- 排序（Sort）
- Append
- DDL
- 创建索引
- Create table…

不是开启的worker数量越多，查询效率越高，这个也是系统有成本考虑在内。

案例

非全列

```
TEST=#  explain analyze select distinct id,name from t_sel3 a;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=283734.00..284034.57 rows=30057 width=8) (actual time=15558.067..15568.442 rows=30000 loops=1)
   Group Key: id, name
   ->  Seq Scan on t_sel3 a  (cost=0.00..222934.00 rows=12160000 width=8) (actual time=0.131..4313.924 rows=12160000 loops=1)
 Planning Time: 0.122 ms
 Execution Time: 15572.812 ms
(5 行记录)

TEST=# create index vvv ON t_sel3  using btree(id,name);
TEST=#  set enable_seqscan=off;
SET
TEST=#  explain analyze select distinct id,name from t_sel3 a;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=0.43..376764.43 rows=30057 width=8) (actual time=0.070..7214.919 rows=30000 loops=1)
   ->  Index Only Scan using vvv on t_sel3 a  (cost=0.43..315964.43 rows=12160000 width=8) (actual time=0.068..3773.315 rows=12160000
loops=1)
         Heap Fetches: 0
 Planning Time: 0.092 ms
 Execution Time: 7222.304 ms
(5 行记录)
```

案例

```
TEST=# explain analyze select * from t_sel union select * from t_sel2;
------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=138008.95..144608.95 rows=660000 width=68) (actual time=670.129..1430.997 rows=90000 loops=1)
   ->  Sort  (cost=138008.95..139658.95 rows=660000 width=68) (actual time=670.126..1271.635 rows=660000 loops=1)
         Sort Key: t_sel.id, t_sel.name, t_sel.addr
         Sort Method: external merge  Disk: 12936kB
         ->  Append  (cost=0.00..20068.00 rows=660000 width=68) (actual time=0.013..212.659 rows=660000 loops=1)
               ->  Seq Scan on t_sel  (cost=0.00..5084.00 rows=330000 width=10) (actual time=0.013..44.205 rows=330000 loops=1)
               ->  Seq Scan on t_sel2  (cost=0.00..5084.00 rows=330000 width=10) (actual time=0.008..54.208 rows=330000 loops=1)
 Planning Time: 0.077 ms
 Execution Time: 1441.016 ms
(9 行记录)
```

分开有索引

```
TEST=# create index aaa on t_sel using btree(id,name,addr);
CREATE INDEX
TEST=# create index bbb on t_sel2 using btree(id,name,addr);
CREATE INDEX
TEST=# explain analyze select distinct * from t_sel a union select distinct b.* from t_sel2 b;
                                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=48298.63..48958.63 rows=66000 width=68) (actual time=716.272..813.160 rows=90000 loops=1)
   ->  Sort  (cost=48298.63..48463.63 rows=66000 width=68) (actual time=716.269..769.074 rows=180000 loops=1)
         Sort Key: a.id, a.name, a.addr
         Sort Method: external merge  Disk: 3536kB
         ->  Append  (cost=0.42..40306.27 rows=66000 width=68) (actual time=0.244..583.273 rows=180000 loops=1)
               ->  Unique  (cost=0.42..19657.41 rows=33000 width=10) (actual time=0.242..276.470 rows=90000 loops=1)
                     ->  Index Only Scan using aaa on t_sel a  (cost=0.42..17182.41 rows=330000 width=10) (actual time=0.239..172.076 rows=330000 loops=1)
                           Heap Fetches: 330000
               ->  Unique  (cost=0.42..19658.87 rows=33000 width=10) (actual time=0.061..273.380 rows=90000 loops=1)
                     ->  Index Only Scan using bbb on t_sel2 b  (cost=0.42..17183.87 rows=330000 width=10) (actual time=0.060..170.853 rows=330000 loops=1)
                           Heap Fetches: 330000
 Planning Time: 0.217 ms
 Execution Time: 821.385 ms
(13 行记录)
```

CETC | KINGBASE®

THANKS

成为世界卓越的数据库产品与服务提供商