DM DBA 手记之 MySQL 移植到 DM

1 概述

随着国家对信创工程的日益重视，目前采用有自主知识产权的国产数据库将成为主流。

DM 数据库和 MySQL 体系结构上存在差异，SQL 语法也存在一定的差异。DM 数据库针对 MySQL 做了部分兼容性。但由于有根本性的差异，兼容度不高。从 MySQL 迁移到 DM 数据库，DM 数据库提供了自动数据迁移工具，但在开发级别还需要一定的人工干预。

MySQL 到 DM 的移植主要有以下几个方面的工作：

1．分析待移植系统，确定移植对象。

2．通过数据迁移工具 DTS 完成常规数据库对象及数据的迁移。

3．通过人工完成 MSQL 的移植。

4．移植完成后对移植的结果进行校验，确保移植的完整性和正确性。

5．对应用系统进行移植、测试和优化。

2 待移植系统分析

2.1 应用系统情况分析

应用后台操作系统	Red Hat Linux
数据库后台操作系统	Red Hat Linux
后台数据库	MySQL5.7
应用开发平台	JAVA
应用开发接口	JDBC
需要移植的数据库对象	表（数据量）、分区表 视图

	自定义类型 触发器 存储过程、函数 其他

2.2 数据库对象分析

分析系统中历史数据库中含有哪些表，包括业务表和系统表，以及关联结构。同时明确是否将所有数据表均导入国产化数据库，并提出解决方案。

通过以下方式统计 MySQL 数据中的对象以及表数据量：

```
--根据指定用户统计用户下的对象数目
SELECT count(*) TABLES, table_schema FROM information_schema.TABLES
where table_schema = '数据库名称' GROUP BY table_schema;//表数目
select `name` from MySQL.proc where db = '数据库名称' and `type` = 'PROCEDURE'; //  存储过程
select `name` from MySQL .proc where db = '数据库名称' and `type` = 'FUNCTION' //  函数
```

```
--创建移植辅助表，统计每个表的数据量并插入到移植辅助表中
create table MySQLs_tables(tab_owner varchar(100),tab_name varchar(100),tab_count int);

insert into MySQLs_tables select table_name,table_rows from information_schema.TABLES where TABLE_SCHEMA = '数据库名称' order by table_rows desc;
select * from MySQL_tables;
```

从 MySQL 移植到 DM，要求必须创建新的用户和表空间，不要把数据迁移到系统管理员 SYSDBA 用户下和 MAIN 表空间下。

首先需要分析本次移植 MySQL 源库需要移植的是哪一个或者哪几个用户的数据，然后分别创建这些需要移植的用户和对应的表空间。

3 准备移植环境

3.1 DM 移植环境

3.1.1 版本选择

达梦数据库内部会有定期的版本更新说明和版本发版通知，在进行项目移植的之前，一定要先根据内部通报情况和自己所在技术团队的讨论，确定一个版本，尽量以最新版本且无额外另行通知的版本，保证已经出现的问题在即将移植的系统中不再出现。

版本优先选择完整安装版本（无完整安装版本的平台例外），避免数据库客户端和服务器端存在版本不匹配带来的额外工作量，达梦在不同平台的不同版本上，安装包都会有差异，一定要采用严格匹配的原则，除非得到达梦原厂技术人员的允许，尽量减少干扰性的问题出现。

3.1.2 初始化库

初始化库，关键的点在于对初始化参数的设置，本章节明确是从 MySQL 移植到 DM 数据库，所以具体的初始化参数建议如下：

（1）关于页大小 PAGE_SIZE。在 DM 数据库中，页大小可以为 4KB、8KB、16KB 或者 32KB，从 MySQL 移植到 DM，建议设置页大小为 16KB，一旦创建好了数据库，在该库的整个生命周期内，页大小都不能够改变。除了每个字段的最大长度限制外，每条记录总长度不能大于页面大小的一半。如果系统中存在或者以后可能存在含有较长的字符串类型的表，可以按需调整，最大为 32KB（也可以设置为 8KB）。页大小设置越大，最后数据文件的物理大小就会越大，系统运行时，每次从磁盘调入内存的数据单位也就越大，所以此处要慎重。

（2）关于簇大小 EXTENT_SIZE。数据文件使用的簇大小，即每次分配新的段空间时连续的页数，只能是 16 页或 32 页，缺省使用 16 页，从 MySQL 移植到 DM 使用默认值就可。

（3）关于大小写敏感 CASE_SENSITIVE。DM 为了兼容不同的数据库，在

初始化数据库的时候有一个参数字符串比较大小写敏感，用于确定数据库对象及数据是否区分大小写，默认为区分，不可更改。建议 MySQL 和 SQLSERVER 迁移过来的系统，使用大小写不敏感。

（4）关于字符集 CHARSET。建议采用默认值 GB18030，如果需要国际字符可以采用 Unicode，GB18030 数字字母占 1 个字节，普通汉字占 2 个字节，部分繁体及少数民族文字占 4 字节，Unicode 在达梦中采用 UTF-8 编码格式，欧洲的字母字符占 1 到 2 个字节，亚洲的大部分字符占 3 个字节，附加字符为 4 个字节。如果只存储中文和字母数字，一般来说 GB18030 更节省空间一些。

3.1.3 INI 中兼容参数

DM 的 INI 参数文件中针对从 MySQL 移植到 DM，有几个专门的参数，这里将详细介绍。

compatibility	使用效果及建议
COMPATIBLE_ MODE	是否兼容其他数据库模式。0 ：不兼容，1： 兼容 SQL92 标准，2 ：兼容 ORACLE ，3 ：兼 容 MS SQL SERVER ，4 ：兼容 MySQL ，5 ：兼 容 DM6，6：兼容 Teradata，所以当从 MySQL 移植到 DM 时，修改值为 4；

在 INI 参数的 compatibility 部分，还有其它的一些参数，在涉及到之前，尽量保持默认值，在移植准备的环节，先只调整这个参数就可以了，其它参数，在移植过程中再具体分析。

3.1.4 创建用户和表空间

从 MySQL 移植到 DM，要求先创建好待使用的用户和这个用户的表空间，不要把数据移植到系统默认的管理员 SYSDBA 用户下和 MAIN 表空间下。

MySQL 的体系架构是单实例多库，DM7 以上的版本是单库多实例的架构，MySQL 可能是一个 root 用户访问多个库，访问前切换一下当前库即可。从 MySQL 迁移到达梦的时候就需要针对 MySQL 中的每一个库在达梦里面创建一个用户和表空间来对应。例如 MySQL 中有一个库 test，达梦里面先创建一个表空间 tab_test，然后创建一个用户 test，指定默认表空间为 tab_test。

迁移 MySQL 中 test 库的数据的时候，用 root 用户连接 MySQL，指定当前库为 test；用 test 用户连接达梦，这样就把 MySQLtest 库中的数据迁移到了达梦 test 用户中。

我们在做 MySQL 移植的时候要先分析本次移植需要从源库中移植哪一个库或者哪几个库的数据，然后为每一个库，分别在达梦中创建独立的表空间和用户；大多数情况下，我们需要移植的数据所在的 MySQL 实例里面有多个库，并不是所有的库都需要移植，所以再移植准备阶段，一定要和相关技术负责人员沟通明确清楚。

3.2 MySQL 移植环境

在从 MySQL 向 DM 进行移植准备阶段，也需要注意 MySQL 的移植环境：

（1）严禁在生产环境中直接迁移。因为移植首先是一个测试的工作，所以移植应该避免从 MySQL 生产环境数据库中直接进行移植，需要提前向应用开发商提出其搭建一个测试环境，准备 MySQL 需要移植的环境和数据。直接从生产库上进行数据移植，有很多风险存在，例如会影响生产库的效率，引发崩溃的可能等等。

（2）工具准备。这里推荐 Navicat 工具，一般需要有一个工具来连接到需要移植的 MySQL 环境，以便进行移植数据的确认和初步的分析，当然也可以使用 MySQL 其他的客户端工具，MySQL 是一个开源数据库，可供选择的工具比较多，使用起来比较方便；

4 数据迁移

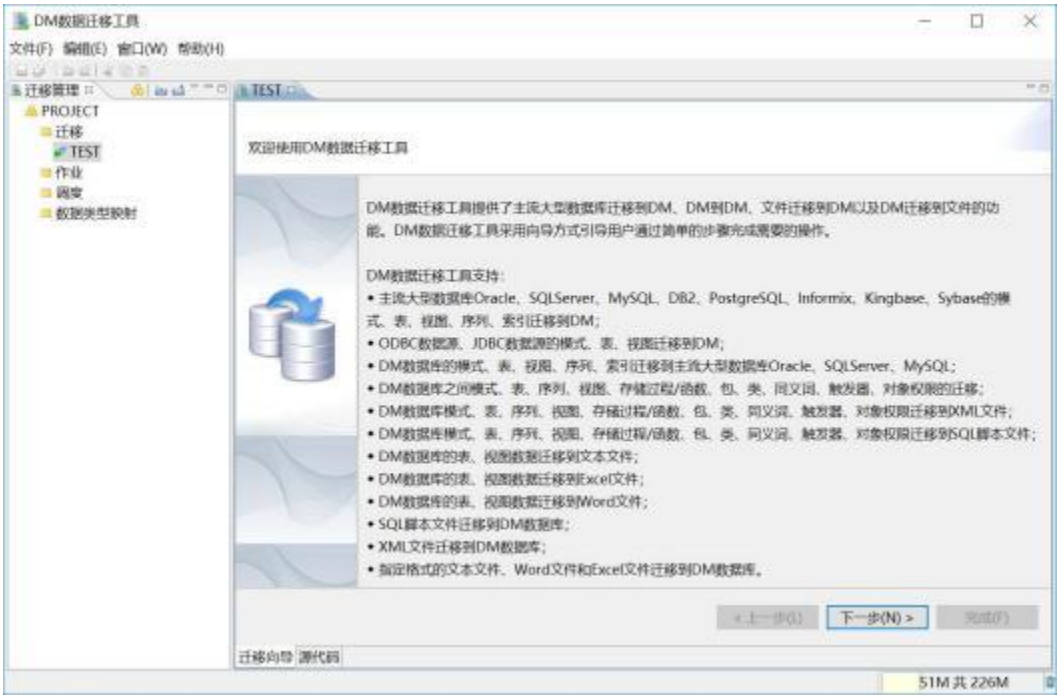
通过达梦数据库数据迁移同步工具 DTS 将原系统中结构化数据迁移到国产化数据库，源端数据库支持 Oracle、达梦、MySQL 等国内外多种关系型数据库管理系统，目的端数据库支持国产化数据库：

DM 数据迁移工具支持：

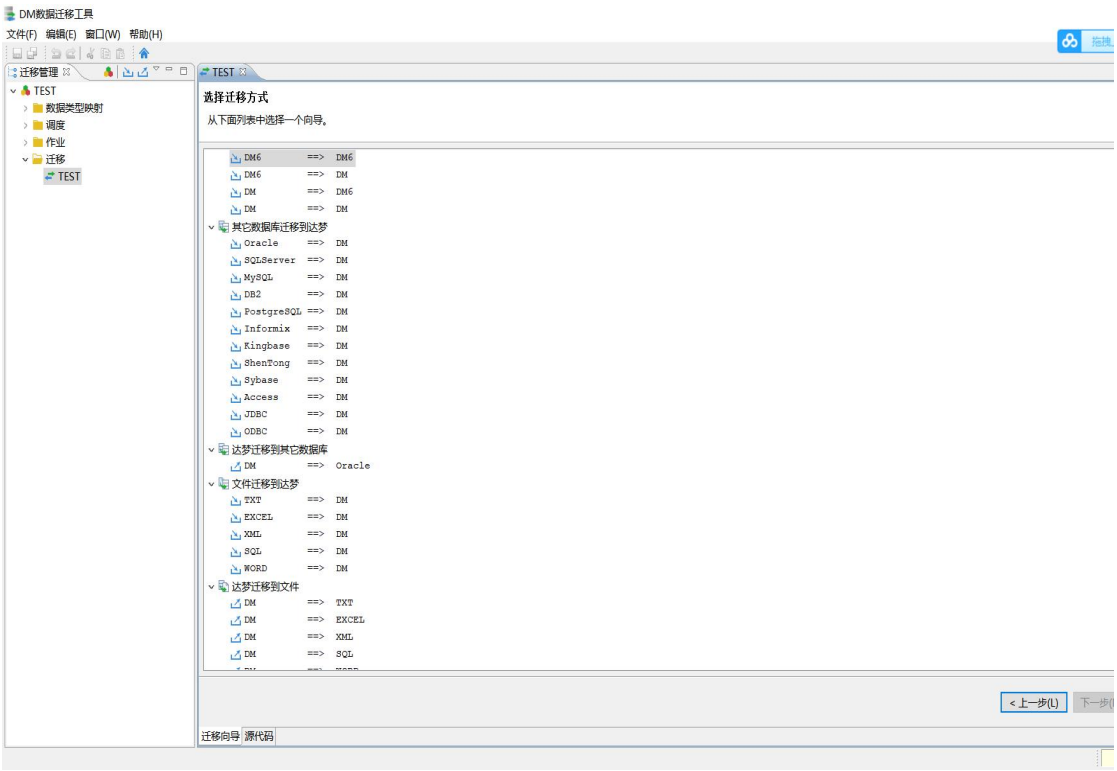
- DM 数据库之间模式、表、序列、视图、存储过程/函数、包、触发器、对象权限的迁移；

- 主流大型数据库 Oracle、SQLServer、MySQL、DB2、PostgreSQL、Informix、Kingbase、Sybase 的模式、表、视图、序列、索引迁移到 DM；
- DM 的模式、表、视图、序列、索引迁移到主流大型数据库 Oracle、SQLServer、MySQL；
- ODBC 数据源、JDBC 数据源的模式、表、视图迁移到 DM；
- DM 数据库模式、表、序列、视图、存储过程/函数、包、触发器、对象权限迁移到 XML 文件，SQL 脚本文件；
- DM 数据库的表、视图数据迁移到文本文件，Excel 文件，Word 文件；
- 指定格式的文本文件,Excel 文件，Word 文件，XML 文件和 SQL 脚本文件迁移到 DM 数据库。

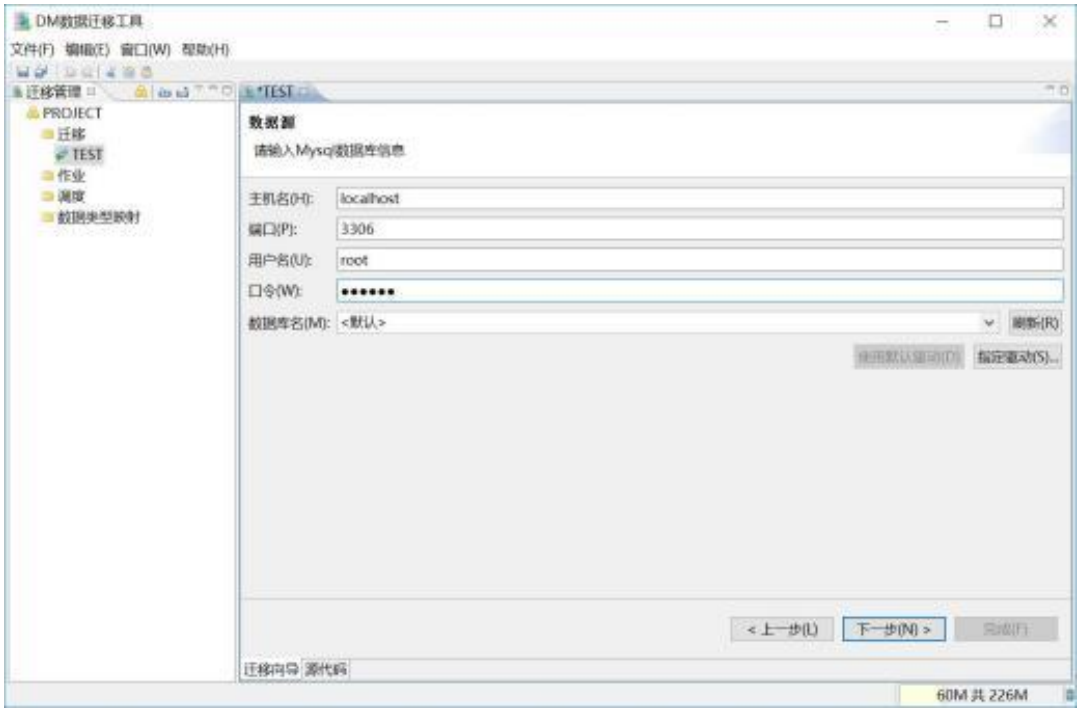
4.1 数据迁移步骤



（1）选择迁移方式



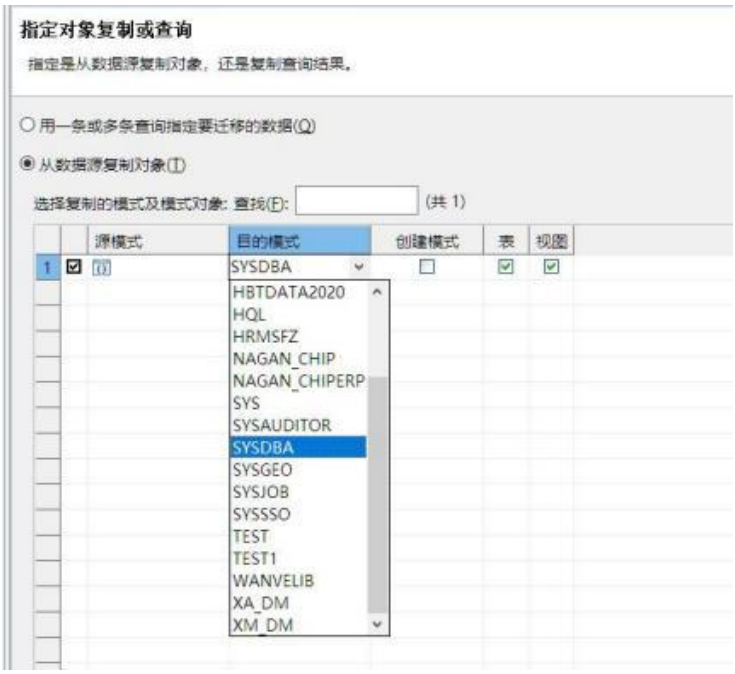
（2）连接迁移数据源

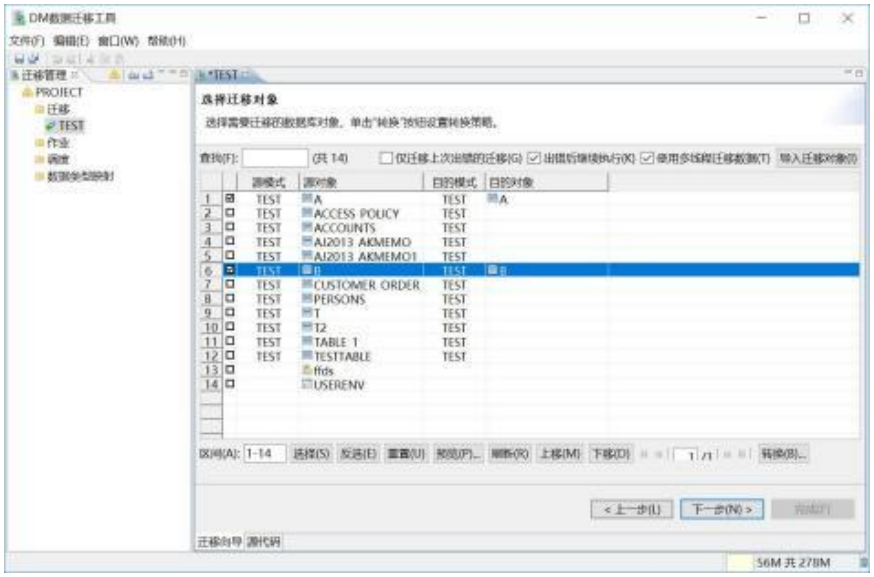


这里需要注意选择对应的要迁移的数据库名称，不要使用默认，如果下拉框没有可选的，可以点击右边的刷新。

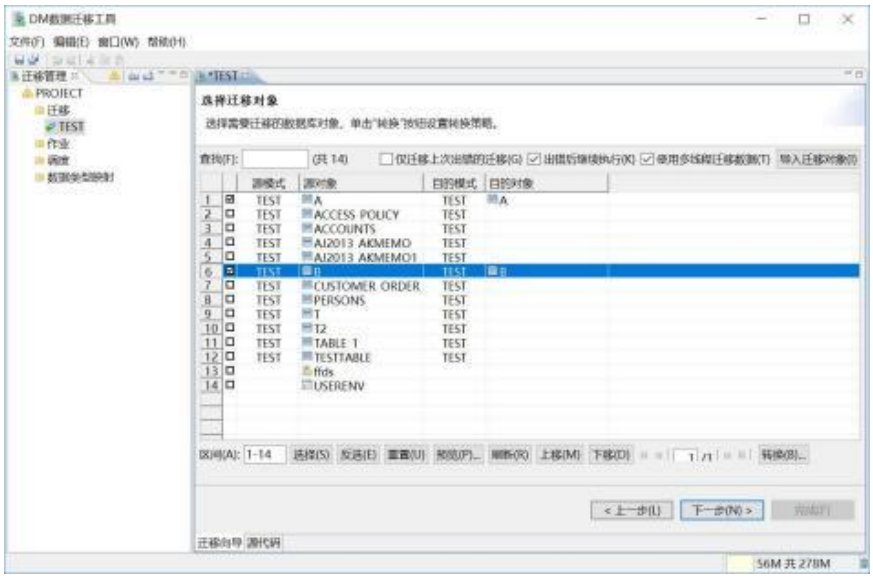
（3）连接迁移数据目的端

（4）从数据源复制对象,包括模式及模式对象，目录，公共同义词或者上下文（目的模式选择自己要用的模式）。

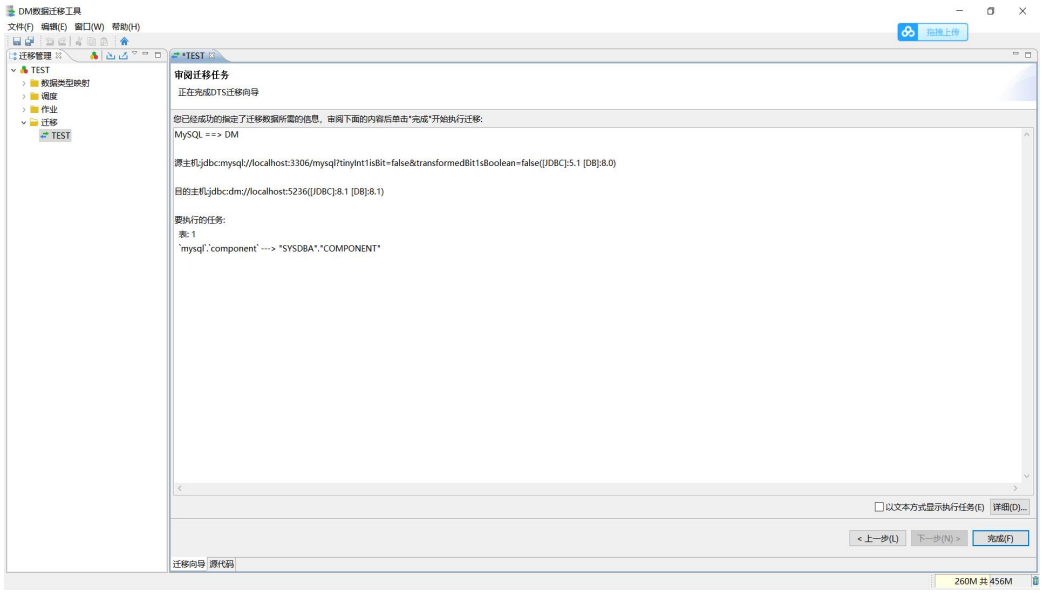




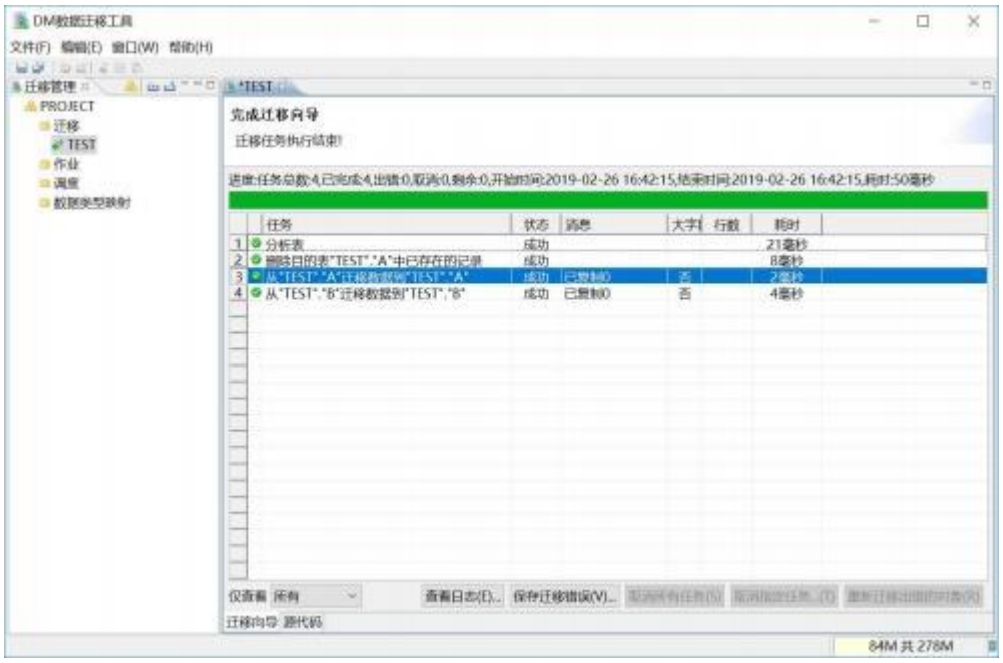
（5）选择迁移对象



（6）审阅迁移任务



（7）执行迁移任务



4.1.1 CURRENT_TIMESTAMP 字段类型的处理

MySQL 的字段可以定义 CURRENT_TIMESTAMP 类型，实际后台是触发器帮助更新行数据的时间戳，达梦可以用相同的触发器原理仿真，就是需要一些修改的工作量。

实现最后更新时间功能，即数据新建为新建时间，数据修改时自动更新为当前时间。

测试示例如下：

```
drop TABLE T1;
CREATE TABLE T1(
A INT NOT NULL,
B DATETIME(6) DEFAULT sysdate,
C CHAR(10),
PRIMARY KEY(A)
);

insert into t1(a,c) values (1,'a');
insert into t1(a,c) values (2,'b');

create or replace trigger tg_ 1
before update on t1 for each row
begin
```

```
:new.b=sysdate;
end;
/

select * from t1;
/*
A       B
1       2019-06-21 15:25:32.792000
2       2019-06-21 15:25:32.813000
*/
update t1 set c='aa' where a=1 ;
select * from t1;
/*
A       B
1       2019-06-21 15:26:02.758000
2       2019-06-21 15:25:32.813000
*/
```

4.1.2 数据迁移常见问题

4.1.2.1 无效的数据类型

请核查字段定义中的 int 类型，是否有精度，如果有，请直接去掉。即 int(10)需要改成 int，去掉精度。（达梦数据库的 int 类型，不需要、也不能设置精度）。

```
create table test1(v1 int(10),v2 varchar(20));
需要更改成
create table test1(v1 int,v2 varchar(20);
```

4.1.2.2 TIMESTAMP 类型

在 MySQL 中时间类型 TIMESTAMP 默认 default 设置为“0000-00-0000:00:00”，而在 DM 中 TIMESTAMP 类型数据不能为“0000-00-0000:00:00”,在 DM 中是不合法的,必须在'0001-01-0100:00:00.000000'到'9999-12-3123:59:59.999999'之间。

可通过修改直接修改 MySQL 中的业务表数据后进行数据迁移或通过 DTS

的使用 SQL 进行迁移数据，详见应对措施一。

4.1.2.3 DEFAULT 设置

对于字段的 default 设置，默认的字符或字符串默认值，需要添加单引号"，如 achar(10)default‘abc’。

4.1.2.4 TEXT 类型

对于 text 字段类型，若在多条 SQL 中存在对 text 字段类型的 distinct 处理，可根据具体情况决定是否将 text 字段类型改变为 varchar 类型，或对查询结果进行其他方式的过滤。

4.1.2.5 精度放大 3 倍

在 2019-9-18 以前的 DM 迁移工具中，如果 MySQL 源端的表字符集为 utf8，迁移到达梦的时候 CHAR、VARCHAR 类型的精度会自动放大三倍。如果不能接收这种精度的放大，有两个办法：

1．把 MySQL 的表定义导出成文本，手动修改为 DM 语法。

2．去官网下载 2019-9-18 以后的版本。

4.2 创建触发器、存储过程

对系统中的表和视图，可以使用达梦 DTS 工具进行迁移。如果还用到了自定义类型、存储过程、函数、触发器等，需要导出源端数据库上的定义，手动进行等价改写，可以参考后续步骤。

首先，可以先从 MySQL 导出用户对象 sql 脚本，MySQL 导出存储过程及函数的执行命令：

MySQLdump-hhostname-uusername-ppassword-ntd-Rdatabasename>prorandfunc.sql

然后，将导出的 sql 脚本，经人工修改之后，再导入到达梦数据库中运行。

4.2.1 常见函数处理

4.2.1.1 DATE_FORMAT()函数

以 date_format 函数为例，MySQL 中的 date_format 函数可以在 DM 数据库中使用 to_char 或 to_date 函数改写，改写后在达梦中可以达到相同的效果，示例如下：

```
--MySQL
select date_format(sysdate(), '%Y 年%m 月') from dual
-- DM
select translate(to_char(sysdate, 'yyyy-mm#'),'-#','年月') from dua

--MySQL
select DATE_ FORMAT(C_ FIRST_TIME,'%Y-%m-%d %H') FROM DUAL;
-- DM
select TO_CHAR(C_FIRST_TIME, 'YYYY- MM- DD HH24') FROM DUAL;
```

4.2.1.2 IF()函数

MySQL 中的 if()函数在当前测试版本 DM 中不支持，需采用 casewhen 进行替代，后续新版本可能会支持。

4.2.1.3 CONVERT()函数

DM 的 convert()函数中的 type 在前，value 在后，而 MySQL 数据库中 convert()函数则恰恰相反，对于 cast()函数的用法则一致，测试示例如下：

```
--MySQL
CONVERT(CASE WHEN TEMP_STA.c_data_value &lt; 0 THEN NULL ELSE TEMP_STA.c_data_value
END, SIGNED)    AS "ONLINEUSER",

-- DM
CONVERT( INTEGER,CASE       WHEN      TEMP_STA.c_data_value      &lt;0 THEN NULL
ELSE
```

```
TEMP_STA.c_data_value END)    AS "ONLINEUSER"
```

4.2.1.4 CAST()函数

DM 中的 cast()函数的用法虽然和 MySQL 一致，但使用效果存在不同，MySQL 中对于数值类型的 value 转 char 类型没有限制，DM 中则存在限制，会报“数据转换失败”等报错，可将 cast(数值类型的 valueaschar)转换为 cast(数值类型的 valueasvarchar),对于 unsigned 类型可以根据实际情况做对应改变或确定是否有必要进行转换。

4.2.1.5 FIND_IN_SET()函数

MySQL 字符串函数 find_in_set(str1,str2)函数就是查询字段(strlist)中包含(str)的结果，返回结果为 null 或记录，也就是返回 str2 中 str1 所在的位置索引，str2 必须以","分割开。同 like 相比，like 是广泛的模糊匹配，字符串中没有分隔符，而 Find_IN_SET 是精确匹配，字段值以英文”,”分隔，Find_IN_SET 查询的结果要小于 like 查询的结果。

而在 DM 中没有对应的函数，需要编写自定义函数，如下：

```
CREATE OR REPLACE
FUNCTION SYSDBA. FIND_ IN_SET(piv_str1 varchar2, piv_str2 varchar2, p_sep
varchar2 := ',')
RETURN NUMBER IS
l_idxnumber:=0; --  用于计算 piv_str2 中分隔符的位置
strvarchar2(500);    --  根据分隔符截取的子字符串
piv_strvarchar2(500) := piv_str2; --  将 piv_str2 赋值给 piv_str
resnumber:=0; --  返回结果
BEGIN
--  如果 piv_str 中没有分割符，直接判断 piv_str1 和 piv_str 是否相等，相等 res=1
IF instr(piv_str, p_sep, 1) = 0 THEN
IF piv_str = piv_str1 THEN
res:= 1;
END IF;
ELSE
--  循环按分隔符截取 piv_str
LOOP
```

```
l_idx := instr(piv_str,p_sep);
--  当 piv_str 中还有分隔符时
IF l_idx > 0 THEN
--  截取第一个分隔符前的字段 str
str:= substr(piv_str,1,l_idx- 1);
--  判断 str  和 piv_str1  是否相等，相等 res=1  并结束循环判断
IF str = piv_str1 THEN
res:= 1;
EXIT;
END IF;
piv_str := substr(piv_str,l_idx+length(p_sep));
ELSE
--  当截取后的 piv_str  中不存在分割符时，判断 piv_str 和 piv_str1 是否相等，相
等 res=1 IF piv_str = piv_str1 THEN
res:= 1;
END IF;
--  无论最后是否相等，都跳出循环
EXIT;
END IF;
END LOOP;
--  结束循环
END IF;
--  返回 res
RETURN res;
END FIND_ IN_SET;
```

4.2.1.6 GROUP_CONCAT 函数

对于 group_concat 函数，可以采用 DM 的 LISTAGG 函数进行替换，如下为 MySQL 和 DM 中的函数应用和效果，如下：

```
--MySQL
select    group_concat(ifnull(OPEN_STEP, '10') order by ID ASC separator '、、')
from ZHZB_DEV.BID_PACKAGE BP;

10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、
10、、10....

select    group_concat(ifnull(OPEN_STEP, '10') order by ID ASC separator '、、')
```

```
from ZHZB_DEV.BID_PACKAGE BP
group by AGENT_ ID;

10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、 10、、10、、10、、10 10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、、、、 10、、10、、10、、
10、、10、、10、、10、、10、、20、、20、、10、、10
-- DM
select LISTAGG(ifnull("BP"."OPEN_STEP", '10'),'、、') WITHIN GROUP(ORDER BY
"BP"."ID" ASC) from ZHZB_DEV.BID_PACKAGE BP;
10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、
10、、10、、10、、10、、10、、10、、10、、、10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、
10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、
10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、
10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、
10、、10、、10

select LISTAGG(ifnull("BP"."OPEN_STEP", '10'),'、、') WITHIN GROUP(ORDER BY
"BP"."ID" ASC) from ZHZB_DEV.BID_PACKAGE BP
group by "BP"."AGENT_ID" ;

--10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、
10、、10、、10、、10、、10
--10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、
10 、 、 10 、 、 10 、 、 10 、 、 10 、 、 10 、 、 10 、 、 10 、 、
10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、 10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、
10、、10、、10、、10、、10、、10、、
10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10、、10
```

4.2.1.7 DATE_ADD 函数

date_add 函数若对添加时间间隔的表达式进行求值，可采用 DM 的 TIMESTAMPADD 函数进行替代，例子如下：

```
--MySQL
select DATE_ADD(sysdate(), INTERVAL 1 YEAR);
--2020-07-02 11:24:18
-- DM
select TIMESTAMPADD(SQL_TSI_YEAR, 1,sysdate());
--2020-07-02 11:27:56.000000
```

4.2.1.8 其他函数

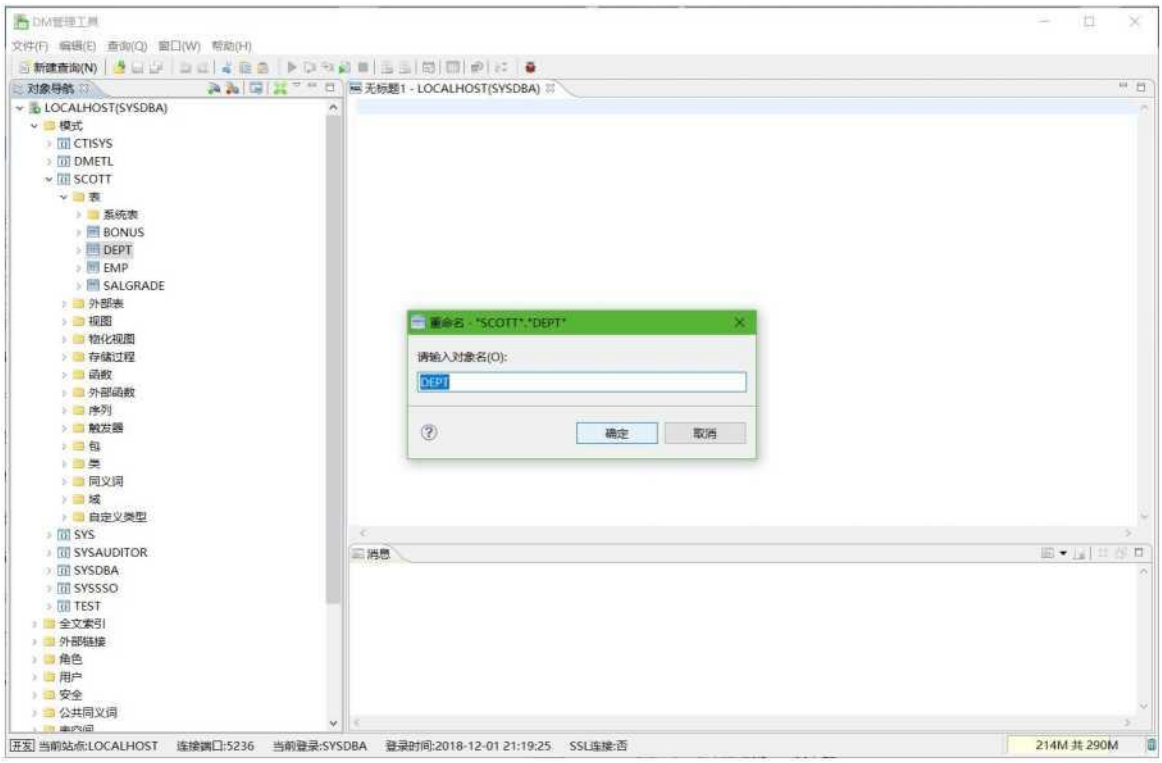
对于 MySQL 独有的函数或基于 MySQL 语法创建的自定义函数，在 DM 中需要基于 DM 的 SQL 语法进行改写，实现相同的功能。

4.2.1.9 时间间隔表达式

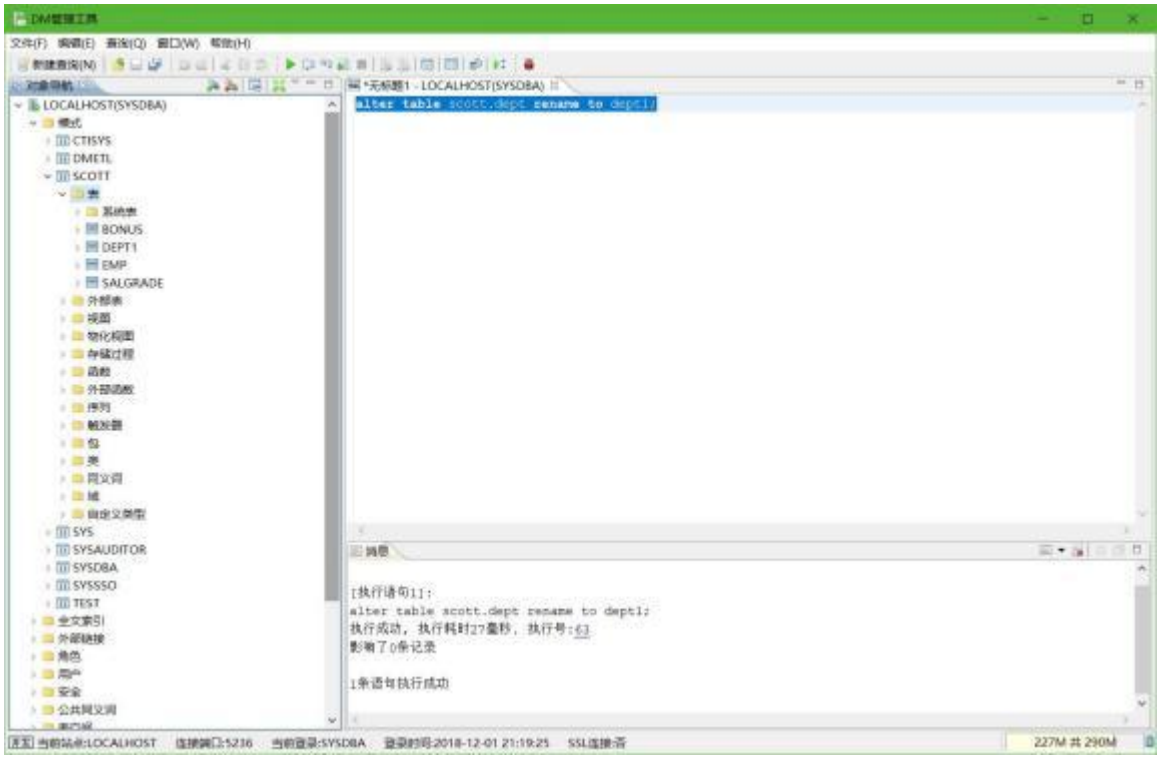
对于类似 date_format(("S"."CREATE_TIME" + interval 1 year),'%Y')、date_format((now() + interval -(1) year),'%Y'))的时间间隔表达式，对于 interval 关键词后的正数需添加单引号"，如 interval '1' year，对于负数，需改写，如 interval '-1'。

4.3 管理工具的基本使用

修改表名称等基本使用可以通过达梦数据库管理工具进行，在管理工具中选择表，右键-重命名，对表进行修改。



也可以通过 sql 语句进行修改，使用方式如下：



其他使用方式，通过达梦图形化客户端工具（manager.exe）都可以通过类似的方式实现。

5 核对数据库移植结果

- **统计达梦数据基础信息**

```
--统计页大小
select page;
--统计编码格式
select unicode;
--统计大小写敏感参数
select case_sensitive;
```

- **统计达梦数据中的对象以及表数据量**

a. 根据指定用户统计用户下的各对象类型和数目

```
Select object_type count* from all_objects where owner = 'OA8000_DM2015' group by
Object_type
```

b. 统计指定用户下所有的对象，并记录到新的记录表中

```
create table dm_objects(obj_owner varchar(100),obj_name varchar(100),obj_type
varchar(50));
insert into dm_objects select owner,object_name,object_type from
all_objects where
owner='OA8000_DM2015';
```

c. 统计每个表的数据量到表数据记录表

```
create table dm_tables(tab_owner varchar(100),tab_name varchar(100),tab_count int);

declare
begin
    for rec in (select owner,object_name from all_objects where owner='OA8000_DM2015'
and object_type='TABLE') loop
        execute immediate 'insert into dm_tables select ''' || rec.owner || ''',''' || rec.object_name || ''',count(*) from ' || rec.owner || '.' ||
rec.object_name;
    end loop;
end;

select * from dm_tables;
```

- **对比达梦数据库中对象和 MySQL 库中对象以及数据量差异**

比对表数据量，找出数据量不相等的表：

```
select a tab_owner a tab_name a tab_count-b tab_count from MySQL_tables a  dm_tables b
where a tab_owner=b tab_owner and a tab_name=b  tab_name
and a tab_count-b tab_count<>0
;
```

6 数据库移植完毕后的收尾工作

- 更新统计信息

数据核对完成无问题后， 应进行一次全库的统计信息更新工作。 统计信息更新脚本示例如下：

```
DBMS_STATS.GATHER_SCHEMA_STATS(
        'HNSIMIS',          -- HNSIMIS  为模式名
         100,
          FALSE,
        'FOR ALL COLUMNS SIZE AUTO');
```

更新统计信息的目的在于大批量迁移数据后，可能会导致数据库优化器根据错误的统计信息得到错误的查询计划，严重影响查询性能。

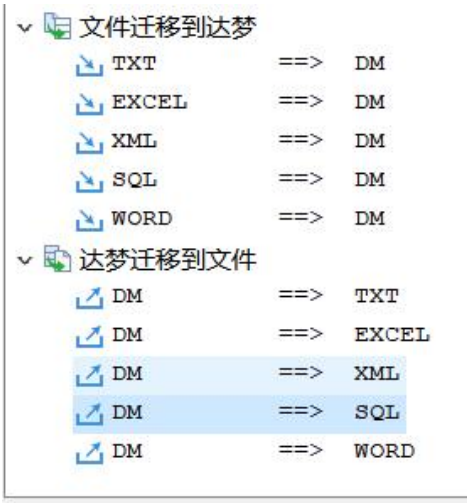
- 数据备份

再对数据更新完统计信息后，在数据量不大，磁盘空间足够的情况下应进行一次数据备份工作。数据备份有两种方式：正常停止数据库后，拷贝备份 data 文件夹；或者开启归档日志后，进行物理备份。

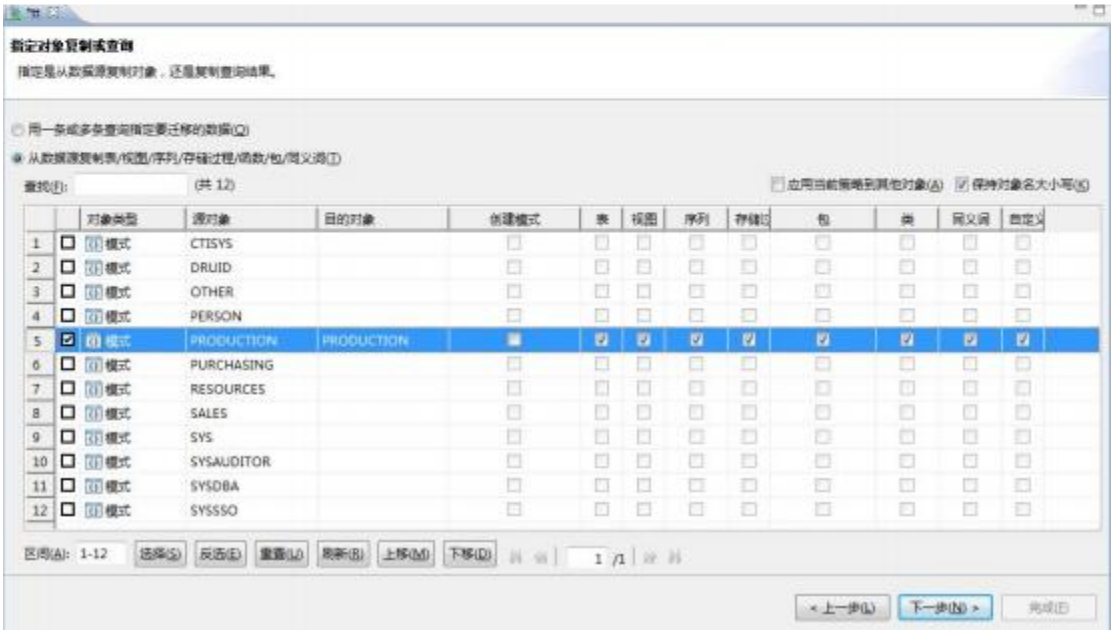
- 整理对象脚本

整理所有数据库对象脚本，这是为了对项目移植情况进行记录和备份，方便再次进行数据迁移。备份的数据库对象脚本包括：序列定义及当前值，表定义，索引定义，视图定义，函数定义，存储过程定义，包及包体定义、自定义类型和同义词定义。

脚本对象导出可通过达梦数据迁移 DTS 工具来进行。执行步骤如下：


文件迁移到达梦
TXT ==> DM
EXCEL ==> DM
XML ==> DM
SQL ==> DM
WORD ==> DM
达梦迁移到文件
DM ==> TXT
DM ==> EXCEL
DM ==> XML
DM ==> SQL
DM ==> WORD


文件
请输入SQL脚本文件信息。
定义脚本文件(B): d:\test.sql
浏览(W)...
数据脚本文件:
使用对象定义脚本文件(S)
对每个表使用单独的数据脚本文件(I)
所有数据脚本使用一个单独的文件(O)
文件编码(E): GBK
迁移选项:
仅迁移对象定义(A)
仅迁移数据(D)
迁移对象定义和数据(C)
定义脚本:
可执行(G)
不可执行(H)


指定对象复制或查询
指定是从数据源复制对象，还是复制查询结果。
用一条或多条查询指定要迁移的数据(Q)
从数据源复制表/视图/序列/存储过程/函数/包/同义词(T)
查找(F):
(共 12)
应用当前策略到其他对象(A)
保持对象名大小写(K)
对象类型
源对象
目的对象
创建模式
表
视图
序列
存储过
包
类
同义词
自定义
1 模式 CTISYS
2 模式 DRUID
3 模式 OTHER
4 模式 PERSON
5 模式 PRODUCTION PRODUCTION
6 模式 PURCHASING
7 模式 RESOURCES
8 模式 SALES
9 模式 SYS
10 模式 SYSAUDITOR
11 模式 SYSDBA
12 模式 SYSSSO
区间(A): 1-12
选择(S)
反选(E)
重置(U)
刷新(R)
上移(M)
下移(D)
1 /1
< 上一步(L)
下一步(N) >
完成(F)

后面直接点“下一步”即可。

7 应用移植

应用移植一般包括：需修改连接串、URL、数据库方言、SQL 语句、配置 JNDI。

数据库迁移到达梦数据库之后，应用需要对应适配迁移到达梦数据库中，针对应用移植到达梦数据库采用以下修改方式：

URL 连接串修改方式如下：

```
//      定义 DM JDBC   驱动串
String jdbcString = "dm.jdbc.driver.DmDriver";
//      定义 DM URL 连接串
String urlString = "jdbc:dm://localhost :5236";
```

数据库方言配置方式如下：

```
<properties>
<property  name="hibernate.dialect"  value="org.hibernate.dialect.DmDialect"  />
<property   name="hibernate.connection.driver_class"
    value="dm.jdbc.driver.DmDriver" />
 <property name="hibernate.connection.url" value="jdbc:dm://localhost :5236" />
<property name="hibernate.connection.username" value="TEST" />
<property name="hibernate.connection.password" value="123456789" /></properties>
```

如果数据库是集群，则需要配置 dm_svc.conf，配置内容如下：

```
TIME_ZONE=(480)
LANGUAGE=(cn)
SHRSG=(10.212.133.11:5236,10.212.133.12:5236)
```

如果应用程序对应的操作系统是 window，则将 dm_svc.conf 放在 C:\Windows\System32 目录下；如果操作系统是 linux，则放在/etc/目录下。

7.1 应用移植前的说明

7.1.1 关于 SQL 连接的一个说明

MySQL 和 DM 都支持 join 的常用的几种连接方式。

首先，MySQL 不支持全外连接，需采用左外连接和右外连接做 unionall 的

方式实现，这里反过来说，就是其他数据库（包含达梦）是支持全连接的，如下所示：

```
--DM 全外连接的写法：
drop table test2;

create table test2 (v1 int,v2 int);
insert into test2 values (1,1);
insert into test2 values (3,2);

select a.*,b.* from test2 a full join test2 b on a.v1=b.v2;
结果集：    （ 一条匹配的，匹配不上的，左表和右表的均显示出来）

1  1  1  1
      3  2
3  2



--但是对应 sql 在 MySQL 会报错，需要改写成这样：
select a.*,b.* from test2 a left join test2 b on a.v1=b.v2 union
select a.*,b.* from test2 a right join test2 b on a.v1=b.v2;
```

在迁移时，如果我们看到 MySQL 中类似的 union 不改写，对于业务逻辑是没有影响的，语法上也不会报错，但是如果放过来，把一个 union 的两个 select，改成成一个 select，效率会有很大的提升。

其次，是 Join 的一个语法需要改写，MySQL 支持 a join b 不带 On 的写法，非 MySQL 不行，需要进行对应改写，改写方法为去掉 join 即可，如下：

```
--MySQL 的写法：
create table test1 (v1 int);

select a.v1 from test1 a join test1 b;

-- DM 的写法：
create table test1 (v1 int);

select a.v1 from test1 a ,test1 b;
```

7.1.2 其他

MySQL 在语法方面允许很多不是常规的用法存在，也允许有一些不严谨的地方存在，如果应用系统构建时规避了一些不常规的用法，那么移植起来会比较顺手。也就是说，MySQL 自身的“行为方式”本身不太正统，比如大小写不敏感、非 SQL 标准的语法支持、SQL 二义性等。如果要在迁移到其他数据库后，保证应用没问题，需要注意。

对于遇到的其他问题，如多表的联合查询语法、含有特殊函数的 SQL 语句等，可以根据实际情况进行解决，MySQL 对于外键等对象的处理逻辑同 DM 存在诸多不同，需因地制宜。

7.2 JAVA 接口

7.2.1 驱动包说明

达梦 JDBC 驱动分为 DMJdbcDriver14、DMJdbcDriver15、DMJdbcDriver16，分别对应 Jdk1.4、Jdk1.5、(Jdk1.6\jdk1.7)；

达梦 dialect 驱动分为:

DmDialect-for-hibernate2.0.jar

DmDialect-for-hibernate2.1.jar

DmDialect-for-hibernate3.0.jar

DmDialect-for-hibernate3.6.jar

DmDialect-for-hibernate4.0.jar

分别对应 hibernate2.0、2.1、3.0、3.6、4.0；

达梦提供不同 hibernate 和 jdk 版本的方言包，用户可根据开发环境选择对应的方言包版本，相关方言包驱动在如下目录;

Jdbc 驱动包路径：\%DM_HOME%\jdbc\

dialect 驱动包路径：\%DM_HOME%\jdbc\dialect\

7.2.2 修改配置

//定义 DM JDBC 驱动串

StringjdbcString = "dm.jdbc.driver.DmDriver";

//定义 DM URL 连接串

StringurlString = "jdbc:dm://SHRSG?comOra=true";--集群的服务名连接方式

//StringurlString = "jdbc:dm://127.0.0.1:5236";--单机连接方式

//如果使用了 hibernate，需要进行 dialect 的配置修改，如下：

hibernate.dialect = org.hibernate.dialect.DmDialect

注：

上面 URL 连接串中的各连接属性及取值范围列表如下：

属性名称	说明	是否必须设置
“HOST”	主机地址，包括 IP 地址、localhost 或者配置文件中主机地址列表对应的变量名，如 dm_svc.conf 中的’SHRSG’	是
“PORT”	端口号，服务器登录端口号	否
“user”	登录用户	是
“password”	登录密码	是
“socketTimeout”	套接字超时时间，默认 0	否
“continueBatchOnError”	批量操作出错时，是否继续，默认 true；取值（true/True，false/False）	否
“escapeProcess	是否进行语法转义处理，默认 true；取值（true/True，false/False）	否
“autoCommit”	是否自动提交，默认 true；取值（true/True，false/False）	否
“maxRows”	批量操作最大行数，默认 0	否
“RowPrefetch”	预取行数，默认 10	否
“LobMode”	Lob 模式，默认 1；取值（1 分批缓存到本地,2 一次将大字段数据缓存到本地）	否
“stmtPoolSize”	语句句柄池大小，默认 0	否

“comOra”	是否与 oracle 兼容，默认 false；取值（true/True，false/False）	否
“ignoreCase”	是否忽略大小写，默认 true，取值（true/True，false/False）	否
“infoLevel”	是否屏蔽执行语句是抛出的异常，默认 1，取值（1 屏蔽，0 不屏蔽）	否
“alwayseAllowCommit”	在自动提交开关打开时，是否允许手动提交回滚	否
“batch Type”	批处理类型，默认 1，取值（1 进行批量绑定 2 不进行批量绑定）	否
“maxCachedPstmtSize ”	缓存准备执行语句句柄的数量，默认 0	否
“traceLeval ”	追踪级别，默认 0（暂未实现）	否
“appName”	客户端应用程序名称	否
“session Timeout”	会话超时时间，默认 0	否
“isCompress”	是否压缩消息，默认 false，取值（true/True ,false/False）	否
“sslFilePath”	指定 ssl 加密文件的路径	否
“sslKeystorePass”	指定 ssl 加密文件的指令	否
“resultSetType”	指定默认创建结果集类型，1003-readonly，1004-unsensitive，1005-insensitive，默认 1003	否
“kerberosLoginConfPath”	Kerberors 认证登录配置文件路径	否

7.2.3 示例程序

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class jdbc {

//定义 DM JDBC  驱动串
String driver= "dm.jdbc.driver.DmDriver";
//定义 DM URL 连接串
```

```
String url= "jdbc:dm://SHRSG?comOra=true";
//定义连接用户名
String username="TEST";
//定义连接用户口令
String password="123456789";
//定义连接对象
Connection conn = null;

//  加载 JDBC 驱动程序
public void loadJdbcDriver() throws Exception{

    try {
              System.out.println("Loading DM Driver...");
              Class.forName(driver);
        }
        catch (Exception e) {
        //打印异常信息
        e.printStackTrace();
        }
}
//连接数据库
public void connect() throws SQLException {
     try{
            System.out.println("Connecting to DM Server...");
            conn = DriverManager.getConnection(url,username,password);
          }
          catch(Exception e){
          //打印异常信息
          e.printStackTrace();
        }
  }

//关闭数据库连接
public void disConnect() throws Exception{
    try {
          System.out.println("Closing DM Server...");
          conn.close();
        }
    catch (SQLException e) {
```

```
    //打印异常信息
      e.printStackTrace();
      }
  }
//创建表 test
public void createTableTest() throws Exception{
    try{
          String sql_1 = "create table test(c1 int, c2 TEXT);";
          Statement stmt = conn.createStatement();
          stmt.executeUpdate(sql_1);
        }
        catch(Exception e){
        e.printStackTrace();//打印异常信息

      }
    }

//执行插入语句
public void insert() throws SQLException{
        try{
              String sql = "INSERT INTO test VALUES(?,?)";
              PreparedStatement pst = conn.prepareStatement(sql);
              pst.setInt(1, 1);
              pst.setString(2,"test");
              pst.executeUpdate();
          }
          catch(Exception e){
              e.printStackTrace();
    }
    System.out.println("Closing insert...");
  }

//执行 SQL 语句
public void exec_sql() throws Exception{
      Statement stmt=conn.createStatement();
      //查询 test1 表，返回结果集
          String sql = "select * from test where c1 = 1";
          ResultSet rs = stmt.executeQuery(sql);
          System.out.println(sql);
```

```
 /*
       String a = "''";
       ResultSet rs = null;
       if(a=="''"){
           String sql = "select * from test where id = ?";
           PreparedStatement pst = conn.prepareStatement(sql);
           pst.setString(1, a);
           pst.execute();
           System.out.println(sql);*/


     while (rs.next())
           {
               System.out.println(rs.getInt("c1")+rs.getString("c2"));
             }
         rs.close();
         stmt.close();
       }
public static void main(String[] args) throws Exception {
     try{
           // TODO Auto-generated method stub
             jdbc jc = new jdbc();
             jc.loadJdbcDriver();
             jc.connect();
             jc.createTableTest();
             jc.insert();
             jc.exec_sql();
             jc.disConnect();
           }
           catch(Exception e){
             //打印异常信息
               e.printStackTrace();
         }
     }
 }
```

7.3 DOTNET 接口

.NETDataProvider是.NETFramework编程环境下的数据库用户访问数据库的

编程接口，用于连接到数据库、执行命令和检索结果。在数据源和代码之间创建了一个最小层，以便在不以功能为代价的前提下提高性能。

使用 OCI 连接达梦数据库，需要用到 DM 的文件列表（在\%DM_HOME%\bin 目录下可以找到）如下：DmProvider.dll，DmProvider.dll。

7.3.1 数据类型

.NETFramework 在 System.Data.DbType 中定义了.NETFramework 数据提供程序的字段、属性或 Parameter 对象的数据类型，DM 支持的数据类型的对应关系如下表所示：

序号	DbType 类型	DM 类型
1	AnsiString	VarChar
2	AnsiStri ngFixed Length	VarChar
3	Binary	Binary
4	Boolean	Bit
5	Byte	Byte
6	Currency	Decimal
7	Date	Date
8	DateTime	DateTime
9	Decimal	Decimal
10	Double	Double
11	Guid	VarChar
12	Int16	Int16
13	Int32	Int32
14	Int64	Int64
15	Object	Blob
16	SByte	SByte
17	Single	Float
18	String	VarChar

19	StringFixed Length	Char
20	Time	Time
21	UInt16	UInt16
22	UInt32	UInt32
23	UInt64	UInt64
24	VarNumeric	XDEC

7.3.2 示例程序

```
using System;
using System.Collections.Generic;
using System.Text;
using Dm;
namespace DMDemo
{
class Demo
{
//返回结果
static int ret = 1;
static DmConnection cnn = new DmConnection();
[STAThread]
static int Main(string[] args)
{
try
{
cnn.ConnectionString = "Server=localhost; User Id=SYSDBA; PWD=SYSDBA";
cnn.Open();
Demo demo = new Demo();
demo.TestFunc();
cnn.Close();
}
catch(Exception ex)
{
Console.WriteLine(ex.Message);
}
```

```
Console.Read Line();
return ret;
}
public void TestFunc()
{
DmCommand command = new DmCommand();
command.Connection = cnn;
try
{
string a, b, c;
command.CommandText = "SELECT NAME, AUTHOR, PUBLISHER FROM
PRODUCTION.PRODUCT;";
DmDataReader reader = command.ExecuteReader();
while(reader.Read())
{
a = reader.GetString(0);
b = reader.GetString(1);
c = reader.GetString(2);
Console.WriteLine("NAME：   " + a);
Console.WriteLine("AUTHOR：   " + b);
Console.WriteLine("PUBLISHER：   " + c);
Console.WriteLine("------------------");
}
}
catch(Exception ex)
{
Console.WriteLine(ex.Message);
ret = 0;
}
}
}
}
```

8 系统测试与优化

数据库和应用系统移植完毕后开启 sql 日志，对系统进行全面测试，排除移植过程中错误的地方，对慢的 sql 语句进行优化，开启日志的方法如下：

```
--设置 SQL 过滤规则，记录全部的 SQL
```

```
SELECT SF_SET_SYSTEM_PARA_VALUE('SQL_TRACE_ MASK','1',0, 1);
--排查错误的时候也可以设置过滤规则为 23，只记录报错的 SQL
SELECT SF_SET_SYSTEM_PARA_VALUE('SQL_TRACE_ MASK','23',0, 1);
--开启 SQL 日志：
SP_SET_ PARA_VALUE(1, 'SVR_ LOG', 1);
```

在功能测试和性能测试的时候可以开启 SQL 日志，然后通过日志分析工具从执行时间和执行次数两个维度对 SQL 日志进行分析，生产分析结果，然后根据分析结果对系统性能进行优化。

使用日志分析工具时最好采用 32k 页大小的 DM 作为分析库。