

分布式数据仓库&SQL引擎Inceptor

庞磊 | 2020年11月

目录 > CONTENTS

- ① Inceptor简介
- ② Inceptor原理
- ③ Inceptor安装与配置
- ④ Inceptor SQL
- ⑤ Inceptor PL/SQL

The background of the slide features a complex network diagram. It consists of numerous small, light-gray circular nodes connected by thin, dark-gray lines. These connections form a dense web of triangles and other geometric shapes, creating a sense of interconnectedness and data flow. The overall aesthetic is clean and modern, typical of a technical or data-related presentation.

1 chapter

Inceptor简介

- ✓ 什么是Inceptor
- ✓ 适用场景
- ✓ 项目应用

➤ 定位

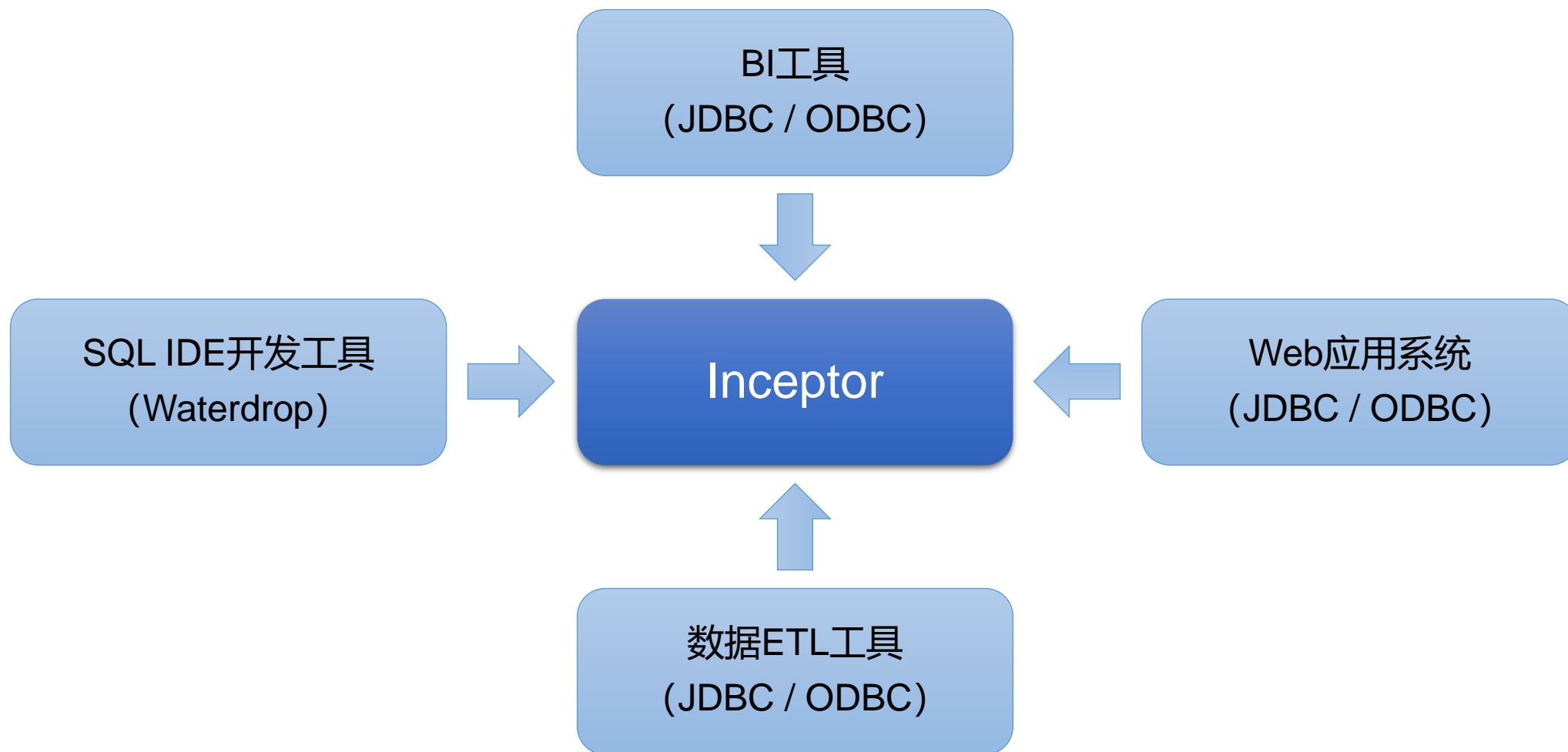
- 分布式通用SQL引擎
 - 支持Slipstream、ArgoDB、Hyperbase和Search
 - 构建星环新一代逻辑数据仓库
- 分布式数据仓库系统
- 基于Hive和Spark打造
- 用于离线分析和交互式分析（Holodesk → ArgoDB）

➤ 特点

- Hadoop领域中SQL支持最完善
 - 支持SQL99/2003标准（95%）
 - 业内唯一支持存储过程，包括Oracle PL/SQL、DB2 SQL PL等
 - 业内唯一支持Oracle、DB2、Teradata方言
 - 原有业务和系统平滑迁移

➤ 特点

- 支持完整的分布式事务处理
 - 保证分布式事务处理的ACID特性
 - 支持批量增删改查的分布式事务处理
 - 使用MVCC（Multi-Version Concurrency Control）算法自动提供并发控制
- 优异的大数据处理和分析性能
 - 世界第一个通过10TB TPC-DS的99个复杂业务场景测试
 - 基于内存/SSD的列式存储支持对数十亿条记录的秒级交互式分析（Holodesk → ArgoDB）
 - 与Apache Hive相比，数据分析处理速度有显著提升
 - 对于大规模数据集（10TB以上），数据分析处理速度比MPP有显著提升
- 提供便捷的SQL、PL/SQL开发调试辅助工具Waterdrop

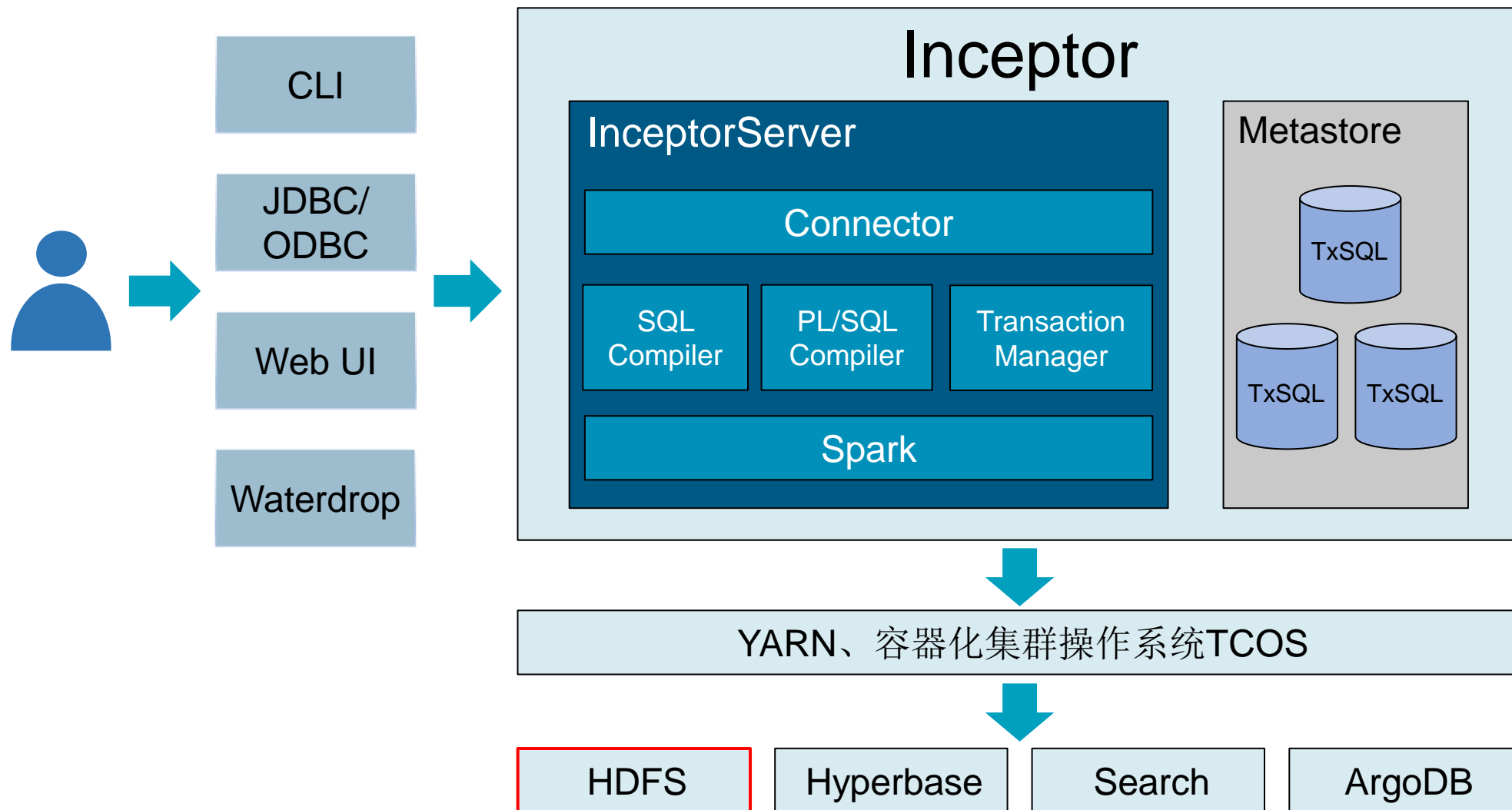




2 chapter

Inceptor原理

- ✓ 系统架构
- ✓ 数据模型



➤ InceptorServer计算引擎

- Connector
 - 为BI、ETL工具提供标准JDBC、ODBC接口
- SQL Compiler
 - SQL Parser: SQL语法解析器
 - Rule-based & Cost-based Optimizer: 规则和代价优化器
 - Code Generator: 代码生成器
- PL/SQL Compiler
 - Procedure Parser: 存储过程解析器
 - CFG Optimizer: 控制流优化器
 - Parallel Optimizer: 并行优化器

➤ InceptorServer计算引擎

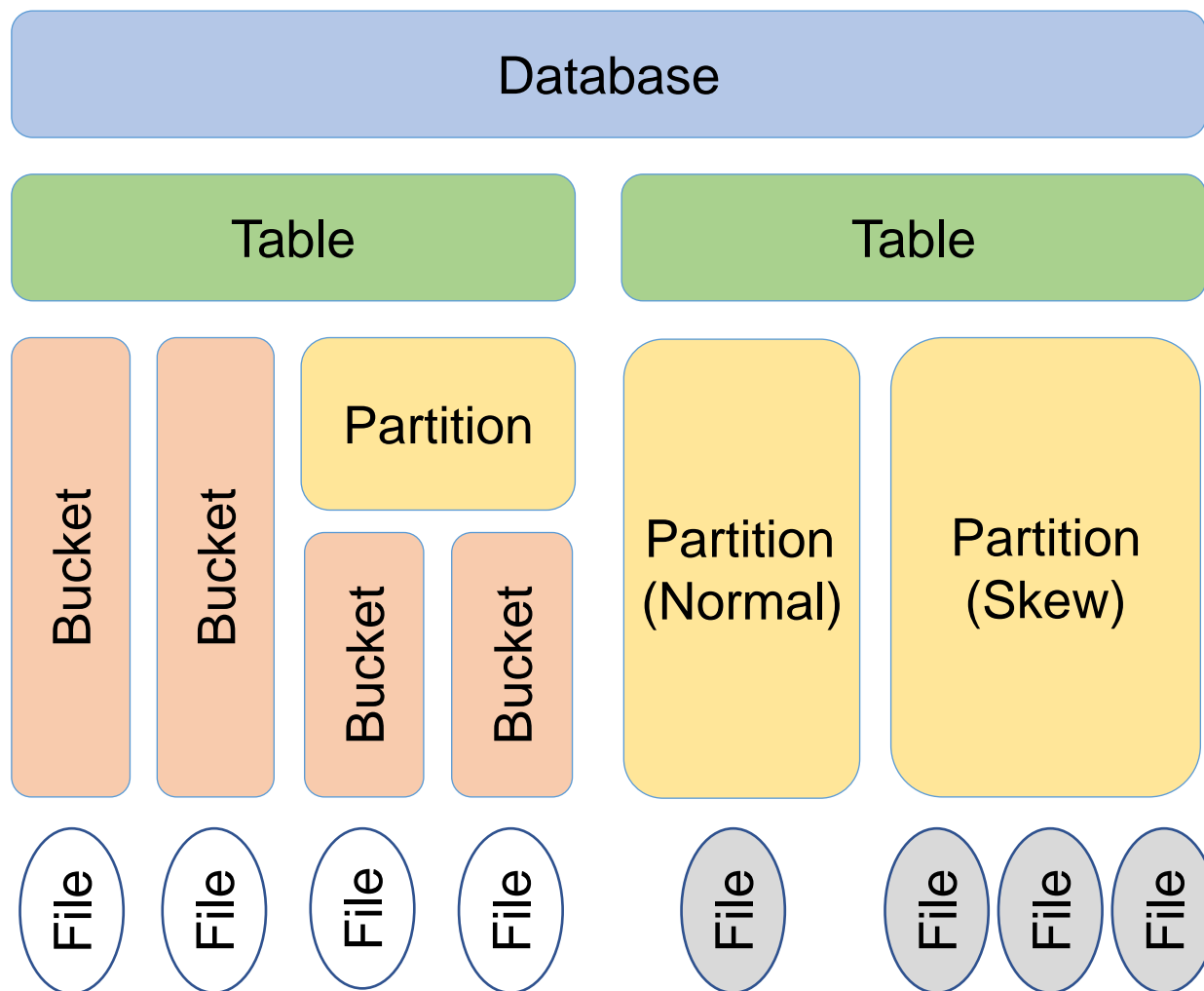
- Transaction Manager
 - Distributed CRUD: 分布式增删改
 - Concurrency Controller: 事务并发控制器
- Spark
 - 官方认证Transwarp发行版
 - 大数据量下的高稳定性
 - 计算效率更高
 - 功能扩展
- 计算资源
 - Executor是资源调度的基本单元
 - Web可视化配置

➤ Metastore

- 存储Inceptor元数据（包括Inceptor表、Slipstream流表、Hyperbase表、Search表等）
- 元数据主要包括：数据库、表、分区、分桶等信息
- 元数据存储在TxSQL（MySQL集群）

➤ Client

- CLI：Beeline
- JDBC/ODBC
- Web UI
 - HUE：开源的Apache Hadoop UI系统
 - Polit：星环打造的轻量级自助式BI分析工具
- Waterdrop
 - 高效的Inceptor SQL IDE工具
 - 支持主流的关系数据库和数据仓库，如：Oracle、DB2、MySQL、Hive、Teradata、SAP等



➤ 数据库 (Database)

- 数据库是一个包含若干表的命名空间和磁盘目录，类似于RDBMS中的数据库
- 系统会为每个数据库创建一个目录，目录名=数据库名.db
- 通常无法删除包含表的数据库，应该先删除表，再删除数据库

➤ 表 (Table)

- 表是数据管理和存储的基本对象，包含若干字段，类似于RDBMS中的表
- 表的元数据存储在Metastore中，表的实际数据存储在HDFS、Hyperbase和Search中
- 按所有权分类
 - 托管表 (Managed Table, 又称内表)
 - ①系统具有内表的完全控制权，负责管理它的生命周期
 - ②元数据存储在Metastore中
 - ③表数据默认存储在数据库目录下的对应子目录中（目录名=表名），或者人工创建的外部目录中
 - ④删除内表时，会同时删除表数据，以及Metastore中的元数据

➤ 表 (Table)

- 按所有权分类

- 外部表 (External Table, 又称外表)

- ①系统不具有外表的完全控制权

- ②元数据存储于Metastore中，表数据通常存储在指定的外部存储目录中

- ③删除外表时，不会删除表数据，但会删除Metastore中的元数据

- 按存储格式分类

- Text表

- ①系统默认的表类型，无压缩，行存储，仅支持批量Insert

- ②分析查询的性能较低，主要用于导入原始文本数据时建立过渡表

- ORC表

- ①优化的列式存储，轻量级索引，压缩比高，仅支持批量Insert

- ②Inceptor计算的主要表类型，适用于数据仓库的离线分析，通常由Text表生成

➤ 表 (Table)

- 按存储格式分类

- ORC事务表

- ①由ORC表衍生而来，继承了ORC表的所有特性，支持完整CURD（单条和批量Insert、Update、Delete），以及分布式事务处理

- ②多版本文件存储，定期做压缩，10~20%的性能损失

- Holodesk表

- ①基于内存/SSD的列式存储，内置索引和Cube，速度快，压缩率比ORC表略低，仅支持批量Insert

- ②适用于海量数据的复杂、高性能查询，如：交互式分析

- Hyperbase表

- ①数据存储Hyperbase上，支持多种索引，以及Insert、Update、Delete

- ②适用于高并发精确查询，支持半结构化、非结构化数据存储

- ES表

- 原始数据和索引数据都存储在ElasticSearch中，支持模糊查询和全文检索，适用于综合搜索

➤ 分区 (Partition)


- 目的：减少不必要的全表扫描，缩小查询范围，提升查询效率
- 含义：将表按照某个或某几个字段（分区键）划分为更小的数据集
- 分区数据存储在表目录下的子目录中，一个分区对应一个子目录，目录名为“分区键=value”

➤ 分桶 (Bucket)

- 含义：通过分桶键哈希取模 ($\text{key hashcode} \% N$) 的方式，将表或分区中的数据随机地分发到 N 个桶中，桶数 N 一般为质数，桶编号为 $0, 1, \dots, N-1$
- 作用
 - 数据划分：随机
 - 数据聚合：Key 相同的数据在同一个桶中

➤ 读时模式

- 含义：数据写入数据仓库时，不检查数据的规范性，而是在查询时再验证
- 特点
 - 数据写入速度快，适合处理大规模数据
 - 查询时处理尺度很宽松（弱校验），尽可能恢复各种错误



3 chapter

Inceptor安装与配置

- ✓ Inceptor安装
- ✓ Inceptor配置
- ✓ Inceptor命令行

➤ 安装方式

- TDH部署过程中，选择Inceptor服务



➤ 安装方式

- TDH部署完成后，在Transwarp Manager中单独添加Inceptor服务

1.选择服务

2.分配角色

3.配置服务

4.配置安全

5.服务总览

6.安装

请选择集群

选中的集群环境将决定可安装的服务种类

cluster1

选择需要安装的服务

在当前选中集群环境下可安装的服务才能被选择，当某服务被选中时，右边框将相应地显示该服务可安装的版本



TXSQL

TxSQL是一个分布式的关系型数据库



SHIVA

Shiva是一个通用的分布式存储服务



INCEPTOR

Inceptor是基于内存的交互式SQL分析引擎

选择服务的版本

商业版

社区版

Q 版本名称，表述

☒ transwarp-5.1.2-final
更新日志

☐ transwarp-5.1.1-final
更新日志

☐ transwarp-5.1.0-final
更新日志



➤ 配置方式

- Transwarp Manager → Inceptor → 配置

<

Inceptor1 | ● HEALTHY

▶ ■ ✕ 更多操作 ▾

📊 概要

👤 角色

🔧 配置

🛡️ 安全

⚙️ 插件

📁 操作

🏠 主页 > Inceptor1

可在本页编辑服务的配置。修改或者增加配置项后请点击“保存更改”按钮，更改的配置项才被保存。

🔍 executor

全部配置 ▾


+ 增加自定义参数

配置项	配置类型	配置文件	值	描述
executor.container.limits.cpu	预定义		-1	🔄 恢复推荐值
executor.container.limits.memory	预定义		-1	🔄 恢复推荐值
executor.container.requests.cpu	预定义		-1	🔄 恢复推荐值
executor.container.requests.memory	预定义		-1	🔄 恢复推荐值
executor.memory.ratio	预定义		-1	🔄 恢复推荐值
executor.number.eachnode	预定义		1	🔄 恢复推荐值
EXTRA_EXECUTOR_OPTS	预定义			
inceptor.executor.cores	预定义		<节点特定, 请点击以分别修改>	
inceptor.executor.memory	预定义		<节点特定, 请点击以分别修改>	
ngmr.executors.perjob	预定义		1	🔄 恢复推荐值

➤ Beeline

- Beeline是HiveServer 2支持的一个新命令行Shell，它是基于SQLLine CLI的JDBC客户端
- InceptorServer使用Beeline作为命令行工具
- TDH Client中包含Beeline，安装后即可使用
- 先认证登录，然后直接输入并执行SQL语句

```
/* 无认证 */  
# beeline -u jdbc:hive2://<server_ip | hostname>:10000/<database_name>  
eg: beeline -u jdbc:hive2://172.16.140.12:10000/default  
  
/* LDAP认证 */  
# beeline -u "jdbc:hive2://<server_ip | hostname>:10000/<database_name>" -n <username> -p <password>  
eg: beeline -u "jdbc:hive2://172.16.140.12:10000/default" -n hive -p 123456  
  
/*Kerberos认证 */  
# beeline -u "jdbc:hive2://<server_ip | hostname>:10000/<database_name>; principal=<principal_name>"  
eg: beeline -u "jdbc:hive2://172.16.140.12:10000/default; principal=hive/tdh-12@TDH"
```



4

chapter

Inceptor SQL

- ✓ SQL数据类型
- ✓ SQL DDL
- ✓ SQL DML

数据类型	描述	示例
TINYINT	1字节（8位）有符号整数，从-128到127	1
SMALLINT	2字节（16位）有符号整数，从-32768到32767	1
INT	4字节（32位）有符号整数，从-2147483648到2147483647	1
BIGINT	8字节（64位）有符号整数，从-9223372036854775808到9223372036854775807	1
FLOAT	4字节单精度浮点数	1.0
DOUBLE	8字节双精度浮点数	1.0
DECIMAL	不可变的、任意精度的、有符号的十进制数	1.012,1e+44
BOOLEAN	TRUE/FALSE	TRUE
STRING	字符串	'a', "a"
VARCHAR	可变长度的字符串	'a', "a"
DATE	日期，格式：'YYYY-MM-DD'，从'0001-01-01'到'9999-12-31'	'2014-01-01'
TIMESTAMP	时间戳，表示日期和时间，格式：'YYYY-MM-DD HH:MM:SS.ffffffff'，可达到小数点后9位精度（纳秒级别）	'2014-01-01 00:00:00'
INTERVAL DAY/MONTH/YEAR	一段以年、月或日为单位的时间	INTERVAL '10' day

数据类型	描述	示例
ARRAY	一组有序字段，所有字段的数据类型必须相同	ARRAY(1,2)
MAP	一组无序的键/值对，键的类型必须是原生数据类型，值的类型可以是原生或复杂数据类型；同一个MAP的键的类型必须相同，值的类型也必须相同	MAP('a',1;'b',2)
STRUCT	一组命名的字段，字段的数据类型可以不同	STRUCT('a',1,1.0)

JDBC	Inceptor
Null	Void
CHAR	String
VARCHAR	Varchar
NVARCHAR	Varchar
LongVarchar	String
LongNVarchar	String
Numeric	Decimal
Decimal	Decimal
Bit	Boolean
Boolean	Boolean
TinyInt	TinyInt
SmallInt	SmallInt
Integer	Int
BigInt	BigInt

JDBC	Inceptor
Real	Float
Float	Float
Double	Double
Binary	Binary
VarBinary	Binary
LongVarBinary	Binary
Date	Date
Time	Timestamp
TimeStamp	Timestamp
IntervarYM	interval_year_month
IntervalDS	interval_day_time
Struct	Struct
Array	Array

Oracle	Inceptor
CHAR	String
VARCHAR	Varchar
NCHAR	Varchar/String
Varchar2	String
NVarchar2	String
Number(p,s)	Decimal(p,s)
Number	Decimal(38,0)
Number(p)	Decimal(p,0)
Decimal	Decimal
Bit	Boolean
Boolean	Boolean
TinyInt	TinyInt
SmallInt	SmallInt
Integer	Int
Long	BigInt
Long Raw	BigInt
Raw	Binary
Float	Float

Oracle	Inceptor
BinaryFloat	Float
Double	Double
BinaryDouble	Double
CLOB	Binary
NCLOB	Binary
BLOB	Binary
BFile	Binary
Date	Date
Timestamp	Timestamp
Timestamp With Timezone	N/A
Timestamp With Local Timezone	N/A
Interval Year To Month	interval_year_month
Interval Day To Second	interval_day_time
Struct	Struct
Array	Array
RowId	N/A
URowId	N/A

DB2	Inceptor
CHAR	String
NCHAR	String
VARCHAR	String
NVARCHAR	String
Boolean	Boolean
SmallInt	SmallInt
Integer	Int
BigInt	BigInt
Numeric	Decimal
Decimal	Decimal
DecFloat	N/A
Real	Float

DB2	Inceptor
Float	Float
Double	Double
CLOB	Binary
BLOB	Binary
NCLOB	Binary
DBCLOB	Binary
Graphic	String
VarGraphic	String
Date	Date
Timestamp	Timestamp
Time	Time
XML	N/A

➤ 数据库操作

- Create/Drop/Alter/Use/Describe/Show Database(s)

➤ 表的基本操作

- Create/Drop/Alter/Truncate/Describe/Show Table(s)

➤ 表的高级操作

- 分区
 - 单值分区：静态分区、动态分区
 - 范围分区
- 分桶

➤ 函数操作

- Create/Drop/Describe/Show Function(s)

➤ 创建数据库

```
CREATE DATABASE [IF NOT EXISTS] <database_name>  
  [COMMENT '<database_comment>']  
  [WITH DBPROPERTIES ('<property_name>'='<property_value>', ...)];
```

➤ 删除数据库（先删表后删库）

```
DROP DATABASE [IF EXISTS] <database_name>;
```

➤ 修改数据库属性

```
ALTER DATABASE <database_name> SET DBPROPERTIES ('<property_name>'='<property_value>', ...);
```

➤ 列出所有数据库

```
SHOW DATABASES;
```

➤ 切换数据库、查看数据库详情

```
USE <database_name>;  
DESCRIBE <database_name>;
```

➤ 完整建表语句

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS]
  [<database_name>.<table_name>
  [(<col_name> <data_type> [COMMENT '<col_comment>'] [, <col_name> <data_type> ...])]
  [COMMENT '<table_comment>']
  [PARTITIONED BY (<partition_key> <data_type> [COMMENT '<partition_comment>']
    [, <partition_key> <data_type>...])]
  [CLUSTERED BY (<col_name> [, <col_name>...])
    [SORTED BY (<col_name> [ASC|DESC] [, <col_name> [ASC|DESC]...])]
    INTO <num_buckets> BUCKETS
  [
    [ROW FORMAT <row_format>]
    [STORED AS (TEXTFILE|ORC|CSVFILE)]
    | STORED BY '<storage.handler.class.name>' [WITH SERDEPROPERTIES (<...>)]
  ]
  [LOCATION '<file_path>']
  [TBLPROPERTIES ('<property_name>'='<property_value>', ...)];
```

➤ 创建内表

- 通过自定义Schema来创建内表
 - 存储在数据库目录下的同名子目录中
 - 删除内表时，同时删除表数据和元数据

```
CREATE TABLE <table_name>  
[(<col_name> <data_type> [, <col_name> <data_type> ...]);
```

- 通过已存在的表或视图来创建内表
 - 只复制Schema，不复制数据

```
CREATE TABLE <table_name>  
LIKE <existing_table_or_view_name>;
```

- 通过查询结果来创建内表
 - 既复制Schema，又复制数据

```
CREATE TABLE <table_name>  
AS SELECT <select_statement>;
```

➤ 创建外表

- 创建外表时，通常要指定数据存储位置
- 删除外表时，只删除元数据，表数据保留

```
CREATE EXTERNAL TABLE <table_name>  
  (<col_name> <data_type> [, <col_name> <data_type> ...])  
  [LOCATION '<file_path>'];
```

➤ 创建临时表

- 临时表仅在当前Session可见，Session结束后即被删除，且不支持分区
- 如果临时表和永久表重名，在当前Session中该表名指向临时表，同名的永久表无法被访问

```
CREATE TEMPORARY TABLE <table_name>  
  (<col_name> <data_type> [, <col_name> <data_type> ...]);
```

➤ 删除表

- 对于内表，表的元数据和数据都会被删除；对于外表，只删除元数据

```
DROP TABLE <table_name>;
```

➤ 修改表

```
/* 表重命名 */
```

```
ALTER TABLE <table_name> RENAME TO <new_table_name>;
```

```
/* 修改表属性 */
```

```
ALTER TABLE <table_name> SET TBLPROPERTIES ('<property_name>' = '<property_value>' ... );
```

```
ALTER TABLE <table_name> SET SERDEPROPERTIES ('<property_name>' = '<property_value>' ... );
```

```
ALTER TABLE <table_name> SET LOCATION '<new_location>';
```

```
/* 增加、删除、修改、替换列 */
```

```
ALTER TABLE <table_name> ADD COLUMNS (<col_spec> [, <col_spec> ...])
```

```
ALTER TABLE <table_name> DROP [COLUMN] <col_name>
```

```
ALTER TABLE <table_name> CHANGE <col_name> <new_col_name> <new_col_type>
```

```
ALTER TABLE <table_name> REPLACE COLUMNS (<col_spec> [, <col_spec> ...])
```

➤ 清空表

- 清空表中的数据，但不删除元数据，该操作用于内表，不能用于外表

```
TRUNCATE TABLE <table_name>;
```

➤ 查看表详情

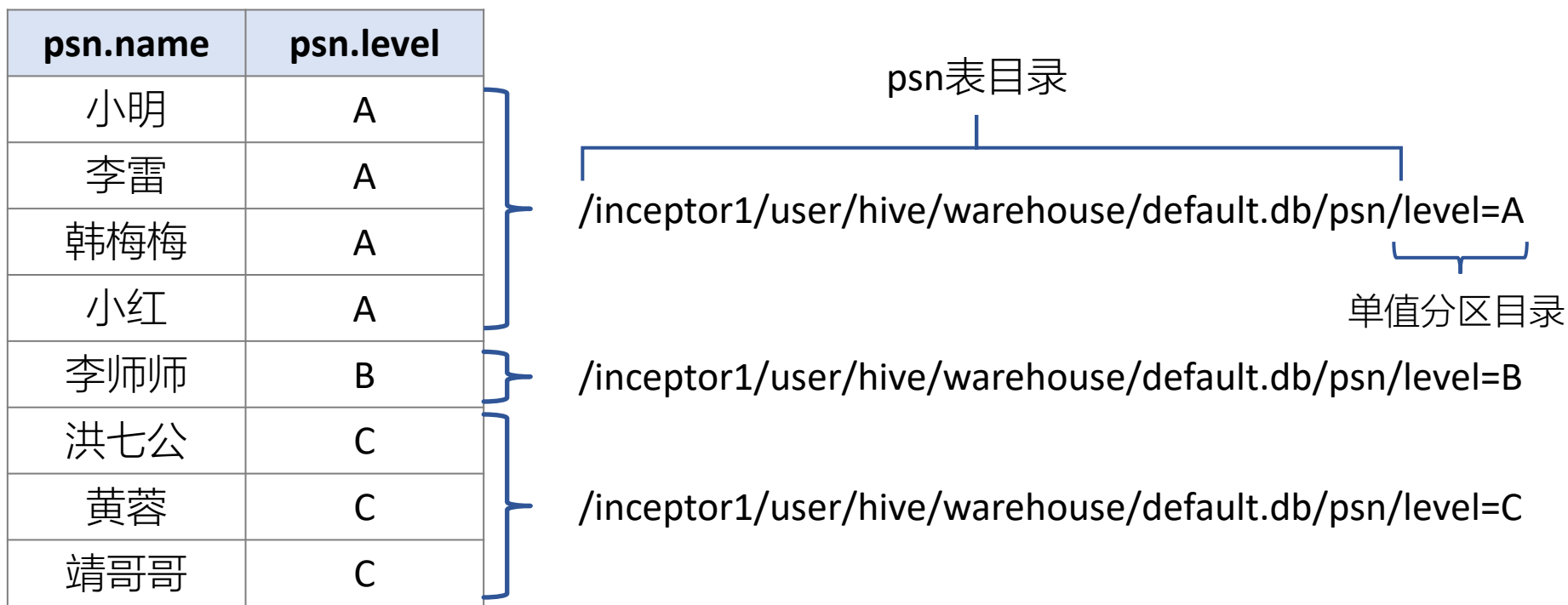
```
DESCRIBE TABLE <table_name>;
```

➤ 列出当前数据库的所有表

```
SHOW TABLES ;
```


➤ 分区

- 分区表将数据按分区键的键值存储在表目录的子目录中，目录名为“分区键=键值”
- Inceptor支持TEXT表、CSV表、ORC表和Holodesk表的分区操作
- 单值分区：一个分区对应分区键的一个值
- 范围分区：一个分区对应分区键的一个范围（区间）



➤ 分区：单值分区

- 创建单值分区时，分区键不能和表结构中的列重复，因为分区键已存储在分区目录名中，分区数据中不包含分区键，否则会造成数据冗余。另外，只写元数据，不创建分区目录。

```
CREATE [EXTERNAL] TABLE <table_name>
  (<col_name> <data_type> [, <col_name> <data_type> ...])
  PARTITIONED BY (<partition_key> <data_type>, ...)
  [CLUSTERED BY ...]
  [ROW FORMAT <row_format>]
  [STORED AS TEXTFILE|ORC|CSVFILE]
  [LOCATION '<file_path>']
  [TBLPROPERTIES ('<property_name>'='<property_value>', ...)];
```

- 单值分区可分为两类
 - 单值静态分区：导入数据时，必须手动指定目标分区
 - 单值动态分区：导入数据时，系统可以动态判断目标分区

➤ 分区：单值分区

- 将数据写入静态分区

/* 覆盖写入 */

```
INSERT OVERWRITE TABLE <table_name>  
  PARTITION (<partition_key>=<partition_value>[, <partition_key>=<partition_value>, ...])  
  SELECT <select_statement>;
```

/* 追加写入 */

```
INSERT INTO TABLE <table_name>  
  PARTITION (<partition_key>=<partition_value>[, <partition_key>=<partition_value>, ...])  
  SELECT <select_statement>;
```

/* 将HDFS数据或本地文件移动到分区目录下 */

```
LOAD DATA [LOCAL] INPATH '<path>' [OVERWRITE] INTO TABLE <tablename>  
  PARTITION (<partition_key>=<partition_value>[, <partition_key>=<partition_value>, ...]);
```

➤ 分区：单值分区

- 将数据写入动态分区

```
/* 开启动态分区支持，并设置最大分区数*/  
set hive.exec.dynamic.partition=true;  
set hive.exec.max.dynamic.partitions=2000;  
/* <dpk>为动态分区键， <spk>为静态分区键 */  
INSERT (OVERWRITE | INTO) TABLE <table_name>  
    PARTITION ([<spk>=<value>, ..., ] <dpk>, [..., <dpk>])  
SELECT <select_statement>;
```

- 一张表可同时被静态和动态分区键分区
- 动态分区键应出现在所有静态分区键之后
 - 因为HDFS上的动态分区目录下不能包含静态分区的子目录
- 向单值静态分区写入数据时，没有任何机制保证分区键和数据的正确性，所以用户必须自己确保数据写入正确的分区

➤ 分区：单值分区

- 将单个分区大小和分区数量控制在合理范围内
 - 尽量减少使用多层分区，因为分区数量等于所有分区键的分区个数的乘积，这可能导致分区数量过多
 - 对于日期时间类型的列，建议使用范围分区，因为使用单值分区会导致分区过多
 - 选择分区键和向分区写入数据时，应预估分区数据量，尽量避免数据倾斜

➤ 分区：添加、删除、重命名、清空分区

```
ALTER TABLE <table_name> ADD PARTITION (<partition_key>=<value>);  
ALTER TABLE <table_name> DROP PARTITION (<partition_key>=<value>);  
ALTER TABLE <table_name> PARTITION (<partition_key>=<value>) RENAME TO PARTITION  
    (<partition_key>=<new_value>);  
TRUNCATE TABLE <table_name> PARTITION (<partition_key>=<value>);
```

➤ 分区：范围分区

- 每个分区对应分区键的一个区间，凡是落在指定区间内的记录都被存储在对应的分区下
- 各范围分区按顺序排列，前一个分区的最大值即为后一个分区的最小值
- 创建范围分区

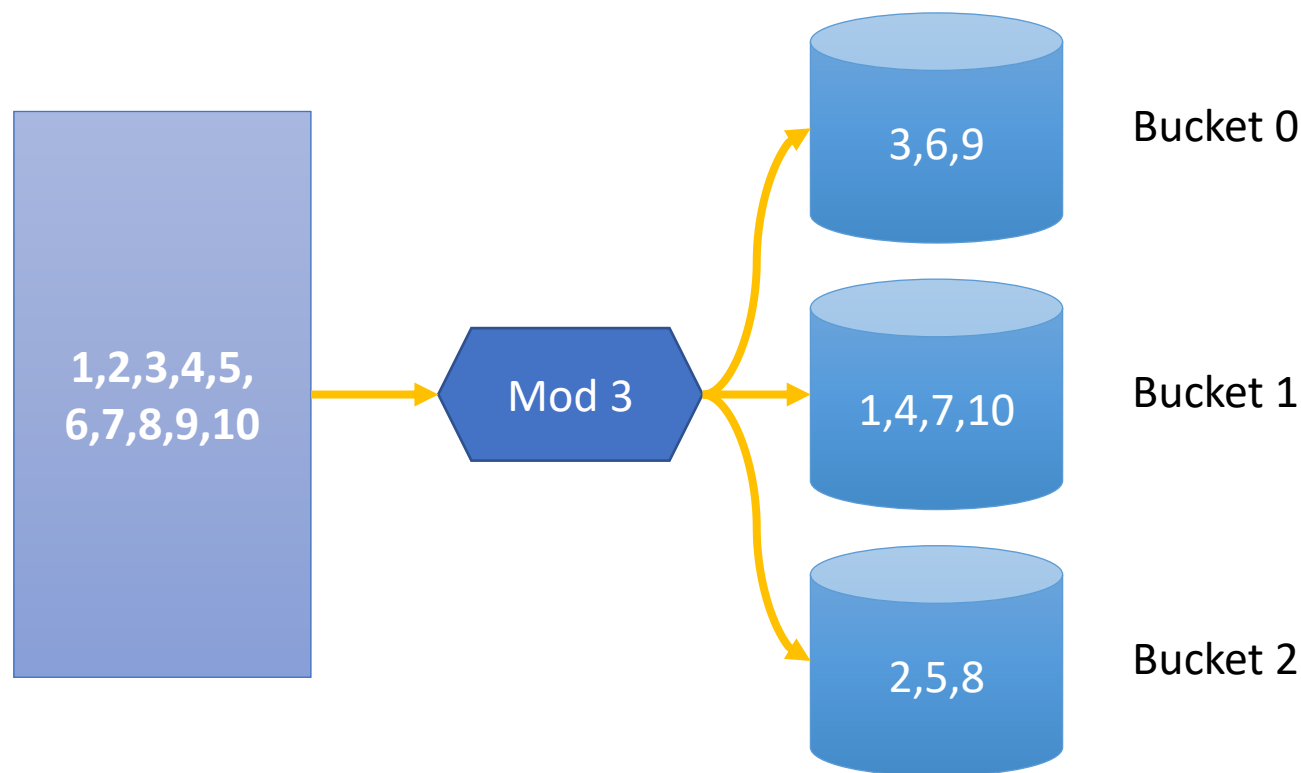
```
CREATE [EXTERNAL] TABLE <table_name>
  (<col_name> <data_type>, <col_name> <data_type>, ...)
  PARTITIONED BY RANGE (<partition_key> <data_type>, ...)
    (PARTITION [<partition_name>] VALUES LESS THAN (<cutoff>),
      [PARTITION [<partition_name>] VALUES LESS THAN (<cutoff>),
        ...
      ]
      PARTITION [<partition_name>] VALUES LESS THAN (<cutoff> | MAXVALUE)
    )
  [ROW FORMAT <row_format>] [STORED AS TEXTFILE | ORC | CSVFILE]
  [LOCATION '<file_path>']
  [TBLPROPERTIES ('<property_name>'='<property_value>', ...)];
```

➤ 分区：范围分区

- 所有范围分区均需手工指定，可在建表时就添加分区，也可在建表后通过Alter Table添加或删除分区
- 分区范围为前闭后开区间[最小值, 最大值)，即Values Less Than的含义
- 可将最后一个分区的最大值定义为MAXVALUE，关键词MAXVALUE代表分区键的最大值
- 支持以Insert Into/Overwrite...Select的形式向范围分区中写入数据，写入时无需像单值静态分区一样指定分区键的值，形式上类似于单值动态分区写入
- 不支持将文件直接导入（Load Data）范围分区
- 不支持范围分区和单值分区混用来进行多层分区

➤ 分桶 (Bucket)

- 含义：通过分桶键哈希取模（ $\text{key hashcode} \% N$ ）的方式，将表或分区中的数据随机地分发到 N 个桶中，桶数 N 一般为质数，桶编号为 $0, 1, \dots, N-1$
- 目的：通过改变数据的存储分布，提升取样、Join等特定任务的执行效率



➤ 分桶

- 创建分桶表

- INTO BUCKETS 设定桶的数量, SORTED BY 设定桶内排序, 默认升序

```
CREATE [EXTERNAL] TABLE <table_name>
  (<col_name> <data_type> [, <col_name> <data_type> ...])
  [PARTITIONED BY ...]
  CLUSTERED BY (<col_name>)
    [SORTED BY (<col_name> [ASC | DESC] [, <col_name> [ASC | DESC]...))]
  INTO <num_buckets> BUCKETS
  [ROW FORMAT <row_format>]
  [STORED AS TEXTFILE | ORC | CSVFILE]
  [LOCATION '<file_path>']
  [TBLPROPERTIES ('<property_name>'='<property_value>', ...)];
```

➤ 分桶

- 将数据写入分桶表

- 分桶表在创建的时候只定义Schema，且数据写入时系统不会自动分桶，所以需要先人工分桶再写入
- 写入分桶表只能通过Insert，而不能通过Load，因为Load只导入文件，并不分桶
- 如果分桶表创建时定义了排序键，那么数据不仅要分桶，还要排序
- 如果分桶键和排序键不同，且按降序排列，使用Distribute by ... Sort by分桶排序
- 如果分桶键和排序键相同，且按升序排列（默认），使用Cluster by分桶排序

```
/* 开启强制分桶，并设置Reduce任务数为分桶数 */
```

```
set hive.enforce.bucketing = true;
```

```
set mapreduce.job.reduces=4;
```

```
/* 写入分桶表 */
```

```
INSERT (OVERWRITE | INTO) TABLE <table_name>
```

```
    SELECT <select_expression>, <select_expression>, ... FROM <table_name>
```

```
    DISTRIBUTE BY <col_name> SORT BY <col_name> [ASC | DESC] [, col_name [ASC | DESC], ...]
```

```
    [CLUSTER BY <col_list> ]
```

➤ 分桶

- 与分区键不同，分桶键必须是表结构中的列
- 分桶键和分桶数在建表时确定，不允许更改
- 每个桶的文件大小应在100~200MB之间（ORC表压缩后的数据）
- 通常先分区后分桶
- Inceptor支持TEXT表、CSV表、ORC表、ORC事务表和Holodesk表的分桶操作，且ORC事务表必须分桶（为了支持分布式事务）

➤ 创建、删除临时函数

- Inceptor生命周期有效，并非Session有效，重启Inceptor后该函数失效

```
/* 添加jar包 */  
ADD JAR[S] <local_or_hdfs_path>;  
CREATE TEMPORARY FUNCTION <function_name> AS <class_name>;  
DROP TEMPORARY FUNCTION [IF EXISTS] <function_name>;
```

➤ 创建、删除永久函数

```
CREATE PERMANENT FUNCTION <function_name> AS <class_name>  
    [USING JAR|FILE|ARCHIVE '<file_uri>' [, JAR|FILE|ARCHIVE '<file_uri>'] ];  
DROP PERMANENT FUNCTION [IF EXISTS] <function_name>;
```

➤ 列出所有函数、查看函数详情

```
SHOW FUNCTIONS;  
DESCRIBE FUNCTION <function_name>;
```

➤ SQL DML类型

- 数据导入
 - Load导入
 - Insert导入
- 数据导出：Insert导出
- 查询：Select...From

➤ SQL DML注意事项

- 常见的DML只能用于ORC事务表、Hyperbase表
 - 例如：单行数据的增删改操作，Insert values Into、Delete、Update
- 如果写入普通表，则将指定的文件或查询结果放入表对应的目录中，该目录不能有子目录
- 如果写入分区表，则必须将文件放入对应的分区目录中

➤ 数据预处理

- 要求：文件编码为UTF-8，\n为换行符，否则需要预处理
- 处理编码
 - 如果是ASCII码，进入外表中文显示不正确

```
/* 查看文件编码类型和换行符 */
```

```
# file $filename
```

```
/* 方法一：提前处理，先转码再上传
```

```
iconv是转码工具，-f源编码格式，-t目标编码格式 */
```

```
# iconv -f gbk -t utf-8 $sourceFile > $targetFile
```

```
/* 方法二：先建外表后处理
```

```
先上传文件，建立外表后中文为乱码，执行以下语句后，再读外表中文显示正确 */
```

```
ALTER TABLE <tableName> SET SERDEPROPERTIES('serialization'='GBK');
```


➤ 数据预处理

- 处理换行符

- Windows文件用\r\n换行，而Unix文件用\n换行，所以必须处理Windows换行符，将\r\n转换为\n

```
/* 方法一：利用dos2unix工具，将Dos文件转换为Unix文件 */
```

```
# dos2unix $fileName
```

```
/* 方法二：利用vi或vim，删除Windows文件中的\r，将所有\r替换为空 */
```

```
# vim $fileName
```

```
:%s/\r//g
```

➤ 将文件导入表或分区 (Load导入)

- 将文件中的数据导入已存在的表或分区
- 仅将数据文件移动到表或分区的目录中，不会对数据进行任何处理，如分桶、排序
- 不支持动态分区导入
- 不推荐使用Load导入数据：在安全模式下的权限设置步骤较多
- 推荐的数据导入方法：创建外表，并将外表Location设置为数据文件所在目录
- INPATH选项
 - <path>是文件存储路径，如果标注LOCAL，指向本地磁盘路径，不标则指向HDFS路径
 - <path>既可指向文件也可指向目录，系统会将指定文件或目录下所有的文件复制/移动到表中
 - <path>是HDFS路径时，执行操作的用户必须是<path>的Owner，同时Inceptor用户必须有读写权限

```
LOAD DATA [LOCAL] INPATH '<path>' [OVERWRITE] INTO  
TABLE <tablename>  
[PARTITION (<partition_key>=<partition_value>, ...)];
```

➤ 将查询结果导入表或分区 (Insert导入)

- 单值静态分区导入

```
INSERT (OVERWRITE | INTO) TABLE <table_name>  
    PARTITION (<partition_key>=<partition_value>[, <partition_key>=<partition_value>, ...])  
    SELECT <select_statement>;
```

- 单值动态分区导入

- 静态分区存在的问题是需要手工输入大量的Insert语句
- 动态分区写入无需手工指定分区，而是让系统根据查询结果自动推断出分区

```
/* 开启动态分区支持，并设置最大分区数*/  
set hive.exec.dynamic.partition=true;  
set hive.exec.max.dynamic.partitions=2000;  
/* <dpk>为动态分区键， <spk>为静态分区键 */  
INSERT (OVERWRITE | INTO) TABLE <table_name>  
    PARTITION ([<spk>=<value>, ..., ] <dpk>, [..., <dpk>])  
    SELECT <select_statement>;
```

➤ 将数据导出到本地或HDFS (Insert导出)

- 将查询结果导出到本地目录，可能生成多个文件
- 与导入数据不一样，不能用Insert Into导出数据，只能用Insert Overwrite
 - 标注LOCAL代表导出到本地磁盘，不标代表导出到HDFS
 - DIRECTORY指定数据导出的文件目录
 - ROW FORMAT指定文件的行格式，STORED AS指定文件格式，不指定使用默认值

```
INSERT OVERWRITE [LOCAL] DIRECTORY <directory>  
SELECT <select_statement>  
[ROW FORMAT : DELIMITED  
  [FIELDS TERMINATED BY char [ESCAPED BY char] ]  
  [COLLECTION ITEMS TERMINATED BY char]  
  [MAP KEYS TERMINATED BY char]  
  [LINES TERMINATED BY char]  
  [NULL DEFINED AS char] ]  
[STORED AS TEXTFILE|ORC|CSVFILE]
```

➤ Select查询

- 过滤: Where、Having
- 排序: Order By、Sort By
- 分桶与聚合: Distribute By、Cluster By、Group By
- 连接: Join

```
SELECT [ALL | DISTINCT] <select_expression>, <select_expression>, ...  
  FROM [<database_name>.<table_name>  
  [WHERE <where_condition>  
  [GROUP BY <col_list>  
  [HAVING <having_condition>  
  [ORDER BY <col_name> [ASC|DESC] [, col_name [ASC|DESC], ...] ]  
  [CLUSTER BY <col_list> |  
    [DISTRIBUTE BY <col_list>] [SORT BY <col_name> [ASC|DESC] [, col_name [ASC|DESC], ...] ] ]  
  [LIMIT (M,)N | [OFFSET M ROWS FETCH NEXT | FIRST] N ROWS ONLY];
```

➤ 过滤

- **Where:** 对全表数据进行过滤，即在查询结果分组之前，将不符合条件的数据过滤掉，且条件中不能包含聚合函数
- **Having:** 对Group By产生的分组进行过滤，即在查询结果分组之后，将不符合条件的组过滤掉，且条件中常包含聚合函数
- **执行次序:** Where → Group By → Having

/* Where全表过滤 */

```
SELECT * FROM user_info WHERE reg_date < 20120000 AND level = 'A' OR level = 'B';
```

```
SELECT * FROM user_info WHERE reg_date BETWEEN 20100000 AND 20120000;
```

```
SELECT name FROM user_info WHERE level IN ('A', 'B', 'C', 'D');
```

```
SELECT name FROM user_info WHERE level NOT IN ('A', 'B', 'C', 'D');
```

/* Having分组过滤 */

```
SELECT name, avg(age) FROM user_info
```

```
WHERE reg_date < 20100000
```

```
GROUP BY level HAVING BY avg(age) < 30
```

➤ 排序

• Order By（全局排序）

- 功能：将所有数据交给一个Reduce任务计算，实现查询结果的全局排序
- 缺点：如果数据量很大，只有一个Reduce会耗费大量时间

```
SELECT <select_expression>, <select_expression>, ...  
FROM <table_name>  
ORDER BY <col_name> [ASC|DESC] [,col_name [ASC|DESC], ...]
```

• Sort By（局部排序）

- 功能：在每个Reduce任务中对数据进行排序，即局部排序
- 当启动多个Reduce任务时，Order By输出一个文件且全局有序，Sort By输出多个文件且局部有序

```
SELECT <select_expression>, <select_expression>, ...  
FROM <table_name>  
SORT BY <col_name> [ASC|DESC] [,col_name [ASC|DESC], ...]
```


➤ 分桶与聚合

• Distribute By

- 通过哈希取模的方式，实现数据按Distribute By列值分桶
- 将Distribute By列值相同的数据发送给同一个Reduce任务，实现数据按指定列聚合
- 通常与Sort By合并使用，实现先聚合后排序，且Distribute By必须在Sort By之前

```
SELECT <select_expression>, <select_expression>, ...  
FROM <table_name>  
DISTRIBUTE BY <col_list>  
[SORT BY <col_name> [ASC|DESC] [, col_name [ASC|DESC], ...] ]
```

• Cluster By

- 如果Distribute By列和Sort By列完全相同，且按升序排列，那么Cluster By = Distribute By ... Sort By

```
SELECT <select_expression>, <select_expression>, ...  
FROM <table_name>  
CLUSTER BY <col_list>
```

➤ 连接Join

- 等价连接
 - ON子句中的连接条件必须是等值条件
 - ON子句不能包含OR、BETWEEN、IN

```
SELECT <select_expression>, <select_expression>, ...  
FROM <table_name> JOIN <table_name> JOIN <table_name>, ...  
ON <equi_join_condition>
```

- 不等价连接
 - 不等价连接条件只能出现在Where子句中
 - 不等价连接与笛卡尔积类似，很容易返回大量结果，执行这样的操作必须格外小心

```
SELECT <select_expression>, <select_expression>, ...  
FROM <table_name> JOIN <table_name> JOIN <table_name>, ...  
ON <equi_join_condition>  
WHERE <non_equi_join_condition>
```



5 chapter

Inceptor PL/SQL

- ✓ PL/SQL简介
- ✓ PL/SQL基础

- Procedure Language & Structured Query Language
- 一种具有流程控制功能的数据库程序设计语言
- 优点
 - 过程化
 - 模块化
 - 运行错误可处理
 - 提供大量内置程序包
- 方言

方言类型	是否默认	Beenline设置命令
Oracle	是	<code>!set plsqlClientDialect oracle</code> <code>set plsql.server.dialect=oracle;</code>
DB2	否	<code>!set plsqlClientDialect db2</code> <code>set plsql.server.dialect=db2;</code>

➤ PL/SQL数据类型

- %type属性
 - 基于变量或数据库表的列定义一个变量。该变量的类型与被参照变量或列的类型相同；当被参照变量的类型改变之后，新定义变量的类型也会随之改变
- %rowtype属性
 - 基于数据库表定义一个记录型变量。该变量的字段名和字段类型就是被参照表的字段名和字段类型；当被参照表的结构发生变化时，新定义变量也会发生改变
- 标量类型
 - 用于保存单个值，Inceptor支持的标量类型有Int、String、Double、Boolean等
- 复合类型
 - 如：Record、Collection等

➤ PL/SQL语句块

- 既可是是一个没有名字的语句块，也可是一个命名的语句块（存储过程和函数）
- 由四个基本部分组成：声明、执行体开始、异常处理、执行体结束

```
/* 声明（可选） */  
DECLARE  
transid STRING  
/*执行体开始（必要） */  
BEGIN  
SELECT trans_id into transid from transactions where acc_num=6513065  
DBMS_OUTPUT.put_line(transid)  
/*异常处理（可选） */  
EXCEPTION --WHEN too_many_rows  
THEN  
DBMS_OUTPUT.put_line ('too many rows')  
/*执行体结束（必要） */  
END;
```

➤ PL/SQL存储过程

```
/* 创建存储过程 */  
CREATE OR REPLACE PROCEDURE hello_world()  
IS  
DECLARE  
l_message STRING := 'Hello World!'  
BEGIN  
DBMS_OUTPUT.put_line (l_message)  
END;  
  
/* 调用存储过程 */  
BEGIN  
hello_world()  
END;
```

➤ PL/SQL存储过程：参数的三种模式

- IN模式

- 参数的默认模式。调用过程的时候，实参的值会被传递到该过程中；在过程内部，形参是只读的且不能更改

- OUT模式

- 调用过程的时候，会忽略所有实参的值；在过程内部，形参即可读又可写

- INOUT模式

- IN和OUT的复合模式。调用过程的时候，实参的值会被传递到该过程中；在过程内部，形参即可读又可写；过程结束时，会将形参的内容赋给实参

➤ PL/SQL实用命令

/* 查看已有的PLSQL函数和存储过程（如不指定db_name，则为当前数据库）*/

SHOW PLSQL FUNCTIONS [db_name]

/* 查看已有的PLSQL包（如不指定db_name，则为当前数据库）*/

SHOW PLSQL PACKAGES [db_name]

/* 查看指定PLSQL函数或存储过程的详细信息（EXTENDED列出创建函数的原文）*/

DESC PLSQL FUNCTIONS [EXTENDED] <function_name>

/* 查看指定PLSQL包的详细信息（EXTENDED列出创建包的原文）*/

DESC PLSQL PACKAGES [EXTENDED] <package_name>

/* 列出正在运行的PLSQL程序的session_id */

PS PLSQL

/* 通过session_id终止正在运行的PLSQL程序 */

KILL PLSQL <session_id>

➤ PL/SQL对分号的支持

- Inceptor默认对PL/SQL语句的分号是不支持的，换句话说，只有在语句块最后使用来标志语句块结束
- 通过命令来手动打开支持。手动打开后，需要在数据块后面加上一个只包含斜杠（/）的单独行来标志数据块结束

```
/* 打开支持分号*/  
!set plsqlUseSlash true  
  
/*关闭支持分号 */  
!set plsqlUseSlash false
```



Q&A

TRANSWARP
星环科技