DM DBA 手记之 SQLServer 移植到 DM

1 概述

随着国家对自主可控的日益重视，目前在党政机关、军队、大型央企等行业和区域中面临越来越多的国产化。

DM 数据库和 SQLServer 结构存在差异，TSQL 语法也存在差异。DM 数据库针对 SQLServer 做了部分兼容性。但由于有根本性的差异，兼容度不高。从 SQLServer 迁移到 DM 数据库，DM 数据库提供了自动迁移工具，但在开发级别还需要人工干预进行同步。

SQLServer 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 TSQL 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 移植过程

2.1 待移植系统分析

应用后台操作系统	WINDOWS 2008
数据库后台操作系统	WINDOWS 2008
后台数据库	SQLServer
应用开发平台	.NET
应用开发接口	.Net Provider
需要移植的数据库对象	表（数据量） 视图 触发器 存储过程、函数

对待移植系统进行分析，确定需要移植的数据库对象，给出移植列表，给用户确认，作为移植的依据，给出 SQLServer 的统计脚本。

2.1.1 统计 SQLServer 数据库基础信息

```
--统计编码格式
SELECT COLLATIONPROPERTY ( 'Chinese_PRC_Stroke_CI_AI_KS_WS', 'CodePage')
--936  简体中文 GBK
--950  繁体中文 BIG5
--437  美国/加拿大英语
--932  日文
--949  韩文
--866  俄文
--65001 unicode UFT-8
```

2.1.2 统计 SQLServer 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 type,COUNT(*) from sys.all_objects where schema_id=1 and
parent_object_id=0 and is_ms_shipped=0 group by 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 sql_objects(obj_owner varchar(100),obj_name
varchar(100),obj_type varchar(50));
insert into sql_objects select DB_NAME(),name,type from sys.all_objects
where schema_id=1 AND type IN ('U','V','FN','P','TR');
select * from sql_objects;
```

C.统计每个表的数据量到表数据记录表

```
create table sql_tables(tab_owner varchar(100),tab_name
varchar(100),tab_count int);
insert into sql_tables
```

```
select DB_NAME(),a.name, b.rows
from sysobjects AS a INNER JOIN sysindexes AS b ON a.id = b.id
where (a.type = 'u') AND (b.indid IN (0, 1))
order by a.name,b.rows desc;
select * from sql_tables;
```

2.2 准备移植环境

本节讨论的内容是关于对移植环境的准备工作，鉴于移植工作的最终目的可能不同，我们需要对目的做一下分类，分类之后，可以更好的明确我们的环境准备工作的需求，从而使移植的工作更加的高效；

（1）仅做移植兼容性测试。这里指的是用户或者开发商对与移植可能性和技术工作量的一个评估和确认工作，也就是尝试性的移植，移植后可能并不会立刻进行产品级的应用功能、性能、稳定性测试，在这中情况下，我们一般搭建最基础的移植环境即可，用虚拟机和物理机服务器都可以进行，且对配置无特别要求，满足基本运行条件即可；

（2）为替换 SQLServer 上线运行进行正式移植。在这种情况下，移植完成后，会对应用进行产品级全访问的功能点测试、性能测试、压力测试以及稳定性测试等标准流程的测试，在这种情况下搭建移植环境，一定要优先采用物理服务器搭建，并且对于物理服务器的相关硬件配置要提出要求，提出要求的配置根据系统数据量规模、性能要求、并发规模、可用性 要求等基本情况向测试方提出建议，因为要想移植后测试效果好硬件的支撑是必不可少的， 如果是在很低配置情况下，又想得到很好的移植和测试效果，这本身就是不合理的。

对于服务器配置有要求，包括 CPU、内存、OS、磁盘（本地盘、阵列），架构方面（单机、多机）包括采用的集群架构方式。

2.2.1 DM 移植环境

- **选择合适的版本**

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，再进行项目移植

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。

- **选择合适的初始化参数**

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 SQLServer 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，建议设置页大小为 8KB，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，建议该参数设置为 16 或者 32。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，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，使用大小写不敏感以便和原来系统匹配。

（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。

- **合理配置 INI 参数**

DM 的 INI 参数文件中针对从 SQLServer 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 标佳，2：兼容 ORACLE，3:兼容 MSSQLSERVER，4:兼容 MYSQL，5:兼容 DM6，6:兼容 Teradata，所以当从 SQLServer 移植到 DM 时，修改值移植到 DM 时，修改值为 3；
MS_PARSE_PERMIT	是否支持 MSSQLSERVER 的语法。 0:不支持；1:支持； 2：在 MS_PARSE_PERMIT=1 的基础上，兼容 MSSQLSERVER 的查询项中支持"标识符=列名"或"@变量名=列名"用法。当 COMPATIBLE_MODE=3 时，MS_PARSE_PERMIT 的实际值为 1

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。

- **创建用户和表空间**

从 SQLServer 移植到 DM，要求必须创建用户自己的用户模式名和表空间，不要把数据移植到系统默认的管理 SYSDBA 用户下和 MAIN 表空间下。

首先需要分析本次移植 SQLServer 源库需要移植的是哪一个或者哪几个库的数据，针对不同的库，对应达梦数据库中不同的用户，然后分别创建这些需要移植的用户的独立表空间；移植设计过程中，最好设计达梦数据库用户名称和 SQLServer 中库名相同，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。

创建属于各个表空间的用户及模式。便于数据的导入导出，同时增加数据隔离性，在遇到问题的时候，更容易分析问题数据所在的位置。

2.2.2 SQLServer 移植环境

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

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 SQLServer 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 SQLServer 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。

工具准备。这里推荐 SQLServer 自带的 SQLServerManagementStudio 工具，一般需要有一个工具来连接到需要移植的 SQLServer 环境，以便进行移植数据的确认和初步的分析，当然也可以使用外部第三方的工具，SqlDbx 是一个第三方工具，轻量简便，使用起来比较方便。

2.3 常规对象及数据迁移

常规对象指的是表和视图，都可以通过达梦提供的数据迁移工具从 SQLServer 完整的迁移到达梦数据库。

2.3.1 制定迁移计划

根据 2.1 节分析的情况，制定迁移计划：

（1）选择合理的迁移顺序：先迁移序列、再迁移表、最后迁移视图。

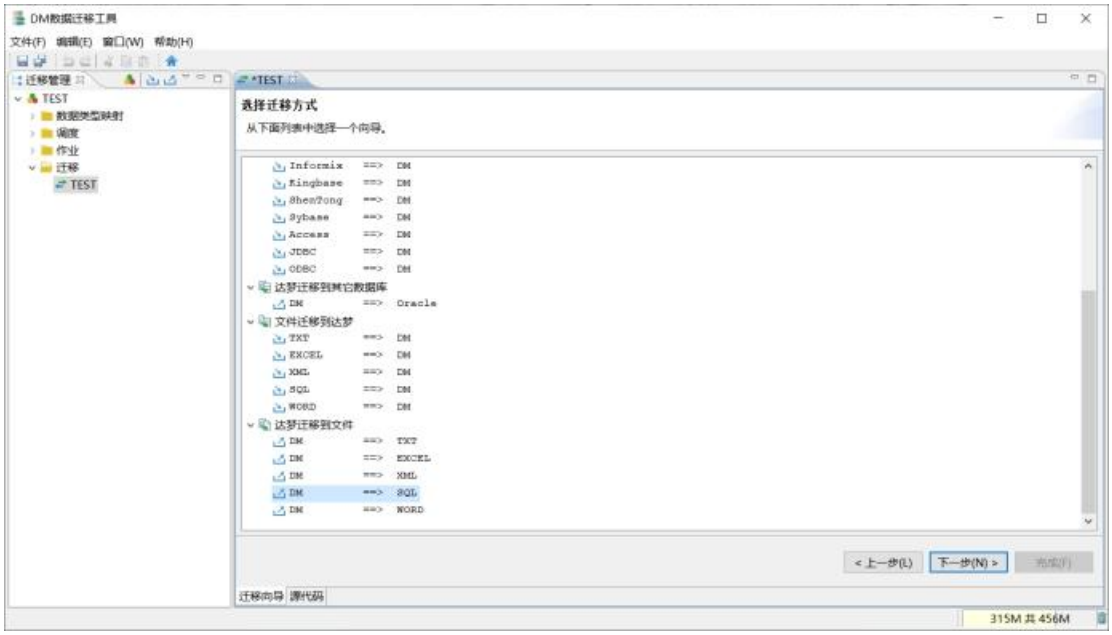
（2）对于数据量大的表单独迁移。

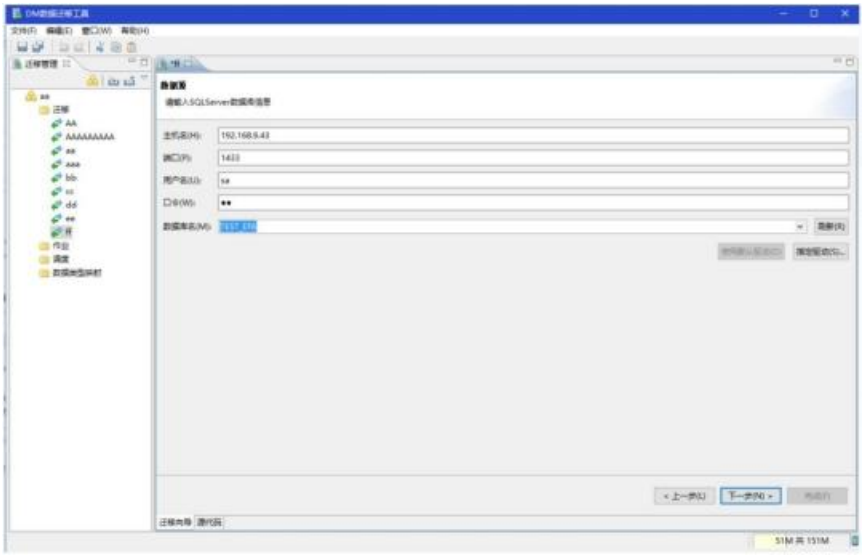
（3）对于分区表如果数据量没有超过 1 亿建议迁移成普通表，在分区列上创建索引对于大字段较多的表，需要修改批量的行数，以免造成迁移工具内存溢出。

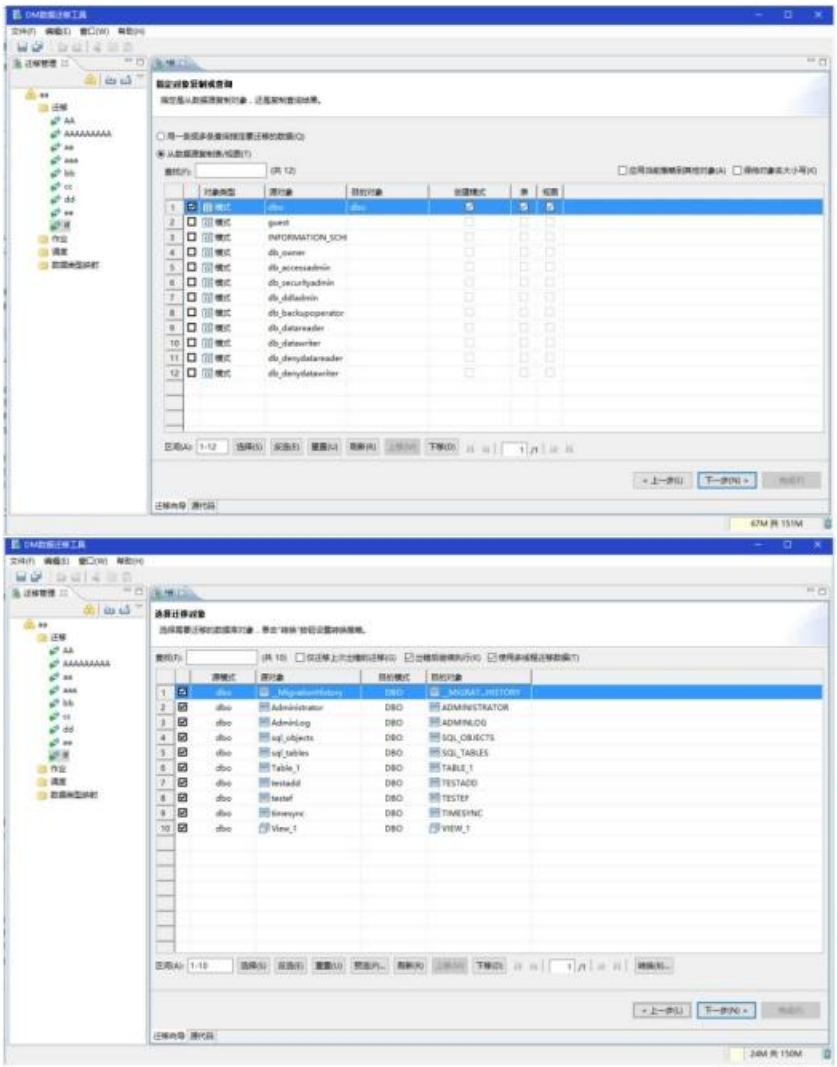
2.3.2 表对象迁移

- **一次性迁移**

对于表比较少，数据量不大的系统，可以通过 DTS 采取一次性迁移，全部选中即可。

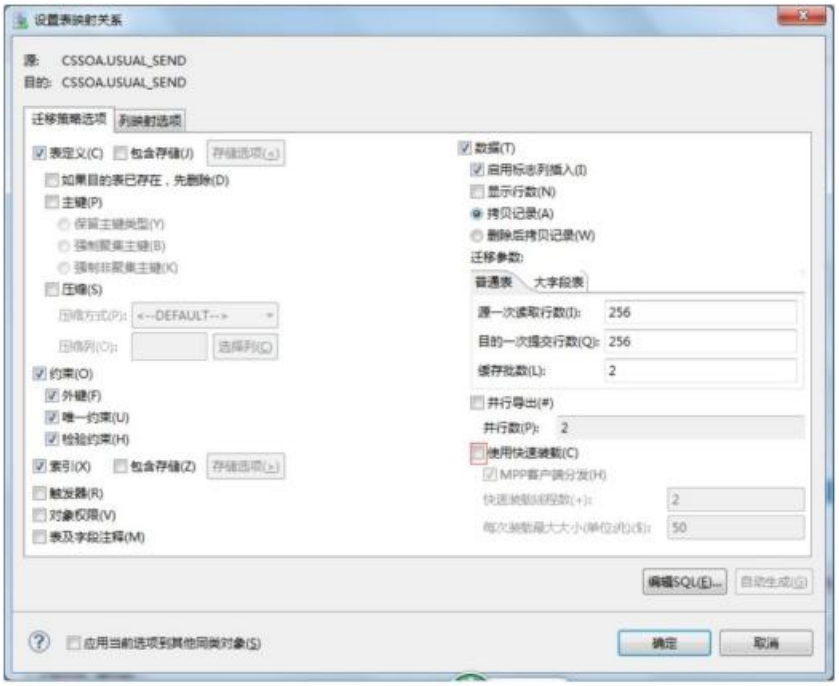






- **分批次迁移**

对于表比较多，数据量大的系统，建议先迁移小表再进行大表的迁移，迁移时最好不用快速装载功能。

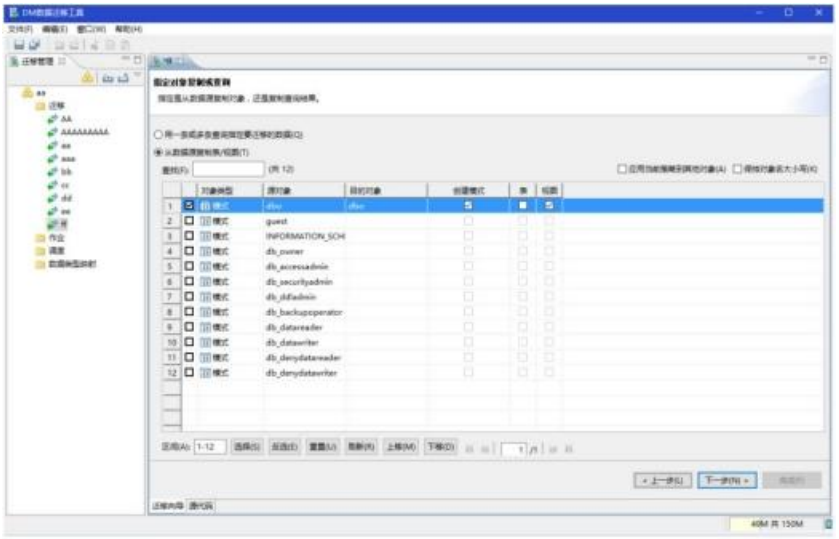


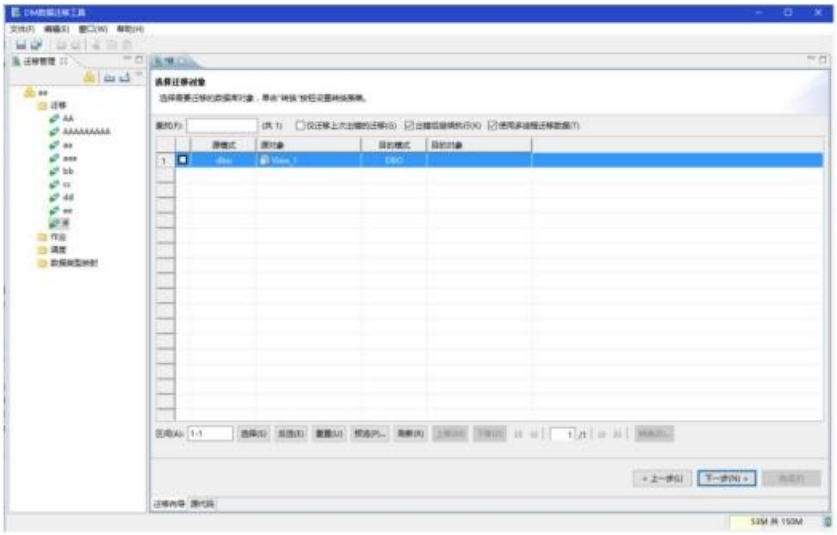
2.3.3 视图对象迁移

- **普通视图对象迁移**

（1）使用 DTS 工具迁移视图，此方法适用于批量迁移视图对象。DTS 工具使用方法可参考帮助-帮助主题。

（2）从源 SQLServer 中获取视图定义，在目的库手动创建视图，此方法适用于所需迁移对象较少，或者对方法一中迁移出错的视图单独处理。





- **索引(物化)视图对象迁移**

从源 SQLServer 中获取物化视图定义，在目的库手动创建物化视图。

- **视图对象迁移可能遇到的问题及注意点**

（1）视图查询对象依赖迁移及权限授予。由于视图查询依赖于相关的表或者其他数据库对象，在迁移视图之前需要首先迁移视图所依赖的对象，并授予视图用户相关对象的权限。

（2）目的端物化视图刷新方式支持及设置。由于目的端 DM 视具体架构（单库、MPP）的不同存在对物化视图日志表的支持程度的差异，迁移物化视图时，需要视目的端架构，考虑是否变更物化视图刷新方式（增量刷新改为全量刷新）。

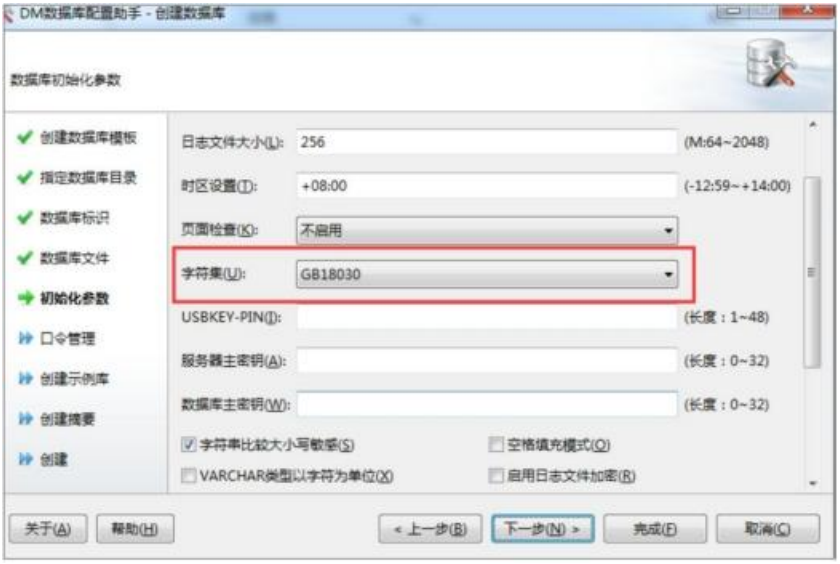
2.3.4 处理迁移过程中错误

- **字符串截断**

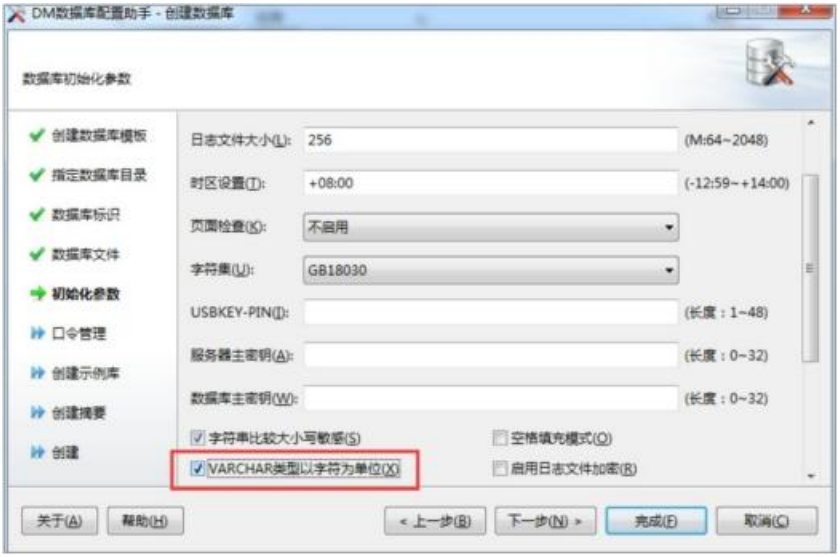
从 SQLServer 迁移到 DM 的时候，出现字符串截断的一般都是字段中含有中文，出现这种问题是因为 DM 初始化的时候选择的字符集是 Unicode（即 utf-8），该字符集的国际标准是一个汉字占 3 个字节，而 SQLServer 中默认情况下一个汉字占 2 个字节，此时迁移的时候就会报下面的错误：

遇到该错误有 3 种解决办法。

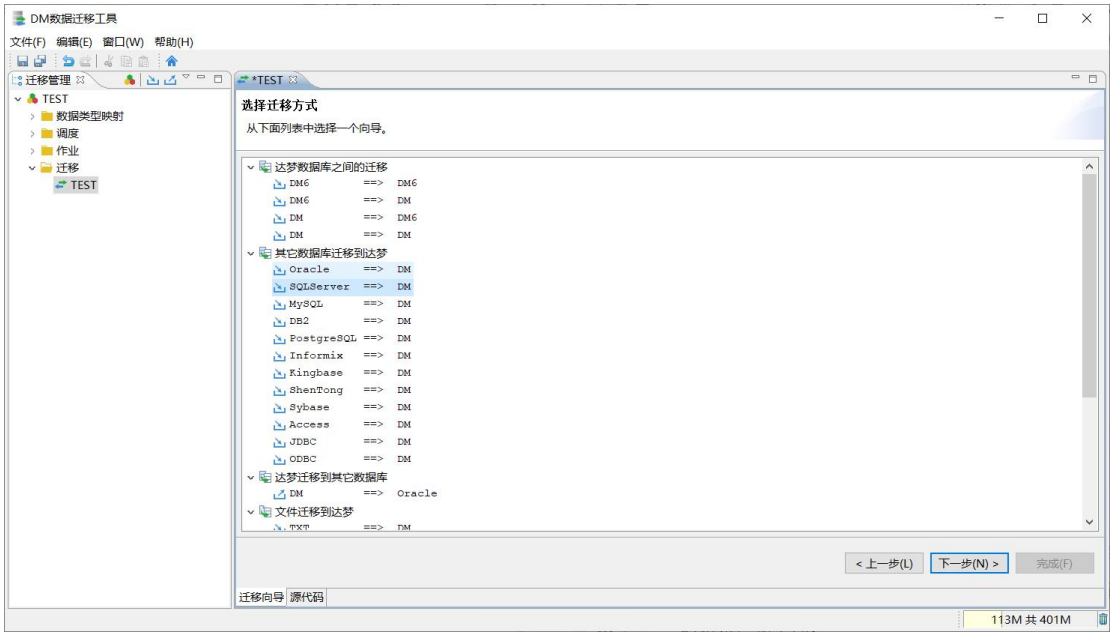
（1）是在初始化的时候，字符集选择 gb18030，如下图：



（2）是选择 VARCHAR 类型以字符为单位，如图：



（3）因为前面 2 种都需要重新初始化数据库，第三种不需要重新初始化数据库即可解决，即在选择迁移方式的时候，选择字符长度映射关系为 2，如下图：



- **记录超长**

DM 在初始化的时候，选择的页大小影响后面表每行数据的长度，表每行的长度之和（普通数据类型）不能超过一页大小，如果超过 1 页大小即报记录超长的错误，如下图：

解决方法：

（1）找到表中 varchar 类型比较长的（如 varchar（8000）这种），修改成 text 类型；

（2）初始化的时候页大小选择 32k。（对于表中 varchar2 类型较长，并且字段较多的情况不太适合，这种情况采用方法 1 解决）

- **违反唯一性约束**

这种情况是因为表中设置了唯一性约束或者主键约束，但是数据中有重复记录造成的。这种情况有可能是原始库的约束被禁用了，或者数据重复迁移造成的。

解决办法：在确定源数据没有问题的情况下，迁移的时候选择删除后再拷贝，如下图：

在迁移界面中，先中要迁移的表，然后点击转换。

在弹出的窗口中选择删除后拷贝，如下图：

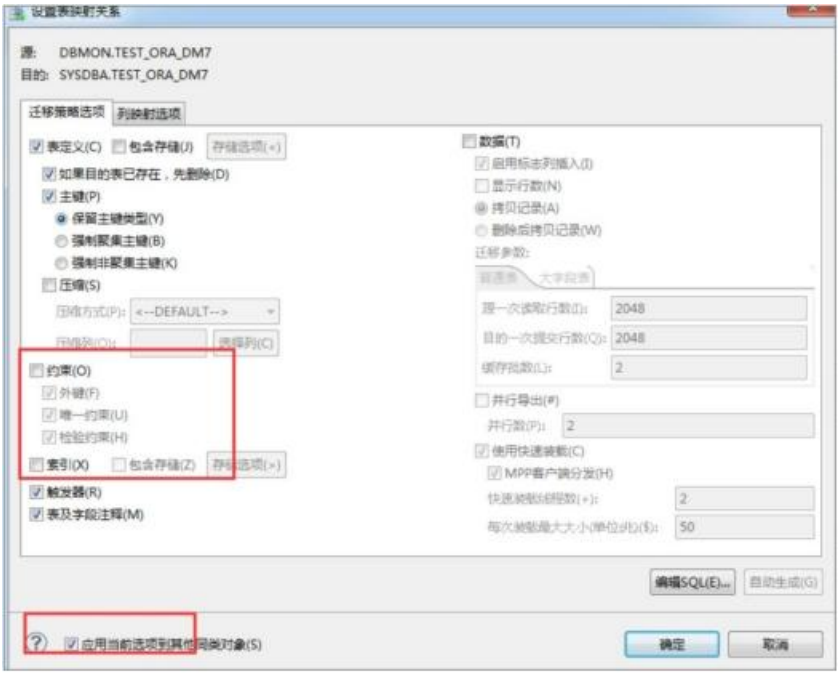
这样就可以迁移成功。

- 违反引用约束

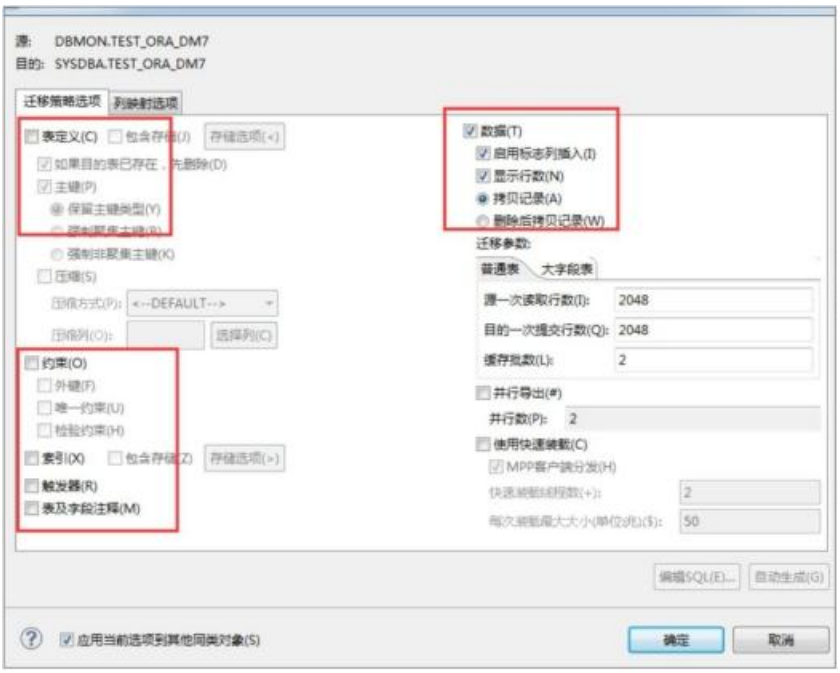
这种问题主要是由外键约束造成的，父表的数据没有迁移，先迁移了子表的数据，错误如图所示：

解决办法：迁移的时候先不迁移外键等约束，在选择好要迁移的表时，点击转换，按照下面步骤迁移。

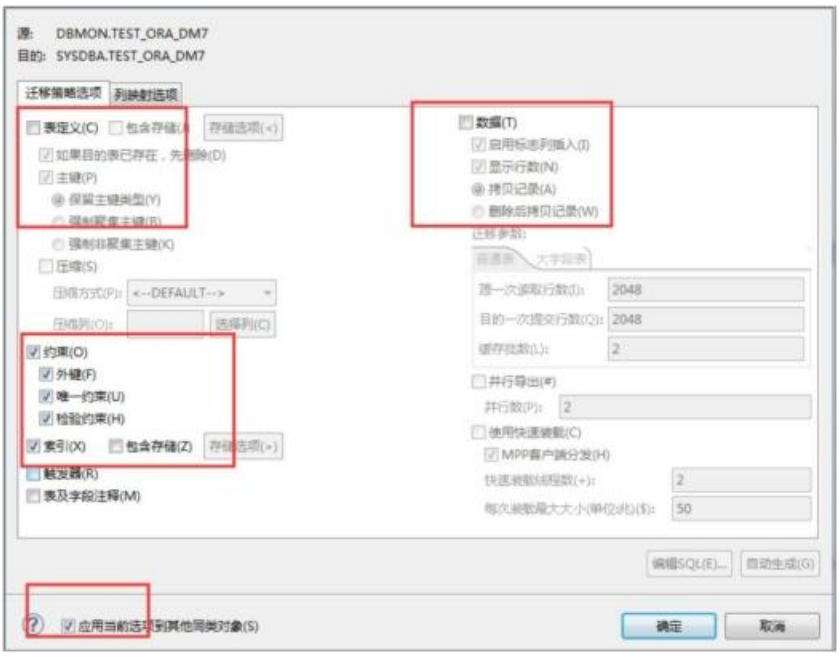
（1）第一次只选择表定义，不选择约束等，如图：



（2）第一次迁移完成后（确保没有错误），第二次只选择数据，如图：



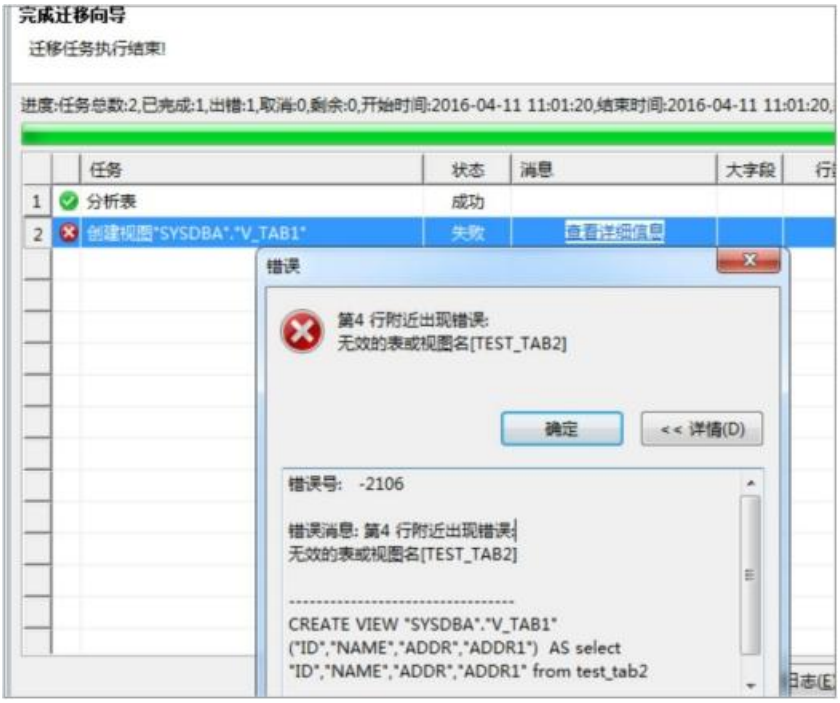
（3）第三步选择约束、索引等，如图



按照上面三步迁移，基本上都可以顺利迁移成功。

- **视图迁移过程中顺序问题：无效的用户对象**

这个问题一般是因为在迁移视图之前，没有将视图依赖的表迁移过去，如图所示：



解决办法：严格按照迁移的顺序，先迁移表，然后再迁移视图、存储过程、函数等的顺序迁移即可。

错误码及错误描述信息的对应

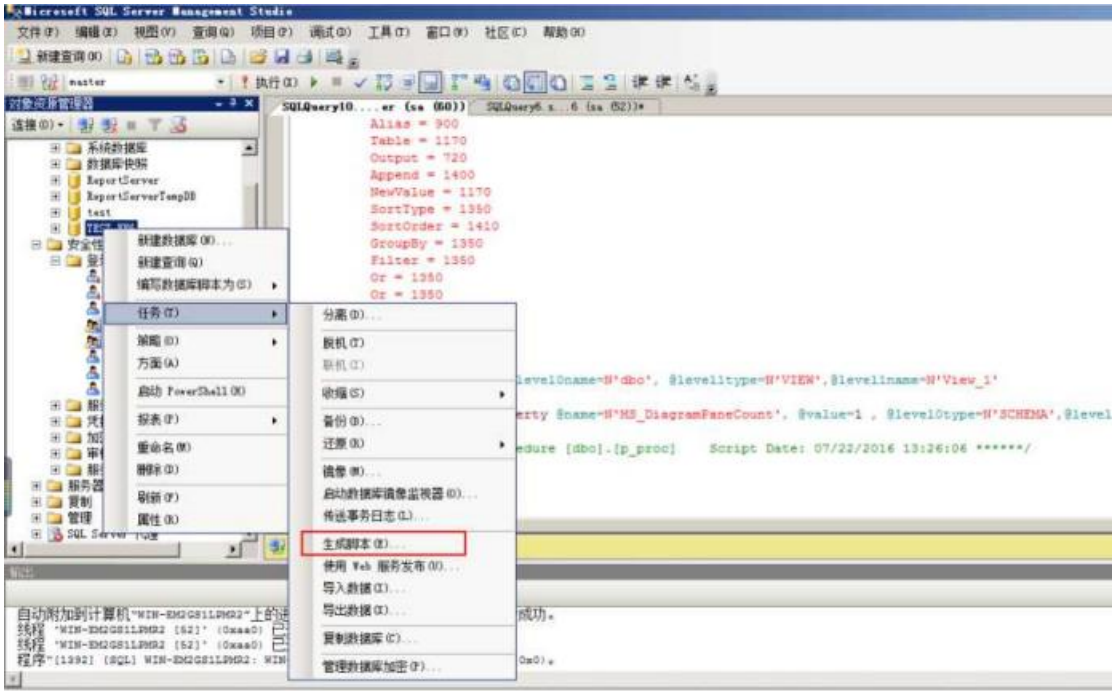
错误码	信息描述
0	字符串截取
-2665	记录超长
-6602	违反唯一性约束
-6607	违反引用约束
-2107	无效的对象

2.4 TSQL 移植

接下来对自定义类型、存储过程、函数、触发器进行移植，可以使用达梦 DTS 工具进行迁移，由于 SQLServe 与 DM 的语法差异较大，都需要进行手工移植。

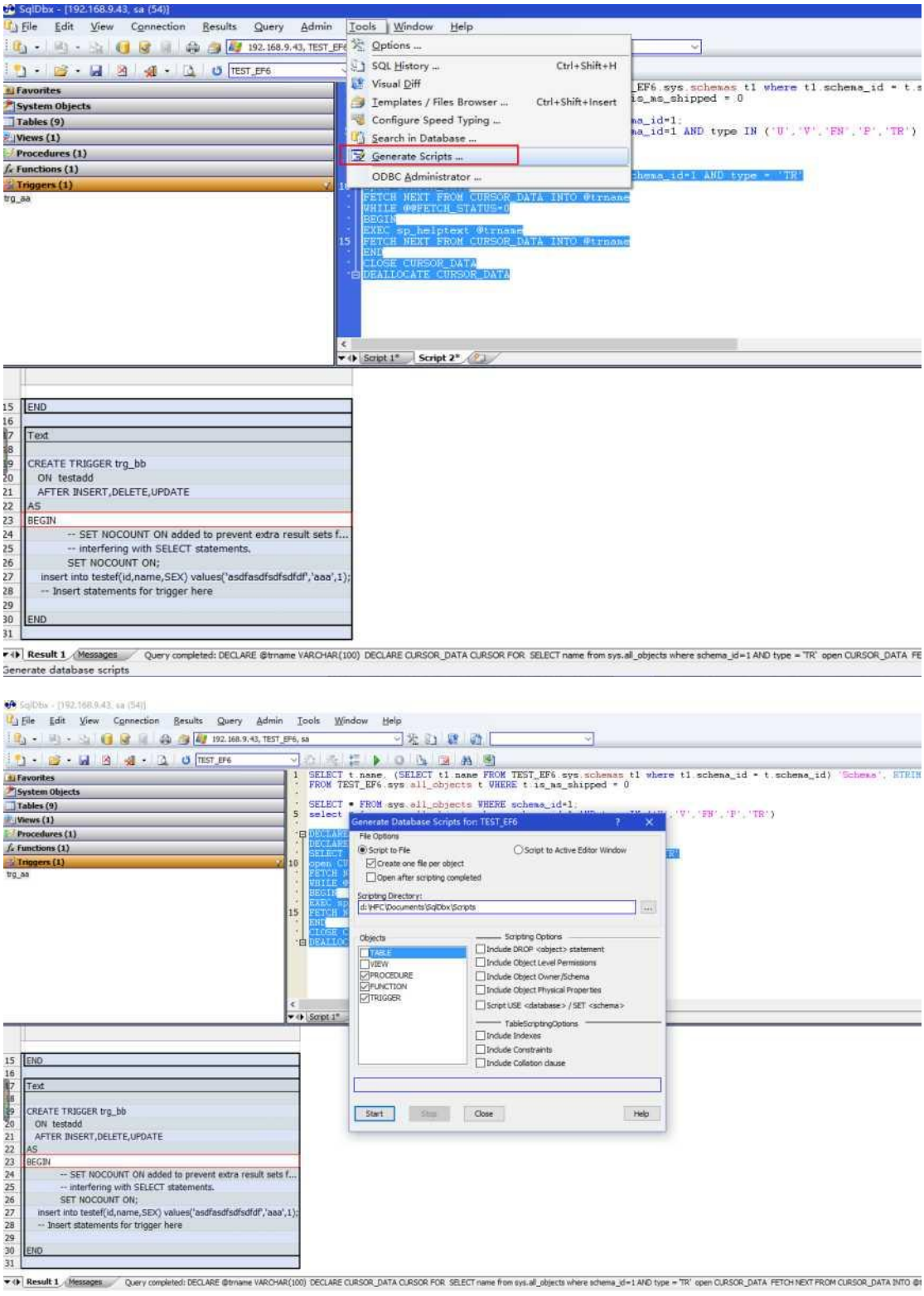
2.4.1 导出待移植对象的脚本

- **方法一：使用 SQL Server Management Studio、SqlDbx 等工具导出自定义类型、存储过程、函数、触发器等 PL/SQL 脚本。**



在 SQL Server Management Studio 工具中选中需要导出的库，右键-任务-生成脚本。

在 SqlDbx 工具中选中需要导出的库，选择菜单工具-生成脚本。



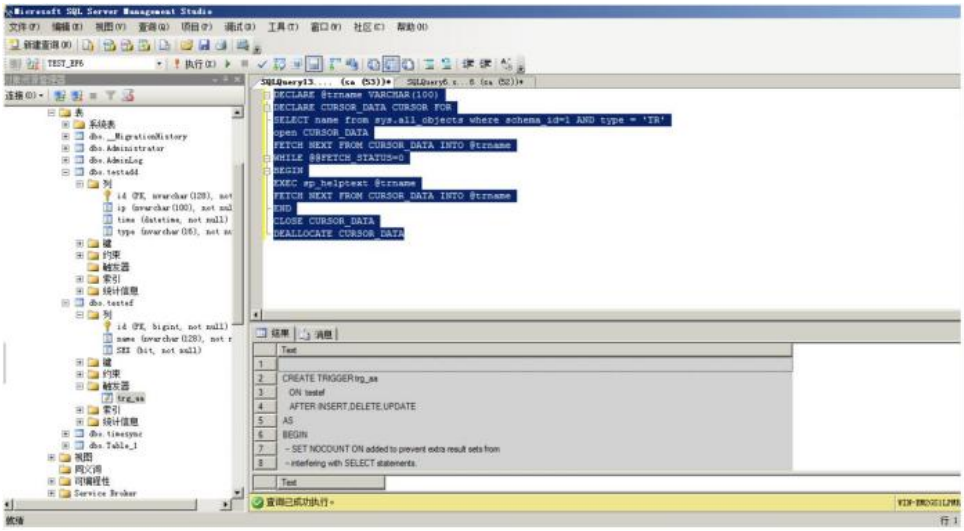
- **方法二：通过 SQL 导出**

  导出某个库中的全部触发器:

```
DECLARE @trname VARCHAR(100)

DECLARE CURSOR_DATA CURSOR FOR
```

```
SELECT name from sys.all_objects where schema_id=1 AND type = 'TR'
open CURSOR_DATA
FETCH NEXT FROM CURSOR_DATA INTO @trname
WHILE @@FETCH_STATUS=0
BEGIN
EXEC sp_helptext @trname
FETCH NEXT FROM CURSOR_DATA INTO @trname
END
CLOSE CURSOR_DATA
DEALLOCATE CURSOR_DATA
```



导出存储过程：

```
DECLARE @trname VARCHAR(100)
DECLARE CURSOR_DATA CURSOR FOR
SELECT name from sys.all_objects where schema_id=1 AND type = 'P'
open CURSOR_DATA
FETCH NEXT FROM CURSOR_DATA INTO @trname
WHILE @@FETCH_STATUS=0
BEGIN
EXEC sp_helptext @trname
FETCH NEXT FROM CURSOR_DATA INTO @trname
END
```

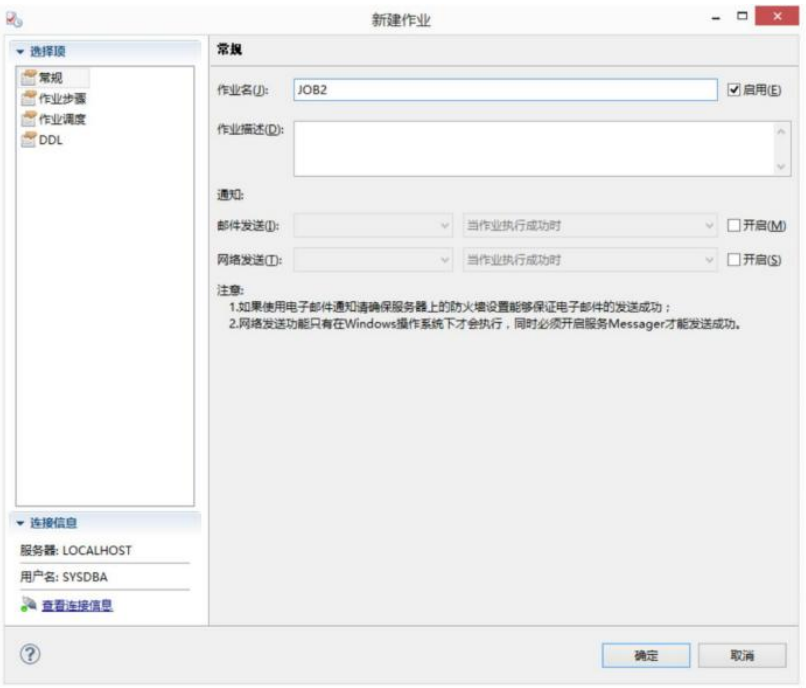
```
CLOSE CURSOR_DATA
DEALLOCATE CURSOR_DATA
```

导出函数：

```
DECLARE @trname VARCHAR(100)
DECLARE CURSOR_DATA CURSOR FOR
SELECT name from sys.all_objects where schema_id=1 AND type = 'FN'
open CURSOR_DATA
FETCH NEXT FROM CURSOR_DATA INTO @trname
WHILE @@FETCH_STATUS=0
BEGIN
EXEC sp_helptext @trname
FETCH NEXT FROM CURSOR_DATA INTO @trname
END
CLOSE CURSOR_DATA
DEALLOCATE CURSOR_DATA
```

SQLServer 的作业：

由于和达梦机制不同，需要了解具体意义后在达梦的代理中重新配置，可以在最后进行该部分的移植：使用管理工具登录，代理->创建代理环境->作业->新建作业->根据需求配置作业步骤和作业调度即可。



2.4.2 运行脚本并处理错误

- **语法分析错误：**

```
[执行语句1]:
create or replace PROCEDURE p1()
as
begin
select sysdate link from dual;
end;
执行失败(语句1)
第 4 行，第 16 列[link]附近出现错误:
语法分析出错
```

（1）由于使用达梦的保留字冲突导致，建议如果可以更换尽量更换，如果不行可以采用屏蔽关键字的方法进行屏蔽。

（2）字符兼容问题，是否使用了中文的标点符号

（3）SQLServer 和 DM 有语法不一致的地方，需要根据具体问题具体分析。

2.5 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;
```

```
--通过编码格式
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.统计指定用户下的所有对象，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.统计各个表的数据量到表数据记录表

```
1. create table dm_tables(tab_owner varchar(100),tab_name
varchar(100),tab_count int);
2. begin
for rec in (select owner,object_name from all_objects where
owner='OA8000_DM2015' and object_type='TABLE') loop
Begin
execute immediate 'insert into dm_tables select '''|| rec.owner
||''','''|| rec.object_name ||''',count(*) from '|| rec.owner || '.'
|| rec.object_name;
exception when others then
print rec.owner || '.' || rec.object_name || 'get count error';
end;
end loop;
end;
3. select * from dm_tables;
```

- 对比达梦数据库中对象和 SqlServer 库中对象以及数据差异

A.比对对象，找出没有迁移的对象

```
select * from sql_objects where (obj_owner,obj_name) not in (
select obj_owner,obj_name from dm_objects
) --and obj_type='TABLE'
```

B.对比表数据量，找出数据量不相等的表。

```
select a.tab_owner,a.tab_name,a.tab_count-b.tab_count from sql_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
```

2.6 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

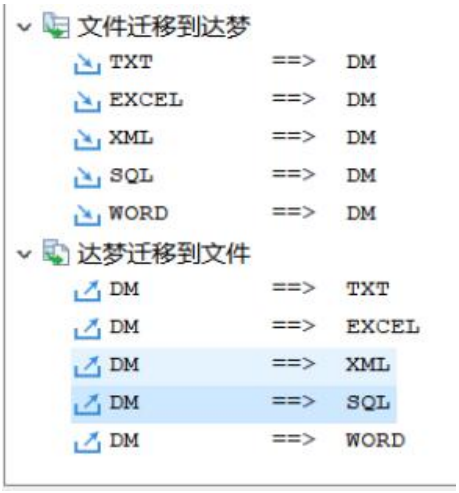
再对数据更新完统计信息后，在数据量不大，磁盘空间足够的情况下应进行一次数据备份工作。数据备份有两种方式：正常停止数据库后，拷贝备份 data 文件夹；或者开启归档日志后，进行物理备份。

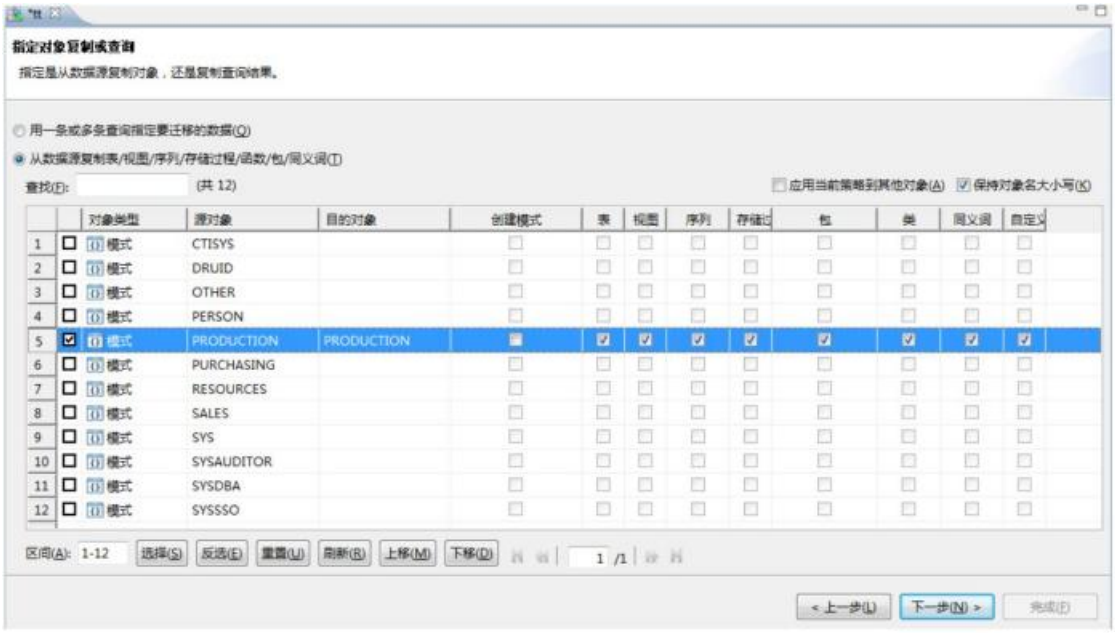
- 整理对象脚本

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：





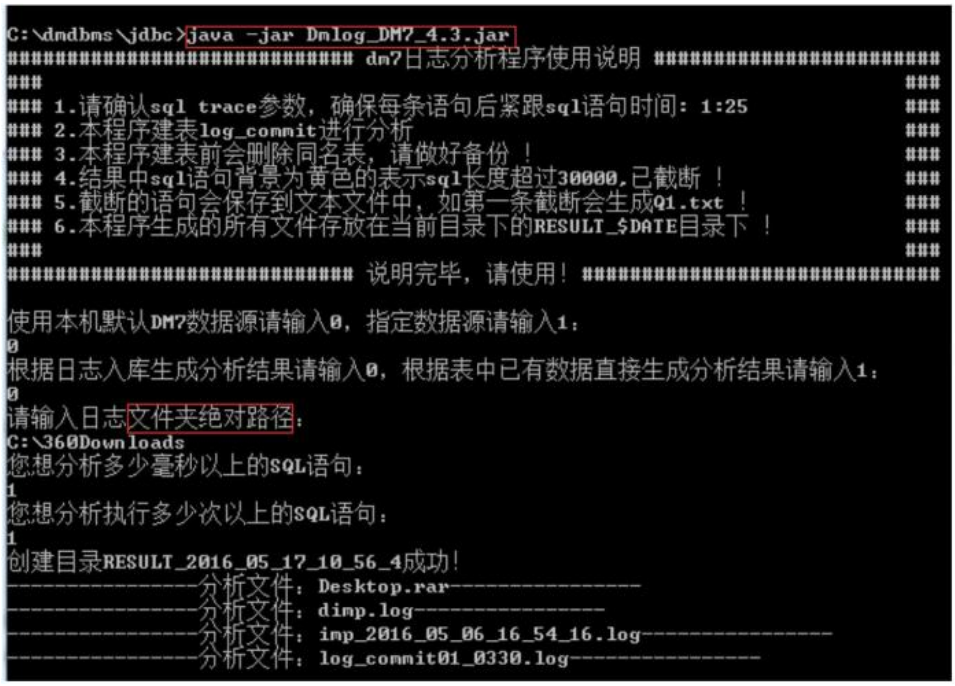
后面直接点击“下一步“即可。

2.7 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。



附录、DM 与 SQLServer2008 对比

1、数据类型

红色标识为有区别的数据类型

类别	SQLSERVER	DM	描述
字符类型	Char(n)	Char(n)	N 为 1~8000 字符之间
	Nchar(n)	Nchar(n)	N 为 1~4000Unicode 字符之间
	Ntext	NCLOB	最多为 2^{30}-1Unicode 字符
	Text	Text	最多为 2^{31}-1 字符
	Varchar(n)	Varchar(n)	N 为 1~8000 字符之间
	Varchar(max)	Text	最多为 2^{31}-1 字符
精确数值数据类型	bit	bit	0、1 或 Null
	tinyint	tinyint	0～255 之间的整数
	smallint	smallint	-32768～32767 之间的整数
	int	int	-2147483648～2147483647
	bigint	bigint	-9223372036854775808～9223372036854775807 之间的整数
	numeric(p,s)或 decimal(p,s)	numeric(p,s) 或 decimal(p,s)	精度范围是 1=<P<=38 标度范围 0<S<38
	money	money	固定精度：19,标度：4
	smallmoney	money	SQLSERVER 精度：-214748.3648～214748.3647 DM 固定精度：19，标度：4
近似数值数据类型	float[(n)]	float[(n)]	-1.7E+308～1.7E+308
	real()	real()	-1.7E+308～1.7E+308
二进制数据类型	Binary(n)	Binary(n)	N 为 1~8000 十六进制数字之间
	Image	Image	最多为 2^{31}-1

			十六进制数位
	Varbinary(n)	Varbinary(n)	N 为 1~8000 十六进制数字之间
	Varbinary(max)	BLOB	最多为 2^{31}-1 十六进制数字
	Timestamp	Binary(8)	时间戳类型，返回一个二进制类型，非时间类型
日期和时间数据类型	Date	Date	日期类型
	Datetime	Datetime	日期时间类型
特殊类型	Uniqueidentifier	Varchar(36)	全局唯一标识符(Globally Unique Identi^er, GUID)。

2、常用函数

2.1 有区别的函数

序号	SQLSERVER	DM	功能简要说明
日期时间函数			
1	DATENAME	DAYNAME MONTHNAME	返回日期中的星期或月份名称
字符串函数			
1	STR	TO_CHAR	将数值类型转换成字符串类型
2	CHARINDEX	LOCATE	在字符串中从特定位置开始查找目 标串的位置
数值函数			
1	/	GREATEST(n1,n2,n3)	求 n1、n2 和 n3 中的最大浮点数
2	/	LEAST(n1,n2,n3)	求 n1、n2 和 n3 中的最小浮点数
3	LOG 10(n)	LN(n)	求数值 n 的自然对数
4	/	MOD(m,n)	求数值 m 被数值 n 除的余数
5	/	SINH(n)	求数值 n 的双曲正弦值

6	/	TANH(n)	求数值 n 的双曲正切值
7	/	TO_NUMBER(char [,fmt])	将 CHAR、VARCHAR、VARCHAR2 等 类型的字符串转换为 DECIMAL 类型的数值
8	/	TRUNC(n[,m])	截取数值函数
9	/	TRUNCATE(n[,m])	截取数值函数，等价于 TRUNC
10	/	TO_CHAR(n [, fmt [,'nlsparam']])	将数字类型的数据转换为 VARCHAR 类型输出
数值函数			

1	NEWID	GUID	SERVER 返回带-格式的 36 位字符串
			DM 返回不带-的格式的 32 位字符 串
			--DM 中可以自定义一个 newid 函数 保持兼容：
			create or replace function newid() return varchar(36)
			as
			tmp varchar(36);
			begin
			return tmp=substr(guid,1,8) 11 '-'||
			substr(guid,1,8) || '-' || substr(guid,9,4) || '-' || substr(guid,13,4) || '-' || substr(guid,16,4) || '-' || substr(guid,20,12) ; end;

2	CAST	CAST DATEDIFF	不同数据类型直接的类型转换，SQLSERVER 中将日期类型转换为 FLOAT 类型时，DM 中对应用 DATEDIFF 函数实现
3	CONVERT	CONVERT TO_CHAR	类型转换，SQLSERVER 中将日期类型转换为字符类型时，DM 中对应用 TO_CHAR 函数实现

2.2 无区别的函数

序号	SQLSERVER	DM	功能简要说明
集函数			
1	AVG	AVG	返回平均值
2	MAX	MAX	返回最大值
3	MIN	MIN	返回最小值
4	COUNT	COUNT	返回记录总数
5	SUM	SUM	返回累加求和值
日期时间函数			
1	DATEADD	DATEADD	日期加法函数
2	DATEDIFF	DATEDIFF	日期减法函数
3	DATEPART	DATEPART	返回日期中部分值
4	GETDATE	GETDATE	返回系统当前时间
字符串函数			
1	LEN	LEN	返回字符串的长度
2	SUBSTRING	SUBSTR\SUBSTRING	返回字符串中的子串
3	RTRIM	RTRIM	从字符串的右边开始去掉空格
4	LTRIM	LTRIM	从字符串的左边开始去掉空格
5	LOWER	LOWER	将字符串变成小写
6	UPPER	UPPER	将字符串变成大写
7	REPLACE	REPLACE	在字符串中进行查找替换
数值函数			
1	ABS(n)	ABS(n)	求数值 n 的绝对值
2	ACOS(n)	ACOS(n)	求数值 n 的反余弦值
3	ASIN(n)	ASIN(n)	求数值 n 的反正弦值
4	ATAN(n)	ATAN(n)	求数值 n 的反正切值
5	ATN2(n1,n2)	ATAN2(n1,n2)	求数值 n1/n2 的反正切值
6	CEILING(n)	CEIL(n)或 CEILING(n)	求大于或等于数值 n 的最小整数

7	COS(n)	COS(n)	求数值 n 的余弦值
8	COSH(n)	COSH(n)	求数值 n 的双曲余弦值
9	COT(n)	COT(n)	求数值 n 的余切值
10	DEGREES(n)	DEGREES(n)	求弧度 n 对应的角度值
11	EXP(n)	EXP(n)	求 e 的 n 次幂值，e=2.71828183...
12	FLOOR(n)	FLOOR(n)	求小于或等于数值 n 的最大整数
13	LOG(n1,n2)	LOG(n1,n2)	求数值 n2 以 n1 为底数的对数
14	PI()	PI()	得到常数 n
15	POWER(n1,n2)	POWER(n1,n2)	求数值 n2 以 n1 为基数的指数
16	RADIANS(n)	RADIANS(n)	求角度 n 对应的弧度值
17	RAND([n])	RAND([n])	求一个 0 到 1 之间的随机浮点数
18	ROUND(n[,m])	ROUND(n[,m])	求四舍五入值函数
19	SIGN(n)	SIGN(n)	判断数值的数学符号
20	SIN(n)	SIN(n)	求数值 n 的正弦值
21	SQRT(n)	SQRT(n)	求数值 n 的平方根
22	TAN(n)	TAN(n)	求数值 n 的正切值
数值函数			
1	CAST	CAST DATEDIFF	不同数据类型直接的类型转换，SQLSERVER 中将日期类型转换为 FLOAT 类型时，DM 中对应用 DATEDIFF 函数实现
2	CONVERT	CONVERT TO_CHAR	类型转换，SQLSERVER 中将日期类型转换为字符类型时，DM 中对应用 TO_CHAR 函数实现
3	@@IDENTITY	@@IDENTITY	返回当前会话中自增列的当前值

3、TSQL 与 DM-SQL

3.1 有区别的语法

变量的定义命名规范

```
--SQL SERVER 中存储过程中定义变量采用@开通，
--在 DM 中@开头的字符不是合法的标识符，所有将所有变量定义中的@替换成下划线
```

变量定义方式

```
--SQL SERVER 中存储过程中定义变量
@StartDate as datetime
在 DM 中要将 as 去掉，改为_Start Date  datetime
```

为每句加上分号

```
--SQL SERVER 存储过程中每条 SQL 语句结束不用加分号
--DM 语法更为严谨，每句结束需要加上分号
```

XML 用法改造

```
--SQL SERVER 中使用 OPENXML 方法来解析 XML
--DM 中使用标准的 XQUERY 语言来解析 XML
--使用 SF_XMLQUERY 函数实现 SQL SERVER 中 OPENXML 的功能
```

临时表用法改造

```
--SQL SERVER 中使用#开头定义临时表
--DM 定义全局的临时表
--创建一个全局临时表
CREATE GLOBAL TEMPORARY TABLE T_USERLIST(USERNAMEVARCHAR(500));
```

强制类型转换

```
--SQL SERVER 中使用 CAST 函数可以将 DATE 类型转换为 FLOAT
--DM 中不能将 DATE 类型强制转换为 FLOAT
--可以使用 DATEDIFF 函数来实现同样的功能
```

循环语句

```
-- SQLServer
```

WHILE

设置重复执行SQL语句或语句块的条件。只要指定的条件为真，就重复执行语句。可以使用BREAK和CONTINUE关键字在循环内部控制WHILE循环中语句的执行。

语法

```
WHILE Boolean_expression
   { sql_statement | statement_block }
   [ BREAK ]
   { sql_statement | statement_block }
   [ CONTINUE ]
```

示例

A.在嵌套的IF...ELSE和WHILE中使用BREAK和CONTINUE

在下例中，如果平均价格少于$30，WHILE循环就将价格加倍，然后选择最高价。如果最高价少于或等于$50，WHILE循环重新启动并再次将价格加倍。该循环不断地将价格加倍直到最高价格超过$50，然后退出WHILE循环并打印一条消息。

```
USE pubs
GO
WHILE (SELECT AVG(price) FROM titles) < $30
BEGIN
      UPDATE titles
            SET price = price * 2
      SELECT MAX(price) FROM titles
      IF (SELECT MAX(price) FROM titles) > $50
            BREAK
      ELSE
            CONTINUE
END
PRINT 'Too much for the market to bear'
```

B.在带有游标的过程中使用 WHILE

以下的 WHILE 结构是名为 count_all_rows 过程中的一部分。下例中，该 WHILE 结构测试

用于游标的函数@@FETCH_STATUS 的返回值。因为@@FETCH_STATUS 可能返回－2、-1 或 0，所以，所有的情况都应进行测试。如果某一行在开始执行此存储过程以后从游标结果中删除，将跳过该行。成功提取(0)后将执行 BEGIN...END 循环内部的 SELECT 语句。
--DM 循环语句 存储模块支持三种基本类型的循环语句，即 LOOP 语句、WHILE 语句和 FOR 语句。LOOP 语句循环重复执行一系列语句，直到 EXIT 语句终止循环为止；WHILE 语句循环检测一个条件表达式，当表达式的值为 TRUE 时就执行循环体的语句；FOR 语句对一系列的语句重复执行指定次数的循环。以下对这三种语句分别进行介绍。 LOOP 语句 LOOP 语句的语法如下： LOOP 语句序列 END LOOP LOOP 语句实现对一语句系列的重复执行，是循环语句的最简单形式。它没有明显的终点，必须借助 EXIT 语句来跳出循环。以下是 LOOP 语句的 WHILE 语句 WHILE 语句的语法如下： WHILE<条件表达式>LOOP 语句序列 END LOOP WHILE 循环语句在每次循环开始以前，先计算搜索条件，若该条件为 TRUE，语句序列被执行一次，然后控制重新回到循环顶部。若搜索条件计算为 FALSE，则结束循环。当然，也可以通过 EXIT 语句来终止循环。以下是 WHILE 语句的用法举例： CREATE OR REPLACE PROCEDURE P_ WHILE(A IN OUT INT) AS BEGIN WHILE A>0 LOOP PRINT A; A:=A-1; END LOOP;

END;

FOR 语句

FOR 语句的语法如下：

FOR <循环计数器> IN [REVERSE] <下限表达式 > .. <上限表达式> LOOP

语句序列

END LOOP

触发器类型

--SQLServer

都有 INSERT、DELETE 和 UPDATE 触发器；SQLServer2000 只有语句级触发器

FOR EACH STATEMENT

--DM

都有 INSERT、DELETE 和 UPDATE 触发器；DM 有元组级触发器 FOR EACH ROW 和语句级触发器 FOR EACH STATEMENT

触发器激发顺序

--SQLServer

AFTER 触发器只有在触发 SQL 语句中指定的所有操作都已成功执行后才激发。所有的引用级联操作和约束检查也必须成功完成后，才能执行此触发器。如果仅指定 FOR 关键字，则 AFTER 是默认设置。

INSTEADOF 执行触发器而不是执行触发 SQL 语句，从而替代触发语句的操作。在表或视图上，每个 INSERT、UPDATE 或 DELETE 语句最多可以定义一个 INSTEADOF 触发器。然而，可以在每个具有 INSTEADOF 触发器的视图上定义视图。

--DM

如果有语句级前触发器的话，先运行该触发器；

对于受语句影响每一行：如果有行级前触发器的话，运行该触发器；

执行该语句本身；

如果有行级后触发器的话，运行该触发器；

如果有语句级后触发器的话，运行该触发器。

变异表

--SQLServer

SQL Server 允许触发器的递归调用，并且不对变异表做检查。

递归触发器允许发生两种类型的递归：

使用间接递归时，应用程序更新表 T1，从而激发触发器 TR1，该触发器更新表 T2。在这种情况下，触发器 T2 将激发并更新 T1。

使用直接递归时，应用程序更新表 T1，从而激发触发器 TR1，该触发器更新表 T1。由于表 T1 被更新，触发器 TR1 再次激发，依此类推。

--DM

变异表就是当前被 DML 语句修改的表。对 DM 触发器来说，变异表就是触发器在其上进行定义的表。在默认情况下元组级触发器体中的 SQL 语句不能读或修改触发语句的变异表。在某些特殊的环境下需要触发器的递归调用，可以忽略变异表的检查，方法是修改 DM.ini 参数 IGNORE_MUTATING_CHECK = 0 为 1。

临时表

```
-- SQLServer
--SQL SERVER 中使用#开头定义临时表
--可以主动创建
CREATE TABLE #T_USERLIST(USERNAME VARCHAR(500));
--也可以由系统自动创建
Select * into #test from table1;
--SQLSERVER 中的临时表只对当前会话可见，对其他会话不可见
--SQLSERVER 中临时表用完后会随着会话的断开而自动删除
```

```
-- DM
CREATE GLOBAL TEMPORARY TABLE T_USERLIST(USERNAME VARCHAR(500));
--达梦中临时表必须由用户主动创建
--DM 中可以定义全局的临时表，不同的会话都可以使用，每个会话只能看到自己的数据，看不到其他会话的数据
--DM 中临时表中数据会随着会话的断开而自动删除，但表不会自动删除，需要用户手动删除
```

存储过程中变量定义差异

```
-- SQLServer
```

@StartDate as datetime, --SQLSERVER 中可以在 as 和 begin 之间定义， 也可以在存储过程体的任何位置定义变量；
-- DM --DM 中需要将 as 去掉,如下： _ StartDate datetime; --DM 中变量定义只能在 as 和 begin 之间

XML 用法

-- SQLServer SQL SERVER 中使用 OPENXML 方法来解析 XML
-- DM --DM 中使用标准的 XQUERY 语言来解析 XML,使用 SF_XMLQUERY 函数实现 SQL SERVER 中 OPENXML 的功能

类型转换

SQL SERVER 中使用 CAST 函数可以将 DATE 类型转换为 FLOAT，DM 中不能将 DATE 类型强制转换为 FLOAT，可以使用 DATEDIFF 函数来实现同样的功能 --SQLSERVER 中 SQL 语句： (cast(workendtime as float)- cast(workstarttime as float))*24 --DM 中 SQL 语句，取两个日期之间的小时数 datediff(HH,workstarttime,workendtime) SQL SERVER 中使用 Convert 函数将日期类型转换成字符串类型，如： Convert(varchar(7),’2012-12-12’,126)； DM 中使用 to_char 函数将日期类型转换成字符串类型，如： to_char(’2012-12-12’,’yyyy-mm’) --SQLSERVER 方式 declare @duration varchar(100) declare @rtn varchar(100)

```
declare @h int,@m int,@s int
set @duration='1923234'
set @duration=cast(@duration as bigint)
set @h=@duration/1000/60/60
set @m=(@duration-@h*1000*60*60)/1000/60
set @s=(@duration-@h*1000*60*60-@m*1000*60)/1000
set @rtn=cast(@h as varchar(10))+'小时'+cast(@m as varchar(10))+'分钟'+cast(@s
as varchar(10))+'秒'
print @rtn
--DM 方式
declare
        v_duration varchar(100):='1923234';
        v_rtn varchar(100);
        v_times int;
        v_h int;
        v_m int;
        v_s int;
BEGIN
          set v_times = cast(v_duration as bigint);
        set v_h = v_times/1000/60/60;
        set v_m = (v_times-v_h*1000*60*60)/1000/60;
          set v_s = (v_times-v_h*1000*60*60-v_m*1000*60)/1000;
          set v_rtn = v_h || '小时' || v_m||'分钟'||v_s||'秒';
          print v_rtn;
END
```

语句块用法

```
--SQL SERVER 格式
```

```
update test set c=1
if @@rowcount>0
insert into test1 values(‘test’)
--DM 格式
Begin--语句块要用 begin end 包起来
update test set c=1; --加上分号
if sql%rowcount>0 then
--@@rowcount 改为 sql%rowcount
insert into test1 values(‘test’); --加上分号
end if; --加上 end if
end
```

获取自增列

```
--SQL SERVER 格式
create table test(
c1 int identity(1,1),
c2 varchar(10)
);
create table test1(
c1 int identity(1,1),
c2 varchar(10)
);
truncate table test;
truncate table test1;
declare @a int,@b int
insert into test(c2) values('aa')
set @a = @@identity; --@a 为 1，为 test 表 c1 列当前最大值
insert into test1(c2) values('aa')
insert into test1(c2) values('aa')
set @b = @@identity; --@b  为 2，为 test1 表 c1 列当前最大值
```

```
print@a
print@b


--DM 格式
declare
a int;b int;
begin
insert into test(c2) values('aa');
a := ident_current('test');
insert into test1(c2) values('aa');
insert into test1(c2) values('aa');
b := ident_current('test1');
print a;
print b;
end;
--DM 与 sqlserver 兼容模式：
declare
a int;b int;
begin
insert into test(c2) values('aa');
set a = @@identity;
insert into test1(c2) values('aa');
insert into test1(c2) values('aa');
set b= @@identity;
print a;
print b;
end;
```

行合并

```
--SQLSERVER 使用方式
```

```
DECLARE @sql NVARCHAR(max)
select @sql=isnull(@sql + ',' , '') + priv_name from DMETL_PRIVILEGE
PRINT @sql
--DM 使用方式
DECLARE   v_sql   NVARCHAR(max)
begin
select wm_concat(priv_name) into v_sql from DMETL_PRIVILEGE;
PRINT v_sql;
end;
```

保留字

SQLSERVER 中 rowid、Percent,class，Create 不是保留字

DM 中 rowid、Percent、class，Create 是保留字，所以不能用作对象名和参数名，需要改为 rowid1、Percent1、class1、Create1

定界标识符

SQLSERVER 中采用中括号[]作为定界标识符，一般需要用保留字做对象名时采用定界标识符括起来以避免语法错误；

DM 中默认采用双引号""作为定界标识符，为兼容 SQLSERVER 的用法，可以在 DM 数据库的配置文件:dm.ini 中修改 MS_PARSE_PERMIT 参数为 1，这样 DM 中就可以使用[]作为定界标识符了。

游标用法

```
--游标取值
--SQL SERVER 中采用以下方式取值：
FETCH NEXT FROM CURSOR_NAME
INTO @ID,@NAME;
--DM 中采用以下方式取值：
FETCH CURSOR_NAME INTO _ID,_NAME;
--游标循环
--SQL SERVER  中采用以下方式循环：
While __fecht_status=0
```

```
Begin
end
--DM 中采用以下方式循环：
Loop
  Exit when CURSOR_NAME%notfound;
End loop;
--关闭游标
--SQL SERVER  中采用以下方式关闭游标：
CLOSE CURSOR_NAME
DEALLOCATE    CURSOR_NAME
--DM  中采用以下方式关闭游标：
CLOSE CURSOR_NAME
```

3.2 无区别的语法

项目	SQLServer	DM
语句块	相同	数据类型需要预先定义
赋值语句	相同	相同
条件语句	相同	相同