TRANSWARP
星 环 科 技

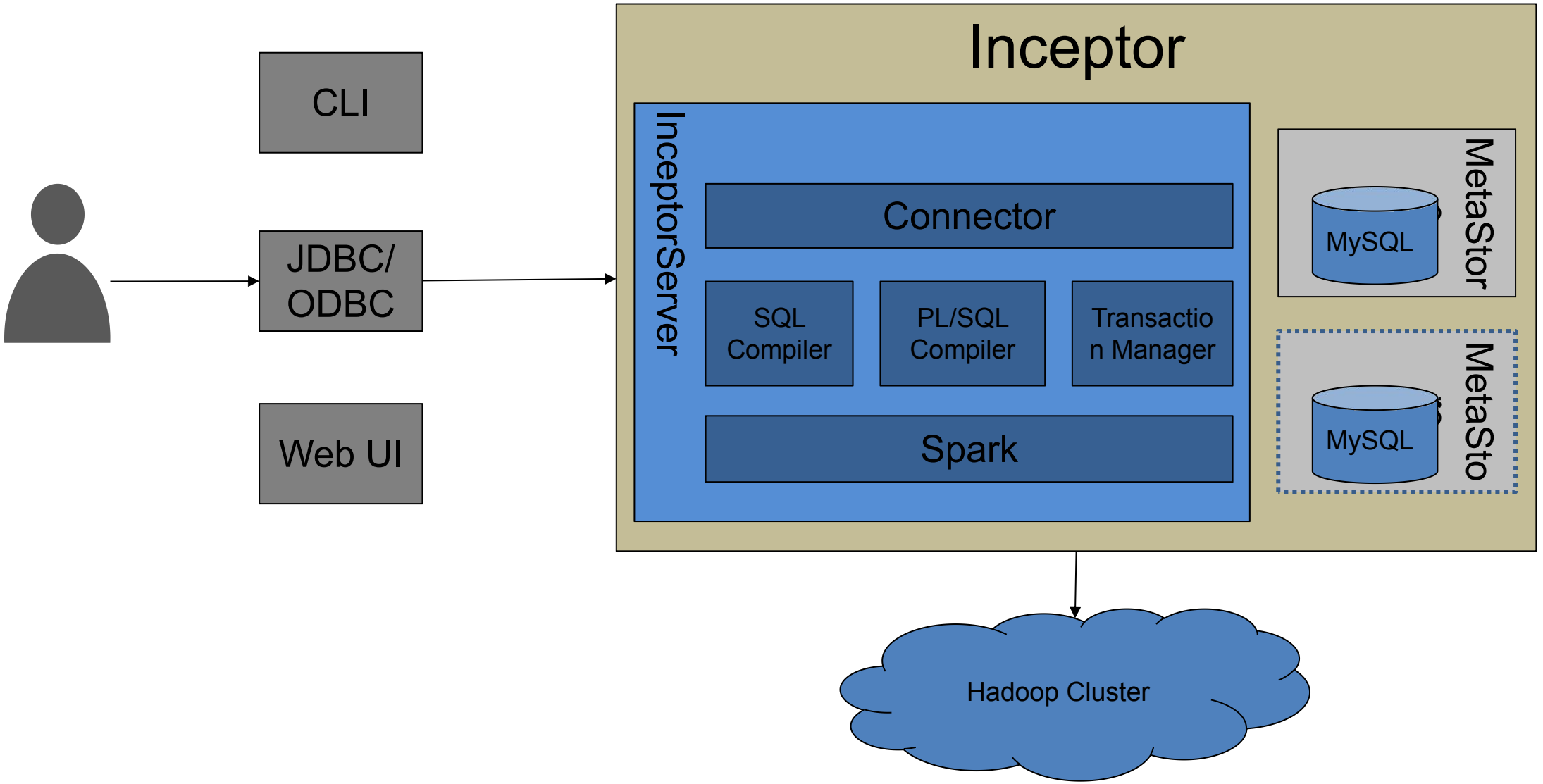
分布式SQL引擎Inceptor

星环科技
www.transwarp.io
2017年7月5日

大纲

- Inceptor架构
- Inceptor SQL
  - 基本概念
  - DDL
  - DML
- Inceptor PL/SQL
- Inceptor中表的类型
- Inceptor运维
- Inceptor优化

Inceptor架构


CLI
JDBC/
ODBC
Web UI
Inceptor
InceptorServer
Connector
SQL
Compiler
PL/SQL
Compiler
Transactio
n Manager
Spark
MetaStor
MySQL
MetaSto
MySQL
Hadoop Cluster

Inceptor访问方式

TRANSWARP 星 环 科 技

- beeline
  - 通过thrift接口对InceptorServer进行访问
  - 支持权限访问控制
- JDBC/ODBC
  - 支持JDBC 4.0/ODBC3.5标准接口
  - 被上层应用调用，如业务系统、ETL工具、BI工具等
- Web UI
  - 支持HUE通过Web服务直接运行SQL

InceptorServer

TRANSWARP 星 环 科 技

- Connector
  - 对BI/ETL工具提供标准JDBC、ODBC接口
- SQL2003 Compiler
  - 语法解析器SQL Parser
  - 优化器Rule Based Optimizer + Cost Based Optimizer
  - 代码生成 Code Generator
- PL/SQL Compiler
  - 存储过程解析器Procedure Parser
  - 控制流优化器CFG Optimizer
  - 并行优化器Parallel Optimizer
- Transaction Manager
  - 分布式增删改Distributed CRUD
  - 事务并发控制器Concurrency Controller

InceptorServer-计算引擎

TRANSWARP 星 环 科 技

- Spark
  - 官方认证transwarp发行版
  - 大数据量下的高稳定性
  - 更高计算效率
  - 功能扩展
- 计算资源
  - 以Executor形式存在
  - Executor为一组用于分布式计算的计算资源
  - 可在界面上进行配置

MetaStore

- 存储Inceptor元数据
  - 一个专门的服务进程
  - 一个用于存储元数据的MYSQL数据库
- 主要元数据
  - 数据库database信息
  - 表信息以及字段属性
  - 分区分桶信息

MetaStore HA

TRANSWARP 星 环 科 技

- 实现高可用性
  - 安装Inceptor服务时安装两个Metastore角色
- Master-Master-Active-Passive模式
  - 两个Master节点，同时只有一个MYSQL对外提供服务
  - 另一个MYSQL同步Active的数据
  - 自动failover切换
  - Passive节点对用户为只读模式

大纲

- Inceptor架构
- Inceptor SQL
  - 基本概念
  - DDL
  - DML
- Inceptor PL/SQL
- Inceptor中表的类型
- Inceptor运维
- Inceptor优化

基础数据类型

TRANSWARP 星 环 科 技

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-922337203685477580 8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	'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.fffffffff'，可达到小数点后9位（纳秒级别）精度	'2014-01-01 00:00:00'
INTERVAL DAY/MONTH/YEAR	用于存储一段以年,月或日为单位的时间	INTERVAL '10' day

复杂数据类型

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)

数据类型DB2与Inceptor对比

TRANSWARP 星 环 科 技

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

数据类型oracle与Inceptor对比

TRANSWARP 星 环 科 技

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

Oracle	Inceptor
Float	Float
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

数据类型JDBC与Inceptor对比

TRANSWARP 星环科技

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

Inceptor库表对象

TRANSWARP 星 环 科 技

- 数据库（DATABASE）
  - 创建新数据库

```
transwarp> CREATE DATABASE test_db;
```

  - 查看数据库

```
transwarp> DESCRIBE DATABASE test_db;
test_db hdfs://ns/inceptor1/user/hive/warehouse/test_db.db
```

  - 查看Inceptor中的所有数据库

```
transwarp> SHOW DATABASES;
default
my_db
test
test_db
```

Inceptor库表对象

TRANSWARP 星 环 科 技

- 表（TABLE）
  - 托管表： CREATE TABLE语句默认创建托管表。Inceptor对它有所有权。用DROP语句删除托管表时，Inceptor会将表对应的目录下的数据全部删除。
  - 外表： 外表用CREATE EXTERNAL TABLE语句创建，它必须存储在HDFS上，而不是本地。Inceptor对它无所有权。Inceptor和Hadoop系统中的其他组件都可以对这张表进行操作。用DROP语句删除外部表时，Inceptor删除表在metastore中的元数据而不删除文件。
  - 创建外部表时，可以使用Location'hdfs_path'指令指定外部表在HDFS中的路径。
  - 表的schema：表的schema是关于表的元数据，schema内容包括表中的列名，列的数值类型，分区键，分区键,数值类型，列的注解等等，表的存储路径，表的SerDe等等信息。

```
transwarp> CREATE TABLE test_table (col INT, col2 STRING COMMENT 'this is
a comment on col2') COMMENT 'this is a comment on test_table' PARTITIONED BY (col3 DOUBLE);
```

Inceptor库表对象

TRANSWARP 星 环 科 技

- 表（TABLE）
  - 表的schema：需要查看表的schema可以使用DESCRIBE和DESCRIBE EXTENDED语句。

DESCRIBE语句可以显示列名，列的数值类型和列的注解表。

```
DESCRIBE test_table;
col int None
col2 string this is a comment on col2
col3 double None
```

  - 要查看指定数据库下所有的表可以使用SHOW TABLES指令。

```
transwarp> SHOW TABLES;
test_table
```

  - SHOW CREATE table_name查看表的DDL。

```
transwarp> SHOW CREATE TABLE table_name;
```

大纲

- Inceptor架构
- Inceptor SQL
  - 基本概念
  - DDL
  - DML
- Inceptor PL/SQL
- Inceptor中表的类型
- Inceptor运维
- Inceptor优化

DDL

- 创建/删除/修改数据库：CREATE/DROP/ALTER DATABASE
- 创建/清空/删除表：CREATE/TRUNCATE/DROP TABLE
- 表的分区：PARTITIONED BY子句
- 表的分桶：CLUSTERED BY子句
- 修改表/分区/列：ALTER TABLE/PARTITION/COLUMN
- 创建/修改/删除视图：CREATE/DROP/ALTER VIEW
- 创建/删除函数：CREATE/DROP FUNCTION
- 查看已有数据库/表/函数：SHOW
- 描述表和数据库：DESCRIBE和DESCRIBE EXTENDED

创建数据库

- CREATE DATABASE

```
CREATE DATABASE [IF NOT EXISTS] database_name
[COMMENT database_comment][LOCATION hdfs_path]
[WITH DBPROPERTIES
(property_name=property_value, ...)]
```

- DROP DATABASE

```
DROP DATABASE database_name
```

- USE DATABASE

```
USE database_name
```

基础建表语句

TRANSWARP 星 环 科 技

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name [ASC|DESC], ...)] INTO
num_buckets BUCKETS]
[SKEWED BY (col_name, col_name, ...) ON ([(col_value, col_value, ...), ...|
col_value, col_value, ...])
[STORED AS DIRECTORIES]]
[
[ROW FORMAT row_format]
[STORED AS file_format]
| STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]
]
[LOCATION hdfs_path]
[TBLPROPERTIES (property_name=property_value, ...)]
[AS select_statement];
```

基础建表语句

- 简单的CREATE TABLE语句

```
CREATE TABLE table_name (col1 col_type1, col2 col_type2, ...)
```

- 创建临时表

```
CREATE TEMPORARY TABLE table_name(column_name data_type, column_name data_type...)
```

- 创建外表

```
CREATE EXTERNAL TABLE table_name (col1 data_type1, col2 data_type2, ...)
ROW FORMAT row_format
STORED AS file_format
LOCATION 'hdfs_path'
```

- CTAS创建一个表来存放一次查询的结果

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

表的分区

- Inceptor支持对表的单值分区和范围分区
  - 在物理上，将表中的数据按分区放在表目录下的对应子目录中，一个分区对应一个子目录
  - 在逻辑上，分区表和未分区表没有区别

  - 创建单值分区表

```
CREATE TABLE table_name (col1 data_type1, col2, data_type2, ... )
PARTITIONED BY (partition_key1 data_type1, partition_key2 data_type2...)

分区改变了Inceptor对数据的存储— Inceptor会在表的目录下加上分区对应的子目录：
…/part_user_info/acc_level=A
…/part_user_info/acc_level=B
…/part_user_info/acc_level=C
…/part_user_info/acc_level=D
…/part_user_info/acc_level=E
```

  - 当我们做一个区内的查询时，Inceptor只需要去读取对应子目录下的信息

```
transwarp> SELECT * FROM partition_user_info WHERE acc_level ='A';
```

表的分区

- 关于分区的建议
  - 分区的目的是减少扫描成本。所以单个分区的大小和总分区数目都应该控制在合理范围内。
  - 使用多层分区带来的直接问题是总分区个数过多，因为总分区个数是所有分区键对应分区个数的乘积。所以我们建议尽量减少使用多层分区。
  - 对于时间、日期一类的值，使用单值分区会导致分区过多。推荐使用*范围分区(RANGE PARTITION)*。

表的分区

- 范围分区
  - 范围分区时，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会被放入对应的分区下。
  - 各个分区之间按顺序排列，前一个分区的最大值即为后一个分区的最小值，第一个分区的最小值为该字段类型所允许的最小值。
  - 您可以将最后一个分区的上限指定为MAXVALUE，关键词MAXVALUE代表该字段类型所允许的最大值。

```
CREATE TABLE table_name (column_name column_type, ...)
PARTITIONED BY RANGE (partition_key_name, partition_key_type, ...)
(PARTITION [partition_name] VALUES LESS THAN (partition_value1),
PARTITION [partition_name] VALUES LESS THAN (partition_value2),
...
PARTITION [partition_name] VALUES LESS THAN (MAXVALUE))
```

表的分区

TRANSWARP 星 环 科 技

- 范围分区注意事项
  - 所有range partition分区均需要手工指定，您可以在建表的时候就建好分区，也可以在建表后通过alter table语句为表添加或删除分区。
  - 分区的范围为* [最小值, 最大值)* 前闭后开区间，即value less than的字面意义
  - 不支持 从文件导入范围分区表。
  - 支持INSERT INTO/OVERWRITE…SELECT…FROM…；形式向范围分区表中插入数据，插入时不需要像单值分区一样指定分区字段的值，形式上类似于动态分区插入。
  - *支持*使用ALTER TABLE语法添加和删除范围分区。
  - 不支持 和单值分区混用来进行多层分区。
  - Inceptor没有任何机制在将数据写入分区时保证分区键的正确性，所以向表中填入数据时，用户必须自己确保记录导入或者插入正确的对应分区中。

- 分区表
  - Inceptor支持单值分区和范围分区
  - 在物理上，将表中的数据按分区放在表目录的对应子目录中
  - 在逻辑上，分区表和未分区表没有区别
  - 分区在创建表时完成，也可以通过Alter Table来添加或者删除
  - 范围分区时，每个分区对应分区键的一个区间。前一个分区的最大值即为后一个分区的最小值
  - 单值分区insert需要指定插入分区，范围分区支持动态插入

- 注意事项
  - 显示指定分区时，inceptor对分区数据写入不做正确性检查
  - 不推荐多层分区
  - 分区字段应选择时间、日期类型
  - 推荐使用范围分区

修改表/分区

- 重命名

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

- 改动表的注释

```
ALTER TABLE table_name SET TBLPROPERTIES ('comment' = new_comment);
```

- 添加分区

```
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec
[LOCATION 'location1'];
注意，该目录必须是HDFS目录，不可以是文件。
```

- 添加range partition

```
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION VALUES LESS THAN (values)
```

- 删除分区

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec;
动态分区：
ALTER TABLE table name DROP PARTITION partition_name;
```

表的分桶

- 为表分桶可以让某些任务（比如取样，join等）运行得更快。表的分桶在建表时完成。

```
CREATE TABLE table_name (col_name data_type, col_name data_type, ...)
CLUSTERED BY (col_name,)
[SORTED BY (col_name, col_name,...)[ASC|DESC]]
INTO n BUCKETS

INTO n BUCKETS指定桶的数量
SORTED BY根据指定的键排序。ASC表示升序，DESC表示降序。默认顺序是升序。
和分区键不同，分桶键必须是表中的列。
```

  - 向分桶表写入数据只能通过INSERT，而不能LOAD。

```
正确的给分桶表写入数据
set hive.enforce.bucketing = true;

INSERT OVERWRITE TABLE table_name SELECT * FROM table_name;
```

分桶表

- 分桶表
  - 分桶表通过改变数据的存储分布，对查询起到一定的优化作用
  - 和分区键不同，分桶键必须是表中的列
  - 分桶默认对分桶列做hash取模的方式，分桶数应为质数
  - 每个桶的文件大小应在100MB-200MB之间（orc表压缩后的数据）
  - 不同表对同一字段分桶，且当两张表桶数相同时，数据会分配到同一个节点，join时减少shuffle
  - 事务表必须制定分桶，且分桶字段是不可以被更新的

删除表

- DROP TABLE语句

```
transwarp> DROP TABLE table_name;
```

  - 当被删除的表是托管表时，表的元数据和表中数据都会被删除。
  - 如果被删除的表是外部表，则只有它的元数据会被删除。

创建/删除函数

- 创建临时函数

用户也可以通过ADD JAR向class path加jar包。

CREATE TEMPORARY FUNCTION function_name AS class_name;
以上语句创建一个由class_name实施的临时函数。

- 删除临时函数

DROP TEMPORARY FUNCTION [IF EXISTS] function_name

注意：临时函数是Inceptor生命周期有效，并非Session有效，即在重启Inceptor后该函数将不再存在，在重启Inceptor前在各个session间都是有效的

创建/删除函数

TRANSWARP 星 环 科 技

- 创建永久函数

用户也可以通过ADD JAR向class path加jar包。

CREATE PERMANENT FUNCTION function_name AS class_name;
以上语句创建一个由class_name实施的永久函数。

- 删除永久函数

DROP PERMANENT FUNCTION [IF EXISTS] function_name

注意：永久函数是永久有效，即在重启Inceptor后该函数仍存在

SHOW

- 查看数据库

```
SHOW DATABASES;
```

- 查看表

```
SHOW TABLES;
```

- 查看分区

```
SHOW PARTITIONS table_name;
```

- 查看指定表的建表语句

```
SHOW CREATE TABLE ([db_name.]table_name)
```

- 查看函数

```
SHOW FUNCTIONS;
```

大纲

- Inceptor架构
- Inceptor SQL
  - 基本概念
  - DDL
  - DML
- Inceptor PL/SQL
- Inceptor中表的种类
- Inceptor运维
- Inceptor优化

- 数据操作语言（DML）
  - 导入数据：LOAD
  - 插入数据：INSERT
  - 命令将指定的文件或查询的结果放入表对应的目录中，这个目录不能有子目录
  - 如果被填充的表是分区表，则必须指定将文件放入对应的分区中
  - SQL中常用的UPDATE，DELETE和INSERT VALUE INTO只能对Hyperbase表和事务表使用

导入数据

- LOAD
  - LOAD语句将文件中的数据导入已创建的表。
  - 导入操作仅仅将数据文移动到表或分区所对应的地址中，Inceptor不会在LOAD时对数据进行任何处理。

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]
```

filepath（文件路径） 是存放数据文件的地址。如果不加上LOCAL关键字，filepath是HDFS上的路径。
filepath可以指向一个文件也可以指向一个目录。当它指向一个文件时，Inceptor会将文件移入表内。当它指向一个目录时，Inceptor会将目录下所有的文件移入表内。

如果加上了OVERWRITE选项，那目标表或者分区已有的内容会被导入的文件覆盖；如果不加OVERWRITE选项，导入的文件不覆盖已有文件。但是如果目标表或者分区中存在文件和被导入的文件重名，那么原先的文件会被新文件覆盖。

插入数据

- 普通INSERT

```
INSERT [OVERWRITE|INTO] TABLE tablename1 [PARTITION (partcol1=val1,
partcol2=val2 ...)[IF NOT EXISTS]]select_statement1 FROM from_statement
```

- 动态分区插入
  - 查询结果中分区很多带来的另外一个问题是需要手工输入非常多的INSERT语句。
  - 动态分区插入可以不用指定分区，而是让Inceptor根据查询结果自动推断出分区。

```
set hive.exec.max.dynamic.partitions=2000;
（动态分区的最大分区数）
set hive.exec.dynamic.partition=true;
（设置为true允许使用dynamic partition）

INSERT [OVERWRITE|INTO] TABLE tablename PARTITION (partcol1[=val1],
partcol2[=val2] ...)
select_statement FROM from_statement;
```

写入本地文件系统

INSERT OVERWRITE **LOCAL** DIRECTORY <directory> [**ROW** FORMAT <row_format>] [STORED **AS** <file_format>]

SELECT ... **FROM** ...

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**]

注意，这个语句将查询结果写入一个 目录，而不是文件，写入的结果可能是多个文件。

写入本地文件系统要加上 LOCAL 关键字；

ROW FORMAT指定文件的行格式，不指定使用默认值；

STORED AS指定文件格式，不指定则使用默认值；

写入HDFS

INSERT OVERWRITE DIRECTORY <directory> [**ROW** FORMAT <row_format>] [STORED **AS** <file_format>]
SELECT ... **FROM** ...
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**]

注意，这个语句将查询结果写入一个 目录，而不是文件，写入的结果可能是多个文件。
ROW FORMAT指定文件的行格式，不指定使用默认值；
STORED AS指定文件格式，不指定则使用默认值；

数据导入流程

- 数据预处理

file $filename
可以看到编码格式和以什么为换行符
格式最好是UTF-8和\n换行的，可以直接使用，否则需要进行预处理

- 编码
  - 如果是ASCii码，进入外表中文显示不正确

方法一：提前处理
iconv -f gbk -t utf-8 $sourceFile > $targetFile
iconv是转码工具，-f源编码格式，-t目标编码格式。转码后再上传。

方法二：外表处理
先上传，建立外表后中文为乱码，此时
alter table $tableName set serdeproerties('serialization'='GBK');
再读外表，中文显示正确。

数据导入流程

TRANSWARP 星 环 科 技

- 数据预处理

- 换行符
  - 如果是CRLF换行的，是windows下产生的文件，以\r\n换行，unix系统用\n换行，需要进行调整。

方法一：用dos2unix工具转换
dos2unix $fileName
dos2unix指令顾名思义，dos系统下的文件转换成unix系统下的文件。如果没有dos2unix，直接安装一个：*yum install doc2unix*

方法二：直接vi或vim删除\r
vim $fileName
:%s/\r//g
将所有\r替换为空。

SELECT语句

TRANSWARP 星 环 科 技

```
SELECT [ALL | DISTINCT] select_expression, select_expression, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[CLUSTER BY col_list
| [DISTRIBUTE BY col_list] [SORT BY col_list]
]
[LIMIT number];
```

ALL和DISTINCT告诉Inceptor是否要返回查询中出现的重复行。选择DISTINCT选项Inceptor会不返回重复行；选择ALL则返回所有行，无论重复与否。不选默认为选择ALL。

LIMIT限制返回的行数。Inceptor随机选择返回的行。

SELECT语句

- WHERE和HAVING
  - 查询中的一个常见操作是使用过滤来获得对用户有用的信息。
  - WHERE子句和HAVING子句的区别在于，一次查询中如果有WHERE子句，Inceptor会先执行WHERE子句的过滤条件再执行SELECT语句，而查询中如果用到了HAVING子句，Inceptor会先执行SELECT语句，再执行HAVING子句。
  - 换句话说，WHERE在SELECT之前过滤，而HAVING在SELECT之后过滤。

```
SELECT * FROM partition_user_info WHERE reg_date <20100000;
SELECT * FROM partition_user_info WHERE reg_date < 20120000 AND
acc_level = 'A' OR acc_level = 'B';
SELECT * FROM partition_user_info WHERE reg_date >20100000 AND reg_date <20120000;
SELECT * FROM partition_user_info WHERE reg_date BETWEEN 20100000
AND 20120000;
SELECT name FROM user_info WHERE acc_level IN ('A', 'B', 'C');
SELECT name FROM user_info WHERE acc_level NOT IN ('A', 'B', 'C');
```

SELECT语句

- WHERE和HAVING
  - 如果过滤条件受带GROUP BY的查询结果影响，那么就不能用WHERE子句来过滤，而要用HAVING子句。

```
transwarp> SELECT acc_num, max(price*amount)
FROM transactions
WHERE trans_time<'20140630235959'
GROUP BY acc_num
HAVING max(price*amount)>5000;
```

SELECT语句

- ORDER BY

```
SELECT select_statement ORDER BY col_name [ASC|DESC] [,col_name_2 [ASC|DESC],...]
```

  - ORDER BY对查询结果进行 全排序(total ordering)，所以所有数据都会经过一个单独的reducer。如果数据很多，只有一个reducer会导致计算花费大量时间。
  - ORDER BY子句后必须跟着LIMIT子句

- SORT BY

```
SELECT select_statement SORT BY col_name [ASC|DESC] [,col_name_2 [ASC|DESC],...]
```

  - SORT BY在每个reducer之内排序，来达到 局部有序(local ordering)。
  - ORDER BY和SORT BY 的语法几乎完全一样，但是当一个任务用到了不止一个reducer，ORDER BY和SORT BY的输出会不一样。

- DISTRIBUTE BY与SORT BY合用

```
SELECT select_statement
DISTRIBUTE BY col_name_1
SORT BY col_name_2 [ASC|DESC] [,col_name_3 [ASC|DESC],...]
```

  - 所有DISTRIBUTE BY列的列值相同的记录会被放进同一个reducer中。

SELECT语句

- CLUSTER BY

```
SELECT * FROM user_info DISTRIBUTE BY acc_level SORT BY acc_level;
SELECT * FROM user_info CLUSTER BY acc_level;
```

  - 如果DISTRIBUTE BY和SORT BY子句中的列是同一个而且SORT BY顺序选择是升序，那么DISTRIBUTE BY col SORT BY col可以用CLUSTER BY col来代替，效果完全相同。

- GROUP BY

```
SELECT select_expression, select_expression, ...
GROUP BY groupby_expression [, groupby_expression, ...]
```

  - GROUP BY子句将查询结果按列值并组，也就是指定列列值相同的将被并入同组。
  - GROUP BY常常与聚合函数合用—— 将查询结果按列值并组，然后再对每组分别使用聚合函数。
  - 如果过滤条件受带GROUP BY的查询结果影响，那么就不能用WHERE子句来过滤，而要用HAVING子句
  - 注意：在WHERE子句中不能有聚合函数，因为Inceptor在执行GROUP BY子句之前就会执行WHERE子句。

SELECT语句

- JOIN

```
SELECT select_expression, select_expression, ...
FROM table_reference JOIN table_reference JOIN table_reference, ... [ON
join_condition]
```

  - join_condition 连接条件是等值条件（这种连接叫做等价连接）
  - 注意:不支持JOIN条件中带有OR条件,也不支持JOIN条件中使用BETWEEN/IN。

- 不等价连接

```
SELECT select_expression, select_expression, ...
FROM table_reference1 JOIN table reference_2
ON equi_join_condition
WHERE non_equi_join_condition
```

  - 要执行不等价连接，ON子句中的连接条件必须是等价条件，不等价条件体现在WHERE子句中的过滤条件中。
  - 不等价连接和笛卡尔积相像，很容易返回大量结果,执行这样的操作必须格外小心。

INSERT/UPDATE/DELETE语句。。。

TRANSWARP 星环科技

大纲

- Inceptor架构
- Inceptor SQL
  - 基本概念
  - DDL
  - DML
- Inceptor PL/SQL
- Inceptor中表的种类
- Inceptor运维
- Inceptor优化

PL/SQL基础

TRANSWARP 星 环 科 技

- PL/SQL(Procedure Language & Structured Query Language)是一种具有流程控制的数据库程序设计语言。

- 好处
  - 过程化
  - 模块化
  - 运行错误的可处理性
  - 提供大量内置的程序包

- 方言选择

方言类型	是否默认	CLI 设置命令	Beeline 设置命令
Oracle	是	set plsql.client.dialect=oracle; set plsql.server.dialect=oracle;	!set plsqlClientDialect oracle set plsql.server.dialect=oracle;
DB2	否	set plsql.client.dialect=db2; set plsql.server.dialect=db2;	!set plsqlClientDialect db2 set plsql.server.dialect=db2;

- 数据类型
  - %type属性：即基于变量声明一个变量，该变量的类型为基于字段的类型，当基于变量类型发生变化时，改变量的类型也会发生改变。
  - %rowtype属性：即基于表声明一个记录型变量，该记录型变量的字段名和字段类型分别为表中的字段名和字段类型，当基于表结构发生变化时，该记录型变量也会发生改变。
  - 标量类型的含义是存放单个值。Inceptor中支持的标量类型为INT/STRING/DOUBLE等Inceptor中所有支持的数据类型。
  - 复合类型：Record，Collection等

PL/SQL基础

- PL/SQL语句块
  - PL/SQL语句块可以是一个没有名字的语句块，也可以是命名的语句块，即存储过程和函数
  - PL/SQL块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

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

- 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;
```

- 带参数的存储过程
  - IN类型：默认模式，在调用procedure的时候，procedure的实参的值被传递到该procedure，在procedure的内部，procedure的形参是只读的；不指定则默认为该类型；
  - OUT类型：在调用procedure的时候，不能传递给procedure实参的值，在procedure的内部，procedure的形参是只可写的；
  - INOUT类型：在调用procedure的时候，实参的值可以被传递给该procedure，在其内部，形参可以被读出也可以被写入，该procedure结束时，形参的内容将赋给调用时的实参

- 优化手段

是否优化	HiveServer1设置命令	HiveServer2设置命令
是	set plsql.optimize.dml.use.queryplan=true;	!set plsqlOptimizeDmlUseQueryplan true
否	set plsql.optimize.dml.use.queryplan=false;	!set plsqlOptimizeDmlUseQueryplan false

  - **作用**：将PLSQL中的DML语句一次性预编译成执行计划，省去频繁对语法树做语义分析的开销。
  - **适用范围**：循环体中有DML语句，或是一次编译反复运行的场景（streamsql）

PL/SQL基础

TRANSWARP 星 环 科 技

- 实用命令

SHOW PLSQL FUNCTIONS [db_name]
查看已有的plsql函数和存储过程，db_name为可选，不指定db_name的话为当前数据库

SHOW PLSQL PACKAGES [db_name]
查看已有的plsql包，db_name为可选，不指定db_name的话为当前数据库

DESC PLSQL FUNCTIONS [EXTENDED] function_name
查看某一plsql函数或存储过程的详细信息，加EXTENDED会列出创建该function的原文

DESC PLSQL PACKAGES [EXTENDED] PACKAGE_NAME
查看某一plsql包的详细信息，加EXTENDED会列出创建该package的原文

PS PLSQL
列出正在运行的plsql程序的session_id，仅在hiveserver2中有效

KILL PLSQL <session_id>
通过session_id终止正在运行的plsql程序，仅在hiveserver2中有效

PL/SQL基础

TRANSWARP 星 环 科 技

- 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。
  - 我们可以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。

HiveServer1中打开方式

```
set plsql.use.slash=true; //打开支持分号
set plsql.use.slash=false; //关闭支持分号
```

HiveServer2中打开方式

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

大纲

- Inceptor架构
- Inceptor SQL
  - 基本概念
  - DDL
  - DML
- Inceptor PL/SQL
- Inceptor中表的类型
- Inceptor运维
- Inceptor优化

表类型

TRANSWARP 星 环 科 技

- Text表
  - 无压缩，行存储，只支持批量insert，默认建表类型
  - 主要对导入文本数据建立过渡表
- ORC表
  - 优化的列式存储，轻量级索引，压缩比高，只支持insert
  - Inceptor中数仓离线分析的主要表类型，可由Text表生成
- ORC事务表
  - 由ORC表衍生而出，支持insert、update、delete以及事务操作
  - 多版本文件存储，定期做compaction，10~20%的性能损失
- Holodesk表
  - 基于内存/SSD的分布式列式存储结构，内含索引和cube，压缩率比orc略低，只支持insert操作
  - 主要用于数据交互式分析，以及报表工具的实时展现
- Hyperbase表
  - 数据存储Hyperbase上，支持多种索引以及insert、update、delete
  - 用于高并发的索引历史数据查询，支持非结构化数据存储

Inceptor文件格式

- 文件格式决定了表文件的存储格式。有两种方法可以指定文件格式：
  - 通过STORED AS “文件格式名”， 比如STORED AS TEXTFILE或者STORED AS ORC。
  - 通过STORED AS INPUTFORMAT … OUTPUTFORMAT … 直接指定输入格式（InputFormat）和输出格式（OutputFormat）

```
STORED AS TEXTFILE

STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
```

- 行格式（Row Format）
  - 行格式决定了行和行中字段的格式。行格式由DELIMITED子句或者SERDE子句指定。
  - Inceptor的默认SerDe是Lazy Simple SerDe
  - 使用Lazy Simple SerDe时，我们需要使用DELIMITED子句来告诉Inceptor文本文件中字段、MAP、STRUCT、UINIONTYPE的分隔字符

Text表

- 默认分隔符

```
CREATE TABLE ...
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED BY '\002'
MAP KEYS TERMINATED BY '\003'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

- 多字段分界
  - 使用多字段分界可以大大减少分界字段在数据本身中出现的概率
  - 多字段分界文本的SerDe不是LazySimpleSerDe，而是MultiDelimitSerDe

```
CREATE TABLE table_name (col_name data_type, ...)
ROW FORMAT SERDE
'org.apache.hadoop.hive.contrib.serde2.MultiDelimitSerDe'
WITH SERDEPROPERTIES
('input.delimited'='delimiting_characters')
```

Holodesk表

TRANSWARP 星 环 科 技

- 创建内存表
  - Inceptor支持建立内存表，内存表中的数据会在机器运行时一直存储在内存中，所以将一些常用查询结果存储在内存表内可以大大提高计算速度
  - Inceptor提供checkpoint机制，将计算数据同步写入HDFS中，可以保证在存储了内存表的机器当机时，内存表中的数据可以从HDFS中直接读取恢复而不需要重新进行查询计算
  - 通过CTAS在建表，建表时数据即填入

```
CREATE TABLE table_name TBLPROPERTIES ("cache" = "cache_medium",
"cache.checkpoint"="true|false",
[“holodesk.index"=“column_name_index,column_name_index2"],
[“holodesk.dimension"=“column_name_dim1,column_name_dim2"])
AS SELECT select_statment
```

  - cache值为’RAM’或’SSD’，指定计算缓存的介质
  - cache.checkpoint指定是否设置checkpoint
  - holodesk.index属性为一个或多个字段创建index
  - holodesk.dimension属性为一组或多组字段创建cube

ORC事务表

- 建表
  - ORC事务表必须是分桶表
  - 表属性里需要加上TBLPROPERTIES ("transactional"="true")
  - 此外，如果表的数据量特别大，建议使用分区的ORC事务表。

```
//非分区表
CREATE TABLE table_name (column_name data_type, column_name data_type, ...)
CLUSTERED BY (column_name) INTO n BUCKETS
STORED AS ORC
TBLPROPERTIES ("transactional"="true")

//分区表
CREATE TABLE table_name (column_name data_type, column_name data_type, ...)
PARTITIONED BY (partition_key data_type)
CLUSTERED BY (column_name) INTO n BUCKETS
STORED AS ORC
TBLPROPERTIES ("transactional"="true")
```

桶的个数对事务处理的性能有关键性的影响，我们建议您设置合理的个数，一般是CPU个数的倍数，并且每个桶平均的大小控制不要超过200MB或者一百万行记录。

ORC事务表

- INSERT

```
单条插入
//非分区表
INSERT INTO table_name VALUES (value, value, ...)
//分区表
INSERT INTO table_name PARTITION (partition_key = value) VALUES (value, value, ...)


批量插入
//非分区表
INSERT INTO TABLE table_name SELECT select_statment;
//分区表
INSERT INTO TABLE table_name PARTITION (partition_key = value) select_statement;
```

ORC事务表

- UPDATE

```
单条更新
//非分区表
UPDATE table_name SET column_name = value WHERE filter_statement
//分区表
UPDATE table_name PARTITION (partition_key = value) SET (column_name = value) WHERE
filter_statement

批量更新
//非分区表
UPDATE table_name SET (column, column, ...) = (SELECT select_statement WHERE
filter_statement)
//分区表
UPDATE table_name PARTITION (partition_key = value) SET (column, column, ...) =
(SELECT select_statement WHERE filter_statement)
```

ORC事务表

- UPDATE
  - 请确保等号左右的列数相同并且对应列的数据类型也相同。
  - 在用查询往事务表中更新数据时，必须保证查询的结果的行数和表中被更新的行数一致，否则整个更新的逻辑就会有错误。
  - 因此，Inceptor要求(SELECT select_statement WHERE filter_statement)中必须要有过滤条件filter_statement, 并且这个filter_statement必须建立和被更新表的关联条件。

ORC事务表

- DELETE

```
删除部分记录
//非分区表
DELETE FROM table_name WHERE filter_statement
//分区表
DELETE FROM table_name PARTITION (partition_key = value) WHERE filter_statement

删除全部记录
//删除表中全部记录（对分区表和非分区表都适用）
DELETE FROM table_name
//从分区表中删除一个分区的全部记录
DELETE FROM table_name PARTITION (partition_key = value)
```

ORC事务表

- MERGE INTO
  - MERGE语句用来合并UPDATE和INSERT语句。
  - 通过MERGE语句，根据一张表或子查询的连接条件对另外一张表进行查询，连接条件匹配上的进行UPDATE，无法匹配的执行INSERT。
  - 这个语法仅需要一次全表扫描就完成了全部工作，执行效率要高于INSERT＋UPDATE。

```
//非分区表
MERGE INTO table
USING { table | view | subquery } alias
ON ( condition )
WHEN MATCHED THEN merge_update_clause
WHEN NOT MATCHED THEN merge_insert_clause
//分区表
MERGE INTO table
PARTITION (partition_key = value)
USING { table | view | subquery } alias
ON ( condition )
WHEN MATCHED THEN merge_update_clause
WHEN NOT MATCHED THEN merge_insert_clause
```

注意，MERGE的源表必须有化名。

事务处理

TRANSWARP 星环科技

- 模式选择
  - 默认情况下Inceptor关闭Transaction Mode
  - 要对ORC表进行事务处理，需要通过下面的开关打开ORC表对应的Transaction Mode

```
SET transaction.type = inceptor;
```

- 提交和回滚

事务处理指令为BEGIN TRANSACTION（开始事务），COMMIT（提交事务）和ROLLBACK（回滚/撤回事务）。
一次事务以BEGIN TRANSACTION开始，以COMMIT或ROLLBACK结束。

```
BEGIN TRANSACTION
sql_statements
sql_statements
...
COMMIT|ROLLBACK
```

Transactional ORC DDL

TRANSWARP 星环科技

- 指定session事务类型
  - Inceptor/Hyperbase/none
- DDL 要素
  - 选择一个“好”字段做bucket
  - 指定一个“有效”的bucket数量，参考因素：
    - CPU数量
    - Bucket内数据会排序，为了性能，每个bucket里记录行数最好不要超过300万
  - 插入数据会默认增加一个 row__id 字段，用作排序
    - Transactionid
    - Bucketid
    - rowid

```
set transaction.type=inceptor
```

```
create table acidTable (
   name varchar(10),
   age int, degree int)
clustered by (age) into 10 buckets
stored as orc
TBLProperties (“transactional”=“true”)
```

```
create table partitionAcidTable (
  name varchar(10),
  age int) partition by (city string)
clustered by (age) into 10 buckets
stored as orc
TBLProperties (“transactional”=“true”)
```

```
create table range_int_table (
    id int, value int )
partitioned by range (id) (
    partition less1 values less than (1)
    partition less10 values less than (10)
) clustered by (value) into 3 buckets
stored as orc
TBLProperties (“transactional”=“true”)
```

Compaction

- Inceptor CRUD => Multiple Version
- Version => Warehouse一个目录

```
[root@test-02 ~]# hadoop fs -ls /inceptor1/user/hive/warehouse/acidjoin1/
Found 6 items
drwxr-xr-x   - dekker hadoop          0 2015-06-03 22:37 /inceptor1/user/hive/warehouse/acidjoin1/delta_0012261_0012261
drwxr-xr-x   - dekker hadoop          0 2015-06-03 22:37 /inceptor1/user/hive/warehouse/acidjoin1/delta_0012262_0012262
drwxr-xr-x   - dekker hadoop          0 2015-06-03 22:37 /inceptor1/user/hive/warehouse/acidjoin1/delta_0012263_0012263
drwxr-xr-x   - dekker hadoop          0 2015-06-03 22:37 /inceptor1/user/hive/warehouse/acidjoin1/delta_0012264_0012264
drwxr-xr-x   - dekker hadoop          0 2015-06-03 22:37 /inceptor1/user/hive/warehouse/acidjoin1/delta_0012265_0012265
drwxr-xr-x   - dekker hadoop          0 2015-06-03 22:37 /inceptor1/user/hive/warehouse/acidjoin1/delta_0012266_0012266
```

- 查询数据时合并一条记录多个版本的数据
- 自动或者手动Compaction保证存储或者查询的高效
  - Alter table acidTable compact ‘major’
  - Alter table acidTable compact ‘minor’

```
[root@transwarp-demo2 ~]#  hadoop fs -ls /inceptor1/user/hive/warehouse/crud001.db/test_policy
drwxrwxrwx   - yarn hadoop          0 2015-08-05 12:11 /inceptor1/user/hive/warehouse/crud001.db/test_policy/base_0013418
```

- Metastore 来控制compaction
  - 有3种独立的线程，分别检测是否要发起 compaction, 执行compaction 和清理compaction
  - Compaction 是一个map reduce任务，Yarn上能够监控

Hyperbase表

- 在Inceptor Shell中处理Hyperbase表
- create

```
CREATE TABLE table_name
(row_key_column data_type,
column_name_1 data_type, column_name_2 data_type, ...)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES
("hbase.columns.mapping" = ":row_key_column, column_family:column_qualifer_1,
column_family:column_qualifier_2, ...")
TBLPROPERTIES ("hbase.table.name" ="hbase_table_name")
```

Inceptor表的第一列必须是对应Hyperbase表中的Row Key。
Inceptor表中剩余的列将是对应Hyperbase表中的所有column_family:column_qualifier组合。
hbase_table_name是对应的Hyperbase表的表名。
指定org.apache.hadoop.hive.hbase.HBaseStorageHandler表示创建hyperbase表

Hyperbase表

- 在Inceptor中对映射表进行操作
- 单条insert

```
INSERT INTO table_name (column_name1, column_name2, ...) VALUES (value1, value2, ...)
```

- UPDATE

```
UPDATE table_name SET column_name = value, column_name = value, ... WHERE filter_condition;
UPDATE table_name SET (col1, col2, col3, ...) = (SELECT col1, col2, col3 FROM [from source]);
在SET后面的字段列表及查询语句中需要包含对应的rowkey字段（如col1）
```

- DELETE

```
DELETE FROM table_name WHERE filter_condition
```

Hyperbase表

TRANSWARP 星 环 科 技

- Hyperbase优化参数

SET hyperbase.reader=true;

是否通过HFile快照读取数据。开关开启后，HDFS上会为HFile创建一个快照，读数据时不通过HFile API去Server端获取，而是直接读取快照并进行计算。

SET hyperbase.reader.autoflush=true;

是否在读取HFile快照之前时自动flush出内存数据。在hyperbase.reader设置为true后，访问数据时并不会读取内存中的数据，如果开启此开关，读取快照时之前，会先将内存中的数据flush出去。

大纲

- Inceptor架构
- Inceptor SQL
  - 基本概念
  - DDL
  - DML
- Inceptor PL/SQL
- Inceptor中表的种类
- Inceptor运维
- Inceptor优化

重要参数

- 路径参数

ngmr.fastdisk.dir
Holodesk的存储介质，默认为内存RAM，如果有SSD，需配置成相应路径
ngmr.localdir
Inceptor的shuffle目录，需要检查是否有其他非法分区；如果SSD空间较富裕，应配置为SSD的对应路径

- 资源分配策略
  - 分为按比例与按固定大小分配
  - 正常CPU为0.5（50%）
  - 内存与CPU相对应，一个CPU为1.7~2G内存
  - 集群异构严重的情况，每个节点的资源使用情况尽量一致，使用按比例分

管理页面

TRANSWARP 星 环 科 技

- inceptorserver:4040界面
  - 确认集群状态，Executor的运行状态
  - 资源配置情况
- 4040标签页
  - Stage（默认）

名称	运维用途
Active Stages	若有运行较慢的SQL，可进入此Stage查看详细信息分析
Completed Stages	可查看之前完成的SQL的stage详细信息
Failed Stages	如果出现了执行出错的SQL，进入此fail stage查看里面的详细Errors信息进行分析

  - Holodesk内存表信息查看
  - Executors

常用命令

TRANSWARP 星环科技

- jps 查看节点上进程状态

进程名	解释
NgmrServer	InceptorServer进程，仅存在于InceptorServer节点上
CoarseGrainedExecutorBackend	各Executor的进程，存在于所有Executor节点上
CliDriver	与InceptorServer连接后的客户端

- 日志位置

名称	日志位置	解释
InceptorServer日志	/var/log/inceptorsql[x]/hive-server.log	仅存于InceptorServer所在节点，记录从提交SQL到执行完成的日志信息。
Executor日志	/var/log/inceptorsql[x]/ spark-executor.log （注：旧版TDH在/usr/lib/ngmr/work下）	存于所有Executor节点上，存放task在Executor节点上执行过程中打印的日志
ExecutorGC日志	/var/log/inceptorsql[x]/ spark-executor.gc.log （注：旧版TDH在/usr/lib/ngmr/logs/下）	存于所有Executor节点上，存放GC日志。当Executor出现较严重的GC情况时，查看该日志

常用命令

- Java进程检查工具

命令	简单范例	解释
jps	jps	用以列举出当前节点上当前用户的java进程，建议以root或者sudo权限来列举
jstack	sudo –u yarn jstack 79201	当JAVA进程出现CPU瓶颈时可通过该命令搜集信息查看该进程当前正在RUNNABLE的地方。 （注：请务必在一定时间内多打几次jstack，仅靠一次jstack的日志信息不足以）
jinfo	jinfo 79201	用以查看java进程的具体环境配置，一般用来检查CLASSPATH以防依赖包版本冲突或不匹配的情况
jstat	jstat –gc 79201 1000 1000	查看java进程运行过程中的内存占用情况，可用来监控Executor的内存增长速度和GC时间
jmap	jmap –histo 79201	打印当前进程的对象分配空间

通用问题

- 语法解析报错

FAILED: Parse Error: line 1:60 missing FROM at 'table' near '<EOF>‘
请检查自己的sql语法，是否遗漏了相关关键字或者打错字了

FAILED: Error in semantic analysis: …
请先阅读该报错，对于“Table not found”或者列不存在等简单信息请自行处理，如果有其他确实难以理解的报错，请查看 InceptorServer日志，将其中的Exception的CallStack详细信息记录以及SQL一并发送给后台工程师。

通用问题

TRANSWARP 星 环 科 技

- SQL执行出错

[Hive Error]: Query returned non-zero code: 1, cause: FAILED: Execution Error, return code 1 from io.transwarp.inceptor.execution.SparkTask. …….

出现错误可从两处尝试获取报错信息：
InceptorServer日志
4040界面Stage列表的failedstage里面查看详细Error
在查看failed stage里面的fail task时，需要注意的一点是确认fail的task是否全部集中在一台或某几台机器上，如果是的话可先检查这几台机器的环境问题等因素。
其他情况具体问题具体分析

- SQL有任务一直处于RUNNING状态
  - 数据倾斜场景
  - 请执行以下SQL统计表对应的reduce key信息，将其中数据量最大的reducekey组合查找出来

```
Select key,key1,count(*) cnt from tableA group by key,key1 order by cnt desc limit 20;
```

  - 正常的生产数据，过多null值。加where条件解决
  - 正常的生产数据，某一值过多。做特殊处理，结果Union all
  - 导入数据期间由于格式转换出现错误，出现过多的null。重新清理数据

Inceptor常见故障排查及解决

TRANSWARP 星 环 科 技

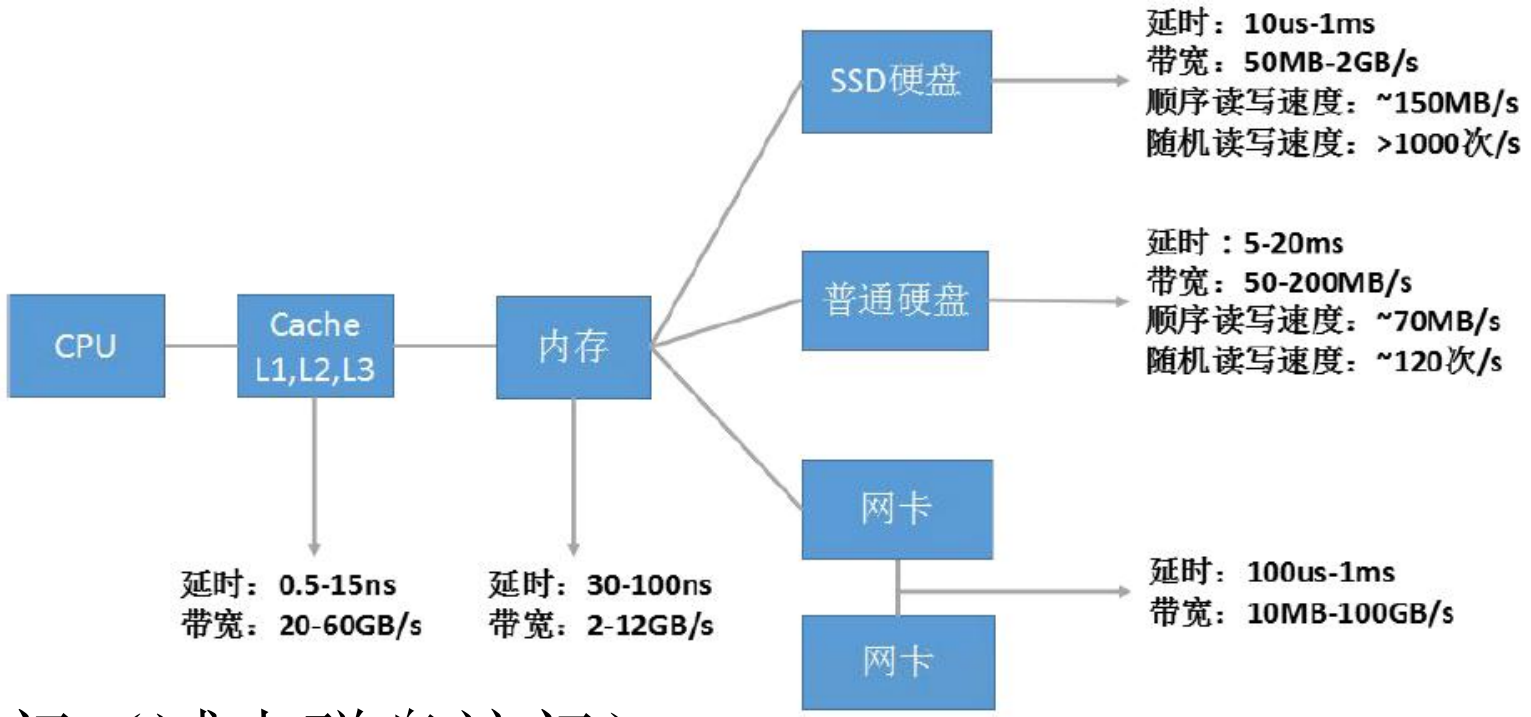
- MySQL 无法启动：
  - 多见于异常断电引起的重启失败，删除 /hadoop/mysql/mysql.sock该文件 service mysqld restart 即可解决。

- Inceptor Server Executor无法启动：
  - 参考Executor日志，通常引起Executor无法启动的原因有：某个目录没有写权限；某块磁盘损坏无法创建目录；Yarn上改节点没有可分配给Inceptor Executor使用的资源等。

大纲

- Inceptor架构
- Inceptor SQL
  - 基本概念
  - DDL
  - DML
- Inceptor PL/SQL
- Inceptor中表的种类
- Inceptor运维
- Inceptor优化

优化方法概述



- 减少数据访问（减少磁盘访问）
- 减少中间结果量（减少网络传输或磁盘访问）
- 减少交互次数（减少网络传输、减少调度开销）
- 改进算法，减少服务器CPU开销（减少CPU及内存开销）

Inceptor优化

- 减少数据访问
  - 索引是传统数据库减少数据访问、提升访问速度的常用方法
  - Inceptor没有索引，可以通过分区、分桶、各种查询Filter过滤器达到类似效果
- 返回更少的数据
  - 只选取需要的列，避免select *的情况
- 使用存储过程
  - 减少了编译次数提高了执行效率
  - 在网络交换中代替了大量的SQL语句，使网络通信减少，提升通信效率
- 减少数据库服务器CPU运算
  - 当过滤条件类型不匹配时，每次运算需要对操作数进行转换
  - 建表时尽可能把字段安排成相同的数据类型
  - 对于Like的模糊查询，尽量使用In-List的方式实现

SQL分类及优化策略

- 大表与大表的普通Join
  - 关注是否Shuffle过大导致磁盘溢出，增加reduce数目解决
  - 尽量避免大表直接与大表join，在条件允许的情况下先做与小表有关的JOIN
  - 通过bucket join来解决
- 大表与小表的Join
  - 大表与小表Join时，需采用MapJoin
  - 过滤后小表的行数不能太大，限制在20万条记录以内
- Shuffle特别多
  - 重复的数据特别多，会导致Shuffle量大
  - 保证查询任务语义可实现的情况下用Group By去重

SQL分类及优化策略

TRANSWARP 星 环 科 技

- Map Task数量多执行时间短
  - Map Task和数据块的数量与大小是相关联的
  - Map过多且执行时间多说明有大量小文件
  - 可以通过打开自动合并参数减少Map数量

```
//打开自动合并开关
SET ngmr.partition.automerge = TRUE;
//merge n个任务合并成一个，默认值是3
SET ngmr.partition.mergesize = n;
//merge后的单个任务数据量不能超过m,默认值是8MBytes，-1为忽略限制
SET ngmr.partition.mergesize.mb = m;

选取automerge参数时，设计下限时，保证单个Task的处理时间不要低于2s，上限的时候，每个Task的GC时间占比例不要超过20%
```

SQL分类及优化策略

TRANSWARP 星 环 科 技

- Reduce Task数量多执行时间短
  - Reduce Task数量很多
  - Shuffle Read很少或者每个Task执行时间很短
  - 减少Reduce Task的数量

```
//N为相应的任务数量
SET mapred.reduce.tasks = N;
```

SQL分类及优化策略

- 使用临时表优化的场景
  - 通过创建临重用该逻辑产生的中间结果集
  - 通过创建临时表理清顺序并明确优化手段
  - 临时表放在临时目录下，自动释放

```
CREATE TEMPORARY TABLE temp_table_name AS SELECT …;
```

如果SQL有WITH-AS短语，将该部分内创建成临时表

不涉及两张表之间关联的查询称为非关联子查询
如果语句中有非关联子查询，提取子查询中内容创建临时表

SQL分类及优化策略

- Map Join
  - 如果两张被连接的表中有一张比较小（100MB以下），那么可以通过MAP JOIN来提高执行速度。
  - MAP JOIN会将小表放入内存中，在map阶段直接拿另一张表的数据和内存中表数据做匹配，由于省去了shuffle，速度会比较快。

```
SELECT /*+ MAPJOIN(b) */ select_expression, select_expression, ...
FROM table_reference JOIN table_reference ON join_condition
```

  - Inceptor已经有了自动MAP JOIN的功能，就是在有一张表在100MB一下时，Inceptor会自动执行MAP JOIN。

CBO优化

- Cost Based Optimization(基于代价的优化)
  - CBO优化框架通过对执行计划进行等价变化，估算出量化的计划代价，最终选择代价最小的作为最终的执行计划
  - CBO能够自动对执行计划进行评估和优化，不需要手工干预业务逻辑
  - Inceptor的CBO主要用于多表JOIN案例的执行计划优化

```
//Step1：analyze表，收集统计信息
ANALYZE TABLE table_name PARTITION(column_partition) COMPUTE STATISTICS;
ANALYZE TABLE table_name PARTITION(column_partition) COMPUTE STATISTICS
FOR COLUMNS;

//Step2:打开CBO开关
SET hive.cbo.enable = TRUE/FALSE;
```

SQL迁移及优化策略

TRANSWARP 星 环 科 技

- 整体思路
  - Step1：先解决语法不兼容问题
  - Step2：选择若干有典型代表功能的SQL测试，不要一次性执行所有语句
  - Step3：查看4040端口页面，观察是否有异常
    - Task数目十分庞大
    - Stage运行速度很慢
  - 针对以上发现，进行性能优化
    - 查看8180，观察系统资源使用情况
    - 分析数据，统计行数以及分布情况
    - 分布执行计划，是否需要Join优化
    - ……

TRANSWARP
星 环 科 技

Q&A