# 一 数据仓库

## 1.数据仓库分层

### 1.2 主题层

FDM主题层:  
团体

事件

协议

产品

财务

渠道

资产

公共

汇总

报表

## 2.遇到的困难

## 3.平时工作中如优化重复性的工作

## 4.给一个需求场景 简述如何设计开发

# 二 大数据

## 1. 星环大数据

ORC 表：ORC的全称是(Optimized Record Columnar)

创建普通临时表：

create table if not exists test\_orc\_tmp(

  name string,

  gender string,

  cnt BIGINT

)row format delimited fields terminated by '|' stored as textfile;

创建ORC表：

drop table test\_orc;

create table if not exists test\_orc(

  name string,

  gender string,

  cnt BIGINT

)STORED AS ORC;

创建测试数据  vi orc.txt

goldenkey0|male|111

goldenkey1|male|112

goldenkey2|male|113

goldenkey3|male|114

goldenkey4|male|115

goldenkey5|male|116

goldenkey6|male|117

将测试数据导入临时普通表：

LOAD DATA INPATH '/tmp/orc.txt' OVERWRITE INTO TABLE test\_orc\_tmp;

将临时普通表的数据插入到ORC表：

INSERT INTO TABLE test\_orc SELECT \* FROM test\_orc\_tmp;

查看ORC表是否有数据：

select \* from test\_orc\_tmp;

ORC知识链接：

<https://www.jianshu.com/p/36e4f0137744>

Inceptor命令行快速入门

任务：在本地创建文本，导入到Inceptor新建的表格中。

1. 登录安装有Inceptor Server的节点，新建一个文本文件quickstart.txt,并将其放在tmp文件夹中。

vim /tmp/quickstart.txt

2. 向quickstart.txt中写入两行文字，如下图所示。



输入后按Esc键，输入:wq保存退出。

3. 命令行连接Inceptor

• InceptorServer 1

transwarp -t -h localhost

• InceptorServer 2 简单认证

beeline -u "jdbc:hive2://localhost:10000/default"

• InceptorServer 2 LDAP认证

beeline -u "jdbc:hive2://localhost:10000/default" -n <username> -p <password>

• InceptorServer 2 Kerberos认证

kinit <your\_principal>

beeline -u "jdbc:hive2://localhost:10000/default;principal=hive/node1@TDH"

4. 新建外表quickstart

CREATE EXTERNAL TABLE quickstart (a INT, b STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY

'|';

5. 将位于本地的quickstart.txt导入至表格quickstart中

LOAD DATA LOCAL INPATH '/tmp/quickstart.txt' INTO TABLE quickstart;

注意如果Inceptor使用LDAP或者Kerberos认证，您需要是这个文件的owner。

6. 查看表格内容，结果为：



3.3.2.1.1. 直接建表

语法：

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] ①

[<database\_name>.]<table\_name>　 ②

[(<column\_name> <data\_type> [COMMENT '<column\_comment>'] ③

[, <column\_name> <data\_type> ...])]

[COMMENT '<table\_comment>'] ③

[PARTITIONED BY (<part\_key> <data\_type> [COMMENT '<partition\_comment>'] ③

[, <part\_key> <data\_type>...])] ④

[CLUSTERED BY (<col\_name> [, <col\_name>...]) ⑤

[SORTED BY (<col\_name> [ASC|DESC] [, <col\_name> [ASC|DESC]...])] ⑥

INTO <num\_buckets> BUCKETS]

[

[ROW FORMAT <row\_format>] ⑦

[STORED AS (TEXTFILE|ORC|CSVFILE)] ⑧

| STORED BY '<storage.handler.class.name>' [WITH SERDEPROPERTIES (<...>)] ⑨

]

[LOCATION '<hdfs\_path>'] ⑩

[TBLPROPERTIES ('<property\_name>'='<property\_value>', ...)]　 ⑪

① TEMPORARY 为临时表选项，EXTERNAL 为外表选项。

② 加上 <database\_name>. 则将表建在指定的数据库中。

③ 可以为列、表和分区用 COMMENT 加注释，注意注释要放在单引号中。

④ 指定分区键，具体使用方式请参考“分区表”。

⑤ 分桶子句，具体使用方式请参考“分桶表”。

⑥ 桶内排序选项，ASC 为升序，DESC 为降序。

⑦ 行格式，在建TEXT表时使用，具体细节请参考“TEXT表”部

⑧ 指定文件格式，该选项在建TEXT表，ORC表或CSV表时使用。

⑨ 指定使用的storage handler。

⑩ 指向HDFS上的一个目录。这个选项我们推荐只在建外表时使用，也就是和 EXTERNAL 选项合用（虽然

也可以在建内表时使用，但是我们　不建议 这样做）。 该路径必须是绝对路径，并且执行操作的用户必

须是这个路径指向的目录或文件的owner。如果 <hdfs\_path> 指向的目录不存在，Inceptor会尝试新建这

个目录，但是安全模式下Inceptor可能没有在指定路径新建目录的权限，所以星环科技建议尽量避免让

<hdfs\_path> 指向不存在的目录。

⑪ 表属性，由键值对表示，在建Holodesk表，ORC事务表和CSV表时使用，具体细节请参考“ORC表”，

“Holodesk表”和“CSV表”。

6.14. 窗口函数

窗口函数是SQL中一类特别的函数。和聚合函数相似，窗口函数的输入也是多行记录。不同的是，聚合函数的作用于由GROUP BY子句聚合的组，而窗口函数则作用于一个窗口，这里，窗口是由一个 OVER子句 定义的多行记录。聚合函数对其所作用的每一组记录输出一条结果，而窗口函数对其所作用的窗口中的每一行记录输出一条结果。一些聚合函数，如sum, max, min, avg,count等也可以当作窗口函数使用

3.8. 数据稽查

基本介绍

用户利用INSERT或者LOAD导入数据时有时会无意将脏数据录入业务表，查询语句在访问这些脏表时，可能会

运行失败或者返回错误结果。为避免此情况，Inceptor提供数据稽查功能。具体实施时，数据稽核会根据规

则将脏数据写入指定的脏数据表（Error Table），并标明每一条脏数据为何非法，在数据导入完成后，将返

回总共记录数、导入记录数的接口、或者数据质量报告，以方便监控程序判断以及处理。也可以仅打印出显

示报错信息。实现上述特性都是为了使数据稽查能够在脏数据存在的情况下尽可能的保护系统或保证业务的

顺畅执行。

进行数据稽查时，系统将对如下问题数据报错并记录至Error Table：

1. 字段值中含有定界标识符，导致读取数据时一行数据被误读为两行。因为错误的切分方式会使读入记录

的字段数与定义不符，系统将通过检测列数总个数来识别这样的脏数据。

2. 以目标结果表的类型判断为标准，进行类型匹配与类型转换，如果类型不匹配，则输出脏数据。

3. 在通过UDF结合过滤条件，实现其他的数据转换以及过滤时，对不匹配的数据记录打印报错或记录

于Error Table。

4. 对不符合NOT NULL限制的记录报错。

完整的数据稽查功能是按照如下的处理流程实现的，建议用户在使用数据稽查功能以及设置相关配置时，结

合该流程决定配置参数：

1. 用户可以在创建一个外表的同时指定Log Error Table。

2. 当从外表读取数据的时候，每解析一行记录，若访问到上述四种无效数据，就将该数据写入Error

Table。

3. 允许指定REJECT策略，即当错误率达到一定的行数或者比例时，就停止读取。

• 目前数据稽查功能只对外表开放。

• 对于SELECT或过滤操作，可将脏数据写入Error Table；但是无法录入GROUP BY或者

复杂SQL中的问题数据，目前仅仅将报错忽略掉。

• 聚合函数遇到脏数据时不报错而是直接跳过脏数据去计算合法数据。如： 对于

“SELECT count(column\_name) FROM table\_name”，在数据稽查的保护下，如

果table\_name有非法数据，返回结果会少于实际行数，但是不会出现相关报错信息。

• 不论是否进行稽查，count(\*)和count(1)在任何情况下都会输出满足条件的包括脏数

据的总行数。

3.8.3. 相关语法

指定Error Table

创建外表时指定Log Error Table。

CREATE EXTERNAL TABLE table\_name (column1 datatype1, column2 datatype2, ...)

LOG ERRORS INTO error\_table\_name [OVERWRITE]

[SEGMENT REJECT LIMIT n [ ROWS | Percent ] ]

• error\_table\_name是存放当前表脏数据信息的Error Table的名称，如果不存在系统会自动创建一个。只

能在创建表时指定。注意关键字LOG ERRORS INTO。

• 若允许Overwrite Error Table，需在相应位置写“OVERWRITE”，否则忽略。禁止Overwrite的坏处是每

次都会在原有记录的基础上写入脏数据信息，使该Error Table不断扩增；好处是能追溯至更久远的信息。用户应根据需求设置。

• 若需执行REJECT策略，应补充“SEGMENT REJECT LIMIT n…”部分，如果没有需求就忽略。n是REJECT阈

值，表示REJECT之前允许的非法数据的行数或比例。 应注意 ，分布式结构下，由于语句的执行被切分

交给不同task实现，所以n是相对于一个task中的数据行数而言的，而并非总数据行数。

• 没有指定Error Table的表也可以受到数据稽查的保护，只是错误信息不会收录于Error Table。

• 指定了Error Table并不代表启动了数据稽查，需要通过后面将介绍的开关控制。

例 36. 创建外表指定关联的Error Table

• 创建employee\_err表，导入employee.txt中的记录（包含脏数据）。并指定Error Table

为employee\_error\_table，允许Overwrite，采用LIMIT=2的Reject策略，即访问脏数据大于两行时

就停止执行。

CREATE EXTERNAL TABLE employee\_err (

id int NOT NULL,

name string,

age tinyint,

degree string,

onboard int

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION '/DataAudit/employee'

LOG ERRORS INTO employee\_error\_table

SEGMENT REJECT LIMIT 2 ROWS;

SELECT \* FROM employee\_err;

+-------+--------+-------+---------+----------+

| id | name | age | degree | onboard |

+-------+--------+-------+---------+----------+

| 1 | "YH1" | NULL | PG | 2013 |

| 2 | "LC1" | 29 | BC | 2013 |

| 3 | "KY1" | 22 | PG | 2013 |

| 4 | "MH1" | 23 | PG | 2013 |

| 5 | "LX1" | 32 | PG | 2014 |

| 1 | "YH2" | 36 | PG | 2013 |

| 2 | "LC3" | 29 | BC | 2013 |

| 3 | "KY3" | 22 | PG | 2013 |

| 4 | "MH3" | 23 | PG | 2013 |

| 5 | "LX3" | NULL | PG | 2014 |

| NULL | "AM5" | 36 | FG | 2014 |

+-------+--------+-------+---------+----------+

控制开关

下面是和数据稽查相关的三个开关，用于控制其工作特性：

SET inceptor.data.audit = true/false; ①

SET inceptor.strict.evaluate = true/false; ②

SET inceptor.notnull.audit = true/false; ③

① 数据稽查总开关，开启时会做脏数据和NOT NULL检查，默认关闭。

② 是否在遇到脏数据时报Exception，默认关闭。

③ 是否对NOT NULL Constraint进行检查，默认关闭。



• inceptor.data.audit是Top Level开关，设为true后，后面两个开关强制为true，启

动inceptor.data.audit之后再对两个子开关进行设置将不起作用。

• 后两个开关可以在inceptor.data.audit关闭时做设置。区别在于，如果开

启data.audit，关于脏数据和NOT NULL限制的报错都会写入Error Table；若在关

闭data.audit后启动strict.evaluate或notnull.audit，执行对应检测时报错将被打

印于界面。

3.9. TEXT表

TEXT表是文本格式的表，是Inceptor默认的表格式。在数据量大的情况下，TEXT表的统计和查询性能都比较

低；TEXT表也不支持事务处理，所以通常用于将文本文件中的原始数据导入Inceptor中。针对不同的使用场

景，用户可以将其中的数据放入ORC表或Holodesk表中。

Inceptor提供两种方式将文本文件中的数据导入TEXT表中：

• 建外部TEXT表，让该表指向HDFS上的一个目录，Inceptor会将目录下文件中的数据都导入该表。星环科技推荐使用这个方式导数据。

• 建TEXT表（外表内表皆可）后将本地或者HDFS上的一个文件或者一个目录下的数据 LOAD 进该表。这

种方式在安全模式下需要多重认证设置，极易出错，星环科技 不推荐 使用这个方式导数据。

3.9.1. 建TEXT表

由于TEXT表通常用于将文本文件中的原始数据导入Inceptor中，最常见的建表方式是通过直接定义列建表。CREATE TABLE LIKE 和 CREATE TABLE AS SELECT 也可以用于建TEXT表，用法简单明了，请参考

“CREATE/DROP/ALTER TABLE”章节，这里不赘述。

语法：通过定义列建表

CREATE [TEMPORARY] [EXTERNAL] TABLE <table\_name> ①

(<column\_name> <data\_type>, <column\_name> <data\_type>, ...)

[PARTITIONED BY ...] ②

[CLUSTERED BY ...] ③

[ROW FORMAT ...] ④

[STORED AS TEXTFILE] ⑤

[LOCATION '<hdfs\_path>'] ⑥

[TBLPROPERTIES ('<property\_name>'='<property\_value>', ...)]; ⑦

① [TEMPORARY] 为临时表选项，[EXTERNAL] 为外表选项。

② 分区选项，TEXT表可以分区，本章不详细讨论如何分区，请参考“分区表”章节。

③ 分桶选项，TEXT表可以分桶，本章不详细讨论如何分桶，请参考“分桶表”章节。

④ 指定字段分隔符，如果不指定将使用Inceptor的默认分隔符，具体细节请参考分隔符部分。

⑤ 指定表存储为TEXTFILE（文本文件）。因为TEXTFILE是Inceptor的默认存储格式，该选项可省略。

⑥ 指向HDFS上的一个目录。这个选项我们推荐只在建外表时使用，也就是和 EXTERNAL 选项合用（虽然

也可以在建内表时使用，但是我们　不建议 这样做）。 该路径可以是一个绝对路径，比如

/user/alice/employee 或者一个完整的URL：hdfs://nameservice1/user/alice/employee。执行该建表操作的用

户必须是这个路径指向的目录或文件的owner。如果 <hdfs\_path> 指向的目录不存在，Inceptor会尝试新

建这个目录，但是安全模式下Inceptor可能没有在指定路径新建目录的权限，所以星环科技建议尽量避免

让 <hdfs\_path> 指向不存在的目录。

⑦ 表属性，由键值对表示。

例 45. 建外表并指定其在HDFS上的路径

CREATE EXTERNAL TABLE employee\_id (id INT) LOCATION '/user/alice/employee\_id';

注意事项

• 执行该操作的用户必须是指定的HDFS目录的owner。

• 表的行格式（由 ROW FORMAT 指定）和文件格式（由 STORED AS 指定）须和HDFS目录下的

数据一致，保证Inceptor能够正常解析。

3.9.2.1. 单字符分隔符

单字符分隔符使用 ROW FORMAT DELIMITED 指定，语法如下。分隔符的指定可以使用字符本身或者字

符的八进制ASCII编码，例如“\001”等。

语法

ROW FORMAT DELIMITED

[FIELDS TERMINATED BY '<column\_delimiter>' [ESCAPED BY '<char>']] ①

[COLLECTION ITEMS TERMINATED BY '<complex\_type\_delimiter>'] ②

[MAP KEYS TERMINATED BY '<kv\_delimiter>'] ③

[LINES TERMINATED BY '<newline\_char>'] ④

① <column\_delimiter> 为列分隔符，默认值为“\001”。如果文本中的实际数据包含指定的列分隔符，可以

用 ESCAPED BY 指定转义符，将列分隔符和实际数据进行区别，转义符无默认值。

② <complex\_type\_delimiter> 为复杂数据类型（ARRAY/MAP/STRUCT）中的字段分隔符，默认值为'\002'。

③ <kv\_delimiter> 为\*MAP\* 类型中将键和值分隔的分隔符，默认值为'\003'。

④ <newline\_char> 为换行符，默认值为'\n'。

例 46. 自定义列分隔符

已知有一份原始数据在HDFS上的　/user/alice/employee 目录下，各列由“,”分开，如下：

1,Alice

2,Bob

以这份数据为源建表，我们需要将“,”指定为列分隔符：

CREATE EXTERNAL TABLE employee (id INT, name STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY

',' LOCATION '/user/alice/employee';

建表完成后查看表中内容可得：

SELECT \* FROM employee;

+*-----+--------+*

| id | name |

+*-----+--------+*

| 1 | Alice |

| 2 | Bob |

+*-----+--------+*

3.10. CSV表

CSV表的数据来源是CSV文件。CSV文件是纯文本文件，文件中包含数据以及分隔符。和TEXT表相似，CSV表最常见的使用场景是用于建外表，将CSV文件中的数据导入Inceptor，星环科技 不建议在任何计算场景中使用CSV表。计算时，应该总是将CSV表中的数据用INSERT … SELECT 语句插入ORC或者Holodesk表。建CSV表时，在建表语句中用 STORED AS CSVFILE 指定存储格式为CSV文件。另外，还可以在TBLPROPERTIES 中自定义分隔符和NULL字符。下面我们进行详细的介绍。

3.11. ORC表

ORC非事务表的建表只需在建表语句中用 STORED AS ORC 指定存储格式为ORC即可。ORC事务表的建表则需要几个额外的重点步骤：

• 为表分桶：为了保证增删改过程中的性能，我们要求ORC事务表必须是部分排序或者全局排序的，但是全局排序又过于耗费计算资源，因此我们要求ORC表必须是分桶表。

• 在 TBLPROPERTIES 里需要加上 "transactional"="true"，以标识这是一个要用作事务操作的表。

• 如果表的数据量特别大，**建议在分桶的基础上再分区，ORC事务表支持单值分区和范围分区。**

ORC事务表相对与Inceptor中的其他表支持更多CRUD（增删改）语法，包括：

• INSERT INTO … VALUES

• UPDATE

• DELETE

• MERGE INTO

本章将重点介绍　ORC事务表 的建表、导数据以及CRUD。

3.11.2. 建ORC事务表

语法

//非分区表

CREATE TABLE <table\_name> (<column> <data\_type>, <column> <data\_type>, ...)

CLUSTERED BY (<bucket\_key>) INTO <n> BUCKETS

STORED AS ORC

TBLPROPERTIES ("transactional"="true");

//单值分区表（Unique Value Partition）

CREATE TABLE <table\_name> (<column> <data\_type>, <column> <data\_type>, ...)

PARTITIONED BY (<partition\_key> <data\_type>)

CLUSTERED BY (<bucket\_key>) INTO <n> BUCKETS

STORED AS ORC

TBLPROPERTIES ("transactional"="true");

//范围分区表（Range Partition）

CREATE TABLE <table\_name> (<column> <data\_type>, <column> <data\_type>, ...)

PARTITIONED BY RANGE(<partition\_key1> <data\_type>, <partition\_key2> <data\_type>, ...) (

PARTITION [<partition\_name\_1>] VALUE LESS THAN(<key1\_bound\_value1>, <key2\_bound\_value1>, ...),

PARTITION [partition\_name\_2] VALUE LESS THAN(key1\_bound\_value2, key2\_bound\_value2, ...),

...

)

CLUSTERED BY (<bucket\_key>) INTO <n> BUCKETS

STORED AS ORC

TBLPROPERTIES ("transactional"="true");

1. 分区表中的分区键partition\_key不能和表中的列重复。

2. 在表结构中，分区键被系统排在非分区键之后。

关于更多单值分区和范围分区的内容，请参考分区表。

举例

[$host] SET transaction.type=inceptor;

[$host] CREATE TABLE ta (name STRING, age INT) CLUSTERED BY (age) INTO 2 BUCKETS STORED AS ORC

TBLPROPERTIES ("transactional"="true"); ①

[$host] CREATE TABLE tg (name STRING, gpa DOUBLE) CLUSTERED BY (gpa) INTO 4 BUCKETS STORED AS ORC

TBLPROPERTIES ("transactional"="true"); ②

[$host] CREATE TABLE test (a INT, b STRING, c DOUBLE) PARTITIONED BY (date STRING) CLUSTERED BY (a)

INTO 8 BUCKETS STORED AS ORC TBLPROPERTIES ("transactional"="true"); ③

[$host] create table t1(value int) partitioned by range(id int)

(

partition less1 values less than (1),

partition less10 values less than (10),

partition less100 values less than (100)

)

clustered by (value) into 5 buckets stored as orc TBLPROPERTIES ("transactional"="true"); ④

① 创建非分区ORC表。

② 创建非分区ORC表。

③ 创建单值分区ORC表。

④ 创建范围分区ORC表。

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。

3.11.3. ORC事务表的CRUD

ORC事务表比其他表支持更多CRUD（增删改）语法，包括：

• INSERT INTO … VALUES

• UPDATE

• DELETE

• MERGE INTO

3.12. Holodesk表

Holodesk对满足以下特征的场景表现出了极强的处理能力，极力建议对这些场景创建Holodesk表：

1. 当机器拥有很大的内存或者部署了SSD时。

2. 过滤高的场景，包括单表扫描和多表MapJoin等。

3. 聚合率高的场景，例如GROUP BY之后，信息被大量聚合。

3.13. 基于定宽文本文件建外表

3.14. 分区表

Inceptor中只支持对TEXT表、ORC表和CSV表分区，不支持对Holodesk表分区。

在逻辑上，分区表和未分区表没有区别；在物理上，分区表中的数据按分区键的值放在HDFS上表目录下的对应子目录中，一个分区对应一个子目录。例如一张表user\_acc\_level按acc\_level分区：

3.14.1. 单值分区表

3.14.2. 范围分区表

本

## 2.hadoop

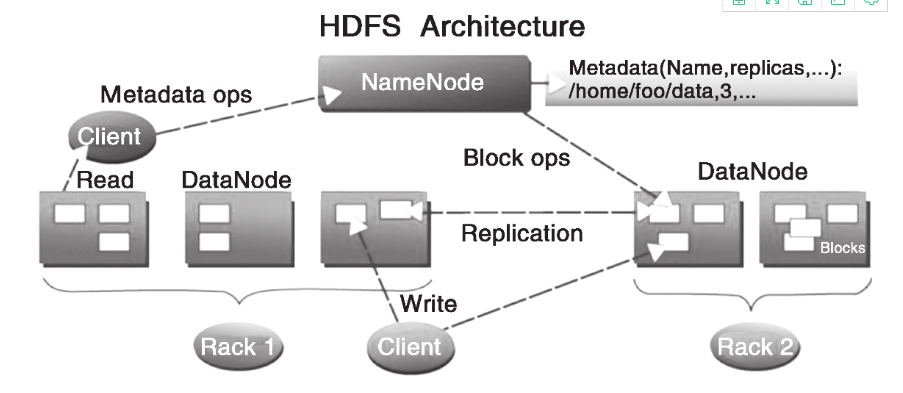
这篇文章讲的比较好：

<https://www.cnblogs.com/binarylei/p/9028375.html>

**二、Hadoop 存储 – HDFS**

Hadoop 的存储系统是 HDFS(Hadoop Distributed File System)分布式文件系统，对外部客户端而言，HDFS 就像一个传统的分级文件系统，可以进行创建、删除、移动或重命名文件或文件夹等操作，与 Linux 文件系统类似。

但是，Hadoop HDFS 的架构是基于一组特定的节点构建的(见图s)，这些节称节点(NameNode，仅一个)，它在 HDFS 内部提供元数据服务；第二名称节点(Secondary NameNode)，名称节点的帮助节点，主要是为了整合元数据操作(注意不是名称节点的备份)；数据节点(DataNode)，它为 HDFS 提供存储块。由于仅有一个 NameNode，因此这是 HDFS 的一个缺点(单点失败，在 Hadoop2.x 后有较大改善)。



存储在 HDFS 中的文件被分成块，然后这些块被复制到多个数据节点中(DataNode)，这与传统的 RAID 架构大不相同。块的大小(通常为 128M)和复制的块数量在创建文件时由客户机决定。名称节点可以控制所有文件操作。HDFS 内部的所有通信都基于标准的 TCP/IP 协议。

**（1）名称节点（NameNode）**

它是一个通常在HDFS架构中单独机器上运行的组件，负责管理文件系统名称空间和控制外部客户机的访问。NameNode决定是否将文件映射到DataNode上的复制块上。对于最常见的3个复制块，第一个复制块存储在同一机架的不同节点上，最后一个复制块存储在不同机架的某个节点上。

**（2）数据节点（DataNode）**

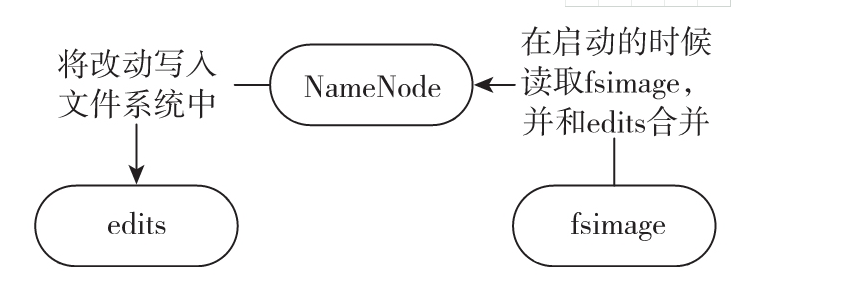
数据节点也是一个通常在HDFS架构中的单独机器上运行的组件。Hadoop集群包含一个NameNode和大量DataNode。数据节点通常以机架的形式组织，机架通过一个交换机将所有系统连接起来。

数据节点响应来自HDFS客户机的读写请求。它们还响应来自NameNode的创建、删除和复制块的命令。名称节点依赖来自每个数据节点的定期心跳（heartbeat）消息。每条消息都包含一个块报告，名称节点可以根据这个报告验证块映射和其他文件系统元数据。如果数据节点不能发送心跳消息，名称节点将采取修复措施，重新复制在该节点上丢失的块。

**（3）第二名称节点（Secondary NameNode）**

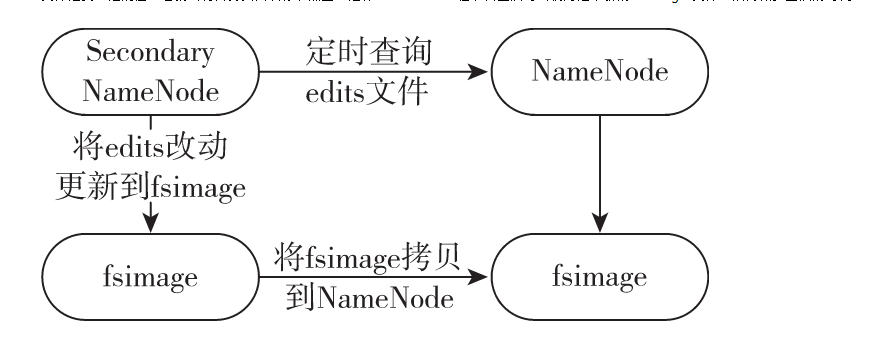
第二名称节点的作用在于为HDFS中的名称节点提供一个Checkpoint，它只是名称节点的一个助手节点，这也是它在社区内被认为是Checkpoint Node的原因。

如图 1-3 所示，只有在NameNode重启时，edits才会合并到fsimage文件中，从而得到一个文件系统的最新快照。但是在生产环境集群中的NameNode是很少重启的，这意味着当NameNode运行很长时间后，edits文件会变得很大。而当NameNode宕机时，edits就会丢失很多改动，如何解决这个问题呢？

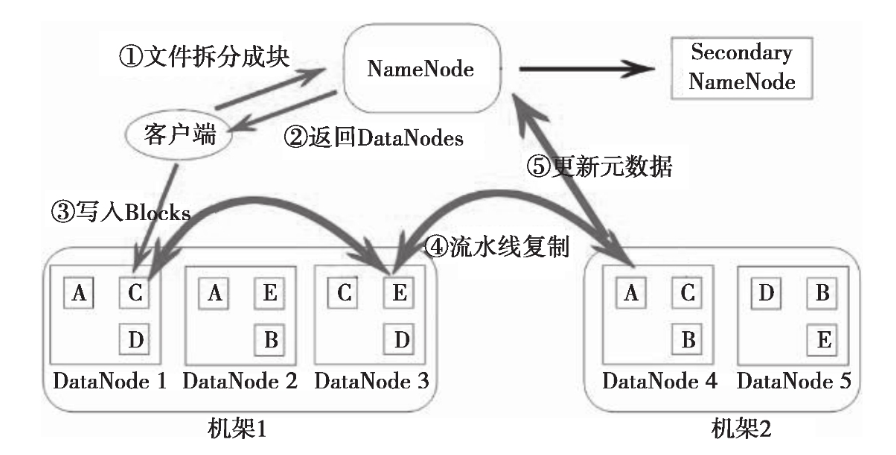


fsimage 是 NameNode 启动时对整个文件系统的快照；edits 是在 NameNode 启动后对文件系统的改动序列。

如图 1-4 所示，Secondary NameNode 会定时到 NameNode 去获取名称节点的 edits，并及时更新到自己 fsimage 上。这样，如果 NameNode 宕机，我们也可以使用 Secondary-NameNode 的信息来恢复 NameNode。并且，如果 Secondary NameNode 新的 fsimage 文件达到一定阈值，它就会将其拷贝回名称节点上，这样 NameNode 在下次重启时会使用这个新的 fsimage 文件，从而减少重启的时间。



举个数据上传的例子来深入理解下HDFS内部是怎么做的，如图 1-5 所示。



文件在客户端时会被分块，这里可以看到文件被分为 5 个块，分别是：A、B、C、D、E。同时为了负载均衡，所以每个节点有 3 个块。下面来看看具体步骤：

1. 客户端将要上传的文件按 128M 的大小分块。
2. 客户端向名称节点发送写数据请求。
3. 名称节点记录各个 DataNode 信息，并返回可用的 DataNode 列表。
4. 客户端直接向 DataNode 发送分割后的文件块，发送过程以流式写入。
5. 写入完成后，DataNode 向 NameNode 发送消息，更新元数据。

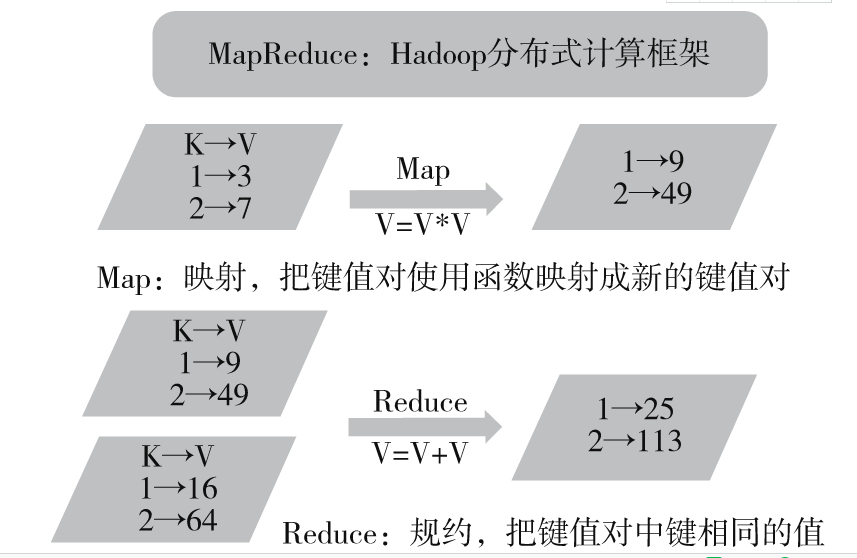
这里需要注意：

1. 写 1T 文件，需要 3T 的存储，3T 的网络流量。
2. 在执行读或写的过程中，NameNode 和 DataNode 通过 HeartBeat 进行保存通信，确定 DataNode 活着。如果发现 DataNode 死掉了，就将死掉的 DataNode 上的数据，放到其他节点去，读取时，读其他节点。
3. 宕掉一个节点没关系，还有其他节点可以备份；甚至，宕掉某一个机架也没关系；其他机架上也有备份。

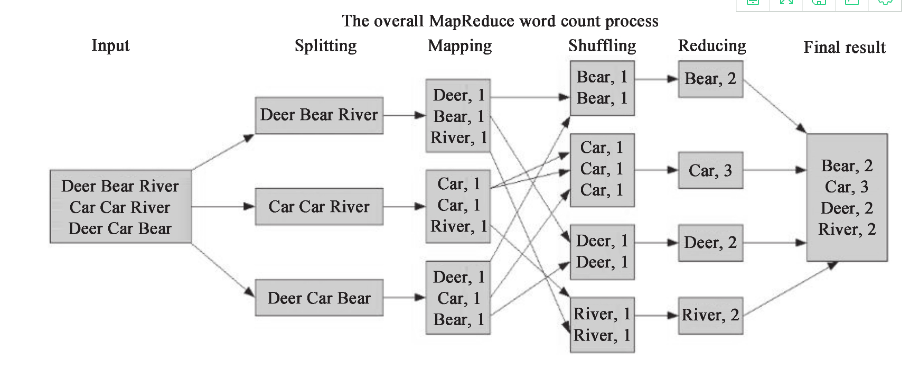
**三、Hadoop 计算 — MapReduce**

MapReduce 是 Google 提出的一个软件架构，用于大规模数据集（大于1TB）的并行运算。概念“Map（映射）”和“Reduce（归纳）”以及它们的主要思想，都是从函数式编程语言借来的，还有从矢量编程语言借来的特性。

当前的软件实现是指定一个 Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的 Reduce（归纳）函数，用来保证所有映射的键值对中的每一个共享相同的键组，如图 1-6 所示。

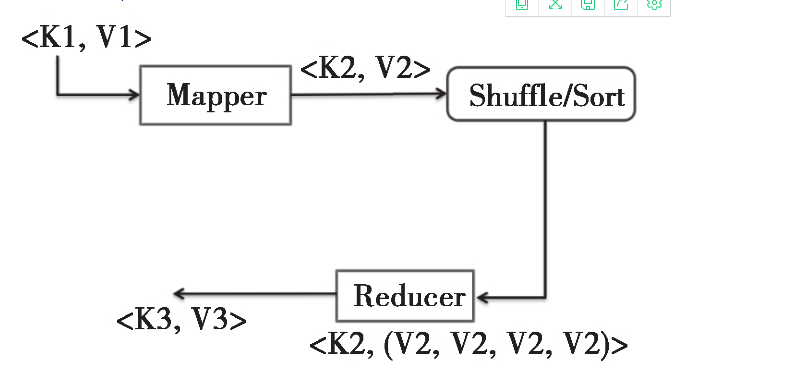


下面将以 Hadoop 的“Hello World”例程—单词计数来分析MapReduce的逻辑，如图 1-7 所示。一般的 MapReduce 程序会经过以下几个过程：输入（Input）、输入分片（Splitting）、Map阶段、Shuffle阶段、Reduce阶段、输出（Final result）。



1. 输入就不用说了，数据一般放在 HDFS 上面就可以了，而且文件是被分块的。关于文件块和文件分片的关系，在输入分片中说明。
2. 输入分片：在进行 Map 阶段之前，MapReduce 框架会根据输入文件计算输入分片（split），每个输入分片会对应一个 Map 任务，输入分片往往和 HDFS 的块关系很密切。例如，HDFS 的块的大小是 128M，如果我们输入两个文件，大小分别是 27M、129M，那么 27M 的文件会作为一个输入分片（不足 128M 会被当作一个分片），而 129MB 则是两个输入分片（129-128＝1，不足 128M，所以 1M 也会被当作一个输入分片），所以，一般来说，一个文件块会对应一个分片。如图 1-7 所示，Splitting 对应下面的三个数据应该理解为三个分片。
3. Map 阶段：这个阶段的处理逻辑其实就是程序员编写好的 Map 函数，因为一个分片对应一个 Map 任务，并且是对应一个文件块，所以这里其实是数据本地化的操作，也就是所谓的移动计算而不是移动数据。如图 1-7 所示，这里的操作其实就是把每句话进行分割，然后得到每个单词，再对每个单词进行映射，得到单词和1的键值对。
4. Shuffle 阶段：这是“奇迹”发生的地方，MapReduce 的核心其实就是 Shuffle。那么 Shuffle 的原理呢？Shuffle 就是将 Map 的输出进行整合，然后作为 Reduce 的输入发送给 Reduce。简单理解就是把所有 Map 的输出按照键进行排序，并且把相对键的键值对整合到同一个组中。如图 1-7 所示，Bear、Car、Deer、River 是排序的，并且 Bear 这个键有两个键值对。
5. Reduce 阶段：与 Map 类似，这里也是用户编写程序的地方，可以针对分组后的键值对进行处理。如图 1-7 所示，针对同一个键 Bear 的所有值进行了一个加法操作，得到 <Bear，2> 这样的键值对。
6. 输出：Reduce 的输出直接写入 HDFS 上，同样这个输出文件也是分块的。

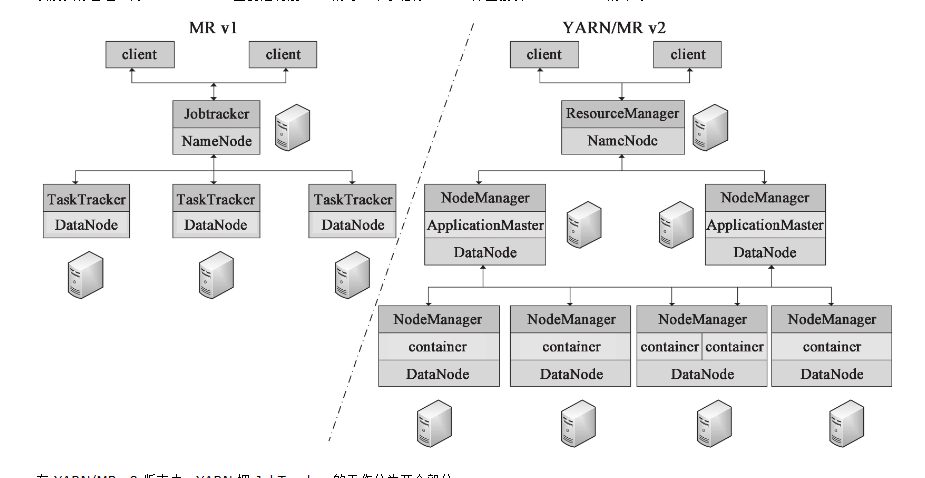
说了这么多，其实 MapReduce 的本质用一张图可以完整地表现出来，如图 1-8 所示。



MapReduce 的本质就是把一组键值对 <K1，V1> 经过 Map 阶段映射成新的键值对 <K2，V2>；接着经过 Shuffle/Sort 阶段进行排序和“洗牌”，把键值对排序，同时把相同的键的值整合；最后经过 Reduce 阶段，把整合后的键值对组进行逻辑处理，输出到新的键值对 <K3，V3>。这样的一个过程，其实就是 MapReduce 的本质。

Hadoop MapReduce 可以根据其使用的资源管理框架不同，而分为 MR v1 和 YARN/MR v2 版本，如图 1-9 所示。

在 MR v1 版本中，资源管理主要是 Jobtracker 和 TaskTracker。Jobtracker 主要负责：作业控制（作业分解和状态监控），主要是 MR 任务以及资源管理；而 TaskTracker 主要是调度 Job 的每一个子任务 task；并且接收 JobTracker 的命令。



在 YARN/MR v2 版本中，YARN 把 JobTracker 的工作分为两个部分：

1. ResourceManager（资源管理器）全局管理所有应用程序计算资源的分配。
2. ApplicationMaster 负责相应的调度和协调。

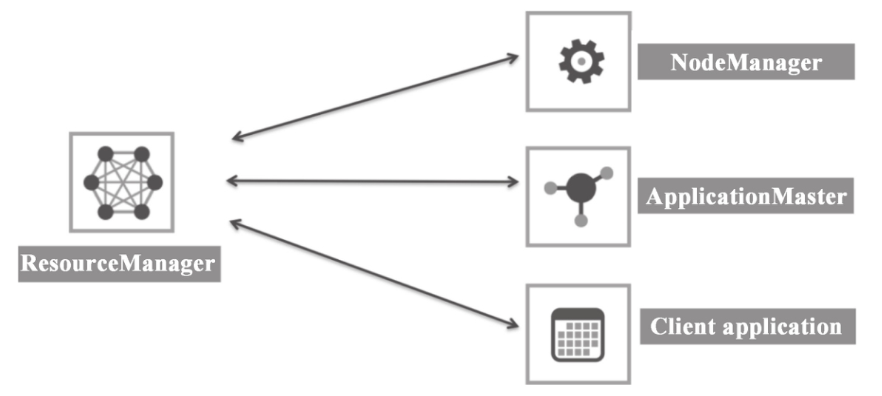
NodeManager 是每一台机器框架的代理，是执行应用程序的容器，监控应用程序的资源（CPU、内存、硬盘、网络）使用情况，并且向调度器汇报。

**四、Hadoop 资源管理 — YARN**

在上一节中我们看到，当 MapReduce 发展到 2.x 时就不使用 JobTracker 来作为自己的资源管理框架，而选择使用 YARN。这里需要说明的是，如果使用 JobTracker 来作为 Hadoop 集群的资源管理框架的话，那么除了 MapReduce 任务以外，不能够运行其他任务。也就是说，如果我们集群的 MapReduce 任务并没有那么饱满的话，集群资源等于是白白浪费的。所以提出了另外的一个资源管理架构 YARN（Yet Another Resource Manager）。这里需要注意，YARN 不是 JobTracker 的简单升级，而是“大换血”。同时 Hadoop 2.X 也包含了此架构。Apache Hadoop 2.X 项目包含以下模块。

* Hadoop Common：为 Hadoop 其他模块提供支持的基础模块。
* HDFS：Hadoop：分布式文件系统。
* YARN：任务分配和集群资源管理框架。
* MapReduce：并行和可扩展的用于处理大数据的模式。

如图 1-10 所示，YARN 资源管理框架包括 ResourceManager（资源管理器）、Applica-tionMaster、NodeManager（节点管理器）。各个组件描述如下。



**（1）ResourceManager**

ResourceManager 是一个全局的资源管理器，负责整个系统的资源管理和分配。它主要由两个组件构成：调度器（Scheduler）和应用程序管理器（ApplicationManager，AM）。

Scheduler 负责分配最少但满足 Application 运行所需的资源量给 Application。Scheduler 只是基于资源的使用情况进行调度，并不负责监视/跟踪 Application 的状态，当然也不会处理失败的 Task。

ApplicationManager 负责处理客户端提交的 Job 以及协商第一个 Container 以供 App-licationMaster 运行，并且在 ApplicationMaster 失败的时候会重新启动 ApplicationMaster（YARN 中使用 Resource Container 概念来管理集群的资源，Resource Container 是资源的抽象，每个 Container 包括一定的内存、IO、网络等资源）。

**（2）ApplicationMaster**

ApplicatonMaster 是一个框架特殊的库，每个 Application 有一个 ApplicationMaster，主要管理和监控部署在 YARN 集群上的各种应用。

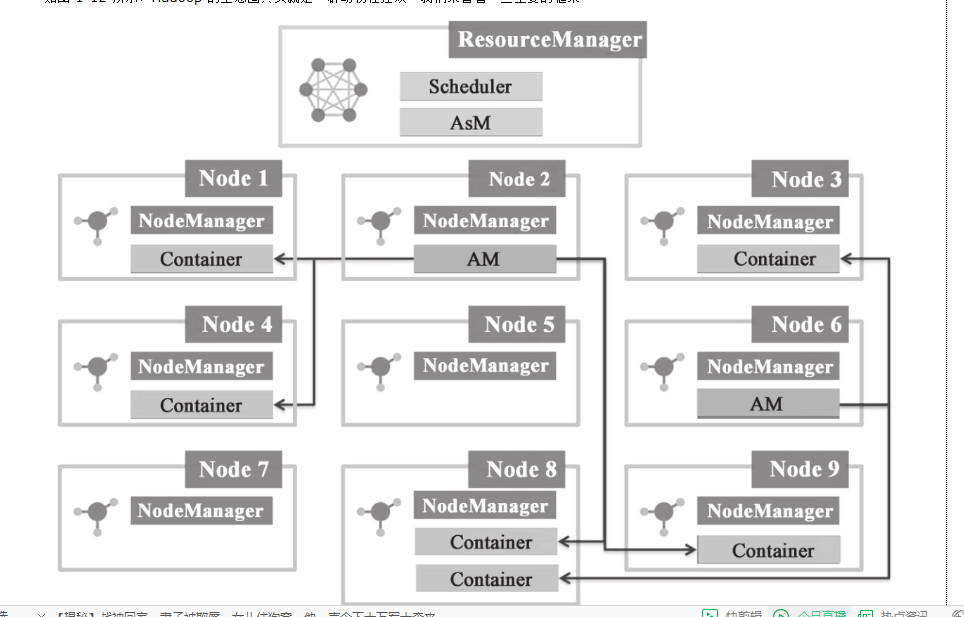
**（3）NodeManager**

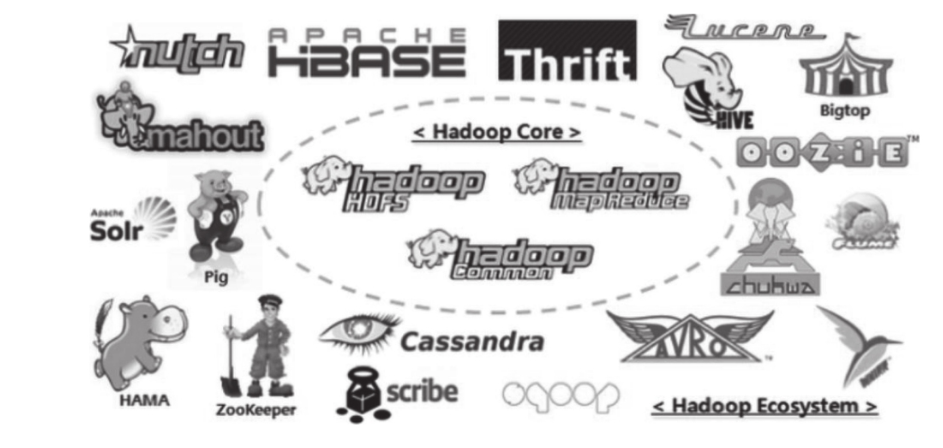
主要负责启动 ResourceManager 分配给 ApplicationMaster 的 Container，并且会监视 Container 的运行情况。在启动 Container 的时候，NodeManager 会设置一些必要的环境变量以及相关文件；当所有准备工作做好后，才会启动该 Container。启动后，NodeManager 会周期性地监视该 Container 运行占用的资源情况，若是超过了该 Container 所声明的资源量，则会 kill 掉该 Container 所代表的进程。

如图 1-11 所示，该集群上有两个任务（对应 Node2、Node6 上面的 AM），并且 Node2 上面的任务运行有 4 个 Container 来执行任务；而 Node6 上面的任务则有 2 个 Container 来执行任务。

**五、Hadoop 生态系统**

如图 1-12 所示，Hadoop 的生态圈其实就是一群动物在狂欢。我们来看看一些主要的框架。



**（1）HBase**

HBase（Hadoop Database）是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用 HBase 技术可在廉价 PC Server 上搭建起大规模结构化存储集群。

**（2）Hive**

Hive 是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。

**（3）Pig**

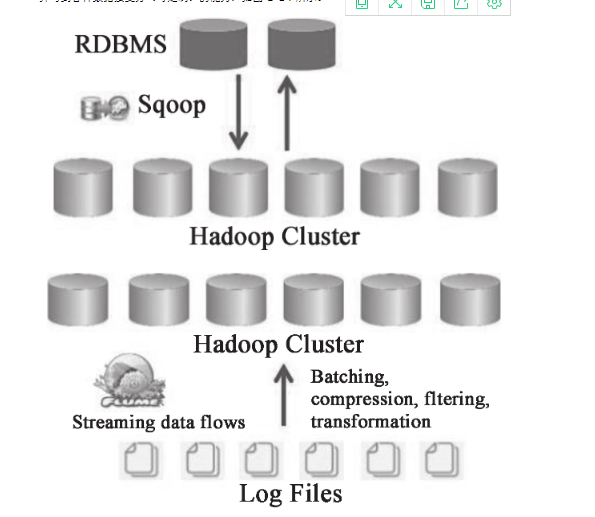
Pig 是一个基于 Hadoop 的大规模数据分析平台，它提供的 SQL-LIKE 语言叫作 Pig Latin。该语言的编译器会把类 SQL 的数据分析请求转换为一系列经过优化处理的 Map-Reduce 运算。

**（4）Sqoop**

Sqoop 是一款开源的工具，主要用于在 Hadoop（Hive）与传统的数据库（MySQL、post-gresql等）间进行数据的传递，可以将一个关系型数据库中的数据导入 Hadoop 的 HDFS 中，也可以将 HDFS 的数据导入关系型数据库中，如图 1-13 所示。

**（5）Flume**

Flume 是 Cloudera 提供的一个高可用、高可靠、分布式的海量日志采集、聚合和传输的系统，Flume 支持在日志系统中定制各类数据发送方，用于收集数据。同时，Flume 提供对数据进行简单处理并写到各种数据接受方（可定制）的能力，如图 1-14 所示。



**（6）Oozie**

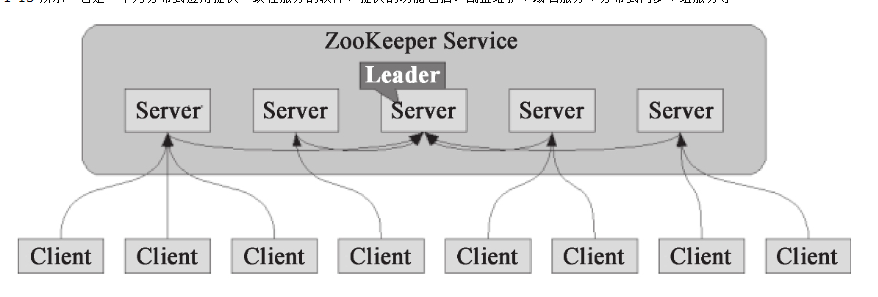
Oozie 是基于 Hadoop 的调度器，以 XML 的形式写调度流程，可以调度 Mr、Pig、Hive、shell、jar 任务等。

主要的功能如下。

1. Workflow：顺序执行流程节点，支持 fork（分支多个节点）、join（将多个节点合并为一个）。
2. Coordinator：定时触发 Workflow。
3. Bundle Job：绑定多个 Coordinator。

**（7）Chukwa**

Chukwa 是一个开源的、用于监控大型分布式系统的数据收集系统。它构建在 Hadoop 的 HDFS 和 MapReduce 框架上，继承了 Hadoop 的可伸缩性和鲁棒性。Chukwa 还包含了一个强大和灵活的工具集，可用于展示、监控和分析已收集的数据。



**（9）Avro**

Avro 是一个数据序列化的系统。它可以提供：丰富的数据结构类型、快速可压缩的二进制数据形式、存储持久数据的文件容器、远程过程调用 RPC。

**（10）Mahout**

Mahout 是 Apache Software Foundation（ASF）旗下的一个开源项目，提供一些可扩展的机器学习领域经典算法的实现，旨在帮助开发人员更加方便快捷地创建智能应用程序。Mahout 包含许多实现，包括聚类、分类、推荐过滤、频繁子项挖掘。此外，通过使用 Apache Hadoop 库，可以有效地将 Mahout 扩展到云中。

# 三 数据治理

# 四 Java基础

# 五 版本控制工具

$ git merge dev-20191119

# 六、面试记录

## PAJF面试

## 1. 挑选项目经历中最有价值的一个讲出来，如果让你重构，你会怎么样做才能做得更好？

SQL基础：

select \* from A

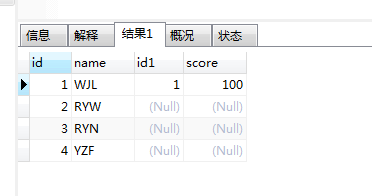


select \* from B



select \* from A

LEFT JOIN B ON A.id = B.id AND A.id =1



select \* from A

LEFT JOIN B

ON A.id = B.id

WHERE B.id =1



这两种写法的区别在于，过滤条件放在ON的后面是在联合之前就进行过滤，放在WHERE后面是在联合之后的结果集上进行过滤。

索引失效了，怎么弄都没有用怎么办

一直调

Reorg一下表？

### 2.数据库基础知识

文章参考地址：

<https://wenku.baidu.com/view/c73b15b17e21af45b207a850.html>

<https://www.jianshu.com/p/c65c4ec0fb3f>

<https://blog.csdn.net/dosthing/article/details/87954213>

#### （1） 数据库设计的三范式 原则及为了解决什么问题

为了建立冗余较小、结构合理的数据库，设计数据库时必须遵循一定的规则。在关系型数据库中这种规则就称为范式(英文Normal Form,缩写NF)。范式是符合某一种设计要求的总结。要想设计一个结构合理的关系型数据库，必须满足一定的范式

在实际开发中最为常见的设计范式有三个：

1）.第一范式（确保每列保持原子性）

第一范式是最基本的范式。如果数据库表中的所有字段都是不可分解的原子值，就说明了该数据库表满足了第一范式。数据表中的每一列（字段），必须是不可拆分的最小单元，也就是确保每一列的原子性，而不是集合。

实例：如以下订单表，买家地址列并不符合第一范式，需要继续拆分



上表所示的订单遵循了第一范式的要求，这样对用户使用城市进行分类的时候就非常方便，也提高了数据的性能

2）.第二范式（确保表中的每列都和主键相关）

第二范式在第一范式的基础之上更进一步。第二范式需要确保数据库中的每一列和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，只能保存一种数据，不可以把多种数据保存在同一张数据库表中。

满足1NF的基础上，要求：表中的所有列，都必须依赖于主键，而不能有任何一列与主键没有关系（一个表只描述一件事情）。第二范式消除表的无关数据。

将上述订单表拆分成多张表，拆分后如下图



3）.第三范式（确保每列都和主键列直接相关，而不是间接相关）

第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关，列和列之间不存在相互依赖关系。要求一个关系中不包含已在其它关系已包含的非主关键字信息。

满足2NF的基础上，任何非主属性不依赖于其他非主属性（在2NF的基础上消除依赖传递）（也表明不允许数据存在冗余的现象）。

上述表中，虽然单价和订购数量可以计算出总价，但是单价是直接和商品相关的，并不是和订单直接相关，所以不满足第三范式，需要继续拆分



**（2）数据库的事务性**

除了数据库设计的三大范式之外，事务处理也是保证数据完整性的重要手段。事务是单独的工作单元，该单元可以包含多个操作以完成一个完整的任务。锁是在多用户环境中对数据访问的限制。事务和锁确保了数据的完整性。

**事务处理**

提交commit，当所有的操作步骤都被完整执行后，称该事务被提交。

回滚rollback，由于某一操作步骤执行失败，导致所有都没有被提交，则事务必须回滚，即回到事务执行前的状态。

**事务ACID属性**

事务处理的特性，每一个事务都有他们所共有的特性，叫做ACID特性，分别是原子性atomicity、一致性consistence、隔离性Isolation、持久性Durability

1）原子性，事务的原子性表示事务执行过程中，把事务作为一个工作单元处理，一个工作单元可能包含若干个操作步骤，每个操作步骤都必须完成才算完成，若因任何原因导致其中的一个步骤操作失败，则所有操作步骤失败，前面的步骤必须回滚。

2）一致性，事务的一致性保证数据处于一致状态。如果事务开始时处于一致状态，则事务结束时也应处于一致状态，不管事务成功还是失败。

3）隔离性，事务的隔离性保证事务访问的任何数据都不会受到其他事务所做的任何改变的影响，直到该事务完成。

4）持久性，事务的持久性保证假如事务执行，则它在系统中产生的结果应该是持久的。

（3）数据库五大约束

1、主键约束(Primay Key)

唯一性，非空性

2、唯一约束(Unique)

唯一性，可以空，但只能有一个

3、检查约束(Check)

对该列数据的范围、格式的限制(如：年龄、性别等)

4、默认约束(Defaut)

该数据的默认值

5、外键约束（Foreign Key）

需要建立两表间的关系

#### （2） 数据库设计的拉链表 必须字段

本文链接：<https://blog.csdn.net/weixin_45399233/article/details/100601551>

本文链接：<https://blog.csdn.net/u012965373/article/details/81515463>

本文链接：https://blog.csdn.net/fhy36897/article/details/89553784

增量数据与历史库做成拉链表

首先介绍几个表概念：

**全量表**：每天的所有的最新状态的数据

**增量表**：每天的新增数据

**拉链表**：维护历史状态，以及最新状态数据

**流水表**：对于表中的每一个记录都会修改，可以用于反映实际记录的变更

拉链表VS流水表：

拉链表：通常是对账户信息的历史变动进行处理保留的结果，用于统计业务相关情况

流水表：每天的交易形成的历史，用户统计账户及客户的情况

在数据仓库的数据模型设计过程中，经常会遇到这样的需求：

1）. 数据量比较大；

2）.表中的部分字段会被update，如用户的地址，产品的描述信息，订单的状态等；

3）.需要查看一个时间点或者时间段的历史快照信息，比如，查看一个订单早历史某一个时间点的状态，比如，查看某一个用户在过去某一个时间内，更新过几次等等；

4）变化的频率和比例不是很大，比如，总共有1000万的会员，每天新增和发生变化的有10万左右；

5）如果对这个表每天都保留一份全量，那么每次全量中会保存很多不变的信息，对存储是极大的浪费。

拉链历史表，技能满足反应数据的历史状态，又可以最大程度的节省存储。.

拉链表案例：

1）有一张订单表，2019-06-20 这天里面有三条订单数据：



2）到了2019-06-21这天，表中有5条记录：



3）到了2019-06-22日，表中有6条记录：



这种存储方式有什么问题呢？

1.只保留一份全量，则数据和6月22日的记录一样，如果要查看6月21日订单001的状态，则无法满足。

2、每天都保留一份全量，则数据仓库中的该表共有14条记录，但好多记录都是重复保存，没有任何变化，如订单002,004，数据量大了会造成很大的存储浪费；

如果设计成历史拉链表，如下：



拉链表中字段解释：

1）、dw\_begin\_date 表示该条记录的生命周期的开始时间，dw\_end\_date表示该条记录的生命周期结束时间；

2）dw\_end\_date=’9999-12-31’表示该条记录目前处于有效状态；

3）如果查询当前所有有效的记录，则select \* from order\_his where dw\_end\_date =’9999-12--31’

4)如果查询2019-06-21的历史数据，则select \* from order\_his where dw\_begin\_date <= ‘2019-06-21’ and dw\_end\_date >= ‘2019-06-21’,这条语句会查询到以下记录：



可以看出，这样的历史拉链表，技能满足对历史数据的要求，又能很大程度的节省存储资源。

举例：













**需要参考博客，讲的很清楚，自己整理的由于篇幅限制截图不全。**



**增量抽取**

每天，从源系统订单表中，将前一天的增量数据抽取到ODS层的增量数据表。这里的增量需要通过订单表中的创建时间和修改时间来确定：



**增量刷新历史数据**

从2019-08-22开始，需要每天正常刷新前一天（2019-08-21）的增量数据到历史表。

**第一步，通过增量抽取，将2019-08-21的数据抽取到ODS：**



ODS增量表中2019-08-21的数据如下：



**第二步，通过DW历史数据(数据日期为2019-08-20)，和ODS增量数据(2019-08-21)，刷新历史表：**













#### （3） 数据库调优如何看调优的，怎么解决的，看什么字段

#### （4）给一个场景如何开发

#### （5）常用数据库语句，如何写，分组排序取最大/小

原文链接：https://blog.csdn.net/qq\_25221835/article/details/82762416

<https://blog.csdn.net/qq_25221835/article/details/82762416>

语法格式：row\_number() over(partition by 分组列 order by 排序列 desc)

在使用 row\_number() over()函数时候，over()里头的分组以及排序的执行晚于 where 、group by、  order by 的执行。

例一：

表数据：

create table TEST\_ROW\_NUMBER\_OVER(

id varchar(10) not null,

name varchar(10) null,

age varchar(10) null,

salary int null

);

select \* from TEST\_ROW\_NUMBER\_OVER t;

insert into TEST\_ROW\_NUMBER\_OVER(id,name,age,salary) values(1,'a',10,8000);

insert into TEST\_ROW\_NUMBER\_OVER(id,name,age,salary) values(1,'a2',11,6500);

insert into TEST\_ROW\_NUMBER\_OVER(id,name,age,salary) values(2,'b',12,13000);

insert into TEST\_ROW\_NUMBER\_OVER(id,name,age,salary) values(2,'b2',13,4500);

insert into TEST\_ROW\_NUMBER\_OVER(id,name,age,salary) values(3,'c',14,3000);

insert into TEST\_ROW\_NUMBER\_OVER(id,name,age,salary) values(3,'c2',15,20000);

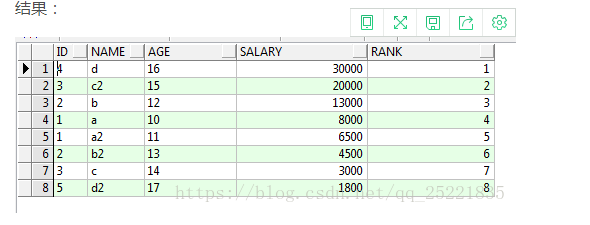
insert into TEST\_ROW\_NUMBER\_OVER(id,name,age,salary) values(4,'d',16,30000);

insert into TEST\_ROW\_NUMBER\_OVER(id,name,age,salary) values(5,'d2',17,1800);

一次排序：对查询结果进行排序（无分组）

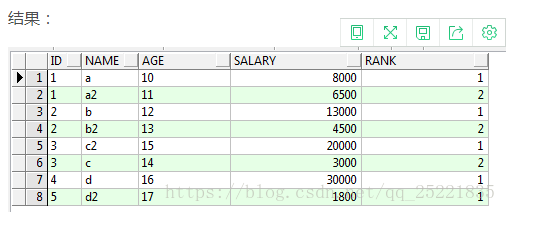
select id,name,age,salary,row\_number()over(order by salary desc) rn

from TEST\_ROW\_NUMBER\_OVER t

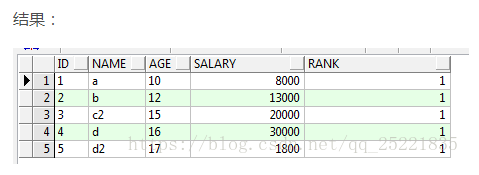


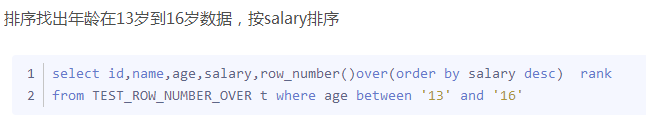
进一步排序：根据id分组排序

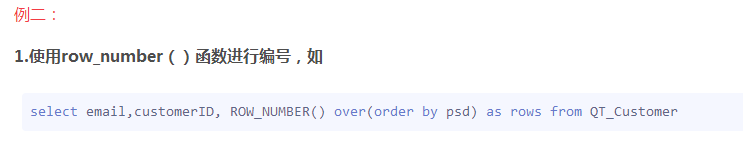
1. select id,name,age,salary,row\_number()over(partition by id order by salary desc) rank
2. from TEST\_ROW\_NUMBER\_OVER t



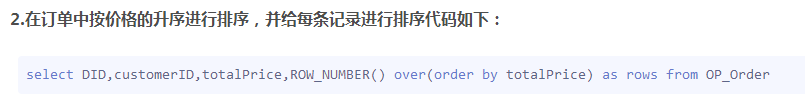
1. select \* from(select id,name,age,salary,row\_number()over(partition by id order by salary desc) rank
2. from TEST\_ROW\_NUMBER\_OVER t)
3. where rank <2



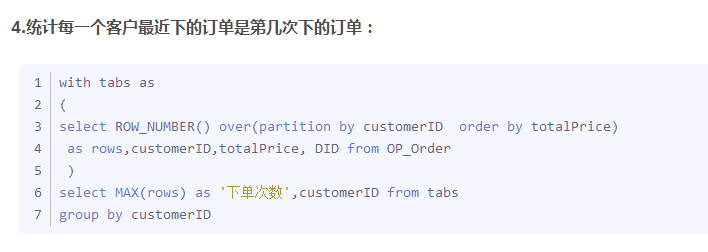




原理：先按psd进行排序，排序完后，给每条数据进行编号。







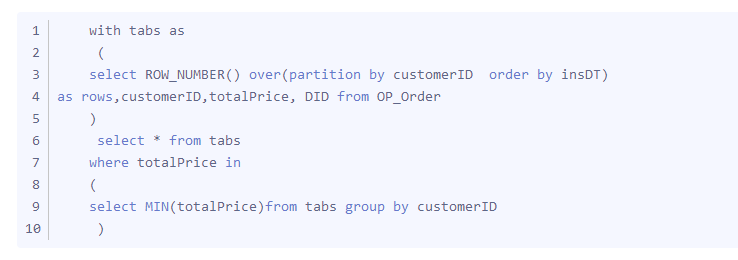
5.统计每一个客户所有的订单中购买的金额最小，而且并统计改订单中，客户是第几次购买的：

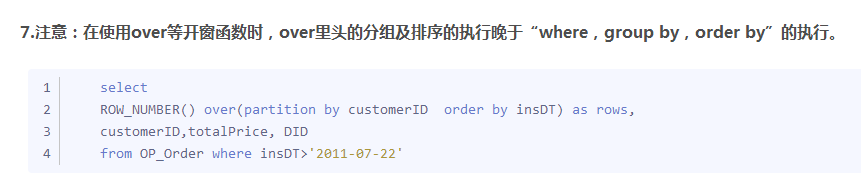
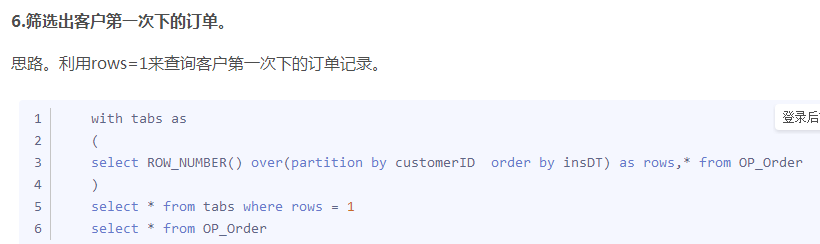
思路：利用临时表来执行这一操作。

1.先按客户进行分组，然后按客户的下单的时间进行排序，并进行编号。

2.然后利用子查询查找出每一个客户购买时的最小价格。

3.根据查找出每一个客户的最小价格来查找相应的记录。





## 3.JAVA会不会

（1）spring /springMVC 的区别，有没有接触过springboot

## 4.SHELL脚本

（1） 定时语句/命令怎么写

（2） 查找一个文件中某个字符串出现的次数

## 5.其他

（1） 平时 有没有打建过系统环境，SVN,服务器之类的，如何搭建的

## 6.为什么要换工作，平时有没有对新技术进行钻研

# 七、读书笔记

第一章 决策支持系统的发展

当数据从操作型环境传向数据仓库环境时，需要对数据进行集成。

第二章 数据仓库环境

数据仓库是一个面向主题的、集成的、非易失，随时间变化的用来支持管理人员决策的数据集合。