目录

1. 实验一：数据仓库运维项目实战

1.1. 实验目的

完成本实验，您应该能够：

- 掌握数据仓库的设计方法
- 掌握数据建仓的流程

1.2. 实验要求

- 熟悉 Hadoop 组件 Sqoop、Hive 的常用指令
- 熟悉 HQL 语句的用法
- 熟悉 HDFS Shell 常用命令

1.3. 实验环境

本实验所需之主要资源环境如表 1-1 所示。

表 1-1 资源环境

服务器集群	3 个以上节点，节点间网络互通，各节点最低配置：双核 CPU、8GB 内存、100G 硬盘
运行环境	CentOS 7.4
服务和组件	HDFS、YARN、MapReduce 等，其他服务根据实验需求安装。各组件版本如下： ● Hadoop 2.7.1 ● Hive 2.0.0 ● Sqoop 1.4.7 ● MySQL 5.7.18

1.4. 实验视图

数据仓库运维项目实战流程如图 1-1

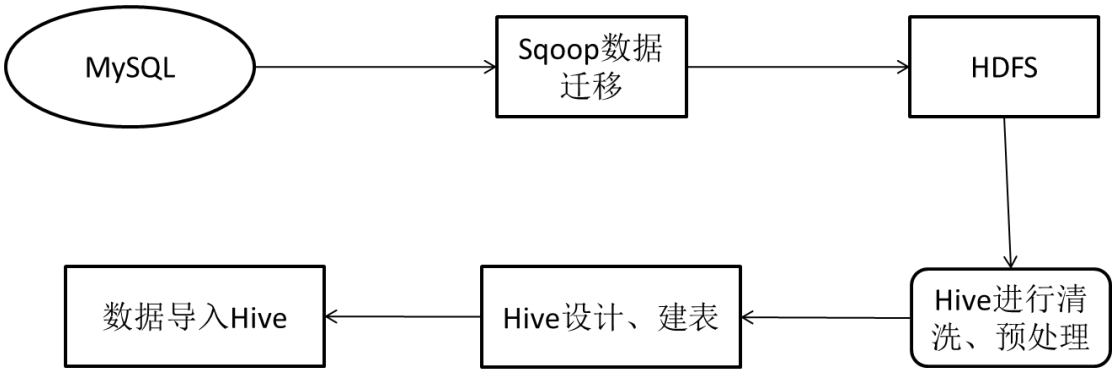

图 1-1 数据仓库项目流程

1.5. 试验过程

1.5.1. 实验任务一：数据导入

1.5.1.1. 步骤一：数据准备

本实验共需要 7 张数据表，存储在 Mysql 数据库中。各表字段说明如下所示：

表 1-2 数据表字段说明

序号	表名	字段	含义
1	movies	movieID	电影 ID
		movieName	电影名
2	user_taggedmovies	userID	用户 ID
		movieID	电影 ID
		tagID	标签 ID
		date_day	日
		date_month	月
		date_year	年
		date_hour	时
		date_minute	分
		date_second	秒
3	user_ratedmovies	userID	用户 ID
		movieID	电影 ID
		rating	评分
		date_day	日
		date_month	月
		date_year	年
		date_hour	时
		date_minute	分
		date_second	秒
4	tags	id	标签 ID
		value	标签值
5	movie_countries	movieID	电影 ID
		country	地区
6	movie_directors	movieID	电影 ID
		directorID	导演 ID
		directorName	导演名
7	movie_actors	movieID	电影 ID
		actorID	演员 ID
		actorName	演员名
		ranking	演员排名

在 Mysql 数据库中的访问用户名为 root，密码为 Password123$。使用 Sqoop 将 Mysql 中的所有表数据导入到 HDFS 集群上，集群路径为/data，文件名与表名保持一致。

1.5.1.2. 步骤二：数据导入

在/home/hadoop 中创建文件夹 data，并创建配置文件 import_conf1。首先导入 movie 表：

```
[hadoop@master ~]$ mkdir data && cd data
[hadoop@master data]$ vi import_conf1
```

```
import
--connect
jdbc:mysql://master:3306/movie
--username
root
--password
Password123$
--target-dir
/data/movie
--table
movies
--m
1
```

执行导入：

```
[hadoop@master data]$ sqoop --options-file import_conf1
```

检查是否已成功导入：

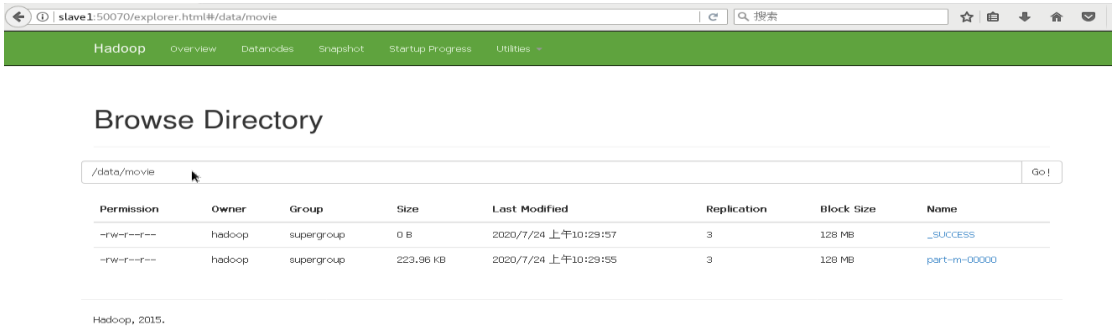

图 1-2 成功导入后的数据示意图

剩余 6 个表使用相同方法导入，全部导入后在浏览器界面查看：

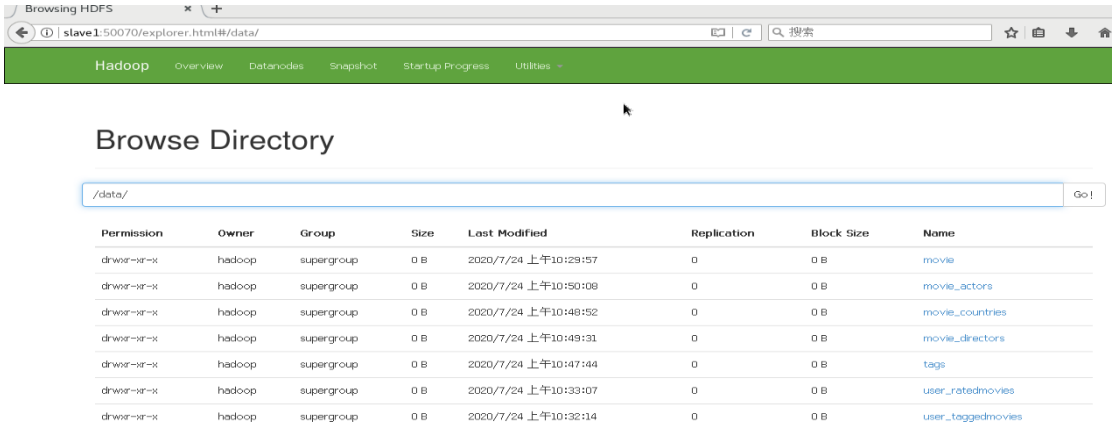

图 1-3 所有数据全部导入后的集群数据

1.5.2. 实验任务二：清洗与预处理

数据表 user_ratedmovies 表中存在一定的脏数据，在进行分析之前，需要对该表进行清洗。

1.5.2.1. 步骤一：清洗空值

1. 数据表的关键字段 rating 存在空值，需要清洗掉这些空值记录。
2. 首先，需要在 hive 中创建临时外部表：

```
hive> create external table tmp_user_ratedmovies
    > (userID string,movieID string,rating double,
    > date_day int,date_month int,date_year int,
    > date_hour int,date_minute int,date_second int)
    > row format delimited fields terminated by ','
    > location '/data/user_ratedmovies';
```

3. 查看表行数：

```
[hadoop@master data]$ hdfs dfs -cat /data/user_ratedmovies/part-m-00000 | wc -l

892547
```

4. 对空值字段进行清洗：

```
hive> create table tmp_user_ratedmovies_1 as
    > select * from tmp_user_ratedmovies
    > where rating is not null;
```

5. 查看行数：

```
hive> select count(*) from tmp_user_ratedmovies_1;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the fut
ure versions. Consider using a different execution engine (i.e. spark, tez) or using Hi
ve 1.X releases.
#省略若干行
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.1 sec   HDFS Read: 2941361
0 HDFS Write: 7 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 100 msec
OK
c0
891785
Time taken: 18.83 seconds, Fetched: 1 row(s)
```

清洗完空值数据后，剩余数据行数 891785 行。

1.5.2.2. 步骤二：清洗重复数据

1. user_ratedmovies 表中存在重复数据，需要进行清洗，否则影响后期数据分析。

```
hive> create table tmp_user_ratedmovies_2 as
    > select distinct
    > userID,movieID,rating,
    > date_day,date_month,date_year,
    > date_hour,date_minute,date_second
    > from tmp_user_ratedmovies_1;
```

2. 清洗后查看文件行数：

hive> select count(*) from tmp_user_ratedmovies_2;

WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the fut ure versions. Consider using a different execution engine (i.e. spark, tez) or using Hi ve 1.X releases.

Query ID = hadoop_20200601095910_0cb40913-ff1b-4be9-9b6e-75567bf5d4b5

Total jobs = 1

Launching Job 1 out of 1

#省略若干行

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.97 sec HDFS Read: 282556 00 HDFS Write: 7 SUCCESS

Total MapReduce CPU Time Spent: 2 seconds 970 msec

OK

c0

855664

Time taken: 19.407 seconds, Fetched: 1 row(s)

清洗完成后剩余 855664 行。

1.5.2.3. 步骤三：清洗非法数据

1. 对 user_ratedmovies 的 rating 字段不在合法区间内的数据进行清洗。用户评分数据位于[0,5]，凡是不属于该区间的数值均为非法数据：

hive> create table tmp_user_ratedmovies_3 as

> select * from tmp_user_ratedmovies_2

> where rating between 0 and 5;

2. 查看行数：

hive> select count(*) from tmp_user_ratedmovies_3;

WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the fut ure versions. Consider using a different execution engine (i.e. spark, tez) or using Hi ve 1.X releases.

Query ID = hadoop_20200601095910_0cb40913-ff1b-4be9-9b6e-75567bf5d4b5

Total jobs = 1

#省略若干行

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.07 sec HDFS Read: 282533 68 HDFS Write: 7 SUCCESS

Total MapReduce CPU Time Spent: 4 seconds 70 msec

OK

c0

855598

Time taken: 20.581 seconds, Fetched: 1 row(s)

全部清洗完毕后，文件剩余 855598 行。

1.5.2.4. 步骤四：其他表的创建

1. Hive 中创建外部表，映射到 HDFS 文件，为后续的分析做准备。首先对 Movies 表进行映射，该表包含字段 movieID 和 movieName：

hive> create external table movies

>(movieID string,movieName string)

```
>row format delimited fields terminated by ','
>location '/data/movie';
```

2. 将 user_taggedmovies 映射到 hive 中，hive 表名保持不变，也为 user_taggedmovies，包含 userID,movieID,tagID,date_day,date_month,date_year,date_hour,date_minute,date_second 等字段：

```
hive> create external table user_taggedmovies
    > (userID string,movieID string,tagID string,
    >date_day int,date_month int,date_year int,
    >date_hour int,date_minute int,date_second int)
    >row format delimited fields terminated by ','
    >location '/data/user_taggedmovies';
```

3. 将 tags 表映射到 hive 中，表名保持不变，包含字段 id，tag_value：

```
hive> create external table tages
    > (id string,tag_value string)
    >row format delimited fields terminated by ','
    >location '/data/tags';
```

4. 将 movie_director 映射到 hive 中，表名保持不变，包含字段 movieID,directorID,directorName 等字段：

```
hive> create external table movie_directors
    > (movieID string,directorID string,directorName string)
    >row format delimited fields terminated by ','
    >location '/data/movie_directors';
```

5. 将 movie_countries 映射到 hive 中，表名保持不变，包含字段 movieID,country：

```
hive> create external table movie_countries
    > (movieID string,country string)
    >row format delimited fields terminated by ','
    >location '/data/movie_countries';
```

6. 将 movie_actors 映射到 hive 中，表名保持不变，包含字段 movieID,actorID,actorName,ranking：

```
hive> create external table movie_actors
    > (movieID string,actorID string,actorName string,ranking int)
    >row format delimited fields terminated by ','
    >location '/data/movie_actors';
```

1.5.3. 实验任务三：数据建仓

本项目数据较为整齐，ODS 和 PDW 层可以合并为一层，将清洗后的评分表和其他初始数据入库；根据最终的分析任务要求，我们需要电影、平均评分、评价数、电影导演、电影演员、电影类别、地区进行关联查询，得到一张聚合后的宽表作为 MID 层数据保存下来。APP 层为最终的分析数据。

1.5.3.1. 步骤一：PDW 层数据导入

将 1.5.2.4 创建的 6 个外部表和清洗后的表 tmp_user_ratedmovies_3，入库到 Hive 仓库中，作为 PDW 层数据。数据表名在原表名前加上 PDW_前缀进行区分：

```
hive>create table PDW_movie_actors as select * from movie_actors;
hive>create table PDW_movie_countries as select * from movie_countries;
hive>create table PDW_movie_directors as select * from movie_directors;
hive>create table PDW_movies as select * from movies;
hive>create table PDW_tags as select * from tages;
hive>create table PDW_user_ratedmovies as select * from tmp_user_ratedmovies_3;
hive>create table PDW_user_taggedmovies as select * from user_taggedmovies;
```

1.5.3.2. 步骤二：MID 层导入

由于我们需要分析电影与评分、标签、导演、演员等关联关系，因此需要将它们关联起来为下一步数据分析做准备。

1. 首先创建电影评分基本信息表 MID_movie_rating_info，包括电影 ID，电影名，电影平均评分，电影评论数，所属地区，导演 ID，导演名等字段：

```
hive> create table MID_movie_rating_info as
    > select m_r.movieID,mv.movieName,m_r.count_rating,m_r.avg_rating,mc.country,
md.directorID,md.directorName
    > from
    > (select m.movieID,avg(r.rating) as avg_rating,count(r.rating) as count_rating
    > from PDW_movies m join PDW_user_ratedmovies r
    > on m.movieID=r.movieID
    > group by m.movieID order by count_rating desc,avg_rating desc) m_r
    > join PDW_movies mv on m_r.movieID=mv.movieID
    > join PDW_movie_countries mc on m_r.movieID=mc.movieID
    > join PDW_movie_directors md on m_r.movieID=md.movieID;
```

2. 电影演员表 MID_movie_actors，包括电影 ID，电影名，所有演员个数，按排名统计各阶段演员个数（本项目统计前 5 个数为 level1，6-20 为 level2，21 以及以后为 level3），所有演员：

```
hive> create table MID_movie_actors as
    > select m.movieName,m_a.* from
    > (select m.movieID,count(ma.actorID),
    > concat_ws('|',collect_set(ma.actorID)) actorID,
    > concat_ws('|',collect_set(ma.actorName)) actorName,
    > sum(case when ma.ranking<=5 then 1 else 0 end) level1,
    > sum(case when ma.ranking>5 and ma.ranking<=20 then 1 else 0 end) level2,
    > sum(case when ma.ranking>20 then 1 else 0 end) level3
    > from PDW_movies m
    > join PDW_movie_actors ma on m.movieID=ma.movieID
    > group by m.movieID) m_a
    > join PDW_movies m on m.movieID=m_a.movieID;
```

3. 电影标签 MID_movie_tags，包括电影 ID，电影名，所有标签个数，所有标签；

```
hive> create table MID_movie_tags as
    > select m.movieName,m_t.*
    > from
    > (select m.movieID,concat_ws(',',collect_set(t.tag_value))
    > from PDW_user_taggedmovies ut
```

```
> join PDW_movies m on m.movieID=ut.movieID
> join PDW_tags t on t.id=ut.tagID
> group by m.movieID) m_t
> join PDW_movies m on m.movieID=m_t.movieID;
```

4. 演员表 MID_actors，包括演员 ID，演员名，电影个数，各个级别的电影个数（本项目中电影评分在[4,5]为 leve1，在[3,4]位 level2，低于 3 分位 level3）；

```
hive> create table MID_actors as
> select distinct m_a_d.* from(
> select m_a.actorID,ma.actorName,m_a.moviecounts,m_a.level1,m_a.level2,m_a.level3
> from
> (select ma.actorID,count(ma.movieID) moviecounts,
> sum(case when mr.avg_rating>=4 and mr.avg_rating<=5 then 1 else 0 end) level1,
> sum(case when mr.avg_rating>=3 and mr.avg_rating<4 then 1 else 0 end) level2,
> sum(case when mr.avg_rating<3 then 1 else 0 end) level3
> from PDW_movie_actors ma
> join MID_movie_rating_info mr on mr.movieID=ma.movieID
> group by ma.actorID) m_a
> join PDW_movie_actors ma
> on ma.actorID=m_a.actorID) m_a_d;
```

5. 导演表 MID_directors，包括导演 ID，导演名，电影个数，各个级别的电影个数；

```
hive> create table MID_directors as
> select distinct m_d_d.*
> from
> (select md.directorName,m_d.*
> from
> (select md.directorID,count(md.movieID) moviecounts,
> sum(case when mr.avg_rating>=4 and mr.avg_rating<=5 then 1 else 0 end) level1,
> sum(case when mr.avg_rating>=3 and mr.avg_rating<4 then 1 else 0 end) level2,
> sum(case when mr.avg_rating<3 then 1 else 0 end) level3
> from PDW_movie_directors md
> join MID_movie_rating_info mr on mr.movieID=md.movieID
> group by md.directorID) m_d
> join PDW_movie_directors md
> on md.directorID=m_d.directorID) m_d_d;
```

1.5.4. 实验任务四：业务调用

MID 层又称为数据集市层，汇聚了全部的业务数据。在实际应用中，不同的业务场景所需要的字段也不同。比如销售部关心最受欢迎的电影，需要查看评论最高的 50 部电影；同时由于评分与评论数量相关，评论数量太低的电影，说明关注度较少，因此还需要提取评论数量。

1. 查询 MID_movie_rating_info 表获取评论最多，排名最靠前的 50 部电影，作为 APP 层数据放到 hive 仓库中：

```
hive> create table APP_top50Movies as
    > select * from MID_movie_rating_info order by
    >MID_movie_rating_info.count_rating desc,avg_rating desc limit 50;
```

2. 查看表数据的前 10 行：

```
hive> select * from APP_top50Movies limit 10;
OK
2571    The Matrix      1670    4.173952095808383       USA     andy_wachowski A
ndy Wachowski
4993    The Lord of the Rings: The Fellowship of the Ring       1576    4.082487
30964467        New Zealand     peter_jackson   Peter Jackson
356     Forrest Gump    1568    3.9301658163265305      USA     robert_zemeckisR
obert Zemeckis
296     Pulp Fiction    1537    4.238451528952505       USA     quentin_tarantin
o       Quentin Tarantino
5952    The Lord of the Rings: The Two Towers   1528    4.030104712041885       U
SA      peter_jackson   Peter Jackson
2858    American Beauty 1472    4.107676630434782       USA     sam_mendes      S
am Mendes
7153    The Lord of the Rings: The Return of the King   1457    4.09402882635552
5       USA     peter_jackson   Peter Jackson
480     Jurassic Park   1448    3.4305939226519335      USA     steven_spielberg
Steven Spielberg
318     The Shawshank Redemption        1441    4.365371269951423       USA     f
rank_darabont   Frank Darabont
2959    Fight Club      1434    4.25278940027894        USA     david_fincher   D
avid Fincher
Time taken: 0.079 seconds, Fetched: 10 row(s)
```

图 1-4 Top50Movies 查询结果